mistrovství

Ľuboslav Lacko

Android

KOMPLETNÍ PRŮVODCE VÝVOJÁŘE

- Návrh layoutu i aplikační logiky
- Senzory a jimi poskytovaná data
- Určování polohy a navigace
- Napojení aplikace na sociální sítě a cloudové služby
- Vývoj pro Android Wear a Android TV
- Testování a ladění, umístění aplikace do obchodu

computer press

Mistrovství Android

Vyšlo také v tištěné verzi

Objednat můžete na
www.computerpress.cz
www.albatrosmedia.cz



Ľuboslav Lacko
Mistrovství – Android – e-kniha
Copyright © Albatros Media a. s., 2017

Všechna práva vyhrazena.
Žádná část této publikace nesmí být rozšiřována
bez písemného souhlasu majitelů práv.

ALBATROS MEDIA a.s.

Kniha byla zakoupena na serveru alza.cz.

Kupující: Martin Kadlas

Adresa: Konevova 910, 13000 Praha 3, cz

ID 20110-220827794950106221 12-174734-209

Upozorňujeme, že kniha je určena pouze pro potřeby kupujícího.
Kniha jako celek ani žádná její část nesmí být volně šířena na internetu, ani jinak dále zveřejňována.
V případě dalšího šíření neoprávněně zasáhnete do autorského práva s důsledky dle platného autorského zákona a trestního zákoníku.
Neoprávněným šířením knihy poškodíte rozvoj elektronických knih v České republice.

Tak nám, prosím, pomozte v rozvoji e-knih a chovejte se ke knize, k vydavatelům, k autorům a také k nám fér.

Ľuboslav Lacko

Mistrovství Android

Computer Press
Brno
2017

Obsah

Úvod 15

Kapitola 1
Nástroje pro vývoj 17

- **Instalace Javy** 17
- **Instalace vývojového prostředí Android Studio** 20
- **První spuštění a konfigurace Android Studia** 22
- **Průběžná aktualizace** 26
- **Vytvoření projektu aplikace** 27
- **Odemknutí zařízení pro vývoj** 35
  - Spuštění aplikace na reálném zařízení 36
  - Monitorování spuštěné aplikace 38
- **Seznámení se s vývojovým prostředím** 39
  - Editor kódu v jazyce Java 46
  - Java pro migranty 48
  - Gradle 49
- **Možnosti současného hardwaru** 52
- **Referenční zařízení** 52
- **Vytvoření emulátoru** 54
- **GitHub** 57
  - Vytvoření účtu 58
  - Vytvoření repozitáře 60
  - Vytvoření větve (branch) 62
  - Úprava souborů 63

Kapitola 2
Anatomie Androidu 65

- **Multiplatformní operační systém** 65
- **Historie verzí** 66
- **Starší verze** 66
- **Android 5.0 Lollipop** 68
- **Android 6.0 Marshmallow** 70
- **Android 7.0 Nougat** 72
- **Stručně o architektuře Androidu** 74
  - Linux Kernel 75
  - Hardware Abstraction Layer (HAL) 75

Nativní knihovny 76
Android Runtime – ART 76
Java API Framework 76
Aplikace 77

Kapitola 3
Anatomie aplikace 79

Dříve než začnete vyvíjet – filozofie aplikace 79
Základní součásti aplikace pro Android 83
Aktivity (Activity) 83
Služby (Services) 84
Broadcast receivers 84
Poskytovatelé obsahu (Content providers) 84
Aktivita a její životní cyklus 84
Životní cyklus aktivity 86
onCreate() 87
onStart() a onResume() 87
onPause() 87
onStop() 88
onDestroy() 88
Příklad – životní cyklus aktivity 88
Příklad – ukládání a obnovení stavu aktivity 91
Intent 92
Odevzdávání údajů a výsledků 93
Intent Filter 94
Příklad – Anatomie projektu 96
Vytvoření projektu 96
Výběr cílové platformy 98
Výběr typu hlavní aktivity 99
Složky a soubory tvořící projekt 100
Jednoduchý vizuální návrh uživatelského rozhraní 103
Morphing – změna typu prvku 106
Definice uživatelského rozhraní 107
Uživatelské rozhraní aplikace na různých zařízeních 109
Aplikační kód 111
Odstranění tlačítka FloatingActionButton z Basic Activity 113
Definice objektů ve zdrojích (resources) 113
Textové řetězce 115
Pole řetězců 115
Množné číslo podstatných jmen 115
Modifikátory 116
Aplikační manifest 117
Identifikační grafika aplikace 119

Kapitola 4
Škálovatelný design 121
Designový styl Material 121
Přizpůsobení rozlišení obrazovky 125
Metrika designu 127
Pravidlo násobků rozměrů 131
Variabilita zařízení a obrazovek 131

Kapitola 5
Uživatelské rozhraní a události 135
Návrh uživatelského rozhraní 136
Kontejnery na rozmístění prvků 137
LinearLayout 137
RelativeLayout 139
FrameLayout 142
TableLayout 143
GridLayout 145
AbsoluteLayout 147
CoordinatorLayout 147
Vnoření kontejnerů 148
ScrollView 150
Příklad – definice rozložení prvků 151
Obsluha události 156
Obsluha událostí ve vizuálním návrhu 159
Příklad – spuštění jiné aktivity 161
Aplikační logika 162
Vytvoření nové aktivity 165
Ladění aplikace 169

Kapitola 6
Fragmenty 173
Fragmenty 174
Životní cyklus fragmentu 175
Statický a dynamický layout 176
Příklad 1 – změna orientace 178
1. Vytvoření layoutu fragmentů 178
2. Hlavní aktivita 179
3. Vytvoření tříd fragmentů 180
4. Úprava třídy hlavní aktivity 181
5. Test aplikace 182
Příklad 2 – hlavní aktivita využívající fragment 182
Příklad 3 – aplikace se dvěma panely 185
1. Vytvoření layoutu fragmentů 185

2. Hlavní aktivita 186
3. Vytvoření tříd fragmentů 187
4. Úprava třídy hlavní aktivity 188
5. Test aplikace 188
Příklad 4 – aplikace master-detail se dvěma panely 189
1. Zobrazované údaje 190
2. Hlavní aktivita 190
3. Kód fragmentu na zobrazení seznamu objektů 191
4. Kód fragmentu na zobrazení seznamu detailů 193
5. Aktivita na zobrazení detailu 194
6. Test aplikace 195

Kapitola 7
Aktivity a intenty 197
Intent 197
Explicitní intenty 197
Příklad – explicitní Intenty 200
Implicitní Intenty 204
Příklad – implicitní Intenty 205

Kapitola 8
Interakce s uživatelem 209
Dotyky a gesta 209
Příklad – zobrazení událostí dotyků 211
Příklad – zobrazení vícenásobných dotyků 214
Detekce standardně používaných gest 218
Příklad – detekce gest 219
Příklad – gesto pinch 221
Příklad – univerzální počítadlo 223
Příklad – využití vlastních gest aplikace 226
Floating Action Button 230
Nabídka funkcionality – menu 232
Možnosti aplikační lišty App Bar 233
Navigace pomocí menu 234
Přizpůsobení a rozdělení lišty Action Bar 239
Navigace na jinou aktivitu (a zpět) 240
Rozdělení obsahu na karty 242
Skrývání aplikační lišty 248
Využití vysouvacího panelu Navigation Drawer 250

Kapitola 9
Seznamy objektů 257
Jak to funguje? 257
Třída ListActivity 259
Příklad – ListActivity s formátovaným zobrazením položek 263
Návrh uživatelského rozhraní 263
Aplikační kód 265
Příklad – ListActivity s kontextovým menu položky 267
Příklad ListView se statickými údaji v poli řetězců 270
Návrh uživatelského rozhraní 270
Aplikační kód 271
Příklad – ListView s údaji v kódu 272
Návrh uživatelského rozhraní 272
Aplikační kód 272
Příklad – výběr více položek ze seznamu 273
Návrh uživatelského rozhraní 274
Aplikační kód 276
Příklad – Zobrazení hierarchické struktury 279
Návrh uživatelského rozhraní 279
Aplikační kód 281
Seznam objektů s obrázky 286
Návrh uživatelského rozhraní 286
Aplikační kód 287
Příklad – Aktivita Master/Detail Flow 291
Vytvoření projektu 291
Struktura projektu 293
Přizpůsobení 294
Přidání objektu do seznamu 298
Seznamy objektů využívající RecyclerView a CardView 301
RecyclerView 301
CardView 302
Příklad RecyclerView a CardView 302

Kapitola 10
Databáze a práce s údaji 309
Ukládání údajů 309
Třída SharedPreferences 309
Příklad – uložení nejvyššího dosaženého skóre hry 310
Příklad – PreferenceFragment 313

Ukládání údajů do souboru 316
Příklad – ukládání do souboru v interní paměti 317
Příklad – ukládání do souboru v externí paměti 320
Ukládání dočasných souborů 322
Databáze SQLite 323
Pokusy se zabudovanou databází SQLite v emulátoru 324
Interakce aplikace s databází 326
Java třídy na přístup k SQLite databázi a jejím údajům 327
Příklad – čtenářský deník 328
Rozbor řešení 328
Datový model 331
Vytvoření databázové tabulky 332
Vkládání záznamů do databáze 333
Aktualizace záznamů 334
Vymazání záznamů 334
Výběr údajů z databázové tabulky 335
Návrh uživatelského rozhraní 338
Hlavní aktivita 338
Aktivita na přidání záznamu 341
Aktivita na editování existujícího záznamu 342
Programování aplikační logiky 344
Hlavní aktivita 345
Aktivita na přidání záznamu 346
Aktivita na editování záznamu 347
Údaje z webu ve formátech JSON a XML 349
Rozbor zadání 349
Zdroj údajů pro aplikaci 349
Návrh uživatelského rozhraní 351
Aplikace na načítání údajů ve formátu JSON 353
Aplikace na načítání údajů ve formátu XML 357
Varianta využívající Document Object Model 357
Varianta využívající XmlPullParser 360
Statická data ve formátu XML 365
Aplikační kód 368

Kapitola 11
Grafika 371

Zobrazení obrázku přes XML návrh 372
Zobrazení obrázku programově 373
Zobrazení obrázku z internetu 375
Rozbor řešení 375
Povolení přístupu k internetu 375
Návrh uživatelského rozhraní 375
Aplikační kód 376

Základní grafické tvary – staticky 377
Základní grafické tvary – dynamicky 379
Vykreslování na Canvas 381
Kreslení dotykem na Canvas 382
Dynamické vykreslování na Canvas s využitím View 384
Dynamické vykreslování na Canvas s využitím SurfaceView 387
Animace 390
Metody klasické animace 391
Příklad animace TransitionDrawable 391
Příklad animace AnimationDrawable 392
Příklad animace Tween 394
Property Animation 396
Příklad na ilustraci principu fungování Property Animation 398
Příklad komplexnějšího využití Property Animation 399
Animace prvků uživatelského rozhraní 401
Příklad – animace přesunu tlačítek 402

Kapitola 12
Multimédia 407

Příklad – přehrávání zvuku 408
Příklad – přehrávání videa 412
Příklad – nahrávání zvuku 414
Příklad – Snímání fotografie I 418
Příklad – Snímání fotografie II 423
Příklad – Snímání fotografie s využitím externí aplikace 426
Příklad – Nahrávání videa 427

Kapitola 13
Senzory a komunikace 431

Integrované senzory smartphonů a tabletů 431
Získávání údajů ze senzorů 432
Identifikace senzoru a jeho možností (capabilities) 433
Příklad – zobrazení údajů z akcelerometru 434
Příklad – zobrazení filtrovaných údajů z akcelerometru 438
Příklad – magnetický kompas 442
Senzor osvětlení 446
Náklonoměr 446
Snímač otisků prstů 447
Komunikace přes Bluetooth 455

Kapitola 14
Mapy a navigace 461
Lokalizační a mapové služby 461
Příklad – určení polohy zařízení 462
Příklad – určení polohy zařízení 468
Zobrazení polohy na mapě 472
Přípravné kroky 473
Instalace Google Play Services 473
Úvodní příklad 474
Vytvoření Google Maps API klíče 474
Příklad zobrazení polohy na mapě 478
Příklad – Zobrazení polohy v jiné aplikaci schopné zobrazit mapy 479
Příklad – zobrazení aktuální polohy na mapě 483

Kapitola 15
Sociální sítě 487
Vytvoření aplikace využívající Facebook 487
Facebook for developers 487
Sdílení obsahu z aplikace na Facebooku 489
Uživatelské rozhraní na sdílení obsahu na Facebook 490
Vytvoření aplikace pro Android přihlašující se k Facebooku 491
Příklad aplikace na posílání statusů a obrázků 497

Kapitola 16
Cloudové služby 509
Google Cloud Messaging 509
Firebase Cloud Messaging 510
FCM zprávy 512
Příklad na posílání zpráv 514
Konfigurace FCM 518
Přijímání notifikací na popředí 520
Posílání zpráv přes Azure Notification Hub 523
Vytvoření projektu v Android Studiu 523
Vytvoření FCM 523
Vytvoření Azure Notification Hub 525
Přidání služby Google Play do projektu aplikace pro Android 527
Aplikační logika v kódu v Javě 529
Připojení aplikace pro Android k mobilní službě 535
Microsoft Azure Mobile Apps 536
Vytvoření aplikace využívající službu Azure Mobile Apps 537
Konfigurace služby 538

Kapitola 17
Služby a broadcasty 543
Služba a její životní cyklus 544
Příklad – přehrávání hudby na pozadí 546

Kapitola 18
Poskytování obsahu 551
Třída ContentProvider 551
Ukládání kontaktů v zařízení s Androidem 553
Příklad – přístup ke kontaktům v zařízení 553
Příklad – přístup ke kontaktům, načítání údajů na pozadí 558
Příklad – aktualizace údajů 559
Příklad – vytvoření aplikace, která bude poskytovat údaje 562
Příklad – vytvoření aplikace, která bude poskytovat údaje z databáze 565

Kapitola 19
Vývoj pro nositelná zařízení 573
Zaměření na účel 573
Nabídka: Kontextový stream 575
Dotaz: Cue cards 575
Vytvoření emulátoru Android Wear 576
Projekt aplikace pro Android Wear 579
Posílání notifikace do hodinek 584
Hospodaření s plochou displeje 587
Aplikace na popředí 589

Kapitola 20
Novinky verze 7.0 Nougat 595
Vylepšené notifikace 595
Podpora více oken (Multi-Window) 601
Jak povolit režim Freeform na zařízení nebo emulátoru s Androidem 7.0 603
Životní cyklus aplikace podporující Multi-Window 604
Příklad aplikace podporující Multi-Window 605
Režim obraz v obraze (Picture-In-Picture) 608
Vývoj aplikace pro Android TV 609
Specifika televizoru z hlediska běhu aplikací 609
Vytvoření projektu aplikace 610
Spuštění aplikace pro Android TV na emulátoru 613

Kapitola 21
Publikování aplikací do aplikačního obchodu Google Play 615
Reklama v mobilních aplikacích 615
Formáty reklamy v mobilních aplikacích 616
Odhad profitu z reklamy v aplikacích 616
Specifika a cílení reklamy 617
Modely platby od inzerentů 619
Kritéria úspěšnosti aplikace 619
Google AdWords 620
Zviditelněte se ve vyhledávání Google Play 621
Možnosti a povinnosti vývojáře vůči aplikačnímu obchodu a publikované aplikaci 622
Registrace vývojářského účtu 622
Vytvoření stránky vývojáře 623
Vytvoření balíčku aplikace na publikování 624
Pravidla publikování aplikace 626
Publikování aplikace 628
Snímky obrazovky 629
Ikona ve vysokém rozlišení 630
Hlavní grafika 630
Propagační grafika 630
Propagační video 631

Rejstřík 633

Úvod

Android je v současnosti nejpopulárnější operační systém pro mobilní zařízení, jeho tržní podíl překračuje 80 %. Proto je tato platforma obrovskou výzvou pro vývojáře aplikací. Více než miliarda majitelů chytrých telefonů a tabletů s tímto operačním systémem má obrovský tržní potenciál nejen pro placené, ale i volně stáhnutelné aplikace, protože i v této kategorii mohou úspěšné aplikace svým tvůrcům vydělávat.

Na rozdíl od iOS, kde potřebujete na vývoj, aspoň v konečné fázi, počítač s operačním systémem OS X, na vývoj aplikace pro Android můžete použít vývojářský počítač jakékoliv platformy, tedy Windows, Linux i OS X. Všechny klíčové nástroje jsou zdarma, takže k vývoji mobilních aplikací pro Android nejsou zapotřebí žádné investice. Aby bylo možné spouštět aplikace na emulátoru, vývojářský počítač s operačním systémem Windows musí mít minimálně 4 GB RAM.

Problém nepředstavuje ani etapa testování, jelikož není důležité otestovat aplikace na „vlajkové lodi", tam určitě nebude mít žádné problémy s výkonem, ale spíše na levných, méně výkonných telefonech, které jsou samozřejmě mezi lidmi mnohem více rozšířeny. Díky stále rychlejším emulátorům dokážete provést celý vývoj a teoreticky i testování většiny aplikací, které nevyužívají speciální hardwarovou výbavu, aniž byste měli hardwarové zařízení k dispozici fyzicky, avšak v závěrečné fázi před publikováním do aplikačního obchodu doporučujeme důkladně otestovat aplikaci i na „železe", nejlépe na telefonu i tabletu.

Publikace předpokládá určité předběžné znalosti:

- Znalost programovacího jazyka Java; postačí i znalost C#, jelikož tento jazyk je odvozený od Javy a je mu velmi blízký
- Znalost základních principů objektově orientovaného programování
- Zkušenosti s používáním moderních integrovaných vývojových prostředí
- Znalost SQL (nejlépe SQLite, ale postačuje základní všeobecný přehled)
- Znalost základů XML, jelikož v tomto formátu se realizuje návrh uživatelského rozhraní

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu připravilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

Computer Press
Albatros Media a.s., pobočka Brno
IBC
Příkop 4
602 00 Brno

nebo

sefredaktor.pc@albatrosmedia.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih nějakou najdete, ať už v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2301* po klepnutí na odkaz Soubory ke stažení. (Nejsou-li žádná errata zatím k dispozici, není odkaz Soubory ke stažení dostupný.)

KAPITOLA

1

Nástroje pro vývoj

Vstupním bodem k získání vývojářských nástrojů, návodů a příkladů je stránka *http://developer.android.com*. K vývoji aplikací budete potřebovat dva balíky softwaru:

- Java Development Kit (JDK)
- Android Studio

Ještě před rokem Google doporučoval používat vývojové prostředí Eclipse doplněné o Android Development Tools (ADT) a Android Software Development Kit, jelikož Android Studio bylo dlouho k dispozici pouze ve verzi Early Access Preview a instalace vývojářských nástrojů byla mnohem složitější.

Změna oficiálně doporučeného vývojového prostředí představuje v tomto případě jednoznačně krok vpřed. Změnily se některé technologie, například překladač využívající automatizační nástroj Gradle či IntelliJ IDEA Java IDE. Základní principy a filozofie zůstaly zachovány, změnilo se jen uživatelské rozhraní. I nadále budete využívat programovací jazyk Java a návrh uživatelského rozhraní se stále ukládá, případně i vytváří, v dokumentu XML. Díky tomu je migrace z Eclipse na Android Studio rychlá a bezproblémová. A ještě jedna důležitá informace: Projekty vytvořené v Eclipse můžete velmi snadno importovat do Android Studia bez nutnosti předchozího exportu.

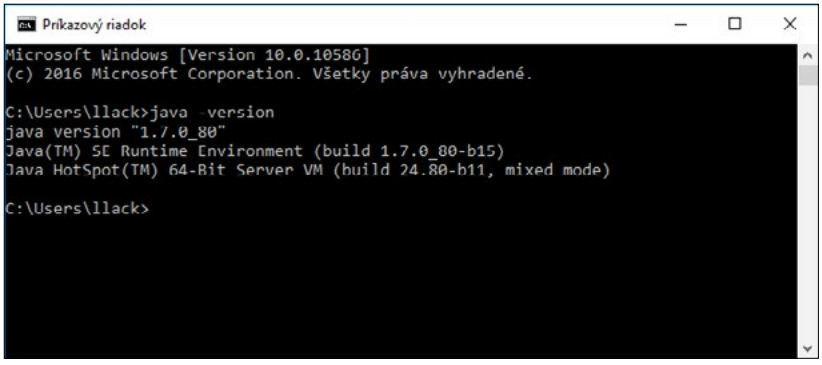
Instalace Javy

Java je objektově orientovaný programovací jazyk, jehož syntaxe je podobná C nebo C++. Java je interpretovaný jazyk, což znamená, že program se nepřekládá přímo do strojového kódu, ale do takzvaného bajtkódu Java. Ten je následně interpretován virtuálním strojem. Java je tak jazyk nezávislý na platformě.

O tom, zda máte, nebo nemáte nainstalovaný Java SE Development Kit a v jaké verzi, se přesvědčíte snadno. Stačí do konzolové aplikace zadat příkaz `java -version`.

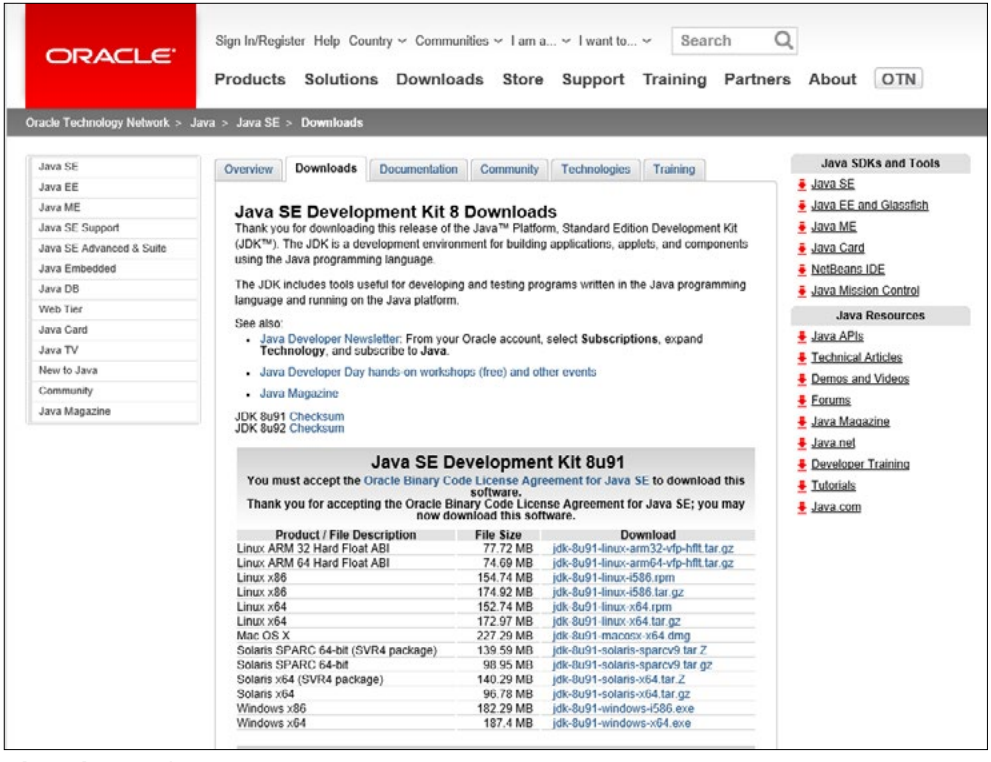
Témata kapitoly:

- **Instalace Javy**
- **Instalace vývojového prostředí Android Studio**
- **První spuštění a konfigurace Android Studia**
- **Průběžná aktualizace**
- **Vytvoření projektu aplikace**
- **Odemknutí zařízení pro vývoj**
- **Seznámení se s vývojovým prostředím**
- **Možnosti současného hardwaru**
- **Referenční zařízení**
- **Vytvoření emulátoru**
- **GitHub**

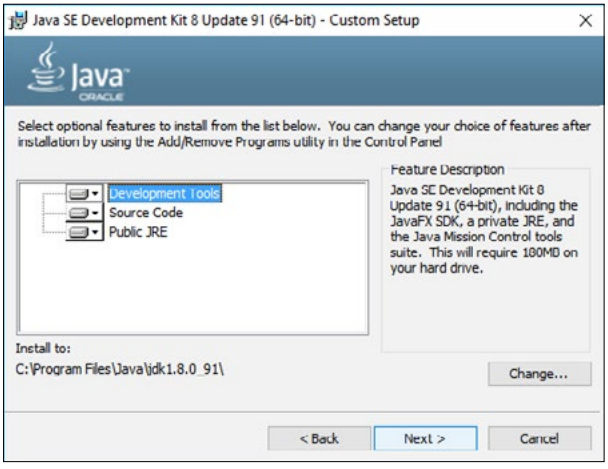


Obrázek 1.1: Identifikace nainstalované verze JDK

Platforma Java byla produktem společnosti Sun. Tuto společnost i se všemi produkty a řešeními získala společnost Oracle, která Javu momentálně spravuje. Z jejích stránek je možné Java SDK stáhnout a nainstalovat. Javu SE Development Kit ve verzi 8, případně novější, stáhnete pro platformy Windows a Linux, pro platformu macOS stáhnete Javu ze stránek společnosti Apple. V závislosti na verzi operačního systému stáhněte 32- nebo 64bitovou variantu JDK.

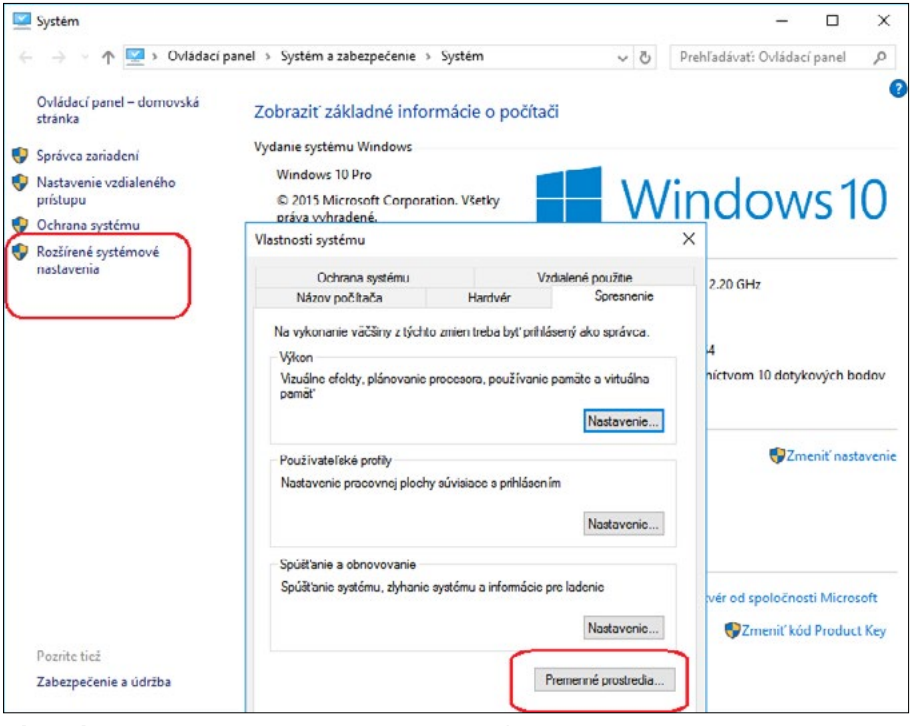


Obrázek 1.2: Vyberte správnou verzi JDK pro váš operační systém

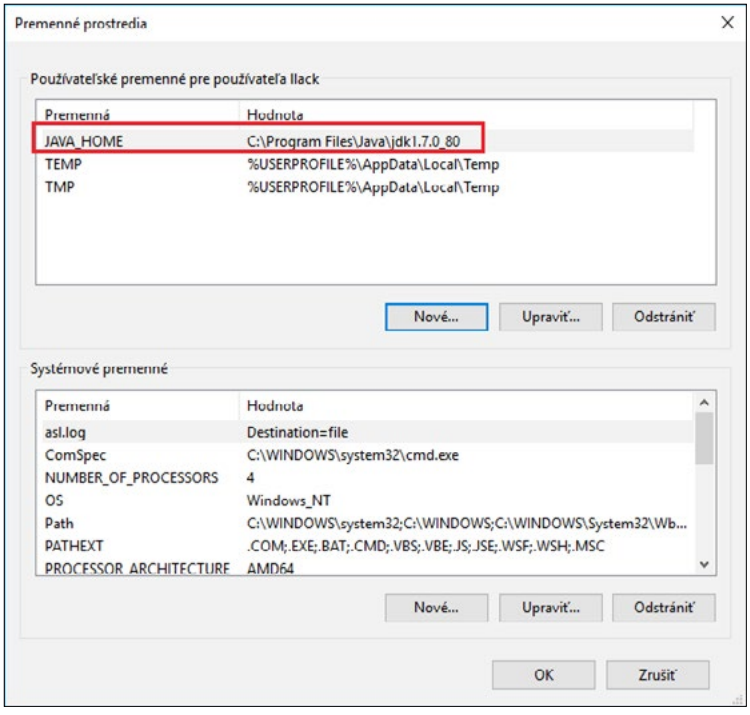


Obrázek 1.3: Instalace JDK

Pokud Android Studio nenajde vámi nainstalované JDK, je potřeba nastavit ještě uživatelskou proměnnou `JAVA_HOME`, aby aplikace používající JDK mohly správně pracovat. V systémových nastaveních Windows 10 klepněte na **Rozšířená systémová nastavení** a v zobrazeném dialogu klepněte na tlačítko **Proměnné prostředí**. Přidejte novou uživatelskou proměnnou `JAVA_HOME` a do ní textový řetězec adresáře, ve kterém máte nainstalované JDK.



Obrázek 1.4: Rozšířené systémové nastavení Windows 10



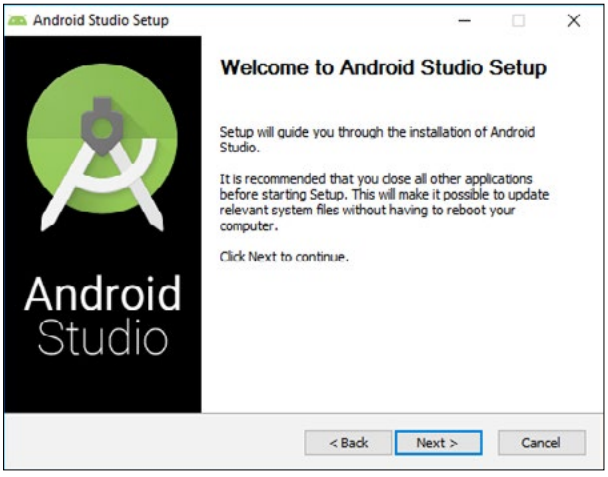
Obrázek 1.5: Nastavení uživatelské proměnné

Instalace vývojového prostředí Android Studio

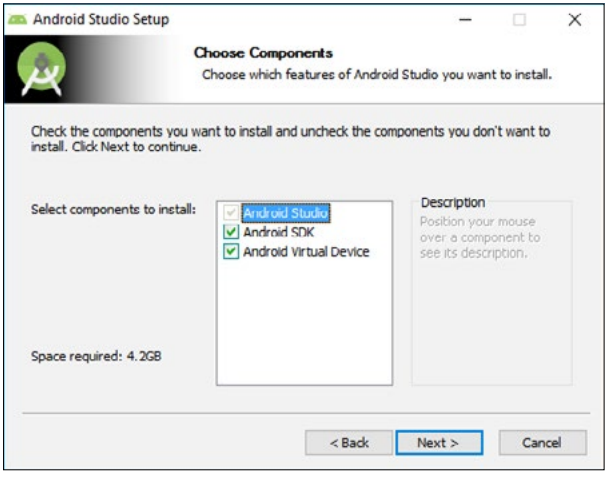
Android Studio jako oficiální nástroj na vývoj aplikací pro Android je k dispozici ke stažení na stránce *developer.android.com* pro platformy Windows, Linux a macOS. Dříve než zahájíte instalaci, nainstalujte balík Java SE Development Kit (viz předchozí podkapitolu). Instalace Android Studia je jednoduchá a proběhne doslova na jedno klepnutí. Stačí stáhnout instalační soubor a spustit ho.

V konfiguračním dialogu instalace doporučujeme ponechat všechny implicitně označené komponenty, tedy:

- Android Studio
- Android SDK
- Android Virtual Device



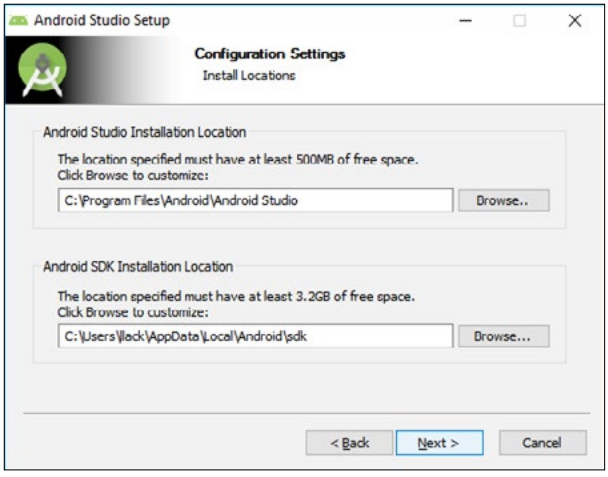
Obrázek 1.6: Instalace vývojového prostředí Android Studio



Obrázek 1.7: Android Studio – výběr komponent k instalaci

V následujícím dialogu se zobrazí složky, do kterých se nainstaluje Android Studio a Android SDK.

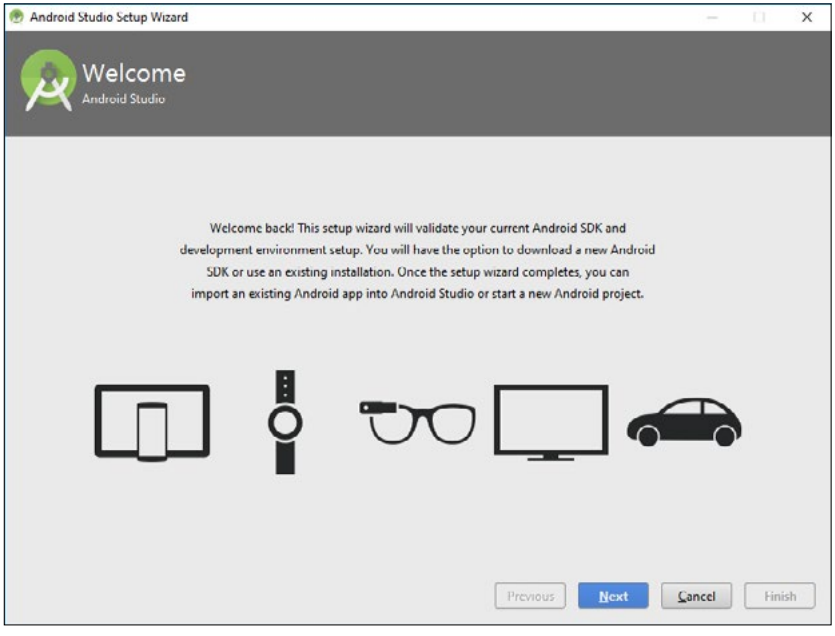
Během instalace se můžete setkat s oznámením firewallu, že Android Studio potřebuje využívat síťovou komunikaci, a firewall si od vás vyžádá povolení k přístupu.



Obrázek 1.8: Android Studio – implicitní adresáře k instalaci

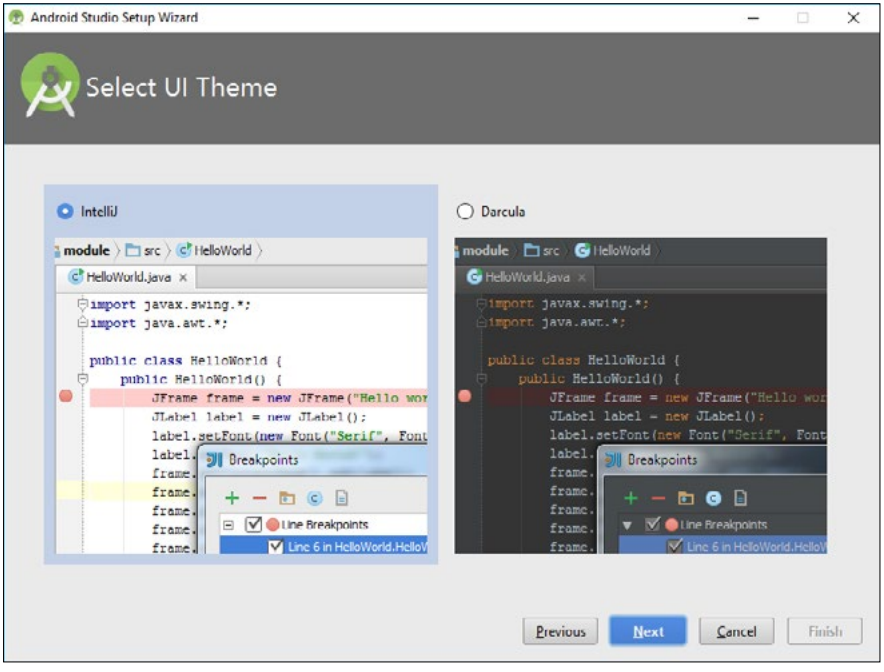
První spuštění a konfigurace Android Studia

V úvodním dialogu **Android Studio Setup Wizard** narazíte na oznámení, že bude následovat kontrola dostupnosti a aktuálnosti verzí Android SDK (Software Development Kit) a v případě potřeby se nainstalují potřebné aktualizace.



Obrázek 1.9: Android Studio Setup Wizard

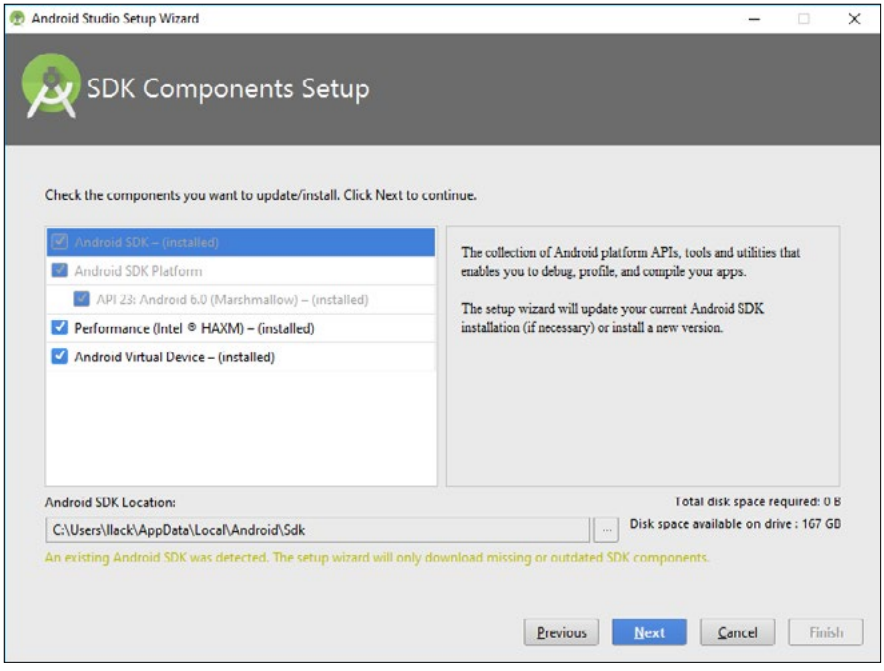
Můžete si zvolit typ instalace **Standard** nebo **Custom**. Pro běžné nasazení doporučujeme ponechat označenou volbu **Standard**.



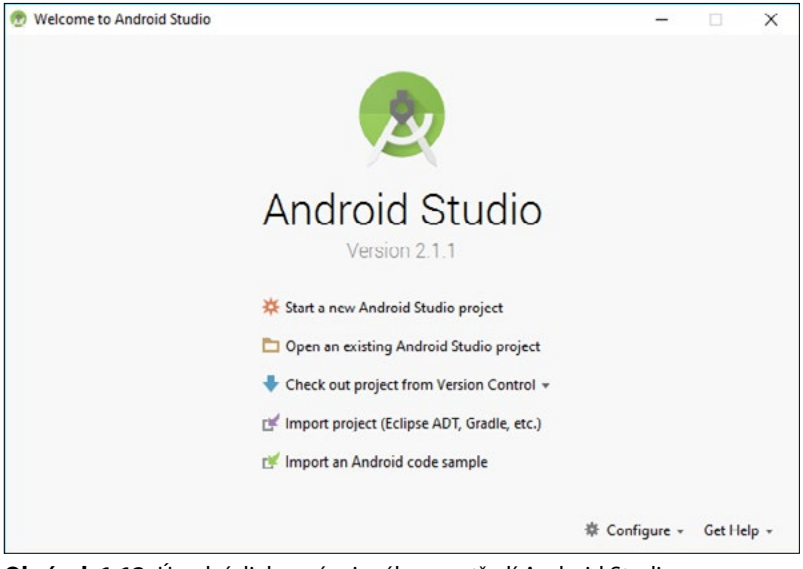
Obrázek 1.10: Volba Custom umožní vybrat barevné schéma uživatelského rozhraní Android Studia

Následuje volba komponent. Pokud už Android SDK máte nainstalované, například pokud jste vyvíjeli aplikace v Eclipse, doinstalují se jen aktualizace chybějící komponenty.

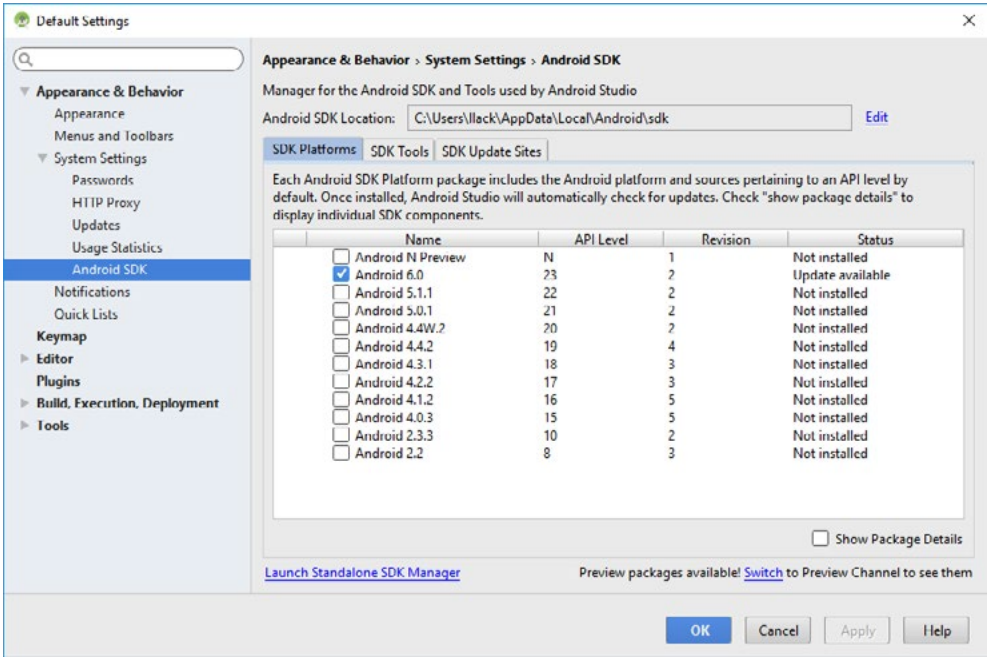
Po nainstalování a prvním spuštění doporučujeme zkontrolovat, pro které verze Androidu máte nainstalovanou podporu. V úvodním dialogu proto ve spodní části klepněte na položku **Configure** a z nové nabídky vyberte **SDK Manager**.



Obrázek 1.11: Volba Custom umožní vybrat barevné schéma uživatelského rozhraní Android Studia



Obrázek 1.12: Úvodní dialog vývojového prostředí Android Studio



Obrázek 1.13: SDK Manager

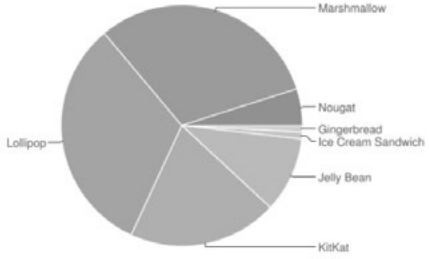
V okně **Android SDK Manager** si všimněte tří karet: **SDK Platforms**, **SDK Tools** a **SDK Update Sites**. Na kartě **SDK Platforms** je seznam SDK pro jednotlivé majoritní verze Androidu. Implicitně je nainstalované SDK pro Android 6.0 Marshmallow, jehož API má verzi 23. Jelikož v době psaní této knihy stále ještě dominovaly nižší verze, doporučujeme nainstalovat i je. V této publikaci budeme jako nejnižší verzi používat Android 5.0.1, tedy verzi API 21. Při rozhodování, zda vyvíjet aplikace, které budou kompatibilní i s verzí Android 4.4 KitKat, případně 4.1 JellyBean, vám poslouží graf procentuálního zastoupení jednotlivých verzí Androidu na *http://developer.android.com/about/dashboards/index.html*. Situaci v době psaní rukopisu dokumentuje obrázek.

Stále mají poměrně velké procentuální zastoupení i starší verze. Podle svého uvážení označte položky, které chcete instalovat, tedy verze Androidu, pro které chcete vyvíjet a testovat aplikace.

Vyvíjet aplikace tak, aby byla zachovaná kompatibilita se staršími verzemi až na výjimky, které jen potvrzují pravidlo, nemá smysl. Chytré telefony nemají dlouhou životnost, takže přístrojů s kdysi velmi rozšířenou verzí 2.2 je dnes již mizivé procento. Verze 3.0 HoneyComb byla určena jen pro tablety a přístrojů s touto verzí se v našich končinách prodalo méně než šafránu. Doporučujeme vydat se spíše cestou inovací. Každý rok je pro vývojáře v předstihu k dispozici předběžná podoba (preview) nové verze, aby si mohli nastudovat využití jejích funkcí. Tuto verzi je možné nainstalovat na novější Nexusy.

Myslíme si, že je velmi užitečné, abyste si SDK pro novou verzi Androidu nainstalovali, vytvořili si pro tyto platformy emulátory a jako vývojáři se s nimi v předstihu seznámili.

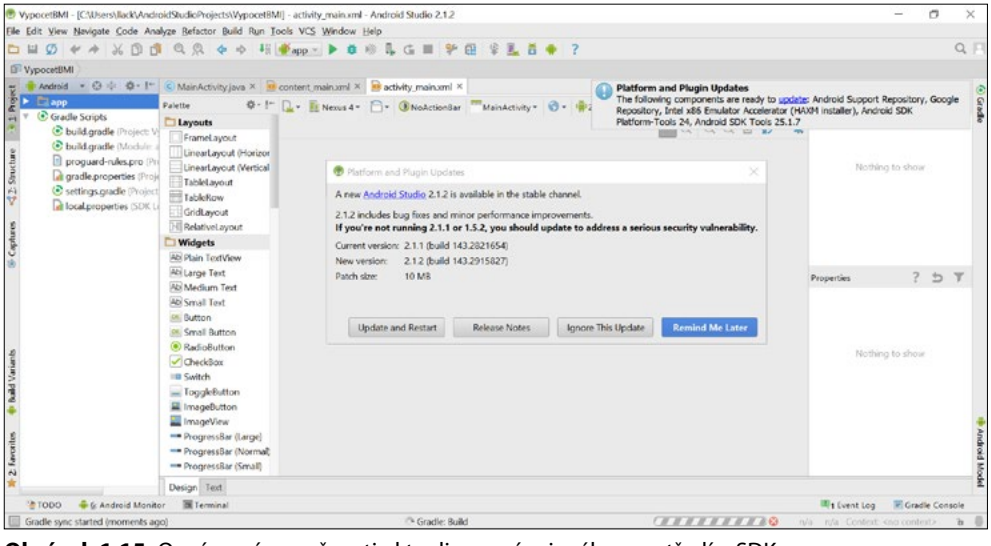
Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.9%
4.1.x	Jelly Bean	16	3.5%
4.2.x		17	5.1%
4.3		18	1.5%
4.4	KitKat	19	20.0%
5.0	Lollipop	21	9.0%
5.1		22	23.0%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	4.5%
7.1		25	0.4%



Obrázek 1.14: Procentuální zastoupení verzí Androidu (duben 2017)

Průběžná aktualizace

Pokud je k dispozici nová verze, opravný balíček, doplněné SDK, vylepšený emulátor a podobně, Android Studio vám to po spuštění oznámí a zobrazí odkazy ke stažení. Doporučujeme udržovat vývojové prostředí aktuální. Jedině tak máte jistotu, že vaše aplikace budou naplno využívat všechny možnosti operačního systému.



Obrázek 1.15: Oznámení o možnosti aktualizace vývojového prostředí a SDK

Nadpis této podkapitoly – *Průběžná aktualizace* – můžete chápat ve dvou rovinách. Udržujte aktuální nejen vývojové prostředí, ale i vaši aplikaci. Každá nová verze Androidu přinese nové a vylepšené funkce, ze kterých by vaše aplikace s vysokou pravděpodobností mohla profitovat. Proto doporučujeme tyto funkce nastudovat, a má-li to má význam, implementovat je formou aktualizace do vaší aplikace.

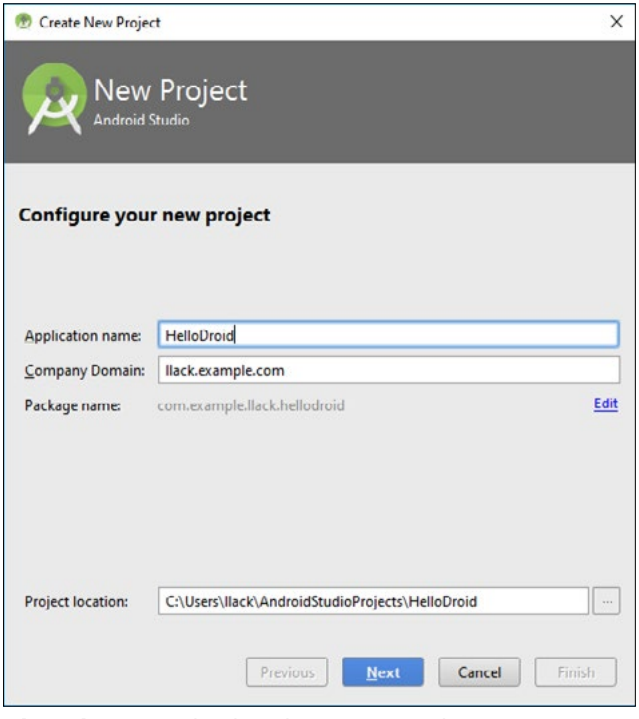
Vytvoření projektu aplikace

Možná se ptáte, proč se pouštíme do vytváření projektu mobilní aplikace dříve, než představíme aspoň v hrubých rysech architekturu operačního systému Android. Důvod je jednoduchý. Pokud se aplikace dá přeložit a spustit nejprve na emulátoru a následně na reálném zařízení, máte jistotu, že máte správně nainstalované a nakonfigurované vývojové prostředí, překladač jazyka Java, Android SDK, emulátor a propojení na reálné zařízení.

První projekt má v tomto případě i motivační význam, jelikož doslova na jedno klepnutí a bez jakéhokoliv programování vytvoříte aplikaci typu „Hello World", která vypíše na obrazovku text.

Začátečníci se v tomto příkladu nemusí snažit pochopit souvislosti. Návod na vytvoření cvičného projektu je uveden jako obrázkový postup krok za krokem.

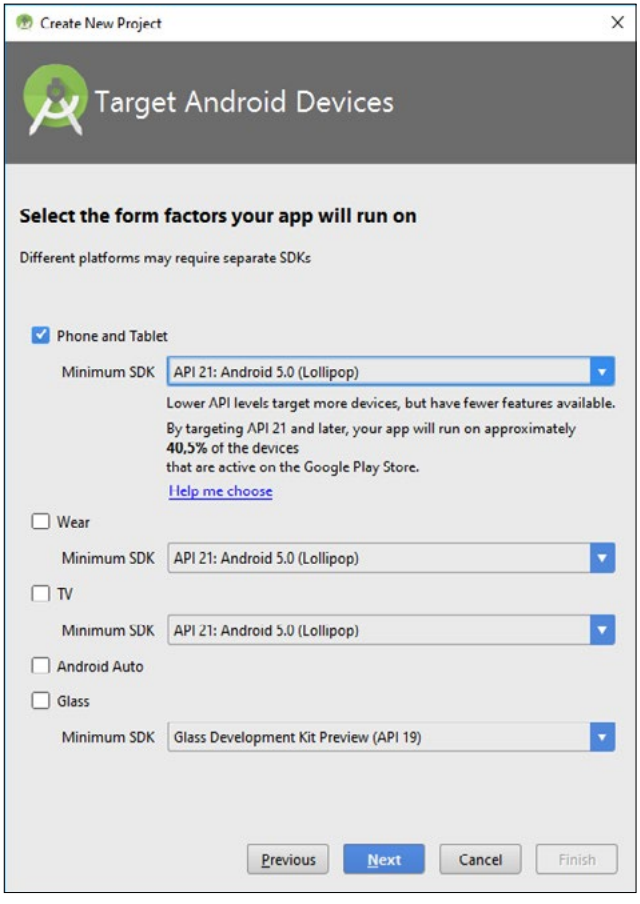
Spusťte vývojové prostředí Android Studio. V úvodním dialogu klepněte na položku **Start a new Android Studio project**. V úvodním dialogu průvodce vytvořením projektu je potřeba zadat název projektu a případně i firemní či vývojářovu doménu.



Obrázek 1.16: Android Studio Setup Wizard

Důležitý je výběr minimální verze SDK. Momentálně je nejvyšší komerčně dostupnou verzí Android 7.0 (Nougat) – verze API 24. Jako nejnižší verze je v aktuální verzi Android Studia implicitně nastavená verze 4.0.3 (IceCreamSandwich) – verze API 15. Můžete tak bez omezení používat nové prvky uživatelského rozhraní. V komentáři je uvedeno, jaké procento uživatelů využívá tuto a vyšší verze Androidu. Statistika se týká zařízení využívajících aplikační obchod Google Play. Doba životnosti mobilního telefonu je maximálně dva až tři roky, podobně je tomu i u tabletů, takže přístrojů se staršími verzemi rapidně ubývá. V našich projektech budeme používat jako nejnižší verzi Android 5.0 (Lollipop) – verze API 21. V době psaní publikace mělo tuto (a vyšší) verzi nainstalovano více než 68 procent zařízení a jejich podíl bude rychle stoupat.

Všimněte si, že můžete vytvářet projekty nejen pro telefony a tablety, ale i pro zařízení Android Wear, například hodinky, automobily, televizory a brýle Google Glass.

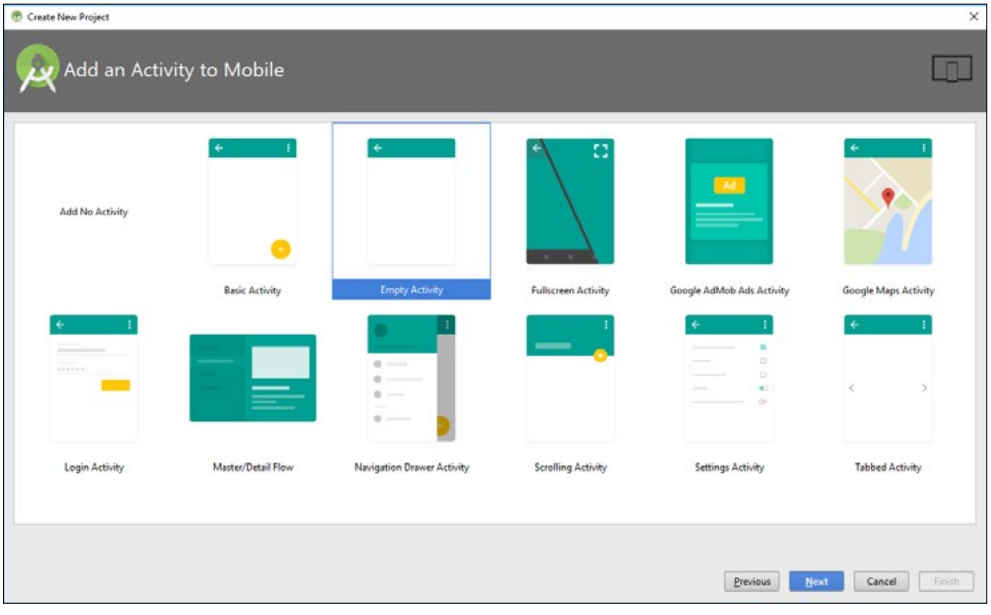


Obrázek 1.17: Výběr nejnižší podporované verze Androidu

Z projektu tedy můžete vytvořit aplikace pro dvě nebo více sesterských platforem. Vytvoření tandemu více projektů je logické. Uživatel bude mít s sebou nositelné zařízení zároveň s telefonem a tato zařízení budou mezi sebou komunikovat. Také může z telefonu nebo tabletu ovládat televizor.

Platforma Android Wear je příležitostí pro vývojáře na vytvoření užitečných informací pro nové typy zařízení. Návrh a vývoj aplikace pro zařízení, která nosíme na sobě, například hodinky, brýle a podobně, vzhledem ke specifikám těchto zařízení vyžaduje jinou filozofii než návrh aplikací pro mobilní telefony. Pokud si do tabletu nebo telefonu uživatel nainstaluje aplikaci využívající součinnost s hodinkami, část aplikace běžící v hodinkách se do nich nainstaluje automaticky.

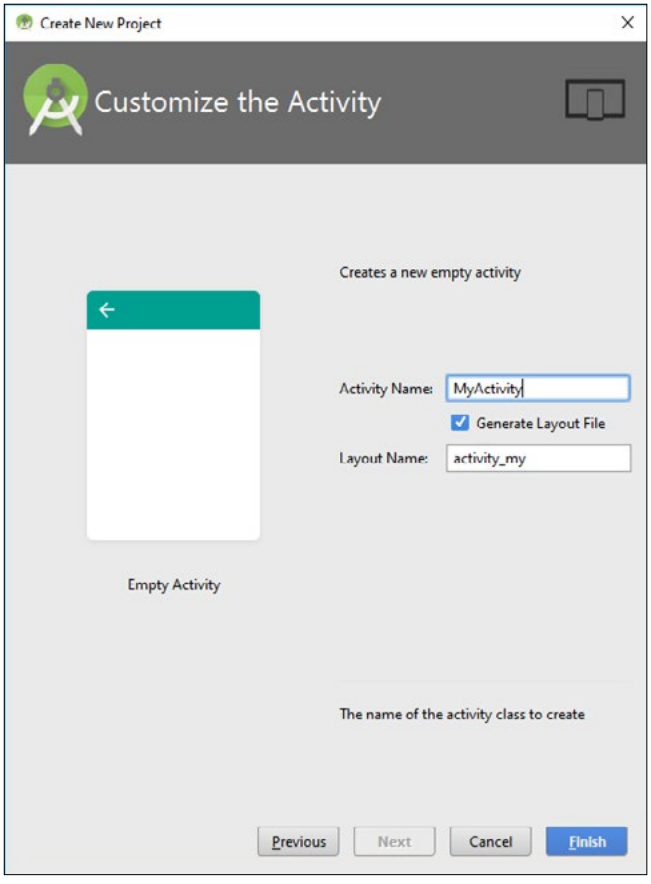
V dalším dialogu můžete zvolit několik typů pro hlavní aktivity aplikace včetně mapové či aktivity typu master-detail. Další postup se odvíjí od nastaveného typu hlavní aktivity. Například pokud jste zvolili typ master-detail, systém vás vyzve k definování názvu položky a skupiny položek. V našem příkladu se spokojíme s implicitně vybranou volbou **Empty Activity**.



Obrázek 1.18: Výběr typu hlavní aktivity

V následujícím dialogovém okně **Customize the Activity** můžete nakonfigurovat detaily pro vybraný typ aktivity. V poli **Activity Name** změňte název hlavní aktivity na `MyActivity`. Ponechejte označenou volbu **Generate Layout File** a klepnutím na tlačítko **Finish** vytvořte projekt.

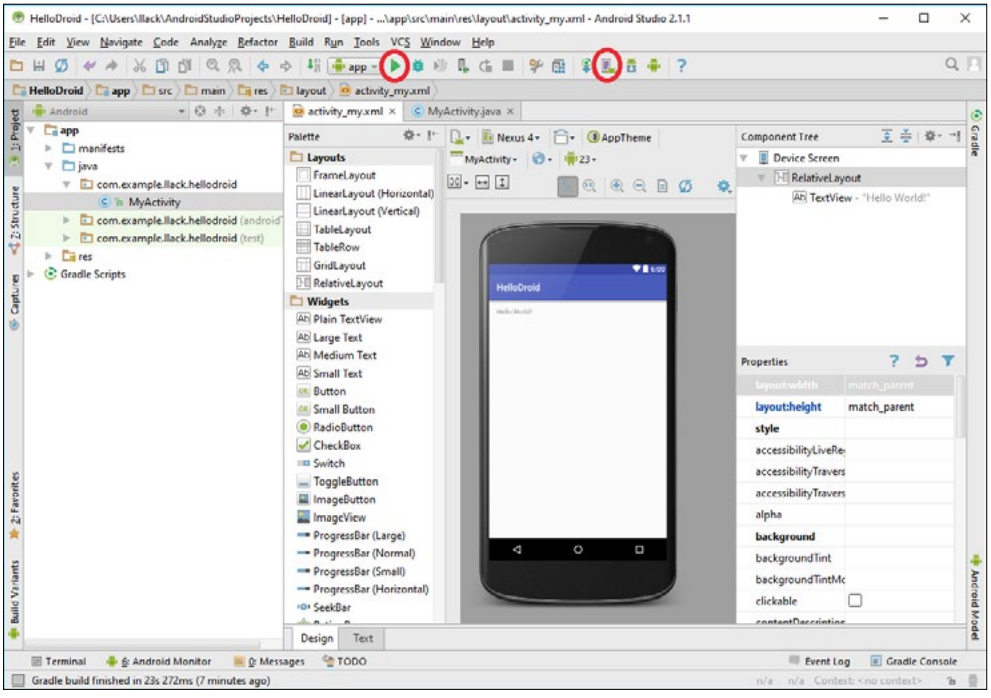
Než se vytvoří všechny potřebné soubory tvořící projekt, bude se několik sekund zobrazovat pouze šedé okno Android Studia s ukazatelem průběhu v dolní části a odhadem času, kolik sekund ještě příprava zabere. Pokud jste dosud nepovolili firevall pro Javu, zobrazí se vám dialog, ve kterém to můžete provést.



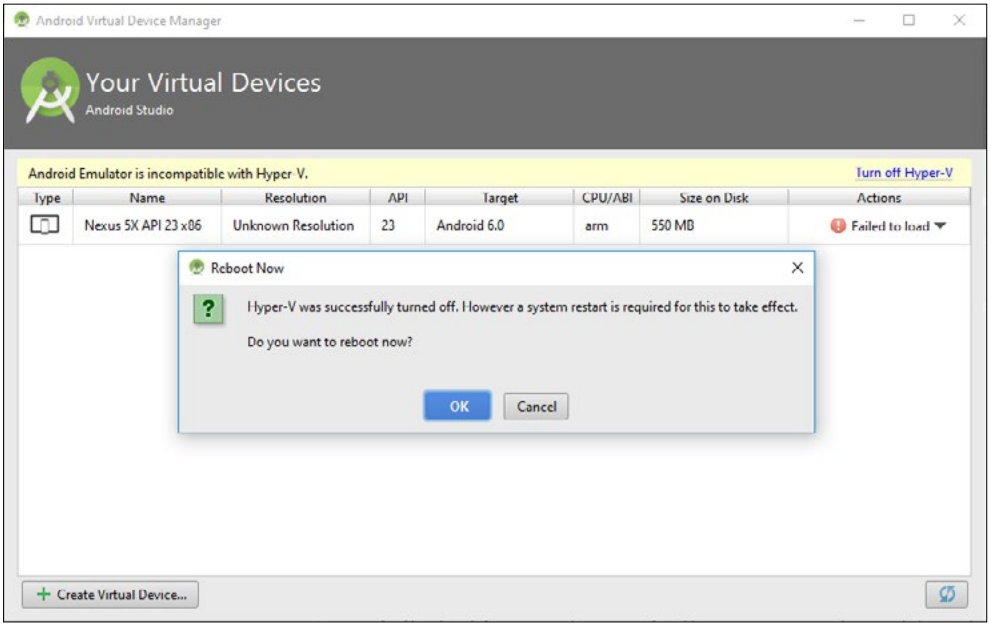
Obrázek 1.19: Vytvoření hlavní aktivity

Potom se zobrazí pracovní obrazovka včetně vizuálního návrhu uživatelského rozhraní aplikace. Zatím bude mít aplikace pouze jedinou funkcionalitu – zobrazí text „Hello World“. S anatomií projektu se seznámíte v dalších kapitolách. V této fázi se pokusíte projekt spustit, nejprve na emulátoru a následně na reálném zařízení.

Klepnutím na zelenou šipku můžete aplikaci spustit v emulátoru mobilního zařízení. V dialogu **Device Chooser** ponechejte označenou volbu **Launch Emulator**. V poli **Android virtual device** bude implicitně nastavený emulátor zařízení Nexus 4. Ponechejte tuto implicitní volbu. Pokud jste vytvořili více emulátorů, například emulátor zařízení typu chytrý telefon a emulátor zařízení typu tablet, vyberte nejvhodnější emulátor pro daný příklad. Některé emulátory nejsou kompatibilní s hypervizorem Hyper-V, který je součástí novějších verzí Windows. V tom případě budete muset klepnutím na odkaz hypervizor vypnout a vývojářský počítač restartovat.

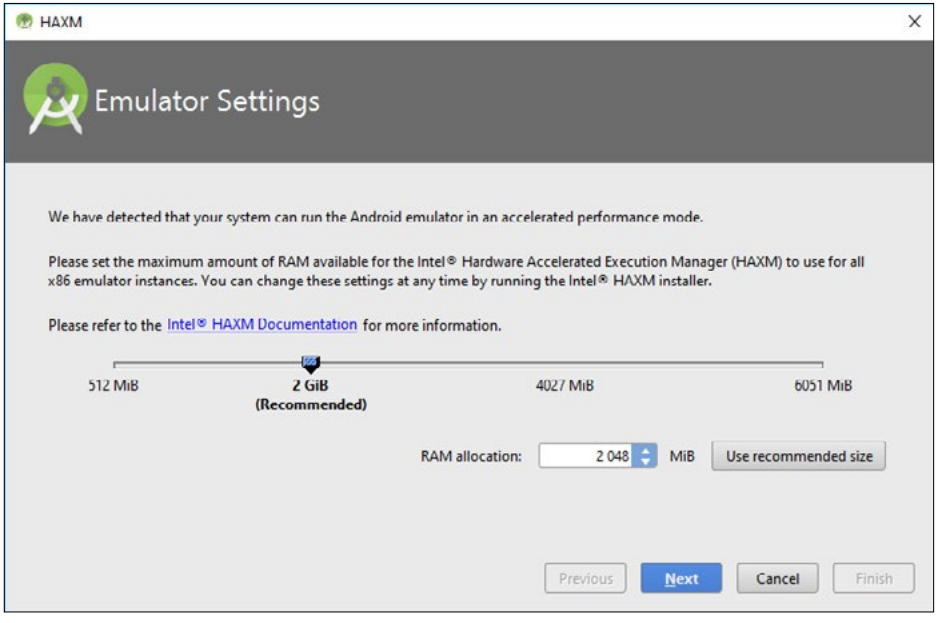


Obrázek 1.20: Nový projekt v Android Studiu. Označené jsou ikony na spuštění aplikace a ikona AVD manageru na konfiguraci emulátorů



Obrázek 1.21: Oznámení o nutnosti vypnutí Hyper-V

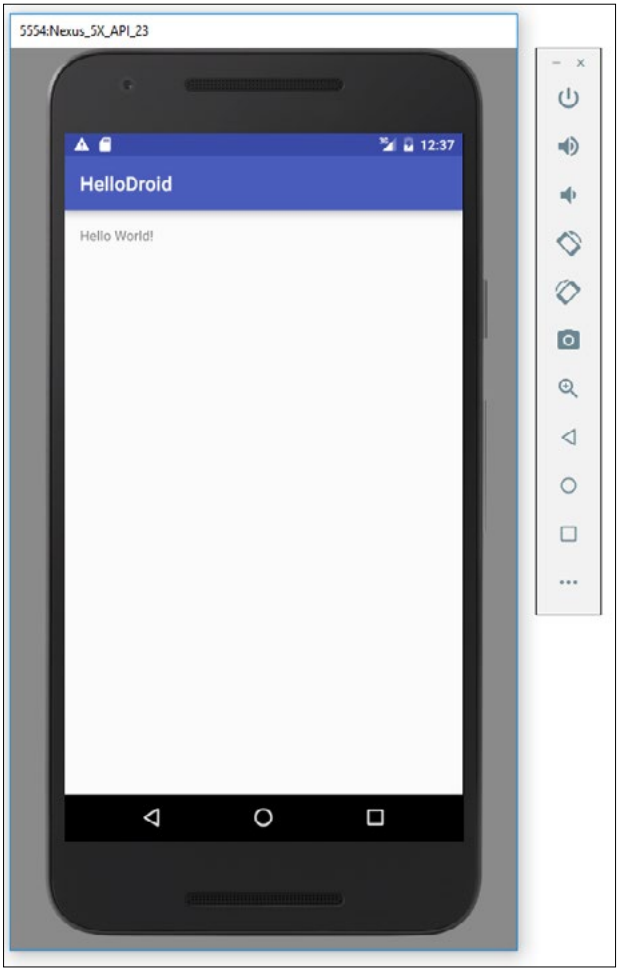
Po restartu vynuceném vypnutím hypervizoru Hyper-V bude potřeba nainstalovat Intel Hardware Accellerated Execution Manager (HAXM) a vyhradit mu adekvátní část paměti RAM. V našem případě měl počítač instalovaných 8 GB RAM a pro HAXM jsme vyhradili doporučenou velkost 2 GB.



Obrázek 1.22: Instalace Intel Hardware Accellerated Execution Manager (HAXM)

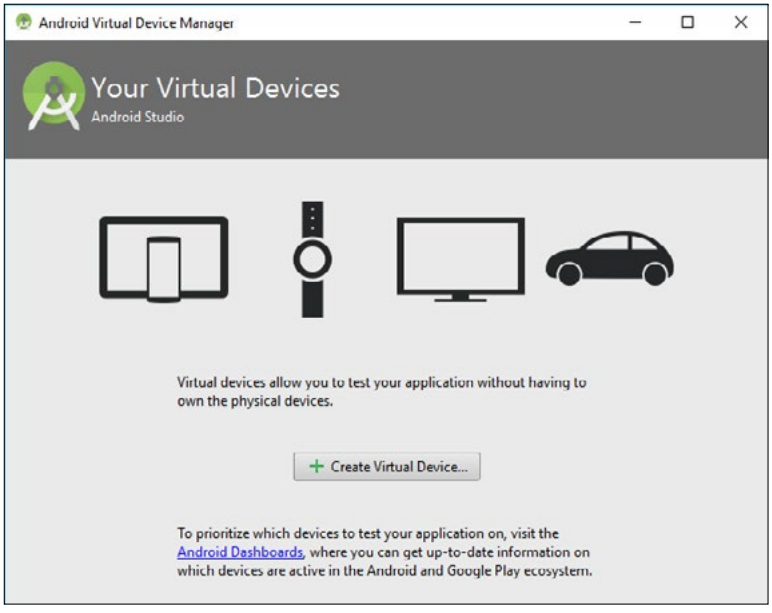
Tento úkon, tedy vypnutí Hyper-V a instalace HAXM, se samozřejmě provádí pouze jednou. Proto jsme zařadili projekt typu Hello World do kapitoly o instalaci, jelikož až po přeložení, linkovaní, spuštění na emulátoru a následně i spuštění projektu na reálném zařízení máte jistotu, že vývojové prostředí, SDK a emulátory jsou správně nainstalované a nakonfigurované.

Po náběhu emulátoru a jeho odemknutí se vaše aplikace automaticky spustí. V okně **Console** v dolní části pracovní plochy můžete sledovat průběh sestavení projektu a jeho zavedení do emulátoru.

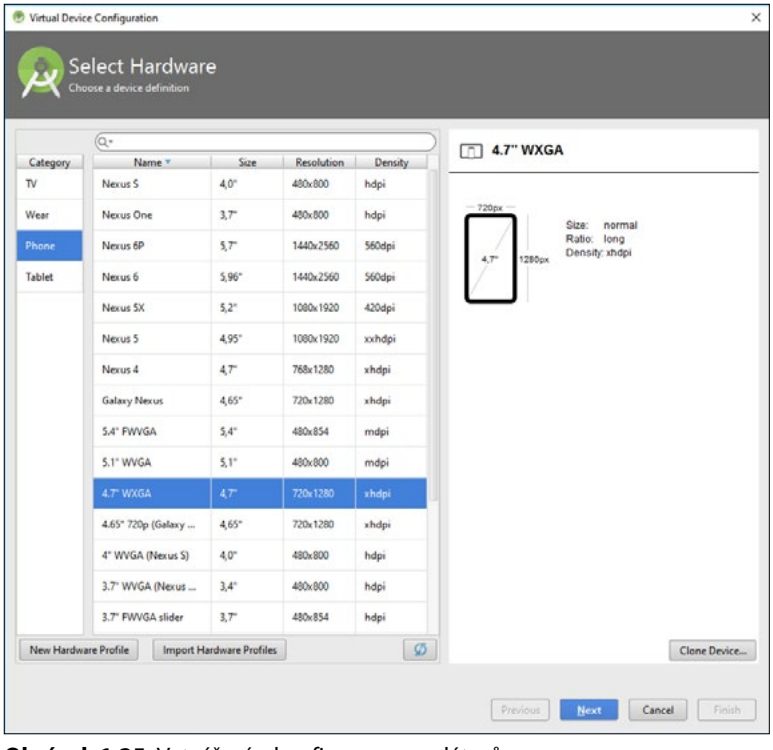


Obrázek 1.23: Spuštění aplikace v emulátoru

Pokud by se vám nepodařilo spustit implicitně nastavený emulátor, například pokud nemáte na vývojářském počítači dostatek operační paměti, můžete si nakonfigurovat jiný emulátor s menšími nároky na hardwarové zdroje. V Android Studiu klepněte na ikonu **AVD manager** (čtvrtá zleva). Zobrazí se **Android Virtual Device Manager**.



Obrázek 1.24: Android Virtual Device Manager



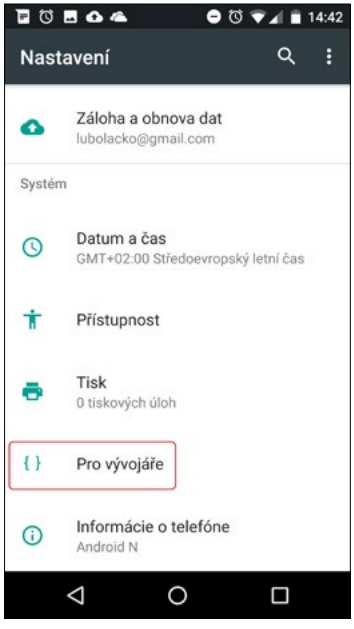
Obrázek 1.25: Vytváření a konfigurace emulátorů

Odemknutí zařízení pro vývoj

Abyste mohli vytvářet a spouštět aplikace na svém zařízení (bez ohledu na to, zda se jedná o telefon nebo tablet) musíte nejdříve aplikovat následující postup. V aplikaci **Nastavení** přejděte do nabídky **Informace o zařízení** a následně vyhledejte položku **Číslo sestavení**. U některých nadstaveb Androidu je tato položka vnořená na obrazovce **Informace o softwaru**. Další postup se vám možná bude zdát aprílový, ve skutečnosti však není. Klepejte postupně na položku **Číslo sestavení**, musíte klepnout až sedmkrát. Pokud například po šestém klepnutí přestanete, Android vás povzbudí, že se jedná o seriózní postup, aby se z vás stal vývojář, a zobrazí vám oznámení „Jste pouhé 1 klepnutí od toho, abyste se stali vývojářem", jež avizuje, kolikrát je potřeba ještě klepnout. Po posledním klepnutí se zobrazí oznámení „Nyní jste vývojářem" a v **Nastavení** přibude položka **Pro vývojáře**.

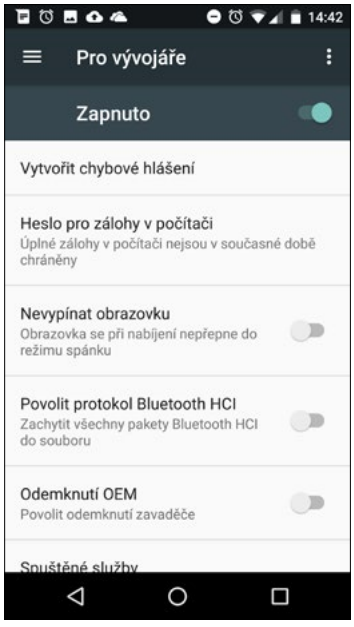


Obrázek 1.26: Postup odemknutí položky Pro vývojáře nalezení položky Číslo sestavení



Obrázek 1.27: Položka Nastavení Pro vývojáře po odemknutí

Proč takový neobvyklý postup? Na zařízeních s operačním systémem Android 4.2 a vyšším je položka **Pro vývojáře** implicitně skryta a musíte ji nejprve zobrazit. Vývojářský režim totiž umožňuje přímé zavedení aplikace z vývojářského počítače do zařízení s Androidem přes USB a to může být v případě aplikace z neznámého až podezřelého zdroje značně riskantní. V případě vývoje vlastní aplikace nic neriskujete, víte přesně, jakou aplikaci jste vytvořili a co bude dělat.

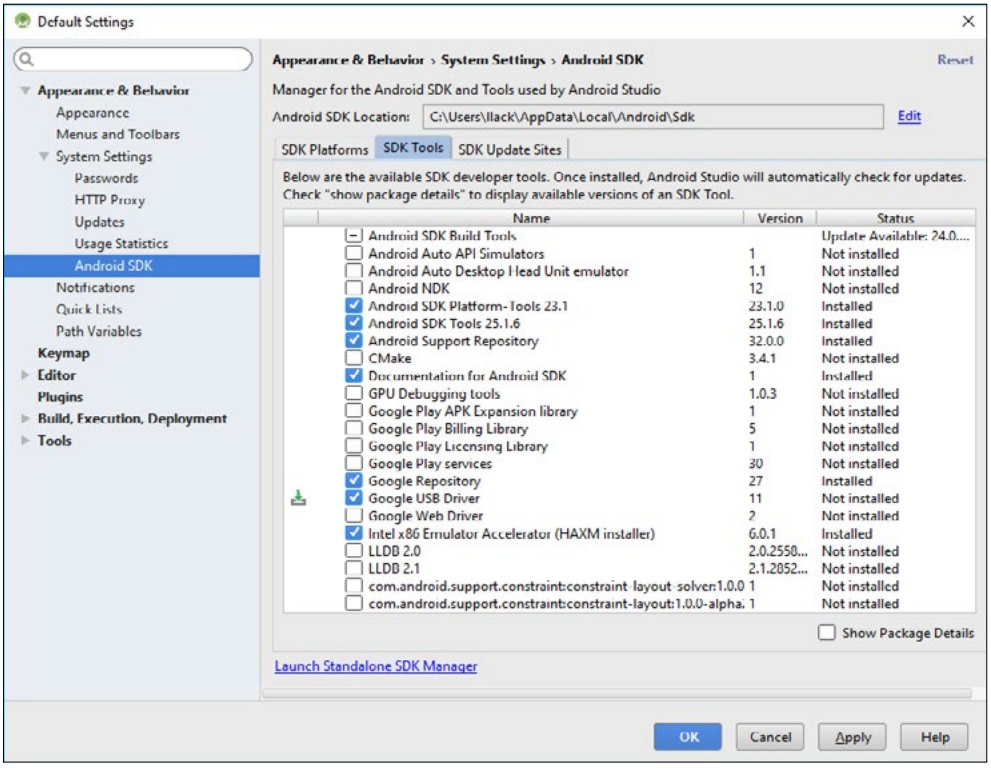


Obrázek 1.28: Seznam voleb, které můžete nastavit v nabídce Pro vývojáře

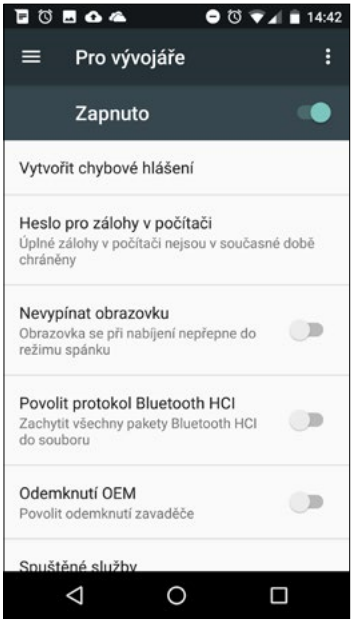
Na straně počítače potřebujete pro komunikaci se zařízením s Androidem USB ovladače pro ADB (Android Debug Bridge). Buď použijete ovladač Google USB Driver, který je součástí Android SDK, nebo ovladač dodaný výrobcem zařízení.

Spuštění aplikace na reálném zařízení

Aby bylo možné aplikaci spustit na reálném zařízení připojeném přes USB, je potřeba na vývojářském počítači nainstalovat USB ovladače pro ADB (Android Debug Bridge). Potom stačí připojit zařízení, které má povolené ladění přes USB. Pro zařízení Nexus, případně některé další, postačí Google USB Driver, který doinstalujete přes **SDK Manager** na kartě **SDK Tools**. Pro ostatní zařízení je potřeba ovladač doinstalovat ze stránky výrobce. Některá zařízení (Lenovo, Huawei apod.) mají možnost po připojení zařízení přes USB nastavit, aby se zařízení chovalo jako virtuální CD ROM, na kterém jsou ovladače.

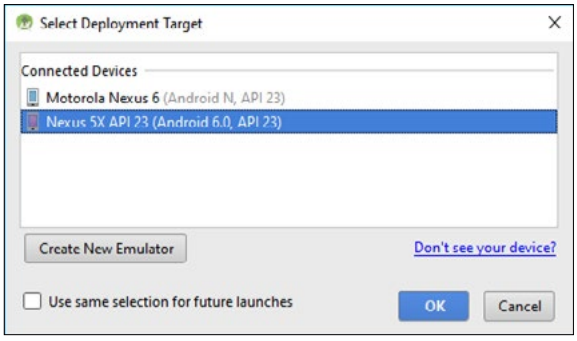


Obrázek 1.29: Instalace ovladače Google USB Driver přes SDK Manager



Obrázek 1.30: Zapnutí ladění přes USB na připojeném zařízení

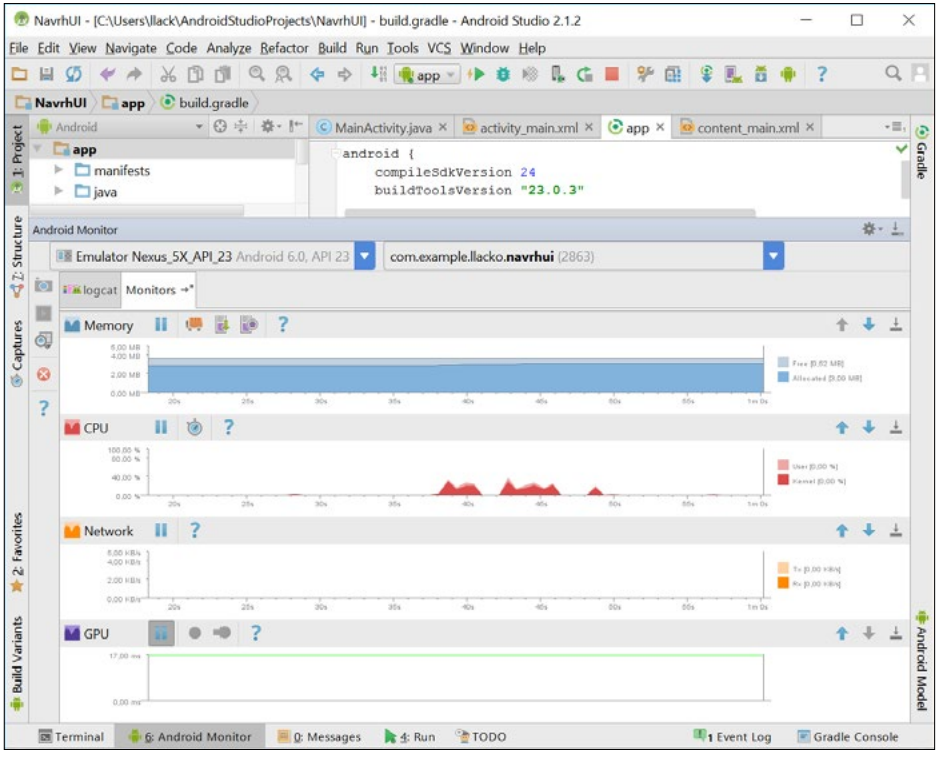
Po správném nakonfigurování USB ovladače pro ADB se při pokusu o spuštění aplikace zobrazí nabídka možností.



Obrázek 1.31: Výběr hardwarového zařízení ke spuštění aplikace

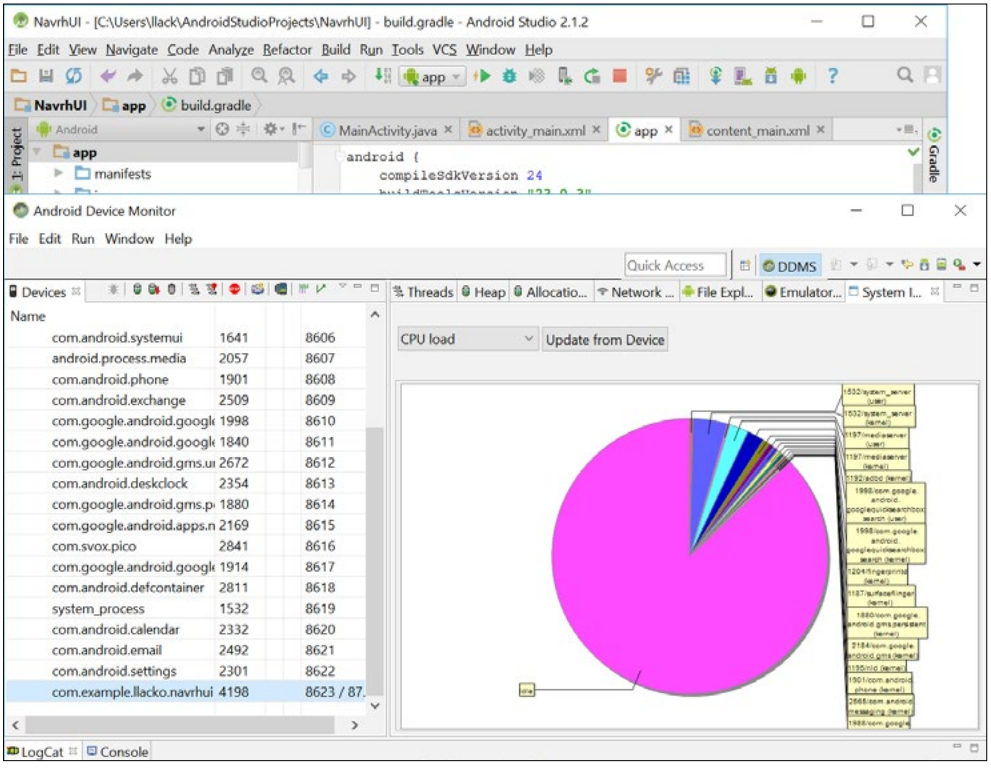
Monitorování spuštěné aplikace

Hodně zajímavých informací se dozvíte i za běhu aplikace. Po spuštění aplikace – ať už v emulátoru, nebo na reálném zařízení připojeném přes USB – se v dolní části plochy vývojového prostředí Android Studio zobrazí okno **Android Monitor**. Okno má dvě záložky: **LogCat**, kde se vypisují události, ať už implicitní, nebo události, které do protokolu zapisujete sami. Příklad najdete v části o životních cyklech aktivity. Na druhé záložce **Monitor** můžete například sledovat, jak vaše aplikace zatěžuje procesor, GPU, paměť, či síťové připojení.



Obrázek 1.32: Monitorování spuštěné aplikace v okně Android Monitor

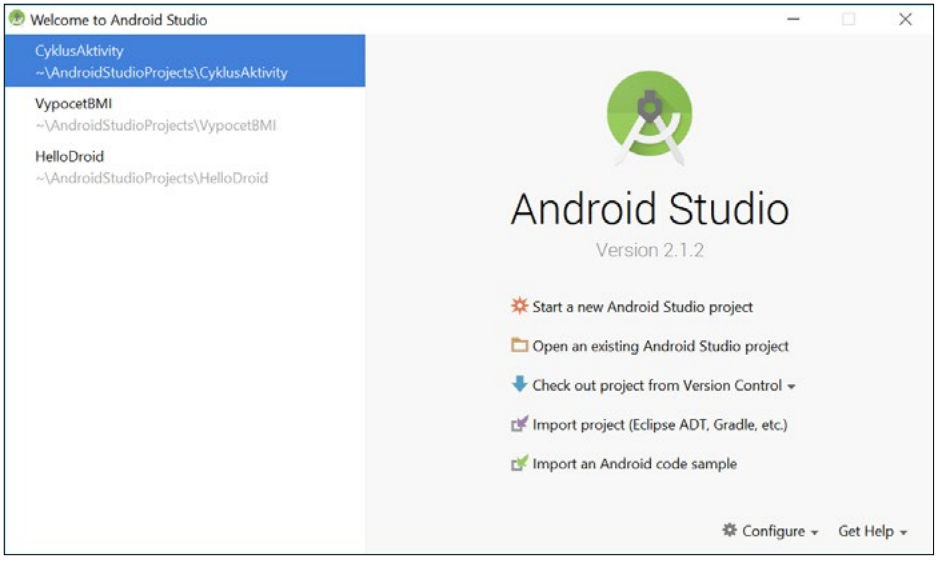
K monitorování spuštěné aplikace můžete využít i **Android Device Monitor**. Z Android Studia tento nástroj spustíte pomocí nabídky **Tools → Android → Android Device Monitor**.



Obrázek 1.33: Monitorování spuštěné aplikace pomocí nástroje Android Device Monitor

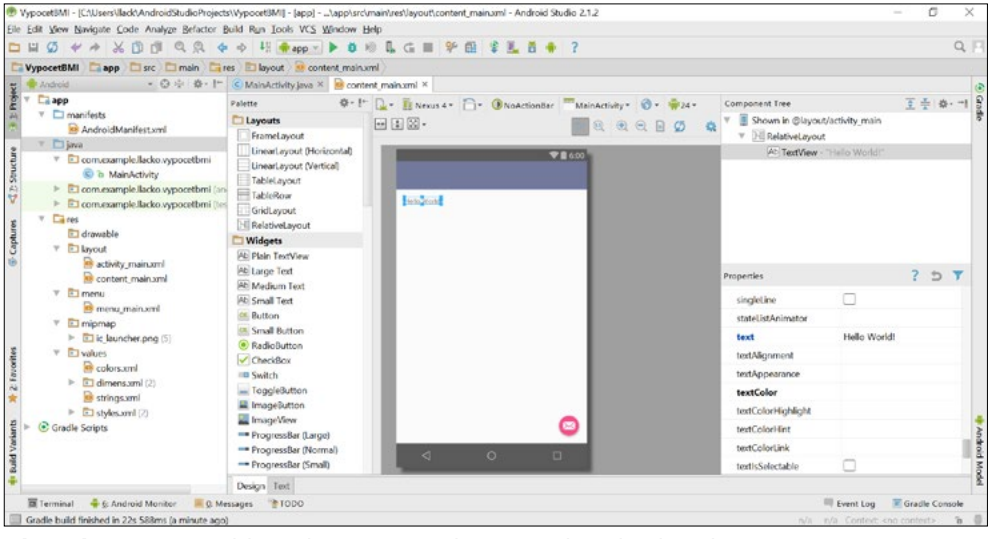
Seznámení se s vývojovým prostředím

Po spuštění vývojového prostředí Android Studio se zobrazí uvítací dialog s nabídkou nejčastěji prováděných činností. Můžete vytvořit nový projekt, případně otevřít existující, rozpracovaný projekt. Pro vývojáře, kteří dosud používali jiné vývojové prostředí, například Eclipse + ADT, je zajímavá možnost přímého importu projektů z dosud používaných vývojářských nástrojů do Android Studia. Také můžete importovat a testovat různé vzorové příklady. Zároveň tento dialog poskytuje informaci o aktuální verzi Android Studia a nabízí možnost konfigurace jeho součástí.



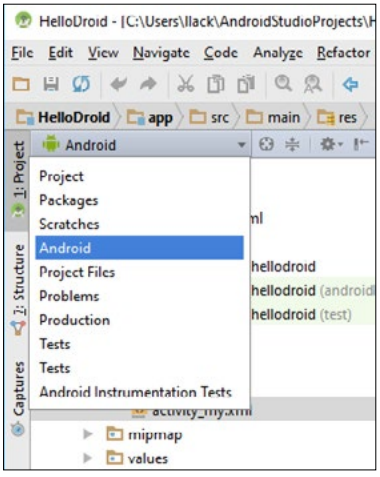
Obrázek 1.34: Úvodní dialog vývojového prostředí Android Studio

Rozmístění pracovních oken vývojového prostředí je na obrázku. Jejich velikost, přesněji poměry velikosti, lze podle potřeby měnit.



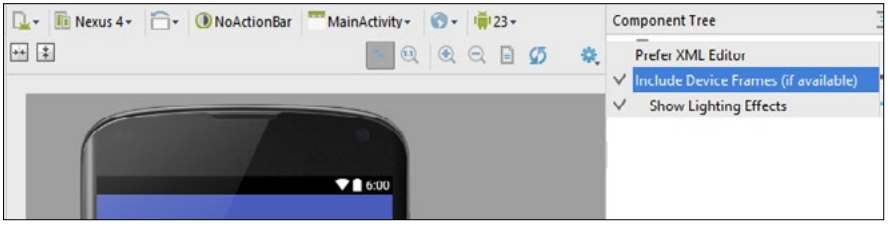
Obrázek 1.35: Uživatelské rozhraní vývojového prostředí Android Studio

V levém svislém podlouhlém okně se zobrazuje struktura projektu uspořádaná přehledně hierarchicky. Detailního zobrazení struktury projektu docílíte nastavením volby **Android**.



Obrázek 1.36: Možnosti zobrazení struktury projektu

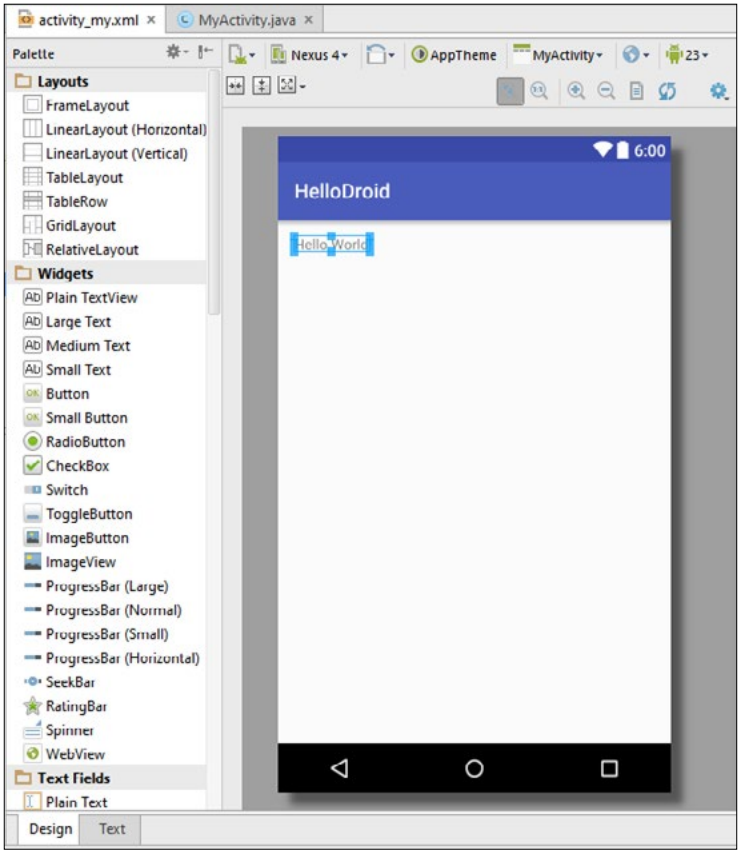
Dominantním při návrhu uživatelského rozhraní je náhledové okno. Po nainstalování se zobrazuje i s rámečkem znázorňujícím tělo telefonu. Na vývojářských noteboocích s menšími displeji je to tak trochu plýtvání místem. Tento rámeček lze ale naštěstí vypnout. Vpravo nahoře na liště okna najdete ikonu s ozubeným kolečkem, která zobrazí nabídku nastavení. V této nabídce zrušte zaškrtnutí políčka **Include Device Frames**. Potom se bude zobrazovat pouze displej, takže budete moct zobrazit větší uživatelské rozhraní.



Obrázek 1.37: Nastavení zobrazení – možnost vypnutí rámečku

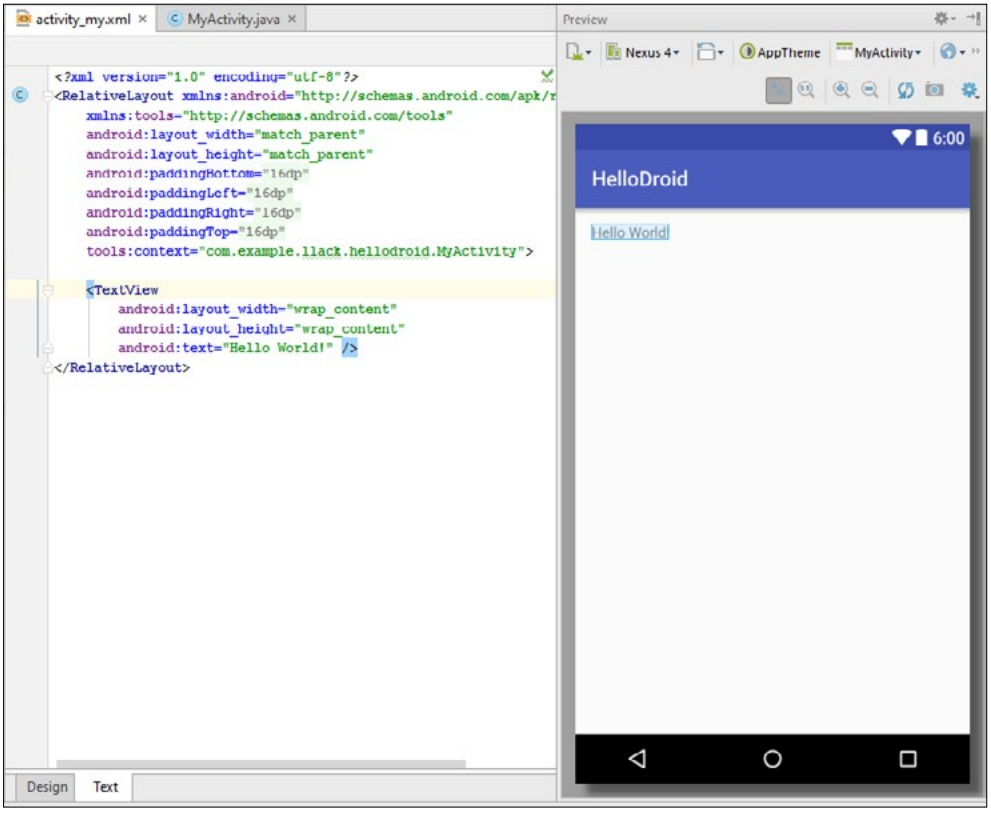
Uživatelské rozhraní můžete navrhnout ve dvou módech: **Design** a **Text**. Přepínají se pomocí karet ve spodní části okna.

V režimu **Design** je vlevo vedle okna s návrhem uživatelského rozhraní ve sloupcovém okně **Palette** zobrazena paleta komponent, uprostřed vizuální reprezentace uživatelského rozhraní a vpravo hierarchie komponent a vlastnosti vybrané komponenty. Komponenty z palety můžete přesouvat na plochu aplikace a vhodně rozmístit.



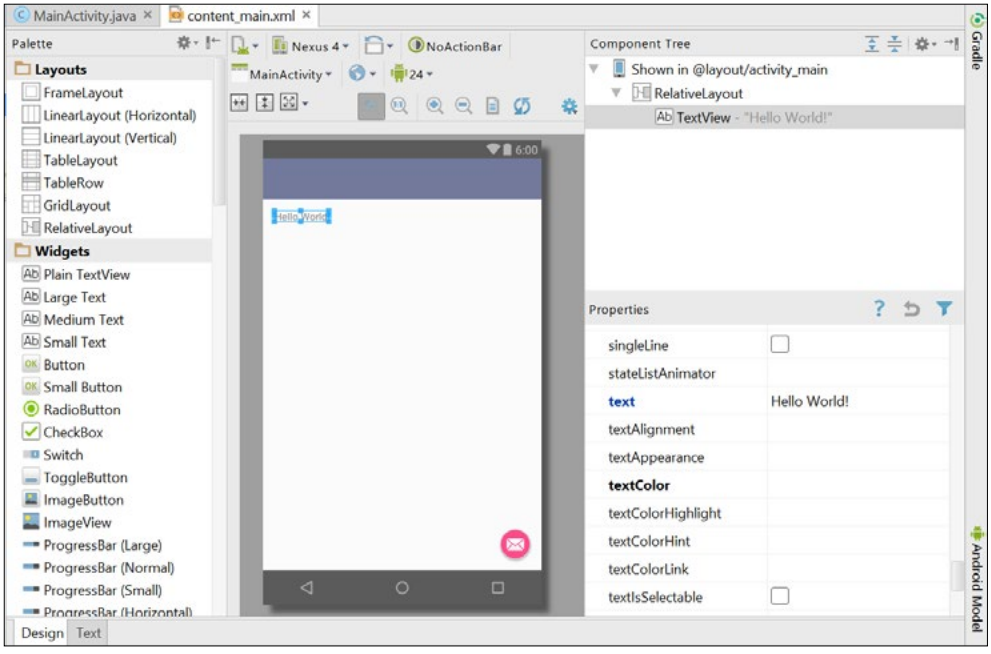
Obrázek 1.38: Návrh uživatelského rozhraní – mód Design

V režimu **Text** se vlevo zobrazuje kód XML definice návrhu uživatelského rozhraní a vpravo jeho vizuální reprezentace. Změny zrealizované v návrhovém kódu se automaticky průběžně promítají do vizuálního zobrazení.



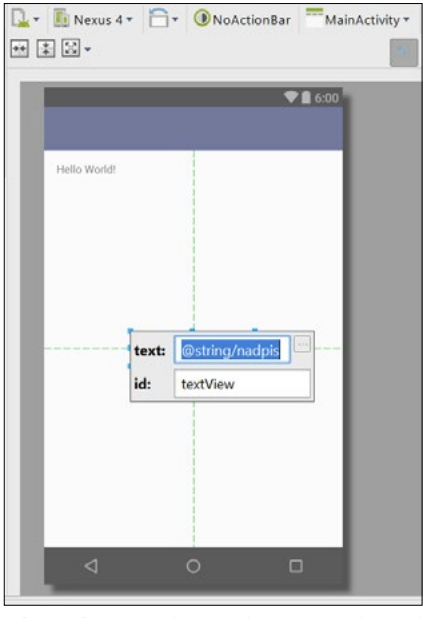
Obrázek 1.39: Návrh uživatelského rozhraní – mód Text

V pravé části pracovní obrazovky Android Studia najdete ještě dvě důležitá okna. V okně **Component Tree** je hierarchická struktura prvků uživatelského rozhraní. Toto okno poskytuje perfektní přehled, jak jsou prvky hierarchicky zapouzdřeny ve vizuálních kontejnerech, případně jak jsou kontejnery zapouzdřeny navzájem. V okně **Properties** se zobrazují vlastnosti vybraného prvku.



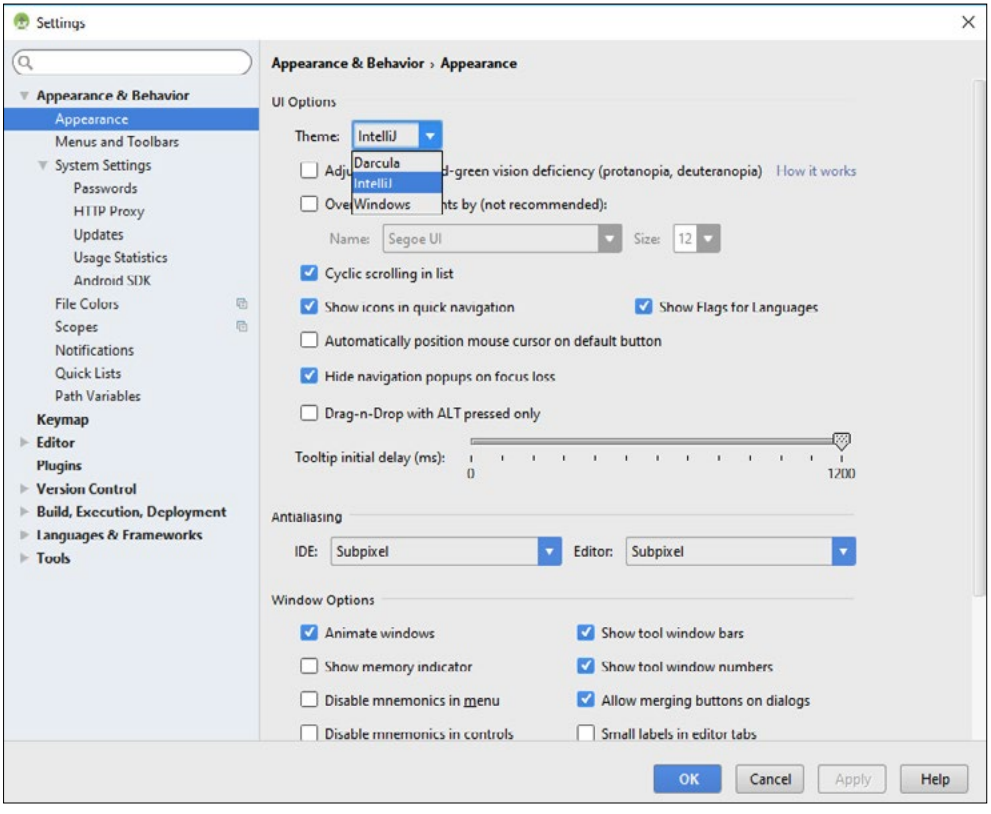
Obrázek 1.40: Vlastnosti vybraného prvku v okně Properties

Klíčovou vlastnost vybraného prvku – například v případě prvku `EditText` je to implicitně definovaný textový řetězec – zobrazíte poklepáním na prvek v návrhovém zobrazení.



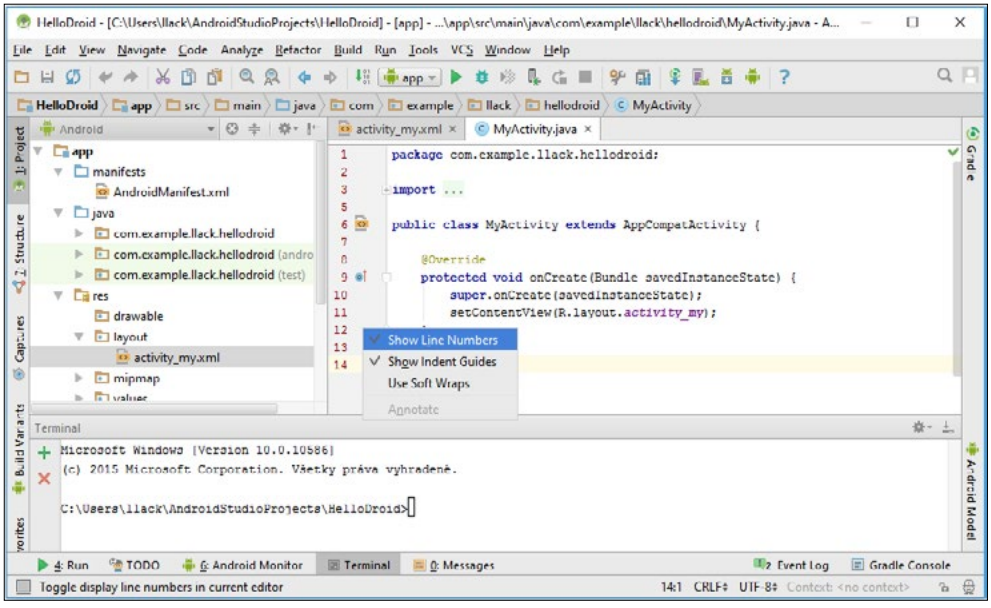
Obrázek 1.41: Klíčové vlastnosti vybraného prvku

Pokud vám více vyhovuje tmavý barevný design uživatelského rozhraní vývojového prostředí Android Studio, pomocí nabídky **File → Settings** zobrazte dialogové okno nastavení a na kartě **Appearance** nastavte položku **Theme** na hodnotu **Darcula**.



Obrázek 1.42: Změna barevného schématu vývojového prostředí Android Studio

U zobrazení zdrojového kódu v programovacím jazyce Java můžete pomocí místní nabídky zapnout zobrazování čísel řádků. Pomůže vám to například při identifikaci a lokalizaci případné chyby.

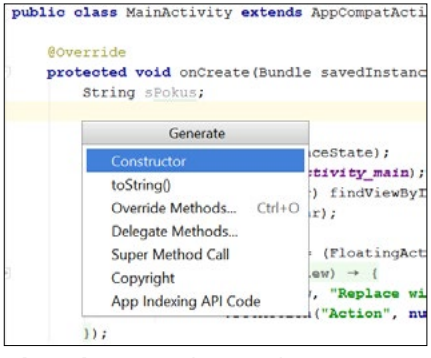


Obrázek 1.43: Zobrazení zdrojového kódu v programovacím jazyce Java

Pro některé často opakované úkony nabízí prostředí klávesové zkratky. Jejich seznam získáte v nabídce **Help** pod položkou **Default Keymap Reference**. Zobrazí se seznam zkratek ve formátu PDF, který si případně můžete vytisknout a umístit na vhodné místo jako tahák.

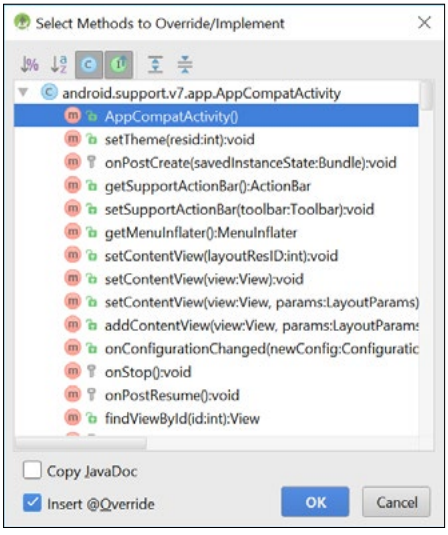
Editor kódu v jazyce Java

Vývojové prostředí Android Studio se snaží co nejvíce zjednodušit a zefektivnit psaní kódu. Pomocí místní nabídky a položky **Generate** nebo klávesové zkratky **Alt+Insert** se zobrazí automatické generování bloků kódu.



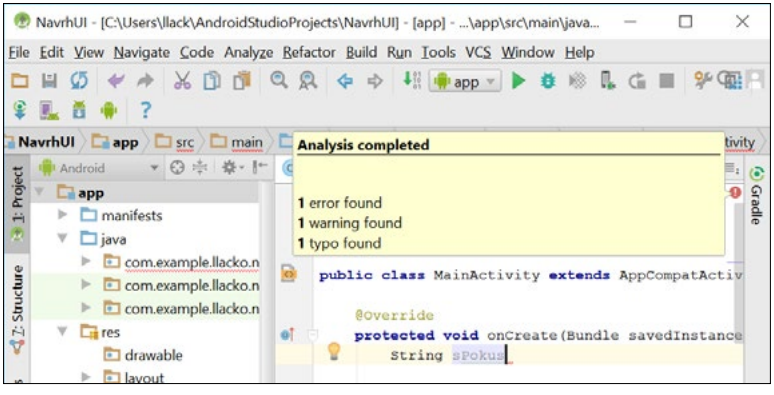
Obrázek 1.44: Dialogové okno s automatickým generováním bloků kódu

Takovýmto způsobem můžete například jednoduše generovat těla kódu metod `Override`.



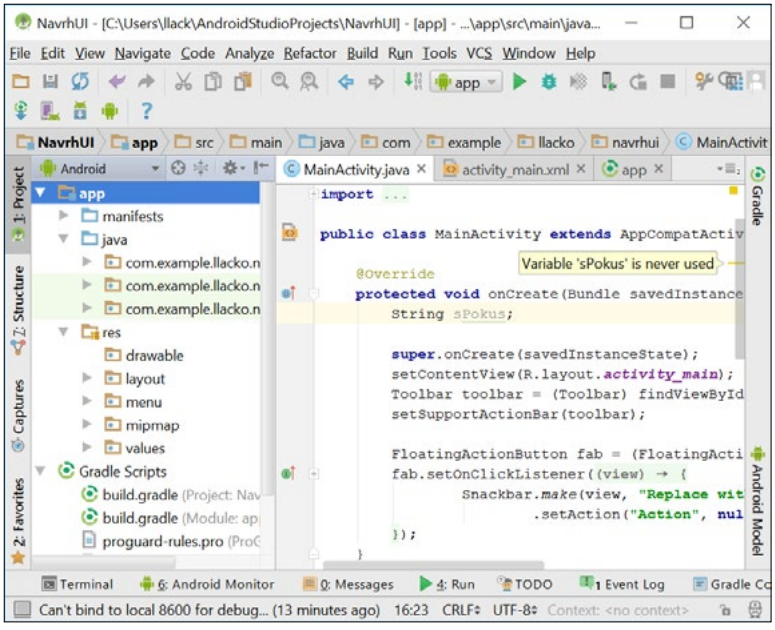
Obrázek 1.45: Nabídka metod Override

Vývojové prostředí neustále kontroluje syntaktickou správnost vašeho kódu. Pokud narazí na problém (v našem případě), chybějící středník za příkazovým řádkem, zobrazí se na liště vpravo červená ikonka s vykřičníkem. Klepnutím na ikonu otevřete nápovědu k problému.



Obrázek 1.46: Výsledek průběžné analýzy kódu

Chyby a upozornění, například na nepoužitou proměnnou, se signalizují na liště vpravo i červenými a žlutými čárkami, které jsou rozmístěny proporcionálně s řádky kódu, na nichž se tyto chyby nacházejí. Klepnutím na čárku se zobrazí podrobnosti.



Obrázek 1.47: Upozornění, v tomto případě na nepoužitou proměnnou

Java pro migranty

Nejjednodušší situaci, co se týká přechodu na programovací jazyk Java, budou mít programátoři v C#. Při zběžném pohledu na segment kódu jsou tyto jazyky téměř k nerozeznání.

Programovací jazyk Java má striktní kontrolu typů a rozsahů polí. Nezná pole znaků, text uzavřený do uvozovek se automaticky konvertuje na objekt typu `String`. Proměnné primitivních typů se automaticky inicializují na `0`, respektive příslušný ekvivalent, a reference objektů se automaticky inicializují na hodnotu `null`.

Java podporuje vícevláknový (multithread) kód s automatickým uzamykáním a dá se využít i tzv. souběžné programování. Pokud něco nevyjde podle představ vývojáře, přichází ke slovu robustní systém obsluhy výjimek. Java neumožňuje používat globální datové struktury ani funkce. Všechny programové struktury musí být zahrnuty v třídách. Namísto oboru jmen (namespace) používá balíčky (packages).

Pozitivní zprávou je, že Java nezná ukazatele, známý to postrach začátečníků v programovacím jazyce C, a nedefinuje automatické standardní (default) ani kopírovací (copy) konstruktory. Namísto destruktorů využívá Java k rušení objektů mechanizmus nazývaný garbage collector, který pomocí metody `finalize()` uvolňuje používané zdroje. Objekt se automaticky zruší v případě, že už dále nebude využíván a současně existuje více takovýchto nepoužívaných objektů.

Co se týká objektově orientovaného programování, všechny třídy jsou organizovány v jednokořenové (single-root) struktuře začínající třídou `Object`. Třídy se definují, ne deklarují. Bez specifikace konkrétního přístupu pro data a metody třídy jsou data a metody automaticky přístupné všem třídám definovaným v rámci balíčku. Využívají se i tzv. vnořené (nested) třídy, které efektivně a pro programátora jednoduše spolupracují s členy nadřazených tříd. Pokud defi-

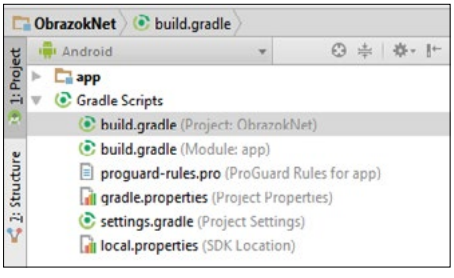
nujete přístupovou úroveň `protected`, můžete k takto definovaným členům třídy přistupovat z libovolného místa v rámci balíčku, avšak mimo něj pouze z podtříd dané třídy. Rodičovská třída je definovaná pomocí klíčového slova `extends` a metody rodičovské třídy voláme pomocí klíčového slova `super`. Toto volání se musí provést na začátku konstruktoru.

Java s výhodou používá přetěžování (override) metod. Název metody a seznam jejích parametrů se nazývá signatura. K vytvoření abstraktní třídy (rozhraní) slouží klíčové slovo `interface`. Následně k definování třídy, která implementuje abstraktní metody rozhraní, slouží klíčové slovo `implements`. K vytvoření metody, která nemůže být přetížena, slouží klíčové slovo `final`.

Gradle

Dosud jsme se nezajímali o to, jak Android Studio zkompiluje či interpretuje zdrojové a návrhové kódy, slinkuje je dohromady se zdroji, vytvoří aplikační balíček, zavede ho do emulátoru nebo reálného zařízení a spustí výslednou aplikaci. Už z předchozí věty je zřejmé, že se jedná o velké množství na sebe navazujících úloh, přičemž následující může (někdy nemusí) být zahájena až po úspěšném ukončení předchozí. Automatizaci úloh při sestavení, zavedení a spuštění aplikace v Android Studiu má na starosti moderní nástroj na automatizaci Gradle, integrovaný přímo v Android Studiu.

Gradle je nástroj na automatizaci procesů, přesněji jazyk na automatizaci. Je to jazyk typu Domain Specific Language (DSL), který umožňuje popsat posloupnost úloh, které chceme zautomatizovat. Je založen na Apache Groovy. Samotný Gradle toho moc nedokáže, jeho skutečná síla spočívá v pluginech. Na sestavování projektů aplikací pro Android se využívá Android plugin for Gradle.



Obrázek 1.48: Struktura zobrazení složky Gradle v Android Studiu

Android Studio umožňuje vytvářet současně více podprojektů, ze kterých se skládá komplexní aplikace. Každý takovýto podprojekt má vlastní *build.gradle* skript. Za názvem *build.gradle* je ještě v závorce uvedeno, že je to build soubor pro celý projekt ve tvaru **(Project: <project name>)**. Skript na nejvyšší úrovni obsahuje globální konfiguraci pro všechny podprojekty (moduly).

Příklad skriptu:

```
// Top-level build file where you can add configuration options
// common to all sub-projects/modules.

buildscript {
  repositories {
    jcenter()
  }
  dependencies {
```

```
    classpath 'com.android.tools.build:gradle:2.1.2'

    // NOTE: Do not place your application dependencies here; they belong
    // in the individual module build.gradle files
  }
}

allprojects {
  repositories {
    jcenter()
  }
}

task clean(type: Delete) {
  delete rootProject.buildDir
}
```

Soubor *settings.gradle* deklaruje jednotlivé podprojekty. Názvy modulů odpovídají názvům adresářů, ve kterých se nacházejí. V jednoduché aplikaci je v tomto souboru jeden řádek:

```
include ':app'
```

Tento soubor, označený **build.gradle (Module: app)**, obsahuje skript:

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 24
  buildToolsVersion "23.0.3"

  defaultConfig {
    applicationId "sk.pcrevue.llacko.obrazoknet"
    minSdkVersion 23
    targetSdkVersion 24
    versionCode 1
    versionName "1.0"
  }
  buildTypes {
    release {
      minifyEnabled false
      proguardFiles getDefaultProguardFile('proguard-android.txt'),
        'proguard-rules.pro'
    }
  }
}

dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  testCompile 'junit:junit:4.12'
  compile 'com.android.support:appcompat-v7:24.0.0'
}
```

V prvním řádku skriptu je deklarovaný plugin `com.android.application`. V sekci `android` jsou údaje o verzích SDK a verzích nástrojů. Následuje sekce `defaultConfig`, která obsahuje ID aplikace, minimální a cílovou verzi SDK.

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 24
  buildToolsVersion "23.0.3"
```

Sekce `buildTypes` obsahuje instrukce, jak spustit ProGuard se souborem APK. A to nejdůležitější nakonec: Sekce `dependencies` definuje vzájemné závislosti modulů, které se budou kompilovat a sestavovat.

Graddle můžete spustit i z příkazového řádku Windows. Příkazem `CD "cesta"`, například

```
CD C:\BT android\ObrazokNet
```

přejdete do adresáře, ve kterém máte projekt vytvořený v Android Studiu a zadáte příkaz

```
> gradlew.bat assembleRelease
```

V okně konzole můžete sledovat průběh sestavení. Při prvním spuštění se budou navíc stahovat a rozbalovat potřebné moduly.

```
C:\BT android\ObrazokNet>gradlew.bat assembleRelease
Incremental java compilation is an incubating feature.
:app:preBuild UP-TO-DATE
:app:preReleaseBuild UP-TO-DATE
:app:checkReleaseManifest
:app:preDebugBuild UP-TO-DATE
:app:prepareComAndroidSupportAnimatedVectorDrawable2400Library UP-TO-DATE
:app:prepareComAndroidSupportAppcompatV72400Library UP-TO-DATE
:app:prepareComAndroidSupportSupportV42400Library UP-TO-DATE
:app:prepareComAndroidSupportSupportVectorDrawable2400Library UP-TO-DATE
:app:prepareReleaseDependencies
:app:compileReleaseAidl UP-TO-DATE
:app:compileReleaseRenderscript UP-TO-DATE
:app:generateReleaseBuildConfig UP-TO-DATE
:app:mergeReleaseShaders UP-TO-DATE
:app:compileReleaseShaders UP-TO-DATE
:app:generateReleaseAssets UP-TO-DATE
:app:mergeReleaseAssets UP-TO-DATE
:app:generateReleaseResValues UP-TO-DATE
:app:generateReleaseResources UP-TO-DATE
:app:mergeReleaseResources UP-TO-DATE
:app:processReleaseManifest UP-TO-DATE
:app:processReleaseResources UP-TO-DATE
:app:generateReleaseSources UP-TO-DATE
:app:incrementalReleaseJavaCompilationSafeguard UP-TO-DATE
:app:compileReleaseJavaWithJavac UP-TO-DATE
:app:compileReleaseNdk UP-TO-DATE
:app:compileReleaseSources UP-TO-DATE
```

```
:app:lintVitalRelease
:app:prePackageMarkerForRelease
:app:transformClassesWithDexForRelease UP-TO-DATE
:app:mergeReleaseJniLibFolders UP-TO-DATE
:app:transformNative_libsWithMergeJniLibsForRelease UP-TO-DATE
:app:processReleaseJavaRes UP-TO-DATE
:app:transformResourcesWithMergeJavaResForRelease UP-TO-DATE
:app:packageRelease UP-TO-DATE
:app:assembleRelease

BUILD SUCCESSFUL
Total time: 14.428 secs
C:\BT android\ObrazokNet>
```

Možnosti současného hardwaru

Hodně uživatelů ani netuší, jaký obrovský výpočetní a často i grafický výkon nosí v kapsách. Zkuste se přenést o 20 let zpět, kdy se začaly do počítačů osazovat procesory Intel Pentium. Byly taktované na 60 MHz, ty nejdražší dosahovaly tehdy až neuvěřitelných 75 MHz. Nejvýkonnější stroje měly k dispozici 64 MB paměti RAM.

A nyní to porovnejte s moderním smartphonem střední třídy. Například Huawei P8 Lite – telefon v cenové hladině 5 000 Kč, tedy žádná vlajková loď ani horká novinka, v době psaní knihy byl už rok na trhu – má osmijádrový procesor s taktem 1,2 GHz, 2 GB paměti RAM a 16 GB úložného prostoru. Dokázali jste si v éře prvních Pentií vůbec představit počítač, natož pak levný mobilní telefon, který bude mít dvacetkrát výkonnější procesor (reálně mnohem víc, protože první Pentium mělo jen jedno jádro a procesory mobilů mají 4 až 8) a třicetkrát více paměti? Stejně neslavně dopadne špičkový počítač nedávné minulosti v porovnání s inteligentními hodinkami. Například hodinky Samsung Gear S2 mají dvoujádrový procesor taktovaný na 1,2 GHz, 512 MB RAM a úložný prostor s kapacitou 4 GB.

Proč tyto věci zmiňujeme? Velký výkon mobilních zařízení je totiž zároveň i velkou výzvou pro vývojáře aplikací, aby tento výkon dokázali využít a poskytnout uživateli své aplikace co nejvyšší přidanou hodnotu. Představte si, že byste před 20 roky, ve zlaté éře PC, kdy pro ně nebyl dostatek kvalitních aplikací a her, položili IT expertovi otázku: Co by dokázalo zařízení, které by bylo stokrát výkonnější a mělo stonásobně více operační paměti a úložného prostoru a navíc by se vešlo do kapsy a minimálně den by vydrželo fungovat na baterii? Odpovědi, či spíše předpovědi, by určitě předstihly možnosti 99,99 % současných mobilních aplikací. Je to tedy obrovská výzva pro vývojáře.

Referenční zařízení

Možná jste už slyšeli o zařízeních s označením Nexus. Produktová rodina Nexus má mezi zařízeními s Androidem výsadní postavení referenčního etalonu. Navrhuje je Google ve spolupráci s některými renomovanými dodavateli telefonů, kteří je následně vyrábí. Nové modely se tradičně představují na konferenci Google I/O spolu s novým operačním systémem. V září 2015 byl představen tandem Nexus 5X od LG a větší Nexus 6P, který vyrábělo Huawei.

Jejich pozice etalonu má svá specifika a zaběhlá pravidla, která se týkají i designu. Dobrou analogií jsou etalony pro základní fyzikální veličiny. Jsou velmi kvalitní, ale bez jakýchkoliv designo-

vých okras. A stejné jsou i Nexusy. Strohost jejich designu má i jiný důvod. Google s nimi nechce konkurovat modelům ostatních výrobců. U Nexusů je zároveň pravidlem, že disponují nejmodernějšími technologiemi, aby na nich vývojáři mohli testovat aplikace, které budou tyto technologie využívat. Nové modely Nexusů mají například porty USB typu C a snímače otisků prstů.

Abychom byli konkrétní, Nexus 5X, který vyrábí LG, má šestijádrový procesor disponující čtyřmi jádry Cortex-A53 pro běžný úsporný provoz a dvěma jádry Cortex-A57 pro zvládnutí špičkové zátěže. K dispozici má 2 GB RAM. Jako grafický čip je použit Adreno 418. Fotoaparát umožňuje natáčet video s rozlišením 4 K.

Jednu z nejatraktivnějších vlastností Nexusů jsme si nechali na konec. Po ohlášení nové preview verze Androidu jsou to vždy právě Nexusy, pro které je tato verze k dispozici. Takže jejich majitelé se mohou těšit, že budou mít nový Android s kódovým označením „N“, případně později s označením „O“, jako první. Řadu telefonů Nexus jako referenčních zařízení pro vývoj a testování aplikací Google ukončil a nahradil novou produktovou řadou špičkových zařízení Google Pixel.

Technické údaje Nexusu 5X:

Procesor a grafika:	Šestijádrový Qualcomm Snapdragon 808, 1,8 GHz, GPU Adreno 418
Paměť:	2 GB RAM, úložný prostor 16 GB nebo 32 GB
Displej:	5,2“ IPS s rozlišením 1 920 × 1 080, 423 ppi
Fotoaparát:	Zadní: 12,3 MP, Přední: 5 MP, oba se světelností f/2.0
Konektivita:	LTE, Wi-Fi 802.11ac, USB-C, Bluetooth 4.2, NFC
Baterie:	2700 mAh, nevyměnitelná
Operační systém:	Android 6.0 Marshmallow
Rozměry a hmotnost:	147 × 72,6 × 7,9 mm, 136 g

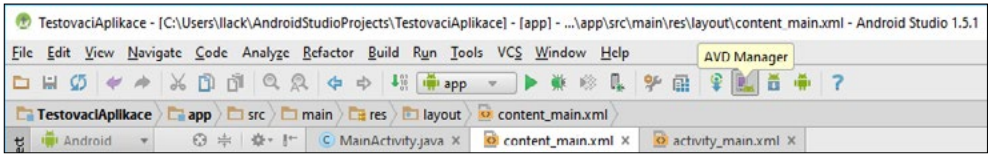


Obrázek 1.49: Referenční telefon NEXUS 5X

Vytvoření emulátoru

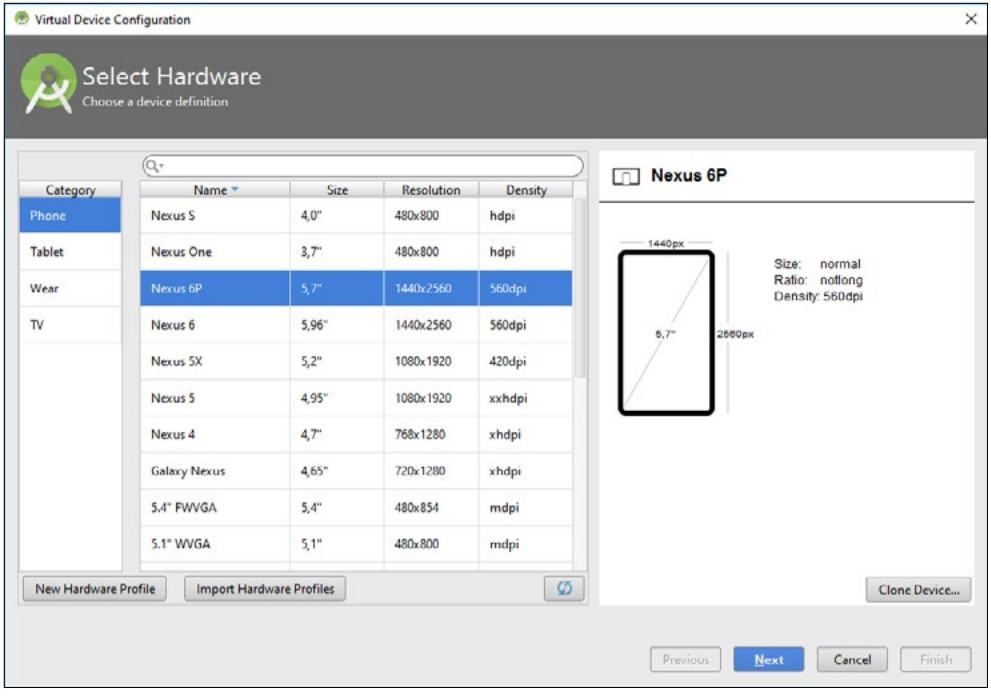
Zdálo by se, že pokud máte k dispozici jeden moderní chytrý telefon a jeden tablet s Androidem, dokážete pokrýt návrh, testování a ladění aplikací. Opak je pravdou. Výhodou a zároveň nevýhodou Androidu je variabilita různých zařízení s různými verzemi systému a různým rozlišením displeje. Neodmyslitelnou pomůckou vývojáře je proto emulátor, na kterém je možné otestovat aplikaci ve více verzích operačního systému Android a na obrazovkách s různým rozlišením.

V aplikaci Android Studio na nástrojové liště vyberte ikonu **AVD Manager**. Zkratka AVD znamená Android Virtual Devices. Zobrazí se dialogové okno aplikace **Android Virtual Device Manager** se seznamem virtuálních zařízení.



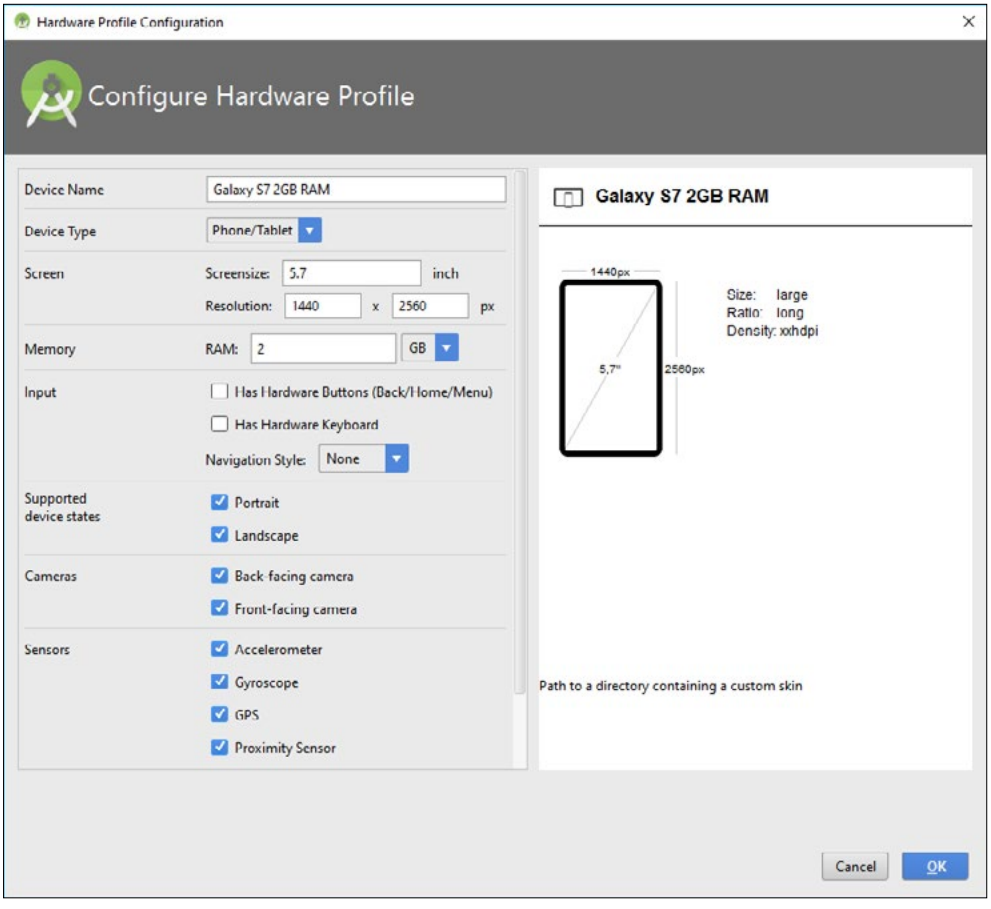
Obrázek 1.50: Ikona AVD Managera v aplikaci Android Studio

Tlačítkem **Create Virtual Device** vytvořte nový emulátor požadovaného zařízení. V dialogovém okně na vytvoření emulátoru jsou předdefinovaná zařízení rozdělena do kategorií. K dispozici jsou kategorie **Phone**, **Tablet**, **Wear** a **TV**. Buď si vyberete některé z předdefinovaných zařízení a tehdy doporučujeme vhodný Nexus s čistým Androidem, nebo pokud máte speciální požadavky, například potřebujete více či méně paměti RAM, vytvořte emulátor požadovaného zařízení klonováním.



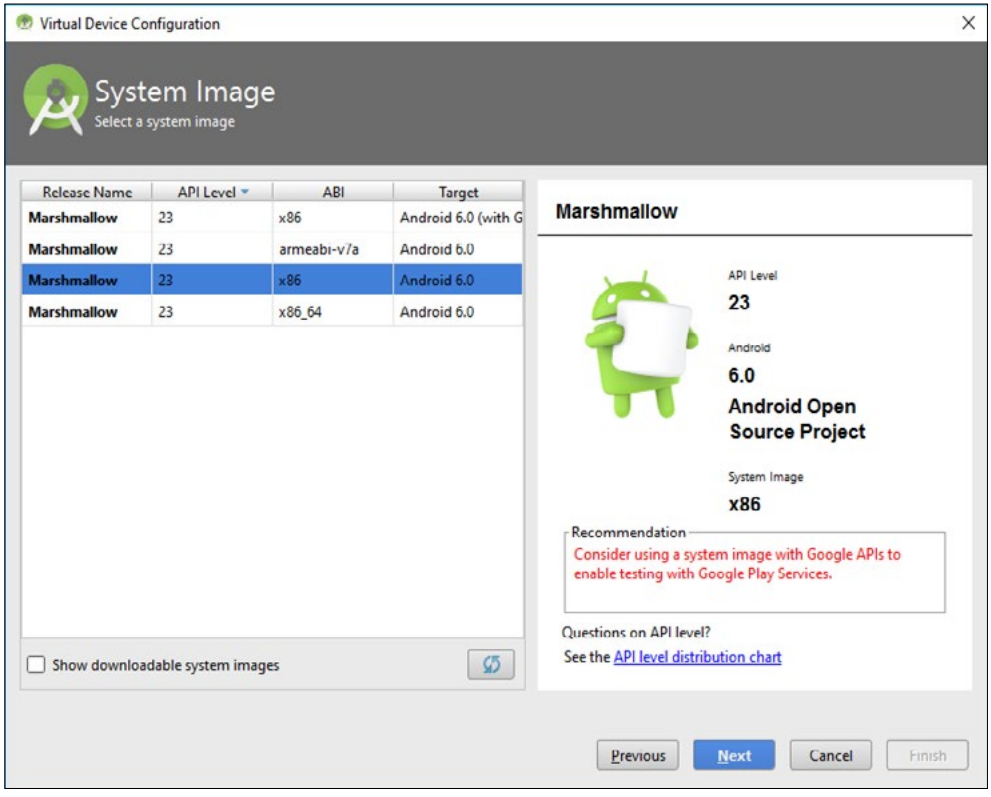
Obrázek 1.51: Vytvoření emulátoru

Například potřebujete emulátor Samgungu Galaxy S7 Edge se 4 GB RAM a displejem QHD s tím, že vám budou stačit 2 GB, abyste příliš neukrojili z paměti vývojářského počítače. Vyberte nejvhodnější podobné zařízení, například Nexus 6P, a klepněte na tlačítko **Clone Device**. Upravte velikost paměti.



Obrázek 1.52: Vytvoření emulátoru klonováním

Následuje výběr verze operačního systému. AVD Manager vám nabídne verzi, která nejrychleji poběží na vašem vývojářském počítači. V našem případě jsme ponechali implicitně vybranou volbu x86, přestože originální zařízení má procesor Exynos nebo Qualcom Snapdragon s architekturou ARM.

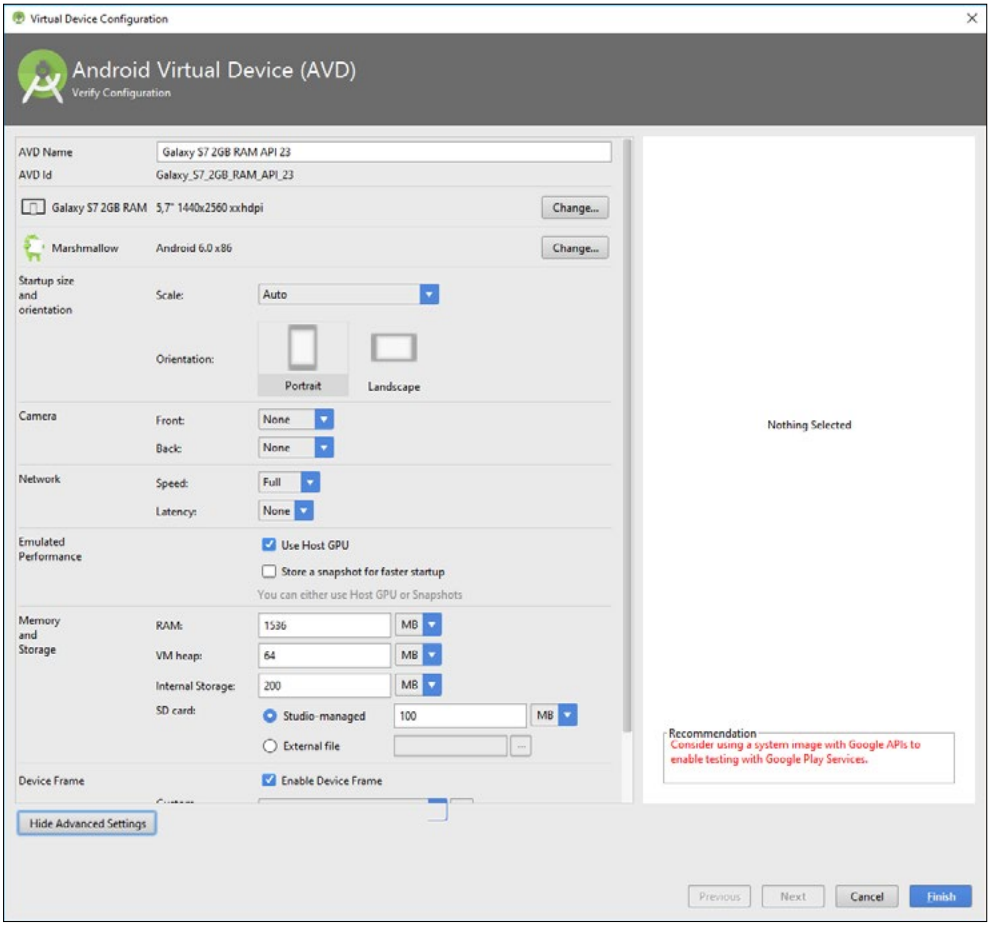


Obrázek 1.53: Vytvoření emulátoru – výběr verze operačního systému

Po potvrzení zadaných parametrů se emulátor přidá do seznamu. Pro účely této publikace doporučujeme vytvořit dva emulátory, oba pro verzi Android 6.0 Marshmallow (API Level 23) nebo 5.0 Lollipop: jeden emulátor telefonu s úhlopříčkou 5 palců a rozlišením 768 × 1280, druhý emulátor tabletu s úhlopříčkou 7 až 10 palců a rozlišením full HD 1 920 × 1 080 pixelů nebo vyšším.

Pokud označíte možnost **Store a snapshot for faster startup**, druhé a další spuštění emulátoru proběhne velmi rychle, protože AVD po zavření uloží svůj aktuální stav. První spuštění emulátoru trvá trochu déle, u dalších spuštění je už doba náběhu přiměřená. Při vytváření emulátoru nezapomeňte nakonfigurovat dostatečnou kapacitu paměti RAM, úložného prostoru a případně i SD karty, pokud ji bude aplikace využívat.

Z důvodu kompatibility vyberte nejnižší předpokládanou verzi systému. Takovéto aplikace budou na vyšších verzích fungovat bez problémů, opačně to ale neplatí.



Obrázek 1.54: Vytvoření emulátoru – nastavení parametrů

Při používání emulátoru může nastat problém s jeho zobrazením. Základní mód většiny telefonů a tabletů s Androidem je totiž „na výšku“, naproti tomu monitory vývojářských počítačů jsou orientované „na šířku“. Jak zobrazit tablet s rozlišením 1 920 × 1 080, případně vyšším, na monitoru s vertikálním rozlišením 768 pixelů? Nemusíte se bát, s implicitním nastavením parametru **Startup size and orientation** se emulátor zobrazí v takovém měřítku, aby maximálně využil plochu vaší obrazovky.

GitHub

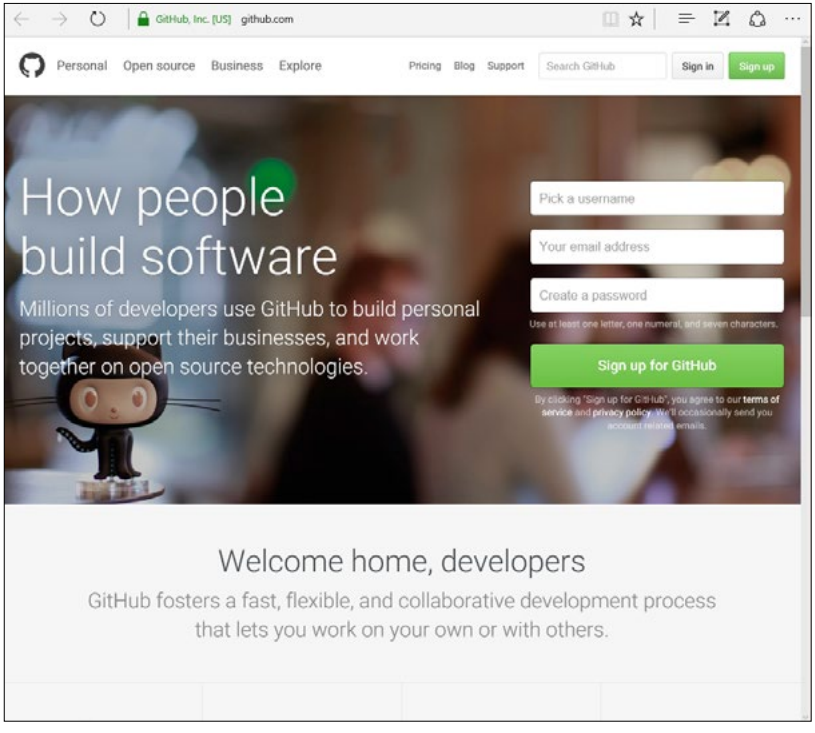
Správa verzí zdrojového kódu aplikací je důležitá především u týmového vývoje, ale díky řadě výhod ji využívají i samostatní vývojáři. Předchází tak mnohým problémům souvisejícím se ztrátou zdrojových a jiných souborů, případně jejich přepsáním. I samostatní vývojáři často pracují na projektu z více míst a počítačů, například v práci, doma, případně u zákazníka. Potřebují pracovat kontinuálně a úpravy rychle realizované u zákazníka se do produkční větve aplikace zapracují až po jejich důkladném vyzkoušení na testovací verzi.

Dobrý systém na správu verzí umožňuje přehledně oddělit aktuálně distribuovanou funkční verzi aplikace od upravovaných verzí a navíc vytvářet pokusné verze, které v konečném důsledku mohou, ale nemusí znamenat na jedné straně slepou větev a na druhé straně revoluční upgrade.

Git je systém na správu verzí, který byl původně navržen pro správu verzí open-source operačního systému Linux. V současnosti je nejrozšířenějším systémem na správu verzí zdrojového kódu aplikací a je populární nejen u hobby vývojářů. GitHub je online služba, která vám umožní vaše data v Git ukládat na online server, abyste je měli vždy přístupná. Navíc umožňuje kompletní správu vašich repozitářů přes webové rozhraní. Služba je k dispozici v základní verzi zdarma, to v případě, že vám nevadí, že vaše kódy budou veřejně přístupné a kdokoliv si je může vyhledat, prohlédnout a případně stáhnout jako celek nebo zkopírovat bloky kódu. Pokud chcete mít repozitář, který není veřejně přístupný, musíte za tuto možnost zaplatit.

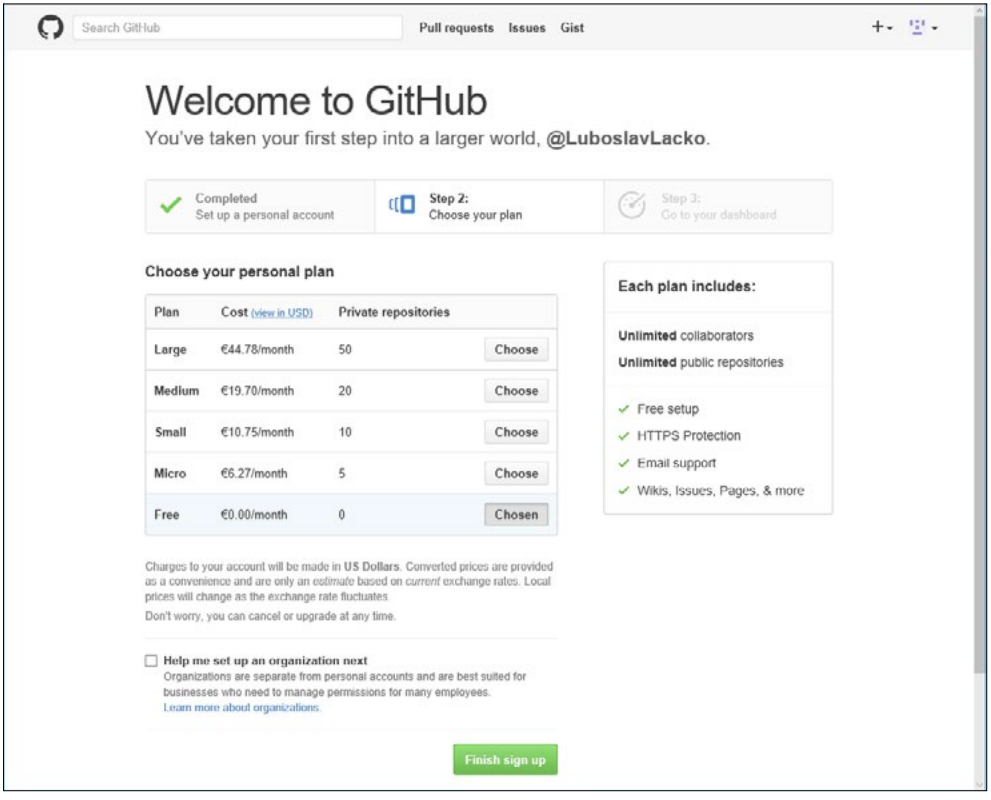
Vytvoření účtu

Vstupním bodem služby je stránka *www.github.com*. Na vytvoření účtu stačí zadat přihlašovací jméno, e-mail a heslo.



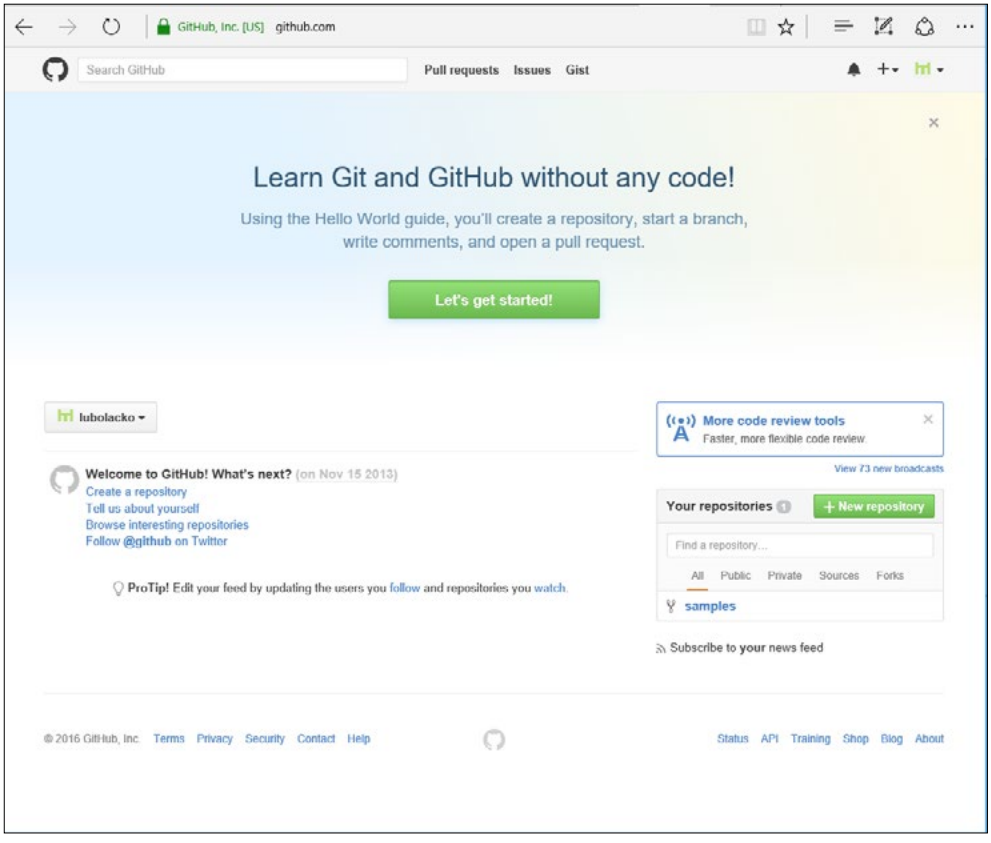
Obrázek 1.55: Vytvoření účtu ve službě github.com

Jak už bylo zmíněno, základní varianta s veřejnými repozitáři je zdarma. Cena placených variant se odvíjí od počtu privátních repozitářů. Všimněte si, že pro každý plán poskytování služby, včetně bezplatného, můžete vytvářet neomezený počet repozitářů a není omezen ani počet participujících na jednotlivých projektech.



Obrázek 1.56: Cenové varianty služby

Plné vytvoření účtu vyžaduje potvrzení odkazu v ověřovacím e-mailu. V opačném případě nebude váš účet plně funkční a webový portál služby vás bude vyzývat k ověření.



Obrázek 1.57: Úvodní stránka zobrazená po vytvoření služby

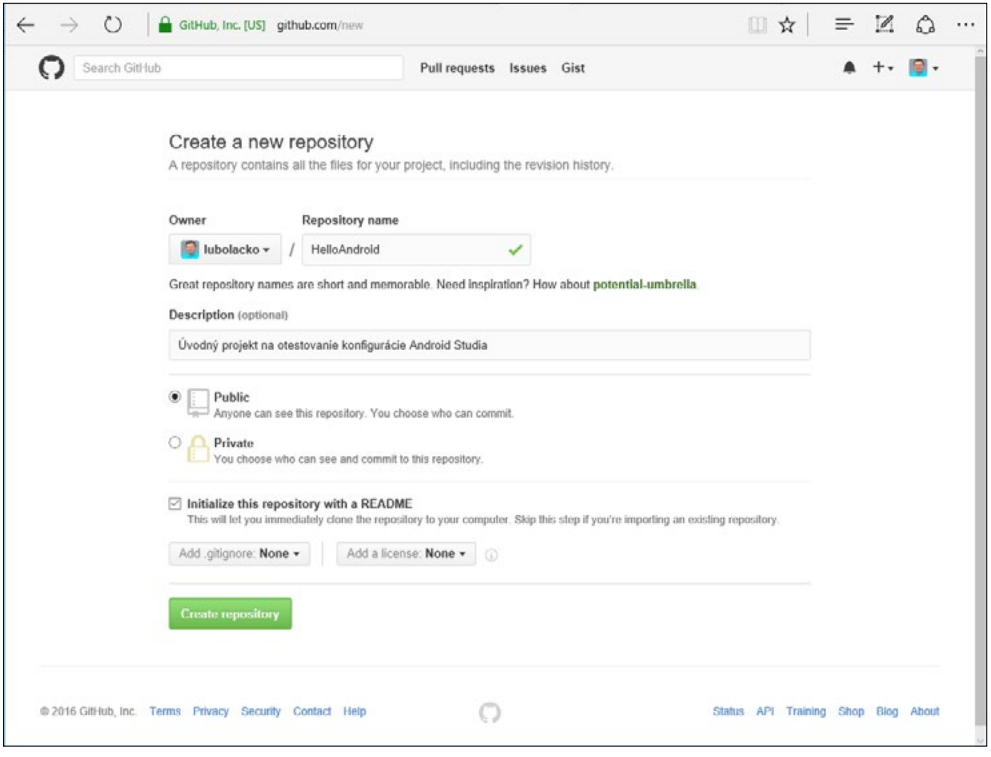
Doporučujeme v profilu aktualizovat fotku vašeho avatara, případně zveřejnit jméno či e-mailovou adresu. Pro komunitu vývojářů je přece důležitá především vzájemná spolupráce a sdílení poznatků.

Vytvoření repozitáře

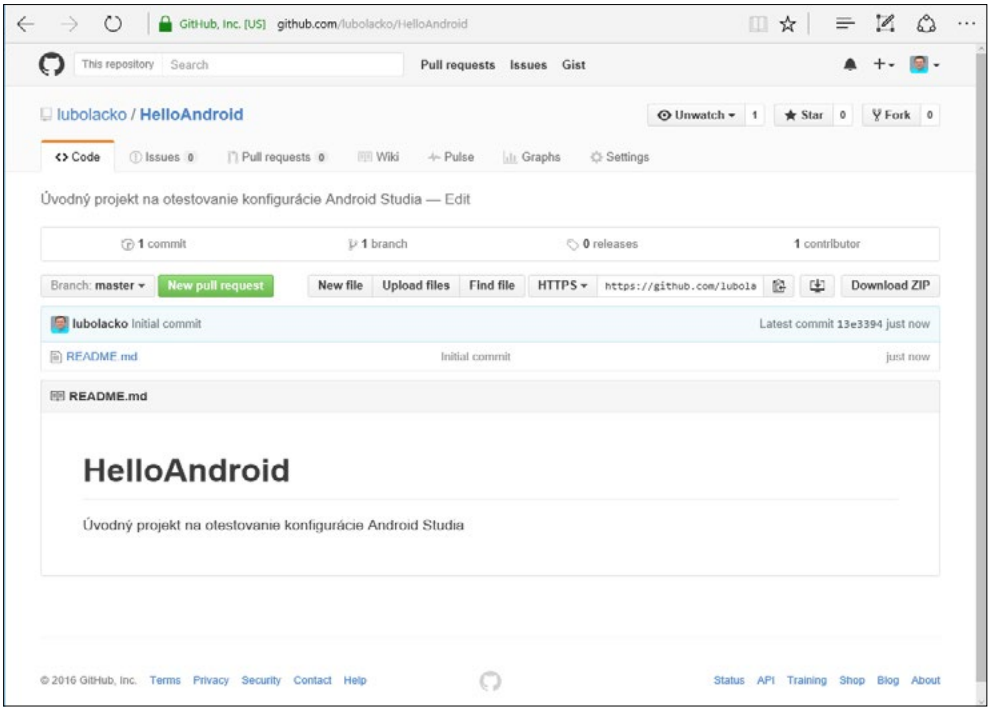
Pokud jste server GitHub dosud nevyužívali, doporučujeme absolvovat krátký, přibližně 10minutový tutoriál, který nabízí hlavní stránka projektu po prvním přihlášení. Nejprve je potřeba vytvořit nový repozitář. Repozitář si nejjednodušeji představíte jako složku, ve které máte umístěný jeden projekt, to znamená všechny soubory, které ho tvoří, včetně různých verzí.

Repozitář vytvoříte klepnutím na ikonu se symbolem„+“ v pravé horní části obrazovky vedle svého avatara. Potvrďte volbu **New repository** a do pole **Repository name** zadejte název vašeho repozitáře. Doporučujeme dodržet radu pod zadávacím polem, abyste vybírali krátké a výstižné názvy. Také doporučujeme do pole **Description** napsat krátký a výstižný jednořádkový popis repozitáře. Pokud používáte free verzi služby, může být váš repozitář jen typu **Public**, tj. veřejně dostupný. Volba **Private** se zpřístupní jen pro placené účty.

Všimněte si volby **Initialize this repository with a README**. Pokud ji zaškrtnete, vytvoří se v novém repozitáři soubor *README.md*. Při seznamování se s GitHubem aktivací této volby de facto zařídíte, že repozitář bude obsahovat aspoň jeden soubor a bude s ním možné pracovat. Zároveň se pro takovýto nyní už plnohodnotný repozitář, i když jen s jedním souborem, vytvoří klon na vašem počítači. Pokud volbu neoznačíte, musíte si první soubory nahrát do repozitáře sami. Tlačítkem **Create repository** vytvoříte nový repozitář.



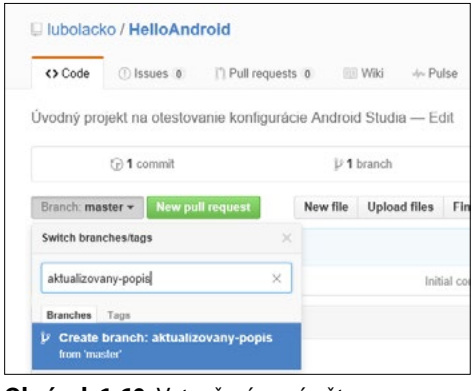
Obrázek 1.58: Vytvoření repozitáře



Obrázek 1.59: Nově vytvořený repozitář. Zatím obsahuje jen soubor README.md

Vytvoření větve (branch)

Každá linie vašeho projektu bude tvořit jednu paralelní větev. Například jedna větev bude aktuální verze aplikace pro Android, která je už publikovaná v aplikačním obchodě, a další větev bude verze, na které právě pracujete, vylepšujete ji, a když ji důkladně otestujete, může se stát finální verzí. Tehdy ji zpravidla sloučíte se svou hlavní (master) větví. Větev Master branch obsahuje produkční verzi projektu, která je v aplikačním obchodě nebo nasazená v ostrém provozu. Tato větev se ve většině případů stává základem na vytváření nových, paralelních větví.



Obrázek 1.60: Vytvoření nové větve

Postup vytvoření nové větve v repozitáři je jednoduchý. Klepněte na prvek s označením **Branch: master**. Zobrazí se okno k zadání nové větve. Větev vhodně a výstižně pojmenujte.

Po tomto úkonu máte v repozitáři dvě větve, v našem případě „master" a „aktualizovany-popis". Zatím jsou stejné. V reálném projektu můžete provádět změny v nové větvi, aniž by se tyto změny promítly do větve master.

Úprava souborů

Git spravovaným souborům přiděluje tři základní stavy:

- **Zapsané (committed)** – data bezpečně uložená ve vaší lokální databázi
- **Změněné (modified)** – v souboru byly provedeny změny, avšak soubor ještě nebyl zapsán do databáze
- **Připravené k zapsání (staged)** – změněný soubor jste v jeho aktuální verzi určili k tomu, aby byl zapsán v další revizi (tzv. Commit)

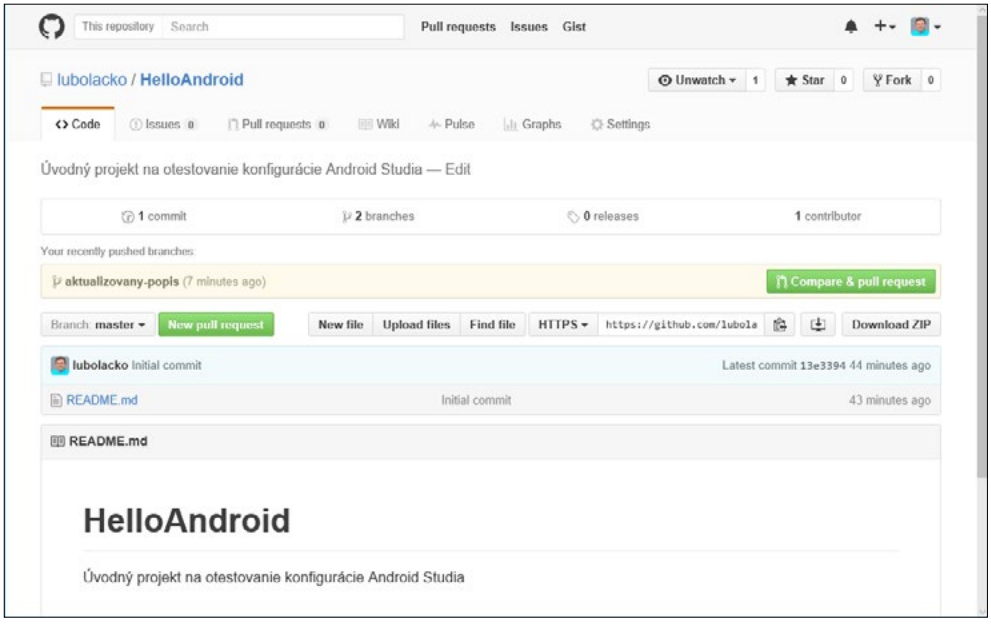
Změny si vyzkoušejte na editování textového souboru *README.md*. Klepněte na název souboru a následně na ikonu tužky v pravém horním rohu. Otevře se okno, ve kterém můžete editovat obsah souboru. Změny potvrdíte tlačítkem **Commit changes**. Operace **Commit** umožňuje uložení provedených úprav, během kterého se vytvoří skupiny změn v příslušných souborech. Během operace **Commit** je potřeba napsat komentář s popisem provedených změn. Později vám tento komentář může značně pomoct, napište proto do něj všechno potřebné o provedených úpravách.

Na synchronizaci zdrojových kódů jsou určeny operace:

- **Push** – odeslání všech naposled provedených operací commit na server.
- **Pull** – aktualizace vašich lokálních souborů ze serveru GitHub. Aktualizuje se jen větev, pro kterou tuto operaci požadujete.

Klepněte na tlačítko **New pull request**. Vyberte větev, v našem případě „aktualizovany-popis", a porovnejte ji s hlavní větví. Pokud souhlasíte se změnami, klepněte na tlačítko **Create pull request**.

Velmi zajímavou funkcí je **Fork** (rozvětvení). Použijete ji tehdy, pokud do svého účtu potřebujete přidat repozitář jiného vývojáře, nejčastěji proto, abyste jeho kód modifikovali a vytvořili na jeho bázi vlastní projekt. Funguje to jednoduše: Otevřete cizí repozitář a klepněte na tlačítko **Fork**. Na funkci Fork se můžete podívat ze dvou úhlů pohledu. Buď vy přeberete repozitář jiného vývojáře, nebo jiný vývojář převzal váš repozitář. Zda a kolikrát se tak stalo, zjistíte podle číselného údaje vedle tlačítka **Fork**. Klepnutím na tlačítko **Network** zjistíte, kdo vytvořil původní repozitář a jaké úpravy se v něm následně prováděly.



Obrázek 1.61: Aktualizovaný popis

Pokud chcete uploadovat soubory z vývojářského počítače na server GitHub, musíte si nainstalovat aplikaci, kterou stáhnete z *https://desktop.github.com*. Nainstalují se dvě aplikace: *GitHub* a konzolová aplikace *GitShell*.

GitHub je de facto sociální síť, která úplně změnila způsob, jakým pracují komunity vývojářů. V současnosti je to největší online úložný prostor ke spolupráci na tvorbě softwarových a jiných projektů, kde se pracuje s textem.

KAPITOLA

2

Anatomie Androidu

Témata kapitoly:

- Multiplatformní operační systém
- Historie verzí
- Starší verze
- Android 5.0 Lollipop
- Android 6.0 Marshmallow
- Android 7.0 Nougat
- Stručně o architektuře Androidu

Android je platforma open-source na bázi Linuxu určená hlavně pro mobilní zařízení, tedy chytré telefony, tablety a stále více se prosazuje i v chytrých hodinkách, televizorech, autech a podobně.

Multiplatformní operační systém

Systém Android vyvíjí organizace Open Handset Alliance, jejíž součástí jsou desítky firem včetně těch nejznámějších v mobilní branži – Google, HTC, Intel, NVIDIA, Qualcomm, Samsung atd. Jde o jeden z mála operačních systémů podporujících více platforem, můžete ho vidět v zařízeních nejrůznějších značek. To však přináší jednu značnou nevýhodu – chybí optimalizace systému na konkrétní platformu, což je silná zbraň Apple iOS. Android je však multiplatformní s možností přizpůsobení a vytvoření nadstaveb (Samsung TouchWiz, HTC Sense a mnohé další). Na druhé straně když Google vydá aktualizaci systému, ta není hned dostupná pro všechna zařízení a uživatel si musí počkat na konkrétní aktualizaci od svého výrobce.

Největší výhodou a zároveň nevýhodou platformy je její otevřenost a možnost úprav, ať už ze strany výrobců, nebo uživatelů. Úpravy se netýkají jen konfigurace či widgetů, ale i firmwaru. Pro Android je k dispozici nejvíce aplikací, mnohé jsou však pochybné kvality, jelikož proces jejich schvalování není tak přísný jako u iOS či Windows. Tablety a telefony s Androidem dodává hodně firem. Je to záruka dynamičtějšího vývoje nových zařízení, například v porovnání s Apple, kde je celý vývoj hardwaru v režii jedné firmy, a zároveň představuje problém, protože aplikace běží na na přístrojích s různým rozlišením displeje a různým výkonem procesoru a grafiky. V praxi to znamená různý komfort uživatelského ovládání. Na rozdíl od ostatních platforem nevydává aktualizace operačního systému centrálně Google, ale výrobci zařízení, takže se u různých zařízení můžete setkat s různými verzemi.

Historie verzí

Android nevyvinul ani nenavrhl Google, ale odkoupil ho. Společnost Android Inc. vznikla v říjnu roku 2003 v kalifornském Palo Alto. O dva roky později, v srpnu 2005 Google odkoupil Android Inc. za 50 milionů dolarů. Dnes má hodnotu vyšší o tři řády. V listopadu 2007 byla založena Open Handset Alliance, která stojí za vývojem Androidu dodnes. V té době vyšel zároveň vývojářský kit (SDK). V září 2008 přišel v USA na trh HTC Dream (G1), první chytrý telefon s Androidem 1.0, který měl na trhu chytrých telefonů podíl zanedbatelného půl procenta. Únor 2009 přinesl Android 1.1 jako aktualizaci pro G1. V dubnu 2009 spatřil světlo světa první masový Android ve verzi 1.5 (Cupcake).

Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, JellyBean, KitKat, Lollipop, Marshmallow, Nougat... Pokud by se v IQ testu objevila takto uspořádaná slova a úkol doplnit následovníka, je zřejmé, že to bude zákusek nebo jiná sladkost začínající písmenem O. Přesně tento systém používá Google ke značení verzí Androidu – názvy zákusků s počátečními písmeny postupně podle abecedy. Že jste nikdy neslyšeli o verzích začínajících písmeny A a B? Tyto verze sice nebyly použité na reálných telefonech, ale existovaly, nejmenovaly se však podle zákusků, nýbrž Alpha a Beta.



Obrázek 2.1: Loga verzí Androidu

Starší verze

Nejprve představíme historické verze:

Android 1.5 Cupcake (*malý muffin v košíčku se šlehačkou* – 30. 4. 2009) – podpora virtuálních klávesnic třetích stran s podporou vlastních slovníků, nahrávání a přehrávání videa ve formátech MPEG-4 a 3GP, možnost kopírovat a vložit obsah přes schránku, podpora videa YouTube a obrázků Picasa, umísťování widgetů na domácí obrazovku, animace přechodů mezi obrazovkami... V době nástupu verze byly v Android Marketu 3 000 aplikací.

Android 1.6 Donut (*koblih s otvorem* – 1. 9. 2009) – integrované vyhledávání Google, aplikační market vylepšený o obrázky a hodnocení uživatelů, práce s více soubory, univerzální vyhledávač, vylepšené hlasové vyhledávání, podpora displejů s vyšším rozlišením (WVGA), bezplatná navigace založená na aplikacích od Googlu, podpora VPN...

Android 2.0/2.1 Eclair (*podlouhlý zákusek s náplní a čokoládou navrchu* – 26. 10. 2009) – nový design uživatelského prostředí, optimalizace výkonu, podpora standardů HTML5, Bluetooth 2.1, podpora Microsoft Exchange a Google Maps 3.x, Live Wallpapers, podpora velkého množství rozlišení displejů. V době nástupu verze bylo v Android Marketu 20 000 aplikací.

Android 2.2 Froyo (*šlehačková špička obložená ovocem* – 20. 5. 2010) – nové uživatelské prostředí a webový prohlížeč s optimalizací JavaScriptu, podpora Flash 10, možnost instalovat aplikace i na paměťovou kartu, vylepšené zálohování, modem USB, galerie 3D a sdílení kontaktů přes Bluetooth, možnost vytvořit hotspot Wi-Fi, významná optimalizace používání paměti a celkového výkonu, možnost automatických aktualizací aplikací z Android Marketu... V době nástupu verze bylo v Android Marketu 100 000 aplikací.

Android 2.3 Gingerbread (*perníček* – 6. 12. 2010) – vylepšená správa napájení, podpora více fotoaparátů, komunikace přes NFC (Near Field Communication), podpora dalších typů senzorů (gyroskop, barometr...), vylepšené stahování velkých souborů, podpora internetových hovorů (VoIP), vylepšená virtuální klávesnice.

Android 3.0 Honeycomb (*medová plástev* – 22. 2. 2011) – první verze určená jen pro tablety, vylepšené uživatelské rozhraní, action bar, zjednodušené notifikace, přizpůsobitelná domovská obrazovka, hardwarová akcelerace, podpora příslušenství USB.

Android 4.0 Ice Cream Sandwich (*„ruská" zmrzlina* – 19. 10. 2011) – tato verze odstraňuje rozdíly mezi verzemi pro mobilní zařízení a verzemi pro tablety. Novinky jsou: aplikace na obrazovce uzamčení, nedávno spuštěné aplikace, vytváření adresářů, možnost zrušit notifikace, zastavení aplikací na pozadí, odemknutí rozpoznáním tváře, převod hlasu na text, nový internetový prohlížeč, podpora videa ve full HD, integrace sociálních sítí do kontaktů.

Android 4.2 Jelly Bean (*barevné želatinové bonbóny* – 9. 7. 2012) – Google Cloud Messaging, podpora uživatelských kont, vylepšené notifikace, vylepšená aktualizace aplikací, widgety na obrazovce uzamčení, vyhledávání Google Now, možnost přepínání uživatelských účtů, výrazné zrychlení vykreslování obrazu, rozpoznávání hlasu i offline.

Android 4.4 KitKat (*čokoládové tyčinky* – 3. 9. 2013) – vyšší výkon, vylepšená podpora zařízení s vícejádrovými procesory, rychlejší multitasking, podpora více cloudových služeb, režim fullscreen pro několik aplikací, podpora krokoměru, infračerveného ovládání, inteligentní vypínání nepotřebných procesů na pozadí.

Číslování verzí operačního systému nekoresponduje s číslováním API. Přiřazení verze API k verzi operačního systému udává tabulka:

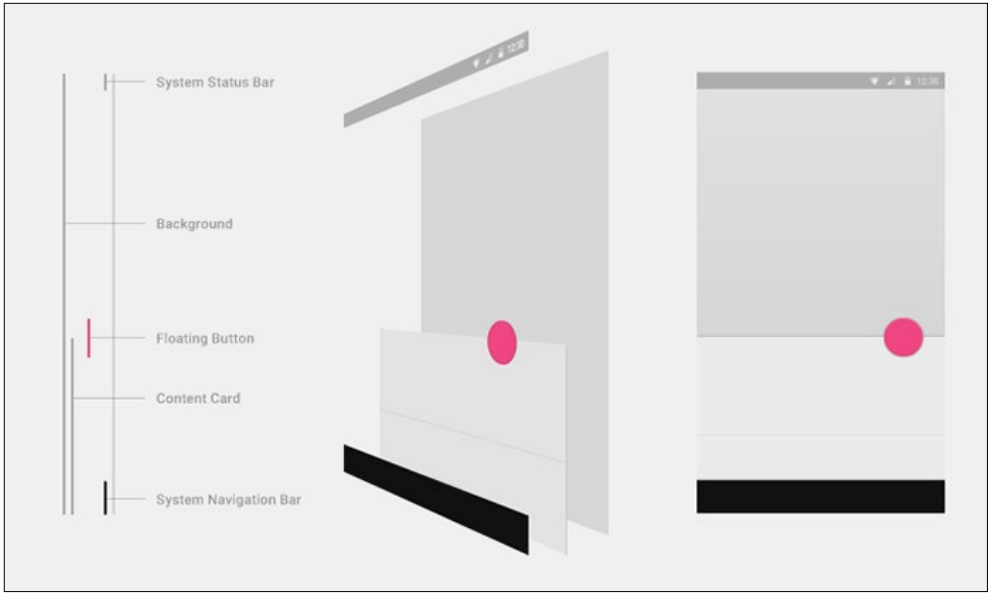
Verze OS	Verze API
Android 1.0	1
Android 1.1	2
Android 1.5 Cupcake	3
Android 1.6 Donut	4
Android 2.0 Eclair	5
Android 2.0.1 Eclair	6
Android 2.1 Eclair	7
Android 2.2–2.2.3 Froyo	8
Android 2.3–2.3.2 Gingerbread	9
Android 2.3.3–2.3.7 Gingerbread	10
Android 3.0 Honeycomb	11
Android 3.1 Honeycomb	12

Verze OS	Verze API
Android 3.2 Honeycomb	13
Android 4.0–4.0.2 Ice Cream Sandwich	14
Android 4.0.3–4.0.4 Ice Cream Sandwich	15
Android 4.1 Jelly Bean	16
Android 4.2 Jelly Bean	17
Android 4.3 Jelly Bean	18
Android 4.4 KitKat	19
Android 5.0 Lollipop	21
Android 5.1 Lollipop	22
Android 6.0 Marshmallow	23
Android 7.0 Nougat	24

Nejnovější, v současnosti nejpoužívanější verze Androidu představíme podrobněji.

Android 5.0 Lollipop

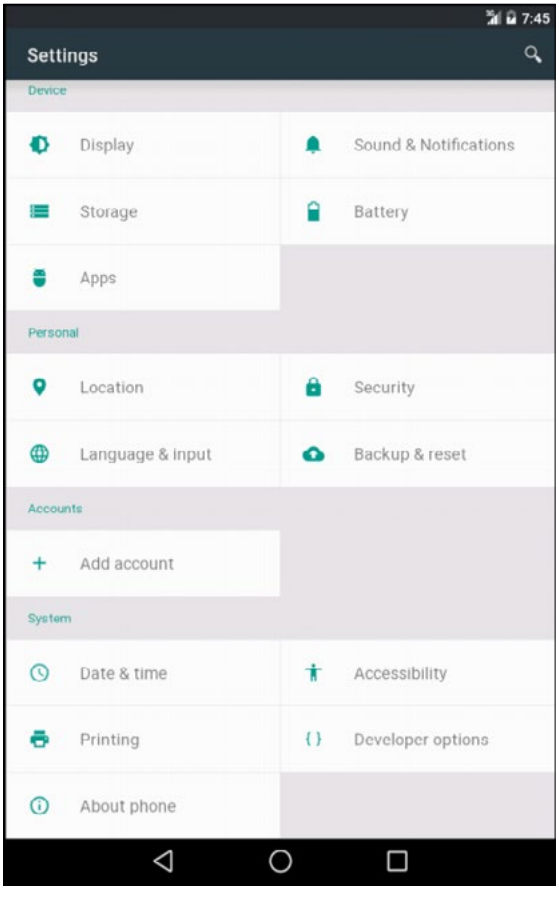
Jednou z velkých změn v nových verzích Androidu je nahrazení dosud používaného virtuálního stroje Dalvik, pod kterým běžely aplikace, novým enginem ART (Android Runtime). Hlavním přínosem enginu ART je možnost kompilovat část kódu aplikací už při jejich instalaci. To významně urychlí spouštění aplikací a sekundárně prodlouží výdrž baterie, jelikož Dalvik proces kompilace vykonával při každém spuštění.



Obrázek 2.2: Design nové verze označovaný jako Material Design je přehlednější a pomocí různých vizuálních efektů, hlavně stínu, se snaží vzbudit zdání třetího rozměru

Například design aplikace *Nastavení* je praktickou ukázkou designových doporučení pro aplikace Androidu 5.0. Z designových novinek je potřeba zmínit inovovanou notifikační lištu, která zobrazuje notifikace podle významu a důležitosti. Notifikace je možné zobrazit i na obrazovce uzamčení. Inovované je i samotné zamykání. Aktivuje se jen v neznámém prostředí, přičemž doma či v práci bude vaše zařízení odemknuté. Na rozpoznání prostředí slouží připojení zařízení k Wi-Fi nebo Bluetooth.

V předešlých verzích Androidu mohli uživatelé nastavovat jas displeje manuálně nebo automaticky (zařízení pomocí senzoru přizpůsobilo jas aktuálním světelným podmínkám). Android 5.0 přichází s novinkou pod názvem Adaptivní jas. Uživatel si manuálně nastaví jas zařízení. V případě, že se ocitne v prostředí s jiným osvětlením, zařízení automaticky upraví jas displeje na takovou hodnotu, aby byl displej stejně dobře čitelný jako při původním nastavení.

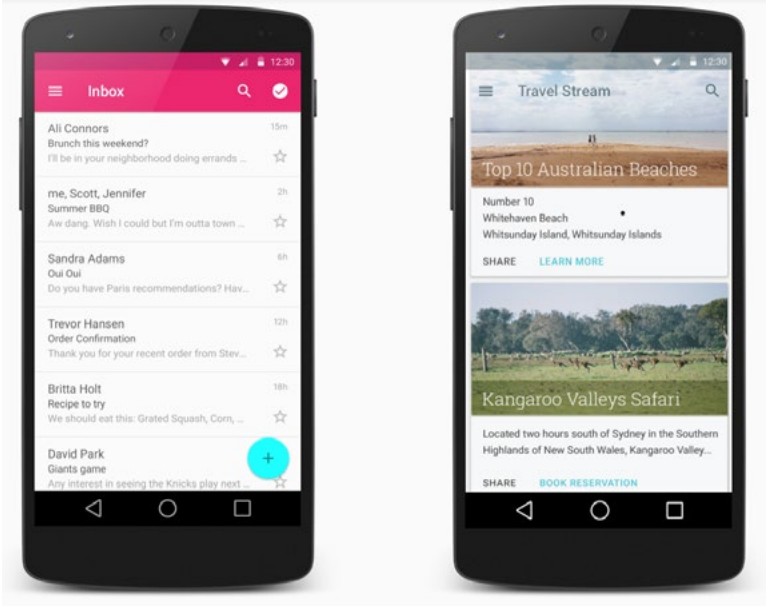


Obrázek 2.3: Design aplikace Nastavení je praktickou ukázkou designových doporučení pro aplikace pro Android 5.0

Přibyla tři nová navigační tlačítka. Jedno z nich slouží k ovládání inovovaného systému multitaskingu. Multitasking funguje na principu karet podobně jako v prohlížeči Google Chrome. Mezi více běžícími aplikacemi se přepínáte posunem ve vertikálním směru.

Přibyly i nové prvky uživatelského rozhraní. Například prvek `RecyclerView` je pokročilejší verzí prvku `ListView`, který poskytuje vyšší výkon pro dynamické pohledy, a použití prvku je z hle-

diska vývojáře jednodušší. Prvek `CardView` umožňuje komplexní zobrazení důležitých informací v rámci karet, aby se zachoval jednotný vzhled.



Obrázek 2.4: Nové prvky uživatelského rozhraní: RecyclerView vlevo a CardView vpravo

Android 6.0 Marshmallow

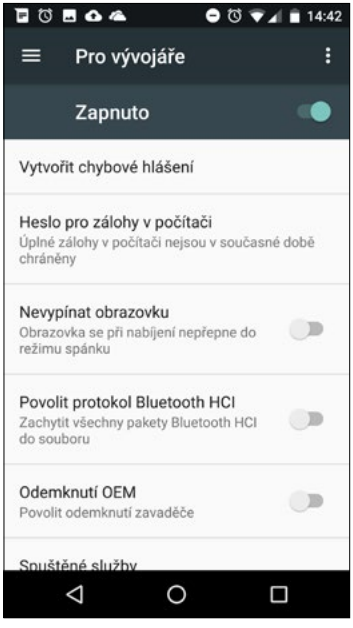
Vylepšením komfortu, funkcionality a bezpečnosti v oblasti uživatelského rozhraní k žádné velké revoluci nedošlo, změny se dají charakterizovat spíše jako designová evoluce. Revoluci v uživatelském rozhraní přinesla předchozí verze Android 5.0 Lollipop svým jednoduchým a přehledným stylem nazývaným Material Design, který se dočkal různých, velmi jemných a nevtíravých vizuálních efektů, například stín se snaží vzbudit zdání třetího rozměru, a především vrstev, které je možné v některých aplikacích posouvat tak, aby se vhodně překrývaly.

Mírně inovovaný design uživatelského rozhraní, který přináší Marshmallow, charakterizujeme jako barevně živější, dynamičtější a intuitivnější. Nejpoužívanější aplikace jsou soustředěny na hlavní obrazovce. Jejím posunem doprava se zobrazí Google asistent, který kromě vyhledávání poskytuje i další důležité informace, například v souvislosti s docházením do práce. Ostatní aplikace nejsou organizovány na vodorovně posouvatelných kartách, jak tomu bylo u starších verzí, ale ve vertikálním plynulém seznamu. První řádek ikon v nabídce aplikací (na telefonu jsou čtyři, na tabletech jich může být i více) je oddělený od ostatních vodorovnou čárou. Zobrazují se zde nejčastěji používané aplikace. Ve zbytku seznamu jsou aplikace seřazeny podle abecedy. Vpravo je posuvník s ukazatelem. Během posouvání se zobrazuje aktuální počáteční písmeno aplikací umístěných na pozici, na které se posuvník právě nachází.

Možnost poskytnout údaje, které mnozí z vás považují za soukromé, výměnou za pokročilé služby je pravděpodobně největší polemikou související s používáním mobilních zařízení. Ve starších verzích Androidu se před instalací aplikace z Google Play zobrazil seznam oprávnění,

která aplikace bude vyžadovat, a museli jste s nimi vyjádřit souhlas. Některé aplikace měly takto deklarovaných oprávnění celou řadu, takže zájemci o aplikaci seznam mechanicky odsouhlasili a aplikaci nainstalovali. S odstupem času neměli šanci si vzpomenout, která aplikace jaká oprávnění využívá.

Nová verze 6.0 vyžaduje povolení oprávnění až při prvním použití příslušné funkcionality, například při prvním použití mikrofonu či kamery. Aby v tom měl přehled i běžný uživatel, seznam oprávnění se výrazně zredukoval. Aplikace budou moct žádat o povolení osmi druhů oprávnění. Jedná se o přístup k poloze, fotoaparátu, mikrofonu, kontaktům, telefonování, posílání SMS, kalendáři a senzorům. Pokud se vám Android nebo některá z aplikací bude jevit jako příliš zvědavá, oprávnění, která by mohla narušit vaše soukromí, jednoduše nepovolíte a aplikace bude fungovat dále, ovšem s částečně omezenou funkcionalitou.



Obrázek 2.5: Povolení nebo zákaz oprávnění pro konkrétní aplikaci

Android 6.0 Marshmallow má implementovanou technologii Doze, která inteligentně a s ohledem na co nejnižší spotřebu energie z baterie spravuje procesy na pozadí a redukuje počet oznámení, pokud zjistí, že se vaše zařízení právě aktivně nepoužívá. Analýzou údajů z akcelerometru Doze snadno zjistí, že se zařízením už hodinu nikdo nepohnul, takže leží na stole a pravděpodobně se na něj nikdo nedívá, proto například sníží interval synchronizace a vypne v tomto režimu nepotřebné procesy.

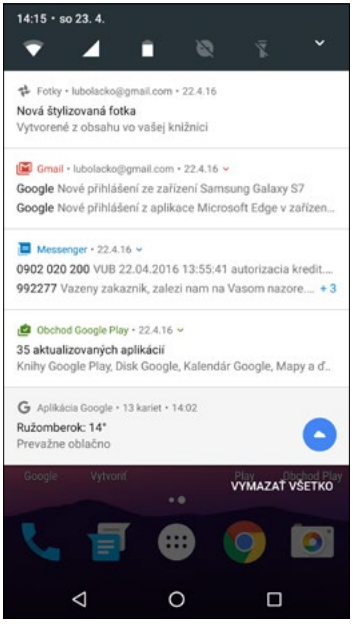
Pokud často používáte více současně spuštěných aplikací a služeb na pozadí, oceníte vylepšený engine ART (Android Runtime), který lépe hospodaří s operační pamětí. Z dalších vylepšení týkajících se hardwaru zmíníme nativní podporu snímání otisků prstů, podporu USB typu C jako nového standardu s oboustranným konektorem nebo nativně podporovaný stylus připojitelný přes Bluetooth, který snímá i změny tlaku při psaní a umožňuje přeprogramování významu tlačítek.

Vývojáři i uživatelé ocení možnost zobrazení webového kontextu přímo v aplikacích na kartách Chrome Custom Tabs s využitím všech vymožeností prohlížeče Chrome, například pamatování hesel, automatické přihlašování a podobně. Vylepšené jsou i vazby mezi aplikacemi. Pokud se například na webové stránce zobrazil odkaz na dokument PDF, v předchozích verzích Androidu jste dostali možnost vybrat si ze seznamu aplikací, které mohou příslušnou činnost, v tomto případě zobrazení specifického typu dokumentu, realizovat. Ne vždy však uživatel začátečník věděl, která aplikace je pro daný účel nejvhodnější, proto nový Android tento proces zautomatizuje a systém si sám vybere nejvhodnější aplikaci.

Android 7.0 Nougat

Největší novinkou uživatelského rozhraní, kterou uživatel velmi rychle objeví, je nová notifikační lišta. Notifikace se zobrazují v komprimovanější a zároveň přehlednější podobě. Oznamovací panel zabírá celou šířku obrazovky, takže dokáže zobrazit více informací. Ikona aplikace v oznámení je mnohem menší, neodděluje ji orámování, ale jen tenká lišta. Pokud chcete zobrazit celý obsah delšího oznámení nebo více oznámení některé aplikace, stačí na nejnovější zobrazení položit prst a posunout ho dolů. Oznámení můžete po přečtení vymazat nebo na něj reagovat, například odpovědět na zprávu v Messengeru nebo jiné aplikaci. Samozřejmě jen v aplikaci, která bude tuto vlastnost podporovat.

Google s velkým předstihem zveřejňuje alfa a následně beta verzi nového operačního systému nejprve pro vývojáře, aby při uvedení finální verze byl k dispozici dostatek aplikací, které budou používat nové funkce. V horní části oznamovacího panelu je pět ikon zástupců zobrazujících informace nebo nastavení. Můžete například zobrazit stav baterie a graf průběhu vybíjení, nastavit režim „Nerušit“, zapnout LED diodu blesku a podobně.



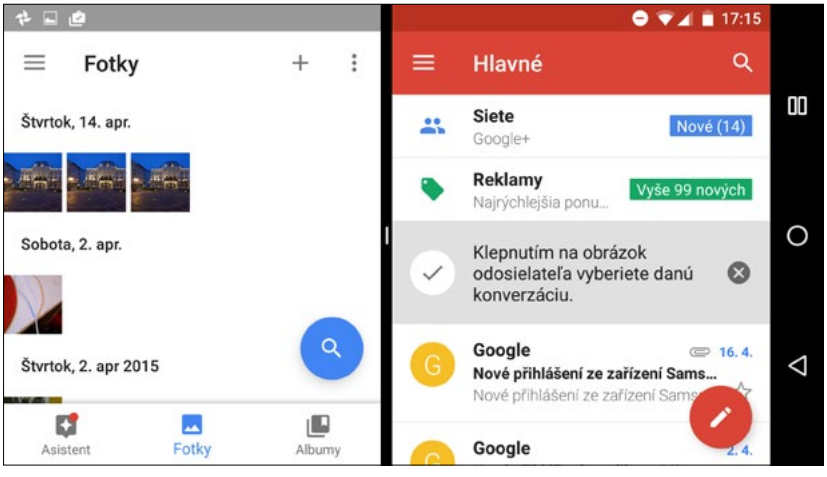
Obrázek 2.6: Nový design oznamovacího panelu

Vpravo je ikona zobrazení nabídky operativního nastavení. Horizontálním přejetím zobrazíte více ikon. Ikony nastavení můžete podle potřeby posouvat a prvních pět se zobrazí nad oznamovacím panelem. Pokud oznámení jemně posunete doleva, zobrazí se ikonka nastavení s ozubeným kolečkem. Pomocí ní můžete nastavit, zda chcete zobrazovat upozornění s nebo bez zvukového signálu nebo je chcete blokovat.

Režim více oken se nejjednodušeji aktivuje zobrazením seznamu aktuálně spuštěných aplikací jedním z trojice tlačítek pod displejem (s ikonou čtverce). Aplikaci, kterou chcete přesunout do druhého okna, uchopíte za lištu a posunete nahoru k hornímu okraji obrazovky, dokud neuvidíte, že se obrazovka rozdělila na dvě plochy, jednu jasnější a druhou tmavší. Potom aplikaci přesunete na jasněji podsvícenou plochu.

V tmavší části obrazovky zůstane zobrazený rotující seznam spuštěných aplikací a z něho vyberete, která aplikace bude běžet v druhém okně. Režim Multi-Window je indikován změnou tvaru ikony na zobrazení seznamu aplikací. Z čtverce se změní na dva obdélníky nad sebou. Tento režim nejvíce oceníte na zařízeních s větším displejem a při orientaci „na šířku". Takto můžete kontrolovat ve dvou různých aplikacích zobrazených v oknech vedle sebe například zprávy elektronické pošty a statusy na sociální síti. V novější verzi nebo na větším displeji bude podporován i variabilní dělicí poměr. Režim ukončíte tak, že dělicí lištu mezi aplikacemi přesunete na jednu nebo druhou stranu podle toho, která z dvojice aplikací má zůstat na displeji.

Ohledně této funkcionality Google apeluje na vývojáře, aby aplikace uměla zjistit, kdy je spuštěná v menším prostoru okna, a zobrazila informace v koncentrovanější podobě nebo aby použila takzvaný kontextový zoom, tedy čím větší plochu má aplikace k dispozici, tím více detailů o vybraném objektu zobrazí. Kromě režimu více oken by mě být k dispozici i režim obraz v obraze. Při pohledu na tuto funkci nelze než konstatovat, že Google podporu více oken konečně implementoval do jádra operačního systému. Výrobci tabletů ji už dávno implementovali do svých uživatelských nadstaveb.

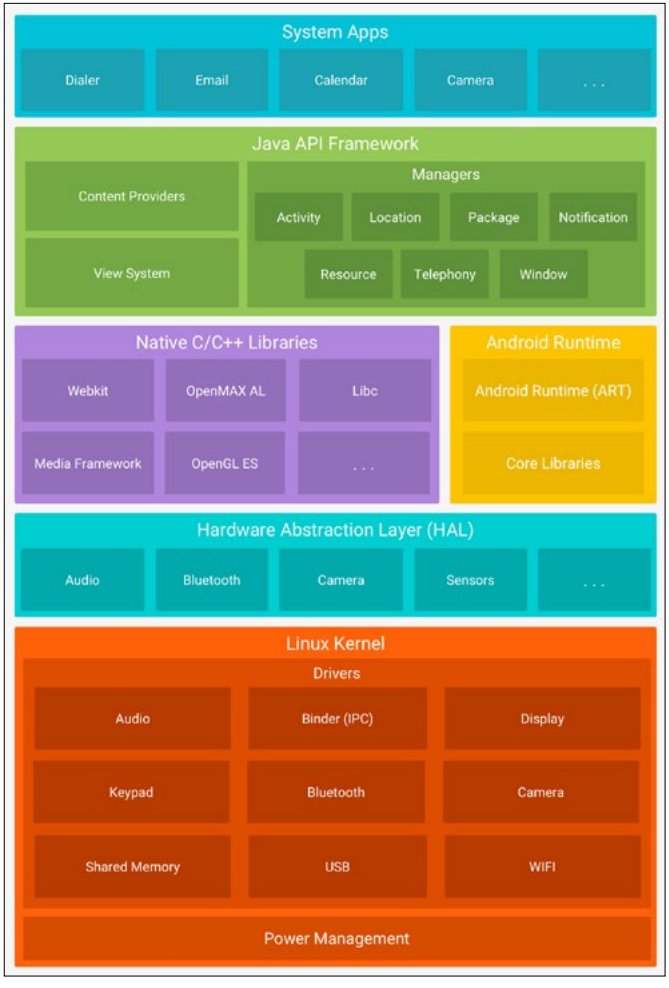


Obrázek 2.7: Režim Multi-Window

V dosavadních verzích Androidu je sice možné měnit velikost písma, aby mohly texty pohodlně číst i lidé se slabším zrakem, ale až Android 7.0 umožňuje plnohodnotně v pěti krocích měnit velikost zobrazení. Mění se nejen velikost písma, ale také ikon a dalších prvků uživatelského rozhraní. Problém s výdrží baterie by měl řešit vylepšený mód „Doze", který inteligentně omezí fungování aplikací, činností a přenosů dat na pozadí tehdy, když je displej vypnutý. Z dalších novinek zmiňme podporu pro 3D dotyk, podporu Unicode 9 a 72 nových znaků emodži a plynulejší hraní her (umožní vývojářům her využívat API Vulkan).

Stručně o architektuře Androidu

Pro vývoj aplikací potřebujete aspoň základní informace o principech a architektuře operačního systému, ve kterém budou aplikace spouštěny. Architekturu Androidu budeme popisovat podle schématu na obrázku systémem „zdola nahoru", tedy od nejnižší architektonické vrstvy.



Obrázek 2.8: Schéma architektury operačního systému Android

Linux Kernel

Základním pilířem – nejnižší vrstvou architektury Androidu – je upravené jádro populárního operačního systému Linux. Úpravy se týkají redukce funkcí a jejich přizpůsobení možnostem mobilních zařízení.

POZNÁMKA

Vývojáři běžných aplikací s jádrem do přímého kontaktu nepřijdou. Při použití ADB (Android Debug Bridge) získáte přístup k příkazovému rozhraní Linux Shell a můžete jeho prostřednictvím zadávat jádru operačního systému příkazy.

Jádro slouží k přímé interakci s hardwarem mobilního zařízení, čímž zabezpečuje úplnou abstrakci od hardwaru pro vyšší softwarové vrstvy. Zabezpečuje správu paměti, správu procesů, základní síťovou vrstvu a ovladače. Řízení procesů umožňuje, aby více procesů běželo současně, aniž by se vzájemně ovlivňovaly.

Na úrovni jádra je implementované i zabezpečení systému, správa napájení, vstupně-výstupní operace či základní grafika. Ovladač pro GSM zabezpečuje funkce telefonu. Podle hardwarové konfigurace jsou na této úrovni i moduly ovladačů pro Bluetooth, EDGE, 3G, Wi-Fi, fotoaparát, GPS, kompas, akcelerometr nebo modul rozhlasového přijímače. Modul Binder implementuje mechanizmus umožňující více procesům sdílet údaje.

Aplikace a služby jsou spouštěny v oddělených procesech a často potřebují vzájemně komunikovat. Na druhé straně je *možnost přímé komunikace mezi procesy jedním z největších bezpečnostních rizik*.

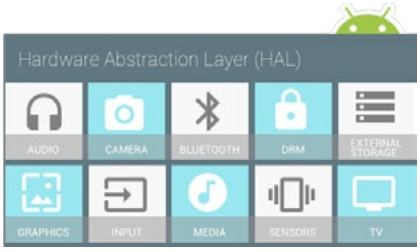
Proto je na platformě Android implementován meziprocesorový ovladač komunikace a volání metod zvaný Binder, který umožňuje více procesům sdílet údaje. Využívá sdílenou paměť, přičemž údaje se odevzdávají jako balíčky. Komunikace procesů je řízena, Binder počítá a mapuje vzájemné reference napříč systémem, takže přesouvané objekty mohou být v případě potřeby identifikovány, vysledovány a zrušeny.

Jak to funguje? Všechny aplikace se při spuštění zaregistrují ve službě Service Manager. Když potřebuje nějaký proces komunikovat s jiným procesem, osloví Service Manager. Ten poskytne procesu potřebné informace a následně se proces obrátí přímo na požadovanou službu. Binder plní při komunikaci mezi procesy úlohu zprostředkovatele.

Součástí jádra je i správa napájení. Zabezpečuje, aby se energeticky nejnáročnější moduly, tedy procesor a obrazovka, při delší nečinnosti vypínaly. Některé aplikace, například přehrávač videa či GPS navigace, potřebují běžet i při delší nečinnosti uživatele. Proto Android disponuje systémem speciálních zámků Wakelocks, které umožňují ponechat zařízení aktivní za běhu požadované aplikace. Na probuzení zařízení z režimu spánku v případě potřeby slouží objekty Alarm Timers.

Hardware Abstraction Layer (HAL)

Kvůli variabilitě hardwarových zařízení, na nichž může Android běžet, byla nad úrovní jádra operačního systému vytvořena vrstva HAL (vrstva abstrahující hardware). HAL vytváří rozhraní pro komunikaci s vyššími vrstvami systému s hardwarem. Vývojáři aplikací díky HAL nemusí přesně znát hardwarové specifikace všech zařízení.



Obrázek 2.9: Moduly vrstvy HAL

Nativní knihovny

Nad jádrem je situovaná vrstva knihoven, které poskytují přímý přístup aplikací k různým komponentám systému Android. Jsou to nativní knihovny napsané v C/C++. Tvoří mezivrstvu mezi různými komponentami vyšších vrstev a linuxovým jádrem.

Modul Surface Manager podporuje funkcionalitu multidotykového displeje. Zabezpečuje výslednou kompozici grafického výstupu více aplikací do souvislého toku dat, který směřuje do grafické vyrovnávací paměti. Z této paměti probíhá vykreslování na obrazovku.

Webkit je určený k renderování a zobrazování webových stránek. Na této úrovni jsou implementovány knihovny médií, grafické 2D a 3D knihovny, přičemž 3D knihovny využívají podporu OpenGL ES (OpenGL for Embedded Systems) s volitelnou možností využití grafických akcelerátorů. Aplikace pracující s údaji mohou využít SQLite. Knihovny v této vrstvě suplují linuxové funkce, o které bylo jádro operačního systému redukováno.

Systémová knihovna LibC je optimalizovaná pro mobilní zařízení, takže obsahuje jen části, které jsou zapotřebí pro Android.

Android Runtime – ART

Aplikace pro Android vytvořená ve vývojovém prostředí Android Studio je sestavena do bajtového mezikódu označovaného i jako formát DEX. Soubory DEX vznikly kompilací z klasických souborů CLASS a JAR. Jsou kompaktnější než klasické soubory CLASS. Když se následně aplikace načítá do zařízení, Android Runtime (ART) používá proces označovaný jako Ahead-Of-Time (AOT) na přeložení bajtového kódu do nativních instrukcí konkrétního procesoru v příslušném zařízení.

Tento formát je znám jako Executable and Linkable Format (ELF). Pokaždé když je aplikace spuštěna, spustí se přímo ELF, což má za následek vyšší výkon aplikací a delší životnost na baterie. To kontrastuje s přístupem Just-In-Time (JIT) kompilace používané ve starších verzích Androidu, kde každá aplikace pro Android představovala samostatný proces využívající vlastní instanci virtuálního stroje DVM (Dalvik Virtual Machine). Tento zabezpečoval běh spustitelných souborů s příponou DEX.

Java API Framework

Aplikační framework obsahuje v aplikacích opakovaně použitelný software, například ovládací prvky, ikony a podobně. Framework je napsán v Javě a je to nejdůležitější vrstva pro vývojáře aplikací. Poskytuje aplikacím základní služby systému.

Package Manager – modul správce balíčků je v podstatě databáze, která udržuje aktuální seznam všech aplikací nainstalovaných ve vašem zařízení. Vizuálním obrazem správce balíčků je domovská obrazovka zařízení. Každá ikona reprezentuje balíček aplikace. Proč jsme zaměřili vaši pozornost právě na tento modul? Každá z aplikací může navázat kontakt s jinými aplikacemi, například za účelem sdílení údajů, nebo jedna aplikace může požádat o služby jiné aplikace.

Window Manager – spravuje okna, která tvoří aplikace. Androidí aplikace využívají většinou dvě a více oken současně. Například pokud si spustíte navenek jednoduchou aplikaci webového prohlížeče, sestává ze dvou oken. V horní části je lišta oznámení zobrazující různé ukazatele, například sílu signálu GSM či Wi-Fi, stav zbývající energie baterie, čas a podobně. O tuto lištu se vy jako vývojář aplikace nestaráte, to je záležitost operačního systému. Webová stránka je zobrazena v hlavním okně aplikace. Aplikace mohou použít různá další okna, například na zobrazení nabídek či dialogových oken.

View System – spravuje širokou paletu společných prvků grafického uživatelského rozhraní, jako jsou ikony, tlačítka, prvky na zobrazení a editování textu a mnohé další.

Aplikační framework obsahuje i službu **Activity Manager**, která spravuje životní cyklus aplikace, **Notification Manager**, **Location Manager** a další moduly, které poskytují přístup k základním zdrojům. Tato vrstva je navržena tak, aby její komponenty byly snadno použitelné a vyměnitelné uživatelem.

Aplikace

Na piedestalu, tedy nejvyšší úrovni architektury operačního systému Android, jsou aplikace, například program na posílání zpráv, navigaci, kalendář, seznam kontaktů a podobně. Anatomie aplikací je popsána v další kapitole.

KAPITOLA

3

Anatomie aplikace

Témata kapitoly:

- Základní součásti aplikace pro Android
- Aktivita a její životní cyklus
- Příklad – Anatomie projektu
- Definice objektů ve zdrojích (resources)

Určitě očekáváte, že začneme zostra vývojem nějaké vzorové aplikace a vysvětlením principů, z čeho se aplikace skládá, jak fungují aktivity, rámce, vazby mezi uživatelským rozhraním a aplikačním kódem. Začneme ale jinak a dovolíme si říct, že začneme tím nejpodstatnějším – aby aplikace měla smysl.

Dříve než začnete vyvíjet – filozofie aplikace

Aplikace pro mobilní zařízení jsou v aplikačních obchodech k dispozici buď bezplatně, nebo zpravidla za velmi nízkou cenu. Alfou a omegou úspěchu – bez ohledu na to, zda je aplikace placená, a pokud ano, jaký model monetizace využívá – je to, aby si ji stáhlo a používalo co nejvíc uživatelů. Při takovémto nastavení konkurenčních podmínek proto záleží na dvou základních faktorech – funkcionalitě a designu. Shrneme několik základních pravidel, která byste při návrhu projektu a později při vývoji aplikace měli respektovat:

Aplikace bude dělat jednu věc, ale bude v tom bezkonkurenčně perfektní. Nejlépe to vysvětlíme na příkladu známé aplikace Shazam, která dokáže rozpoznat, jaká skladba právě hraje v okolí telefonu. De facto má tato aplikace jen jedno velké tlačítko. Po jeho stisknutí aktivuje aplikace mikrofon, chvíli poslouchá a potom vám řekne, jaká písnička ve vašem okolí hraje. Z hlediska uživatele jednoduché, přímočaré a... dokonalé. Momentálně není k dispozici žádný lepší ani rychlejší způsob, jak zjistit, co právě posloucháte. O infrastruktuře na pozadí uživatel aplikace nemá ani tušení, ale to vůbec nevadí. Po stisknutí tlačítka se mu za několik sekund zobrazí název poslouchané skladby. Ani vaše aplikace by neměla řešit více různých, méně souvisejících, či dokonce nesouvisejících věcí, ale jen jednu, zato však perfektně a dokonale.

Jednoduchá a intuitivní obsluha a žádné zadávání dlouhých textů. Limitujícím faktorem jsou především malé rozměry displeje, 5 palců je přece jen málo na čitelné zobrazení přiměřeně

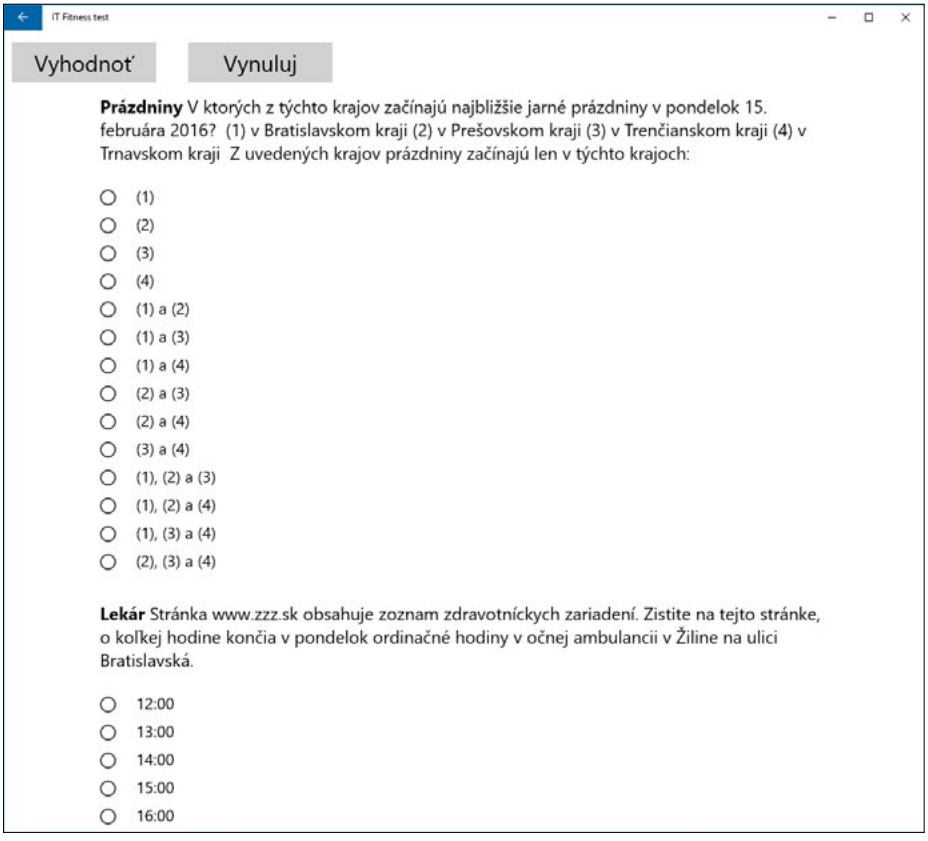
velké části dokumentu. V rozlišení displeje problém není, většina telefonů vyšší střední třídy má displej s rozlišením full HD a vlajkové lodě QHD, tedy 2 560 × 1 440, případně i vyšší. O takovémto rozlišení může řada majitelů notebooků, ale i externích displejů jen snít, oni však mají k dispozici úhlopříčku 13 a více palců a u monitorů zpravidla 22 palců a více. I přes rozlišení displeje má aplikace poměrně malou plochu a při zadávaní textu se ještě musí část plochy obětovat virtuální klávesnici.

Když to sumarizujeme, na zadávání údajů má uživatel k dispozici nakreslenou miniklávesnici, která se navíc ani nedá nahmatat prsty. Takže mobilní sociálně-navigační aplikace typu Waze by neměla naději na úspěch, pokud byste museli vypisovat texty typu: „Policejní hlídka na vjezdu do Brna ve směru od Vyškova na 42 kilometru, kontrolují technický stav a lékárničky." Je nemyslitelné, abyste vyplňovali údaje tohoto typu v klidu domova či kanceláře, natož za jízdy, kdy to zakazuje zákon.

Snad s výjimkou zamilovaných teenagerů lidé nechtějí psát na mobilu žádné delší texty, proto musí vývojáři úspěšné aplikace ťukání do virtuální klávesnice maximálně eliminovat. Pokud je třeba zadat číslo, například časový údaj, použijte kolečka s čísly, jimiž se dá otáčet a navolit tak hodnotu, případně ještě jednodušeji dvě nastavitelné ručičky analogových hodin. Pokud je třeba zadat cílovou adresu například do navigace, měla by aplikace město a ulici identifikovat z prvních několika málo zadaných znaků, případně má aplikace seznam míst podle krajů, následně seznam ulic ve městě a podobně. Úmyslně zdůrazňujeme, že takto zadáváme cílovou adresu. Adresu místa, kde se nacházíme, zadávat netřeba, aplikace musí být schopná zjistit si ji z GPS. Hodně údajů, například SPZ či stav elektroměru, netřeba psát, dá se vyfotit a následně rozpoznat, i když modul rozpoznávání písma a grafických tvarů představuje již velmi náročnou úlohu.

Na mobilu je každá aplikace hrou. Nejlépe to vysvětlíme na konkrétním odstrašujícím příkladu, sice pro jinou platformu, to ale není důležité. Kromě toho, že aplikace měla být zajímavým vědomostním testem, je ve skutečnosti ubíjejícím trápením a ilustruje i nezvládnutou spolupráci mezi vývojářem a zadavatelem aplikace. Vývojář si problém při návrhu uživatelského rozhraní uvědomil a testování aplikace to v plném rozsahu potvrdilo. Žádal zadavatele, aby změnil otázky (o tom, že nejsou časově ani na platformě nezávislé ani nemluvě), zadavatel ale neochvějně trval na svém, a tak získal, co chtěl on, ale zřejmě nikdo jiný. Test IT zručnosti by měl uživatele aplikace zábavnou formou naučit nové věci, a ne ho totálně znechutit a odradit už první otázkou.

Velmi zajímavá a užitečná je filozofie zvaná gamification. Jedná se o pronikání herních prvků do činností, které nemají s hrami nic společného. Často jsou to různé odznaky, žebříčky úspěšnosti, procentuální indikátory průběhu, body nebo virtuální peníze. S nástupem mobilních aplikací se tato filozofie dostává do popředí, možná až na první místo při návrhu koncepce aplikace a jejího ovládání.



Obrázek 3.1: Příklad aplikace, jejíž používání je díky nezvládnuté koncepci, v tomto případě struktuře testů, doslova utrpením

Pokud nepočítáme inteligentní hodinky, je chytrý telefon v současnosti nejosobnějším zařízením téměř každého člověka. A člověk si vždy rád hrál s tím, co bylo jeho naturelu blízké. Zkuste se někoho zeptat na nejoblíbenější aplikaci a po aplikacích sociálních sítí to budou hry, možná nejznámější mobilní aplikace Angry Birds. A ani aplikace sociálních sítí nejsou výjimkou, také poskytují zábavu.

A které aplikace uživatelé nemají rádi, ale musí je používat? Většina lidí uvede mobilní bankovnictví. Ani zde nemusí být aplikace nudná. Kromě různých užitečných funkcí, například skenování čárového či QR kódu z faktury, může mít některé zábavné funkce. Dobrým příkladem je aplikace americké banky PNC. Funkce Virtual Wallet zobrazí peněženku, pod kterou je porcelánové prasátko. Když klient zatřese telefonem, z peněženky se vykutálí mince a padají do prasátka. Tímto způsobem si člověk může přesunout prostředky z běžného na spořicí účet. Nepřímým důsledkem herních prvků jsou lepší ratingy v aplikačních obchodech.



Obrázek 3.2: Zábavné spoření v bankovní mobilní aplikaci

Jedním z nejfrekventovanějších problémů, které stojí za neúspěchem aplikací, je jejich složitost. Platí pravidlo, že by žádná funkcionalita neměla být dostupná na více než tři klepnutí. Pokud člověk dvě minuty bloudí ve složité nabídce, je to recept na neúspěch. Toto není pouze softwarový problém – nikdo nemá rád zbytečně složité produkty.

Marketingové aktivity aneb každá liška svůj chvost chválí. Aby se vývojářům vrátilo jejich úsilí, nesmí zapomínat na marketingové aktivity zaměřené na to, aby měla aplikace co nejlepší hodnocení. Hodnoceními se při výběru aplikace řídí asi 95 % lidí, ale hodnotí pouze 3 %. Je ve vašem zájmu upozornit – až přinutit – spokojené uživatele vaší aplikace, aby ji ohodnotili. Jednou z metod, jak toho dosáhnout, je vytvoření sociální komunity uživatelů, tedy analogie principu fungování Grouponu.

Zajímavým příkladem je aplikace SitOrSquat na vyhledávání a hodnocení veřejných toalet. Jdete na toaletu, vyfotíte její stav, přidáte své hodnocení a to uložíte do databáze. Aplikaci víc oceníte, když nutně potřebujete najít čistou toaletu ve své blízkosti. Možná ani pohled na obrázek neprozradí marketingovou pointu, takže trochu napovíme: Charmin, firma, která aplikaci vytvořila, vyrábí toaletní papír. Jednoduchá, výstižná a vtipná reklama de facto zadarmo.

Úsloví v nadpisu nemusíte brát doslova, liška kromě vlastního může pochválit zároveň i cizí chvost. Dobrým příkladem je aplikace zobrazující obchody ve vaší blízkosti, ve kterých se dá platit kartou. Je logické, že takovouto aplikaci má zájem vytvořit banka. Avšak v obchodě se dá platit jakoukoliv kartou jiné banky, takže vytvoření takovéto aplikace se na první pohled může zdát zbytečné až kontraproduktivní. Je to však vynikající marketing – budou ji používat klienti konkurenčních bank, které takovouto aplikaci nemají.



Obrázek 3.3: Reklamní podtext populární mobilní aplikace

Základní součásti aplikace pro Android

Aplikace pro Android jsou vybudované na čtyřech základních pilířích realizovaných jako třídy v Javě.

Aktivity (Activity)

Aktivita je hlavní třída, která se uživateli zobrazí po spuštění aplikace. Aplikace může sestávat z více aktivit, které si vzájemně odevzdávají údaje. Aktivity umožňují uživatelům přes grafické rozhraní (GUI) přijímat informace od aplikace a ovládat ji. Přes aktivitu se zpravidla implementuje více nebo méně komplexní částečná úloha, kterou má uživatel realizovat, například vyplnit formulář, nastavit parametry, vybrat si položku ze seznamu a podobně. Třída `Activity` je předurčena k tomu, aby zobrazovala uživatelské rozhraní a zachytávala interakce uživatele přes toto rozhraní.

Aktivita by měla být navržena tak, aby umožnila uživateli soustředit se na jednu věc, kterou momentálně potřebuje realizovat, například napsat a poslat textovou zprávu, zadat kontaktní údaje a podobně.

Na tabletech s většími obrazovkami je možné realizovat komplexnější úkony než na menších obrazovkách telefonů, případně je možné u té samé úlohy uživatele lépe navigovat či zobrazit mu přehlednější uživatelské rozhraní s obrázky. Typickým příkladem je seznam typu master-detail, který se na telefonech zpravidla zobrazuje postupně na dvou obrazovkách. Na tabletu je možné zobrazit současně seznam položek a vedle něho detailní informace o vybrané položce.

Uspořádání aktivit je hierarchické, což znamená, že po spuštění aplikace se spustí nejprve hlavní spouštěcí aktivita, z které je následně v případě potřeby možné spouštět ostatní aktivity.

Zbývající tři složky fungují „za oponou“, nemají tedy uživatelské rozhraní.

Služby (Services)

Služby realizují déle trvající operace a operace na pozadí. Také umožňují spolupráci se vzdálenými procesy. Na rozdíl od aktivit běží služby na pozadí a nepotřebují uživatelské rozhraní. Služby umožňují asynchronně (paralelně s hlavním vláknem) provádět operace, jejichž realizace trvá déle. Také umožňují požádat různé procesy o provedení operace a sdílení údajů.

Typickým příkladem služby je přehrávání hudby na pozadí. Přehrávání se inicializuje ve vhodné aplikaci na popředí. V této fázi si uživatel prostřednictvím uživatelského rozhraní aktivity vybere skladby, které chce přehrávat. Když hudba začne hrát, uživatel se může přepnout do jiné aplikace a například kontrolovat elektronickou poštu, prohlížet obrázky a podobně. Přehrávání hudby se však nepřeruší. Pokračuje prostřednictvím služby na přehrávání hudby, například `MediaPlaybackService.java`.

Broadcast receivers

Objekty na vysílání a přijímání poslouchají na pozadí a reagují na události, které se odehrávají na zařízení. Broadcast receivers fungují na principu publish/subscribe. Události jsou zastoupeny objekty typu `Intent` (záměr). Vydavatelé vytvářejí záměry a následně je přes broadcast směrují do vysílání. Zachytávají je přijímače, které mají příslušné záměry objednané nebo registrované.

Dobrým příkladem na ilustraci fungování broadcast receivers je přijetí SMS zprávy nebo e-mailu. V okamžiku, kdy zpráva přijde do telefonu, Android tuto skutečnost uživateli vhodně oznámí. Operační systém samozřejmě nemůže vědět, kdy zpráva přijde, a tak disponuje softwarovou službou, která na přijetí zprávy čeká. Jakmile se tak stane, služba vyšle záměr `SMS_received`. V telefonu je přijímač, který tento záměr přijme a spustí službu, která bude stahovat a ukládat příchozí SMS zprávy. Ve vhodném okamžiku si je uživatel prostřednictvím aplikace, kterou si pro tento účel vybere, zobrazí.

Poskytovatelé obsahu (Content providers)

Poskytovatelé obsahu umožňují ukládání a sdílení dat mezi více aplikacemi a procesy. Aplikace tak mohou přistupovat k údajům ostatních aplikací, které vystupují jako poskytovatelé obsahu. Využívá se rozhraní podobné databázovému, poskytovatelé obsahu jsou ale více než jen databáze.

Aktivitám a službám, včetně jejich životních cyklů, se budou podrobněji věnovat samostatné kapitoly.

Aktivita a její životní cyklus

Na rozdíl od aplikací pro desktopové operační systémy (Windows, Linux) není v kódu aplikace pro Android žádný exaktní vstupní bod, kterým u klasických aplikací bývá statická metoda `main()`. Její úlohou je inicializace proměnných, vizuálních i nevizuálních objektů zabezpečení běhu služeb.

POZNÁMKA

Aktivity je potřeba definovat v souboru *AndroidManifest.xml*.

Aplikace pro Android se skládají z více na sobě nezávislých komponent – aktivit a služeb. Operační systém sám určuje, kdy budou vytvořeny instance aktivit, kdy budou odsunuty na pozadí či zničeny.

Aktivita reprezentuje jednu obrazovku s uživatelským rozhraním. V kódu Javy je aktivita objekt odvozený od třídy `Activity`.

Většina aplikací – nejen pro mobilní zařízení, ale ve všeobecnosti – sestává z několika obrazovek. Na platformě Android se tyto obrazovky nazývají aktivity. Ve správně navržené aplikaci by jednotlivé aktivity měly fungovat samostatně. Měly by mít přesně definované vstupy a výstupy. Proč? Aby bylo možné z aplikace zavolat aktivitu jiné aplikace a uživateli se to jevilo jako kontinuální proces, jako kdyby aktivita jiné aplikace byla integrální součástí aktivity, z které ji spustil.

Aktivita je určitá analogie okna, případně dialogového okna v klasických aplikacích pro Windows. Základní úlohou aktivity je zobrazovat uživateli údaje z nižších vrstev v požadované formě. Každá aktivita je potomkem třídy `android.app.Activity`. Aktivita reaguje na události životního cyklu a překrývá příslušné metody.

Aktivita většinou zabírá celou plochu displeje, ale není to pravidlo. Aktivitu je v případě potřeby možné definovat jako celoobrazovkovou. Každá aktivita má vlastní životní cyklus a je potřeba počítat s tím, že instance aktivit a služeb mohou být v odůvodněných případech kdykoliv odstraněny. Například pokud začnou docházet systémové zdroje, systém se pokusí odstranit nepoužívané komponenty.

POZNÁMKA

Aktivita zodpovídá za uložení svého stavu, tedy množiny údajů, které tento stav definují.

Jedna z aktivit je definovaná jako hlavní (main). Tato aktivita se zobrazí po startu aplikace. Po spuštění další aktivity se předchozí aktivita pozastaví, ale zůstane v paměti paměťové oblasti nazývané Back Stack. Do této oblasti se aktivita ukládá po spuštění. Tehdy zároveň převezme fokus, tj. příznak, že se jedná o aktivní aktivitu zobrazenou na displeji zařízení.

Back Stack obsahuje informace o aktuálně spuštěných aktivitách. Umožňuje uživatelům jednoduše přecházet tam a zpět mezi aktivitami. Aktivity by měly být modulární v tom smyslu, že každá aktivita by měla podporovat jednu záležitost, jednu funkcionalitu. Pod pojmem Back Stack úloha se na platformě Android rozumí sada souvisejících aktivit, které mohou, ale nemusí být součástí té samé aplikace.

Back Stack je úložný prostor, který má zásobníkovou strukturu typu LIFO (Last In First Out).

Aktivita nemá veřejný konstruktor, takže její spuštění je možné ovlivnit pouze parametrem typu `Bundle` v metodě `onCreate`.

Aktivita může spouštět jinou aktivitu včetně aktivit jiné aplikace.

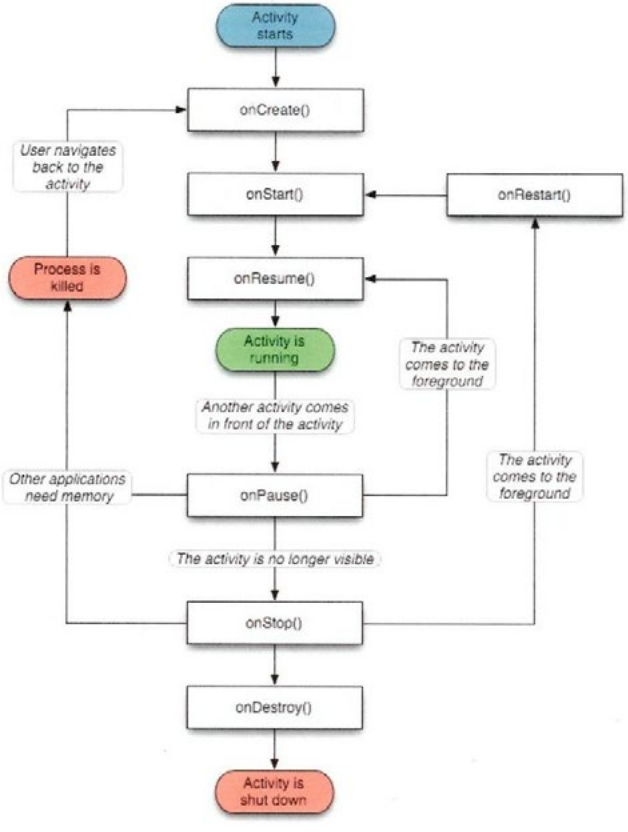
Životní cyklus aktivity

Životní cyklus aktivity je definován metodami, které se spouštějí v přesně definovaných situacích v určeném pořadí.

Životní cyklus aktivity nejlépe pochopíte z diagramu. Má tři hlavní fáze:

- **Aktivita na popředí** – zobrazuje se na displeji a má takzvaný fokus, to znamená, že interaguje s uživatelem. Aktivity běžící na popředí jsou ve stavech Running, případně Resumed.
- **Pozastavená aktivita** – je stále v paměti, je zpravidla částečně viditelná, ztrácí ale fokus, to znamená, že uživatel k ní nemá přístup. Typickým příkladem je překrytí aktivity dialogovým oknem. V případě akutního nedostatku paměti může systém aplikaci v tomto stavu odstranit.
- **Zastavená aktivita** – je kompletně překrytá jinou aktivitou. Aktivita je však stále v paměti v oblasti Back Stacku, takže není problém se k ní v případě potřeby vrátit. V případě nedostatku paměti systém nejprve odstraní tyto aktivity a až potom, pokud nedostatek paměti přetrvává, přijdou na řadu aktivity pozastavené.

Pokud uživatel potřebuje obnovit aktivitu ve stavu pozastavená nebo zastavená a používat ji na popředí, dochází k restartu aktivity.



Obrázek 3.4: Schéma životního cyklu aktivity

Pokud je potřeba realizovat v některé fázi životního cyklu některé specifické akce, je nutné přepsat jednu nebo více těchto metod navázaných na životní cyklus. Mějte na paměti, že tyto metody se vzájemně propojují a také to, že aplikace fungují v součinnosti s operačním systémem, který jejich životní cyklus často direktivně ovlivňuje, například odstraní aplikaci, která není na popředí v případě akutního nedostatku operační paměti.

Podnětem ke spuštění aktivity může být klepnutí uživatele na ikonu aplikace, případně je tato aktivita explicitním nebo implicitním cílem nějakého záměru (Intent). Následující popis průběhu životního cyklu aktivity je nejvhodnější sledovat na diagramu.

onCreate()

Metoda se poprvé aktivuje po spuštění aktivity. V těle metody se zatím na pozadí vytváří uživatelské rozhraní a konfigurují se proměnné a objekty potřebné k běhu aktivity. Po zavolání metody `onCreate()` je aktivita stále zastavená, neviditelná a nekomunikuje s uživatelem. Kód metody `onCreate()` vloží vývojové prostředí do hlavní aktivity nově vytvořeného projektu.

Metoda `onCreate(Bundle savedInstanceState){...}` se volá:

- Při prvním spuštění aktivity
- Pokud aktivita už běžela, ale byla překryta jinou aktivitou a následně se uživatel opět vrátil k původní aktivitě
- Operační systém potřeboval paměť (původní proces mohl být odstraněn)
- Aktivita běží a je vynucená změna zdrojů, například při otočení displeje, připojení nebo odpojení hardwarové klávesnice, změně jazyka uživatelského rozhraní a podobně

V metodě se obvykle řeší zavedení layoutu a inicializace prvků `view`, nastavení proměnných, případně získání dat z volajícího Intentu. Jako parametr je odevzdán objekt `Bundle` s uloženým předchozím stavem aktivity (pokud nějaký existuje).

onStart() a onResume()

Aktivita přechází do popředí. V metodě `onStart()` se realizují činnosti potřebné k tomu, aby bylo možné zobrazit uživatelské rozhraní aktivity. Rozdíl mezi metodami je zřejmý z diagramu. `onStart()` se volá při spouštěních, která následují po předchozím zastavení aktivity. Po spuštění metody `onStart()` obvykle následují spuštění metody `onResume()`. Metoda `onResume()` se volá tehdy, když aktivita přechází z pozadí do popředí.

onPause()

V případě, že je spuštění jiná aktivita, přechází aktivita, která byla spuštěná předtím na popředí, do pozadí. V této metodě je vhodné automaticky uložit změny údajů, se kterými aktivita pracovala, například do databáze, souborů dokumentů a podobně.

Metoda se volá, pokud aktivita přestane být na popředí, například při přechodu na jinou aktivitu, při zobrazení dialogového okna, po stisknutí tlačítka Home.

Po doběhnutí metody `onPause()` *může být proces zlikvidován.*

V metodě se obvykle řeší uložení aktuálních neobnovitelných dat, ukončení snímání údajů ze senzorů (GPS...), ukončení všech animací a operací náročných na CPU, uvolnění zámků typu screenlock, wakelock a podobně.

Operace realizované v metodě `onPause()` *musí být časově nenáročné.*

onStop()

Volá se při zastavení aktivity z jiného důvodu, než je nedostatek paměti. Aktivita je stále v Back Stacku a uživatel ji může znovu zobrazit na popředí. Tehdy se volá metoda `onRestart()`.

POZNÁMKA

Metoda `onStop()` nemusí být nikdy volána.

onDestroy()

Metoda se volá při ukončování životnosti aktivity, bez ohledu na to, zda se tak stane explicitně nebo implicitně.

POZNÁMKA

Metoda `onDestroy()` nemusí být nikdy volána.

Příklad – životní cyklus aktivity

Cílem příkladu je ilustrovat, kdy nastávají jaké fáze životního cyklu aplikace při používání zařízení s operačním systémem Android. Vytvořte projekt aplikace `CyklusAktivity`, přičemž jako typ hlavní aktivity vyberte **Basic**. Hlavní aktivitu vhodně pojmenujte, například `StateChangeActivity`.

Otevřete soubor *StateChangeActivity.java* s kódem aktivity. Zatím jediná metoda, která se vztahuje k životnímu cyklu, je metoda `OnCreate()`. Jejím úkolem je volat superinstanci metody před nastavením uživatelského rozhraní pro aktivitu. Budeme modifikovat tuto metodu tak, abychom získali ladicí výstup přes diagnostické hlášení panelu LogCat vývojového prostředí Android Studio při každém spuštění. Využijeme třídu `Log`, která potřebuje import `android.util.Log`.

Definujte textový řetězec, který vás při ladění aplikace upozorní na změnu stavu, abyste věděli, kde tato změna nastala. První takovýto ladicí text umístěte do metody `onCreate`. Abychom měli více stavů, je třeba přepsat některé další metody, přičemž každá z nich bude obsahovat příslušný ladicí výpis. Tyto metody můžete přidat ručně nebo vytvořit pomocí klávesové zkratky **Alt+Insert**.

```
import android.util.Log;

public class StateChangeActivity extends AppCompatActivity {
  private static final String LAD = "ZmenaStavu";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_state_change);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
    public void onClick(View view) {
    Snackbar.make(view, "Replace with your own action",
      Snackbar.LENGTH_LONG)
      .setAction("Action", null).show();
    }
  });
  Log.i(LAD, "OnCreate");
}

@Override
protected void onStart() {
  super.onStart();  Log.i(LAD, "onStart");
}

@Override
protected void onResume() {
  super.onResume(); Log.i(LAD, "onResume");
}

@Override
protected void onPause() {
  super.onPause(); Log.i(LAD, "onPause");
}

@Override
protected void onStop() {
  super.onStop(); Log.i(LAD, "onStop");
}

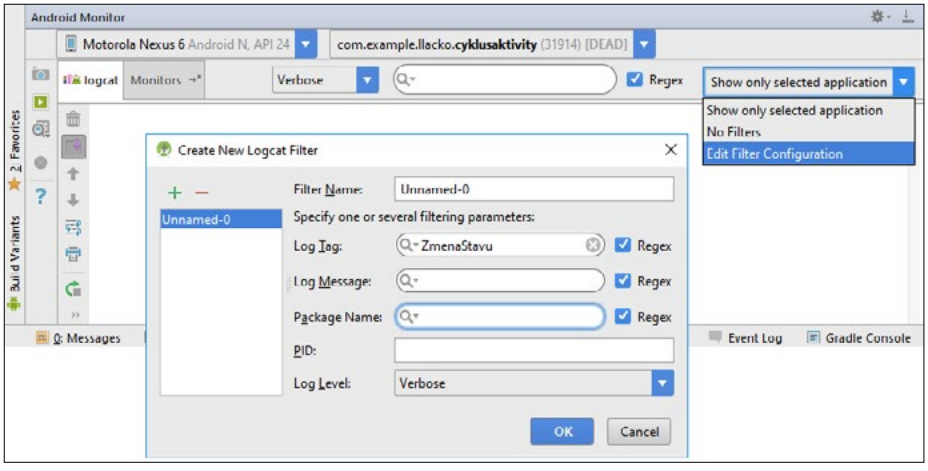
@Override
protected void onRestart() {
  super.onRestart(); Log.i(LAD, "onRestart");
}

@Override
protected void onDestroy() {
  super.onDestroy(); Log.i(LAD, "onDestroy");
}

@Override
protected void onSaveInstanceState(Bundle outState) {
  super.onSaveInstanceState(outState);
  Log.i(LAD, "onSaveInstanceState");
}

@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
  super.onRestoreInstanceState(savedInstanceState);
  Log.i(LAD, "onRestoreInstanceState");
}
```

Ladicí texty se budou vypisovat do okna nástroje Android Monitor. Tento výstup může být nakonfigurován na zobrazení všech nebo jen filtrovaných informací. Pomocí volby **Edit Filter Configuration** v rozevíracím seznamu pro výběr typu filtru definujte nový filtr pro logování.



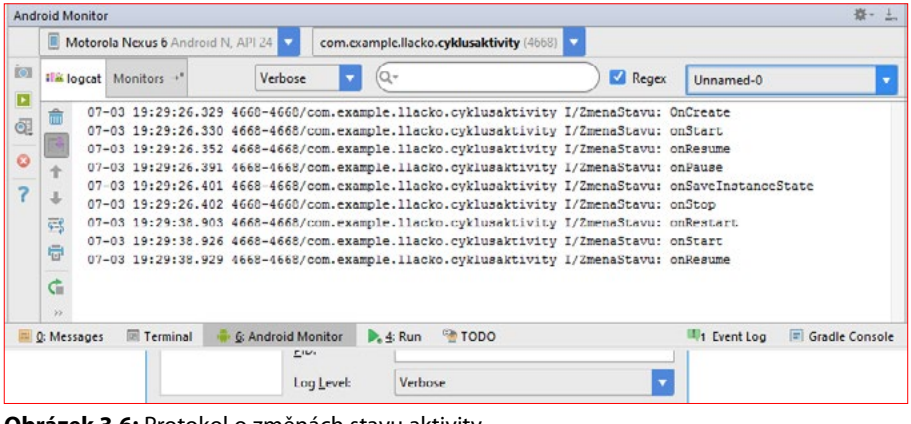
Obrázek 3.5: Definování filtru pro logovací výpisy do okna Android Monitor

Následně spusťte aplikaci, nejlépe na fyzickém zařízení připojeném k vývojářskému počítači přes USB. Po zobrazení prvních několika řádků svědčících o tom, že aplikace byla spuštěna

```
onCreate
onStart
onResume
```

zkuste chvíli se zařízením pracovat, například stiskněte tlačítko Domů. Ve výpisu protokolu se zobrazí posloupnost stavů

```
onPause
onSaveInstanceState
onStop
```



Obrázek 3.6: Protokol o změnách stavu aktivity

Příklad – ukládání a obnovení stavu aktivity

Budeme pokračovat v budování předchozí aplikace, která dokázala přes LogCat vypsat v okně Android Monitor změny stavu. Na ilustraci možností ukládání aktuálních údajů aktivity doplníme prvek `EditText` a budeme sledovat, zda a jak se při změnách stavu zachovají údaje aktuálně zadané uživatelem.

Uživatelské rozhraní aktivity je definované v souboru *content_state_change.xml*. Najdete ho v hierarchii složek projektu **app** → **res** → **layout**. Definice uživatelského rozhraní aktivity obsahuje po vytvoření projektu prvek `TextView` s předdefinovaným textem „Hello world!" Na panelu **Palette** rozviňte složku **Text Fields** a umístěte na plochu aktivity prvek `PlainText`.

V příkladu už máme přepsané metody `onSaveInstanceState()` a `onRestoreInstanceState()`.

Prvek `ExitText` je v souboru *ontent_state_change.xml* definovaný takto:

```
<EditText
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:id="@+id/editText"
 android:layout_centerVertical="true"
 android:layout_centerHorizontal="true"
 android:inputType="text"
 android:saveEnabled="false"
 android:width="200dp" />
```

Nastavte mu ještě atribut `android:saveEnabled="false"`, abyste pro ilustraci, jak to funguje, zakázali automatické ukládání stavu. Spusťte aplikaci a zadejte do prvku `EditText` nějakou hodnotu. Změnu stavu nejjednodušeji navodíte pootočením zařízení. Všimněte si, že při tomto nastavení, kdy jste zakázali automatické ukládání stavu, se vámi zadaný text při potočení zařízení vymaže.

Abychom ukázali, jak to funguje, budeme napsaný text ukládat kódem. Přidejte do zdrojového kódu v souboru *tateChangeActivity.java* import:

```
import android.widget.EditText;
```

Metodu `onSaveInstanceState` doplňte o kód, kde widgetu `EditText` přiřadíte podle ID příslušný prvek uživatelského rozhraní, získáte z něj textový řetězec a uložíte ho.

```
@Override
protected void onSaveInstanceState(Bundle outState) {
  super.onSaveInstanceState(outState);
  Log.i(LAD, "onSaveInstanceState");
  final EditText textBox =(EditText) findViewById(R.id.editText);
  CharSequence sText = textBox.getText();
  outState.putCharSequence("UlozenyText", sText);
}
```

V metodě `onRestoreInstanceState()` text obnovíte:

```
@Override
protected void onRestoreInstanceState(Bundle savedInstanceState) {
  super.onRestoreInstanceState(savedInstanceState);
  Log.i(LAD, "onRestoreInstanceState");
  final EditText textBox = (EditText) findViewById(R.id.editText);
  CharSequence sText = savedInstanceState.getCharSequence("UlozenyText");
```

```
    textBox.setText(sText);
}
```

Nyní zůstane při změně stavu text zachovaný, přesněji řečeno obnoví se.

POZNÁMKA

Znovu připomínáme, že je to jen ilustrační příklad, ve kterém jsme pro prvek `EditText` zakázali implicitní ukládání změny stavu.

Intent

Výhodou Androidu je, že vazby mezi aktivitami jsou slabé a neostré. Vstupy a výstupy aktivit jsou definované pomocí Intentů. Znalci angličtiny okamžitě pochopí, že tento pojem je odvozen od slova intention, tj. úmysl.

Intent je asynchronní zpráva, která nese informaci o požadované akci. Akcí může být spuštění aktivity, služby, případně uživatelsky definovaná akce.

Intenty umožňují vzájemnou komunikaci volně vázaných komponent aplikace pro Android. Z hlediska implementace se jedná o instanci třídy `android.content.Intent`. Intent může obsahovat údaje, které jsou uloženy v instanci třídy `Bundle`.

POZNÁMKA

Intent může spouštět i aktivity jiných aplikací.

Intent vlastně systému oznamuje, co má aktivita v úmyslu. Například uživatel chce poslat SMS, vyhledat údaje ze seznamu kontaktů, aktivovat vestavěný fotoaparát a podobně. Jak systém tento úmysl splní, je jeho záležitostí.

V praxi to funguje tak, že systém vyhledá všechny aktivity, které mohou příslušný intent splnit, a pokud jich najde více, nabídne uživateli možnost výběru. Můžete si změnit aplikaci, která vám zobrazuje seznam kontaktů či vykoná jinou činnost. Následně je vybraná aktivita spuštěna a příslušný intent je jí odevzdán jako vstup.

V souvislosti s intenty se nejčastěji používají metody:

```
intent.setClass(MainActivity.this, NewActivity.class);
```

Metoda slouží k upřesnění, jaká konkrétní třída se má po odeslání intentu spustit. První parametr `Context` udává odkaz na aktuální komponentu. Druhý parametr `Class` definuje třídu, která se má po odeslání intentu spustit.

```
intent.putExtra("pocet", pocet);
```

Metoda slouží k odevzdávání hodnot mezi komponentami. Je možné odevzdat jakýkoliv objekt `Serializable` nebo `Parcelable`. Objekty se mapují pomocí textových řetězců.

Způsob spuštění aktivity je definovaný pomocí příznaků:

- `FLAG_ACTIVITY_NO_ANIMATION` – vypne animaci při spouštění aktivity.
- `FLAG_ACTIVITY_NO_HISTORY` – spuštěná aktivita se neuloží do zásobníku. Když uživatel aktivitu opustí, bude ukončena.

- IFLAG_ACTIVITY_CLEAR_TOP – pokud už je spouštěná aktivita v zásobníku, obnoví se a všechny aktivity nad ní budou ukončeny.
- FLAG_ACTIVITY_REORDER_TO_FRONT – pokud už je spouštěná aktivita v zásobníku, obnoví se a přesune se navrch zásobníku.
- FLAG_ACTIVITY_SINGLE_TOP – pokud je spouštěná aktivita na vrcholu zásobníku, obnoví se a nespustí se nová aktivita.

Příznaky se nastavují metodou:

```
Intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
```

Způsob spuštění aktivity se může definovat i v *AndroidManifest.xml* atributem `android:launchMode`:

- `standard` – vždy bude vytvořena nová instance.
- `singleTop` – vždy bude vytvořena nová instance, pokud už není instance aktivity na vrcholu zásobníku.
- `singleTask` – bude vytvořen nový task, pokud instance aktivity už existuje, dojde k přesměrování na ni. Typickým příkladem je webový prohlížeč či přehrávač multimédií.
- `singleInstance` – jako `singleTask`, ale v rámci tasku je vždy jen jedna aktivita.

POZNÁMKA

Tlačítko Zpět funguje vždy bez ohledu na hodnotu parametru `android:launchMode`.

Odevzdávání údajů a výsledků

Třída `Bundle` umožňuje přenos údajů mezi komponentami. Údaje se ukládají ve formě párových dvojic klíč-hodnota. Přenos údajů přes mechanizmus Bundle je poměrně rychlý, je ale určen jen pro menší objemy dat, řádově kilobajty. Instanci třídy `Bundle` je možné získat pomocí metody třídy `Intent getExtras()`.

Třída `Bundle` bude přenášet libovolné datové typy. Proto je potřeba definovat rozhraní zajišťující serializaci tříd `Serializable` nebo `Parcelable`. Rozhraní `Serializable` umožňuje dynamické rozpoznávání typů, které je však náročné, takže tento způsob je řádově pomalejší než serializace pomocí rozhraní `Parcelable`, které vyžaduje explicitní definici pro serializaci i deserializaci třídy.

Pokud se po spuštění aktivity očekává, že když aktivita skončí, vrátí výsledek, je potřeba aktivitu spustit pomocí metody `startActivityForResult()`. Výsledek se nastaví pomocí metody `setResult()` v metodě `finish()`. V aktivitě, která očekává výsledek od jiné aktivity, je potřeba předefinovat metodu `onActivityResult()`. K dispozici jsou tři předdefinované hodnoty:

- `RESULT_OK = -1`
- `RESULT_CANCELLED = 0`
- `RESULT_FIRST_USED = 1`

Uživatelsky definované hodnoty výsledků aktivity následují od konstanty `RESULT_FIRST_USED`.

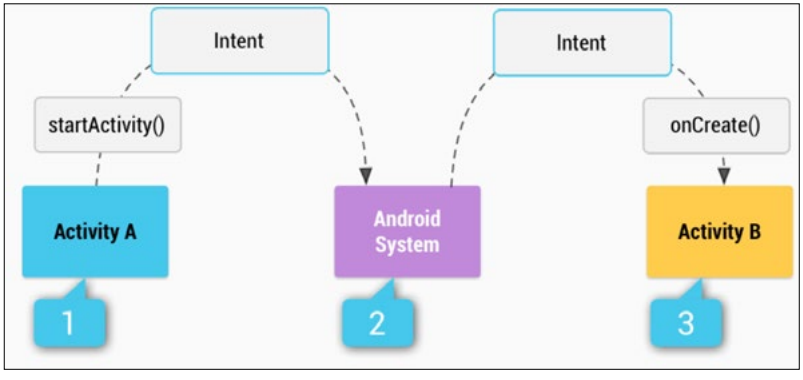
Intent Filter

Aplikace, aktivita či komponenta mohou reagovat na různé druhy záměrů. Android však potřebuje v okamžiku spouštění aplikace vědět, na jaké záměry bude možné reagovat. Každá aplikace má ve svém manifestu v souboru *AndroidManifest.xml* takzvaný `IntentFilter`, který definuje typy záměrů, jež dokáže příslušná aplikace obsloužit.

```
<activity ...
  <intent-filter ...>
   ...
    <action android:name="actionName" />
    ...
  </intent-filter>
  ...
</activity>
```

Například:

```
<activity android:name="ShareActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```



Obrázek 3.7: Princip fungování záměru

Třída `IntentFilter` umožňuje komponentě reagovat pouze na vybraný typ záměrů, které dokáže zpracovat. `IntentFilter` je definovaný kategorií, akcí a typem údajů.

Základní rozdělení záměrů vychází z jejich směrování:

- **Explicitní** – přesně určuje, jakou akci je potřeba provést, kdo a jakým způsobem ji provede a s jakými údaji. Mají explicitně definovanou cílovou aktivitu, kterou je potřeba spustit.
- **Implicitní** – záměr nemá definovanou cílovou aktivitu, ale operaci, která má být vykonaná, proto spustí aktivitu, která může tuto operaci vykonávat. Výběr je ponechán na rozhodnutí operačního systému. Pokud je vhodných aktivit více, musí o výběru rozhodnout uživatel. Typickým příkladem je snímání fotografie nebo zobrazení dokumentu či webové stránky.

Třída `Intent` je v podstatě datová struktura, která se používá ke dvěma účelům. Umožňuje zadat operaci, kterou je potřeba provést, nebo záměr může představovat událost, která nastala v systému a o které je potřeba informovat další komponenty. Pokud aplikace chce provést operaci, ke které potřebuje součinnost jiné aktivity, například vybrat kontakt, zobrazit mapu, sejmout obrázek, vytočit telefonní číslo a podobně, vyjádří to přes objekt `Intent`.

Třída `Intent` má několik důležitých atributů.

Action

Deklaruje typ požadované akce. Například:

- `action_dial` – záměr vytočit telefonní číslo
- `action_edit` – záměr editovat údaje
- `action_sync` – záměr synchronizovat některé údaje ze zařízení s údaji na serveru
- `action_main` – předvolená aktivita aplikace

```
Intent newInt = new Intent();
newInt.setAction(Intent.ACTION_DIAL);
```


Data

Záměry obsahují i datová pole, která obsahují data spojená se záměrem. Data jsou formátovaná jako URI. Například pro zobrazení objektu na mapě:

```
Uri.parse("geo:0,0?q=1600+Pennsylvania+Ave+Washington+DC")
```

Nebo pro vytočení telefonního čísla:

```
Uri.parse("tel:+4211234567")
```

Všimněte si, že řetězce, které obsahují údaje, se nejdříve odevzdávají jako textový řetězec prostřednictvím metody `Uri.parse`. Metoda vrátí objekt URI.

```
Intent newInt = new Intent(Intent.ACTION_DIAL);
newInt.setData(Uri.parse("tel: :+4211234567"));
```

Category

Atribut obsahuje další informace o typu komponenty, která může zpracovat požadavek deklarovaný přes záměr. Například:

- `category_browsable` – tento požadavek může být odevzdán webovému prohlížeči prostřednictvím odkazu ve formě URI
- `category_launcher` – cílová aktivita může být hlavní aktivita úlohy

Type

MIME typ údajů záměru, například `image/*`, `image/png`, `image/jpg`, `text/plain`, `text/html`. Pokud atribut nezadáte, Android se ho bude snažit odvodit. Atribut se nastavuje pomocí metod `Intent.setType(String type)` nebo `Intent.setDataAndType(Uri data, String type)`.

Component

Konkrétní objekt, který by měl provést požadovanou operaci.

Extras

Umožňuje uložit další informace související se záměrem.

Flags

Informace o tom, jak by mělo být zacházeno se záměrem. Například `FLAG_ACTIVITY_NO_HISTORY` zabrání uložení aktivity do zásobníku historie.

Pokud například aktivita může vytočit telefonní číslo, měla by to deklarovat jako standardní akci `intent.action_dial`. Do sekce filtrů záměrů se vloží textový řetězec `android.intent.action.dial`.

```
<activity ...
  <intent-filter ...>
    ...
    <action android:name="android.intent.action.DIAL" />
    ...
  </intent-filter>
  ...
</activity>
```

Pokud je pro daný záměr k dispozici více aktivit, rozhoduje se Android podle priorit, a pokud není, nechá konečné rozhodnutí na uživateli aplikace.

Hodnoty priority mohou být v rozmezí od -1 000 do 1 000, přičemž platí, že vyšší hodnoty znamenají vyšší prioritu.

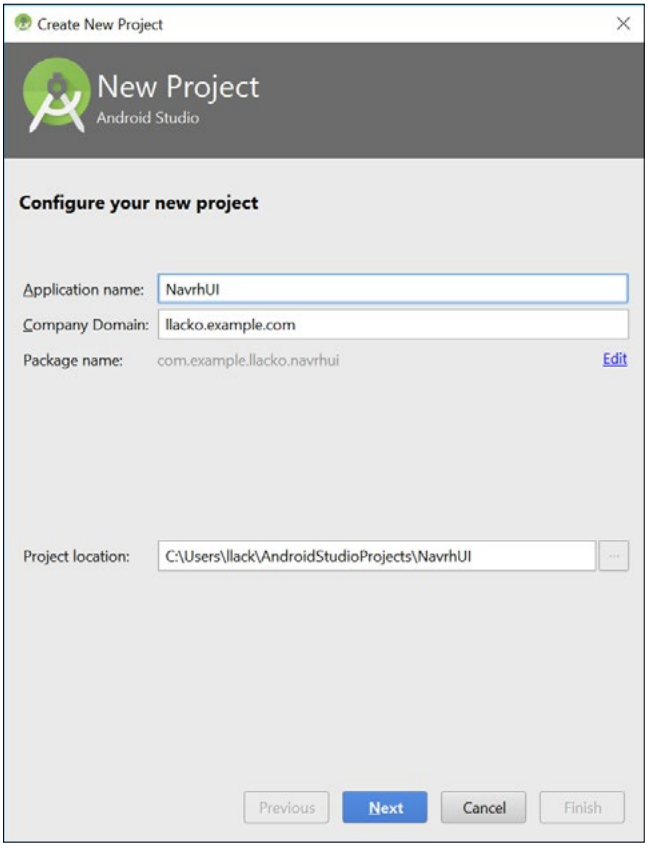
Příklad – Anatomie projektu

Na rozdíl od příkladu v první kapitole, jehož cílem bylo nejrychlejší vytvoření fungující aplikace, aby bylo možné ověřit správnost instalace a konfigurace vývojového prostředí, cílem tohoto projektu je detailně „pitvat" jeho anatomii. To znamená, že popíšeme význam jednotlivých souborů, které projekt tvoří. Je to důležité nejen pro začátečníka (pro něhož tato publikace není primárně určena), ale především pro vývojáře migrující z jiných platforem. Zároveň ukážeme základní principy vizuálního návrhu uživatelského rozhraní aplikace.

V tomto projektu využijeme šablonu hlavní aktivity typu **Basic Activity** (předtím **Blank Activity**), součástí níž je nejen aplikační lišta, ale i takzvané plovoucí tlačítko na ploše aplikace, tj. prvek typu `FloatingActionButton`. Šablona typu **Basic** s předdefinovanou aplikační lištou je velmi užitečná, avšak ne vždy potřebujete zároveň i plovoucí tlačítko, proto v příkladu ukážeme postup, jak toto tlačítko po vytvoření projektu z aplikace odstranit.

Vytvoření projektu

Projekt ve vývojovém prostředí s nainstalovaným a správně nakonfigurovaným Android SDK vytvoříte pomocí volby nabídky **File → New → Android Application Project**. V prvním dialogovém okně vytvoření projektu jsou tři editační pole na zadání názvu:



Obrázek 3.8: Vytvoření nového projektu – zadání názvu

- **Application name** – název zadaný do tohoto pole se bude zobrazovat při spouštění aplikace. Napište do pole vhodný název, v tomto případě „NavrhUI". Po zadání názvu aplikace do pole **Application name** se v poli **Project location** zobrazí stejný název souboru s vynechanými mezerami.
- **Company Domain** – doména firmy, případně vývojáře.
- **Package name** – do tohoto pole se zadává název javového balíčku, do kterého bude projekt aplikace přibalen. Název balíčku musí být jedinečný v rámci všech balíčků nainstalovaných v systému Android. Identifikuje vaši aplikaci v aplikačním systému Androidu. Možná si položíte otázku, jak byl odvozen název `com.example.llacko.navrhui`. Připomíná vám něco? Ano, je to název, který začíná opačným názvem domény, například domény vaší firmy a podobně. Pokud máte doménu například *www.supervyvojar.com* a název vaší aplikace bude `GenialniPomocnik`, název balíčku aplikace v poli **Package name** bude `com.supervyvojar.genialnipomocnik`. Název balíčku můžete změnit klepnutím na odkaz **Edit**.

UPOZORNĚNÍ

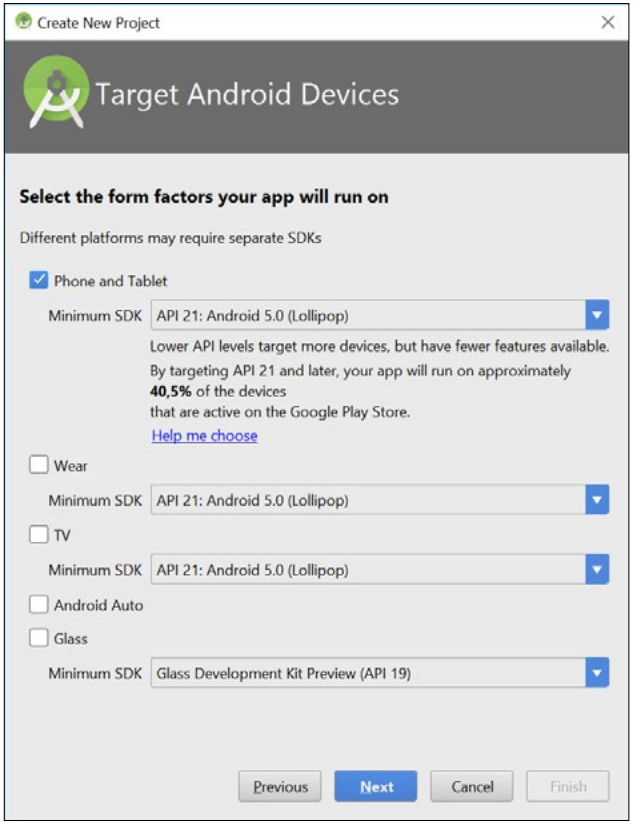
Pokud plánujete aplikaci publikovat na Google Play, nemůžete použít implicitně nastavený jmenný prostor `com.example`. Pro cvičný příklad však takovýto jmenný prostor postačí.

Výběr cílové platformy

Důležitý je výběr minimální verze SDK, to znamená, na jaké škále verzí operačního systému Android bude možné vaši aplikaci spustit. Aktuálně nastavená verze se dá kdykoliv změnit v místní nabídce projektu – po klepnutí pravým tlačítkem myši na název projektu zvolte **Properties → Android**.

V této fázi vytváření projektu je potřeba se zamyslet nad tím, kterou minimální verzi bude vaše aplikace podporovat a vůči které verzi platformy ji budete kompilovat.

Oba parametry můžete sice později v manifestu (soubor *AndroidManifest.xml*) snadno změnit, pokud ale začnete vytvářet aplikace pro novější verzi a později se rozhodnete podporovat i starší verze, musíte přidat knihovny kompatibility. Opačně: pokud se rozhodnete ukončit podporu starších verzí, bude užitečné funkce, které jste předtím realizovali pomocí knihoven kompatibility, upravit tak, aby využívaly přímo funkce zabudované v nové verzi.



Obrázek 3.9: Vytvoření nového projektu – výběr cílové platformy

Nové aplikace proto až na výjimky, které chcete směřovat i pro uživatele starších přístrojů, stačí psát pro verzi API 14: Android 4.4 – KitKat a vyšší. Můžete tak bez omezení používat nové prvky uživatelského rozraní.

Všimněte si, že aplikace je možné vyvíjet pro několik konstrukčních forem zařízení.

- **Phone and Tablet**: standardní platformy pro běh aplikací.
- **Wear**: volbu označíte, pokud má vaše aplikace běžet (i) na platformě Android Wear, tedy na nositelných zařízeních, především na inteligentních hodinkách.
- **TV**: volbu označíte, pokud má vaše aplikace běžet (i) na platformě Android TV na velké obrazovce. U této platformy je však specifická ani ne tak velká obrazovka, ale to, že aplikace se neovládá pomocí dotykového displeje jako na telefonech a tabletech, ale především pomocí dálkového ovládání.
- **Glass**: aplikace pro Google Glass. V době psaní publikace byla tato platforma v útlumu a čekalo se na nástupce.

Pro tuto aplikaci nechejte označenou implicitní volbu **Phone and Tablet.**

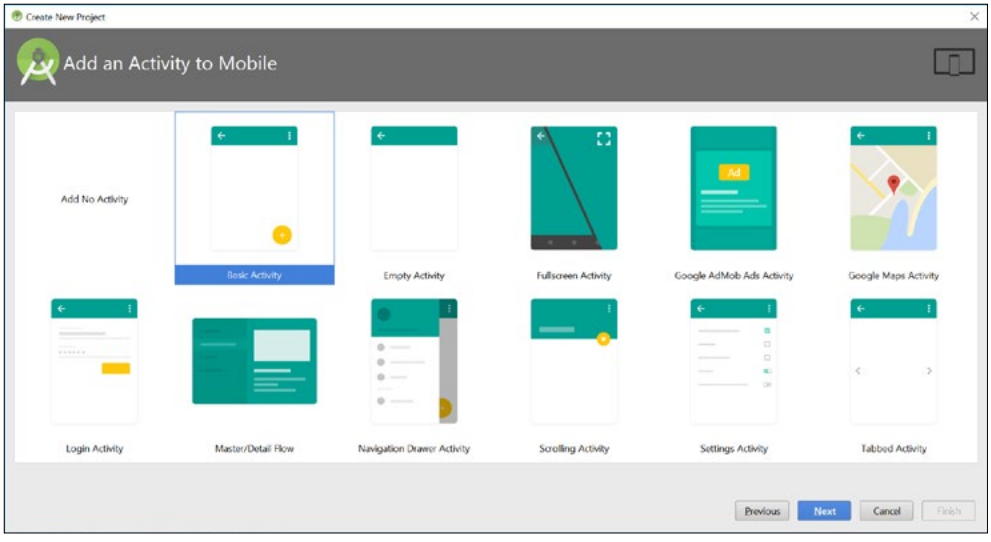
Výběr typu hlavní aktivity

Pojem aktivita je podrobněji definován v druhé kapitole. Proto jen stručně připomeneme, že aktivita je základním prvkem uživatelského rozhraní aplikace.

Aktivita je vizuální reprezentací – prezentační vrstvou aplikace. Aplikace může mít několik aktivit.

K vytvoření hlavní aktivity aplikace je k dispozici několik typů aktivit (v závislosti na verzi SDK):

- **Blank Activity** – jako volitelný prvek navigace se používá navigační lišta **ActionBar** s ovládacími prvky.
- **Empty Activity** – prázdný formulář bez navigačních prvků.
- **Fullscreen Activity** – využívá celou velikost obrazovky.
- **Google Maps Activity** – šablona aktivity s využitím mapové služby Google Maps.
- **Google Play Services Activity** – šablona aktivity s inicializací připojení na reklamní systém Google.
- **Master/Detail Flow** – hodí se pro seznamy objektů typu master-detail. Zobrazovací plocha je rozdělena na dva sloupce. V levém sloupci je seznam objektů a v pravém sloupci jsou zobrazeny detaily vážící se k vybranému objektu.
- **Navigation Drawer Activity** – aktivita využívající **Navigation Drawer**.
- **Scrolling Activity** – aktivita pro obrazovky, které jsou jen výřezem větší virtuální plochy.
- **Settings Activity** – aktivita pro nastavování parametrů aplikace.
- **Tabbed Activity** – aktivita typu **Blank Activity** s možností přepínání složek gestem vodorovného posunu.



Obrázek 3.10: Výběr typu aktivity. Typ uživatelského prostředí názorněji než název vysvětluje obrázek v dialogovém okně

Pro tuto jednoduchou aplikaci označte volbu **Basic Activity**. Uživatelské rozhraní aktivity obsahuje aplikační lištu, nabídku, pracovní plochu a jedno pohyblivé tlačítko akce.

V následujícím dialogovém okně můžete nakonfigurovat detaily pro vybraný typ aktivity:

- **Activity Name**
- **Layout Name**
- **Title**
- **Menu Resource Name**

Zadaný textový řetězec v poli **Activity Name** bude použit k vygenerování souboru s kódem třídy aktivity v programovacím jazyku Java. Stejně zadaný textový řetězec v poli **Layout Name** bude použit pro název souboru XML s definicí uživatelského rozhraní aktivity.

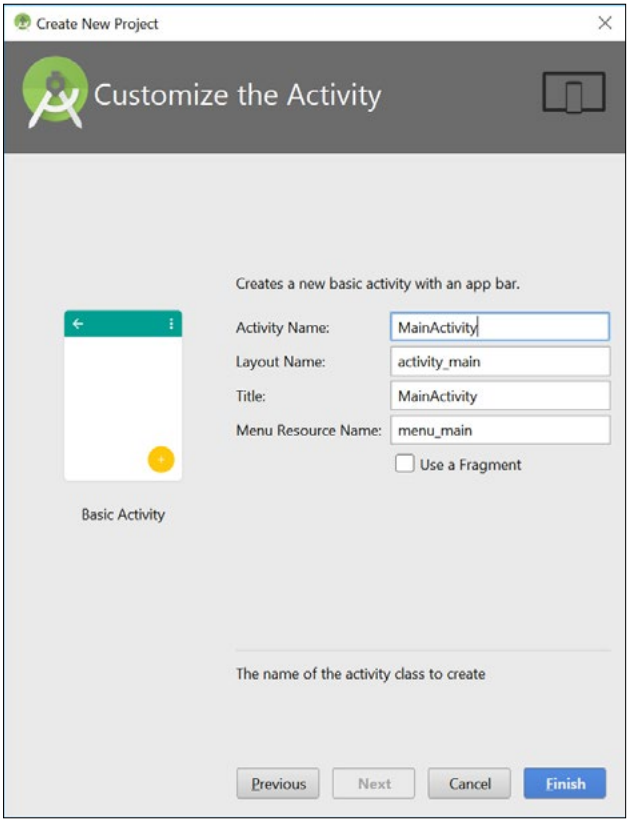
V tomto prvním projektu ponechejte implicitně nastavené hodnoty. Po spuštění aplikace bude zobrazena aktivita s názvem **MainActivity**. Vytvoří se dva soubory.

- *MainActivity.java* – kód třídy aktivity
- *activity_main.xml* – definice uživatelského rozhraní

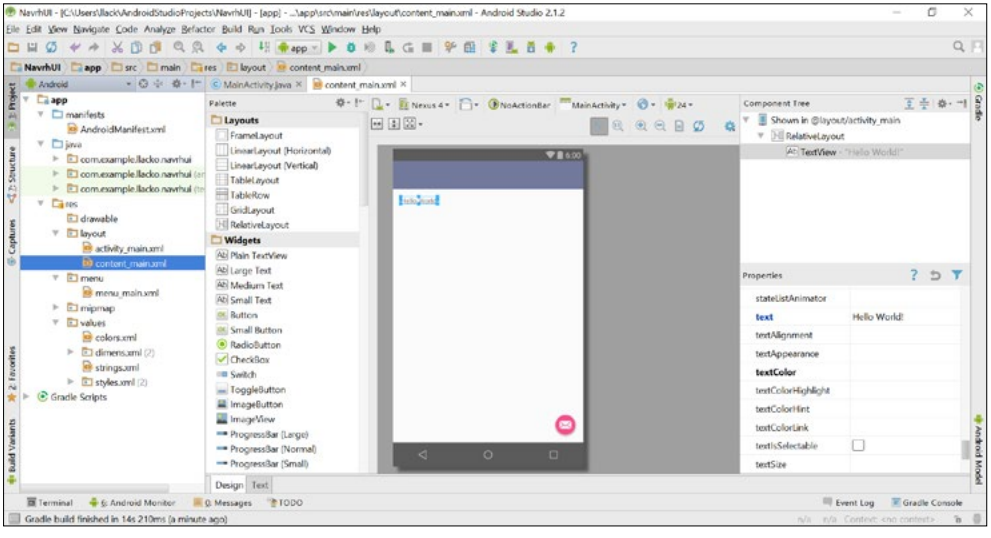
Tlačítkem **Finish** dokončete vytvoření projektu.

Složky a soubory tvořící projekt

Po vytvoření prvního projektu doporučujeme hlavně začátečníkům a vývojářům migrujícím z jiných platforem, aby se seznámili s umístěním a významem klíčových souborů, které tvoří projekt.



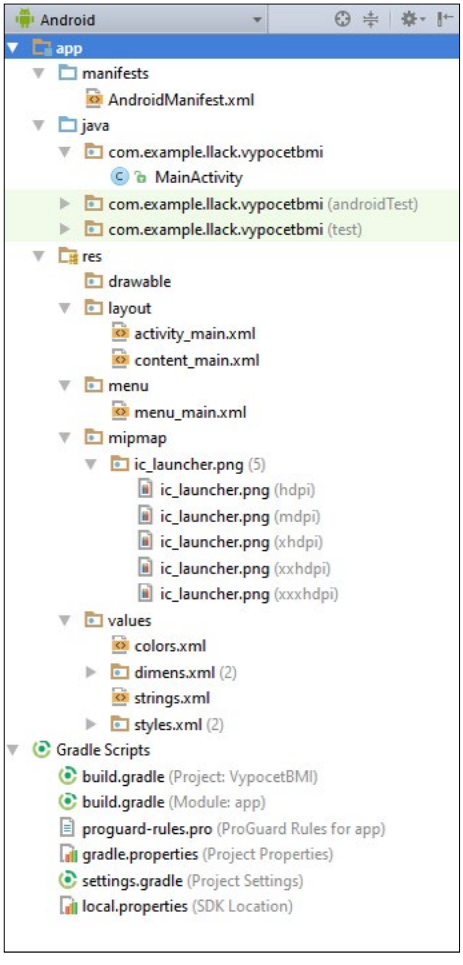
Obrázek 3.11: Konfigurace vybraného typu aktivity



Obrázek 3.12: Rozvržení obrazovky vývojového prostředí po vytvoření projektu

Součásti projektu, se kterými přijdete nejvíce do kontaktu, jsou ve složkách:

- *java* – ve složce se nacházejí soubory s kódem tříd v programovacím jazyce Java.
- *res* – ve složce se nacházejí XML soubory s definicí uživatelského rozhraní.
- *manifestst* – aplikační manifest obsahující důležité informace o projektu.
- *Gradle scripts* – skripty pro kompilátor Gradle. Každý modul má svůj soubor *build.gradle*.



Obrázek 3.13: Složky s nejdůležitějšími součástmi projektu

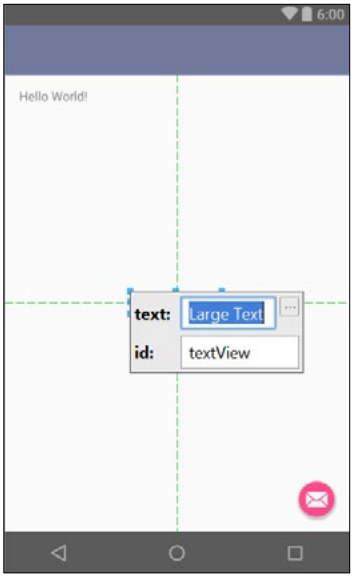
Jednoduchý vizuální návrh uživatelského rozhraní

Dříve než začneme „pitvat“ anatomii projektu aplikace v Android Studiu, ukážeme základní principy vizuálního návrhu uživatelského rozhraní. Požadovaný prvek snadno přesunete z okna **Palette** na požadované místo. Komponenty uživatelského rozhraní můžete též přidat výběrem požadovaného objektu z palety a následným klepnutím na místo, kam se má prvek vložit. Přesuňte ze složky **Widget** na plochu vaší aplikace prvek **Large Text**. Předpokládejme, že to bude statický text. Krokem, který bude logicky následovat, je změna implicitního textového řetězce, jež koresponduje s názvem prvku, tedy „Large text“ na požadovaný text, v našem případě například „Textový reťazec“. Poklepáním na prvek aktivujete malé dialogové okno na zadání textového řetězce (**text**) a identifikátoru (**id**). Změňte text na „Textový reťazec“ a potvrďte.

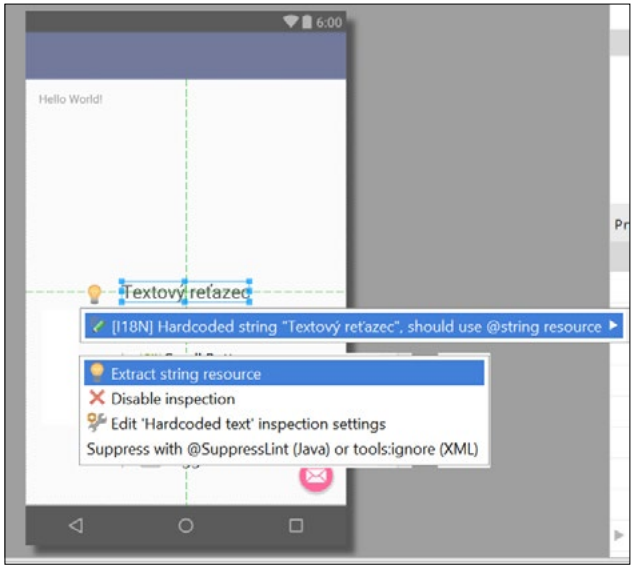
Když zadaný textový řetězec potvrdíte, zobrazí se vlevo vedle prvku ikona žárovky, která indikuje potenciální problém. Pokud na tuto ikonku klepnete, dostanete popis problému a návrh řešení. V tomto případě je popis problému:

```
[I18N] Hardcoded string "Textový reťazec", should use @string resource
```

Nejprve rozšifrujeme kód problému. Asi by vás nenapadlo, že `[I18N]` znamená „internationalization“. Princip odvození je nyní už jasný. Slovo začíná na písmeno „I“, končí na písmeno „N“ a mezi těmito znaky je ještě 18 dalších písmen. Android Studio vás upozorňuje, že byste neměli zadávat texty jako konstanty přímo do definic prvků, protože byste v budoucnosti, pokud bude vaše aplikace úspěšná, mohli mít problémy s lokalizací uživatelského rozraní do jiných jazyků. Proto by všechny texty a podobné hodnoty měly být uloženy jako resource. Potom nebude s lokalizací vaší aplikace žádný problém a bude možné jej realizovat bez jakéhokoliv zásahu do kódu aplikace. Klepněte na šipku napravo vedle varovného oznámení a Android Studio vám nabídne jedno nebo několik možných řešení.

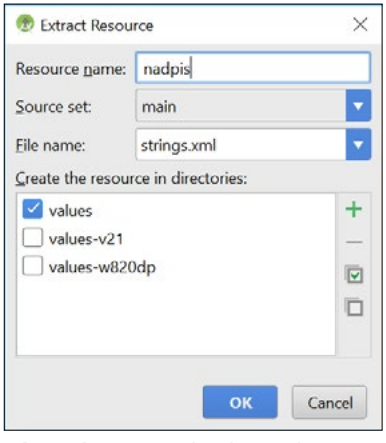


Obrázek 3.14: Změna parametrů prvku, aktivuje se poklepáním



Obrázek 3.15: Problém při návrhu uživatelského rozhraní a návrh možných řešení

V tomto případě je řešením možnost **Extract string resource**. Zadejte název resource a potvrďte. Definice se uloží do souboru *strings.xml*.



Obrázek 3.16: Dialog k extrakci textového řetězce z návrhu prvku do zdrojů

Pojďme se pro zajímavost podívat do souboru **app** → **res** → **values** → **strings.xml**, kde jsou definice textových řetězců. Zdrojům se budeme věnovat podrobněji, takže na tomto místě jen ukážeme obsah souboru.

```
<resources>
  <string name="app_name">NavrhUI</string>
  <string name="action_settings">Settings</string>
  <string name="nadpis">Textový reťazec</string>
</resources>
```

Výhody takovéhoto soustředění textových řetězců při případné lokalizaci aplikace ani nemusíme zdůrazňovat.

V souboru *content_main.xml* je odkaz na příslušný resource.

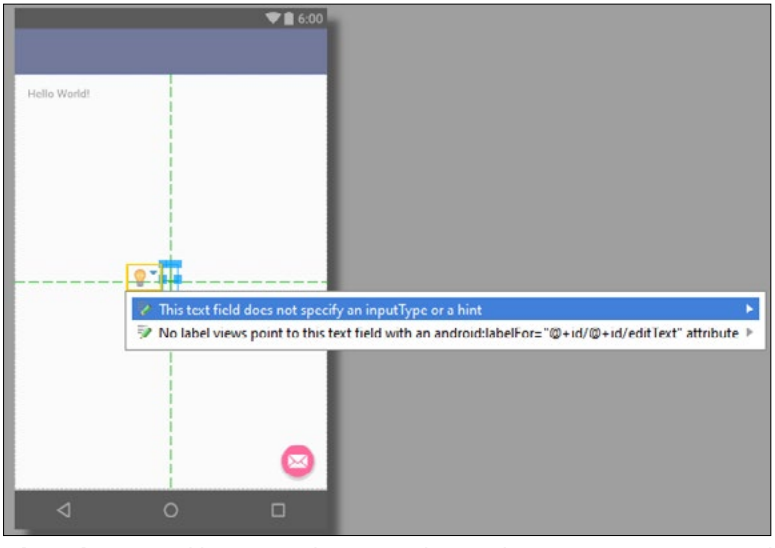
```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textAppearance="?android:attr/textAppearanceLarge"
  android:text="@string/nadpis"
  android:id="@+id/textView"
  android:layout_centerVertical="true"
  android:layout_centerHorizontal="true" />
```

TIP

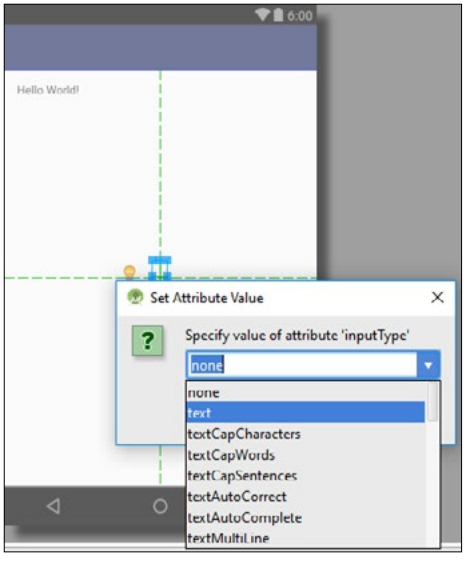
Pokud ve zdrojovém kódu návrhu uživatelského rozhraní klepnete na název resource, v tomto případě **@string/nadpis**, a použijete klávesovou zkratku **Ctrl+B**, otevře se soubor s definicí.

Abychom ilustrovali, jak vám Android Studio pomáhá při návrhu uživatelského rozhraní, doporučujeme přidat na plochu aktivity ještě prvek `EditText`. Při práci s prvkem `EditText` je potřeba deklarovat typ vstupu pro zobrazení. Takto definujete typ textu nebo dat, která budou zadána uživatelem. Například pokud je typ nastaven na zadávání telefonního čísla, uživatel bude moct zadat jen číselné hodnoty tvořící telefonní číslo. Pokud je typ vstupu nastaven na `TextCapCharacters`, vstup bude v předvoleném nastavení na zadávání textu velkými písmeny. Nastavení typu vstupu je možné kombinovat. Pokud chcete umožnit zadávat všeobecný text, nastavte typ textu `text`.

Ikona žárovky naznačuje, že vývojové prostředí pro vás má nápovědu, jak postupovat při konfiguraci prvku umístěného na plochu aktivity při vizuálním návrhu.



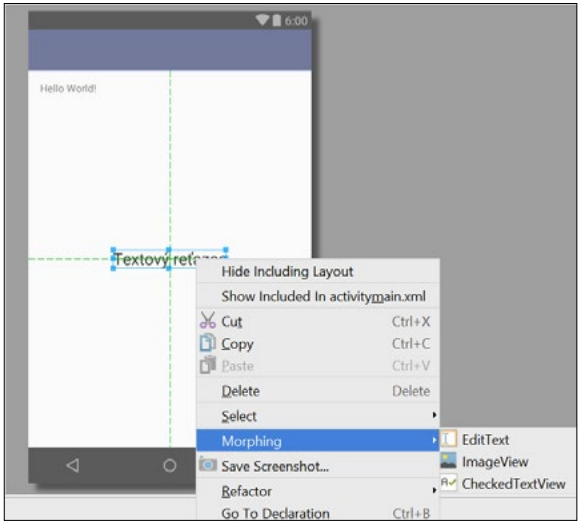
Obrázek 3.17: Po klepnutí na ikonu žárovky se zobrazí možnosti, co je potřeba při definování prvku doplnit



Obrázek 3.18: Nabídka možností hodnot atributů inputType

Morphing – změna typu prvku

Při vývoji a ladění aplikace, pravděpodobně nejčastěji po první interakci designéra s první funkční verzí aplikace, je možné, že vznikne podnět na modifikaci některých prvků, zpravidla na podobné. Například bude potřeba změnit prvek `TextView` na `EditView`, změnit tlačítko na jiný typ a podobně. Android Studio poskytuje možnost tzv. Morphingu, to znamená změny definice prvku na jiný podobný. Položku **Morphing** najdete v místní nabídce vizuálního návrhu daného prvku. V nabídce možností zjistíte, na jaký prvek se dá vámi původně navržený prvek změnit. V případě prvku `TextView` je to `EditText`, `ImageView` a `CheckedTextView`.



Obrázek 3.19: Místní nabídka pro Morphing

Prvek TextView:

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textAppearance="?android:attr/textAppearanceLarge"
  android:text="@string/nadpis"
  android:id="@+id/textView"
  android:layout_centerVertical="true"
  android:layout_centerHorizontal="true" />
```

Změněn na EditText takto:

```
<EditText
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textAppearance="?android:attr/textAppearanceLarge"
  android:text="@string/nadpis"
  android:id="@+id/textView"
  android:layout_centerVertical="true"
  android:layout_centerHorizontal="true" />
```


Morphing samozřejmě provede jen základní změnu v návrhovém kódu a ne všechny parametry původního prvku jsou kompatibilní nebo vyhovující, pro nový prvek. Proto vás vývojové prostředí na tuto skutečnost upozorní a přes nabídku u ikonky žárovky navrhne řešení.



Obrázek 3.20: Návrhy na změny prvku vzniklého přes Morphing

Definice uživatelského rozhraní

Po vytvoření projektu se v hlavním okně uživatelského rozhraní vývojového prostředí zobrazí definice návrhu uživatelského rozhraní. Pokud vyberete šablonu pro typ aktivity **Basic**, definice je ve dvou souborech ve složce */res/layout/*.

Soubor *content_main.xml* obsahuje definici uživatelského rozhraní aktivity:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
```

```
  android:paddingTop="@dimen/activity_vertical_margin"
  app:layout_behavior="@string/appbar_scrolling_view_behavior"
  tools:context="com.example.llacko.navrhui.MainActivity"
  tools:showIn="@layout/activity_main">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="@string/nadpis"
    android:id="@+id/textView"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Soubor *activity_main.xml* obsahuje definici uživatelského rozhraní rámce, tj. aplikační lišty, toolbaru, plovoucího tlačítka a podobně. Dokument XML má jeden kořenový element, v tomto případě `CoordinatorLayout`, do kterého se postupně vnořují další layouty, v tomto případě `AppBarLayout`, a v nich prvky, v původní terminologii `Views`, které tvoří uživatelské rozhraní aplikace.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:fitsSystemWindows="true"
  tools:context="com.example.llacko.navrhui.MainActivity">

  <android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/AppTheme.AppBarOverlay">

    <android.support.v7.widget.Toolbar
      android:id="@+id/toolbar"
      android:layout_width="match_parent"
      android:layout_height="?attr/actionBarSize"
      android:background="?attr/colorPrimary"
      app:popupTheme="@style/AppTheme.PopupOverlay" />

  </android.support.design.widget.AppBarLayout>

  <include layout="@layout/content_main" />
```

```
    <android.support.design.widget.FloatingActionButton
      android:id="@+id/fab"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="bottom|end"
      android:layout_margin="@dimen/fab_margin"
      android:src="@android:drawable/ic_dialog_email" />

</android.support.design.widget.CoordinatorLayout>
```

Význam tohoto kódu nejlépe pochopíte, když přepnete vývojové prostředí do režimu zobrazení grafického návrhu. Přepněte se klepnutím na kartu **Design** v levé dolní části hlavního okna vývojového prostředí. Zobrazí se grafická podoba uživatelského rozhraní nové aplikace. Na ploše aplikace je textové oznámení v horní části a okrouhlé tlačítko s ikonou obálky ve spodní části.

UPOZORNĚNÍ

Prvky uživatelského rozhraní se často nazývají widgety. Zde však nastává nejednoznačnost v terminologii. Uživatel totiž pod pojmem widget rozumí jednoduchou aplikaci, kterou si může umístit na pracovní plochu a která něco zobrazuje, například hodiny, počasí, informace o přijatých zprávách a podobně.

Některé aplikace obsahují samostatné definice rozložení prvků uživatelského rozhraní ve složkách.

- */res/layout-xlarge* – pro displeje s velmi vysokým rozlišením, nejméně 960 × 720 dp
- */res/layout-large* – pro displeje s vysokým rozlišením, nejméně 640 × 480 dp
- */res/layout-land* – definice pro orientaci zařízení na šířku

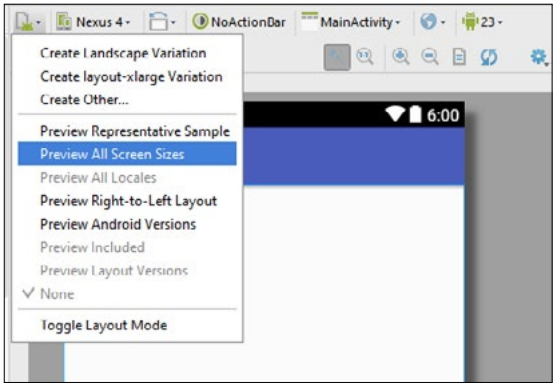
Jednotky dp znamenají density independent pixels.

Případně mohou být definice uživatelského rozhraní v samostatných souborech pro orientaci na výšku a na šířku. Potom může vzniknout například tato struktura:

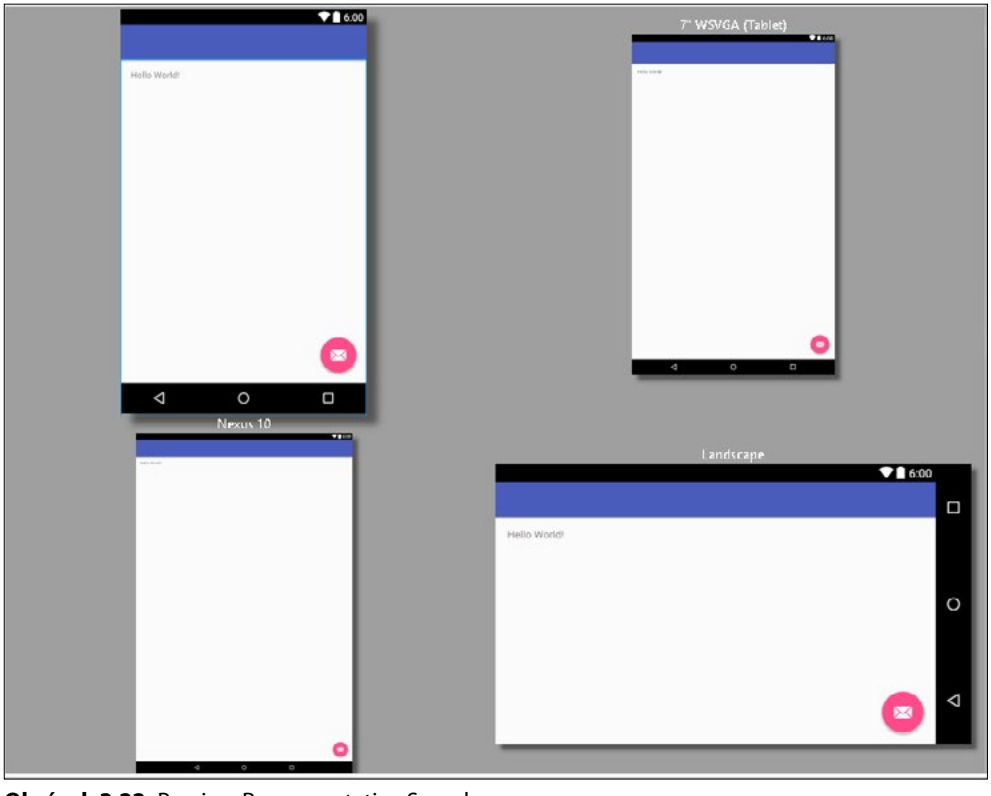
```
MyProject/
  res/
    layout/              # default (portrait)
      main.xml
    layout-land/         # landscape
      main.xml
    layout-large/        # large (portrait)
      main.xml
    layout-large-land/   # large landscape
      main.xml
```

Uživatelské rozhraní aplikace na různých zařízeních

Jelikož je Android implementován na množství zařízení s různou úhlopříčkou displeje, různým rozlišením a různou orientací (na výšku × na šířku), vývojáři ocení režim **Preview All Screen Sizes**, který zobrazí náhled změn uživatelského rozhraní v nejčastěji používaných rozlišeních, nebo **Preview Respresentative Sample**.



Obrázek 3.21: Aktivace módu Preview All Screen Sizes



Obrázek 3.22: Preview Respresentative Sample

Z dosud uvedeného vyplývá, že k návrhu uživatelského rozhraní můžete přistupovat buď deklarativně pomocí vizuálního návrhu, nebo imperativně – psaním kódu XML definice uživatelského rozhraní. Záleží jen na vás, pro který způsob se ve které konkrétní situaci rozhodnete. Nejen pro migrující vývojáře je výhodnější deklarativní způsob, jelikož je mnohem méně pracný a umožňuje podstatně rychlejší návrh. Vizuálnímu návrhu uživatelského rozhraní se věnuje další kapitola.

Aplikační kód

Kód hlavní aktivity je v souboru *MainActivity.java*.

```
package com.example.llack.vypocetbmi;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        Snackbar.make(view, "Replace with your own action",
          Snackbar.LENGTH_LONG)
          .setAction("Action", null).show();
      }
    });
  }

  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
  }

  @Override
  public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
      return true;
```

```
        }

        return super.onOptionsItemSelected(item);
    }
}
```

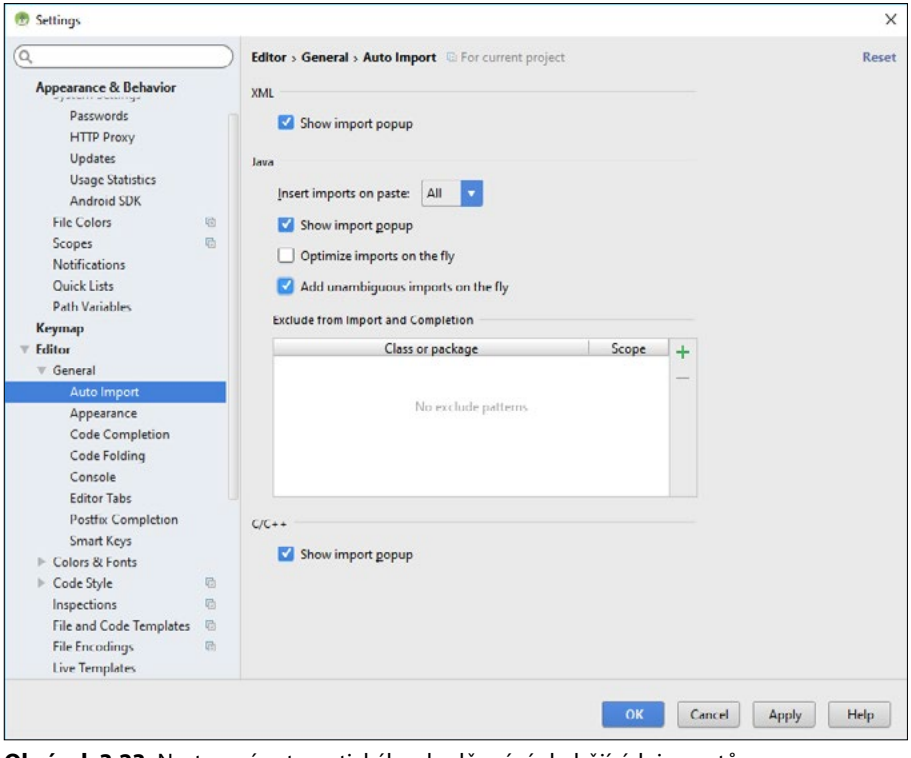
Aplikace může zapouzdřovat více samostatných aktivit, tyto jsou však vzhledem k možnosti platformy řízeny sériově, to znamená, že uživatel s nimi pracuje postupně. Metodu `OnCreate()` volá operační systém při vstupu do akce, takže v této metodě se zpravidla provádí inicializace uživatelského rozhraní.

Metoda `setContentView` přebírá identifikátor layoutu nebo objekt typu `View`. V našem nově vytvořeném projektu se přebírá identifikátor hlavní aktivity `R.layout.activity_main`.

POZNÁMKA

Toto je první a zároveň poslední kód v této publikaci, ve kterém jsou i importy jmenných prostorů tříd. V dalších zdrojových kódech importy pro úsporu místa uvádět nebudeme.

Android Studio sice nemá na doplnění chybějících importů „kouzelnou" klávesovou zkratku **Ctrl+Shift+O** známou z Eclipse, můžete si ale nastavit ve **File → Settings → Editor → General → Auto Import** parametr **Insert imports on paste** na hodnotu **All** a označit volbu **Add unambigious imports on the fly**.



Obrázek 3.23: Nastavení automatického doplňování chybějících importů

Když už byly zmíněny užitečné klávesové zkratky Android Studia, jejich seznam získáte přes nabídku **Help → Default Keymap Reference**.

Odstranění tlačítka FloatingActionButton z Basic Activity

V tomto projektu jsme využili šablonu hlavní aktivity typu **Basic Activity**, součástí níž je nejen aplikační lišta, ale i takzvané plovoucí tlačítko na ploše aplikace, tj. prvek typu `FloatingActionButton`. Jak jsme v úvodu příkladu slíbili, ukážeme postup, jak toto tlačítko po vytvoření projektu z aplikace odstranit.

Ve vizuálním návrhu v souboru *activity_main.xml* nebo pokud jste použili jiný než implicitní název aktivity, odstraňte v souboru začínajícím *activity_* fialové tlačítko v pravém dolním rohu plochy aplikace nebo v XML kódu uživatelského rozhraní vymažte následující blok:

```
<android.support.design.widget.FloatingActionButton
  android:id="@+id/fab"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="bottom|end"
  android:layout_margin="@dimen/fab_margin"
  android:src="@android:drawable/ic_dialog_email" />
```

V souboru *MainActivity.java* odstraňte následující blok kódu:

```
FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {
    Snackbar.make(view, "Replace with your own action",
      Snackbar.LENGTH_LONG)
      .setAction("Action", null).show();
  }
});
```


Definice objektů ve zdrojích (resources)

V aplikacích se využívá množství definic objektů. Typickým příkladem jsou textové řetězce, definice barev, tvarů a podobně. K ukládání definic slouží adresář zdrojů */res*.

Ve složce *res* jsou definice různých objektů v závislosti na charakteru aplikace. Nejčastěji v této složce najdete vnořené podsložky:

- *Layout* – definice rozmístění prvků uživatelského rozhraní
- *Menu* – definice struktury menu
- *Strings* – definice textových řetězců
- *Colors* – definice barev použitých v aplikaci
- *Dimens* – rozměry v jednotkách dp (density independent pixel)
- *Drawable* – definice obrázků (PNG, JPG, GIF) a tvarů pro vykreslování
- *Drawable* – rozlišení – definice obrázků v šesti možných rozlišeních
- *Styles* – definice stylů pro prvky uživatelského rozhraní

- *XML* – datové soubory
- *Anim* – definice animací
- *Interpolator* – definice objektů na řízení animací
- *Mipmap* – definice objektů 3D grafiky
- *Raw* – podsložka pro libovolný typ souborů

TIP

Důrazně doporučujeme vytvářet definice objektů ve zdrojích. Pozdější změnou hodnoty v definičním souboru se automaticky změní daný objekt ve všech layoutech, které ho využívají.

Především z důvodu jednoduché lokalizace do jiných jazyků doporučujeme všechny textové řetězce umístit do souboru *strings.xml*. V návrhovém kódu XML, kódu v Javě a manifestu na tyto řetězce budete jen odkazovat. *V publikaci toto pravidlo občas ve cvičných aplikacích pro názornost a zkrácení výpisů porušíme.*

Příklad souboru *strings.xml*:

```
<resources>
  <string name="app_name">NavrhUI</string>
  <string name="action_settings">Settings</string>
  <string name="nadpis">Textový reťazec</string>
</resources>
```

Kromě textů je možné definovat i formátování jejich výpisů pomocí tagů HTML a sekvencí escape. V některých případech je užitečné definovat množná čísla podstatných jmen pro různé jazyky. Pokud byste vkládali řetězce natvrdo do kódu XML návrhu nebo do kódu Javy, vaše aplikace by se prakticky nedala lokalizovat do dalších jazyků. Při tvorbě aplikace přece vždy předpokládáte, že bude mít úspěch a bude se šířit nebo prodávat v mnoha zemích, a abyste motivovali další zájemce, budete svou aplikaci lokalizovat do nejpoužívanějších světových jazyků (španělštiny, francouzštiny, němčiny, ruštiny).

Například seznam barev použitých v aplikaci byste definovali v souboru *colors.xml*:

```
<resources>
  <color name="red">#ffff0000</color>
  <color name="green">#ff00ff00</color>
  <color name="blue">#ff0000ff</color>
  <color name="gray">#ff808080</color>
  <color name="black">#ff000000</color>
  <color name="white">#ffffffff</color>
  <color name="magenta">#ffff00ff</color>
  <color name="orange">#ffffa500</color>
  <color name="transparent">#00000000</color>
</resources>
```

POZNÁMKA

Vytvořený identifikátor má tvar *R.typ.nazov_suboru_bez_pripony*. Proto musí být názvy souborů s resources unikátní.

Ve složce *values* jsou definice jednoduchých objektů, například textových řetězců. Názvy souborů v této složce nehrají takovou úlohu jako v ostatních případech.

Textové řetězce

Textové řetězce jsou jedním ze základních stavebních kamenů uživatelského rozhraní. Vyskytují se v mnoha widgetech, například `Button`, `EditText`... Můžete definovat samostatné textové řetězce, případně pole řetězců.

V aplikacích často potřebujete nějakým způsobem zvýraznit část textu. Na formátování se využívají tagy známé z kódu HTML:

- `<b>` tučné písmo (**bold**)
- `<i>` kurzíva (*italic*)
- `<u>` podtržený text (underline)

Například:

```
<string name="hello_world">Toto je <b>textový</b> reťazec!</string>
```

Na uvozovky a apostrofy se používají takzvané sekvence Escape začínající obráceným lomítkem.

Pole řetězců

Pole řetězců se hodí například k definování malého objemu statických textových informací.

Například:

```
<resources>
  <string-array name="planety_array">
    <item>Merkur</item>
    <item>Venuse</item>
    <item>Zem</item>
    ...
  </string-array>
</resources>
```

V aplikačním kódu se s polem řetězců pracuje následovně:

```
Resources res = getResources();
String[] saPlanety = res.getStringArray(R.array.planety_array);
```

Množné číslo podstatných jmen

Každý jazyk má při tvorbě množného čísla podstatných jmen určitá specifika. V angličtině je to jednoduché, stačí přidat koncovku –s. V češtině a slovenštině jsou pravidla komplikovanější. Využívají se tři formy. Například:

- Jeden pes
- Dva, tři, čtyři psi
- Pět a více psů

Některé případy jsou nepravidelné, například

- Jeden člověk
- Dva, tři, čtyři lidé
- Pět a více lidí

Namísto pracného definování přes podmínky typu `if()` se využívá univerzální mechanizmus, který rozpoznává definice pro plurály typu zero, one, two, few, many, other.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <plurals name="PocetOsob">
    <item quantity="one">V miestnosti je jedna osoba.</item>
    <item quantity="few"> V miestnosti sú %d osoby.</item>
    <item quantity="other"> V miestnosti je %d osôb.</item>
  </plurals>
</resources>
```

Vhodnou verzi plurálu získáte pomocí funkce `getQuantityString()`.

Modifikátory

Všimněte si, že v názvech některých složek se nacházejí různé zkratky oddělené pomlčkou, například *drawable-hdpi*, *drawable-ldpi*. Jedná se o takzvané modifikátory, v původní terminologii qualifiers, které umožňují přizpůsobit aplikaci různému rozlišení obrazovky, lokalizovat ji do více jazyků a podobně. Předpis pro název podsložky *resources* v případě použití modifikátoru je:

```
<resources_name>-<config_qualifier>.
```

Příklad struktury složek pro více rozlišení:

```
res/
  drawable/
    icon.png
    background.png
  drawable-hdpi/
    icon.png
    background.png
```

Příklady modifikátorů (seřazení podle tabulky z původní dokumentace):

- Jazyk a region: `sk`, `cs`, `en`, `en-rUS`, `en-rGB`
- Šířka displeje: `w<N>dp` například `w720dp`, `w480dp`
- Orientace zařízení: `port`, `land`
- Rozlišení displeje v pixelech: `ldpi`, `mdpi`, `hdpi`, `xhdpi`
- Verze API: například `v15`

Tabulku s kompletním seznamem modifikátorů najdete na *http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources*.

UPOZORNĚNÍ

Pořadí v tabulce je důležité. Můžete totiž použít současně i více modifikátorů a v takovém případě je musíte seřadit podle pořadí, v jakém jsou vyjmenovány v tabulce – shora dolů.

Android využívá prioritu modifikátorů v pořadí od nejpřesnější specifikace po nejvšeobecnější. Například pokud máte pro jazyk a region složky */res/values*, */res/values-en* a */res/values-en-rUS* a tablet je nastaven na americkou angličtinu, systém vyhledává zdroje nejprve ve složce *values-en-rUS*. Pokud neuspěje, hledá o jednu hierarchickou úroveň výše ve složce *values-en*, a pokud ani tam neuspěje, hledá ve složce *values*. Pokud neuspěje ani tam, nezbude nic jiného než vygenerovat chybu.

Aplikační manifest

V seznamování se s anatomií projektu aplikace pro Android pokračujeme prozkoumáním aplikačního manifestu.

Je to hlavní konfigurační soubor pro aplikaci. Definuje jednotlivé komponenty, nastavení konfigurace a oprávnění aplikace.

Aplikační manifest najdete v souboru *AndroidManifest.xml* v hlavní složce projektu. Nejprve prozkoumáme soubor XML a potom si prohlédneme manifest v návrhovém zobrazení. V Android Studiu otevřete manifest a přepněte se na kartu **AndroidManifest.xml** úplně vpravo. Tím zobrazíte manifest ve formátu XML. Všimněte si elementu `<uses-sdk>`, ve kterém je definovaná minimální a aktuální verze Android SDK.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.vypocetbmi"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="19" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.mojeprvniaplikace.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

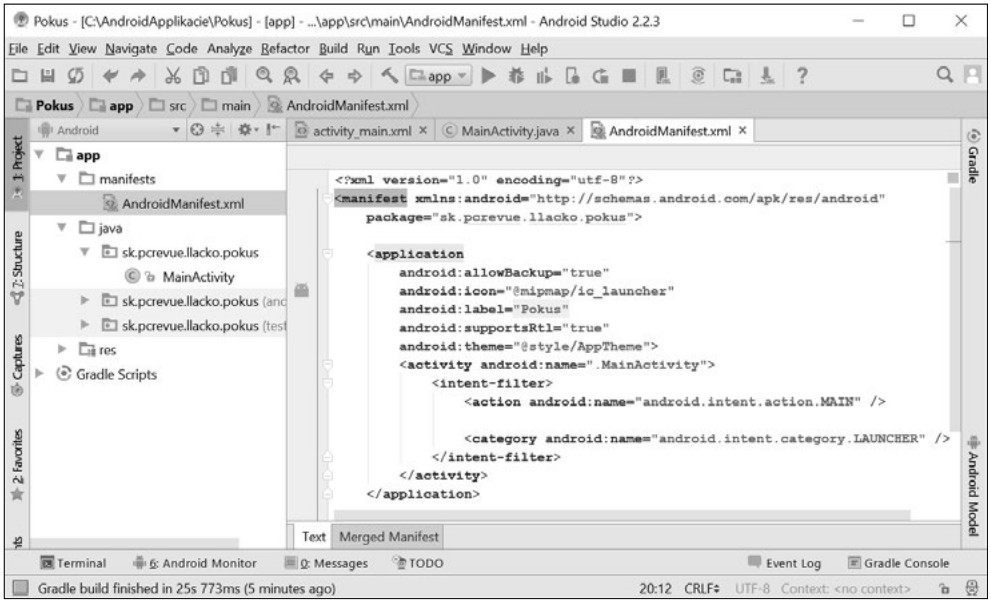
V elementu `<activity>` je seznam všech aktivit aplikace.

Pozornost věnujte i sekci `<intent-filter>`. Intent (záměr) je definovaný jako abstrakce operace, kterou je potřeba provést, nebo jinými slovy definuje přesuny mezi aktivitami. Pomocí záměrů je možné propojit nejen aktivity aplikace, ale i aktivity více aplikací. Aplikace, která chce aktivovat nějakou `Activity`, zavolá metodu `startActivity()`, kde je požadavek přesněji specifikován pomocí parametru. Následně systém vybere objekt `Activity`, který nejvíce vyhovuje specifikaci.

POZNÁMKA

Filtr `<category android:name="android.intent.category.LAUNCHER" />` deklaruje, že ikona aplikace se zobrazí na obrazovce pro spouštění aplikací.

Při editování manifestu velmi pomáhá barevné rozlišení syntaxe.



Obrázek 3.24: AndroidManifest v návrhovém zobrazení

Informace o minimální a aktuální verzi SDK jsou uloženy i v souboru *build.gradle* ve složce *Gradle Scripts*.

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 24
  buildToolsVersion "23.0.3"

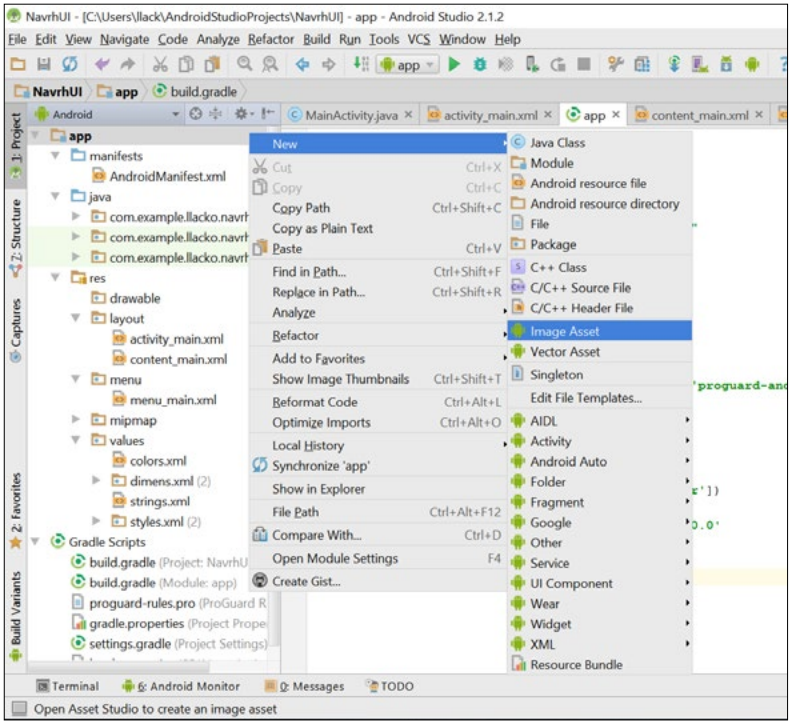
  defaultConfig {
    applicationId "com.example.llacko.navrhui"
    minSdkVersion 21
    targetSdkVersion 24
    versionCode 1
    versionName "1.0"
  }
  buildTypes {
    release {
      minifyEnabled false
      proguardFiles getDefaultProguardFile('proguard-android.txt'),
        'proguard-rules.pro'
    }
  }
}

dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  testCompile 'junit:junit:4.12'
  compile 'com.android.support:appcompat-v7:24.0.0'
  compile 'com.android.support:design:24.0.0'
}
```

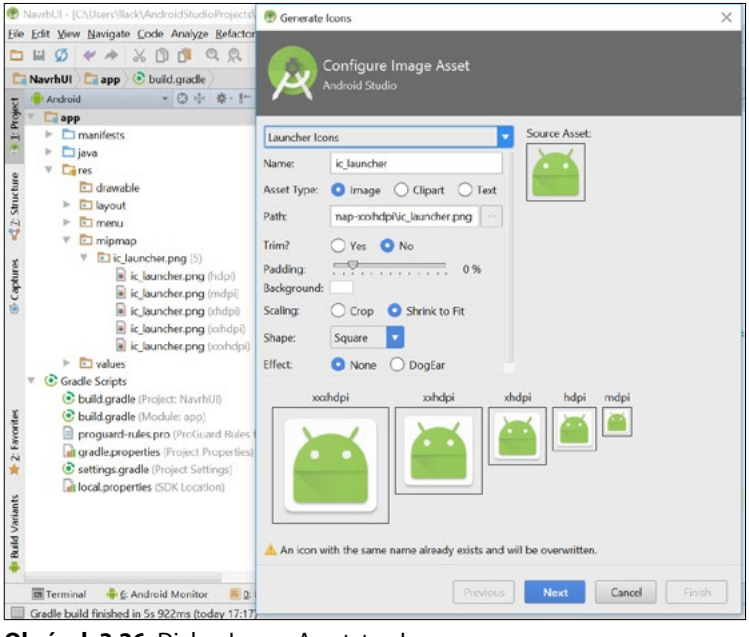

Identifikační grafika aplikace

Každá aplikace obsahuje určité množství grafických prvků, definici ikon, obrázků a jiných typů grafiky, která dotváří design aplikace. Tyto grafické prvky aplikace jsou definovány ve složce projektu **app** → **res** → **mipmap**. Například ve složce **ic_launcher.png** je 5 stejných, avšak různě velkých obrázků zobrazujících ikonu aplikace. Dialog **Image Assets**, pomocí něhož můžete tyto prvky spravovat, zobrazíte v místní nabídce složky **app** (**app** → **New** → **ImageAsset**).

Všimněte si, že ikona je definovaná v pěti velikostech pro různé velikosti a rozlišení obrazovky. Více se dočtete v kapitole věnované designu.



Obrázek 3.25: Aktivace dialogu Image Asset



Obrázek 3.26: Dialog Image Asset, typ Image

KAPITOLA

4

Škálovatelný design

Témata kapitoly:

- Designový styl Material
- Přizpůsobení rozlišení obrazovky
- Metrika designu
- Variabilita zařízení a obrazovek

Zařízení, na kterých budou aplikace spouštěny, mají různou velikost obrazovky a různé rozlišení. Proto by programátoři migrující z klasických aplikací pro Windows měli zapomenout na praxi definování přesných souřadnic polohy a konkrétních rozměrů ovládacích prvků. Na Androidu definujete vzájemné uspořádání prvků, aby se uživatelské rozhraní aplikace mohlo správně zobrazit na libovolném displeji.

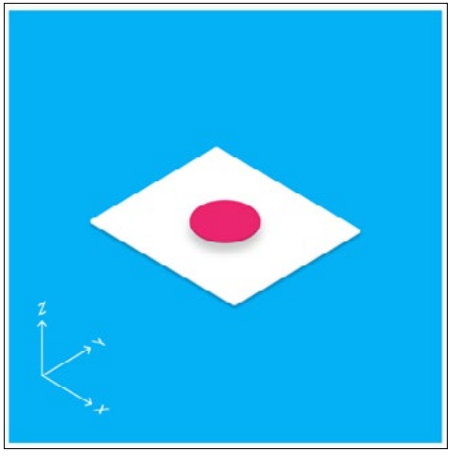
Designový styl Material

„Material design" je způsob/typ designu uživatelského rozhraní vyvinutého společností Google. Je založen na materiálu, který má svou tloušťku a je v prostoru „hmotný". Každý element vrhá stín, může se deformovat, přesouvat v prostoru, seskupit se s jinými elementy a překrývat se, avšak nikdy nemohou být dva elementy na stejném místě. Při celém procesu se pracuje s hloubkou uživatelského rozhraní. Poprvé byl použit nový element – karta, která seskupuje více společných informací do jedné komponenty.

Material design je zobrazení trojrozměrného prostředí obsahující světlo, stíny a plochy z materiálu dané barvou. Všechny významné objekty uživatelského rozhraní mají rozměry v souřadnicích X, Y a Z. Tloušťka materiálu se doporučuje 1 dp (density pixel nezávislý na velikosti obrazovky zařízení). Hlavní světlo vytváří směrové stíny a okolní světlo vytváří měkké stíny. Stíny se tvoří na bázi rozdílu výšky mezi překrývajícími se materiály.



Obrázek 4.1: Princip Material Design



Obrázek 4.2: Navození prostoru

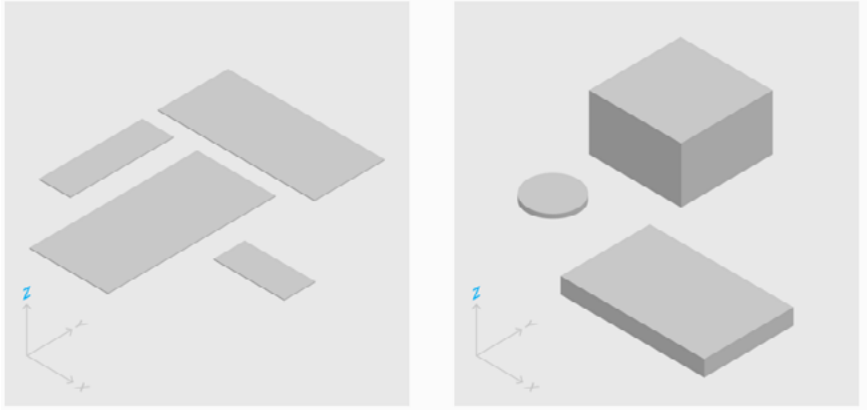
V hmotném prostředí světla osvětlují scénu. Stejný princip využívá i uživatelské rozhraní aplikace akceptující designový styl Material Design. Hlavní světlo vytváří směrové stíny, zatímco okolní světlo vytváří měkké stíny ze všech úhlů. Stíny se vytvoří, když světelné zdroje blokují materiály ploch v různých polohách podél osy Z. Na obrázku je příklad plochy ve virtuální výšce 6 dp.



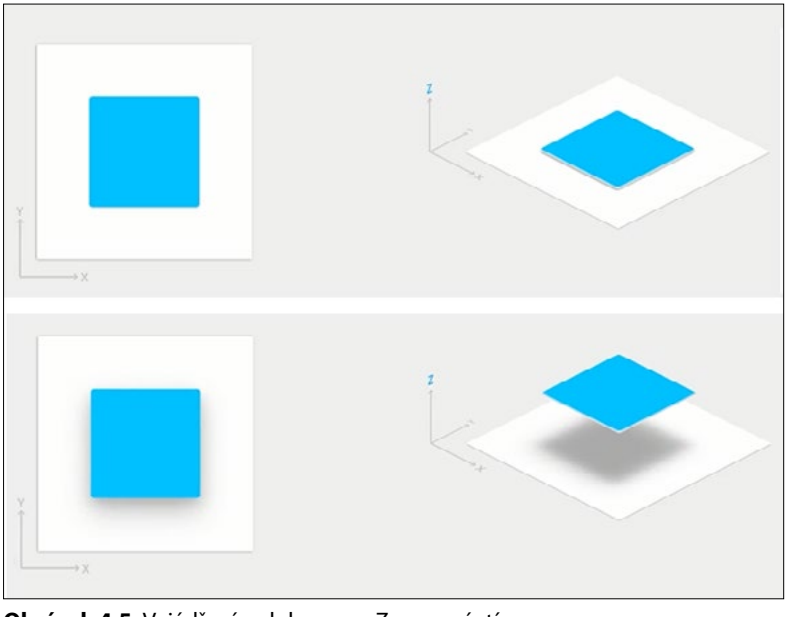
Obrázek 4.3: Navození prostoru pomocí stínu z hlavního a okolního světla. Vlevo je stín od hlavního světla, uprostřed stín od okolního světla a vpravo výsledek – kombinace obou stínů

Plochy uživatelského rozhraní aplikace mohou měnit rozměry, tvar, plochy. Například pokud na ně uživatel klepne, mohou změnit polohu v ose Z a podobně.

Plochy – podobně jako v reálném světě – nemohou zabírat stejné místo v prostoru. Snadno si to představíte na plochách z konkrétního materiálu, například plechu. Můžete položit jednu na druhou, nemohou však splynout v jedné tloušťce. Objekty umístěné nad sebou nebo položené na sebe – podobně jako v reálném světě – vrhají stíny a odrážejí světlo. Definice doporučení Material Design obsahuje parametr nazývaný elevace. Je to rozdíl vzdáleností v ose Z měřený od přední stěny jednoho povrchu k přední stěně druhého povrchu. Rozdíl vzdáleností v ose Z se znázorňuje hloubkou stínu.



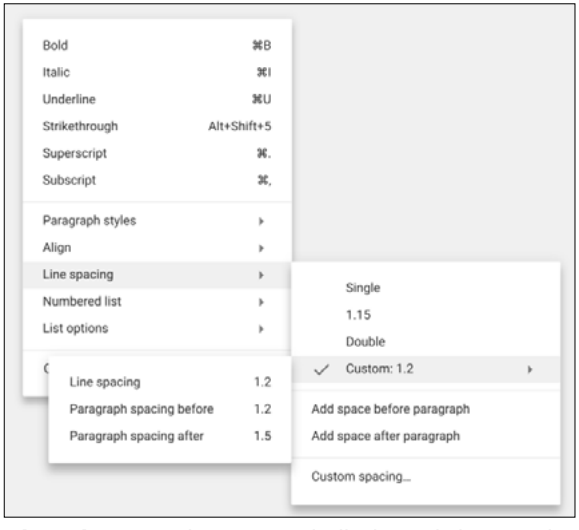
Obrázek 4.4: Materiál může mít různé rozměry, avšak jeho tloušťka je podle designového doporučení Googlu vždy 1 dp. Obrázek vlevo reprezentuje správný design, vpravo nesprávný



Obrázek 4.5: Vyjádření polohy v ose Z pomocí stínu

Elevace může být statická a dynamická, například jako reakce na výběr prvku nebo zvýraznění oznámení, klíčové hodnoty a podobně. V případě, že objekt změní výšku, měl by se, jakmile přestane důvod zvýraznění působit, vrátit do „klidové“ elevace. Komponenty by si měly udržovat konzistentní elevaci i napříč aplikacemi. Typickým příkladem je plovoucí tlačítko dobře známé například z aplikace Messenger.

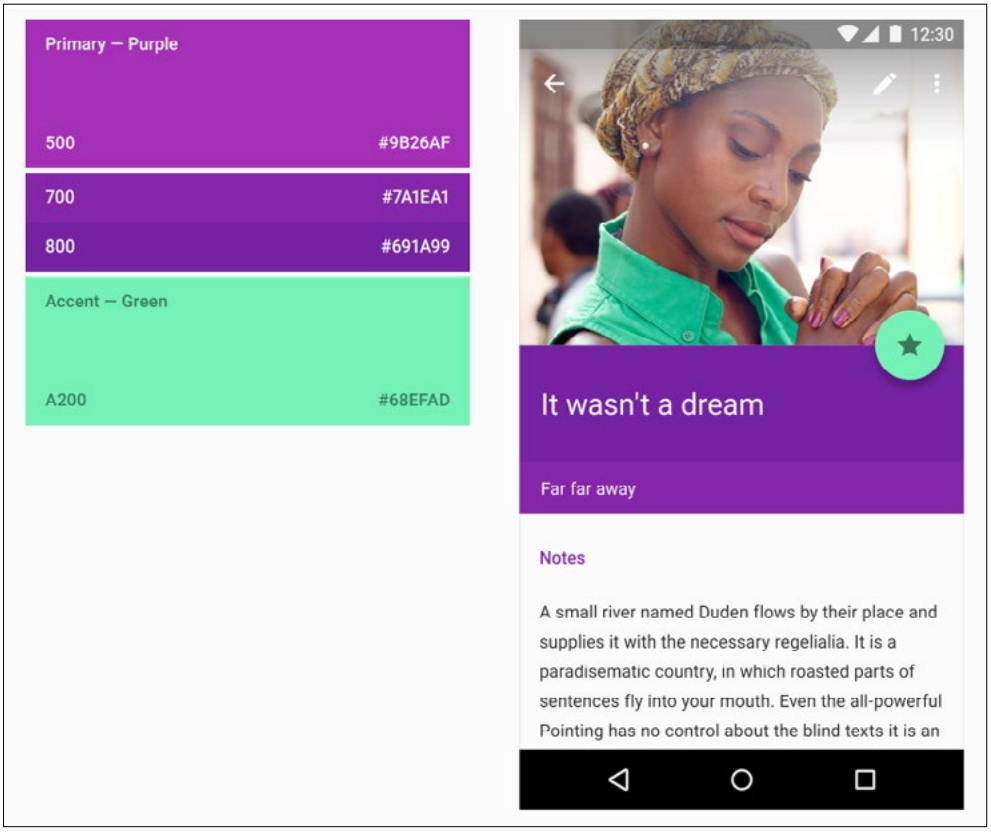
Obrázek 4.6: Plochy nemohou zabírat stejné místo v prostoru. Musí se v ose Z překrývat. Vlevo správně, vpravo nesprávně



Obrázek 4.7: Každá úroveň nabídky by měla být opticky zvednutá o 1 dp

Material design se začal používat od verze Android 5.0 Lollipop a postupně se rozšiřuje a zdokonaluje. V nové verzi Android 7.0 Nougat přibyla v designových doporučeních i takzvaná spodní lišta. Může obsahovat 3–5 tlačítek s ikonami. Tlačítka na této liště musí aktivovat přímo příslušnou funkci, neměla by vyvolávat žádné místní nabídky nebo nastavení. K tomu je určena horní lišta s takzvaným „hamburger“ menu – ikonka s vodorovnými čárkami nad sebou a tlačítkem nastavení s ikonou se třemi tečkami.

Barvami a jejich kombinacemi se zabývá stránka *https://material.google.com/style/color.html #color-color-schemes*. Najdete tam kódy doporučených barev a stínů. Ještě užitečnější inspirací jsou příklady barevných schémat na stránce.



Obrázek 4.8: Příklad doporučení barev v barevném schématu Purple – Green

V sekci pro designéry najdete doporučení pro tvorbu ikon, obrázků v aplikacích a samozřejmě standardních i speciálně navržených ovládacích prvků.

TIP

Doporučujeme po přečtení spustit na telefonu s Androidem 5.0 a vyšším několik aplikací od Googlu, případně jiných renomovaných firem (Adobe, Microsoft...) a zkonfrontovat, zda a jak jsou prezentovaná doporučení respektována a implementována.

Přizpůsobení rozlišení obrazovky

Při návrhu obrázků a ikon je důležité mít na paměti, že vaše aplikace může být nainstalována na různých zařízeních, s různými displeji s různým rozlišením a hustotou pixelů. Geometrickým zvětšením prvku či ikony určené pro malý displej na displeji s rozlišením full HD, případně vyšším, se docílí obrázku bez detailů, často neostrého. Proto máte k dispozici více měřítek, ve kterých můžete vytvářet obrázky. Android u spuštěné aplikace kontroluje vlastnosti displeje

konkrétního přístroje a načítá ikonu v rozlišení odpovídajícím jeho parametrům. Rozlišení se uvádí v dpi (dots per inch), tj. počtu bodů na palec.

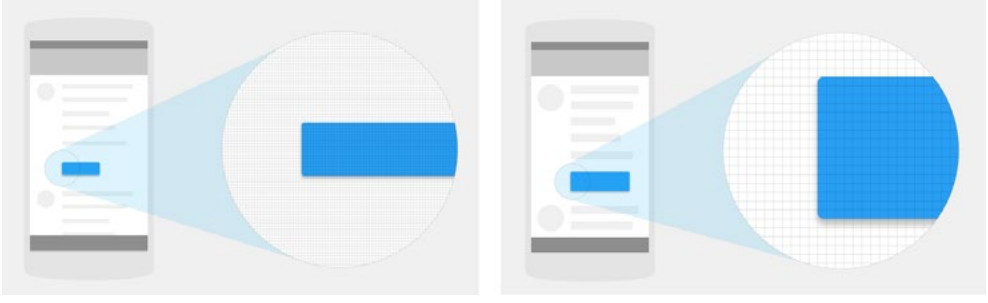
Rozměrové parametry, například rozměry okrajů a rozestupů mezi prvky, se mohou udávat v jednotkách:

- **in** – palce (britská délková jednotka – 2,54 mm)
- **mm** – milimetry
- **pt** – points (1/72 palce)
- **dp** – na rozlišení nezávislé pixely. Je to abstraktní měrná jednotka vztahující se k fyzické hustotě pixelů zobrazovacího zařízení vzhledem k referenčnímu zobrazení 160 dpi
- **sp** – škálovatelné pixely nezávislé na měřítku. Je to jednotka podobná dp, ale bere v úvahu škálování fontů podle uživatelových preferencí
- **px** – rozměr v pixelech aktuálního rozlišení. Nedoporučujeme používat, protože takto definované prvky se nedokáží přizpůsobit velikosti a rozlišení displeje konkrétního zařízení

Podrobnější vysvětlení si zaslouží definování v jednotkách dp (density independent pixel), které jsou definovány vztahem:

1 dp = 160 px/dpi

Z tohoto vztahu logicky vyplývá, že dp vyjadřuje fyzickou velikost displeje, která se na každém zařízení automaticky přepočítá zpět na pixely.



Obrázek 4.9: Zobrazení prvku na displeji s vysokým rozlišením (vlevo) a nízkým rozlišením (vpravo)

Pro rozměry ikon pro různá rozlišení platí tabulka:

mdpi	160 dpi	48 × 48 px	1×
hdpi	240 dpi	42 × 72 px	1,5×
xhdpi	320 dpi	96 × 96 px	2×
xxhdpi	480 dpi	144 × 144 px	3×
xxxhdpi	640 dpi	192 × 192 px	4×

U obrázků je potřeba pro každou hustotu ldpi, mdpi, hdpi a xhdpi vytvořit obrázek v rozdílném rozlišení, a to v poměru 3 : 4 : 6 : 8. Například pro obrázek, který má v ldpi rozměry 180 × 180 pixelů, musíte vytvořit varianty:

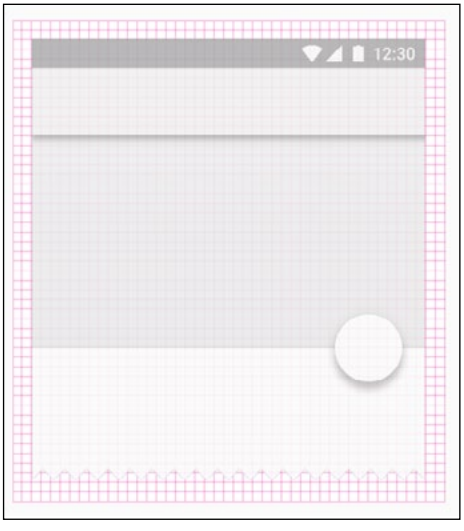
ldpi	180 × 180 px	3×
mdpi	240 × 240 px	4×
hdpi	360 × 360 px	6×
xhdpi	480 × 480 px	8×

Pokud není zadaná hodnota pro příslušné rozlišení, použije se nejbližší vyhovující hodnota.

Metrika designu

Na úvod zdůrazníme dva základní principy návrhu designu aplikace – estetiku a účelnost, které se navzájem synergicky doplňují. Doporučuje se zarovnávání objektů prezentačního rozhraní do mřížky, která vytvoří konzistentní a strukturované rozvržení plochy aplikace. Zarovnání mřížky se předpokládá podle delší strany obrazovky. Tím dosáhnete doporučené siluety aplikace. Prvky uživatelského rozhraní stmeluje mřížka, která určuje rozstup grafických objektů.

Všechny komponenty se doporučuje zarovnat na čtvercovou mřížku s rozměrem pole bez ohledu na to, zda je aplikace určena pro mobilní telefony, tablety či televizory. Ikony v panelu nástrojů se rozmisťují v hustější mřížce se stranou 4 dp. Do takto hustější mřížky se zarovnává i písmo.



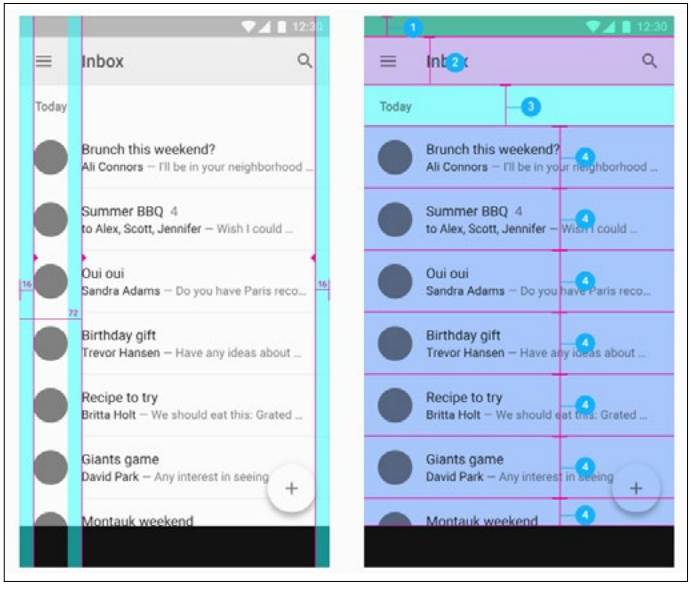
Obrázek 4.10: Zarovnání prvků uživatelského rozhraní do mřížky

Ke zvýšení úhlednosti a čistoty stylu se doporučuje používat okraje: z levé strany 16 dp pro telefon, případně 24 dp pro tablet. Vertikální okraj není nutné řešit, v horní části obrazovky je stavový řádek o výšce 24 dp. Ponechejte co nejvíce prostoru pro obsah. Především neorganizujte obsah pomocí čar a rámečků. K tomuto účelu slouží v Material designu plochy. Optické oddělení obsahu například do sloupců či řádků pomocí okrajů a rozestupů sloupců sice zabere nějaké místo, ale pixely „obětované“ na okraje jsou bohatě kompenzovány tím, že opticky rámují obsah a výrazně ho zpřehledňují, čímž přispívají k lepší orientaci uživatele.

K horizontálnímu oddělení souvisejícího obsahu používejte mezeru: minimálně 16 dp pro telefon a 24 dp pro tablet. K oddělení více grafických seznamů či textových polí se doporučuje mezera 80 dp pro tablet. Pro telefon na výšku bude lepší použít seznam a pole pro detaily na různých, podle potřeby zobrazovaných obrazovkách.

TIP

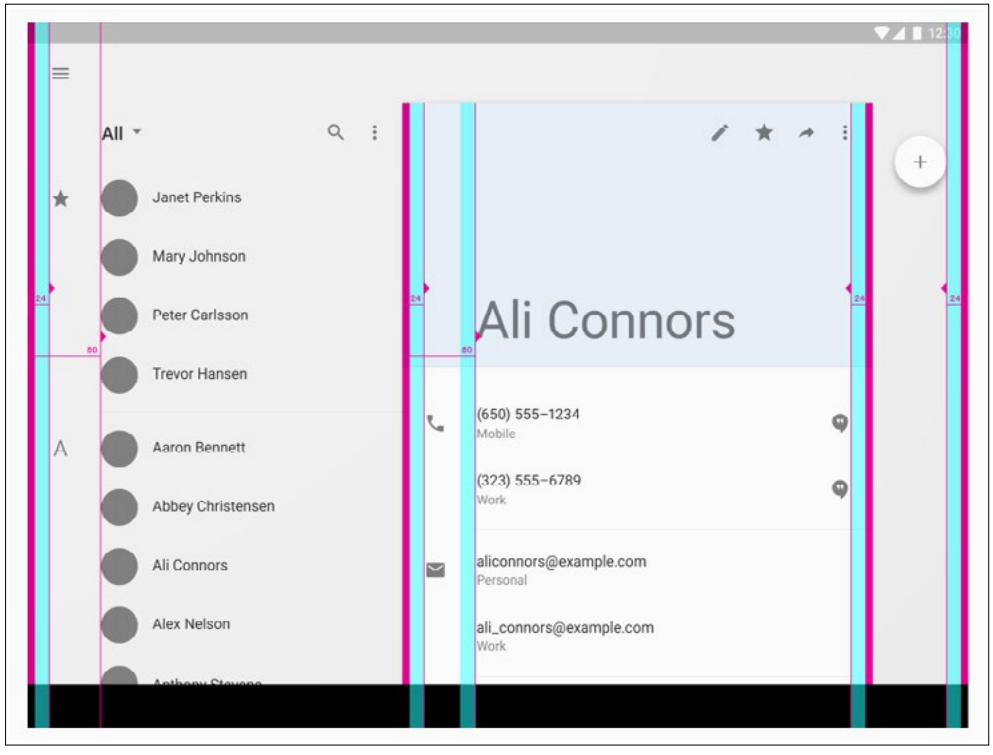
Během případného adaptování starších aplikací na Material design zkuste v prvním kroku ponechat původní rozvržení ovládacích prvků a odstranit jen rámečky a čáry, které rozdělují plochu obrazovky. Často budete překvapeni, jak tento krok přispěl k zpřehlednění prezentačního rozhraní aplikace.



Obrázek 4.11: Vlevo doporučení pro okraje a vzdálenost klíčového sloupce, vpravo vertikální rozestupy

Podobně oddělujte obsah i vertikálně. Mezi položkami seznamu se doporučuje rozestup 72 dp. Doporučená výška jednotlivých polí a řádků seznamu se pro názornost vyjadřuje pomocí legendy:

- Stavový řádek: 24 dp
- Lišta nástrojů (toolbar): 56 dp
- Podnadpis: 48 dp
- Seznam: 72 dp

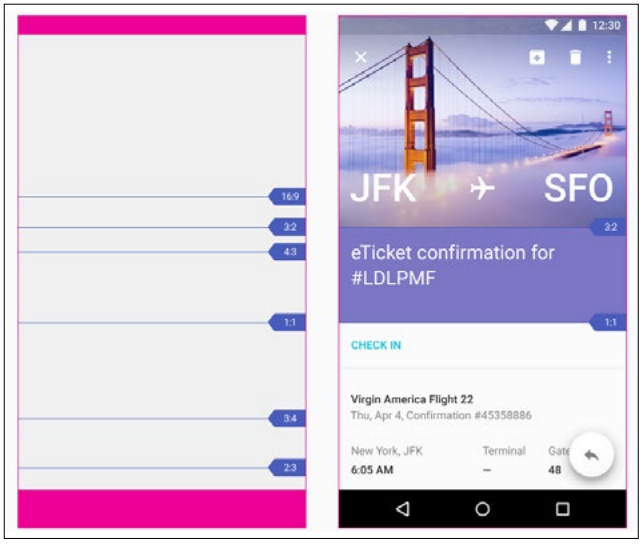


Obrázek 4.12: Doporučení pro okraje a rozestupy sloupců v aplikaci pro tablet

Určitě jste si všimli, že pro poměr stran displeje si výrobci zařízení zpravidla zvolili nějakou osvědčenou hodnotu, nejčastěji 16:9, 4:3, či 3:2. Nebudeme do podrobností řešit estetiku, ale i při tvorbě uživatelského rozhraní Google doporučuje pro obrázky či větší plochy v aplikacích upřednostnit vybrané poměry stran. Doporučené poměry stran ploch jsou 16:9, 3:2, 4:3, 1:1, 3:4, 2:3.

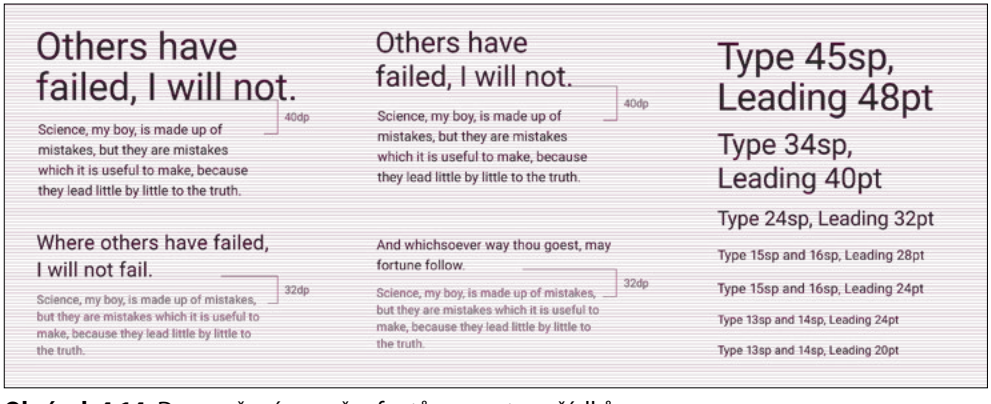
Jen pro úplnost dva jednoduché vzorečky:

- Šířka = Poměr stran * Výška
- Výška = Šířka / Poměr stran



Obrázek 4.13: Doporučené poměry stran ploch uživatelského rozhraní aplikace

Text je jedním z nejdůležitějších výrazových prostředků každé aplikace. Proto je důležité, aby byl textový obsah přehledný. Podle typografie by mělo být uživateli ihned zřejmé, co je důležité, například kde je nadpis, kde jsou podnadpisy, kde najde sumarizované informace, kde detaily, měla by mu být jasná případná hierarchická struktura prezentovaného obsahu.



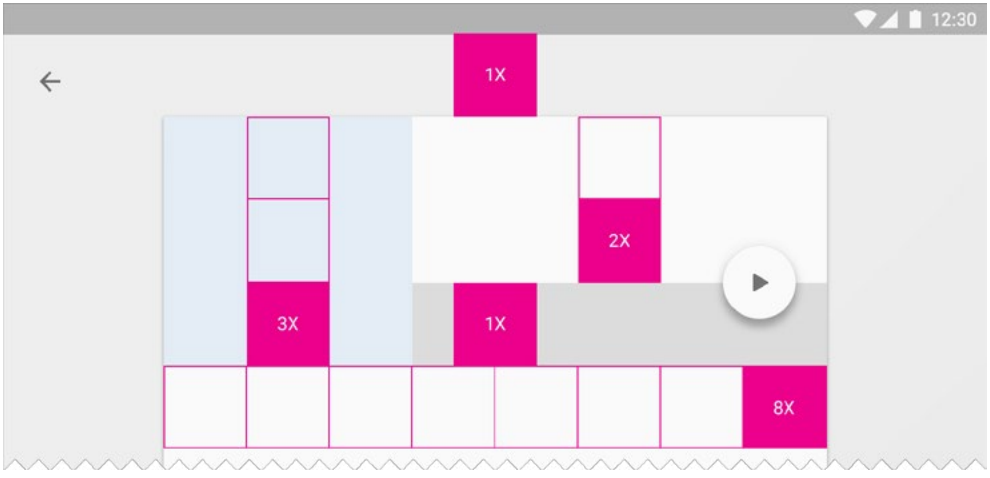
Obrázek 4.14: Doporučené rozměry fontů a rozestupy řádků

Připomínáme, že jednotky znamenají:

- **pt** – points (1/72 palce)
- **sp** – škálovatelné pixely nezávislé na měřítku vztažené k fyzické hustotě pixelů zobrazovacího zařízení vzhledem k referenčnímu zobrazení 160 dpi. Je to jednotka podobná dp, ale bere v úvahu škálování fontů podle uživatelových preferencí

Pravidlo násobků rozměrů

Na závěr části o Material designu uvedeme jedno ze základních designových pravidel. Proč jsou některé designové návrhy uživatelského rozhraní aplikací působivější než jiné? Základní pravidla kompozice platná pro fotografii, například pravidlo třetin či zlatého řezu, platí samozřejmě i pro design, ale ne vždy se dají aplikovat. Je tu však ještě jedno pravidlo, které se týká více designu než malby a fotografie. Zkuste porovnat některé designové grafické výtvory, nemusí to být jen uživatelské rozhraní aplikace, ale třeba i reklamní leták či jiný typografický produkt. Na mnohých grafikách, které se vám líbí, odhalíte, že některé rozměry se v nich často opakují, a to nejen rozměr prvku jako takový, ale i jeho násobky.



Obrázek 4.15: Aplikace pravidla násobků rozměrů

Doporučujeme vzít za základ rozměr některého standardního či dominantního prvku, například výšku panelu akcí. Pokud je například vysoký 56 dp, můžete definovat základní element, jehož násobky budete využívat, na 56 × 56 dp. Určení velikosti dalších prvků bude jen otázkou volby vhodného násobku. Násobky se velmi hodí při změně uživatelského rozhraní aplikace, typickým příkladem je otočení zařízení či paralelní běh dvou aplikací. Pokud jste u orientace na šířku vyžili osminásobek bázového rozměru, na výšku to může být čtyřnásobek a podobně.

POZNÁMKA

Toto pravidlo se lépe aplikuje na uživatelské rozhraní aplikací pro tablety, často ale přijde vhod i u aplikací pro mobily.

Variabilita zařízení a obrazovek

Nikoliv výkon zařízení, ale velikost jeho obrazovky je klíčovým faktorem pro vývoj aplikací. Zařízení můžeme z tohoto hlediska rozdělit na dvě majoritní skupiny – tablety a telefony. Nezapomínejte však, že Android je i na televizorech a nositelných zařízeních, ale pro tyto konstrukční faktory se v současnosti vytvářejí samostatné projekty aplikací. Zatímco tablety mají v závislosti na velikosti zařízení displeje s úhlopříčkou 8–10 palců, některá zařízení vyšší cenové

třídy i více. Rozlišení počínají na HD, tedy 1 280 × 800, 1 366 × 768, případně u lacinějších modelů i 1 024 × 768 pixelů, přes full HD, tj. 1 920 × 1 080 pixelů, až po QHD s 2 560 × 1 440 pixely, případně i více. U tabletů se tedy s uživatelským rozhraním kvůli velikosti a rozlišení obrazovky omezovat nemusíte.

Pokud vytváříte aplikaci určenou primárně pro telefony, musíte brát v úvahu to, že mají malé obrazovky. Zdůrazňujeme malé obrazovky, nikoliv obrazovky s malým rozlišením. Na trhu je hodně telefonů střední třídy s 5" displejem s full HD rozlišením 1 080 × 1 920, u dražších modelů, takzvaných vlajkových lodí, je běžné rozlišení QHD s 2 560 × 1 440 pixely. Nejpoužívanější velikosti displejů jsou 4,5", 4,7", 5" a 5,5". U starších telefonů se setkáte i s rozměrem 4". Jelikož se Apple k tomuto rozměru po několika letech znovu vrátil, můžeme předpokládat, že jeho příklad budou následovat i někteří výrobci telefonů s Androidem.

Fyzická velikost obrazovek přináší několik problémů, jak v uživatelském vstupu, tak v zobrazování výstupů, a je výzvou pro vývojáře aplikací, aby využívali flexibilní návrh uživatelského rozhraní. Pokud v aplikaci navrhnete tlačítko s pevnými rozměry, na vašem 5" telefonu bude dobře vypadat a dobře se ovládat, avšak na 10" tabletu s QHD rozlišením bude sotva viditelné, o možnosti komfortního ovládání ani nemluvě. Navíc musíte brát v úvahu, že uživatelé mají různě silné prsty, a aplikaci navrhnout tak, aby ji mohli pohodlně ovládat všichni.

Musíte brát v úvahu i způsob zadávání textů. Většina zařízení používá virtuální klávesnice, na kterých se sice píše velmi pohodlně, ale zakrývají část obrazovky, v některých případech podstatnou část obrazovky, například v poloze zařízení „na šířku". Proto se snažte zadávání údajů přes klávesnici vyhnout. K zadávání údajů používejte prvky typu rozevírací seznam, seznam, kalendář a podobně. K zadávání číselných hodnot v přiměřeném rozsahu preferujte tlačítka na inkrementování a dekrementování hodnoty před přímým zadáváním.

Také je potřeba brát v úvahu to, jak většina uživatelů drží v ruce telefon a tablet, a od toho můžete odvodit nejpohodlnější polohu tlačítek, aby se na ně dalo dosáhnout palci, a polohu zobrazování údajů, aby si je uživatel nezakrýval. Pokud má aplikace několik tlačítek, první je vhodné umístit vertikálně zhruba uprostřed obrazovky a další tlačítka pod ním, aby se na ně dalo dosáhnout palcem. Na základě průzkumů se zjistilo, že uživatelé nejčastěji drží tablet oběma rukama a snaží se ovládat aplikace pomocí palců. V případě, že to není možné, drží tablet jednou rukou a aplikace ovládají prsty druhé.

Při rozmisťování je potřeba brát v úvahu doporučení, kam je vhodné umístit prvky, které se používají na ovládání dotykem, a kam prvky, které prezentují obsah. Na obrázku si všimněte světlé zóny optimálního umístění ovládacích prvků u levého a pravého dolního rohu. Nejčastěji se tablet bude používat tak, že ho držíte oběma rukama za dolní rohy a zmíněná zóna udává akční rádius palců obou rukou. Ovládací prvky musí mít odpovídající rozměry a měly by být rozmístěny tak, aby se aplikace dala dobře ovládat pomocí palců při obouručním držení tabletu, a to nejen v klidu, tedy když uživatel sedí v křesle, ale i při dynamických scénářích, například během chůze.



Obrázek 3.16: Doporučení pro ideální rozmístění prvků pro ovládání (vlevo) a pro čtení obsahu (vpravo)

Mobilní zařízení se používají jak na výšku, tak na šířku. Automatické natáčení může uživatel vypnout. Pokud tak neučiní, musíte to v aplikaci akceptovat a zařídit, aby uživatelské rozhraní aplikace bylo estetické a ergonomické i po změně orientace. Změna orientace se někdy využívá i ke změně funkcionality. Notoricky známý je příklad kalkulačky, která je na výšku v jednoduchém standardním režimu a v režimu na šířku se přepne do takzvaného vědeckého módu s více funkcemi.

Jelikož displej všech zařízení – ať už telefonů, nebo tabletů – je obdélníkový, při změně orientace je potřeba logicky nejen obraz „natvrdo" překlopit o 90 stupňů, ale také změnit rozložení jednotlivých ovládacích prvků.

Dosud jsme hovořili o prvcích ovládání prezentace obsahu. Pro celkové ovládání aplikace, například k přecházení mezi obrazovkami nebo na vnořování se do hierarchicky organizovaného obsahu, jsou k dispozici další doporučení. Nejdůležitější ovládací prvek je situovaný v levém horním rohu. Je to tlačítko se symbolem „hamburgerového" menu.

Pokud potřebujete uživateli poskytnout možnost zadávání nějakých údajů, případně ho na něco upozornit, tato akce by se měla odehrávat na místě obrazovky, kde jsou zobrazeny informace, které se k příslušné akci nějak váží. Například pokud uživatel pomocí tlačítka vyvolá nějakou aktivitu, informace vážící se k této aktivitě by se měly zobrazovat v blízkosti tlačítka, které je vyvolalo. Lokalizační pravidlo platí i pro chybová hlášení. Chybové hlášení by se mělo zobrazit tam, kde chyba vznikla.

KAPITOLA

5

Uživatelské rozhraní a události

Témata kapitoly:

- **Kontejnery na rozmístění prvků**
- **Příklad – Definice rozložení prvků**
- **Příklad – Spuštění jiné aktivity**
- **Ladění aplikace**

Každý prvek v uživatelském rozhraní aplikace pro Android je podtřídou třídy `View`, přesněji `android.view.View`. Android SDK obsahuje sadu předpřipravených prvků, které se nejčastěji používají k tvorbě uživatelského rozhraní. Typickým příkladem jsou prvky typu `TextView`, `Button`, `Checkbox`, `Progressbar` a mnohé další. Objekty (prvky) uživatelského rozhraní se často označují i jako widgety nebo komponenty. Kromě předdefinovaných standardních prvků, které jsou součástí Android SDK, můžete vytvářet vlastní komponenty. Můžete je vytvářet jako rozšíření existujících tříd nebo vytvořit komponenty úplně nové.

Každý prvek odvozený od třídy `View` zabírá v uživatelském rozhraní odpovídající obdélníkovou plochu. Prvek je zodpovědný za to, co se v tomto obdélníku zobrazuje, a také za to, aby adekvátně reagoval na události, ke kterým dochází v rámci této části obrazovky, například na dotykové události. Obrazovka uživatelského rozhraní se skládá z hierarchie prvků `View`. Na nejvyšší úrovni hierarchie je kořenový element a podřízené (nebo v tomto případě vnořené), prvky jsou umístěny na nižších větvích hierarchického stromu.

Pohled (`view`) může být i kompozitní, to znamená složený z několika pohledů. Takovéto pohledy jsou podtřídy třídy `Android ViewGroup` (`Android.view.ViewGroup`) a tato třída je zase podtřídou `View`. Dobrým příkladem takovéhoto pohledu je `RadioGroup`, prvek, který vizuálně i funkčně zapouzdřuje více objektů `RadioButton`. Je to přesná analogie reálného světa, protože prvek `RadioButton` je ekvivalentem jednoho z více tlačítek, která se na starých radiopřijímačích používala k přepínání vlnových rozsahů. V poloze „zapnuto“ mohlo být vždy pouze jedno ze skupiny tlačítek.

Ale vraťme se k prvkům uživatelského rozhraní aplikace pro Android. Z hlediska struktury kompozitních prvků se tyto skládají z jednoho nadřazeného pohledu odvozeného od třídy `ViewGroup`, který nazýváme také kontejner, nebo výstižněji

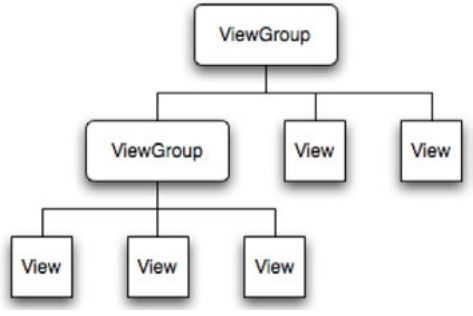
vizuální kontejner, jelikož v sobě zapouzdřuje prvky uživatelského rozhraní, případně tyto mohou být různě hierarchicky sdružené ve vnořených kontejnerech. Pro vnořené komponenty z hlediska syntaxe XML je vizuální kontejner kořenovým elementem.

Návrh uživatelského rozhraní

Základem tvorby interaktivního uživatelského rozhraní jsou pohledy, skupiny pohledů a aktivity.

- `Views` (pohledy) jsou základní třídy pro tvorbu vizuálních prvků uživatelského rozhraní známé i pod názvem widgety. Všechny ovládací prvky uživatelského rozhraní a třídy uspořádání jsou odvozeny od `Views`.
- `ViewGroups` jsou rozšířením pohledů pro tvorbu složených prvků, které, jak vyplývá z názvu, obsahují více `Views`.
- `Activities` (aktivity) jsou ekvivalentem formulářů nebo dialogových oken s ovládacími prvky, které znáte z jiných platforem. Je to nejmenší logická jednotka aplikace s uživatelským prostředím. Může obsahovat propojení na jiné aktivity.

Skupiny pohledů umožňují libovolné uspořádání hierarchie uživatelského rozhraní, takže na rozdíl od jiných platforem má vývojář při definování architektury volnou ruku. Při vytvoření projektu podle šablony **Blank Activity** si v souboru *fragment_main.xml* všimněte, že prvek `<TextView>` je umístěn uvnitř skupiny pohledů `<RelativeLayout>`.



Obrázek 5.1: Hierarchie objektů VievGroup

Nejčastěji se k návrhu uživatelského rozhraní využívají následující prvky:

- `TextView` – prvek umožní zobrazení textového výpisu, například hodnoty nějaké textové proměnné a podobně. Základním parametrem, který nastavujeme u tohoto prvku, je text.
- `EditText` – zatímco prvek `TextView` slouží k jednosměrné komunikaci od aplikace k uživateli, to znamená, že jen zobrazuje výpisy, `EditText` je jednoduché nebo víceřádkové editační okno pro zadávání textových údajů. Prvek podporuje i zarovnávání slov.
- `ImageView` – prvek na zobrazení obrázku.
- `Spinner` – kompozitní prvek na výběr více položek přizpůsobený výběru na malé ploše dotykového displeje telefonu. Seznam možností je možné rozbalit a některou z nich dotykem vybrat.
- `Button` – standardní tlačítko aktivované dotykem.

- `CheckBox` – tento dvojstavový ovládací prvek umožňuje přepínat mezi dvěma hodnotami – pravda/nepravda (true/false). Pokud je symbol prvku zaškrtnut, nabývá hodnotu true.
- `RadioButton` – ovládací prvek `RadioButton` sám o sobě velký smysl nemá, je potřeba použít těchto prvků několik, potom fungují jako přepínač (odtud analogie `RadioButton` – tlačítková sada na přepínání rozsahů rozhlasového přijímače).
- `ProgressBar` – prvek, pomocí něhož aplikace indikuje uživateli probíhající činnost. Pomocí parametru `android:indeterminate` určujete, zda se jedná o `ProgressBar` konečný, který ukazuje průběh v procentech, nebo nekonečný, kde nelze určit, v jakém stadiu dokončení se prováděná akce nachází.

Kontejnery na rozmístění prvků

Jak už bylo uvedeno, k definici rozmístění prvků uživatelského rozhraní slouží takzvané vizuální kontejnery, v původní terminologii `Layouts`. Jsou odvozeny od třídy `ViewGroup`.

Rozmístění prvků musí být flexibilní. Za všechny uvedeme jen dva důvody: různé rozměry obrazovek a změnu orientace zařízení.

Při pootočení zařízení z režimu „na výšku" do režimu „na šířku", případně naopak, je zpravidla zapotřebí nejen geometrická transformace, ale i změna rozmístění prvků, aby co nejlépe využily plochu obrazovky v daném režimu.

LinearLayout

Umožňuje uspořádat vnořené prvky (`views`) horizontálně do jednoho řádku nebo vertikálně do jednoho sloupce. Klíčové parametry pro `LinearLayout` jsou:

- `orientation` – směr uspořádání vnořených prvků.
- `layout_gravity` – takzvané těžiště vizuálního kontejneru, které definuje centrální bod zarovnání vnořených `view`.
- `Layout_weight` – priorita vnořeného `view` v rámci layoutu. Čím vyšší hodnota, tím vyšší priorita. Pokud máte v rámci layoutu tři prvky, přičemž dva z nich mají nastavený parametr `weight` na `0` a třetí na hodnotu `1`, dva prvky s nižší prioritou zaberou prostor daný jejich obsahem a třetí prvek s vyšší prioritou vyplní zbytek plochy layoutu.

V následujícím ilustračním příkladu je uživatelské rozhraní využívající `LinearLayout` s vertikální orientací sestávající ze tří tlačítek. První a třetí nemá nastavený parametr `weight`, což v praxi znamená, že je implicitně nastavený na hodnotu 0. Prostřední tlačítko má nastavený parametr `weight` na hodnotu `1`, takže má vyšší prioritu a zabere na výšku zbylou plochu. Parametr `gravity` je nastaven střídavě na hodnoty `left` a `right`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context="sk.pcrevue.llacko.linearlayoutapp.MainActivity">

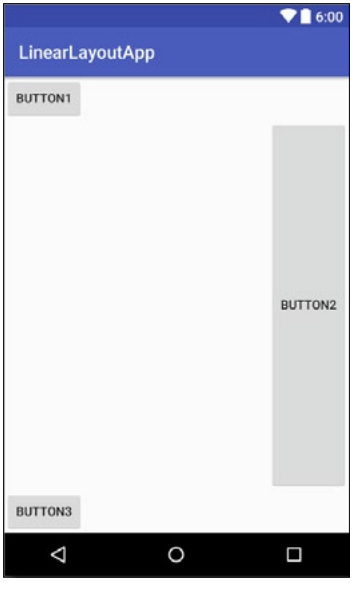
  <Button
```

```
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="left"
    android:text="Button1" />

  <Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="0dp"
    android:layout_gravity="right"
    android:layout_weight="1"
    android:text="Button2" />


  <Button
    android:id="@+id/button3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="left"
    android:text="Button3" />

</LinearLayout>
```



Obrázek 5.2: Ilustrace fungování parametrů grafity a weight

RelativeLayout

Relativní layouty umožňují definování vzájemných pozičních vztahů nejen mezi jednotlivými vizuálními prvky uživatelského rozhraní (views), ale i vůči rodičovskému layoutu. Aby bylo možné definovat vzájemné vazby, musí mít každý prvek jednoznačný identifikátor – atribut ID.

Příklady možností definování polohy:

- layout_above – vůči prvku se zadaným ID
- layout_alignLeft – vůči prvku se zadaným ID
- layout_alignParentLeft – true/false
- layout_centerHorizontal – true/false
- layout_toLeftOf – vůči prvku se zadaným ID

Prvky jsou implicitně umísťovány do levého horního rohu.

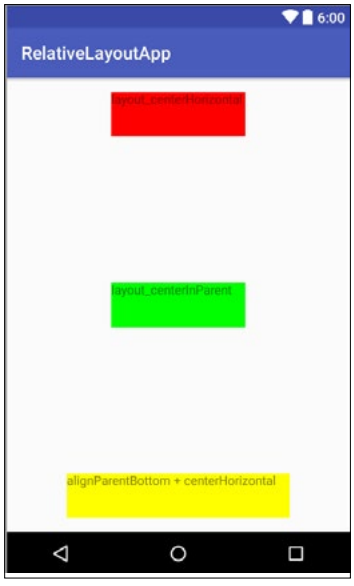
V ilustračním příkladu ukážeme nejprve určení polohy views vůči rodičovskému layoutu.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="sk.pcrevue.llacko.relativelayoutapp.MainActivity">

  <TextView
    android:layout_width="150dp"
    android:layout_height="50dp"
    android:layout_centerInParent="true"
    android:background="@color/green"
    android:text="layout_centerInParent" />

  <TextView
    android:layout_width="150dp"
    android:layout_height="50dp"
    android:layout_centerHorizontal="true"
    android:background="@color/red"
    android:text="layout_centerHorizontal" />

  <TextView
    android:layout_width="250dp"
    android:layout_height="50dp"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:background="@color/yellow"
    android:text="alignParentBottom + centerHorizontal" />
```



Obrázek 5.3: Relativní pozicování prvků vůči rodičovskému view a relativnímu layoutu

Pomocí atributů

- `android:layout_alignLeft`
- `android:layout_alignTop`
- `android:layout_alignRight`
- `android:layout_alignBottom`
- `android:layout_toLeftOf`
- `android:layout_above`
- `android:layout_toRightOf`
- `android:layout_below`

můžete definovat pozice prvků vůči jiným prvkům. V následujícím příkladu jsme jeden prvek ukotvili vůči rodičovskému `view` a druhé dva prvky jsou ukotveny relativně vůči tomuto prvku.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="sk.pcrevue.llacko.relativelayoutapp.MainActivity">

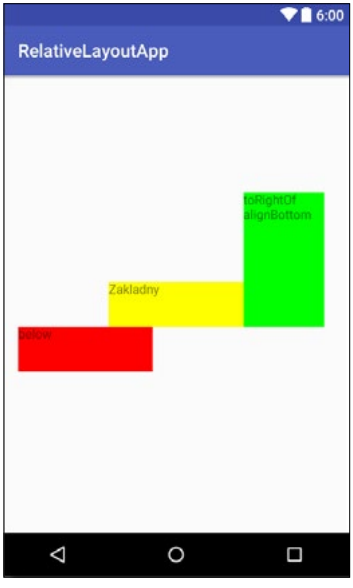
  <TextView
    android:id="@+id/zakladny"
    android:layout_width="150dp"
    android:layout_height="50dp"
```

```
    android:layout_centerInParent="true"
    android:background="@color/yellow"
    android:text="Zakladny"

  <TextView
    android:layout_width="150dp"
    android:layout_height="50dp"
    android:layout_below="@id/zakladny"
    android:background="@color/red"
    android:text="below" />

  <TextView
    android:layout_width="90dp"
    android:layout_height="150dp"
    android:layout_alignBottom="@id/zakladny"
    android:layout_toRightOf="@id/zakladny"
    android:background="@color/green"
    android:text="toRightOf\nalignBottom" />

</RelativeLayout>
```

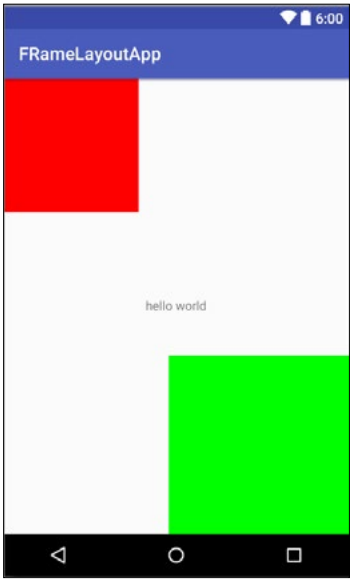


Obrázek 5.4: Relativní pozicování prvků vůči jiným prvkům

Relativní layouty jsou velmi flexibilní, ale pouze v případě, že definujete mezi prvky jen minimální počet vazeb. V opačném případě se stává návrh nepřehledným a vzniknou takzvané kruhové vazby a rozpory, například mezi definováním rozměrů prvků a okrajů vzhledem k rozměru rodičovského layoutu, který zabírá celou šířku obrazovky. Pokud se třeba součet šířky prvku a jeho levého a pravého okraje nerovná šířce rodičovského kontejneru, nastane problém.

FrameLayout

Tento layout umožňuje zobrazit jen jeden prvek (`view`). Využívá se hlavně k zobrazování dynamicky vytvářených fragmentů pro verze operačního systému starší než 3.0 a komplexnějších prvků, například `ScrollView`. Pokud byste do `FrameLayoutu` vložili více prvků, zobrazí se na tom samém místě, přičemž budou zarovnané podle levého horního rohu. Umístění prvků je možné ovlivňovat pomocí atributu `android:layout_gravity`.



Obrázek 5.5: Příklad rozmístění prvků s využitím kontejneru FrameLayout

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.framelayoutapp.MainActivity">

  <View
    android:layout_width="150dp"
    android:layout_height="150dp"
    android:background="@color/red" />

  <View
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:layout_gravity="bottom|right"
    android:background="@color/green" />

  <TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
```

```
    android:text="@string/hello_world" />

</FrameLayout>
```

TableLayout

Do verze API 14 (Android 4.0) se na organizování prvků uživatelského rozhraní do formy tabulek, tedy řádků a sloupců, využíval `TableLayout`. Ke konstrukci řádků tabulky se využívá kontejner `TableRow`. Při vytváření složitějších tabulek oceníte možnost roztáhnout pole přes několik buněk tabulky. Od verze 4.0 se mnohem více využívá intuitivní kontejner `GridLayout`.

V příkladu je tabulka denních maximálních a minimálních teplot. Kvůli zkrácení výpisu jsou odstraněné identifikátory jednotlivých buněk tabulky, text je vložen přímo, a nikoliv pomocí zdrojů jako v reálných aplikacích a pro každý řádek je uvedena jen jedna hodnota.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:collapseColumns=""
  android:shrinkColumns="*"
  android:stretchColumns="*"
  tools:context="sk.pcrevue.llacko.tablelayoutapp.MainActivity">

  <TableRow
    android:id="@+id/tableRowNadpis"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal" >

    <TextView
      android:id="@+id/textViewNadpis"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:padding="8dp"
      android:text="Denné maximá a minimá teplôt"
      android:textSize="13dp"
      android:textStyle="bold">
    </TextView>
  </TableRow>

  <TableRow
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <TextView
      android:text="" >
    </TextView>

    <TextView
      android:gravity="center"
```

```
        android:text="1.5">
      </TextView>

    ...
    </TableRow>

    <TableRow
      android:id="@+id/tableRowHttp"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <TextView
        android:id="@+id/textView10"
        android:text="MIN"
        android:textStyle="bold" >
      </TextView>

      <TextView
        android:id="@+id/textView11"
        android:gravity="center_horizontal"
        android:text="12" >
      </TextView>

    ...
    </TableRow>

    <TableRow
      android:id="@+id/tableRowHttps"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <TextView
        android:id="@+id/textView20"
        android:text="MAX"
        android:textStyle="bold" >
      </TextView>

      <TextView
        android:id="@+id/textView21"
        android:gravity="center_horizontal"
        android:text="26" >
      </TextView>

    ...
    </TableRow>

  </TableLayout >
```



Obrázek 5.6: Příklad rozmístění prvků s využitím kontejneru TableLayout. Všimněte si pomocného pravítka pro vizuální návrh tabulky v Android Studiu.

GridLayout

`GridLayout` byl přidán do API od verze API 14 (Android 4.0). Jak vyplývá názvu, grid je matice prvků, přičemž vývojář má plnou kontrolu nad tím, jak se tyto prvky zobrazí. Pozice prvku se definuje pomocí atributů `layout_width` a `layout_height`. Buňkám je možné nastavovat parametry `android:layout_columnSpan` a `android:layout_rowSpan`, které definují, přes kolik buněk tabulky má být prvek roztažen ve vodorovném nebo svislém směru.

V příkladu je použit `GridLayout` na rozmístění prvků jednoduché aktivity – formuláře na zadání jména a hesla. I v tomto kódu je text vložen přímo, a ne prostřednictvím zdrojů jako v reálných aplikacích.

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:columnOrderPreserved="false"
  android:columnCount="4"
  tools:context="sk.pcrevue.llacko.gridlayoutapp.MainActivity">

  <TextView
    android:text="Registrácia"
    android:textSize="32dip"
    android:layout_columnSpan="4"
    android:layout_gravity="center_horizontal"/>

  <TextView
    android:text="Prihláste sa na odber našich noviniek:"
    android:textSize="16dip"
    android:layout_columnSpan="4"
    android:layout_gravity="left"/>

  <TextView
    android:text="Email:"
```

```
    android:layout_gravity="right"/>

  <EditText
    android:ems="10"/>

  <TextView
    android:text="Heslo:"
    android:layout_column="0"
    android:layout_gravity="right"/>

  <EditText
    android:ems="8"/>
  <Space
    android:layout_row="4"
    android:layout_column="0"
    android:layout_columnSpan="3"
    android:layout_gravity="fill"/>

  <Button
    android:text="Pokračuj"
    android:layout_row="5"
    android:layout_column="3"/>

</GridLayout>
```



Obrázek 5.7: Příklad rozmístění prvků s využitím kontejneru GridLayout. Všimněte si pomocného pravítka pro vizuální návrh v Android Studiu.

AbsoluteLayout

Tento vizuální kontejner umožňuje umísťovat prvky na specifické souřadnice. Velkou nevýhodou je nulová flexibilita, používání tohoto kontejneru proto nedoporučujeme. Aplikace, která ho využívá, nedokáže flexibilně reagovat na změny velikosti a orientace obrazovky zařízení. Dá se použít jako vnořený v některém z flexibilních kontejnerů například na vytváření menších bloků uživatelského rozhraní, které budou mít v rámci bloku fixní polohu.

CoordinatorLayout

Tento kontejner byl zaveden od verze Android 5.0. Je navržen speciálně ke koordinaci vzhledu a fungování aplikační lišty v horní části obrazovky se zbytkem uživatelského rozhraní aplikace. Tento prvek je použit v šabloně projektu **Basic** (předtím **Blank**), kde můžete prozkoumat jeho návrh a fungování v návrhovém kódu aktivity v souboru *activity_main.xml*.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:fitsSystemWindows="true"
  tools:context="com.example.llacko.navrhui.MainActivity">

  <android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/AppTheme.AppBarOverlay">

    <android.support.v7.widget.Toolbar
      android:id="@+id/toolbar"
      android:layout_width="match_parent"
      android:layout_height="?attr/actionBarSize"
      android:background="?attr/colorPrimary"
      app:popupTheme="@style/AppTheme.PopupOverlay" />

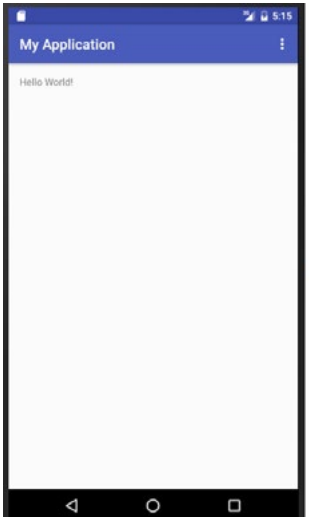
  </android.support.design.widget.AppBarLayout>

  <include layout="@layout/content_main" />

  <android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    android:src="@android:drawable/ic_dialog_email" />

</android.support.design.widget.CoordinatorLayout>
```

Kontejner `CoordinatorLayout` je v definičním souboru aktivity *activity_main.xml* a návrh obsahu pracovního okna pod aplikační lištou v souboru *content_main.xml*.



Obrázek 5.8: Kontejner CoordinatorLayout zapouzdřuje aplikační lištu, případně další ovládací prvky mimo pracovní okno

Vnoření kontejnerů

V dalším příkladu je kombinace dvou vnořených objektů `LinearLayout`, které jsou zapouzdřeny v kontejneru stejného typu. Úlohou vnějšího kontejneru je uspořádat prvky vertikálně. Vnořený kontejner `LinearLayout` udržuje tři tlačítka ve vodorovné poloze. Další vnořený kontejner udržuje dvě textová pole vertikálně.



Obrázek 5.9: Kombinace vnořených kontejnerů

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.vnorenylayoutapp.MainActivity">

  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">

    <Button
      android:text="B1"
      android:gravity="center_horizontal"
      android:layout_width="wrap_content"
      android:layout_height="fill_parent"
      android:layout_weight="1"/>

    <Button
      android:text="B2"
      android:gravity="center_horizontal"
      android:layout_width="wrap_content"
      android:layout_height="fill_parent"
      android:layout_weight="1"/>

    <Button
      android:text="B3"
      android:gravity="center_horizontal"
      android:layout_width="wrap_content"
      android:layout_height="fill_parent"
      android:layout_weight="1"/>

  </LinearLayout>

  <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">

    <TextView
      android:text="Text 1"  android:textSize="15pt"
      android:background="@color/red"
      android:layout_width="fill_parent" android:layout_height="wrap_content"
      android:layout_weight="1"/>

    <TextView
      android:text="Text 2"  android:textSize="15pt"
      android:background="@color/green"
      android:layout_width="fill_parent" android:layout_height="wrap_content"
```

```
        android:layout_weight="1"/>
    </LinearLayout>
</LinearLayout>
```

ScrollView

Obrazovky zařízení v závislosti na rozměrech a rozlišovací schopnosti umožňují současně zobrazit jen určité množství informací. Často však dokumenty, obrázky, webové stránky a jiný kontext potřebují zabrat větší plochu, aby byly čitelně zobrazené. Řešením je prvek `ScrollView`, umožňující svislé rolování, případně prvek `HorizontalScrollView`, umožňující vodorovné rolování, pokud velikost vnořených prvků přesahuje fyzickou velikost displeje. Plocha obrazovky zařízení představuje jakési virtuální okno, které je možné posouvat, a tak postupně po částech zpřístupnit celý obsah. Z hlediska hierarchie je `ScrollView` odvozen od prvku `FrameLayout`. Může zapouzdřovat jeden vnořený prvek.

Pokud potřebujete zobrazit `ScrollView` v konkrétní poloze, tuto nastavíte pomocí parametrů `android:scrollY`, respektive `android:scrollX`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context="sk.pcrevue.llacko.scrollviewapp.MainActivity">

  <ScrollView
    android:id="@+id/scroll_view_1"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1" >

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/text_1" />

  </ScrollView>
  <ImageView
    android:id="@+id/imageView1"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"
    android:contentDescription="@string/text_3"
    android:src="@mipmap/obrazok" />

  <ScrollView
    android:id="@+id/scroll_view_3"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="0.5" >

    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/text_2" />

    </ScrollView>
</LinearLayout>
```



Obrázek 5.10: Příklad použití dvou prvků typu ScrollView

Příklad – Definice rozložení prvků

Námětem příkladu je jednoduchá aplikace, která bude simulovat posílání textové zprávy. Začneme rozmístěním prvků a interakcí mezi dvěma nejpoužívanějšími prvky – dotykovým tlačítkem, pomocí kterého uživatel dá aplikaci impulz, a prvkem pro zadání, případně výpis textu. Po stisknutí tlačítka aplikace provede naprogramované úkony.

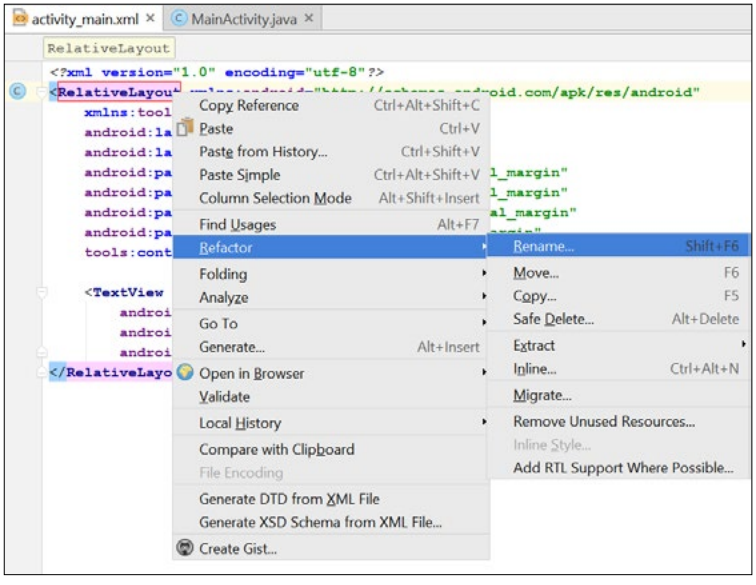
Pravděpodobně se po přečtení zadání chystáte ve vizuálním návrhovém prostředí Android Studia pomocí pár klepnutí přidat na plochu aplikace prvky `EditText` *a* `Button`*. Toto je správný a nejrychlejší postup, z hlediska metodiky ho ale doporučujeme až v dalších příkladech. V prvním příkladu je z hlediska pochopení souvislostí mnohem lepší návrh editováním kódu XML.*

Vizuální návrh uživatelského rozhraní se realizuje nad souborem */res/layout/aktivity_main.xml*. V projektu vytvořeném s využitím šablony **Empty Activity** je k zapouzdření vizuálních prvků uživatelského rozhraní využita skupina pohledů `RelativeLayout`. Tento typ skupiny pohledů umožňuje definování uspořádání polohy dceřiných prvků vzájemně vůči sobě nebo vůči rodičovskému prvku.

Aby byl návrh přímým psaním kódu co nejjednodušší, změňte typ skupiny pohledů `RelativeLayout` na `LinearLayout`. Prvky ve skupině pohledů jsou uspořádány do jednoho sloupce nebo řádku. Směr řádku lze nastavit voláním metody `setOrientation()`. Můžete také určit „virtuální gravitaci“, která určuje zarovnání všech podřízených prvků voláním `setGravity()`, nebo určit, že

konkrétní prvky mohou zaplnit zbývající prostor na rozložení nastavením parametru hmotnosti `LinearLayout.LayoutParams`. Předvolená orientace je vodorovná. Každý dceřiný prvek z `LinearLayout` se objeví na obrazovce aplikace v pořadí, v jakém byl definován v XML. Parametr `weight` definuje „váhu" prvku v příslušné skupině pohledů.

Změnu typu skupiny pohledů z `RelativeLayout` na `LinearLayout` nejjednodušeji provedete pomocí refaktorování. V editoru návrhového kódu XML vyberte počáteční tag `RelativeLayout` a v místní nabídce zvolte **Refactor** → **Rename**. Když budete přepisovat počáteční tag, bude se automaticky přepisovat i koncový.



Obrázek 5.11: Změna typu skupiny pohledů z RelativeLayout na LinearLayout pomocí refaktorování

V návrhovém kódu vymažte prvek `<TextView>` a skupinu pohledů `<RelativeLayout>` nahraďte skupinou pohledů `<LinearLayout>`. Atribut `android:orientation` nastavte na hodnotu `"horizontal"`.

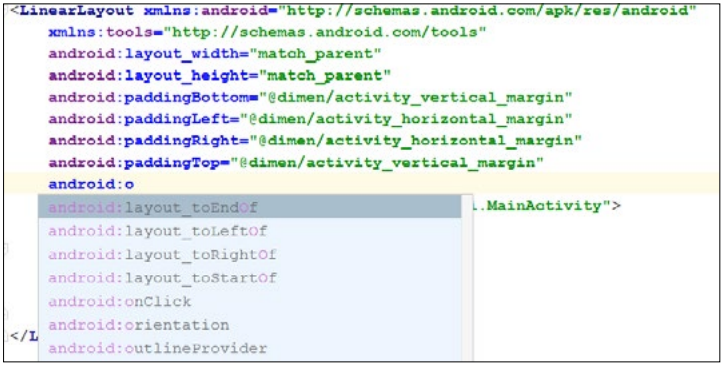
Všimněte si, jak Android Studio pomáhá minimalizovat psaní kódu. Nabízí vám na dokončení nejpravděpodobnější slova. Když začnete psát „an", nabídne vám vývojové prostředí mezi volbami slovo „android:". Volbu potvrďte klávesou **Tab** a začněte psát slovo „orientation". Po napsání písmena „o" je vám toto slovo nabídnuto, vyberte ho a potvrďte klávesou **Tab**. Nakonec zbývá jen vybrat hodnotu parametru ze dvou nabízených možností „horizontal" a „vertical".

Je potřeba nastavit i parametry `android:layout_width` a `android:layout_height`, aby bylo možné určit velikost pro zobrazení. V tomto případě se nastaví oba parametry na hodnotu `"match_parent"`, to znamená, že dceřiný prvek přebere rozměry rodičovského prvku.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:orientation="horizontal"
    tools:context="com.example.llacko.poslispravu.MainActivity">

    ...
</LinearLayout>
```



Obrázek 5.12: Nabídka možností dokončení kódu

POZNÁMKA

`LinearLayout` je kořenový prvek, proto zaplní celou plochu obrazovky.

Uživatelské rozhraní jednoduché aplikace bude sestávat ze dvou prvků. Prvek `<EditText>` bude sloužit k zadání textu a tlačítko, tedy prvek `<Button>`, na vyvolání akce, v tomto případě na vypsání oznámení; v reálné aplikaci by se jím odeslala zpráva.

```
<EditText
  android:id="@+id/etVstup"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginLeft="20dp"
  android:hint="@string/text_vypisu">
  <requestFocus />
</EditText>

<Button
  android:id="@+id/btVypis"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/text_tlacidlaOK" />
```

Parametr `android:hint` obsahuje textový řetězec, který se zobrazí, když je textové pole prázdné. Můžete sice použít přímo textový řetězec, takovéto zjednodušení by vám ale později pravděpodobně přineslo více problémů než užitku. Mnohem systematičtější je použití reference na hodnotu řetězce ve zdrojích v souboru *res/values/strings.xml*.

```
<resources>
  <string name="app_name">VypisOznam</string>
  <string name="text_vypisu">Napiš oznam</string>
  <string name="text_tlacidlaOK">Pošli</string>
</resources>
```

Parametr `android:id` je jedinečný identifikátor prvku vytvořený v množině identifikátorů. Na tento identifikátor se může odkazovat váš aplikační kód, například na změnu některého parametru, či načítání hodnoty a podobně. Znak `@` udává, že se jedná o referenci. Za referencí je typ prostředku (resource), v tomto případě id, a za lomítkem je jméno identifikátoru.

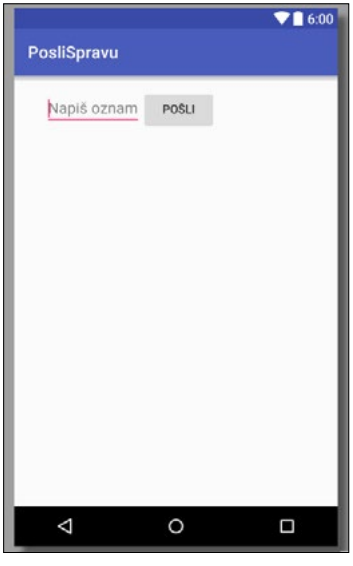
POZNÁMKA

Při prvním definování ID prostředku je potřeba před typ prostředku dát znaménko plus (+). To je signálem pro SDK a kompilátor, aby vytvořily nový prostředek. Další odkazy na ID znaménko plus už nepotřebují.

Parametry v atributech `android:layout_width` a `android:layout_height` udávají šířku a výšku prvku. Hodnota `"wrap_content"` udává, že rozměr by měl být takový, aby se prvek vešel do rodičovského pohledu. Pokud byste použili hodnotu parametru `"match_parent"`, prvek `EditText` by vyplnil celý rodičovský prvek `LinearLayout`, v tomto případě tedy celou obrazovku aplikace.

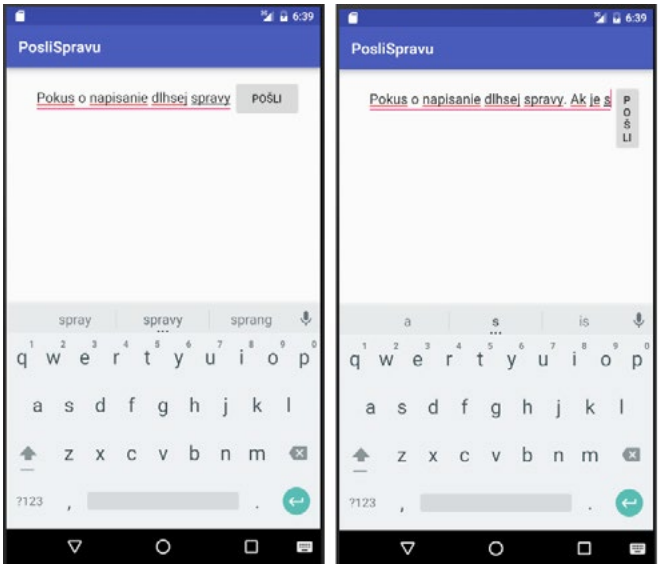
Atribut `android:onClick` definuje metodu, která se spustí po klepnutí na tlačítko.

Po přepnutí se na kartu **Graphical Layout** vidíte vizuální reprezentaci prvků uživatelského rozhraní, které jste přidali do aplikace.



Obrázek 5.13: Uživatelské rozhraní v návrhovém zobrazení

V této fázi, dříve než se dostaneme k podrobnostem ohledně vzájemného rozmístění prvků, by bylo zajímavé aplikaci otestovat. Zatím jsme nedefinovali akci po stisknutí tlačítka, ale to nevadí, budeme zkoumat zobrazení prvků během psaní delšího textu do prvku `EditText`. Všimněte si, co se děje s tlačítkem během postupného zvětšování prvku `EditText`.



Obrázek 5.14: Test uživatelského rozhraní, psaní delší textové zprávy

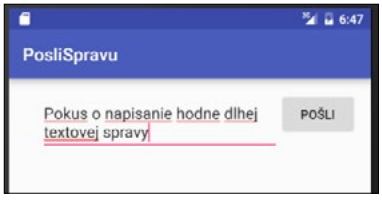
Chování prvku je možné ovlivnit pomocí parametru `android:layout_weight`. Hodnota parametru udává množství prostoru, který prvek zabírá. Rozdělení prostoru je v relativním poměru. Pokud například máte dva prvky, prvnímu zadáte váhu `2` a druhému váhu `1`, prvky si rozdělí prostor v daném směru tak, že první z nich zabere 2/3 a druhý vyplní zbytek, tedy 1/3.

Pokud do pohledu vložíte tři prvky a jednomu z nich nastavíte atribut `weight` na `1`, bude tento mít vyšší váhu než zbylé dva, které mají implicitně zadanou nulu. Tyto dva prvky tedy budou mít plochu danou jejich obsahem a prvek s váhou 1 zabere zbylou plochu pohledu. Nejlépe to pochopíte na následujícím příkladu.

Předvolená hodnota parametru `layout_weight` je `0`, takže pokud pro sousední prvek zadáte nějakou hodnotu větší než `0`, zabere tento prvek zbytek prostoru. V našem příkladu ponechejte prvku `Button` implicitní hodnotu `0` a pro prvek `EditText` změňte hodnotu parametru `layout_weight` na hodnotu `1`. Nyní můžete znovu vyzkoušet chování prvků během zadávání delší zprávy. Pokud pro prvek zadáte parametr `layout_weight`, měli byste změnit šířku prvku na nulu (`0DP`). Systém potom vypočítá relevantní šířku, aby prvek definovaným způsobem vyplnil zbývající prostor.

Upravte definici prvku `EditText`:

```
<EditText
  android:id="@+id/etVstup"
  android:layout_width="0dp"
  android:layout_height="wrap_content"
  android:layout_marginLeft="20dp"
  android:layout_weight="1"
  android:hint="@string/text_vypisu">
  <requestFocus />
</EditText>
```



Obrázek 5.15: Test uživatelského rozhraní, psaní delší textové zprávy po úpravě

Kompletní kód XML návrhu aktivity bude:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${packageName}.${activityClass}" >

  <EditText
    android:id="@+id/etVstup"
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginLeft="20dp"
    android:hint="@string/text_vypisu">
    <requestFocus />
  </EditText>

  <Button
    android:id="@+id/btVypis"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="35dp"
    android:onClick="onClick"
    android:text="@string/text_tlacidlaOK" />

</LinearLayout >
```

Obsluha události

Na ploše hlavní aktivity aplikace (`MainActivity`) je pole na zadání textu a tlačítko, pomocí kterého na podnět uživatele aktivita provede požadovanou činnost. Do elementu kódu XML definice tlačítka jsme přidali následující řádek:

```
android:onClick="onClick"
```

Ten definuje, co se stane po klepnutí na tlačítko. Jelikož je Android neodmyslitelně spojený se zařízeními s dotykovým displejem, aktivace se fyzicky ve většině případů provede klepnutím na tlačítko, u některých specifických zařízení klepnutím elektronickým perem nebo na tabletech i myší.

```
<Button
  android:id="@+id/btVypis"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:onClick="onClick"
  android:text="@string/text_TlacidlaOK" />
```

Hodnota atributu `android:onClick` definuje metodu v aktuální aktivitě (přesněji její kód), kterou systém zavolá po klepnutí na tlačítko. V našem případě se bude jednat o metodu `onClick`.

Potřebujeme zobrazit ne pevně definovaný text, ale textový řetězec zadaný uživatelem v prvku `EditText`. Tento prvek typu `View` najdete pomocí metody `findViewById(R.id.etVstup)`.

Výraz v závorce je takzvaný identifikátor. Identifikátory na Androidu jsou statickými vlastnostmi bázové třídy `R`. Tato třída je generovaná automaticky na základě obsahu složky *res*. Pomocí třídy `R` je možné odkazovat i na jiné soubory XML. Například `R.layout.vystup_dialog` odkazuje na soubor *vystup_dialog.xml* ve složce */res/layout*.

Tělo metody bude v souboru *MainActivity.Java*:

```
public void onClick(View view)
{
  EditText input = (EditText) findViewById(R.id.etVstup);
  String string = input.getText().toString();
  Toast.makeText(this, string, Toast.LENGTH_LONG).show();
}
```

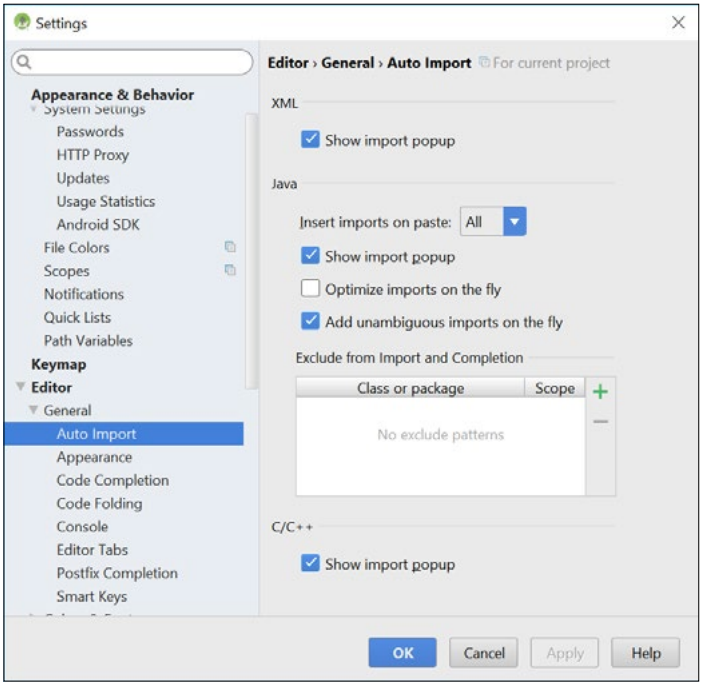
UPOZORNĚNÍ

Metody na obsluhu událostí musí být typu `public void`.

Na výpis použijeme prvek `Toast`. Pojmenování je celkem příznačné. Rámeček s textem oznámení vyskočí podobně jako toast z toustovače a po krátké době se přestane zobrazovat. Čas zobrazení se definuje parametrem `LENGTH_SHORT` nebo `LENGTH_LONG`.

Doporučujeme nastavit Android Studio tak, aby (samozřejmě pokud se dá) doplňovalo chybějící importy. Pomocí nabídky **File** → **Settings** → **Editor** → **General** → **Auto Import** → **Java** zobrazíte dialog, ve kterém označíte aktivní volby:

- **Insert imports on paste**
- **Add unambigious imports on the fly**



Obrázek 5.16: Nastavení doplňování chybějících importů



Obrázek 5.17: Test metody na obsluhu události – výpis oznámení pomocí toastové notifikace

Obsluha událostí ve vizuálním návrhu

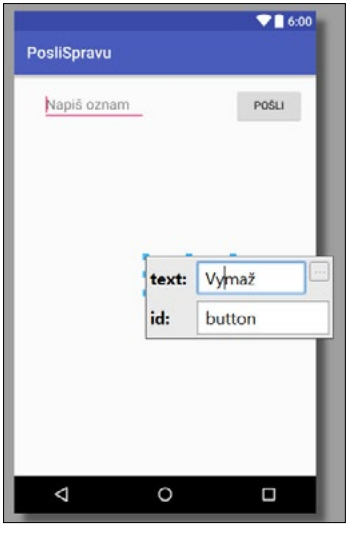
Dosud jsme v příkladu realizovali návrh uživatelského rozhraní přímým psaním kódu XML. Na tomto místě je vhodná příležitost ukázat, jak přiřadíme obsluhu události k prvku navrženému ve vizuálním návrhovém prostředí.

Přidejte na plochu aplikace další tlačítko, ale tentokrát pomocí vizuálního návrhu. Přidejte tlačítko přesunutím z okna **Palette** na vhodné místo plochy aplikace. Funkce tlačítka může být například vymazat napsaný text z prvku `EditText`, takže na něm bude text `"Vymaž"`.

POZNÁMKA

Zjistíte, že přidání tlačítka není jednoduché a nemůžete ho umístit kamkoliv. To je daň za použití jednoduchého layoutu `LinearLayout`.

Někam se vám nakonec tlačítko přeci jen umístit podaří. Bude to vertikálně uprostřed plochy a horizontálně mezi prvkem `EditText` a prvním tlačítkem. Polohu nyní nezkoumáme, cílem bylo jen ukázat, jak někdy „zaplatíme“ za počáteční zjednodušení. Poklepáním na tlačítko zobrazte operativní kontextový dialog na změnu parametrů a změňte text na tlačítku a prvku ID na nějaký výstižnější. Tentokrát se pro jednoduchost spokojíme s přímým vložením textového řetězce do návrhu prvku.



Obrázek 5.18: Nastavení hlavních parametrů prvku

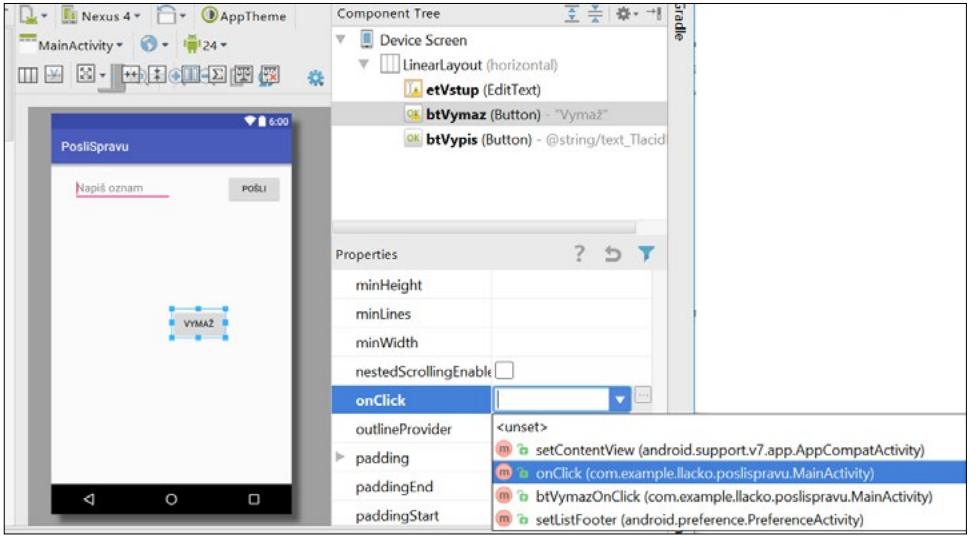
Dosavadním vizuálním návrhem vznikl následující kód:

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Vymaž"
  android:id="@+id/btVymaz"
  android:layout_gravity="center_vertical" />
```

Nejprve přidáme do kódu v souboru *MainActivity.java* prázdné tělo metody na obsluhu události.

```
public void btVymazOnClick(View v) {
  // TO DO
}
```

Přestože se shoduje část názvu, konkrétně námi přidané tlačítko není s touto metodou nijak propojené a po stisknutí tlačítka by se tato a ani žádná jiná metoda nezavolala. V návrhovém zobrazení v okně **Properties** v seznamu vlastností najděte **btVymazOnClick**, zobrazte výběr metod a vyberte metodu, ve které bude obsluha události.



Obrázek 5.19: Nastavení parametru onClick

Změna se samozřejmě projevila i v návrhovém kódu a nyní je už tlačítko propojené s metodou na obsluhu události:

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Vymaž"
  android:id="@+id/btVymaz"
  android:layout_gravity="center_vertical"
  android:onClick="onClick" />
```

Stačí doplnit kód do těla obslužné metody a vyzkoušet fungování aplikace:

```
public void btVymazOnClick(View v) {
  EditText input = (EditText) findViewById(R.id.etVstup);
  input.setText("");
}
```

Příklad – Spuštění jiné aktivity

Námětem aplikace bude výpočet indexu tělesné hmotnosti BMI na základě zadané výšky a hmotnosti osoby. Vzorec pro výpočet je

BMI hodnota = hmotnost v kg / (výška v m)^2

Vytvořte projekt s hlavní aktivitou typu **Empty Activity**. V hlavní aktivitě bude několik pohledů, tedy ovládacích prvků. K zobrazení popisů polí na zadání textu jsou v příkladu použity prvky typu `TextView`. Kromě typu hodnoty naznačíme uživateli i to, v jakých jednotkách aplikace příslušnou hodnotu očekává. Uživatel bude interagovat se dvěma prvky typu `EditText` na zadání výšky a váhy a zadání potvrdí tlačítkem, tedy prvkem typu `Button`.

K rozmístění prvků v souboru *activity_main* využijete vizuální kontejner `LinearLayout`, tentokrát s vertikálním rozmístěním prvků. Jelikož bude uživatel zadávat jen čísla, nakonfigurujeme prvek `EditText` tak, aby bylo prostřednictvím něj možné zadat jen textový řetězec sestávající z číslic. Zařídí to parametr `android:inputType="number"`.

Kompletní návrhový kód v souboru *activity_main.xml* bude:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:orientation="vertical"
  tools:context="com.example.llacko.vypocetbmi.MainActivity">

  <TextView
    android:text="@string/et_labelVyska"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

  <EditText
    android:id="@+id/etVyska"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="number" />

  <TextView
    android:text="@string/et_labelVaha"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

  <EditText
    android:id="@+id/etVaha"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="8dp"
```

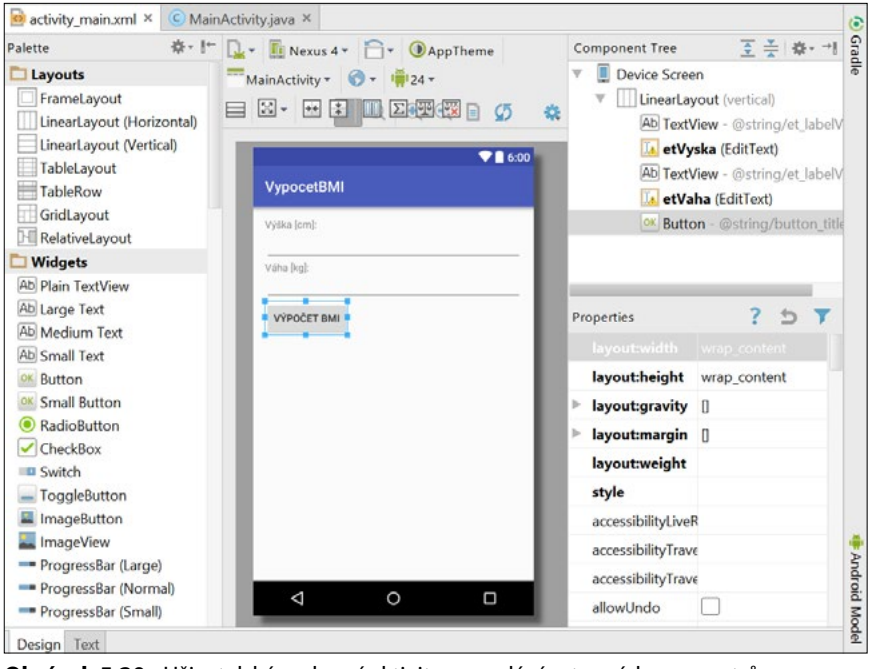
```
    android:inputType="number" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="buttonClicked"
    android:text="@string/button_title" />

</LinearLayout>
```

Tento příklad už považujeme za první reálnou aplikaci, proto nebudeme vkládat textové řetězce přímo do prvků typu `TextView`, ale do zdrojů, tedy do souboru */res/values/strings.xml*.

```
<resources>
  <string name="app_name">VypocetBMI</string>
  <string name="button_title">Výpočet BMI</string>
  <string name="et_labelVyska">Výška [cm]:</string>
  <string name="et_labelVaha">Váha [kg]:</string>
  <string name="nespravne_cislo">Musíš zadať číselnú hodnotu</string>
  <string name="zobraz_bmi">Tvoje BMI je %1$.2f.</string>
</resources>
```



Obrázek 5.20: Uživatelské rozhraní aktivity na zadání vstupních parametrů

Aplikační logika

Obsluha události stisknutí tlačítka se odehrává v metodě `buttonClicked`. V kódu metody je nejprve potřeba zajistit a zkontrolovat zadané údaje. V našem případě je tato kontrola na první

pohled duplicitní, jelikož uživatel může zadat jen číselné hodnoty. Uživatel je však nevyzpytatelný a může zadat například záporné číslo nebo jinou problematickou hodnotu, třeba nulovou výšku, která následně způsobí dělení nulou. Proto aplikace považuje za platný údaj výšku nad 50 cm.

```
public void buttonClicked(View button)
{
  int vyska = dajCislo(R.id.etVyska);
  int vaha = dajCislo(R.id.etVaha);
  if(vyska < 50)oznamNespravneCislo(R.id.etVyska);
  else if(vaha < 0)oznamNespravneCislo(R.id.etVaha);
  else vypocetBMI(vyska, vaha);
}
```

V reálné aplikaci byste uživateli oznámili více detailů, proč je jím zadaný údaj neplatný. V této aplikaci oznámíme pouze to, který údaj je neplatný, zadaný údaj se vynuluje a aplikace na něj přenese fokus, a uživatel může začít zadávat údaj nanovo.

```
protected void oznamNespravneCislo(int nID)
{
  Toast.makeText(this, R.string.nespravne_cislo,
    Toast.LENGTH_SHORT).show();
  EditText etVstup = ((EditText)findViewById(nID));
  etVstup.setText("");
  etVstup.requestFocus();
}
```

Uživatel de facto zadává textový řetězec, který je potřeba konvertovat na číslo. To je úlohou metody `dajCislo`. Na zjištění instance prvků `EditView` se použije metoda `findViewById(int id)`. Parametr je identifikátor požadovaného prvku. Text následně konvertujeme na celé číslo. Pokud se konverze nezdaří, vrátí metoda hodnotu -1, ta se následně v metodě `buttonClicked` identifikuje jako nesprávná a vygeneruje se oznámení.

```
protected int dajCislo(int nID)
{
  EditText etVstup = ((EditText)findViewById(nID));
  String strVstup = etVstup.getText().toString();
  int nCislo = -1;
  try
  {
    nCislo = Integer.parseInt(strVstup);
  }
  catch(NumberFormatException e){}
  return nCislo;
}
```

Ke komunikaci mezi entitami aplikací se využívá `Intent`. `Intent` (záměr) je třída, která obsahuje vazby runtime mezi jednotlivými složkami, například aktivitami a argumenty na odevzdání parametrů. Záměr se odevzdá operačnímu systému a ten se snaží zařídit, aby byl záměr splněn. V případě, že je k dispozici více aktivit schopných splnit záměr, systém umožní volbu aktivity uživateli. Metoda `putExtra` přebírá dva argumenty: textový řetězec s unikátním identifikátorem objektu na odevzdání parametru a samotný objekt, který může tvořit elementární hodnota nebo vhodný serializovatelný objekt. V tomto případě budete odevzdávat hodnotu BMI prostřednictvím proměnné typu `float`, jelikož BMI je desetinné číslo.

```
protected void vypocetBMI(int vyska, int vaha)
{
  float bmi;
  float vyskam = (float)vyska/100;
  bmi = vaha/(vyskam * vyskam);
  Intent intent = new Intent(this, VypisBmiActivity.class);
  intent.putExtra(VypisBmiActivity.BMI, bmi);
  startActivity(intent);
}
```

Kompletní zdrojový kód třídy hlavní aktivity aplikace, tedy té, kde se zadávají a zpracovávají údaje, bude:

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  protected int dajCislo(int nID)
  {
    EditText etVstup = ((EditText)findViewById(nID));
    String strVstup = etVstup.getText().toString();
    int nCislo = -1;
    try
    {
      nCislo = Integer.parseInt(strVstup);
    }
    catch(NumberFormatException e){}
    return nCislo;
  }

  protected void oznamNespravneCislo(int nID)
  {
    Toast.makeText(this, R.string.nespravne_cislo,
      Toast.LENGTH_SHORT).show();
    EditText etVstup = ((EditText)findViewById(nID));
    etVstup.setText("");
    etVstup.requestFocus();
  }

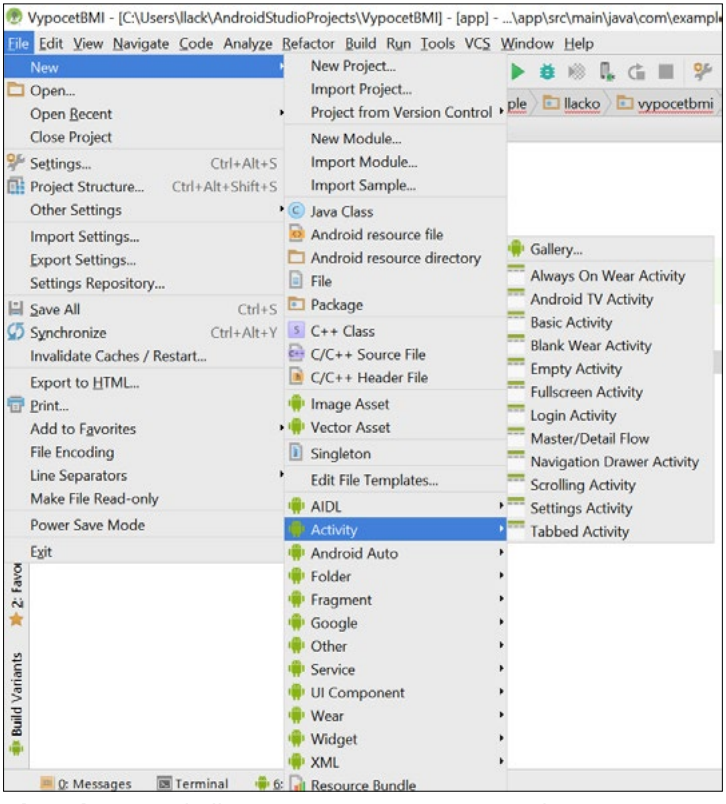
  protected void vypocetBMI(int vyska, int vaha)
  {
    float bmi;
    float vyskam = (float)vyska/100;
    bmi = vaha/(vyskam * vyskam);
```

```
    Intent intent = new Intent(this, VypisBmiActivity.class);
    intent.putExtra(VypisBmiActivity.BMI, bmi);
    startActivity(intent);
  }

  public void buttonClicked(View button)
  {
    int vyska = dajCislo(R.id.etVyska);
    int vaha = dajCislo(R.id.etVaha);
    if(vyska < 50)oznamNespravneCislo(R.id.etVyska);
    else if(vaha < 0)oznamNespravneCislo(R.id.etVaha);
    else vypocetBMI(vyska, vaha);
  }
}
```

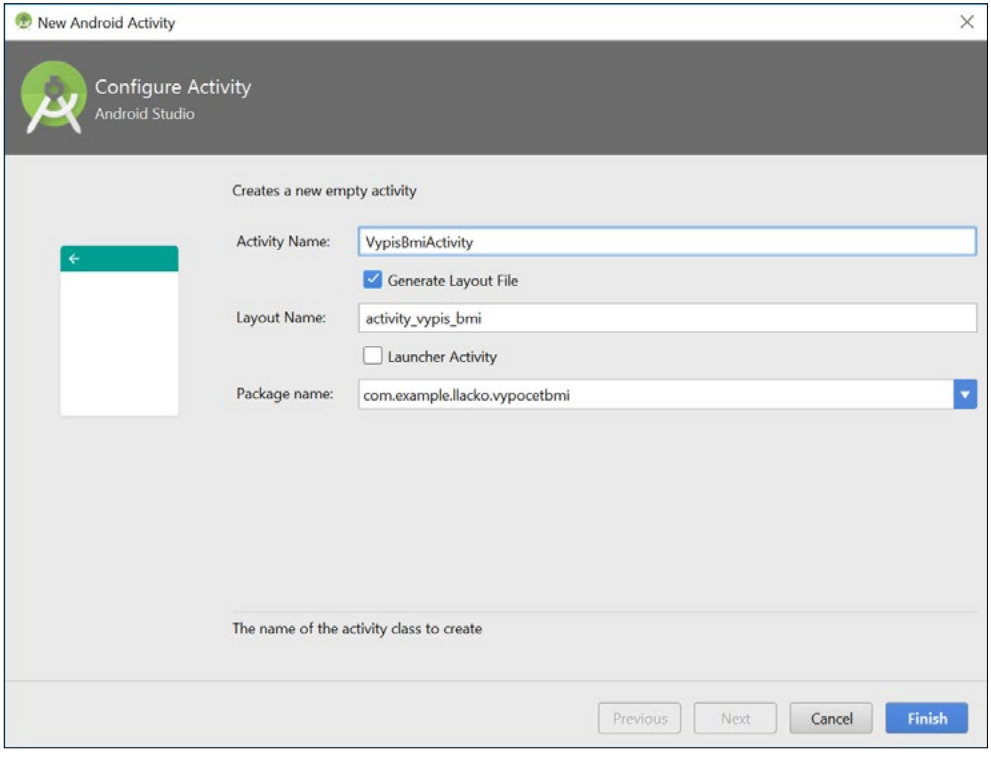
Vytvoření nové aktivity

Následně je potřeba vytvořit novou aktivitu na výpis vypočítaného BMI. K vytvoření aktivity, která obsahuje i definiční soubor XML uživatelského rozhraní, můžete s výhodou použít průvodce vytvořeními aktivity. Spustíte ho přes nabídku **New** → **Activity** → **Empty Activity**. K vytvoření nové aktivity je k dispozici široká nabídka šablon.



Obrázek 5.21: Nabídka možností na vytvoření nové aktivity

Aktivitu můžete nazvat například `VypisBmiActivity`. Od názvu aktivity se odvodí název layoutu, tedy souboru s kódem jejího uživatelského rozhraní. Aktivita bude potřebovat tento definiční soubor ve složce *res*, proto se ujistěte, zda je označena volba **Generate Layout File**.



Obrázek 5.22: Průvodce k vytvoření nové aktivity

Zdrojový kód třídy nové aktivity *VypocetBmiActivity.java* najdete ve složce *src* ve vnořené složce *com.example.nazevprojektu*:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class VypisBmiActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_vypis_bmi);
  }
}
```

Soubor na návrh uživatelského rozhraní *activity_vypis_bmi.xml* vytvořený průvodcem v této aplikaci nevyužijeme. Prvky na výpis vytvoříme jako objekty v kódu.

Informace o vytvořené aktivitě se přidají i do aplikačního manifestu v souboru *AndroidManifest.xml*, který najdete ve složce **app** → **manifests**.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.llacko.vypocetbmi">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".VypisBmiActivity"></activity>
  </application>

</manifest>
```

POZNÁMKA

Pokud jste aktivitu vytvářeli přímo jako třídu, musíte ji přidat do seznamu aktivit v manifestu Android v souboru *AndroidManifest.xml*. Pokud jste využili průvodce, přidání aktivity do manifestu Android zařídí průvodce.

Metoda `Activity.startActivity` odevzdá systému příslušný záměr a oznámí mu, že má spustit nějakou aktivitu. Tím dějství v hlavní aktivitě končí a na popředí vystoupí aktivita `VypisBmiActivity`.

Začněte tvořit kód třídy nové aktivity. V hlavní aktivitě jsme v metodě `vypocetBMI()` odkazovali na textový řetězec s unikátním identifikátorem objektu na odevzdání parametru BMI druhé aktivitě. Připomeňme si kód této metody:

```
protected void vypocetBMI(int vyska, int vaha)

  float bmi;
  float vyskam = (float)vyska/100;
  bmi = vaha/(vyskam * vyskam);
  Intent intent = new Intent(this, VypisBmiActivity.class);
  intent.putExtra(VypisBmiActivity.BMI, bmi);
  startActivity(intent);
```

Proto musíme v kódu aktivity `VypisBmiActivity` tento konstantní řetězec vytvořit.

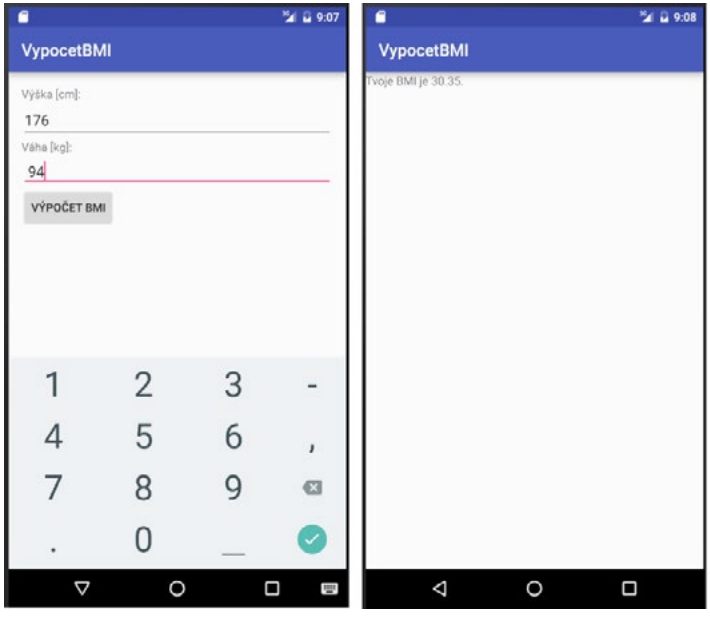
```
public static final String BMI = "bmi";
```

V metodě `onCreate` přebere aktivita `Intent` vytvořený hlavní aktivitou. Pomocí metody `getFloatExtra` získáme hodnotu BMI. Druhý parametr obsahuje implicitní hodnotu, pokud by se vlastnost s daným klíčem v záměru nenacházela.

```
Intent i = getIntent();
float bmi = i.getFloatExtra(BMI, -1);
```

V metodě `zobrazBMI` zobrazí aktivita hodnotu BMI jako součást textového řetězce. Všimněte si v souboru */res/values/strings.xml* řetězce obsahujícího formátovací sekvenci `%1$.2f`, která předepisuje, že první parametr se zobrazí jako desetinné číslo (f – float) s přesností na dvě desetinná místa (.2).

```
<string name="zobraz_bmi">Tvoje BMI je %1$.2f.</string>
```



Obrázek 5.23: Test aplikace – vlevo je hlavní aktivita zobrazená po spuštění aplikace určená k zadání parametrů na výpočet BMI, vpravo je aktivita, která vypíše vypočítané BMI

Kompletní zdrojový kód třídy nové aktivity:

```
public class VypisBmiActivity extends Activity
{
  public static final String BMI = "bmi";

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    TextView tv = new TextView(this);
    setContentView(tv);
    Intent i = getIntent();
    float bmi = i.getFloatExtra(BMI, -1);
    zobrazBMI(bmi, tv);
  }

  protected void zobrazBMI(float bmi, TextView tv)
```

```
    {
      Resources res = getResources();
      String text = res.getString(R.string.zobraz_bmi, bmi);
      tv.setText(text);
    }
  }
```

Nyní můžete aplikaci vyzkoušet.

Samozřejmě můžete aktivitu na výpis vylepšit. Namísto suchého konstatování hodnoty můžete třeba poskytnout komentář, zda se jedná o správnou hmotnost, případně obezitu, zobrazit hodnotu na grafu a podobně.

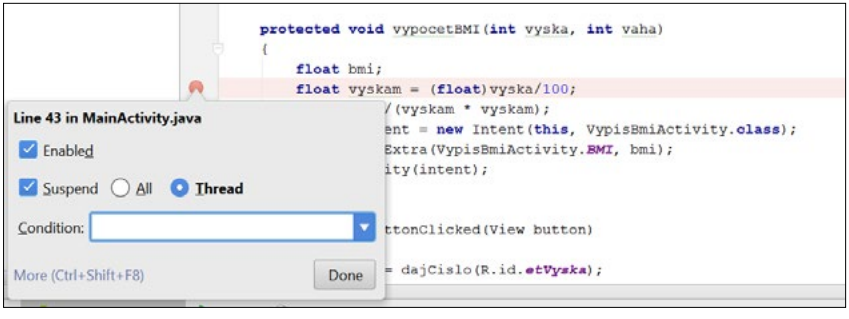
Ladění aplikace

Ladění aplikace nebyla dosud věnována žádná pozornost z jednoduchého důvodu – nebylo co ladit. Příklad výpočtu BMI z předchozího příkladu konečně přinesl nějakou relevantní, i když velmi jednoduchou logiku spočívající ve výpočtu podle vzorce. Zdálo by se, že v takovém triviálním vzorci se toho moc pokazit nedá, ale matematické operace v pohyblivé řádové čárce mají svá specifika. Vyzkoušíme si nejprve ladění správného kódu a potom do metody na výpočet BMI zavedeme úmyslně chybu.

V hlavní aktivitě umístíme zarážku (breakpoint) do metody `vypocetBMI` na řádek

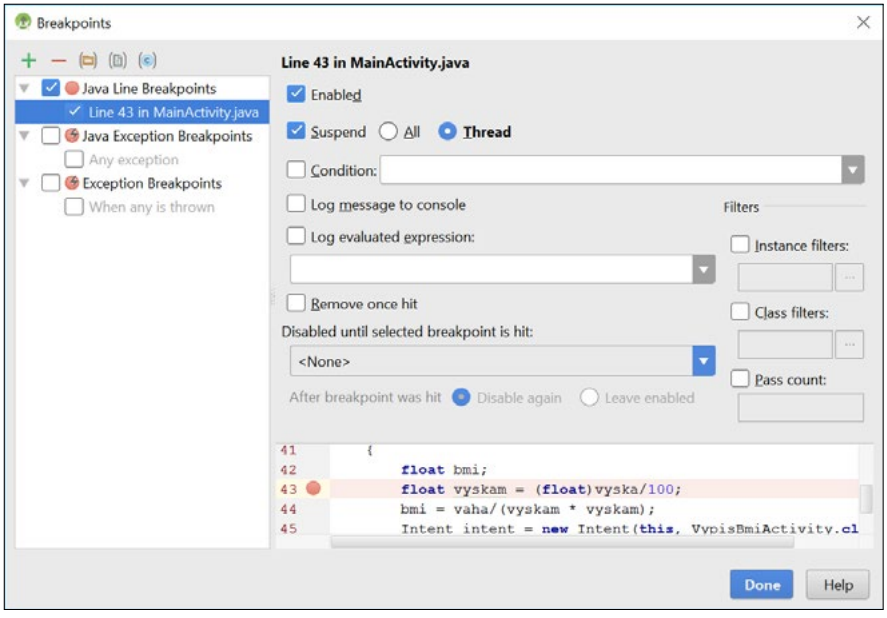
```
float vyskam = (float)vyska/100;
```

K umístění zarážky stačí klepnout na šedý okraj vlevo vedle kódu. Klepnutím na zarážku ji odstraníte. Levým tlačítkem zobrazíte jednoduchý upřesňující dialog, například pro podmínky vážící se k zarážce.



Obrázek 5.24: Nastavení parametrů zarážky (jednoduché)

Klávesovou zkratkou **Ctrl+Shift+F8** nebo klepnutím na odkaz v levé dolní části okna zobrazíte rozšířené dialogové okno pro nastavení zarážky.



Obrázek 5.25: Nastavení parametrů zarážky (rozšířené)

V této jednoduché aplikaci se spokojíme se zarážkou bez definování podmínek.

Následně tlačítkem **Debugg** (tlačítko se symbolem zeleného brouka nalevo od tlačítka na spuštění aplikace) spustíte ladění aplikace. Po zadání hodnot a jejich potvrzení se běh aplikace zastaví na řádku se zarážkou. Vývojové prostředí se přepne do ladicího módu.

Přepínač mezi módem Java a Debug je na panelu nástrojů úplně vpravo.

Na panelu nástrojů najdete i tlačítka ovládající krokování. Při ladění nejčastěji využijete:

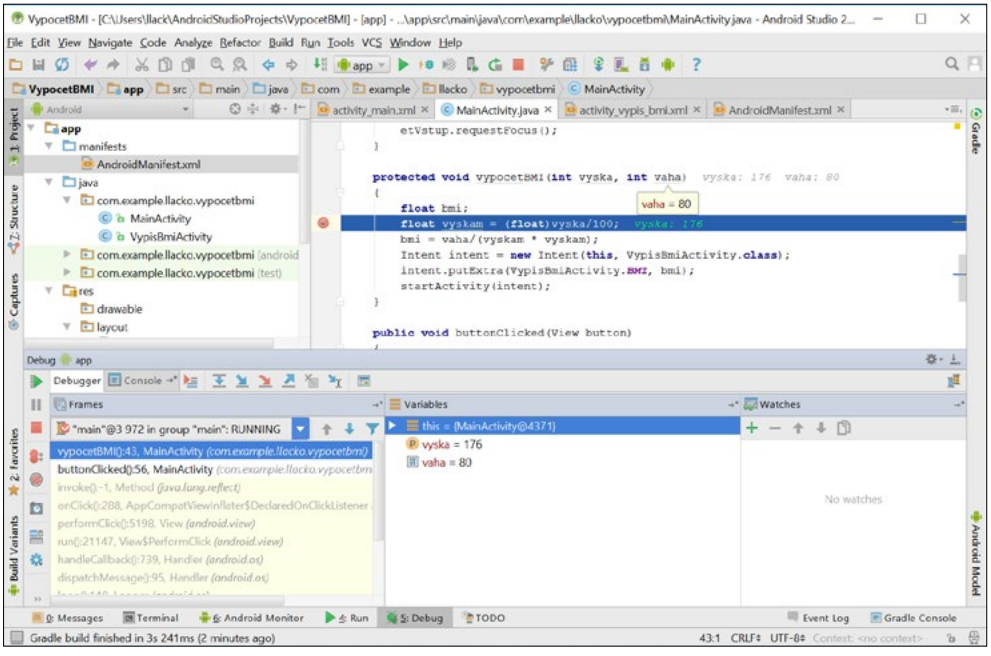
- **Step Into**
- **Step Over**
- **Step Out**

Všimněte si, že pokud umístíte kurzor nad některou proměnnou, vypíše se do bubliny její hodnota.

Můžete například otestovat výpočet po zadání výšky 176 cm a váhy 80 kg. Měla by vám vyjít hodnota BMI 25,8264.

Nyní zkuste do metody vnést drobnou chybu. Správný kód:

```
protected void vypocetBMI(int vyska, int vaha){
  float bmi;
  float vyskam = (float)vyska/100;
  bmi = vaha/(vyskam * vyskam);
  Intent intent = new Intent(this, VypisBmiActivity.class);
  intent.putExtra(VypisBmiActivity.BMI, bmi);
  startActivity(intent);
}
```



Obrázek 5.26: Vývojové prostředí v režimu ladění

Změňte řádek na konverzi centimetrů na metry:

```
float vyskam = (float)vyska/100;
```

na

```
float vyskam = vyska/100;
```

Po opětovném spuštění ladění vám po zadání výšky 176 cm vyjde výška v metrech 1 a výsledné BMI 80. Bez explicitního přetypování na `float` se totiž provede celočíselné dělení, jelikož parametr výška je typu `integer`.

KAPITOLA

6

Fragmenty

Témata kapitoly:

- **Fragmenty**
- **Příklad 1 – Změna orientace**
- **Příklad 2 – Hlavní aktivita využívající fragment**
- **Příklad 3 – Aplikace se dvěma panely**
- **Příklad 4 – Aplikace master-detail se dvěma panely**

S příchodem tabletů začalo pro vývojáře dilema, jak vytvářet aplikace, které by dobře zobrazovaly uživatelské rozhraní na telefonech i tabletech, a přitom v maximální možné míře využily možnosti displeje konkrétního zařízení. Pokud byste aplikaci určenou pro telefony spustili na velkém displeji tabletu, většina obrazovky by zůstala nevyužitá. Opačně, pokud byste aplikaci navrhli primárně pro tablet, na menších displejích telefonů by uživatelské rozhraní aplikace bylo nečitelné nebo by se vůbec nedalo zobrazit. Jedním z řešení vedle vektorové grafiky a relativních layoutů pro rozmístění ovládacích prvků jsou fragmenty.

Fragmenty byly přidány do Androidu ve verzi 3.0 kvůli lepší podpoře uživatelského rozhraní zařízení s velkými obrazovkami, tedy tabletů, notebooků a různých konvertibilních zařízení.

Typickým scénářem pro nasazení fragmentů, který se dá velmi dobře použít k názornému vysvětlení principu, jsou aplikace zobrazující informace typu master-detail. Seznam objektů se vypíše v jednom fragmentu a detailní informace o objektech v druhém. Na zařízeních s malým displejem se zobrazí každý fragment v samostatné aktivitě, takže vyplní celou plochu obrazovky. Na tabletech s velkým displejem se zobrazí oba fragmenty v rámci jedné aktivity vedle sebe zpravidla tak, že seznam objektů se zobrazí vlevo a podrobnosti o vybraném objektu vpravo.

Fragmenty jsou hostovány v aktivitách. Jedna aktivita může hostovat jeden nebo více fragmentů. Životní cyklus fragmentu se váže a je koordinovaný životním cyklem aktivity, ve které je hostován.

Existují dva hlavní způsoby, jako mohou být fragmenty přidány do aktivity:

- staticky – fragment se vloží do layoutu aktivity
- programově – pomocí objektu `FragmentManager`

Fragmenty lze díky podpůrné knihovně Android Compatibility Library (ACL) využívat i v aplikacích pro nižší verze Androidu.

Podpůrná knihovna, jejíž třídy se nacházejí ve jmenném prostoru `android.support.v4.app`, neumožňuje vložení fragmentu do layoutu aktivity pomocí kódu XML.

Hlavními výhodami fragmentů jsou modularita aplikace, opakovatelné použití hotových bloků kódu i návrhu částí uživatelského rozhraní, a především adaptibilita aplikace na různé velikosti a rozlišení obrazovek.

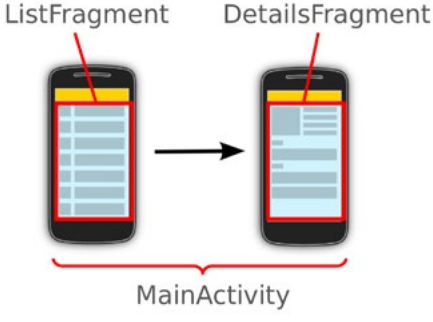
Fragmenty

Fragmenty jsou komponenty, které běží v kontextu aktivity. Fragment zapouzdřuje kód aplikace, takže se dá snáze opakovaně použít, a umožňuje diferencované zobrazování na zařízeních s různými velikostmi displeje. V aplikaci potřebujeme zobrazit strukturu typu master-detail, kde master je zpravidla seznam objektů a detail, jak vyplývá z názvu, obsahuje podrobnosti o vybraném objektu.

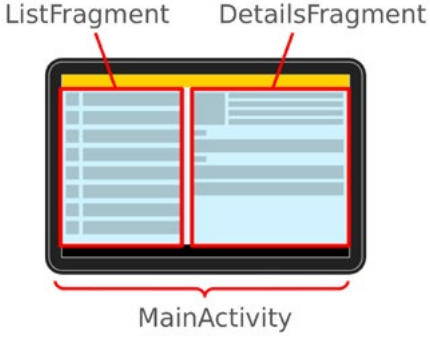
Každá z entit, tedy zobrazení seznamu objektů a detailní informace o objektu, bude zapouzdřena do samostatných fragmentů. Tablet má velký displej, takže dokáže bez problémů zobrazit oba fragmenty vedle sebe v rámci jedné aktivity. Na malém displeji telefonu potřebujete k plnohodnotnému zobrazení stejného obsahu aktivity dvě. První zobrazí seznam objektů a po výběru některého z nich překryje obrazovku druhá aktivita s detailními informacemi o vybraném objektu.

Implementace uživatelského rozhraní je zapouzdřena ve fragmentech. Aktivity fungují pouze jako kontejnery pro fragmenty, přičemž fragmenty v aktivitě se dají dynamicky měnit.

Nejlépe to vysvětlí obrázek. Na menších obrazovkách se zobrazuje pouze jeden fragment a tento umožňuje uživateli přejít do jiného fragmentu. Na větší obrazovce se zobrazí fragmenty `ListFragment` a `DetailsFragment` současně vedle sebe.



Obrázek 6.1: Zobrazení fragmentů List a Details na zařízeních s malým displejem

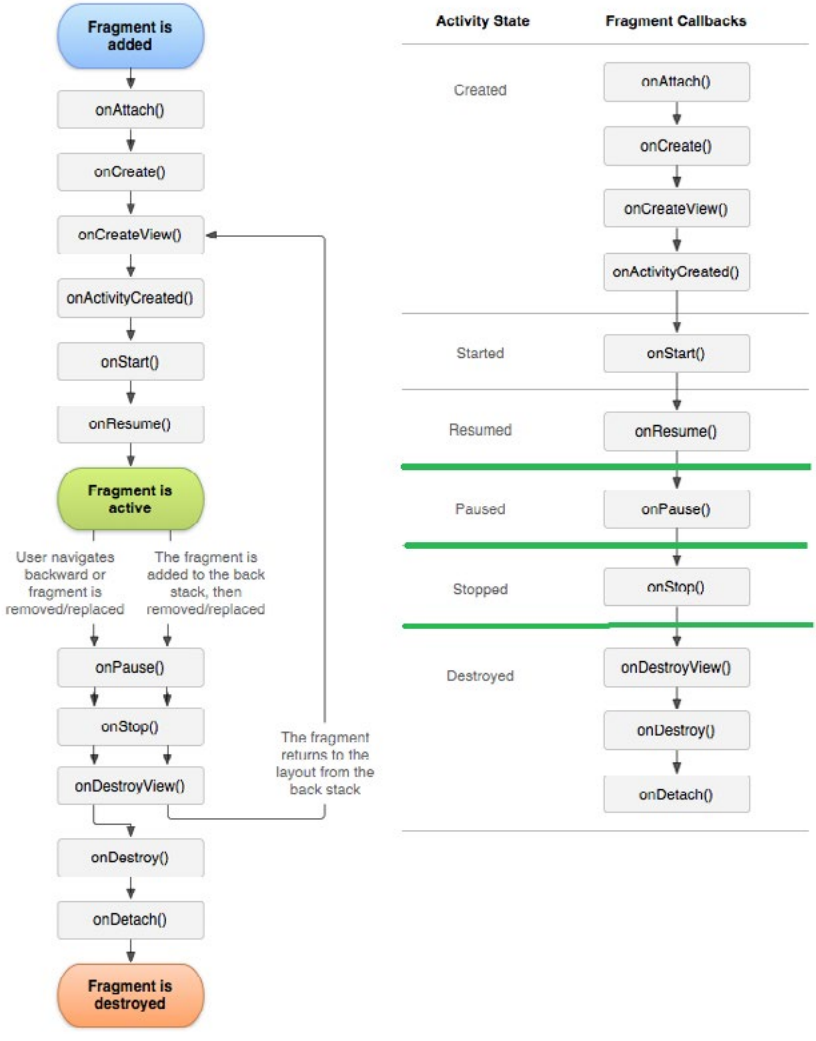


Obrázek 6.2: Zobrazení fragmentů List a Details na zařízeních s velkým displejem

Pokud to sumarizujeme, mějme dva fragmenty: A (seznam) a B (detaily). Na malé obrazovce telefonu při orientaci na výšku se postupně zobrazí dvě aktivity. První bude obsahovat fragment A (seznam) a po klepnutí na některou z položek se zobrazí fragment B (detaily). Na velké obrazovce tabletu nebo telefonu s displejem s vysokým rozlišením při orientaci na šířku se zobrazí oba fragmenty v jedné aktivitě.

Životní cyklus fragmentu

Přestože je fragment relativně samostatný objekt, z hlediska životního cyklu funguje v rámci aktivity a do značné míry je ovlivněn jejím životním cyklem. Volání metod aktivity koresponduje s voláním příslušných metod fragmentu. Například v případě pozastavení aktivity se v každém fragmentu patřícím do pozastavené aktivity aktivuje metoda `onPause()`.



Obrázek 6.3: Schéma životního cyklu fragmentu – vpravo je korelace s životním cyklem aktivity

Pozornějším čtenářům určitě neuniklo, že fragment disponuje některými metodami, které aktivita nemá. Jedná se o metody:

- `onAttach()` – metoda se aktivuje ještě před voláním metody `onCreate()` při asociaci fragmentu s aktivitou, která je fragmentu odevzdávána jako argument.

- `onCreateView()` – aktivuje se při vytváření uživatelského rozhraní. Pokud se jedná o fragment s uživatelským rozhraním, metoda vrátí `View`.
- `onActivityCreated()` – volá se po vytvoření hostitelské aktivity.
- `onDestroyView()` – volá se při odstranění fragmentu z uživatelského rozhraní po metodě `onStop()`.
- `onDetach()` – volá se při zrušení asociace fragmentu s aktivitou.

Podobně jako aktivita může být i fragment ve třech základních stavech:

- **Resumed** (spuštěný) – fragment se zobrazuje na popředí.
- **Paused** (pozastavený) – tento stav nastane, pokud je aktivita, v rámci níž je fragment zapouzdřen, částečně překryta jinou aktivitou, například dialogem.
- **Stopped** (zastavený) – fragment se nezobrazuje, avšak pokud je aktivita v Back Stacku, instance fragmentu stále existuje. Pokud je aktivita odstraněna, odstraní se z Back Stacku i všechny fragmenty zapouzdřené v aktivitě.

POZNÁMKA

Aby bylo možné fungování vláken procesů na pozadí, instance fragmentu může existovat bez ohledu na existenci hostitelské aktivity.

Fragmenty se do Back Stacku neukládají automaticky, ale explicitně pomocí metody `addToBackStack`. Stav fragmentu se ukládá v metodě `onSaveInstanceState()` prostřednictvím třídy `Bundle`. Obnovit stav je možné voláním metody `onCreate()`, případně `onActivityCreated()`. Pomocí metody `setRetainInstrance(boolean retain)` definujete, zda se má fragment při opětovném vytvoření hostitelské aktivity vytvořit nanovo nebo se uchová jeho instance. V takovém případě se místo metody `onDestroy()` aktivuje metoda `onDetach()`, a jelikož instance fragmentu existuje, nevolá se metoda `onCreate()`.

Statický a dynamický layout

Fragmenty můžete do layoutu aktivity vkládat staticky. Nevýhodou je, že takto navržená aktivita se nemůže přizpůsobit velikosti a rozlišení obrazovky.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <fragment android:name="com.example.android.fragments.ZoznamFragment"
    android:id="@+id/naspisy_fragment"
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="match_parent" />

  <fragment android:name="com.example.android.fragments.DetailyFragment"
    android:id="@+id/clanky_fragment"
    android:layout_weight="2"
    android:layout_width="0dp"
```

```
    android:layout_height="match_parent" />

</LinearLayout>
```

Dynamický layout umožňuje vytvořit flexibilní uživatelské rozhraní. Tato metoda umožňuje odevzdat fragmentu údaje, například index vybrané položky, přímo při vytvoření v aktivitě. Využívá vizuální kontejner `FrameLayout` a fragmenty se do aktivity přidávají dynamicky. Dobrým řešením je vytvořit pro fragmenty vizuální kontejnery typu `FrameLayout` jako takzvané placeholders, tj. objekty, které pro fragmenty „rezervují“ místo.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal" >

  <FrameLayout
    android:id="@+id/zoznamcontainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

  <FrameLayout
    android:id="@+id/detailycontainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:visibility="gone" />

</LinearLayout>
```

Následně použijete třídu `FragmentManager` a nahradíte kontejnery fragmenty. Tato třída umožňuje přidávat, odstraňovat a nahrazovat fragmenty v rámci layoutu aktivity. Všechny zmíněné operace se realizují jako transakce prostřednictvím třídy `FragmentTransaction`. Třída `FragmentManager` je v rámci aktivity dostupná přes metodu `getFragmentManager()`.

```
//vytvorenie fragment managera
FragmentManager fm = getFragmentManager();

// pridanie
FragmentTransaction ft = fm.beginTransaction();
ft.add(R.id.moj_placehodler, new MojFragment());
ft.commit();

// nahradenie
FragmentTransaction ft = fm.beginTransaction();
ft.replace(R.id.moj_placehodler, new MojFragment());
ft.commit();

// odstránenie
Fragment fragment = fm.findFragmentById(R.id.moj_placehodler);
FragmentTransaction ft = fm.beginTransaction();
ft.remove(fragment);
ft.commit();
```

Při dynamickém přidávání a vyměňování fragmentů v rámci layoutu aktivity potřebujete často zjistit, zda je daný fragment součástí layoutu. Třída `FragmentManager` má pro tento účel metodu `isInLayout()`:

```
DetailFragment df = (DetailFragment)
  getFragmentManager().findFragmentById(R.id.detail_frag);
if (df==null || ! fragment.isInLayout())
{
  // nova aktivita
}
else
{
  fragment.update(...);
}
```

Příklad 1 – Změna orientace

Námětem první aplikace bude přepínání fragmentů v závislosti na orientaci zařízení. V příkladu vytvoříme dva fragmenty: jeden pro orientaci na výšku (portrait) a druhý pro orientaci na šířku (landscape).

Vytvořte v Android Studiu projekt aplikace, například s názvem `Fragmenty1`. V tomto příkladu nebudeme brát ohled na nižší verze, proto nejnižší API, které můžete nastavit, je API 15: Android 4.0.3 Ice Cream Sandwich. My jsme nastavili API 21: Android 5.0 Lollipop.

POZNÁMKA

Pro nižší verze byste použili podpůrnou knihovnu `android.support.v4.app`, zpřístupňující fragmenty i pro přístroje s verzí Androidu nižší než 3.0. V takovém případě musíte mít nainstalovaný komponent Android Support Library. Přes Android SDK Manager zkontrolujte, zda máte podpůrnou knihovnu nainstalovanou. Pokud ne, doinstalujte ji.

Jako typ hlavní aktivity zvolte **Blank Activity**. Pokud byste zvolili typ **Basic Activity**, v dialogu pro zadání názvu aktivity najdete zaškrtávací pole **Use a Fragment**. Toto pole nechejte neoznačené, v prvním příkladu bude názornější realizovat všechny potřebné kroky manuálně.

Řešení úlohy rozdělíme do několika kroků:

1. Vytvoření layoutu fragmentů

Ve složce *layout* vytvořte dva soubory XML: *portrait_fragment.xml* a *landscape_fragment.xml*. V obou layoutech bude textový řetězec vypsán pomocí prvku `TextView`, pomocí něhož fragmenty po spuštění aplikace identifikujeme.

Soubor *portrait_fragment.xml*:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
```

```
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin">

    <TextView
      android:text="@string/portrait_text"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="30dp"/>

</RelativeLayout>
```

Soubor *landscape_fragment.xml*:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:paddingBottom="@dimen/activity_vertical_margin">

  <TextView
    android:text="@string/landscape_text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="30dp"/>

</RelativeLayout>
```

Pro úplnost uvádíme i soubor *strings.xml* s definicí textových řetězců.

```
<resources>
  <string name="app_name">Fragmenty1</string>
  <string name="action_settings">Settings</string>
  <string name="portrait_text">Portrait - orientácia na výšku</string>
  <string name="landscape_text">Landscape - orientácia na šírku</string>
</resources>
```

2. Hlavní aktivita

Pokračujeme úpravou návrhového kódu XML hlavní aktivity. Pokud jste vytvořili projekt typu **Empty Activity**, následně popisované úpravy budete provádět v souboru *activity_main.xml*. Pokud jste vytvořili projekt typu **Basic Activity**, definice layoutu těla aktivity je v souboru *content_main.xml*. V obou případech nahradíte prvek typu `TextView` s textem `Hello World` kódem na zobrazení fragmentů:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
```

```
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="sk.pcrevue.llacko.fragmenty1a.MainActivity">

    <fragment
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:id="@+id/landscape_fragment" />

    <fragment
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:id="@+id/portrait_fragment" />

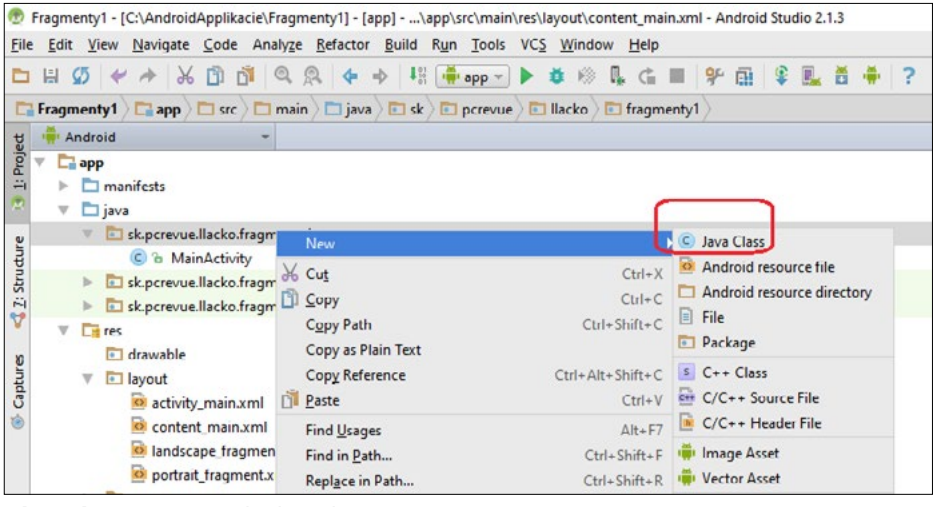
</RelativeLayout>
```

3. Vytvoření tříd fragmentů

Na vytvoření fragmentu potřebujete:

- Třídu odvozenou od `Fragment`
- Prázdný konstruktor třídy
- Rozšíření metody `CreateView()`

Ve složce *Java*, podsložce projektu aplikace, vytvořte dvě třídy: `FragmentLandscape` a `FragmentPortrait`.



Obrázek 6.4: Vytvoření kódu tříd

`LayoutInflater` vloží definici XML dovnitř kontejneru. `SavedInstanceState` zaručí připojení, pokud je fragment obnovený z uloženého stavu.

Soubor *FragmentLandscape.java*:

```
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

public class FragmentLandscape extends Fragment{

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    return inflater.inflate(R.layout.landscape_fragment, container, false);
  }
}
```

Soubor *FragmentPortrait.java* (importy jsou stejné jako v předchozím zdrojovém kódu):

```
public class FragmentPortrait extends Fragment{

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    return inflater.inflate(R.layout.portrait_fragment, container, false);
  }
}
```

4. Úprava třídy hlavní aktivity

Otevřete ve složce *java* soubor *MainActivity.java*. V metodě `onCreate` dejte do komentáře řádek přiřazující hlavní aktivitě statický layout.

```
//setContentView(R.layout.activity_main);
```

Řádek nahraďte kódem využívajícím `FragmentManager`.

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  // Objekt na interakciu fragmentov v rámci akticity
  FragmentManager fragmentManager = getFragmentManager();

  // beginTransaction() povoli operácie s fragmentmi
  FragmentTransaction fragmentTransaction =
    fragmentManager.beginTransaction();

  //zistenie orientacie zariadenai
  Configuration configInfo = getResources().getConfiguration();

  //podľa orientácie vlož správny fragment
  if(configInfo.orientation == Configuration.ORIENTATION_LANDSCAPE){
    FragmentLandscape fragmentLandscape = new FragmentLandscape();
    fragmentTransaction.replace(android.R.id.content,
```

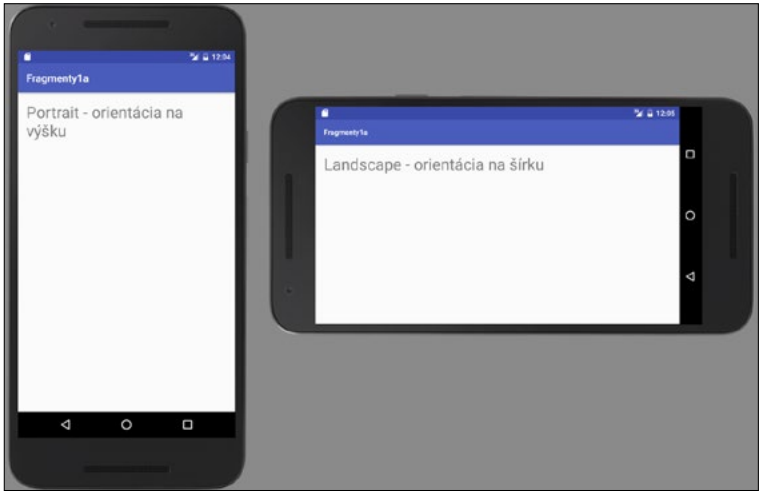
```
        fragmentLandscape);
    } else {
      FragmentPortrait fragmentPortrait = new FragmentPortrait();
      fragmentTransaction.replace(android.R.id.content,
        fragmentPortrait);
    }

    // nahraď fragment tak rýchlo ako je to možné
    fragmentTransaction.commit();

    //setContentView(R.layout.activity_main);
  }
```

5. Test aplikace

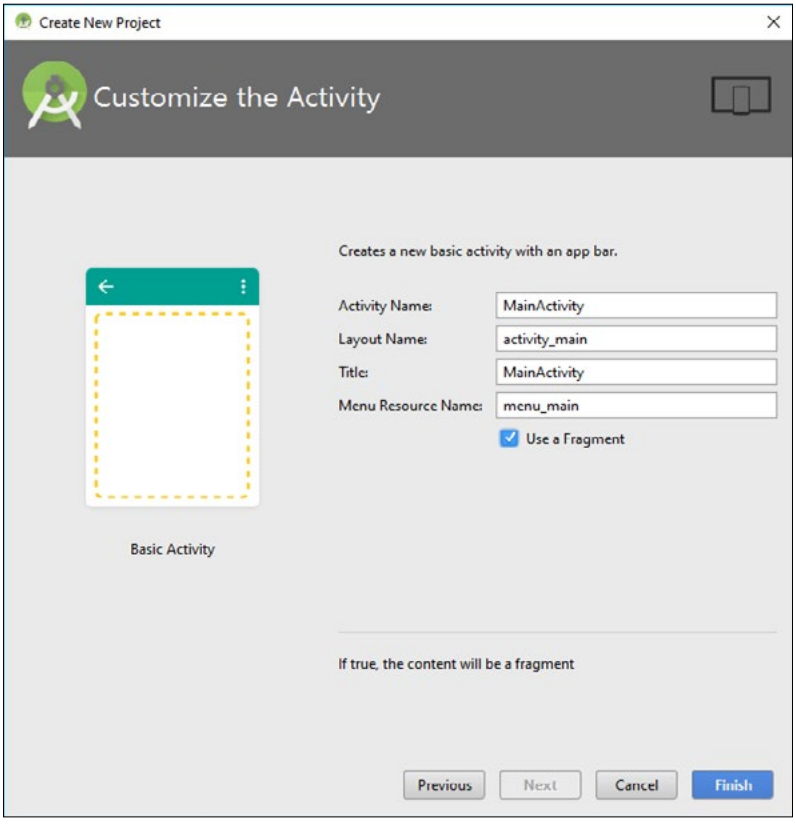
Nyní můžete aplikaci vyzkoušet, nejlépe na reálném zařízení, a sledovat, zda se změní fragment při změně orientace zařízení. Změnu orientace zařízení velmi snadno nasimulujete i na emulátoru.



Obrázek 6.5: Zobrazení různých fragmentů při změně orientace zařízení

Příklad 2 – Hlavní aktivita využívající fragment

Při vytváření projektu typu **Basic Activity** si všimněte zaškrtávacího pole **Use a Fragment**. V předchozím projektu jsme ho nechali neoznačené. V tomto příkladu ukážeme, jaký kód se přidá do projektu, pokud toto pole označíte.



Obrázek 6.6: Vytvoření hlavní aktivity využívající fragmenty. V příkladu označte položku Use a Fragment.

Nejprve si prohlédněte návrhy layoutů *activity_main.xml*, *content_main.xml* a *fragment_main.xml*. Výpisy jsou zkrácené.

V souboru *activity_main.xml* je definovaná aplikační lišta, plovoucí tlačítko a z našeho pohledu je klíčový řádek `<include layout="@layout/content_main" />`:

```
<android.support.design.widget.CoordinatorLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  ...">

  <android.support.design.widget.AppBarLayout
    ...>
  </android.support.design.widget.AppBarLayout>

  <include layout="@layout/content_main" />

  <android.support.design.widget.FloatingActionButton
    .../>

</android.support.design.widget.CoordinatorLayout>
```

Soubor *content_main.xml* obsahuje staticky definovaný fragment:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/fragment"
  android:name="sk.pcrevue.llacko.fragmenty2.MainActivityFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  app:layout_behavior="@string/appbar_scrolling_view_behavior"
  tools:layout="@layout/fragment_main" />
```

Soubor *fragment_main.xml* – kód fragmentu obsahuje definici layoutu, tj. prvek `TextView` zapouzdřený ve vizuálním kontejneru `RelativeLayout`.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="sk.pcrevue.llacko.fragmenty2.MainActivityFragment"
  tools:showIn="@layout/activity_main">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" />

</RelativeLayout>
```

Soubor *MainActivityFragment.java* obsahuje kód:

```
import android.support.v4.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

/**
 * A placeholder fragment containing a simple view.
 */
public class MainActivityFragment extends Fragment {

  public MainActivityFragment() {
  }

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
```

```
        return inflater.inflate(R.layout.fragment_main, container, false);
    }
}
```

Tento projekt můžete následně rozšiřovat o vlastní fragmenty a doplňovat požadovanou funkcionalitu. V porovnání s předchozím příkladem, kde jsme všechno dělali ručně, však tato šablona projektu velkou přidanou hodnotu nepřináší.

Příklad 3 – Aplikace se dvěma panely

Tento příklad ukazuje nejčastější scénář použití fragmentů. Bude to příprava na další příklad, ve kterém budeme definovat dva fragmenty, jeden s informacemi o položkách a druhý s jejich seznamem. Příklad se zaměří na skrývání fragmentu a změny rozměrů při změně orientace. V příkladu nebudeme prioritně řešit způsob přibalení údajů ani interakci typu master-detail, ale zaměříme se na jejich vizuální prezentaci. Proto budeme fragmenty rozlišovat jen podle barvy pozadí a textu. Na telefonu s relativně malým displejem se na celé ploše displeje zobrazí jako hlavní aktivita aplikace detail, v tomto případě hlavní panel. U orientace na šířku se zobrazí vlevo boční panel.

POZNÁMKA

V reálné aplikaci by se vlevo zobrazil seznam objektů. Po klepnutí na některý objekt se na ploše obrazovky zobrazí jiná aktivita s druhým fragmentem s detailními informacemi o vybraném objektu.

Vytvořte projekt, úplně stačí šablona typu **Empty Activity**. Podobně jako v prvním příkladu z této série vytvořte definice layoutů fragmentů. V příkladu jsou fragmenty nazvané *zoznam_fragment.xml* a *detail_fragment.xml*, aby to korespondovalo s následující aplikací.

1. Vytvoření layoutu fragmentů

Nebudeme řešit zobrazování údajů, panely budeme rozlišovat podle barvy a šířky. Tmavě modrý panel bude suplovat seznam objektů a oranžový zobrazení informace o vybraném objektu.

Ve složce *layout* vytvořte dva soubory XML: *zoznam_fragment.xml* a *detail_fragment.xml*. V obou layoutech bude kromě barevného pozadí textový řetězec vypsán pomocí prvku `TextView`, pomocí něhož fragmenty po spuštění aplikace identifikujeme.

Soubor *zoznam_fragment.xml*:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:background="@android:color/holo_blue_dark">

  <TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Zoznam"
    android:textColor="@android:color/white"
    android:textStyle="bold"
    android:textSize="20sp"/>
</RelativeLayout>
```

Soubor *detail_fragment.xml*:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:background="@android:color/holo_orange_light">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="D e t a i l y"
    android:textColor="@android:color/white"
    android:textStyle="bold"
    android:textSize="20sp"/>

</RelativeLayout>
```

2. Hlavní aktivita

V tomto případě upravíme kód hlavní aktivity v souboru *activity_main.xml*. Tento soubor bude definovat layout pro orientaci na výšku (portrait). Fragmenty jsou pojmenované jako hlavní panel (v reálné aplikaci by zobrazoval detaily o objektech) a boční panel (ten by zobrazoval seznam objektů).

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="sk.pcrevue.llacko.fragmenty3.MainActivity">

  <fragment
    android:id="@+id/hlavny_panel"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
```

```
    class="sk.pcrevue.llacko.fragmenty3.FragmentDetail" />

  <fragment
    android:id="@+id/bocny_panel"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    class="sk.pcrevue.llacko.fragmenty3.FragmentZoznam" />

</RelativeLayout>
```

Ve složce *res* vytvořte novou vnořenou složku s názvem *layout-land* a v ní nový soubor *activity_main.xml*. Jak napovídá název složky, bude obsahovat definici layoutu hlavní aktivity při orientaci na šířku. Jediný rozdíl je, že šířka bočního panelu je definovaná fixně na 150 dp.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="sk.pcrevue.llacko.fragmenty3.MainActivity">

  <fragment
    android:id="@+id/hlavny_panel"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    class="sk.pcrevue.llacko.fragmenty3.FragmentDetail" />

  <fragment
    android:id="@+id/bocny_panel"
    android:layout_width="150dp"
    android:layout_height="match_parent"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    class="sk.pcrevue.llacko.fragmenty3.FragmentZoznam" />

</RelativeLayout>
```

3. Vytvoření tříd fragmentů

Ve složce *Java*, podsložce projektu aplikace, vytvořte dvě třídy: `FragmentZoznam` a `FragmentDetail`. Zdrojové kódy jsou zkráceny bez importů.

Soubor *FragmentZoznam.java*:

```
public class FragmentZoznam extends Fragment {
  @Override
```

```
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.zoznam_fragment, container, false);
    return view;
  }
}
```

Soubor *Fragmentdetail.java*:

```
public class FragmentDetail extends Fragment {

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.detail_fragment, container, false);
    return view;
  }
}
```

4. Úprava třídy hlavní aktivity

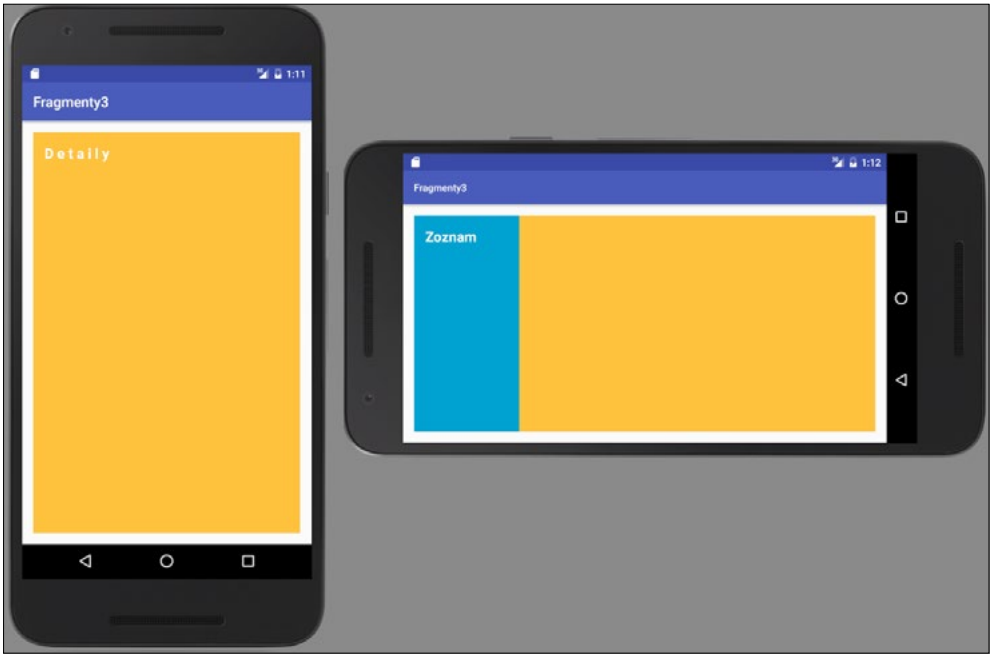
Otevřete ve složce *java* soubor *MainActivity.java*. V metodě `onCreate` se podle orientace zařízení boční panel skrývá (na výšku), nebo zobrazuje (na šířku).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  // skry bočný panel
  int screenOrientation = getResources().getConfiguration().orientation;
  if (screenOrientation == Configuration.ORIENTATION_PORTRAIT) {
    View sidePane = findViewById(R.id.bocny_panel);
    if (sidePane.getVisibility() == View.VISIBLE) {
      sidePane.setVisibility(View.GONE);
    }
  }
}
```

5. Test aplikace

Nyní můžete aplikaci vyzkoušet, nejlépe na reálném zařízení, a sledovat, zda se změní fragment při změně orientace zařízení. Změnu orientace zařízení velmi snadno nasimulujete i na emulátoru.



Obrázek 6.7: Zobrazení a skrytí fragmentů při změně orientace zařízení

Příklad 4 – Aplikace master-detail se dvěma panely

Tento příklad je rozšířením předchozího, ve kterém jsme se soustředili na zobrazení a skrývání fragmentů. V tomto příkladu už vytvoříme reálnou aplikaci, která ukazuje nejčastější scénář použití fragmentů. Budeme definovat dva fragmenty: jeden se seznamem a druhý zobrazí detaily o vybrané položce.

Na telefonu s relativně malým displejem se na celé ploše displeje zobrazí hlavní aktivita aplikace s fragmentem obsahujícím seznam objektů. Po klepnutí na některý objekt se na ploše obrazovky zobrazí jiná aktivita s druhým fragmentem s detailními informacemi o vybraném objektu. Aplikace by v takovéto podobě fungovala i na tabletu, plocha v pravé části displeje by však zůstala nevyužitá. Proto se na tabletu nebo telefonu při orientaci na šířku zobrazí dva fragmenty v rámci jedné aktivity vedle sebe: vlevo seznam objektů a vpravo detailní informace o vybraném objektu.

Námětem aplikace bude zobrazení informací o verzích Androidu. V reálné aplikaci by se údaje načítaly z webu například přes JSON, z lokální databáze a podobně. Klíčové dějství fragmentu nastane v okamžiku výběru některé položky. V závislosti na velikosti displeje, případně jiných kritérií, například orientace zařízení, je potřeba rozhodnout, zda se fragment na výpis detailních informací spustí v nové aktivitě, nebo ve vedlejším sloupci v té samé aktivitě. Aby byla aplikace co nejjednodušší, nepoužijeme jako rozhodovací kritérium o tom, zda aplikace zobrazí každý fragment v samostatné aktivitě nebo oba fragmenty v rámci jedné aktivity, rozlišení ani rozměry displeje, ale orientaci zařízení.

V režimu „na výšku“ se každý fragment zobrazí v samostatné aktivitě a v režimu „na šířku“ se oba fragmenty zobrazí v rámci hlavní aktivity. Uvidíte, že v mnoha případech to není špatná volba, jelikož displeje moderních telefonů mají vysoké rozlišení, typické pro tablety dva roky staré.

Vytvořte projekt, pro hlavní aktivitu úplně stačí nejjednodušší šablona typu **Empty Activity**.

1. Zobrazované údaje

V tomto příkladu použijeme statický seznam údajů. Ve složce *Java* vytvořte třídu `AndroidInfo`. Údaje jsou uloženy ve dvou polích řetězců, přesněji: popisy jednotlivých verzí jsou zkráceny.

```
public class AndroidInfo {
  public static final String[] NAZVY =
  {
    "1.5 Cupcake",
    "1.6 Donut",
    "2.0 Eclair",
    "2.2 Froyo",
    "2.3 Gingerbread"
  };

  public static final String[] POPISY =
  {
    "(malý muffin v košíčku so šľahačkou – 30.4.2009) ...",
    "(šiška s otvorom – 1.9.2009) ...",
    "(podlhovastý zákusok s plnkou a čokoládou 26.10.2009) ...",
    "(šľahačkový špic obložený ovocím – 20.5.2010)...",
    "(perník – 6.12.2010)..."
  };
}
```

2. Hlavní aktivita

Kód třídy *main_activity.java* zůstane beze změn:

```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
}
```

V definici layoutu hlavní aktivity v souboru *activity_main.xml* změňte vizuální kontejner `RelativeLayout` na `Frame Layout`. Stačí kontejner přejmenovat a odstranit atributy, ve kterých je definované odsazení (padding). Do kontejneru doplňte fragment definovaný ve třídě `ZoznamFragment`. Aby vám tento řádek nesvítil červeně jako chyba (reference na neexistující objekt), můžete ve složce *Java*, podsložce projektu, vytvořit třídu `ZoznamFragment`. Kód třídy dopíšete později.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="sk.pcrevue.llacko.fragmenty4.MainActivity">

    <fragment class="sk.pcrevue.llacko.fragmenty4.ZoznamFragment"
      android:id="@+id/zoznam"
      android:layout_width="match_parent" android:layout_height="match_parent"
    />
  </FrameLayout>
```

Podobně jako v předchozím příkladu vytvořte ve složce *res* novou složku *layout-land* a v ní nový soubor *activity_main.xml*. Bude obsahovat dva fragmenty pro seznam objektů a výpis detailů zapouzdřené ve vizuálním kontejneru `LinearLayout`. Nastavení atributů `layout_weight` definuje horizontální rozdělení plochy obrazovky mezi fragmenty v poměru 1 : 3, přičemž širší bude fragment na zobrazení detailních informací.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent" android:layout_height="match_parent">
  <fragment class="sk.pcrevue.llacko.fragmenty4.ZoznamFragment"
    android:id="@+id/zoznam"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1"/>

  <FrameLayout android:id="@+id/detaily"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:background="?android:attr/detailsElementBackground"
    android:layout_weight="3"/>
```

Podobně jako v předchozím příkladu vytvořte definice layoutů fragmentů.

3. Kód fragmentu na zobrazení seznamu objektů

Seznam objektů se bude zobrazovat pomocí prvku `ListView`. Po výběru prvku se aktivuje `DetailsFragment` – buď v té samé aktivitě při orientaci na šířku, nebo v nové aktivitě `DetailsActivity` při orientaci na výšku.

Objekt `ArrayAdapter` napojí pole s údaji na `ListView` `getActivity()`. Metoda vrátí kontext. Objekt na zobrazení seznamu `simple_list_item_activated_1` je verzí `simple_list_item_1`, která umožňuje zvýraznit pozadí vybraného objektu v seznamu.

Kompletní kód *ZoznamFragment.java*:

```
public class ZoznamFragment extends ListFragment {

  boolean mDvaPanely;

  // vybrany prvok ListView
  int mVybrany = 0;

  @Override
  public void onActivityCreated(Bundle savedInstanceState) {
```

```
  super.onActivityCreated(savedInstanceState);

  ArrayAdapter<String> connectArrayToListView = new ArrayAdapter<String>(
    getActivity(),
    android.R.layout.simple_list_item_activated_1,
    AndroidInfo.NAZVY);

  //Pripoj ListView na data
  setListAdapter(connectArrayToListView);
  View detailsFrame = getActivity().findViewById(R.id.detaily);
  // sú dva panely?
  mDvaPanely = detailsFrame !=
    null && detailsFrame.getVisibility() == View.VISIBLE;
  // onSaveInstanceState() pri rotácii
  if (savedInstanceState != null) {
    // obnov pozíciu
    mVybrany = savedInstanceState.getInt("curChoice", 0);
  }

  if (mDvaPanely) {
    getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    showDetails(mVybrany);
  }
}

// zmena orientácie
@Override
public void onSaveInstanceState(Bundle outState) {
  super.onSaveInstanceState(outState);
  outState.putInt("curChoice", mVybrany);
}

@Override
public void onListItemClick(ListView l, View v, int position, long id) {
  showDetails(position);
}

//zobraz detaily
void showDetails(int index) {
  mVybrany = index;
  // Zoznam v ListView a detaily vedľa seba
  if (mDvaPanely) {
    getListView().setItemChecked(index, true);
    // zobrazenie detailov
    DetailFragment details = (DetailFragment)
    getFragmentManager().findFragmentById(R.id.detaily);

    if (details == null || details.getShownIndex() != index) {
      // detail fragment pre vybraný index
      details = DetailFragment.newInstance(index);
      FragmentTransaction ft = getFragmentManager().beginTransaction();
```

```
        //nahradenie detail fragmentu aktuálnym
        ft.replace(R.id.detaily, details);
        ft.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_FADE);
        ft.commit();
      }

    } else {
      // DetailFragment v novej aktivite
      Intent intent = new Intent();
      intent.setClass(getActivity(), DetailsActivity.class);
      intent.putExtra("index", index);
      startActivity(intent);
    }
  }
}
```

4. Kód fragmentu na zobrazení seznamu detailů

Následně vytvořte kód pro fragment zobrazující detaily. Jelikož informace o objektu mohou přesáhnout rozměry obrazovky, návrh fragmentu obsahuje i prvek `ScrollView`, umožňující svislé rolování. Třída `DetailFragment` na zobrazení detailních informací o vybraném objektu je odvozena od třídy `Fragment`. Každá takto odvozená třída musí navíc přepisovat metodu `onCreateView()`. Parametr, v tomto případě index aktuálně vybrané položky, se na rozdíl od aktivit nepředává pomocí záměrů, ale přímo pomocí argumentů přes objekt typu `Bundle`.

Kompletní kód *DetailFragment.java*:

```
public class DetailFragment extends Fragment {

  public static DetailFragment newInstance(int index) {
    DetailFragment f = new DetailFragment();
    // načítanie indexu vybraneho objektu
    Bundle args = new Bundle();
    args.putInt("index", index);
    f.setArguments(args);
    return f;
  }

  public int getShownIndex() {
    return getArguments().getInt("index", 0);
  }

  // LayoutInflator umiestni Fragment na obrazovku
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ScrollView scroller = new ScrollView(getActivity());
    TextView text = new TextView(getActivity());
    int padding = (int)
      TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
      4, getActivity().getResources().getDisplayMetrics());
    text.setPadding(padding, padding, padding, padding);
```

```
      scroller.addView(text);
      text.setText(AndroidInfo.POPISY[getShownIndex()]);
      return scroller;
    }
  }
```

5. Aktivita na zobrazení detailu

Při orientaci na šířku, případně na tabletu (toto v aplikaci neindikujeme), vystačíte s hlavní aktivitou, pomocí které zobrazíte fragment seznamu objektů a vedle něho fragment s detaily o vybraném fragmentu. Na telefonu budete potřebovat další aktivitu na zobrazení detailů. Účelem této aktivity je zobrazení vybraného detailu na samostatné obrazovce. Vytvořte novou třídu `DetailsActivity`.

Soubor *DetailsActivity.java*:

```
public class DetailsActivity extends Activity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //je orientácia na šírku? vtedy túto aktivitu netreba
    if (getResources().getConfiguration().orientation
      == Configuration.ORIENTATION_LANDSCAPE) {
      finish();
      return;
    }

    if (savedInstanceState == null) {
      DetailFragment details = new DetailFragment();
      // odovzdanie indexu vybranej polozky
      details.setArguments(getIntent().getExtras());
      getFragmentManager().beginTransaction().add(android.R.id.content,
        details).commit();
    }
  }
}
```

Novu aktivitu je potřeba přidat do manifestu aplikace v souboru *AndroidManifest.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.fragmenty4">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
```

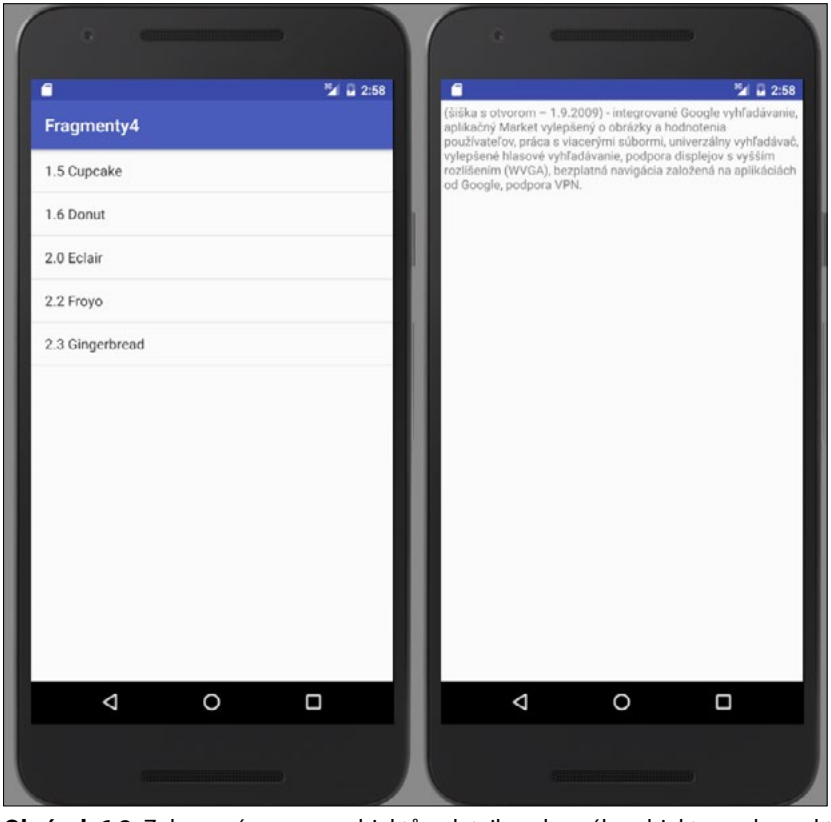
```
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

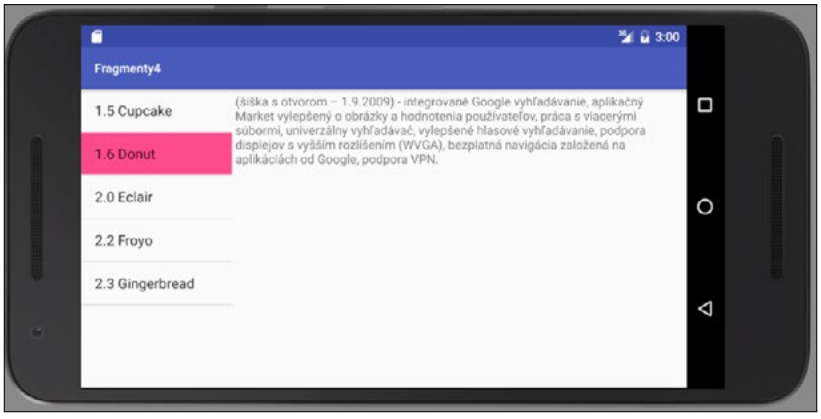
    <activity android:name=".DetailsActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

6. Test aplikace

Následuje velké finále tohoto komplexního příkladu. Nyní můžete aplikaci vyzkoušet, nejlépe na reálném zařízení, a sledovat, zda se změní fragment při změně orientace zařízení. Změnu orientace zařízení velmi snadno nasimulujete i na emulátoru.



Obrázek 6.8: Zobrazení seznamu objektů a detailu vybraného objektu ve dvou aktivitách



Obrázek 6.9: Zobrazení seznamu objektů a detailu vybraného objektu ve dvou fragmentech vedle sebe v jedné aktivitě

Pokud jste nezapomněli přidat `DetailsActivity` do aplikačního manifestu v souboru *AndroidManifest.xml* (to je jen připomenutí nejčastější chyby, kvůli které vám tato aplikace nebude fungovat), můžete aplikaci otestovat. Otestujte zobrazení Zoznam – Detail při orientaci zařízení na výšku, kdy by se měl zobrazit fragment s detailními informacemi v samostatné aktivitě. Při orientaci zařízení na šířku by se měly zobrazovat oba fragmenty vedle sebe.

KAPITOLA

7

Aktivity a intenty

Témata kapitoly:

- **Intent**
- **Explicitní intenty**
- **Implicitní Intenty**

Intenty jsme už zmínili v kapitole věnované anatomii aplikace a jejímu životnímu cyklu. Slouží ke komunikaci v rámci operačního systému a umožňují například to, aby jedna aktivita spustila jinou, případně spustila službu a podobně. Intenty umožňují, aby aplikace deklarovala, že dokáže realizovat nějakou činnost, například otevřít obrázek či přehrát multimediální soubor.

Intent

Intent (`android.content.Intent`) je asynchronní zpráva, která nese informaci o požadované akci. Akcí může být spuštění aktivity, služby, případně uživatelsky definovaná akce. Intent může obsahovat údaje, které jsou uloženy v instanci třídy `Bundle`. Jak vyplývá z názvu odvozeného od slova intention (záměr), vazby mezi aktivitami, jejichž vstupy a výstupy jsou definované pomocí intentů, jsou slabé a neostré.

Je to určitá analogie s reálným světem: záměr, případně úmysl, lze realizovat různými způsoby a celkem dobře se může stát, že se jeho realizace nemusí podařit. Intent operačnímu systému oznamuje, co má aktivita v úmyslu. Například uživatel chce poslat SMS, vyhledat údaje ze seznamu kontaktů, aktivovat vestavěný fotoaparát a podobně. Operační systém tento úmysl může splnit různým způsobem, případně ho splnit nedokáže.

Třída `Intent` je v podstatě datová struktura, která se používá ke dvěma účelům. Umožňuje zadat operaci, kterou je potřeba provést, nebo záměr může představovat událost, která nastala v systému a o které je potřeba informovat další komponenty. Pokud chce aplikace provést operaci, ke které potřebuje součinnost jiné aktivity, například vybrat kontakt, zobrazit mapu, sejmout obrázek, vytočit telefonní číslo a podobně, vyjádří to přes objekt `Intent`.

Explicitní intenty

Tyto intenty přesně určují, jakou akci je potřeba provést, kdo a jakým způsobem ji provede a s jakými údaji. Mají explicitně definovanou cílovou aktivitu, kterou je potřeba spustit. Pomocí intentů je možné cílové aktivitě odevzdat údaje. Typickým pří-

kladem je aplikace se dvěma aktivitami – první obsahuje formulář na zadání údajů a druhá zobrazí výsledky.

```
Intent i = new Intent(this, Activity2.class);
i.putExtra("paramString", "Text pre druhú aktivitu");
i.putExtra("paramInt", 42);
startActivity(i);
```

Konstruktor nebo metoda `setClass()` slouží k upřesnění, jaká konkrétní třída se má po odeslání intentu spustit. První parametr `Context` udává odkaz na aktuální komponentu. Druhý parametr `Class` definuje třídu, která se má po odeslání intentu spustit.

Metoda `putExtra()` slouží k odevzdávání si hodnot mezi komponentami. Je možné odevzdat jakýkoliv objekt `Serializable` nebo `Parcelable`. Objekty se mapují pomocí textových řetězců.

Způsob spuštění aktivity je definovaný pomocí příznaků:

- `FLAG_ACTIVITY_NO_ANIMATION` – vypne animaci při spouštění aktivity.
- `FLAG_ACTIVITY_NO_HISTORY` – spuštěná aktivita se neuloží do zásobníku. Když uživatel aktivitu opustí, bude ukončena.
- `IFLAG_ACTIVITY_CLEAR_TOP` – pokud už je spouštěná aktivita v zásobníku, obnoví se a všechny aktivity nad ní budou ukončeny.
- `FLAG_ACTIVITY_REORDER_TO_FRONT` – pokud už je spouštěná aktivita v zásobníku, obnoví se a přesune se navrch zásobníku.
- `FLAG_ACTIVITY_SINGLE_TOP` – pokud je spouštěná aktivita na vrcholu zásobníku, obnoví se a nespustí se nová aktivita.

Příznaky se nastavují metodou:

```
Intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
```

Způsob spuštění aktivity může být definován i v *AndroidManifest.xml* atributem `android:launchMode`:

- `standard` – vždy se vytvoří nová instance.
- `singleTop` – vždy se vytvoří nová instance, pokud už není instance aktivity na vrcholu zásobníku.
- `singleTask` – vytvoří se nový task, a pokud instance aktivity už existuje, dojde k přesměrování na ni. Typickým příkladem je webový prohlížeč či přehrávač multimédií.
- `singleInstance` – jako `singleTask`, ale v rámci tasku je vždy jen jedna aktivita.

POZNÁMKA

Back Button funguje vždy bez ohledu na hodnotu parametru `android:launchMode`.

Aktivita, která je intentem „oslovena", přijme parametry nebo údaje přes objekt `Bundle` s využitím metody `getIntent().getExtras()`. Metoda `getIntent()` vrátí Intent, který aktivitu spustil, a metoda `getExtras()` vrátí objekt typu `Bundle`, který obsahuje data:

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
```

```
    String paramString = extras.getString("paramString");
    int paramInt = extras.getInt("paramInt");
  }
```

Třída `Bundle` umožňuje přenos údajů mezi komponentami. Údaje se ukládají ve formě párových dvojic klíč-hodnota. Přenos údajů přes mechanizmus `Bundle` je poměrně rychlý, je ale určen pouze pro menší objemy dat, řádově kilobajty. Instanci třídy `Bundle` je možné získat pomocí metody třídy `Intent getExtras()`.

Třída `Bundle` bude přenášet libovolné datové typy. Proto je potřeba definovat rozhraní zajišťující serializaci tříd `Serializable` nebo `Parcelable`. Rozhraní `Serializable` umožňuje dynamické rozpoznávání typů, které je však náročné, takže tento způsob je řádově pomalejší než serializace pomocí rozhraní `Parcelable`, které vyžaduje explicitní definici pro serializaci i deserializaci třídy.

Přenést data z aktivity A do aktivity B, kterou aktivita A spouští přes intent, je jednoduché. Opačně, tedy z aktivity B do aktivity A, tento mechanizmus nefunguje. Řešením je spustit aktivitu B jako subaktivitu. Pokud se po spuštění aktivity očekává, že když aktivita B skončí, vrátí výsledek, namísto metody `startActivity ()`, je potřeba aktivitu spustit pomocí metody `startActivityForResult()`. Výsledek se nastaví pomocí metody `setResult()` v metodě `finish()`. V aktivitě, která očekává výsledek od jiné aktivity, je potřeba předefinovat metodu `onActivityResult()`. K dispozici jsou tři předdefinované hodnoty:

- `RESULT_OK = -1`
- `RESULT_CANCELLED = 0`
- `RESULT_FIRST_USED = 1`

Uživatelsky definované hodnoty výsledků aktivity následují od konstanty `RESULT_FIRST_USED`.

```
startActivityForResult(i, REQUEST_CODE);
```

Aby tento mechanizmus, tedy odevzdávání údajů z aktivity B do rodičovské aktivity A, fungoval, musí mít subaktivita implementovanou metodu `finish()`, ve které se vytvoří objekt typu intent obsahující údaje, které aktivita B vrátí rodičovské aktivitě A. Vrácený kód je `RESULT_OK` nebo `RESULT_CANCELED`.

```
public void finish() {
  Intent ii = new Intent();
  ii.putExtra("vratenyString", "Text pre rodičovskú aktivitu");
  setResult(RESULT_OK, ii);
  super.finish();
}
```

Rodičovská aktivita přebere údaje pomocí metody `onActivityResult()`:

```
protected void onActivityResult(int requestCode,
  int resultCode, Intent data) {
  String sVrateny;

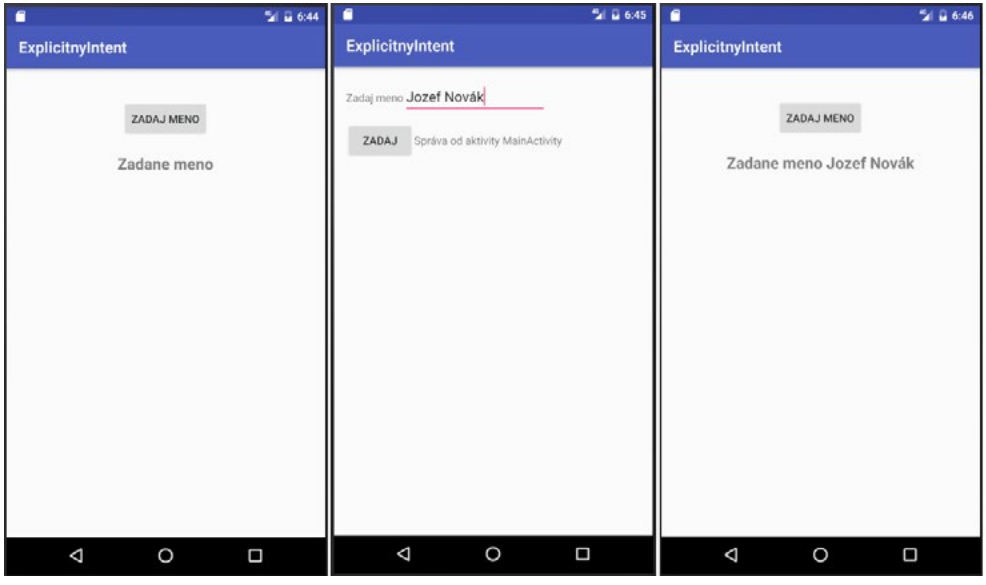
  if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
    if (data.hasExtra("vratenyString")) {
      sVrateny = data.getExtras().getString("vratenyString");
    }
  }
```

```
}
```

Příklad – Explicitní intenty

Příklad se zaměřuje na vzájemnou komunikaci primární a sekundární aktivity. Dříve než začnete číst námět, prohlédněte si snímky hotové spuštěné aplikace.

- Aplikace spustí hlavní aktivitu.
- Uživatel stiskne tlačítko, že chce zadat jméno. Tehdy se spustí sekundární aktivita. Abychom ukázali fungování přenosu údajů z volající do volané aktivity, přenese se z hlavní do sekundární aktivity textový řetězec.
- Uživatel v sekundární aktivitě zadá jméno.
- Po jeho potvrzení se znovu zobrazí hlavní aktivita a zobrazí zadané jméno.



Obrázek 7.1: Test příkladu

Návrhový kód obou aktivit je triviální záležitostí.

Hlavní aktivita *activity_main.xml*:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="20dp"
  android:orientation="vertical"
  android:gravity="center|top"
  tools:context="sk.pcrevue.llacko.explicitnyintent.MainActivity">

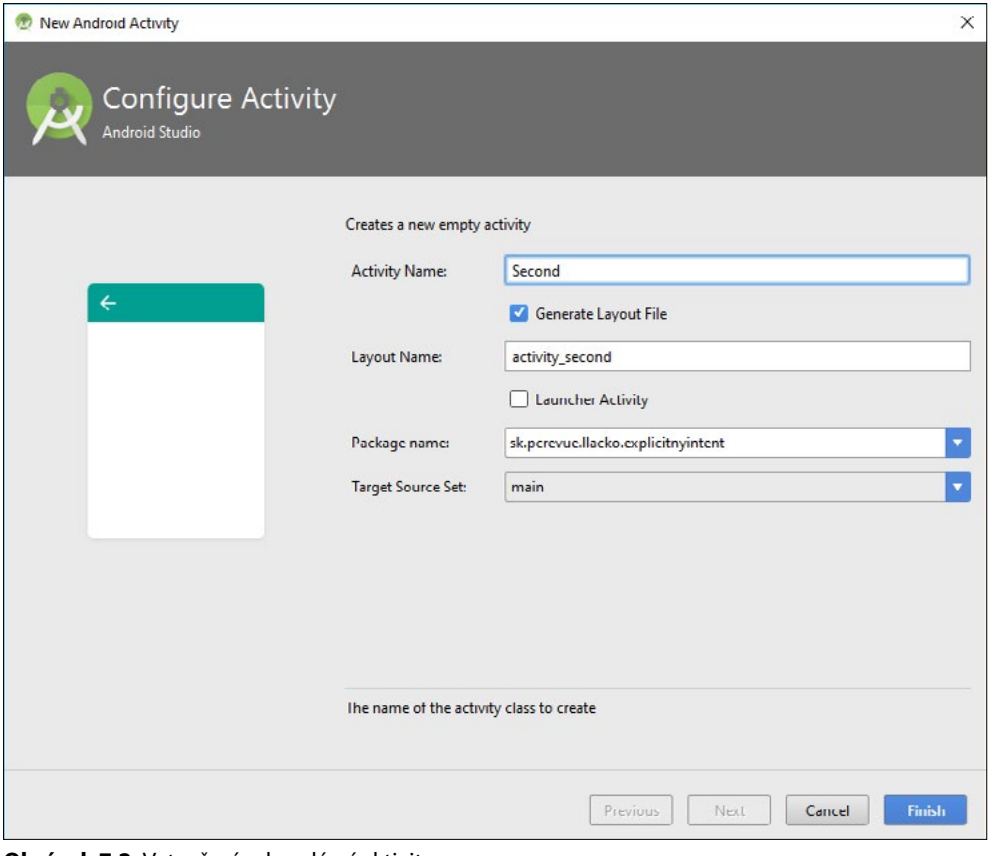
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
        android:text="Zadaj meno"
        android:layout_marginTop="20dp"
        android:onClick="onClick"/>

    <TextView
        android:text="Zadane meno"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
        android:textSize="20sp"
        android:layout_marginTop="20dp"
        android:id="@+id/meno_od_volanej_aktivity"/>

</LinearLayout>
```

Sekundární aktivitu nejjednodušeji vytvoříte pomocí místní nabídky a voleb **New** → **Activity** → **Empty Activity.**



Obrázek 7.2: Vytvoření sekundární aktivity

Sekundární aktivita *activity_second.xml* obsahuje hlavní vizuální kontejner `LinearLayout`, který horizontálně zapouzdřuje dva vnořené kontejnery, ale s vertikální orientací dvojic prvků.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:gravity="left"
  android:padding="20dp"

  tools:context="sk.pcrevue.llacko.explicitnyintent.Second">

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Zadaj meno"/>

    <EditText
      android:layout_width="185dp"
      android:layout_height="wrap_content"
      android:id="@+id/zadane_meno"/>

  </LinearLayout>

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:paddingTop="10dp">

    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Zadaj"
      android:onClick="onPosliMeno"/>

    <TextView
      android:id="@+id/sprava_od_aktivity"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Správa od aktivity"/>

  </LinearLayout>

</LinearLayout>
```

Jelikož v příkladu potřebujeme přenést nejen údaje z hlavní do sekundární aktivity, ale i ze sekundární do hlavní aktivity, která sekundární aktivitu spustila, je zapotřebí spustit sekundární aktivitu jako subaktivitu. Pokud se po spuštění očekává, že když tato aktivita skončí, vrátí volající aktivitě výsledek, namísto metody `startActivity()` je potřeba aktivitu spustit pomocí metody `startActivityForResult()`. Výsledek se nastaví pomocí metody `setResult()` v metodě `finish()`. V aktivitě, která očekává výsledek od jiné aktivity, je potřeba předefinovat metodu `onActivityResult()`.

Kód hlavní aktivity *MainActivity.java*:

```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }


  public void onClick(View view) {
    Intent iMeno = new Intent(this, Second.class);
    // očakávame že aktivita vrátihodnotu
    final int result = 1;
    iMeno.putExtra("volajúcaAktivita", "MainActivity");
    startActivityForResult(iMeno, result);
  }

  @Override
  protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    TextView tvMeno = (TextView) findViewById(R.id.meno_od_volanej_aktivity);
    String sMeno = data.getStringExtra("ZadaneMeno");
    tvMeno.append(" " + sMeno);
  }
}
```

Kód sekundární aktivity *Second.java*:

```
public class Second extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //vyber layout
    setContentView(R.layout.activity_second);
    //intent ktorý otvoril túto aktivitu
    Intent iVolajucejAktivity = getIntent();
    // načítanie odoslaných údajov
    String previousActivity =
      iVolajucejAktivity.getExtras().getString("volajúcaAktivita");
```

```
    TextView callingActivityMessage = (TextView)
    findViewById(R.id.sprava_od_aktivity);

    callingActivityMessage.append(" " + previousActivity);
  }

  public void onPosliMeno(View view) {
    // mano z prvku EditText
    EditText usersNameET = (EditText) findViewById(R.id.zadane_meno);
    String usersName = String.valueOf(usersNameET.getText());
    // intent pre návrat hodnoty volajúcej aktivite
    Intent goingBack = new Intent();
    goingBack.putExtra("ZadaneMeno", usersName);
    setResult(RESULT_OK, goingBack);
    finish();
  }
}
```

Aplikační manifest v souboru *AndroidManifest.xml*:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.explicitnyintent">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".Second"></activity>
  </application>

</manifest>
```

Implicitní intenty

Intent nemá definovanou cílovou aktivitu, ale operaci, která se má provést, a proto spustí aktivitu, která může tuto operaci vykonávat. Výběr je ponechán na rozhodnutí operačního systému. Pokud je vhodných aktivit více, musí o výběru rozhodnout uživatel. Typickým příkladem je snímání fotografie nebo zobrazení dokumentu či webové stránky.

Zjednodušeně by se daly přirovnat k žádosti aktivity typu „Kdo udělá toto?“, se kterou se obrací na operační systém. Aktivita může žádat o zobrazení obrázku, webové stránky, přehrání multimédia a podobně. V praxi to funguje tak, že systém vyhledá všechny aktivity, které mohou

příslušný intent splnit, a pokud jich najde více, nabídne uživateli možnost výběru. Můžete si změnit aplikaci, která vám zobrazuje seznam kontaktů či vykoná jinou činnost. Následně je vybraná aktivita spuštěna a příslušný intent je jí odevzdán jako vstup.

Například:

```
Intent ii = new Intent(Intent.ACTION_VIEW,
  Uri.parse("http://www.dev.android.com"));
```

V souvislosti s intenty se nejčastěji používají metody:

```
intent.setClass(MainActivity.this, NewActivity.class);
```

Připomínáme, že v takovém případě aplikace musí mít příslušnou činnost, v tomto případě přístup na internet, povolený v manifestu v souboru *AndroidManifest.xml*.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Příklad – Implicitní intenty

Námět příkladu se dá vyjádřit jednou větou: „Kdo zobrazí určenou webovou stránku." Uživatelské rozhraní aplikace má jediné tlačítko, v obsluze jeho stisknutí vytvoříme implicitní intent.

Hlavní aktivita *activity_main.xml*:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
   tools:context="sk.pcrevue.llacko.implicitnyintent.MainActivity">

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Zobraz stránku"
    android:layout_marginTop="20dp"
    android:onClick="onClick"/>
</RelativeLayout>
```

Stejně triviální je kód hlavní aktivity:

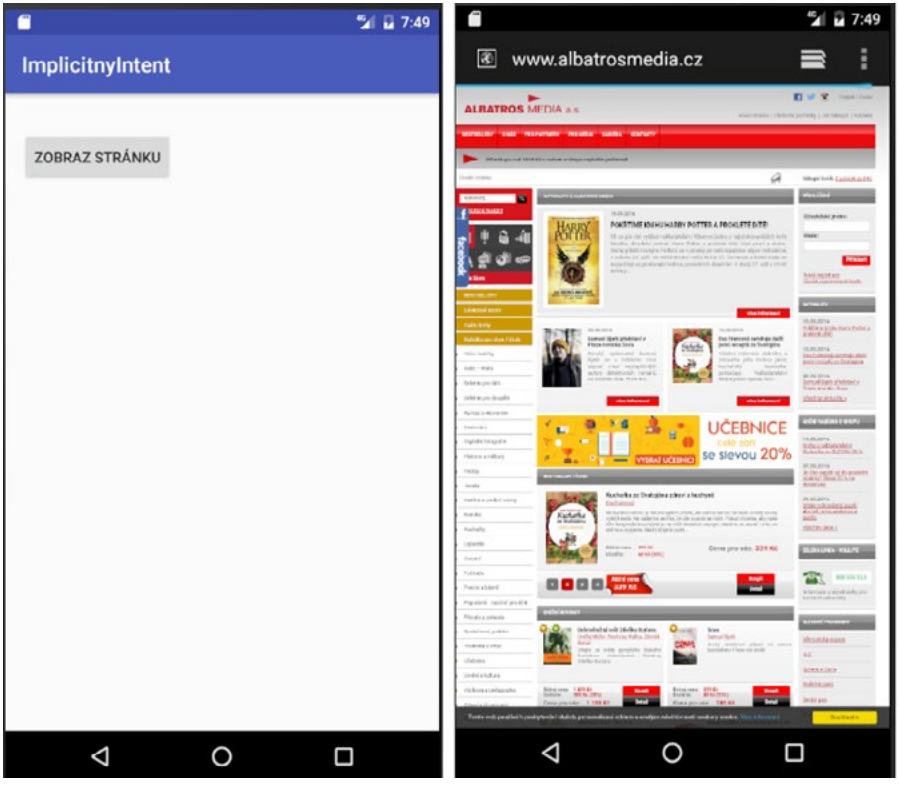
```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void onClick(View view) {
    Intent intent = new Intent(Intent.ACTION_VIEW,
      Uri.parse("http://www.albatrosmedia.cz"));

    startActivity(intent);
  }
}
```

Pokud aplikaci spustíte na emulátoru, jedinou alternativou ke splnění požadavku zobrazit obsah webové stránky je webový prohlížeč, který je součástí operačního systému.



Obrázek 7.3: Test příkladu

Zatím triviální až nudné. Pro lepší názornost reálné situace vytvoříme vlastní aplikaci, která bude také umět prohlížet webové stránky. Hlavní aktivita bude obsahovat pouze prvek `WebView`.

```
<WebView
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/webView1"/>
```

Kód hlavní aktivity:

```
<WebView
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/webView1"/>
```

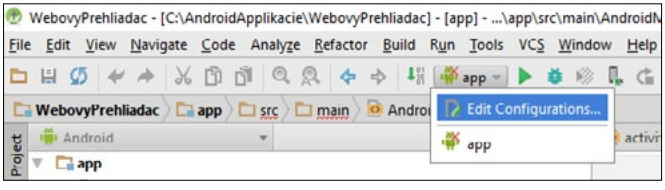
Aplikace, aktivita či komponenta mohou reagovat na různé druhy záměrů. Android však potřebuje v okamžiku spouštění aplikace vědět, na jaké záměry dokáže reagovat. Každá aplikace má ve svém manifestu v souboru *AndroidManifest.aml* takzvaný `Intent-Filter`, který definuje typy záměrů, jež dokáže příslušná aplikace obsloužit.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.webovyprehliadac">

  <uses-permission android:name="android.permission.INTERNET" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:scheme="http" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

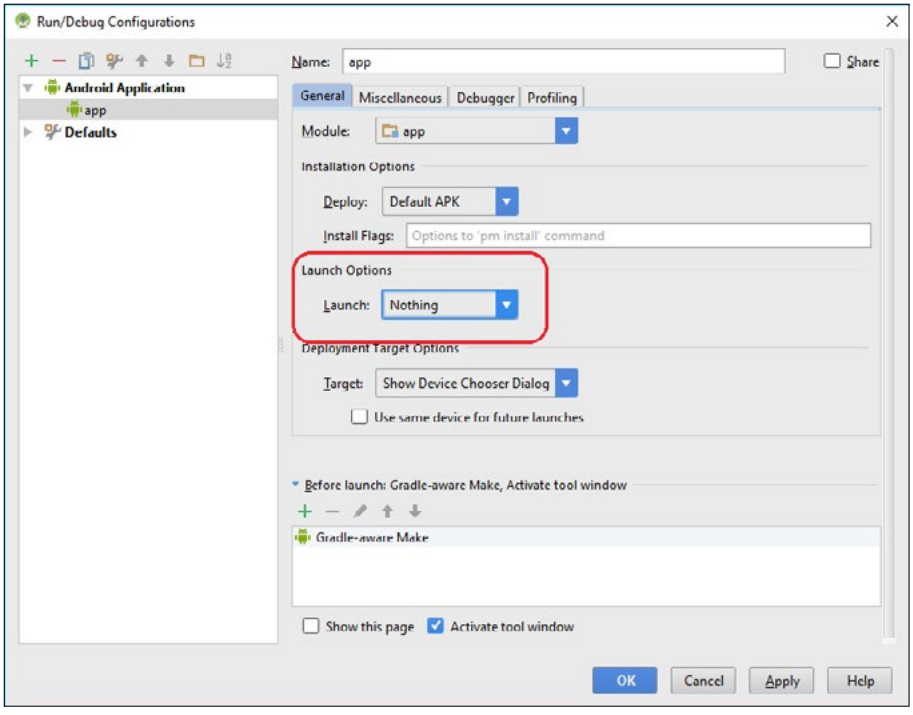
V tomto příkladu však nepotřebujeme, aby Android Studio aplikaci spustilo, ale aby ji do emulátoru či reálného zařízení nainstalovalo. Proto potřebujeme změnit konfiguraci spouštění.



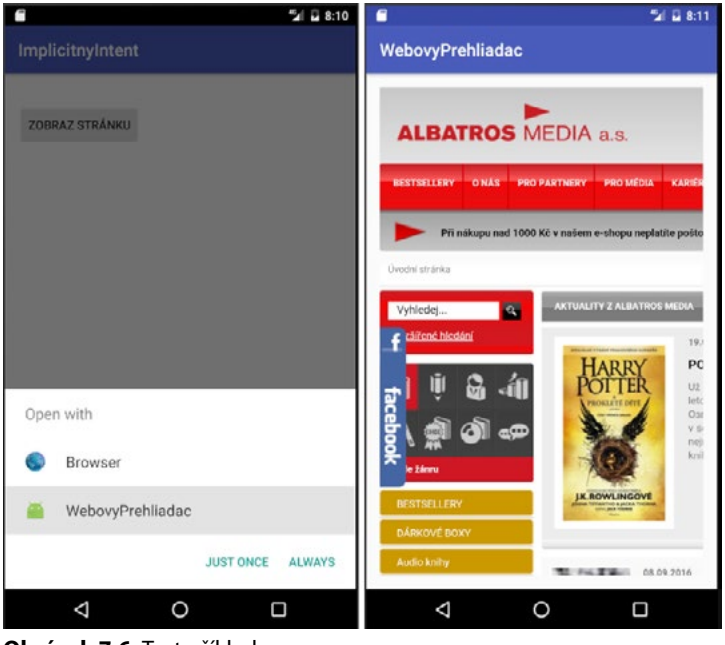
Obrázek 7.4: Změna konfigurace, instalace namísto spuštění

V dialogu spuštění a ladění aplikace nastavte parametr **Launch Option** na hodnotu **Nothing**. Klepnutím na zelenou šipku na horní liště tlačítek se potom aplikace nespustí, ale nainstaluje.

Následně znovu spusťte na emulátoru nebo reálném zařízení aplikaci ImplicitnyIntent. Jelikož je k dispozici více alternativ, jak zobrazit webovou stránku s adresou URL, operační systém nechá vybrat uživatele. Tento výběr může být jednorázový nebo trvalý.



Obrázek 7.5: Změna způsobu spuštění



Obrázek 7.6: Test příkladu

KAPITOLA

8

Interakce s uživatelem

Na rozdíl od klasických desktopových aplikací pro platformy Windows, macOS a Chrome OS, které jsou koncipované na ovládání klávesnicí a myší, případně ekvivalentem myši, telefony a tablety s operačním systémem Android se ovládají prostřednictvím dotykového displeje. Proto musí být filozofie ovládání, nabídka funkcí a zadávání údajů přizpůsobeny dotykové obrazovce jako primární metodě ovládání.

Podpora externí klávesnice je samozřejmostí. Externí klávesnice je však volitelné příslušenství, takže implicitně musíte předpokládat, že ji uživatel nemá, a ovládání pomocí kurzoru myši je potřeba chápat jako doplňkové pro přesnější ukazování. Podobně je to i s perem s ostrým hrotem umožňujícím psát rukou nebo kreslit tenké přesné čáry.

Co se týká myši, Android ji na rozdíl od konkurenčního iOS podporuje. Zařízení, která mají na externí nebo zabudované klávesnici touchpad, mají implementovaný i kurzor myši. Některá zařízení disponují trackballem.

Dotyky a gesta

Implicitní ovládání platformy Android se realizuje pomocí dotyků a gest na dotykové obrazovce. Moderní zařízení umožňují snímat dotyky více prstů včetně dynamiky a tlaku.

Ovládání prstem je intuitivní, bezprostřední. Na rozdíl od dotykového pera, jehož tenký hrot zasáhne plochu přibližně 3×3 pixely, prst zasáhne plochu zhruba desetkrát větší. Nejlépe si to představíte přiložením prstu na některou ikonu a porovnáním rozměrů. Situaci kompatibility uživatelského rozhraní navíc komplikuje škálovatelnost rozměrů displeje od nejmenších mobilních telefonů po tablety. Při návrhu designu aplikace tak musíte brát v úvahu nejen nejnižší verzi operačního systému, na které ještě aplikace poběží, ale i velikost displeje. A aby to bylo ještě komplikovanější, Android umožňuje jako standardní funkci přiblížení/oddálení obrazového kontextu pomocí gesta (oddalování a přibližování dvou prstů).

Témata kapitoly:

- **Dotyky a gesta**
- **Příklad – Zobrazení událostí dotyků**
- **Příklad – Zobrazení vícenásobných dotyků**
- **Detekce standardně používaných gest**
- **Příklad – Detekce gest**
- **Příklad – Gesto pinch**
- **Příklad – Univerzální počítadlo**
- **Příklad – Využití vlastních gest aplikace**
- **Floating Action Button**
- **Nabídka funkcionality – menu**
- **Možnosti aplikační lišty App Bar**
- **Navigace pomocí menu**
- **Rozdělení obsahu na karty**
- **Skrývání aplikační lišty**
- **Využití vysouvacího panelu Navigation Drawer**

Multidotykový displej dokáže zaregistrovat a sledovat více dotyků ve stejném čase. Pokud se jedná o gesto, každý zdroj dotyku – nejčastěji prst – zanechává virtuální dotykovou stopu. Proto je každý z těchto dotykových zdrojů reprezentován jako ukazatel. Pokud systém identifikuje nový zdroj dotyku, vytvoří pro něj jedinečný identifikátor, který je k dispozici po dobu aktivity zdroje dotyku. V některých případech Android vytvoří více ukazatelů v rámci jedné události pohybu. Ke každému ukazateli v rámci události pohybu je možné přistupovat prostřednictvím jeho indexu.

Je důležité si uvědomit, že index není totožný s identifikátorem ukazatele. Identifikátor je konstantní po celou dobu aktivity ukazatele, index, přes který jsou údaje ukazatele dostupné, nemusí být k dispozici.

Na zpracování pohybových gest z různých vstupních zařízení se využívá třída `MotionEvents`. Obsahuje kód akce, tedy gesta a jeho parametry, tedy souřadnice pohybu, jeho dynamiku, tlak a podobně.

Kódy akcí jsou:

- `ACTION_DOWN` – první prst se dotkl obrazovky
- `ACTION_POINTER_DOWN` – předtím už nastala událost `ACTION_DOWN`, nyní se další prst dotkl obrazovky
- `ACTION_POINTER_UP` – předtím už nastaly události `ACTION_POINTER` a `ACTION_POINTER_DOWN`, nyní se jeden z prstů přestal dotýkat obrazovky
- `ACTION_MOVE` – některé z prstů, které se dotýkají obrazovky, změnily svou polohu
- `ACTION_UP` - poslední z prstů, které se dotýkaly obrazovky, se jí přestal dotýkat
- `ACTION_CANCEL` – něco způsobilo zrušení aktuálního gesta

Android se při interpretaci dotyků snaží o zachování konzistence, to znamená, že dotyky následují jeden po druhém a budou se pohybovat ve skupině.

Ke zpracování pohybu prstů na dotykové obrazovce můžete použít některé z následujících metod:

- `getActionMasked()` – metoda vrací kód akce spojené s pohybem
- `getActionIndex()` – metoda vrátí index ukazatele spojeného s kódem akce; takto můžete zjistit index ukazatele, který se právě dotkl obrazovky
- `getPointerId(int pointerIndex)` – vrátí identifikátor ukazatele se zadaným indexem
- `GetPointerCount()` – metoda vrátí počet ukazatelů zainteresovaných na aktuálním pohybu
- `getX(int pointerIndex)` – vrátí X-ovou souřadnici ukazatele se zadaným indexem
- `getY(int pointerIndex)` – vrátí Y-ovou souřadnici ukazatele se zadaným indexem
- `findPointerIndex (int pointerId)` – metoda vrací index spojený s identifikátorem ukazatele

Pokud se prvek typu `View`, respektive jeho vizuální reprezentace, stane objektem dotyku nebo pohybu, volá se metoda `View.onTouchEvent(MotionEvent event)`. Metoda vrátí hodnotu `true`, pokud byl `MotionEvent` zpracován. V opačném případě vrátí hodnotu `false`.

Události spojené s dotyky jsou zpracovány přes listener. `View.OnTouchListener` definuje metody callback, které budou volané při událostech, jako je stisknutí, uvolnění nebo přetažení prstu po obrazovce. Listener se registruje pomocí metody `View.setOnTouchListener()`.

```
FrameLayout frame
frame.setOnTouchListener(
  new RelativeLayout.OnTouchListener() {
    public boolean onTouch(View v, MotionEvent m) {
      // akcia, vracia true;
    }
  }

  public boolean onTouch(View v, MotionEvent m) {
    int nUkazovatelov = m.getPointerCount();
    int ukazovatelId = m.getPointerId(0);
    return true;
  }
```


POZNÁMKA

Metoda `onTouch` se volá dříve, než je událost související s dotykem „doručena" příslušnému prvku `View`, na kterém se dotyk uskutečnil.

V jednoduchých aplikacích můžete zpracovávat jednotlivé dotykové události nezávisle na sobě. Moderní aplikace často potřebují zpracovávat komplexně více dotyků, které jsou součástí složitějšího gesta.

Jednoduché klepnutí dokážete identifikovat jako kombinaci akcií `ACTION_DOWN` a `ACTION_UP`. V případě poklepání se tato sekvence zopakuje dvakrát, tedy jako posloupnost akcí `ACTION_DOWN`, `ACTION_UP`, `ACTION_DOWN` a `ACTION_UP` v rychlém sledu.

V následující tabulce popíšeme posloupnost akcí u složitějšího gesta. Gesto se dá popsat jednoduše – položíme dva prsty na obrazovku a posuneme jimi. Z hlediska snímání se jeden prst dotkne obrazovky jako první. V tabulce ho označíme jako prst1:

Popis	Kód akce	ID
Prst1 – dotyk	ACTION_DOWN	0
Prst2 – dotyk	ACTION_DOWN	1
Prst1 a Prst 2 – posun	ACTION_MOVE	0,1
Prst1 – oddálení	ACTION_POINTER_UP	0
Prst2 – oddálení	ACTION_UP	1

Příklad – Zobrazení událostí dotyků

V prvním příkladu budeme indikovat události související s dotyky. Uživatelské rozhraní bude obsahovat dva prvky `TextView` v horní části obrazovky a celá plocha obrazovky bude k dispozici na dotyky a gesta. Prvky jsou vertikálně zapouzdřeny v kontejneru typu `LinearLayout`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
```

```
    android:id="@+id/activity_main"
    tools:context="sk.pcrevue.llacko.multidotyk1.MainActivity">

    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="stav1"
      android:id="@+id/tw1"
    />
    <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="stav2"
      android:id="@+id/tw2"
    />
  </LinearLayout>
```

Klíčové dějství aplikační logiky se odehrává v metodě na obsluhu událostí od dotyků a pohybů po dotykové obrazovce `ObsluhaDotyku()`. Všimněte si, jak se zjišťuje poloha dotyků, jejich identifikátory a kódy aktuálních akcí.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    LinearLayout lLayout =
      (LinearLayout)findViewById(R.id.activity_main);

    lLayout.setOnTouchListener(
      new LinearLayout.OnTouchListener() {
      public boolean onTouch(View v,
        MotionEvent me) {
        ObsluhaDotyku(me);
        return true;
      }
    }
    );

  }

  void ObsluhaDotyku (MotionEvent me)
  {
    TextView tw1 = (TextView)findViewById(R.id.tw1);
    TextView tw2 = (TextView)findViewById(R.id.tw2);

    int pointerCount = me.getPointerCount();
```

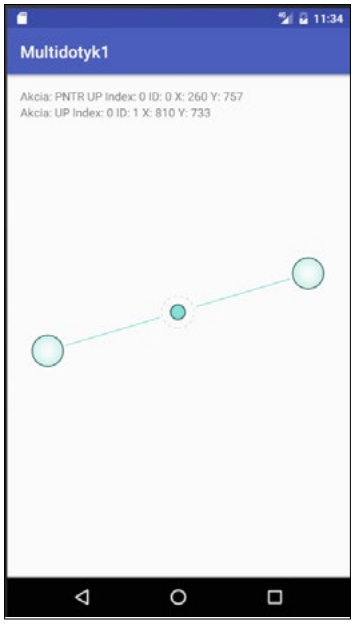
```
      for (int i = 0; i < pointerCount; i++)
      {
        int x = (int) me.getX(i);
        int y = (int) me.getY(i);
        int id = me.getPointerId(i);
        int akcia = me.getActionMasked();
        int nAI = me.getActionIndex();
        String sAkcia;


        switch (akcia)
        {
          case MotionEvent.ACTION_DOWN:
            sAkcia = "DOWN";
            break;
          case MotionEvent.ACTION_UP:
            sAkcia = "UP";
            break;
          case MotionEvent.ACTION_POINTER_DOWN:
            sAkcia = "PNTR DOWN";
            break;
          case MotionEvent.ACTION_POINTER_UP:
            sAkcia = "PNTR UP";
            break;
          case MotionEvent.ACTION_MOVE:
            sAkcia = "MOVE";
            break;
          default:
            sAkcia = "";
        }

        String sStav = "Akcia: " + sAkcia + " Index: " + nAI + " ID: " + id +
          " X: " + x + " Y: " + y;

        if (id == 0) tw1.setText(sStav);
        else         tw2.setText(sStav);
      }
    }
  }
```

Při spuštění na emulátoru doporučujeme po prvním dotyku stisknout klávesu **Ctrl** – tak budete moct emulovat myší dotyk dvěma prsty. Až si aplikaci vyzkoušíte, doporučujeme znovu si přečíst část o významu kódů jednotlivých akcí souvisejících s dotykovým ovládáním.



Obrázek 8.1: Test aplikace znázorňující dotyky a pohyby prstů na dotykové obrazovce. Emulátor po stisknutí klávesy Ctrl znázorňuje multidotyk

Příklad – Zobrazení vícenásobných dotyků

V dalším příkladu budeme indikovat vícenásobné dotyky na obrazovce. Podobně jako se zaběhla aplikace s námětem „Hello World", první aplikace začátečníka pro vývoj na dané platformě, i pro první pokusy s dotykovým ovládáním se zaběhla aplikace indikující dotyky pomocí barevných kruhů. Aplikace nakreslí větší kruh v místě, kde se uživatel dotkne prstem obrazovky. Barva kruhu je zvolena náhodně. Pokud uživatel pohybuje prstem po obrazovce, kruh se přemisťuje. Pokud se uživatel dotýká obrazovky na více místech, zobrazí se více menších kruhů. Čím více dotyků je realizováno souběžně, tím menší kruhy budou.

V layoutu hlavní aktivity využijeme pro vykreslování `FrameLayout`:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/plocha"
  tools:context="sk.pcrevue.llacko.multidotyk2.MainActivity">

</FrameLayout>
```

Ve složce *drawable* definujeme v souboru *znacka.xml* grafický prvek kruh, který se bude vykreslovat:

```
<shape
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="oval">
  <size android:width="10dip" android:height="10dip"/>
  <stroke android:color="#FFFF" android:width="2dip"
```

```
    android:dashWidth="1dip" android:dashGap="4dip"/>
</shape>
```

Vytvořte třídu `DotekView`, která bude mít na starosti vizuální reprezentaci dotyku pomocí kruhu s náhodně vygenerovanou barvou. Třída eviduje i počet současných dotyků a podle tohoto počtu nepřímo úměrně určuje průměr kruhu. Všimněte si, že počet dotyků nemůže být nula, protože se tímto číslem dělí. Pro jednoduchost a přehlednost můžete kód třídy `DotekView` umístit v souboru s kódem hlavní aktivity.


```
private class DotykView extends View
{
  private float xx, yy;
  final static private int MAX_SIZE = 400;
  private int nDotykov = 0;
  final private Paint mPaint = new Paint();

  public DotykView(Context context, float x, float y)
  {
    super(context);
    xx = x;
    yy = y;
    mPaint.setStyle(Paint.Style.FILL);
    Random rnd = new Random();
    mPaint.setARGB(255, rnd.nextInt(256), rnd.nextInt(256),
      rnd.nextInt(256));
  }

  float getXLoc() {return xx;}
  void setXLoc(float x) {xx = x;}
  float getYLoc() {return yy;}
  void setYLoc(float y) {yy = y;}

  void pocetDotykov (int nDotykov) {nDotykov = nDotykov;}

  @Override
  protected void onDraw(Canvas canvas)
  {
    canvas.drawCircle(xx, yy, MAX_SIZE / (nDotykov+1), mPaint);
  }
}
```

V kódu hlavní aktivity je potřeba ve statických polích udržovat údaje o aktuálních dotycích. Pole se dynamicky plní v metodě `onTouch()`.

- Pokud je kód akce `ACTION_DOWN` nebo `ACTION_POINTER_DOWN`, potom přibyl první nebo další dotyk. Tehdy se vytvoří nový kruh na označení aktuálního místa dotyku.
- Pokud je kód akce `ACTION_UP` nebo `ACTION_POINTER_UP`, znamená to, že uživatel zvedl některý prst z obrazovky. Tehdy odstraníme odpovídající kruh. Zjistíme jeho index a následně ID.
- Pokud je kód akce `ACTION_MOVE`, překreslí se kruhy odpovídající dotykům prstů, které se pohnuly. V aplikaci inicializujeme překreslení, jen pokud je pohyb větší než 3 dp.

```
public class MainActivity extends AppCompatActivity {

  private static final int MIN_DELTA = 2;

  final private static LinkedList<DotykView> neaktivneDotyky =
    new LinkedList<DotykView>();
  @SuppressLint("UseSparseArrays")
  final private static Map<Integer, DotykView> aktivneDotyky =
    new HashMap<Integer, DotykView>();

  private FrameLayout frame;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    frame = (FrameLayout) findViewById(R.id.plocha);
    initViews();

    frame.setOnTouchListener(new View.OnTouchListener()
    {
      @Override
      public boolean onTouch(View v, MotionEvent me)
      {
        switch (me.getActionMasked())
        {
          case MotionEvent.ACTION_DOWN:
          case MotionEvent.ACTION_POINTER_DOWN:
          {
            int index = me.getActionIndex();
            int id = me.getPointerId(index);

            DotykView dw = neaktivneDotyky.remove();
            if (dw != null)
            {
              aktivneDotyky.put(id, dw);
              dw.setXLoc(me.getX(index));
              dw.setYLoc(me.getY(index));
              aktualizujDotyky(aktivneDotyky.size());
              frame.addView(dw);
            }
            break;
          }

          case MotionEvent.ACTION_UP:
          case MotionEvent.ACTION_POINTER_UP:
          {
            int index = me.getActionIndex();
            int id = me.getPointerId(index);
```

```
            DotykView dw = aktivneDotyky.remove(id);
            if (dw != null)
            {
              neaktivneDotyky.add(dw);
              aktualizujDotyky(aktivneDotyky.size());
              frame.removeView(dw);
            }
            break;
          }

          case MotionEvent.ACTION_MOVE:
          {
            for (int idx = 0; idx < me.getPointerCount();
            idx++)
            {
              int ID = me.getPointerId(idx);

              DotykView dw = aktivneDotyky.get(ID);
              if (null != dw)
              {

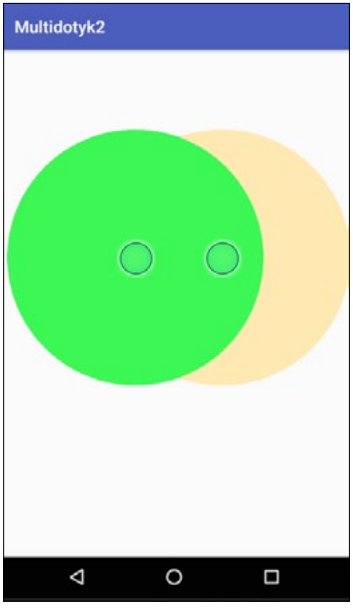
                // pouze pri pohybu nad minimalni vzdalenost
                if (Math.abs(dw.getXLoc() -me.getX(idx)) > MIN_DELTA
                  || Math.abs(dw.getYLoc()- me.getY(idx)) > MIN_DELTA)
                  {
                  dw.setXLoc(me.getX(idx));
                  dw.setYLoc(me.getY(idx));
                  dw.invalidate();
                }
              }
            }
            break;
          }
          default:
        }
        return true;
      }

      private void aktualizujDotyky(int numActive)
      {
        for (DotykView dw : aktivneDotyky.values())
          dw.pocetDotykov(numActive);
      }
    });
  }

  private void initViews()
  {
    for (int idx = 0; idx < 10; idx++)
    {
      neaktivneDotyky.add(new DotykView(this, -1, -1));
```

```
        }
      }
    }
```

Aplikaci otestujte tak, že nejprve přiložíte jeden prst na obrazovku a pohybujte jím. Sledujte překreslování kruhu. Potom přiložte další prsty a pohybujte jimi po obrazovce.



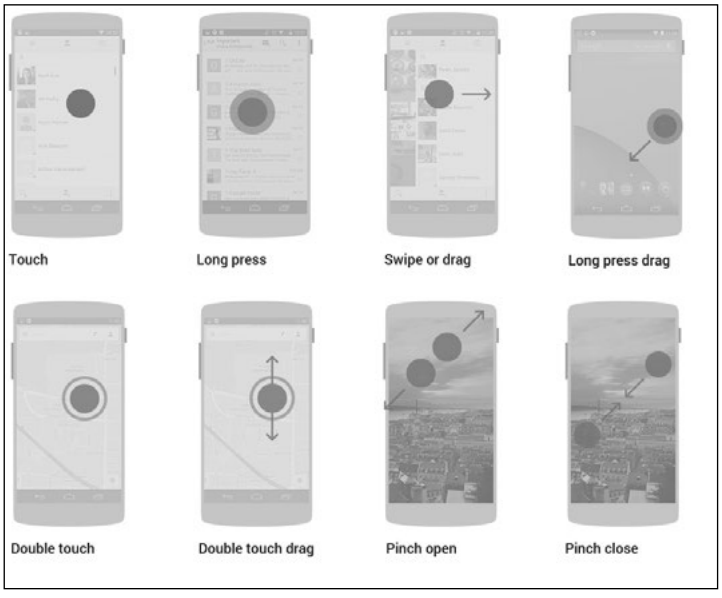
Obrázek 8.2: Test aplikace znázorňující vícenásobné dotyky

Detekce standardně používaných gest

V předchozím příkladu jsme ukázali detekci vícenásobných dotyků včetně pohybu jednotlivých prstů po obrazovce. Android využívá pro některé úkony standardní gesta, na které si uživatelé zvykli a očekávají, že je budou využívat i v aplikacích.

V pojetí operačního systému Android začíná gesto v bodě, kterého se na obrazovce dotkl první prst, pero nebo jiné polohovací zařízení. Končí tehdy, když poslední prst nebo polohovací zařízení opustí povrch displeje.

Na detekci gest se využívá třída `GestureDetector`. Je potřeba vytvořit instanci této třídy a implementovat rozhraní `GestureDetector.OnGestureListener`. V aktivitě přepíšete metodu `onTouchEvent()`.



Obrázek 8.3: Zaběhlá gesta a jejich významy na platformě Android

Příklad – Detekce gest

Podobně jako v předchozí části začneme příkladem, který identifikuje gesta na obrazovce a vypíše, jaké gesto bylo právě realizováno. Využijete prakticky beze změny uživatelské rozhraní vytvořené Android Studiem při vytváření projektu. Jedinou úpravou je přidat ID prvku `TextView`.

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/tw1"
  android:text="Hello World!" />
```

V kódu hlavní aktivity je potřeba vytvořit callback metody pro gesta `onFling()`, `onDown()`, `onScroll()`, `onShowPress()`, `onSingleTapUp()` a `onLongPress()`. V těle metody vypíšeme její název, abychom věděli, k jakým událostem dochází. Je potřeba vytvořit třídu, která implementuje rozhraní `GestureDetector.OnGestureListener`, a pokud je potřeba detekce poklepání, tak i rozhraní `GestureDetector.OnDoubleTapListener`. Mohou to být samostatné třídy nebo – jako v našem příkladu – třídy realizované v rámci třídy hlavní aktivity aplikace.

Kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity
  implements GestureDetector.OnGestureListener,
  GestureDetector.OnDoubleTapListener{

  private TextView tw1;
  private GestureDetectorCompat gDet;

  @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  tw1 = (TextView)findViewById(R.id.tw1);
  this.gDet = new GestureDetectorCompat(this,this);
  gDet.setOnDoubleTapListener(this);
}

@Override
public boolean onTouchEvent(MotionEvent e) {
  this.tw1.onTouchEvent(e);
  return super.onTouchEvent(e);
}

@Override
public boolean onDown(MotionEvent e) {
  tw1.setText ("onDown");
  return true;
}

@Override
public boolean onFling(MotionEvent e1,  MotionEvent e2,
  float velocityX, float velocityY) {
  tw1.setText("onFling");
  return true;
}

@Override
public void onLongPress(MotionEvent e) {
  tw1.setText("onLongPress");
}

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2,
  float distanceX, float distanceY) {
  tw1.setText("onScroll");
  return true;
}

@Override
public void onShowPress(MotionEvent e) {
  tw1.setText("onShowPress");
}

@Override
public boolean onSingleTapUp(MotionEvent e) {
  tw1.setText("onSingleTapUp");
  return true;
} @Override

public boolean onDoubleTap(MotionEvent e) {
```

```
    tw1.setText("onDoubleTap"); return true;
  }

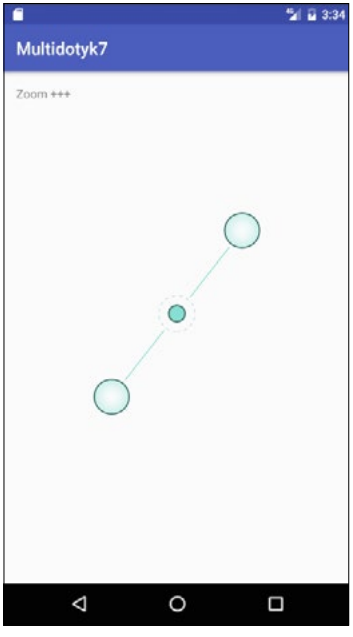
  @Override
  public boolean onDoubleTapEvent(MotionEvent e) {
    tw1.setText("onDoubleTapEvent");
    return true;
  }

  @Override
  public boolean onSingleTapConfirmed(MotionEvent e) {
    tw1.setText("onSingleTapConfirmed");
    return true;
  }

}
```


Příklad – Gesto pinch

V anketě o nejpoužívanější gesto by určitě zvítězilo gesto pinch, které spočívá v položení dvou prstů na displej a jejich následném vzájemném přiblížení a oddálení. Nejčastěji se toto gesto využívá na přibližování a oddalování. Přibližováním prstů k sobě do špičky (odtud název gesta) se obrázek – nebo jiný obsah na obrazovce – zmenšuje a opačným pohybem, kdy se prsty od sebe vzdalují, se obsah na obrazovce zvětšuje. V příkladu se budeme soustředit na identifikaci gest, takže nebudeme řešit zvětšování a zmenšování obsahu, ale pouze vypíšeme, jaké gesto uživatel provedl.



Obrázek 8.4: Test aplikace využívající gesto přibližování a oddalování prstů

Využijete prakticky beze změny uživatelské rozhraní vytvořené Android Studiem při vytváření projektu. Jedinou úpravou je přidat ID prvku `TextView`.

```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/tw1"
  android:text="Hello World!" />
```

V kódu hlavní aktivity je vytvořena třída `MojOnScaleGestureListener`. Je potřeba implementovat tři metody: `onScale()`, `onScaleBegin()` a `onScaleEnd()`. Typ gesta se vypíše v metodě `onScale()`. Metoda `onTouchEvent()` se volá při detekci dotyku nebo gesta.

Při spuštění aplikace na reálném zařízení jsou gesta dvěma prsty intuitivní. V emulátoru je můžete simulovat podržením klávesy **Ctrl** a klepnutím a posunem kurzoru myši.

Kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  TextView tw1;
  ScaleGestureDetector sgDet;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    tw1 = (TextView)findViewById(R.id.tw1);

    sgDet = new ScaleGestureDetector(this, new MojScaleGestureListener());
  }

  @Override
  public boolean onTouchEvent(MotionEvent event) {
    sgDet.onTouchEvent(event);
    return true;
  }

  public class MojScaleGestureListener extends
    ScaleGestureDetector.SimpleOnScaleGestureListener {

    @Override
    public boolean onScale(ScaleGestureDetector detector) {
      float scaleFactor = detector.getScaleFactor();
      if (scaleFactor > 1) tw1.setText("Zoom ---");
      else tw1.setText("Zoom +++");
      return true;
    }

    @Override
    public boolean onScaleBegin(ScaleGestureDetector detector) {
      return true;
    }
```

```
    @Override
    public void onScaleEnd(ScaleGestureDetector detector) {
    }
  }
}
```

Příklad – Univerzální počítadlo

Námětem aplikace bude univerzální počítadlo objektů. Jeho mechanický nebo elektromechanický ekvivalent používají například letušky či organizátoři různých setkání na sčítání návštěvníků. Z hlediska (technického) uživatelského rozhraní se bude jednat o interakci mezi dotykovým tlačítkem a výpisem obsahu proměnné, která se inkrementuje dotykem na tlačítku.

Z hlediska fungování aplikace v širších souvislostech se jedná o interakci vašeho pohledu, který spočine na osobě, jíž chcete připočítat, a impulzu pro klepnutí prstem na displej. Určitě jste si všimli, že tu není žádný prostor na pohled na displej ani hledání počítadla. Proto musí být tlačítko co největší a klepnutí by mělo být nějakým nevizuálním způsobem potvrzeno – buď zvukem, nebo vibrací. U velkého tlačítka hrozí možnost neúmyslného stisknutí při manipulaci s telefonem.

Při eliminování nechtěného dotyku se můžete inspirovat operačním systémem, konkrétně mechanizmem odemykání gestem vodorovného posuvu, které se označuje jako „swipe“ nebo „fling“. Pokud byste objekty počítali tímto gestem, eliminovalo by to problém nechtěného stisknutí tlačítka.

Layout hlavní aktivity využívá kontejner `ViewFlipper`, jelikož budeme animovat posun prvků `TextView` obsahujících číselné hodnoty počítadla. `ViewFlipper` umožňuje animované přechody mezi dvěma a více objekty typu `Views`.

```
<ViewFlipper xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/view_flipper"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context="sk.pcrevue.llacko.multidotyk5.MainActivity">

  <include
    android:id="@+id/include2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    layout="@layout/inc1" >
  </include>

  <include
    android:id="@+id/include1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    layout="@layout/inc2" >
  </include>

</ViewFlipper>
```

Správné fungování prvku `ViewFlipper` vyžaduje definici dvou vložených layoutů. Můžete je nazvat například *inc1.xml* a *inc2.xml*. Oba dva obsahují jeden prvek `TextView` zapouzdřený v kontejneru `LinearLayout`. Ve výpisu je layout *inc1.xml*. Layout *inc2.xml* se liší jen názvem prvku `TextView`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    android:gravity="center"
    android:textSize="300sp" >
  </TextView>

</LinearLayout>
```

Jelikož se zobrazení počítadla nemusí dělit o plochu displeje s tlačítkem a gesto je možné realizovat přes celou obrazovku i přes text počítadla, mohou být číslice počítadla velké, aby byly dobře čitelné.

Abychom aplikaci udělali co nejintuitivnější, posouvají se čísla zprava doleva. Na první pohled to vypadá nelogicky, zdálo by se, že lepší by bylo provádět gesta zleva doprava. Pokud však chceme vytvářet animaci, opačné pořadí je vhodnější. Představte si posloupnost číslic, například

1 2 3 4

a momentálně je zobrazeno číslo 2. Jakým gestem přesunete trojku na pozici dvojky? Přece posunem zprava doleva.

Pokud uživatel provede gesto zprava doleva – takzvané gesto fling –, bude se aktuální `TextView` posouvat vlevo mimo levý okraj obrazovky. Zároveň se za ním z pravé strany bude posouvat `TextView` s vyšším číslem. Všimněte si, že toto gesto funguje jen tehdy, když posunete prstem zprava doleva. Pokud provedete stejné gesto v opačném směru, nic se nestane.

V kódu hlavní aktivity indikujeme gesto „Fling“. Když `GestureDetector` detekuje příslušné gesto, zavolá se metoda `onFling()`. Gesto k aktivování počítadla by mělo být trochu dynamické, energické, aby se nedalo zaměnit s náhodným pomalým vodorovným posunem. Proto počítadlo inkrementujeme pouze tehdy, když rychlost posuvu prstu ve vodorovném směru překročí hodnotu 10 bodů za sekundu.

Animace pohybu se realizuje přepínáním layoutů v metodě `prepniLayout()`.

```
public class MainActivity extends AppCompatActivity {

  private ViewFlipper flipper;
  private TextView textView1, textView2;
  private int nStav, nPocitadlo;
  private GestureDetector gestDetector;
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  textView1 = (TextView) findViewById(R.id.textView1);
  textView2 = (TextView) findViewById(R.id.textView2);

  nStav = 0;
  flipper = (ViewFlipper) findViewById(R.id.view_flipper);
  textView1.setText(String.valueOf(nPocitadlo));

  gestDetector = new GestureDetector(this,
    new GestureDetector.SimpleOnGestureListener() {
    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2,
      float rychlostX, float rychlostY)  {
      if (rychlostX <-10.0f) {
        nStav = nStav == 0 ? 1: 0;
        prepniLayout(nStav);
      }
      return true;
    }
  });
}

@Override
public boolean onTouchEvent(MotionEvent event) {
  return gestDetector.onTouchEvent(event);
}

public void prepniLayout(int switchTo) {
  nStav = switchTo;

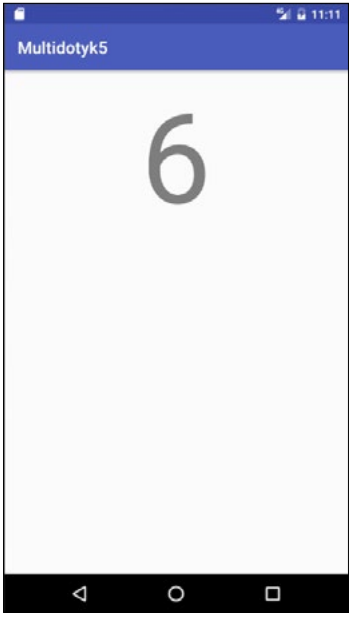
  flipper.setInAnimation(animZprava());
  flipper.setOutAnimation(animZlava());

  nPocitadlo++;
  if (switchTo == 0)
    textView1.setText(String.valueOf(nPocitadlo));
  else
    textView2.setText(String.valueOf(nPocitadlo));
  flipper.showPrevious();
}

private Animation animZprava() {
  Animation zprava = new TranslateAnimation(
    Animation.RELATIVE_TO_PARENT, +1.0f,
    Animation.RELATIVE_TO_PARENT, 0.0f,
    Animation.RELATIVE_TO_PARENT, 0.0f,
    Animation.RELATIVE_TO_PARENT, 0.0f);
  zprava.setDuration(500);
```

```
    zprava.setInterpolator(new LinearInterpolator());
    return zprava;
  }

  private Animation animZlava()  {
    Animation zlava = new TranslateAnimation(
      Animation.RELATIVE_TO_PARENT, 0.0f,
      Animation.RELATIVE_TO_PARENT, -1.0f,
      Animation.RELATIVE_TO_PARENT, 0.0f,
      Animation.RELATIVE_TO_PARENT, 0.0f);
    zlava.setDuration(500);
    zlava.setInterpolator(new LinearInterpolator());
    return zlava;
  }
}
```



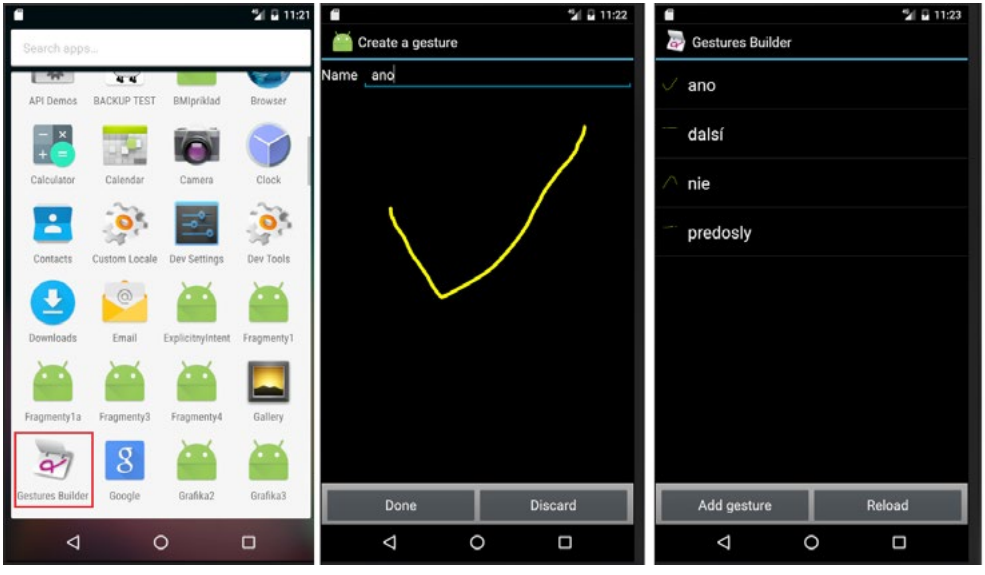
Obrázek 8.5: Test aplikace Počítadlo

Příklad – Využití vlastních gest aplikace

Ne vždy a všude vyhovují standardní gesta používaná operačním systémem. V některých aplikacích může být výhodné definovat si vlastní gesta, která korelují s úkony vykonávanými intuitivně například v „papírové" informatice. Například zkontrolování něčeho se označí fajfkou pomocí tužky nebo fixu v seznamu. Podobná gesta si můžete definovat a využívat je i ve vlastních aplikacích. Například byste mohli v předchozí aplikaci určené k počítání čehokoliv rozlišovat druhy objektů – při počítání osob jdoucích na diskotéku byste mohli rozlišovat muže a ženy.

Možná namítnete, že by se to dalo provést pomocí dvou velkých tlačítek, jenže taková animace by měla velkou nevýhodu bylo by nutné sledovat displej a klepat na správná tlačítka. Pokud byste definovali pro muže například gesto ve tvaru písmena V a pro ženy inverzní gesto, tedy ve tvaru písmena A bez spojovací čáry, mohli byste počítat a rozlišovat osoby bez nutnosti dívat se na displej. Stačí kdekoliv na displeji provést příslušné gesto.

Nejprve je potřeba definovat vlastní gesta. Součástí SDK je aplikace Gestures Builder, která je standardní aplikační výbavou emulátoru. Spusťte emulátor a najděte v seznamu aplikací ikonu **Gestures Builder**. Má tvar poznámkového bloku s řeckým písmenem alfa. Pomocí tlačítka **Add gesture** přidejte gesta, která budete v aplikaci potřebovat. V našem případě jsme definovali čtyři gesta s významem „ano“, „ne“, „další“ a „předchozí“.



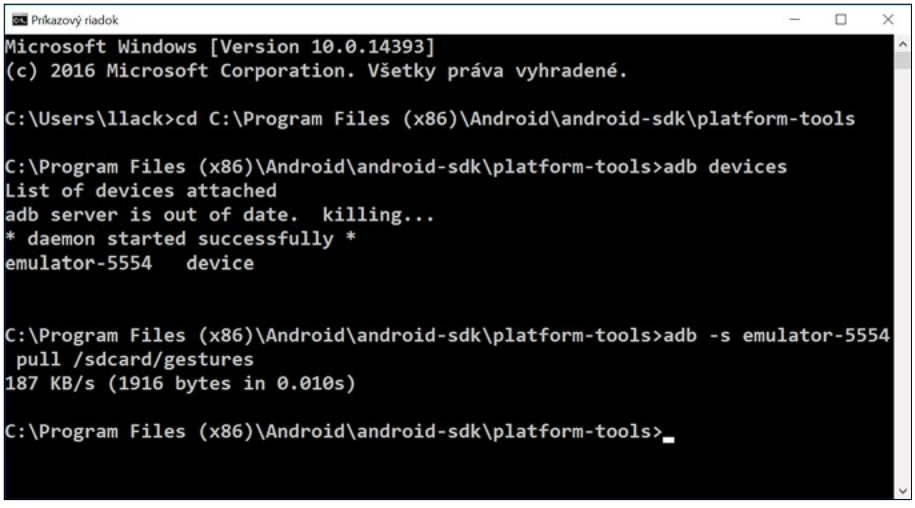
Obrázek 8.6: Definice vlastních gest v aplikaci Gestures Builder

Definice gest budou v emulátoru uloženy v souboru */storage/emulated/0/gestures*. Tento soubor je potřeba zkopírovat do projektu vaší aplikace, například do souboru */res/raw directory*.

Jak však zkopírovat soubor z emulátoru do hostitelského počítače? K tomuto účelu vám poslouží nástroj ADB. U standardní instalace ho najdete ve složce *C:\Program Files (x86)\Android\android-sdk\platform-tools*.

Ve Windows spusťte příkazový řádek a přejděte do složky s aplikací ADB. Následně potřebujete zjistit název spuštěného emulátoru. Zadejte příkaz:

```
adb devices
```



Obrázek 8.7: Zjištění názvu emulátoru a stažení souboru z jeho vnitřní paměti

V našem případě byl název emulátoru emulator-5554.

Obsah souboru s definicemi gest stáhněte příkazem

```
adb -s emulator-5554 pull /sdcard/gestures
```

Lepší, pohodlnější a přesnější je definovat gesta na reálném zařízení. Z reálného zařízení připojeného přes USB kabel stáhnete soubor s definicí gest příkazem

```
adb -d pull /sdcard/gestures
```

Soubor se uloží do pracovního adresáře aplikace **adb**, v našem případě *C:\Users\llack\AppData\Local\VirtualStore\Program Files (x86)\Android\android-sdk\platform-tools*. Snadno ho najdete, když vyhledáte soubory s klíčovým slovem `gestures`. Soubor je binární, takže jeho prohlížení nemá smysl. Soubor později, když vytvoříte projekt aplikace, zkopírujte do adresáře.

Nyní – po přípravné fázi definování uživatelských gest – vytvořte projekt aplikace.

Layout hlavní aktivity obsahuje jediný prvek `android.gesture.GestureOverlayView` navržený tak, aby zabral celou plochu obrazovky. Tento prvek zachytává gesta uživatele.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.multidotyk6.MainActivity">

  <android.gesture.GestureOverlayView
    android:id="@+id/gestures_overlay"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_centerInParent="true" >
  </android.gesture.GestureOverlayView>
</RelativeLayout>
```

V kódu hlavní aktivity se pomocí objektu `Toast` vypíše název rozpoznaného gesta. Ale nepředbíhejme. Pomocí metody `GestureLibraries.fromRawResource()` se nejprve nahraje do operačního systému uživatelova zařízení soubor s definicí speciálních gest, která bude aplikace využívat. Následně se zaregistruje listener událostí. Vyhodnocení shody gesta se vzorem se realizuje v metodě `onGesturePerformed()`.

```
public class MainActivity extends AppCompatActivity
  implements GestureOverlayView.OnGesturePerformedListener {

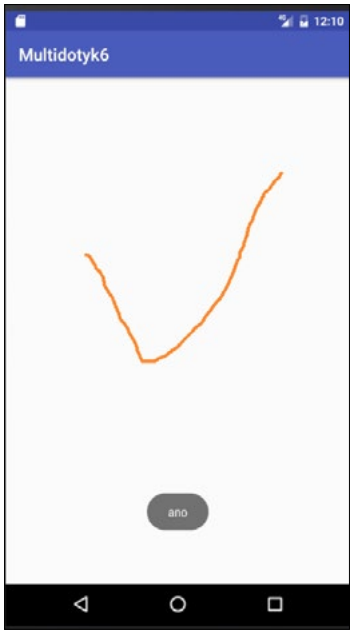
  private GestureLibrary gLibrary;

  @Override
  public void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    gLibrary = GestureLibraries.fromRawResource(this, R.raw.gestures);
    if (!gLibrary.load()) {
      finish();
    }
    GestureOverlayView gestureView = (GestureOverlayView)
      findViewById(R.id.gestures_overlay);
    gestureView.addOnGesturePerformedListener(this);
  }

  public void onGesturePerformed(GestureOverlayView overlay,
    Gesture gesture) {
    ArrayList<Prediction> predictions = gLibrary.recognize(gesture);
    if (predictions.size() > 0)  {
      Prediction prediction = predictions.get(0);
      if (prediction.score > 2.0)
        Toast.makeText(this, prediction.name, Toast.LENGTH_SHORT).show();
      else
        Toast.makeText(this, "Neznámé gesto", Toast.LENGTH_SHORT).show();
    }
  }
}
```

Do projektu je potřeba přidat soubor s definicí gest. Ve složce *res* vytvořte novou složku *raw* a do ní soubor, který jste zkopírovali z emulátoru, nakopírujte.



Obrázek 8.8: Rozpoznání speciálního gesta využívaného konkrétní aplikací

Floating Action Button

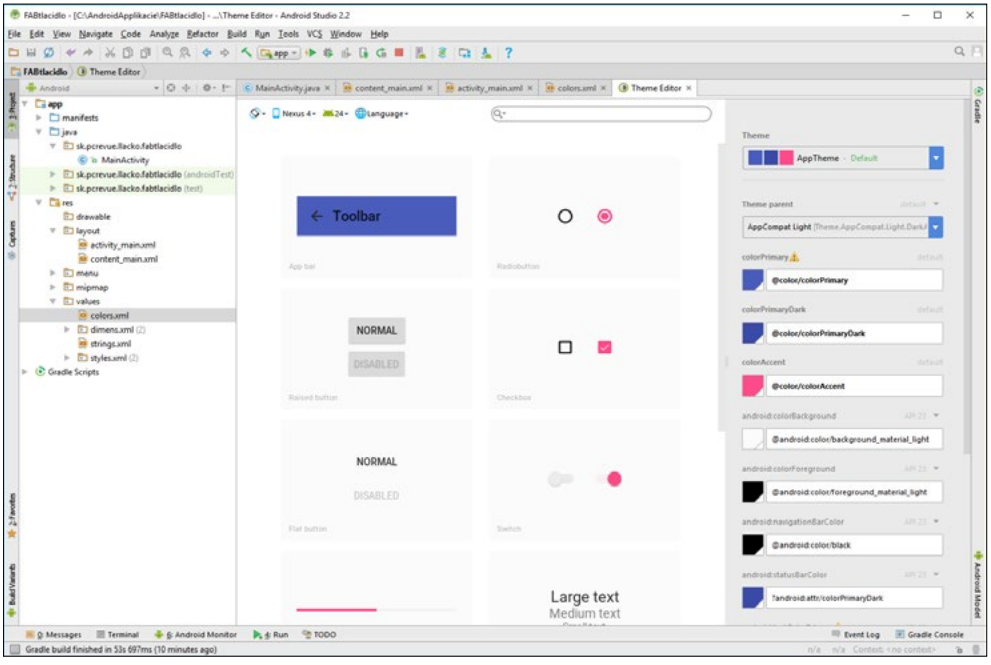
Určitě jste si při vytváření projektů s hlavní aktivitou typu **Basic Activity** všimli, že vývojové prostředí přidá do projektu i takzvané plovoucí tlačítko, v původní terminologii **Floating Action Button**. Překlad plovoucí nebo vznášející se je velmi výstižný, protože tlačítko tohoto typu nebývá zobrazeno stále, ale ve vhodné chvíli se zobrazí, jako kdyby se vznášelo nad povrchem uživatelského rozhraní aplikace. Typickým příkladem je příchozí zpráva v aplikaci Messenger, tlačítko na vytvoření nového kontaktu, napsání nové e-mailové zprávy a podobně.

Tlačítko tohoto typu musí být kruhové s rozměry 56 × 56 dp pro standardní velikost, případně 40 × 40 dp pro velikost mini. Mělo by být umístěno minimálně 16 dp od okraje obrazovky na telefonech a 24 dp na tabletech a jiných zařízeních s velkou obrazovkou. Bez ohledu na velikost musí tlačítko obsahovat vnitřní ikonu s rozměry 24 × 24 dp. Designová pravidla doporučují, aby obrazovka uživatelského rozhraní měla pouze jedno takovéto tlačítko.

Vytvořte projekt s hlavní aktivitou typu **Basic Activity**. Tlačítko **Floating Action Button** je definováno v souboru *activity_main.xml* takto:

```
<android.support.design.widget.FloatingActionButton
  android:id="@+id/fab"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="bottom|end"
  android:layout_margin="@dimen/fab_margin"
  app:srcCompat="@android:drawable/ic_dialog_email" />
```

Všimněte si, že uvnitř tlačítka je standardní Android ikona `@android:drawable/ic_dialog_email` a složka *drawable* v projektu žádnou ikonu neobsahuje. Pokud potřebujete do tlačítka umístit vlastní ikonu, umístěte ji do složky *drawable* a v definičním souboru použijte ve tvaru `@drawable/ic_moje_ikona`. Barva pozadí ikony je definovaná parametrem `accentColor` v souboru *colors.xml*. Soubor bude ve složce *res/layout*. Barvy v tomto souboru jsou součástí barevného schématu aplikace. Můžete ji změnit pomocí nabídky **Tools → Android → Theme Editor**.



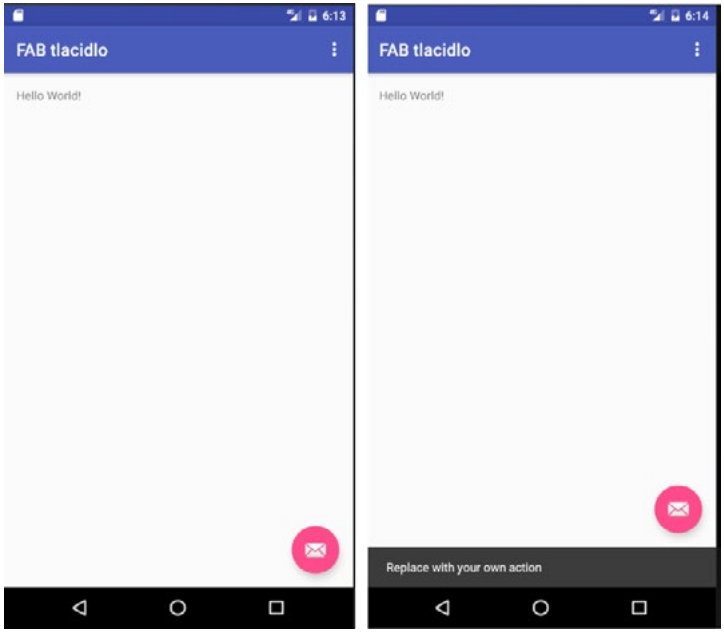
Obrázek 8.9: Theme Editor

Obsluha události stisknutí tlačítka je definovaná v souboru *MainActivity* v metodě `onCreate`:

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
  setSupportActionBar(toolbar);

  FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
  fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
      Snackbar.make(view, "Replace with your own action",
      Snackbar.LENGTH_LONG)
      .setAction("Action", null).show();
    }
  });
}
```

Všimněte si, že aplikace využívá při obsluze události stisknutí plovoucího tlačítka zajímavý a užitečný komponent `Snackbar`. Tento komponent umožňuje zobrazit uživateli informace ve formě panelu, který se zobrazí v dolní části obrazovky. Panel může obsahovat stručnou textovou zprávu a volitelné tlačítko akce, která aktivuje požadovanou akci. `Snackbar` se skryje buď automaticky, nebo po nastavení časového limitu, případně se dá odstranit gestem posunu.



Obrázek 8.10: Tlačítko Floating Action Button a panel Snackbar (vpravo dole)

Nabídka funkcionality – menu

Nabídka funkcionality může být v aplikaci pro Android uživateli prezentována několika způsoby. K dispozici jsou tři typy menu (nabídek):

- **Options menu** – zobrazí se, pokud uživatel stiskne tlačítko menu na aplikační liště (nebo hardwarové tlačítko na starších zařízeních). Tato nabídka slouží k navigaci uvnitř aplikace.
- **Kontextové menu** – váže se k prvku uživatelského rozhraní. Zobrazí se zpravidla podržením prstu na příslušném prvku.
- **Pop-up menu** – na rozdíl od kontextového menu, které aktivuje uživatel, pop-up menu se zobrazí samo v návaznosti na aplikační logiku, pokud je vyžadována součinnost uživatele. Menu je vizuálně ukotveno k prvku, ke kterému se váže. Navigujte uživatele napříč aplikací.

POZNÁMKA

Společným jmenovatelem je schopnost poskytnout uživateli rychlý a co nejjednodušší přístup k požadované funkcionalitě v souladu s designovým záměrem aplikace.

Menu se často využívá i na přepínání mezi aktivitami tvořícími aplikaci z vůle uživatele. Při návrhu uživatelského rozhraní potřebujete navigovat uživatele z hlavní aktivity, která se zobrazí po spuštění aplikace, do jiných aktivit a zpravidla i zpět do hlavní aktivity.

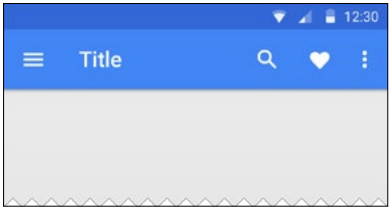
Možnosti aplikační lišty App Bar

Při pohledu na obrazovku se spuštěnou aplikací je v horní části stavová lišta, která se zobrazuje téměř stále, s výjimkou situace, kdy je některá aplikace spuštěna v celoobrazovkovém režimu. Na stavové liště najdete ikony připojení, stavu signálu, stavu baterie, aktuální čas, případně ikonky důležitých notifikací.

POZNÁMKA

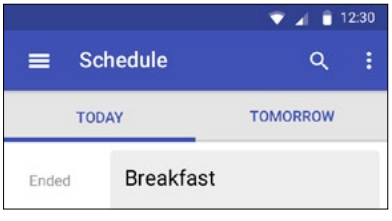
Stavová lišta je v režii operačního systému.

Pod stavovou lištou je aplikační lišta, která je už v režii aplikace. Úplně vpravo zpravidla obsahuje ikonu se třemi tečkami nad sebou, která aktivuje menu, vlevo může být ikona nabídky typu Navigation Drawer. Dominantním prvkem je obvykle název aplikace. V pravé části se často umísťují ikony nejčastěji používaných funkcí, aby se daly aktivovat přímo na jeden dotyk a nebylo nutné vybírat je z nabídky.



Obrázek 8.11: Stavová lišta a aplikační lišta

U aplikací se složitější zobrazovací a navigační strukturou jsou pod aplikační lištou ještě lišty nástrojů a záložek.

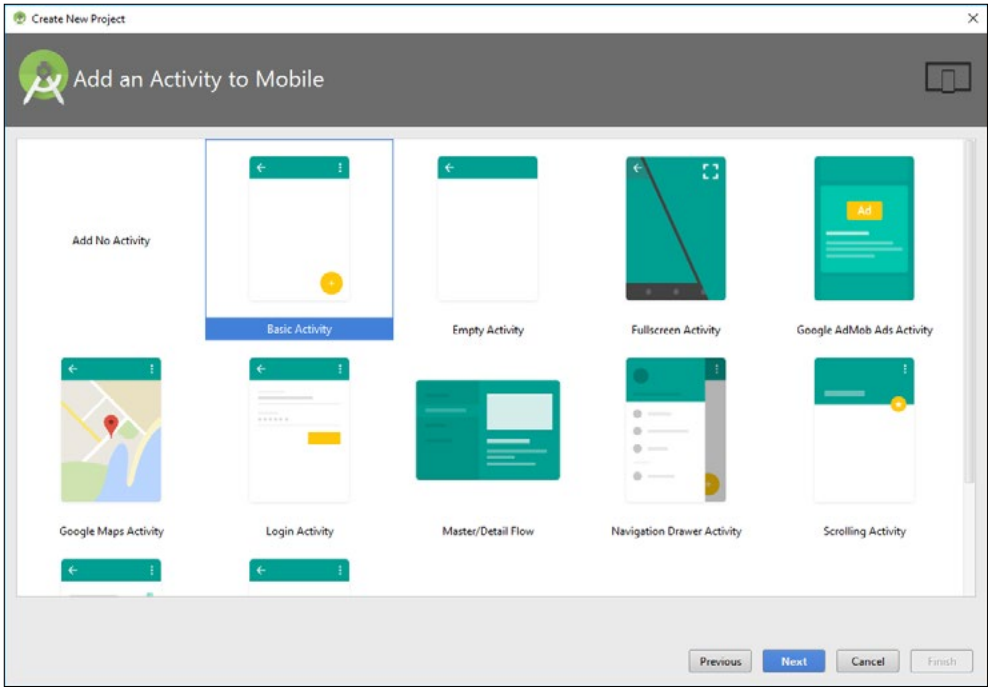


Obrázek 8.12: Lišty nástrojů a záložek

Navigace pomocí menu

Verze Android 3.0 Honeycomb přinesla aplikační lištu App Bar s tlačítky v horní části každé obrazovky k aktivování nejpoužívanějších funkcí. Tato lišta je perzistentní napříč celou aplikací, takže umožňuje konzistentní navigaci a přepínání zobrazení v rámci aplikace. Aplikační lišta je pro uživatele jakýmsi záchytným bodem, standardizovaným prvkem umožňujícím rychle si osvojit ovládání aplikace. Operační systém navíc aplikační lištu elegantně přizpůsobí různým konfiguracím obrazovky.

Vytvořte projekt aplikace, přičemž nastavte jako typ hlavní aktivity **Basic Activity**. Aplikační lišta je totiž už součástí této šablony aktivity. Aby byl projekt co nejjednodušší, odstraňte z něj `FloatingActionButton`. V souboru *activity_main.xml* odstraňte vizuální definici tohoto widgetu a v kódu v metodě `onCreate()` jeho obsluhu.



Obrázek 8.13: Typ hlavní aktivity Basic Activity už má menu typu ActionBar implementované

Android Studio v projektu typu **Basic Activity** vytvoří automaticky u nového projektu i metodu pro Options menu:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  // Inflate the menu; this adds items to the action bar if it is present.
  getMenuInflater().inflate(R.menu.menu_main, menu);
  return true;
}
```

Pokud uživatel po spuštění aplikace aktivuje menu, Android volá v závislosti na typu menu metodu `onCreateOptionsMenu()`, případně `onCreateContextMenu()`, v které pomocí objektu `MenuInflater` naplní menu konkrétními položkami. Pokud si uživatel vybere z nabídky některou položku, volá se následně metoda `onOptionsMenuItem()`, respektive `onContextMenuItem()` v aktuálním kontextu.

Metoda `getMenuInflater()` umožní načítat definici položek menu v XML. Ve složce *res/menu/* najdete soubor *menu_main.xml*, ve kterém můžete budovat nabídku funkcí na aplikační liště. Vytvoříme více položek, abychom mohli demonstrovat rozdělení lišty v případě nedostatku místa na úzkých displejích s malým rozlišením při orientaci „na výšku".


```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  tools:context="sk.pcrevue.llacko.menu1.MainActivity">
  <item
    android:id="@+id/action_settings"
    android:orderInCategory="100"
    android:title="@string/action_settings"
    app:showAsAction="never" />

  <item android:id="@+id/action_vyhladavanie"
    android:icon="@drawable/ic_action_search"
    android:title="Vyhľadaj"
    app:showAsAction="ifRoom"/>

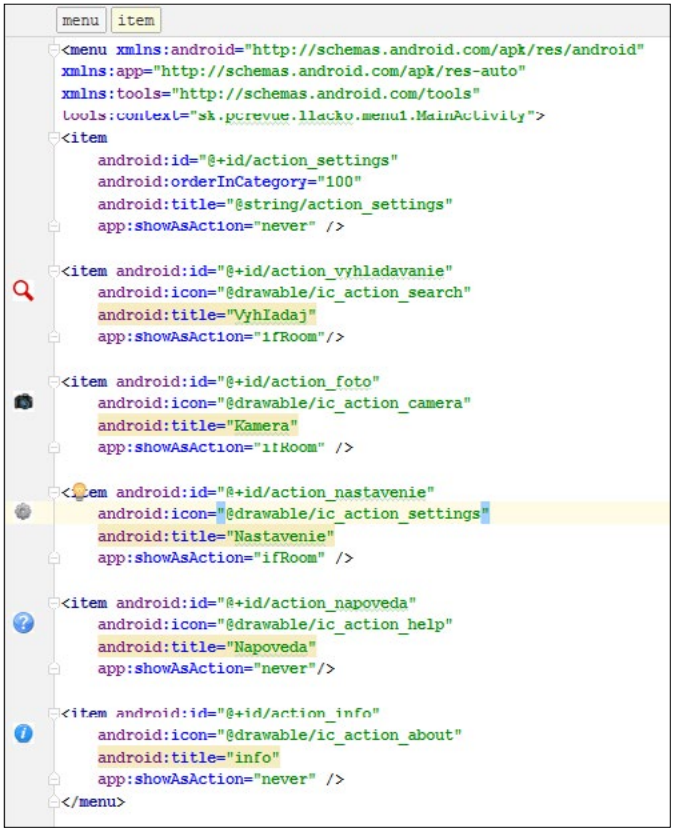
  <item android:id="@+id/action_foto"
    android:icon="@drawable/ic_action_camera"
    android:title="Kamera"
    app:showAsAction="ifRoom" />

  <item android:id="@+id/action_nastavenie"
    android:icon="@drawable/ic_action_settings"
    android:title="Nastavenie"
    app:showAsAction="ifRoom" />

  <item android:id="@+id/action_napoveda"
    android:icon="@drawable/ic_action_help"
    android:title="Napoveda"
    app:showAsAction="never"/>

  <item android:id="@+id/action_info"
    android:icon="@drawable/ic_action_about"
    android:title="info"
    app:showAsAction="never" />

</menu>
```



Obrázek 8.14: Ikonky v menu se v Android Studiu zobrazují u výpisu definičního souboru menu přehledně na liště vlevo

Všimněte si atributu `showAsAction`. Hodnota `"never"` definuje, že položka se zobrazí jen v menu a ne na liště `ActionBar`. Tuto hodnotu jsme zatím nastavili pro poslední dvě položky. Hodnota `"ifRoom"` umožňuje zobrazit položku na liště. Pokud jste definovali ikony, zobrazí se ikona, v opačném případě se zobrazí na liště text. Pokud chcete zobrazit i text, nastavte hodnotu atributu `"withText"`, nebo `"ifRoom|withText"`. Hodnota `"always"` prosadí zobrazení i v případě zařízení s malými displeji, takovéto nastavení ale nedoporučujeme používat. Je to kontraproduktivní, aplikace s více položkami, které by se normálně na displej nevešly, pravděpodobně nebude vypadat dobře.

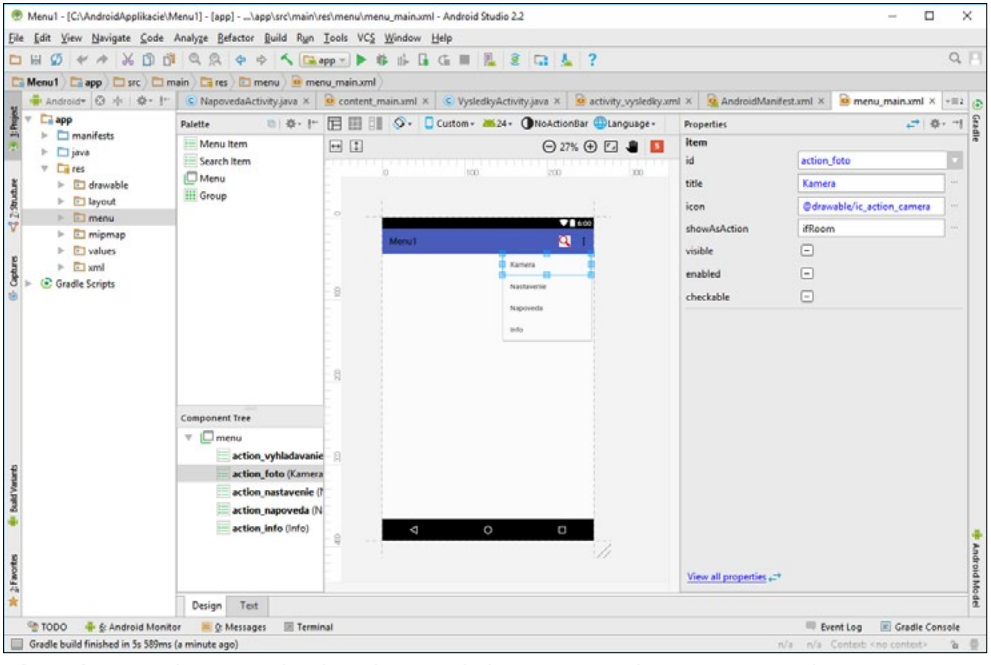
Do složky *res/drawable/* je potřeba přidat ikony pro položky menu na `ActionBar`. Aby aplikace dobře vypadala, doporučujeme přidat do projektu ikony v rozlišeních pro nejpoužívanější rozměry a rozlišení obrazovek do složek:

- *res/drawable-hdpi/* – 48 × 48 pixelů
- *res/drawable-mdpi/* – 32 × 32 pixelů
- *res/drawable-xhdpi/* – 64 × 64 pixelů
- *res/drawable-xxhdpi/* – 96 × 96 pixelů

TIP

Soubory s nejčastěji používanými ikonami, které jsou v souladu s designovými principy aplikací pro Android, si můžete stáhnout na adrese *http://developer.android.com/design/downloads/index.html#action-bar-icon-pack*.

Po rozbalení archívu máte k dispozici sbírku ikon roztříděnou do složek podle motivů (holo_dark, holo_light), přičemž každá ikona je ve více rozlišeních. Ikony překopírujte do odpovídajících složek.



Obrázek 8.15 Od verze Android Studio 2.2 je k dispozici i vizuální nástroj na tvorbu menu. Všimněte si dole záložky Design a Text

V našem definičním souboru jsme dali pro jednoduchost texty přímo do tohoto souboru. V reálné aplikaci doporučujeme všechny textové řetězce ukládat do zdrojů kvůli jednoduché lokalizaci aplikace do jiných jazyků.

```
<resources>
  ...
  <string name="action_vyhledavani">Vyhľadaj</string>
  <string name="action_foto">Kamera</string>
  <string name="action_nastaveni">Nastavie</string>
  <string name="action_napoveda">Nápoveda</string>
  <string name="action_info">Info o aplikácii</string>
</resources>
```

V kódu aktivity upravte metodu onOptionsItemSelected(), v které se zpracovává případné klepnutí na některý z prvků na liště ActionBar.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
  // Handle action bar item clicks here. The action bar will
  // automatically handle clicks on the Home/Up button, so long
  // as you specify a parent activity in AndroidManifest.xml.
  int id = item.getItemId();

  //noinspection SimplifiableIfStatement
  if (id == R.id.action_settings) {
    return true;
  }

  return super.onOptionsItemSelected(item);
}
```

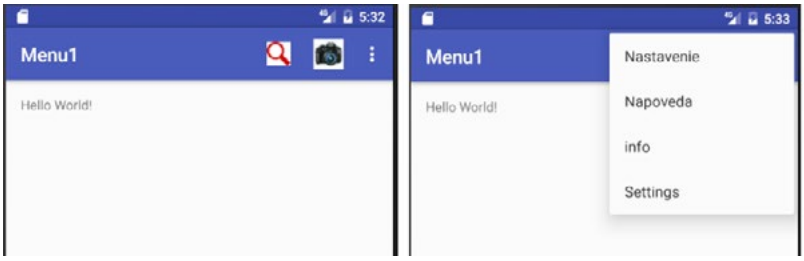
Implicitní kód šablony vložený při vytváření projektu přepište kódem obsahujícím definice akcií, které se mají provést při výběru položky. Aktivní bude zatím pouze položka Nastavení.

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
  switch (item.getItemId())
  {
    case R.id.action_vyhladavanie:
      return true;
    case R.id.action_foto:
      return true;
    case R.id.action_nastavenie:
      aktivujNastavenie();
      return true;
    case R.id.action_napoveda:
      return true;
    case R.id.action_info:
      return true;
    default:
      return super.onOptionsItemSelected(item);
  }
}
```

Jelikož jsme se zatím zaměřili na menu a nikoliv na spouštění nových aktivit, v těle metody aktivujNastavenie() jsme vypsali jen text přes Toast.

```
public void aktivujNastavenie()
{
  Toast.makeText(this, "Nastavenie.",Toast.LENGTH_SHORT).show();
}
```

Po spuštění aplikace stačí aktivovat nabídku pomocí tlačítka v pravé horní části lišty.



Obrázek 8.16: Zobrazení nabídky na liště Action Bar. V dolní části je rozbalená nabídka

Přizpůsobení a rozdělení lišty Action Bar

`ActionBar` je možné přizpůsobit designovému záměru aplikace. Můžete například nastavit barvu popředí a pozadí, dynamicky změnit ikonu libovolné položky a podobně.

```
ActionBar actionBar = getActionBar();
actionBar.setIcon(R.drawable.ico_inaikona);
```

Pokud lišta `ActionBar` obsahuje více tlačítek nebo jiných ovládacích prvků, na zařízeních s úzkým displejem v režimu „na výšku" může nastat problém, aby se prvky na lištu vešly. V takovém případě je možnost rozdělit `ActionBar` na dvě lišty – jedna zůstane v horní části displeje a druhá, rozšířená lišta se zobrazí u spodního okraje. Možnost rozdělit `ActionBar` na dvě lišty se nastavuje v Android manifestu pro příslušnou aktivitu nastavením parametru `uiOptions` na hodnotu `splitActionBarWhenNarrow`, například:

```
<activity
  android:name="com.example.actionbar.MainActivity"
  android:label="@string/app_name"
  android:uiOptions="splitActionBarWhenNarrow">
  ...
</activity>
```

Pro zachování kompatibility se staršími verzemi Androidu se tento parametr nastavuje v elementu `<meta/data>`.

```
<activity uiOptions="splitActionBarWhenNarrow" ... >
  <meta-data android:name="android.support.UI_OPTIONS"
    android:value="splitActionBarWhenNarrow" />
</activity>
```

Na vyzkoušení změňte v definici menu u dvou posledních položek nastavení parametru `showAsAction` na hodnotu `ifRoom`. Po spuštění aplikace zkuste změnu orientace zařízení, případně tuto změnu na emulátoru aktivujte pomocí kombinace kláves **Ctrl+F12**.



Obrázek 8.17: Zobrazení nabídky na liště Action Bar v poloze na šířku, kde je více místa na ikony. V dolní části je rozbalená nabídka

Navigace na jinou aktivitu (a zpět)

Lišta `ActionBar` je ideálním nástrojem k navigaci mezi aktivitami. Pomocí tlačítka Zpět umožňuje realizovat hierarchický vztah mezi obrazovkami.

POZNÁMKA

Je potřeba rozlišovat mezi tlačítkem s významem Zpět na aplikační liště a systémovým tlačítkem Zpět.

Do cvičné aplikace vytvářené v této kapitole přidáme novou aktivitu, jejímž účelem bude zobrazení nápovědy. Nejjednodušeji vytvoříte novou aktivitu ve vývojovém prostředí Android Studio pomocí tlačítka **New** → **Activity**. Aktivitu můžete nazvat například `NapovedaActivity`.

Definice uživatelského rozhraní bude v souboru *activity_napoveda.xml*.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${packageName}.${activityClass}" >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/text_napovedy" />
</LinearLayout>
```

Aby fungoval návrat na rodičovskou aktivitu, je potřeba v souboru *AndroidManifest.xml* tuto hierarchii definovat.

```
<activity
  android:name="com.example.navigace.NapovedaActivity"
  android:label="@string/title_activity_napoveda"
  android:parentActivityName="com.example.navigace.MainActivity" >
</activity>
```

Dceřiná aktivita se spustí v metodě:

```
public void aktivujNapovedu()
{
  Intent i = new Intent(MainActivity.this, NapovedaActivity.class);
  startActivity(i);
}
```

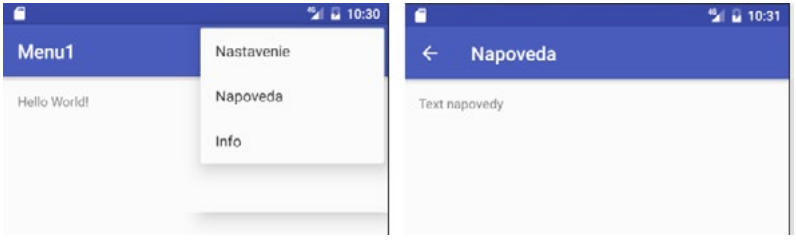
Volání metody přidejte do metody `onOptionsItemSelected()`.

```
...
  case R.id.action_napoveda:
    aktivujNapovedu();
    return true;
...
```

Volání metody přidejte do metody `onOptionsItemSelected()`.

```
...
  case R.id.action_napoveda:
    aktivujNapovedu();
    return true;
...
```

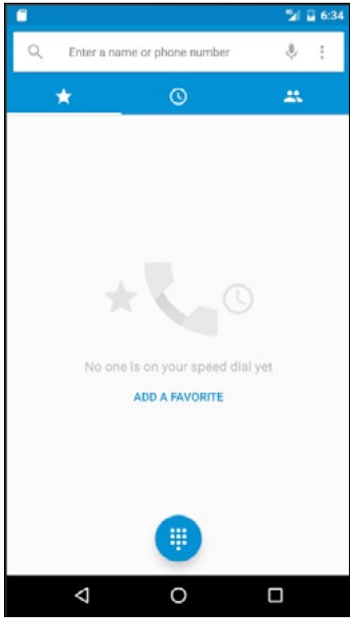
Po spuštění aplikace vyzkoušejte návrat na rodičovskou aktivitu.



Obrázek 8.18: Spuštění dceřiné aktivity. Všimněte si na liště vlevo ikony se šipkou umožňující návrat do rodičovské aktivity

Rozdělení obsahu na karty

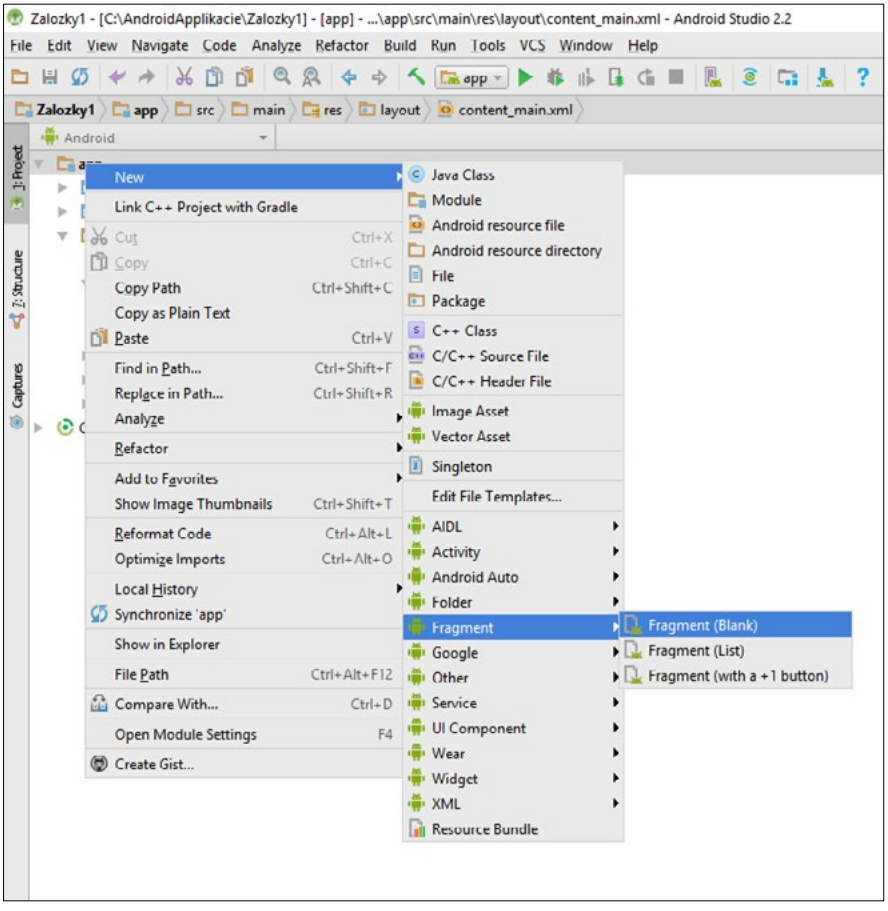
Hodně aplikací potřebuje více zobrazovací plochy, než je k dispozici na malých displejích telefonů. Myslíme tím rozměry, rozlišení displejů špičkových telefonů je často vyšší než u notebooků střední třídy. Jedním z triků, jak virtuálně rozšířit zobrazovací plochu aplikace, jsou karty. V záhlaví karet mohou být buď ikony, nebo texty. Obsah, který se zobrazí na ploše pod kartami, je rozdělen do fragmentů.



Obrázek 8.19: Karty využívá i aplikace na telefonování

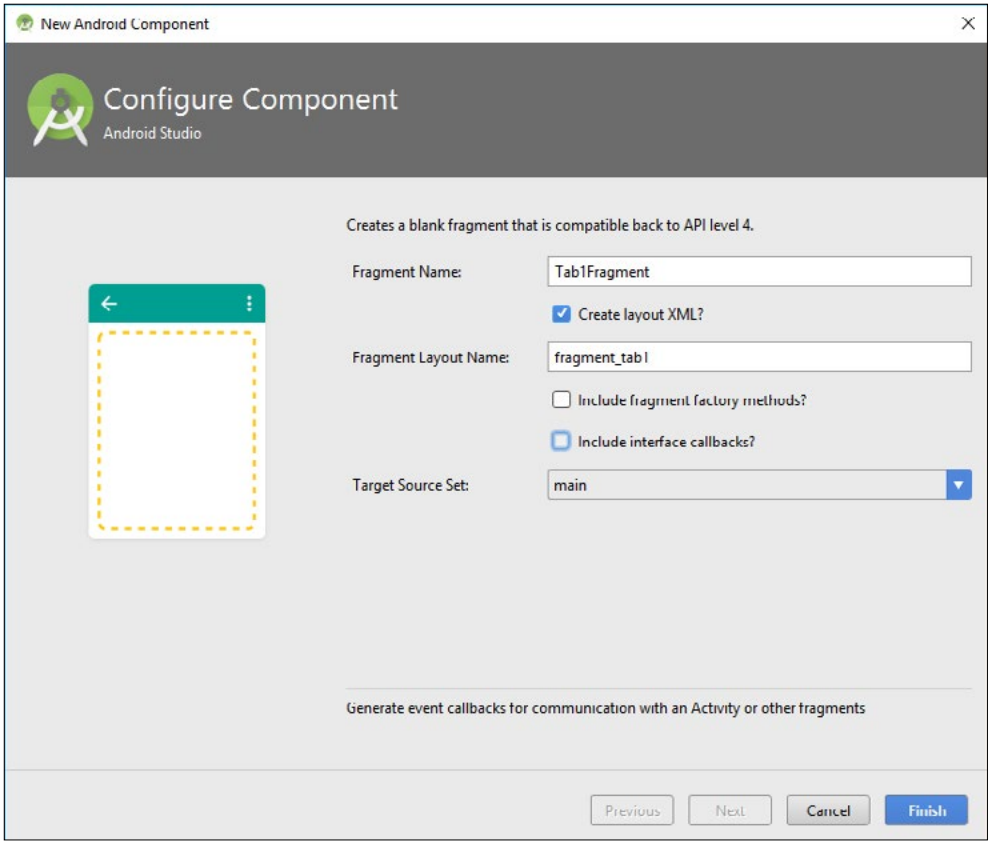
Vytvořte nový projekt a jako typ hlavní aktivity vyberte **Basic Activity**. Aplikace bude mít tři karty a na každé se zobrazí jiný fragment. Dvě karty budou mít v záhlaví text a třetí ikonu. Ke každé kartě bude přiřazen fragment. Fragmenty budou pojmenované `Tab1Fragment`, `Tab2Fragment` a `Tab3Fragment`. Na ploše aplikace pod kartami se bude zobrazovat příslušný fragment podle toho, kterou kartu si uživatel aplikace vybral.

Nový fragment vytvoříte pomocí nabídky **New** → **Fragment** → **Fragment (Blank)**.



Obrázek 8.20: Nabídka vytvoření nového fragmentu

V dialogu na vytvoření fragmentu nechejte neoznačené pole **Include fragment factory methods?** a **Include interface callbacks?**



Obrázek 8.21: Dialog na vytvoření nového fragmentu

Layout fragmentu je v souboru *fragment_tab1.xml*. Obsahuje prvek `TextView`. Jeho text změňte na `Zalozka 1`.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.zalozky3.Tab1Fragment">

  <!-- TODO: Update blank fragment layout -->
  <TextView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text="Zalozka 1" />

</FrameLayout>
```

Stejným postupem vytvořte další dva fragmenty.

V souboru *activity_main* doplňte `TabLayout` a `ViewPager`. `TabLayout` vložte do elementu `AppBarLayout`.

```
<android.support.design.widget.TabLayout
  android:id="@+id/tab_layout"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  app:tabMode="fixed"
  app:tabGravity="fill"/>
```

Element `ViewPager` umístěte za element `AppBarLayout`.

```
<android.support.v4.view.ViewPager
  android:id="@+id/pager"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  app:layout_behavior="@string/appbar_scrolling_view_behavior" />
```

Vytvořte třídu `TabPagerAdapter` odvozenou od `FragmentPagerAdapter`. Tato třída zobrazí fragment podle toho, která karta byla vybrána.

```
public class TabPagerAdapter extends FragmentPagerAdapter {

  int nTab;

  public TabPagerAdapter(FragmentManager fm, int numberOfTabs) {
    super(fm);
    this.nTab = numberOfTabs;
  }

  @Override
  public Fragment getItem(int position) {

    switch (position) {
      case 0: Tab1Fragment t1 = new Tab1Fragment();
        return t1;
      case 1: Tab2Fragment t2 = new Tab2Fragment();
        return t2;
      case 2: Tab3Fragment t3 = new Tab3Fragment();
        return t3;
      default: return null;
    }
  }

  @Override
  public int getCount() {
    return nTab;
  }
}
```

V kódu hlavní aktivity přidáme tři karty. Dvě budou mít v záhlaví text a třetí ikonu. Instance `TabPagerAdapter` je potom přiřazená jako adaptér pro `ViewPager` a `TabLayout`.

```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    TabLayout tl = (TabLayout) findViewById(R.id.tab_layout);

    tl.addTab(tl.newTab().setText("Zalozka 1"));
    tl.addTab(tl.newTab().setText("Zalozka 2"));
    tl.addTab(tl.newTab().setIcon(
      android.R.drawable.ic_dialog_info));

    final ViewPager vp = (ViewPager) findViewById(R.id.pager);
    final PagerAdapter pa = new TabPagerAdapter
      (getSupportFragmentManager(),
      tl.getTabCount());
    vp.setAdapter(pa);

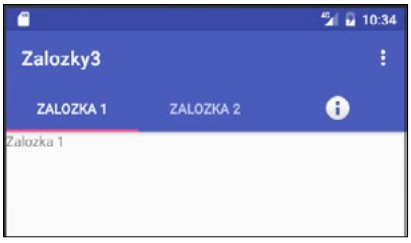
    vp.addOnPageChangeListener(new
      TabLayout.TabLayoutOnPageChangeListener(tl));

    tl.addOnTabSelectedListener(new
      TabLayout.OnTabSelectedListener() {
      @Override
      public void onTabSelected(TabLayout.Tab tab) {
        vp.setCurrentItem(tab.getPosition());
      }

      @Override
      public void onTabUnselected(TabLayout.Tab tab) {
      }

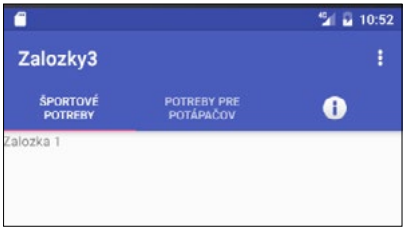
      @Override
      public void onTabReselected(TabLayout.Tab tab) {
      }

    });
  }
  ...
}
```



Obrázek 8.22: Aplikace s kartami

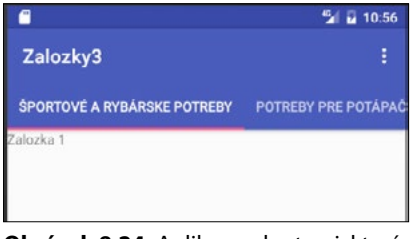
Možná si položíte otázku, jak vyřešit hodně karet nebo karty, které v záhlaví potřebují vypsat delší text. Všimněte si, že v příkladu je `TabLayout` definovaný jako pevný `app:tabMode="fixed"`. Nejprve v kódu hlavní aktivity přiřaďte do záhlaví dlouhé texty. Systém slova v názvu karty automaticky rozdělí do více řádků.



Obrázek 8.23: Aplikace se kartami s dlouhými názvy

Zkuste změnit atribut `app:tabMode` z hodnoty `"fixed"` na `"scrollable"`. Karty budou nyní fungovat jako pás, který můžete vodorovně rolovat.

```
<android.support.design.widget.TabLayout
  android:id="@+id/tab_layout"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  app:tabMode="scrollable"
  app:tabGravity="fill"/>
```



Obrázek 8.24: Aplikace s kartami, které se dají posouvat

Skrývání aplikační lišty

Nemáte pocit, že jsme v předchozích příkladech dosti neuváženě hospodařili s plochou obrazovky? Ponechat záhlaví karet, které se přepínají málokdy, stále zobrazené, je plýtvání. Zobrazovací plocha je na malých displejích cennou devízou, proto je často užitečné aplikační lištu skrývat a zobrazovat ji jen v případě potřeby. Ukážeme to na příkladu, kdy se budeme snažit co největší plochu vyhradit zobrazení obrázku s popisem.

Vytvořte projekt aplikace, přičemž nastavte jako typ hlavní aktivity **Basic Activity**. Aplikační lišta je už součástí této šablony aktivity. Aby byl projekt co nejjednodušší, odstraňte z něj `FloatingActionButton`. V souboru *activity_main.xml* odstraňte vizuální definici tohoto widgetu a v kódu v metodě `onCreate()` jeho obsluhu.

Na nejvyšší úrovni hierarchie tohoto projektu je vizuální kontejner `CoordinatorLayout`. Jak už název napovídá, koordinuje interakce mezi prvky, které jsou v něm zapouzdřeny. Aby koordinace fungovala, je potřeba nastavit vlastnosti těchto prvků. Pro `Toolbar` je potřeba nastavit hodnotu parametru `app:layout_scrollFlags`:

- `scroll` – prvek je možné rolovat mimo obrazovku.
- `enterAlways` – při rolování směrem dolů se `view` zatáhne mimo obrazovku. Naopak rolování dolů způsobí, že se znovu zobrazí.
- `enterAlwaysCollapsed` – při rolování se čeká, dokud se nedosáhne výšky nastavené parametrem `minHeight`. Potom zůstane v této výšce a nebude se rozšiřovat, dokud nedosáhne horní části.

```
<android.support.design.widget.CoordinatorLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:fitsSystemWindows="true"
  tools:context="sk.pcrevue.llacko.zalozky4.MainActivity">

  <android.support.design.widget.AppBarLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:theme="@style/AppTheme.AppBarOverlay">

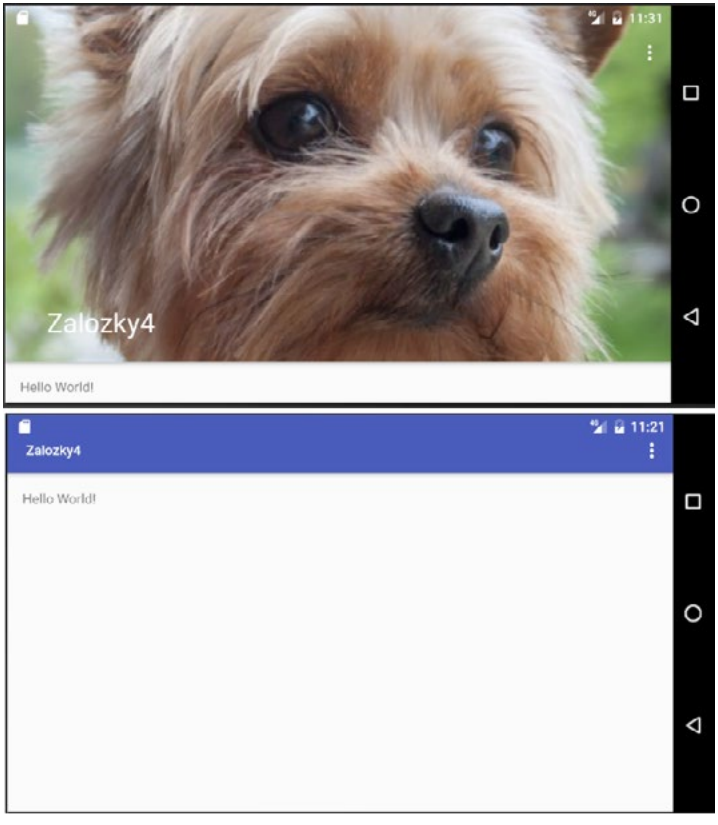
    <android.support.design.widget.CollapsingToolbarLayout
      android:id="@+id/collapsing_toolbar"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      app:layout_scrollFlags="scroll|enterAlways"
      android:fitsSystemWindows="true"
      app:contentScrim="?attr/colorPrimary"
      app:expandedTitleMarginStart="48dp"
      app:expandedTitleMarginEnd="64dp">

      <ImageView
        android:id="@+id/backdrop"
```

```
            android:layout_width="match_parent"
            android:layout_height="350dp"
            android:scaleType="centerCrop"
            android:fitsSystemWindows="true"
            app:layout_collapseMode="parallax"
            android:src="@drawable/york" />

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            app:popupTheme="@style/AppTheme.PopupOverlay"
            app:layout_scrollFlags="scroll|enterAlways"
            app:layout_collapseMode="pin" />

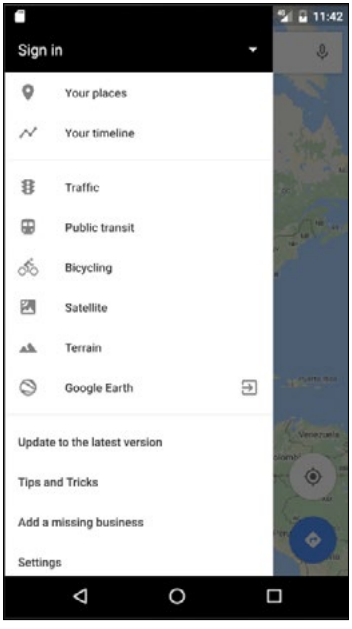
    </android.support.design.widget.CollapsingToolbarLayout>
  </android.support.design.widget.AppBarLayout>
  <include layout="@layout/content_main" />
</android.support.design.widget.CoordinatorLayout>
```



Obrázek 8.25: Gestem (posunem nahoru) zasunete obrázek a zobrazíte lištu. I během doby, kdy je obrázek zobrazen, můžete pomocí ikonky vpravo nahoře aktivovat menu

Využití vysouvacího panelu Navigation Drawer

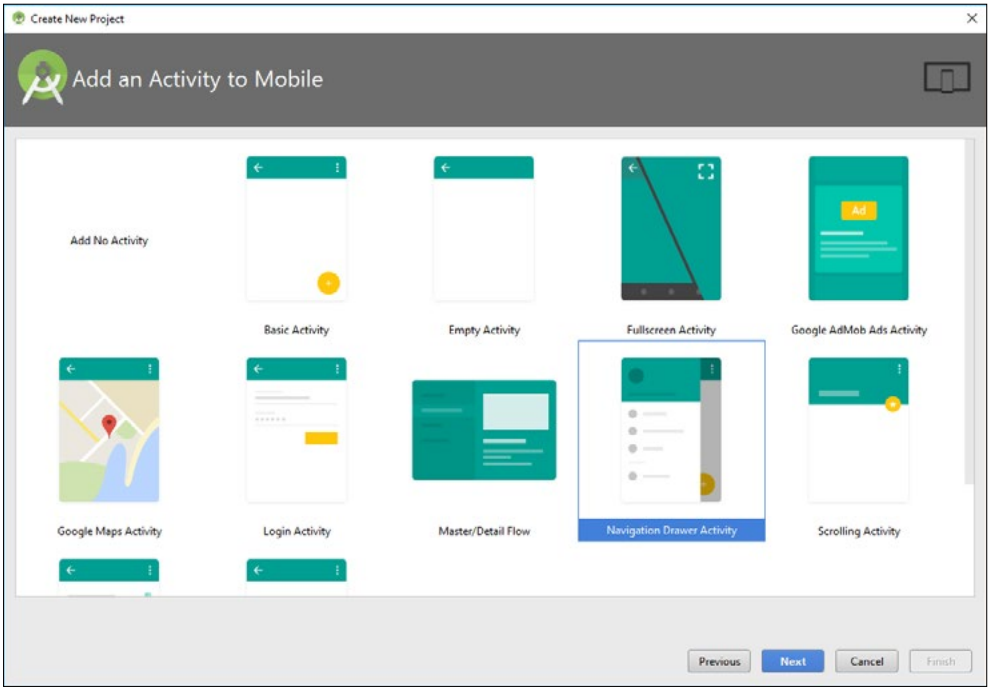
`Navigation Drawer` je vysouvací panel s nabídkou funkcí, který se vysouvá z levé části obrazovky a obsahuje nabídku funkcí na ulehčení navigace v aplikaci a při aktivaci nastavení a jiných funkcí. Jak tento panel funguje, si můžete vyzkoušet v některých aplikacích, které jsou součástí operačního systému, například GooglePlay, Mapy a podobně.



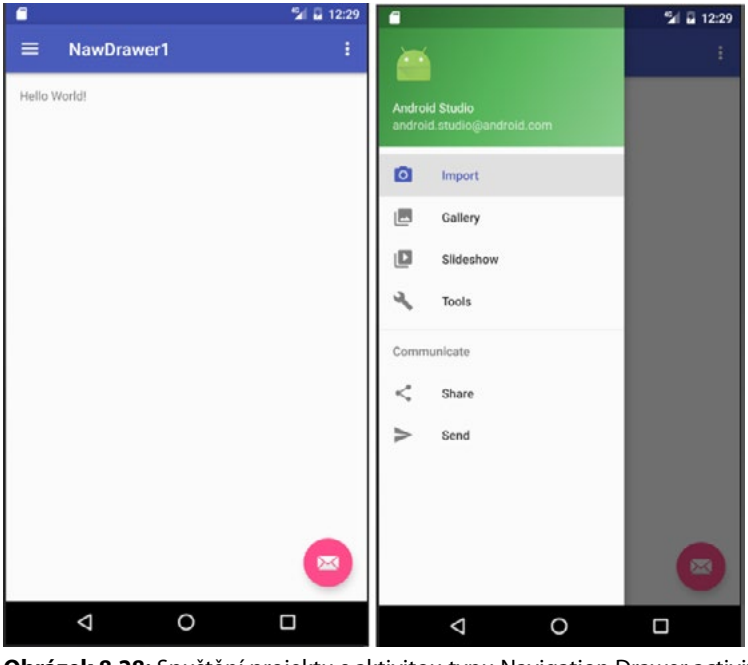
Obrázek 8.26: Navigation Drawer v aplikaci Mapy

Pokud se rozhodnete tento navigační panel použít ve své aplikaci, nejjednodušší je vytvořit projekt, ve kterém vyberete hlavní aktivitu typu **Navigation Drawer activity**.

Po vytvoření projektu s hlavní aktivitou typu **Navigation Drawer activity** doporučujeme nově vytvořený projekt prozkoumat – nejprve z uživatelského hlediska a následně z pohledu vývojáře a designéra.



Obrázek 8.27: Výběr hlavní aktivity typu Navigation Drawer activity



Obrázek 8.28: Spuštění projektu s aktivitou typu Navigation Drawer activity

V souboru *activity_main.xml* jsou všechny prvky zapouzdřeny ve vizuálním kontejneru typu `DrawerLayout`. `DrawerLayout` funguje jako kontejner nejvyšší úrovně pro obsah okna, které umožňuje interaktivní zobrazení panelu, který může být vytažen z jednoho nebo obou vertikálních okrajů okna.

```
<android.support.v4.widget.DrawerLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/drawer_layout"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:fitsSystemWindows="true"
  tools:openDrawer="start">

  <include
    layout="@layout/app_bar_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />

  <android.support.design.widget.NavigationView
    android:id="@+id/nav_view"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_gravity="start"
    android:fitsSystemWindows="true"
    app:headerLayout="@layout/nav_header_main"
    app:menu="@menu/activity_main_drawer" />

</android.support.v4.widget.DrawerLayout>
```

Layout panelu je definován v souboru `nav_header_main.xml`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  android:layout_width="match_parent"
  android:layout_height="@dimen/nav_header_height"
  android:background="@drawable/side_nav_bar"
  android:gravity="bottom"
  android:orientation="vertical"
  android:theme="@style/ThemeOverlay.AppCompat.Dark">

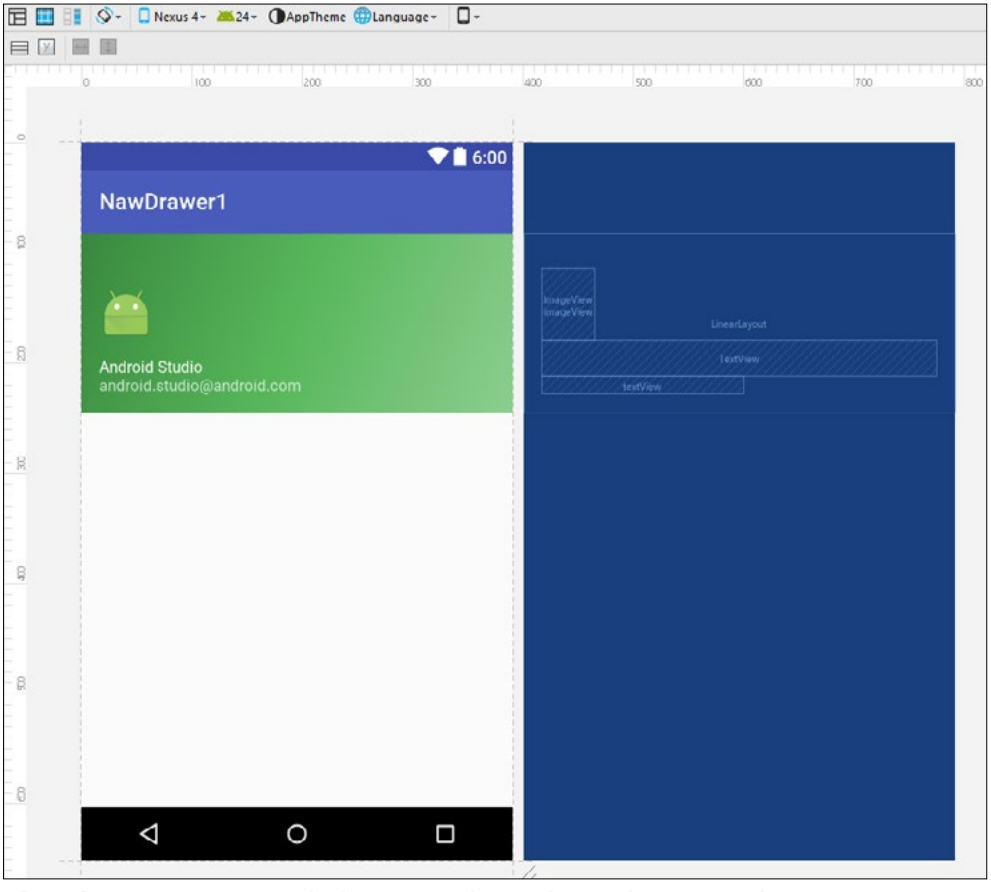
  <ImageView
    android:id="@+id/imageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="@dimen/nav_header_vertical_spacing"
    app:srcCompat="@android:drawable/sym_def_app_icon" />

  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
    android:paddingTop="@dimen/nav_header_vertical_spacing"
    android:text="Android Studio"
    android:textAppearance="@style/TextAppearance.AppCompat.Body1" />

  <TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="android.studio@android.com" />

</LinearLayout>
```

Obrázek 8.29: Názornější než kód XML je návrhové zobrazení layoutu panelu

Položky vysouvacího navigačního panelu jsou definovány v souboru *activity_main_drawer.xml*, který najdete ve složce *menu*. Spusťte znovu aplikaci a podívejte se, jak jsou položky uspořádané. První čtyři jsou uspořádané do skupiny. Další dvě fungují jako menu.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

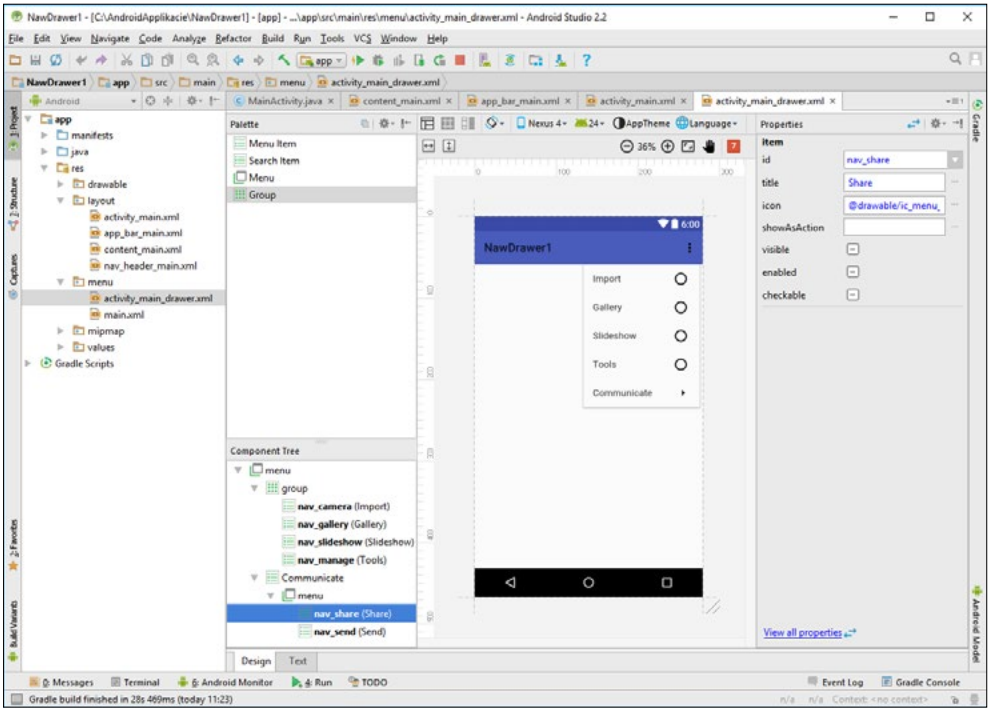
  <group android:checkableBehavior="single">
    <item
      android:id="@+id/nav_camera"
      android:icon="@drawable/ic_menu_camera"
      android:title="Import" />
    <item
      android:id="@+id/nav_gallery"
      android:icon="@drawable/ic_menu_gallery"
      android:title="Gallery" />
    <item
      android:id="@+id/nav_slideshow"
      android:icon="@drawable/ic_menu_slideshow"
      android:title="Slideshow" />
    <item
      android:id="@+id/nav_manage"
      android:icon="@drawable/ic_menu_manage"
      android:title="Tools" />
  </group>

  <item android:title="Communicate">
    <menu>
      <item
        android:id="@+id/nav_share"
        android:icon="@drawable/ic_menu_share"
        android:title="Share" />
      <item
        android:id="@+id/nav_send"
        android:icon="@drawable/ic_menu_send"
        android:title="Send" />
    </menu>
  </item>
```

Strukturu nabídky vysouvacího navigačního panelu lépe pochopíte, když se přepnete do návrhového zobrazení. V tomto zobrazení doporučujeme i nabídky vytvářet a editovat.

V kódu hlavní aktivity jsme nechali jen bloky týkající se `Navigation drawer`. Kód získá referenci na objekt `DrawerLayout` a následně vytvoří objekt `ActionBarDrawerToggle` s odkazem na aktuální aktivitu, objekt `DrawerLayout`, na panelu nástrojů a dva řetězce, které popisují otevření a zavření. Objekt `ActionBarDrawerToggle` je potom přiřazen jako listener.

Metoda `onBackPressed()` řeší zasunutí panelu tlačítkem „Zpět“ pro návrat na předchozí obrazovku aktivity.



Obrázek 8.30: Struktura nabídky vysouvacího navigačního panelu v návrhovém zobrazení

V metodě `onNavigationItemSelected()` se potom aktivuje kód pro jednotlivé volby. V cvičném příkladu se neprovádí žádný kód ani se nevolají žádné metody, jsou tam jen otevírací a zavírací složené závorky.

```
public class MainActivity extends AppCompatActivity
  implements NavigationView.OnNavigationItemSelectedListener {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
    this, drawer, toolbar, R.string.navigation_drawer_open,
      R.string.navigation_drawer_close);
    drawer.setDrawerListener(toggle);
    toggle.syncState();

    NavigationView navigationView = (NavigationView)
      findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
```

```
    }

    @Override
    public void onBackPressed() {
      DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
      if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
      } else {
        super.onBackPressed();
      }
    }

    ...

    @SuppressWarnings("StatementWithEmptyBody")
    @Override
    public boolean onNavigationItemSelected(MenuItem item) {
      // Handle navigation view item clicks here.
      int id = item.getItemId();

      if (id == R.id.nav_camera) {
        // Handle the camera action
      } else if (id == R.id.nav_gallery) {
      } else if (id == R.id.nav_slideshow) {
      } else if (id == R.id.nav_manage) {
      } else if (id == R.id.nav_share) {
      } else if (id == R.id.nav_send) {
      }

      DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
      drawer.closeDrawer(GravityCompat.START);
      return true;
    }
  }
```

Když uživatel klepne na ikonu aktivující navigační panel na aplikační liště, zobrazí se panel. Zda je panel zobrazený, se dá zjistit pomocí volání metody `isDrawerOpen()` objektu `DrawerLayout`.

KAPITOLA

9

Seznamy objektů

Vzhledem k možnostem uživatelského rozhraní nejen telefonů a tabletů, ale i televizorů s Androidem, na kterých mohou též běžet aplikace, se velmi často využívá výběr objektu ze seznamu. Následně aplikace poskytne o vybraném objektu informace, přidá ho do nákupního košíku, zarezervuje, či objedná službu a podobně. Proto seznamům objektů věnujeme velkou pozornost. Seznamy objektů jsme už použili v kapitole 6 při výpisu typu master-detail, kde se v jednom fragmentu zobrazil seznam objektů a v druhém detaily o vybraném objektu.

Cílem příkladů v této kapitole je vypsat seznam objektů. Jako námět využijeme seznam verzí Androidu a odpovídajících API uvedený v druhé kapitole. Postupně budeme vytvářet více variant aplikace, přičemž budeme buď modifikovat, nebo rozšiřovat funkcionalitu. Představíme specializovanou aktivitu `ListActivity`. Umožníme uživateli vybrat více položek a tyto vypíšeme v jiné aktivitě. Zpočátku se zaměříme na funkcionalitu, tedy možnost označování více položek či zobrazení seznamu s hierarchickou strukturou, a potom přidáme i designové prvky včetně zobrazení obrázku ke každé položce.

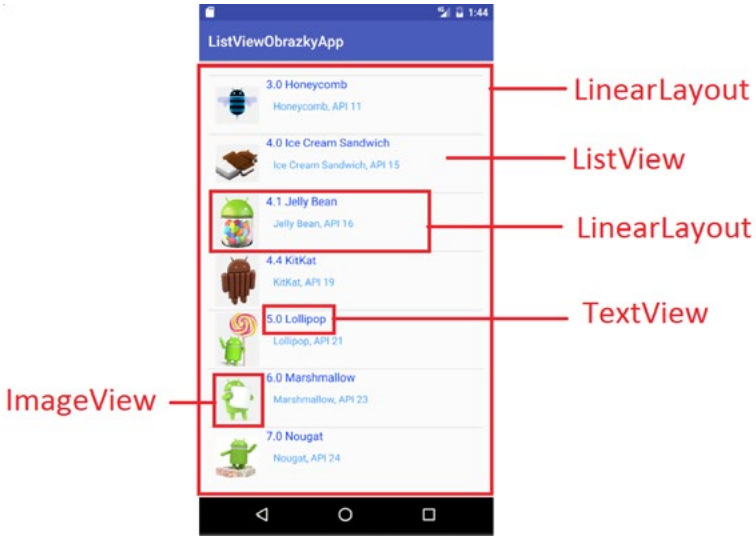
Jak to funguje?

Na konci kapitoly budete umět vytvořit a zobrazit seznam položek s hierarchickou strukturou a obrázky. Nejprve vysvětlíme některé základní principy, jak takovýto seznam vytvořit, naplnit údaji a zobrazit.

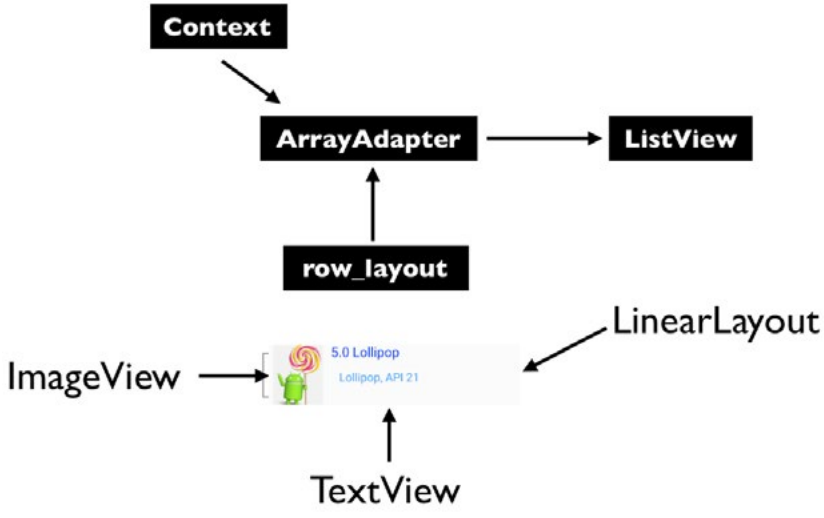
Zobrazovací prvky je potřeba připojit ke zdroji údajů obsahujícímu seznam objektů, které se budou zobrazovat. Pro seznam je to objekt typu `ListAdapter`, v našem případě `ArrayAdapter`, který se využívá k uložení údajů typu pole, například pole objektů, textových řetězců a podobně. Blok `row_layout` na schématu znázorňuje designový návrh jedné položky seznamu. Může se zobrazit například název, krátký popis, obrázek a podobně.

Témata kapitoly:

- **Jak to funguje?**
- **Třída ListActivity**
- **Příklad – ListActivity s formátovaným zobrazením položek**
- **Příklad – ListActivity s kontextovým menu položky**
- **Příklad – ListView se statickými údaji v poli řetězců**
- **Příklad – ListView s údaji v kódu**
- **Příklad – Výběr více položek ze seznamu**
- **Příklad – Zobrazení hierarchické struktury**
- **Seznam objektů s obrázky**
- **Příklad – Aktivita Master/Detail Flow**
- **Přidání objektu do seznamu**
- **Seznamy objektů využívající RecyclerView a CardView**



Obrázek 9.1: Zobrazovací entity seznamu s obrázky



Obrázek 9.2: Princip vytvoření a zobrazení seznamu s obrázky

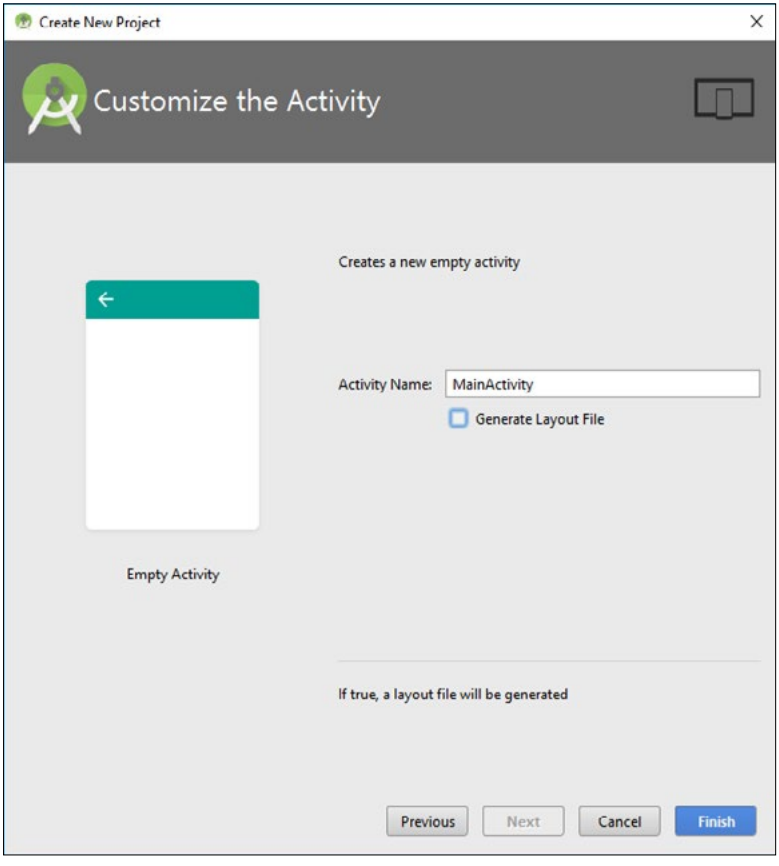
Třída ListActivity

Seznamy objektů jsou natolik používanými komponentami aplikací pro Android, že pro ně byla vytvořena třída `ListActivity`. Pokud potřebujete vytvořit aktivitu, jejíž jedinou úlohou je zobrazit seznam, toto je nejkomfortnější možnost.

POZNÁMKA

Aktivita odvozená od třídy `ListActivity` dokáže zobrazit pouze seznam objektů, nemá dokonce ani layout, takže není možné zobrazit jiné ovládací prvky.

Při vytváření hlavní aktivity nechejte volbu **Generate Layout File** neoznačenou.



Obrázek 9.3: Při vytváření hlavní aktivity nechejte volbu Generate Layout File neoznačenou

Vytvořte aplikaci a v kódu hlavní aktivity v souboru *MainActivity.java* deklarujte, že třída hlavní aktivity je odvozená od rodičovské třídy `ListActivity`.

```
public class MainActivity extends ListActivity
{
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
  }
}
```

Pokud jste zapomněli zrušit volbu **Generate Layout File**, v metodě `onCreate()` z kódu vygenerovaného průvodcem musíte vymazat řádek, který propojuje kód aktivity s layoutem XML.

```
setContentView(R.layout.activity_main);
```

Aktivita odvozená od `ListActivity` totiž nepotřebuje soubor s layoutem, jelikož disponuje zabudovaným layoutem.

UPOZORNĚNÍ

Pokud byste propojení na XML soubor layoutu neodstranili, došlo by k chybě.

Soubor *activity_main.xml* nepotřebujete, a pokud jste ho vytvořili, můžete ho z projektu vymazat.

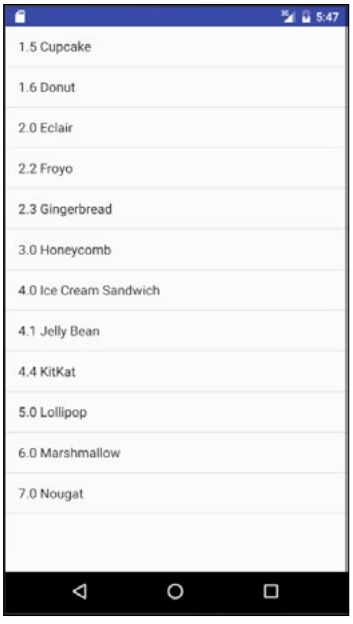
Aktivitu odvozenou od `ListActivity` je potřeba připojit na model, tj. zdroj údajů. V tomto případě model obsahuje seznam objektů, které se budou zobrazovat. Pro seznam je to objekt typu `ListAdapter`, v našem případě `ArrayAdapter`, který se využívá k uložení údajů typu pole, například pole objektů, textových řetězců a podobně.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends ListActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    List<String> verze = Arrays.asList("1.5 Cupcake", "1.6 Donut",
      "2.0 Eclair", "2.2 Froyo", "2.3 Gingerbread", "3.0 Honeycomb",
      "4.0 Ice Cream Sandwich", "4.1 Jelly Bean", "4.4 KitKat",
      "5.0 Lollipop", "6.0 Marshmallow", "7.0 Nougat");
    ArrayAdapter<String> listAdapter =
    new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
      verze );
    setListAdapter(listAdapter);
  }
}
```

Hlavní aktivita aplikace nemá layout, takže její vzhled si můžete prohlédnout až po prvním spuštění aplikace.



Obrázek 9.4: Zobrazení seznamu objektů pomocí ListActivity

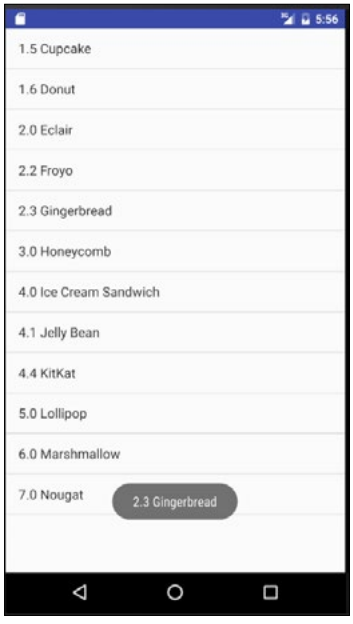
Určitě budete chtít implementovat obsluhu události klepnutí na položku. Upravte kód metody `onCreate()`, aby vypsal informaci o verzi pomocí toastové notifikace (oznámení). Název toast je odvozen od způsobu zobrazení oznámení. Oznámení vyskočí na plochu aplikace podobně jako toast z topinkovače.

```
public class MainActivity extends ListActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    List<String> verze = Arrays.asList("1.5 Cupcake", "1.6 Donut",
      "2.0 Eclair", "2.2 Froyo", "2.3 Gingerbread", "3.0 Honeycomb",
      "4.0 Ice Cream Sandwich", "4.1 Jelly Bean", "4.4 KitKat",
      "5.0 Lollipop", "6.0 Marshmallow", "7.0 Nougat");
    ArrayAdapter<String> listAdapter =
      new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
      verze );
    setListAdapter(listAdapter);

    ListView lv = getListView();
    lv.setTextFilterEnabled(true);
    lv.setOnItemClickListener(new AdapterView.OnItemClickListener()
    {
      public void onItemClick(AdapterView<?> parent, View view,
        int position, long id)
      {
        Toast.makeText(getApplicationContext(), ((TextView) view).getText(),
          Toast.LENGTH_SHORT).show();
```

```
        }
      });
    }
}
```



Obrázek 9.5: Zobrazení vybrané položky ze seznamu objektů pomocí toastu

Kromě klepnutí je možné implementovat i přidržení prstu na dotykovém displeji v místě, kde je zobrazená některá z položek. Takto se zpravidla zobrazují kontextové informace o vybraném objektu, místní nabídka a podobně. Pokud metoda `boolean onItemLongClick()` vrátí hodnotu `true`, klepnutí a následné podržení položky je považováno za zpracované a systém už nevolá listener `nListItemClick()`.

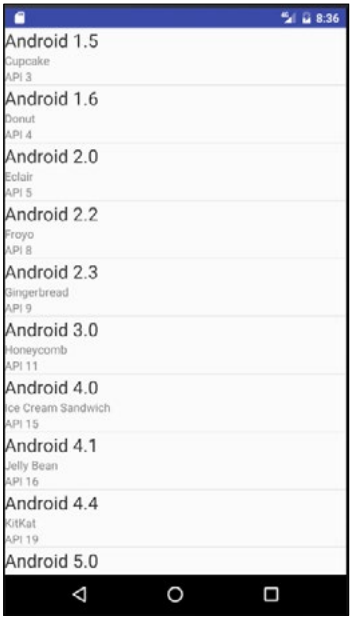
```
protected void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  List<String> verze = Arrays.asList("1.5 Cupcake", "1.6 Donut",...);
  ArrayAdapter<String> listAdapter =
  new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
    verze );
  setListAdapter(listAdapter);

  ListView lv = getListView();
  lv.setTextFilterEnabled(true);
  lv.setOnItemLongClickListener(new AdapterView.OnItemLongClickListener()
  {
    @Override
    public boolean onItemLongClick(AdapterView<?> parent, View view,
      int position, long id)
    {
      Toast.makeText(getApplicationContext(), "Kontext položky: " +
```

```
            ((TextView) view).getText(),
            Toast.LENGTH_SHORT).show();
          return true;
        }
      });
  }
```

Příklad – ListActivity s formátovaným zobrazením položek

V předchozím řešení jsme deklarovali, že aktivita odvozená od `ListActivity` nepotřebuje návrh layoutu v souboru XML. To platí pro zapouzdření aktivity jako celku. Layouty se však často používají na formátované zobrazení položek seznamu. Modifikujeme předchozí příklad a zobrazíme nejen označení a název hlavních edic Androidu, ale i číslo API přiřazené k těmto verzím.



Obrázek 9.6: Cíl příkladu

Podobně jako v předchozím příkladu vytvořte aplikaci a v kódu hlavní aktivity deklarujte, že třída hlavní aktivity je odvozena od rodičovské třídy `ListActivity`. Ani v tomto případě nepotřebujeme definiční soubor XML layoutu hlavní aktivity.

Návrh uživatelského rozhraní

Vytvořte nový layout položky v souboru *list_item.xml*. Prvky uživatelského rozhraní budou zapouzdřeny v kontejneru `LinearLayout`. Konkrétně se bude jednat o tři prvky typu `TextView` pro výpis čísla verze, kódového označení a čísla API. Číslo verze bude vypsáno větším písmem než zbylé dva údaje. Na definování velikosti využijeme předdefinované styly `TextAppearance.Large` a `TextAppearance.Small`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <TextView android:id="@+id/verzie"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    style="@android:style/TextAppearance.Large"/>

  <TextView android:id="@+id/nazov"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    style="@android:style/TextAppearance.Small"/>

  <TextView android:id="@+id/api"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    style="@android:style/TextAppearance.Small"/>

</LinearLayout>
```

Vytvořte třídu obsahující atributy, které chcete zobrazit v každé položce seznamu.

```
public class VerzieAndroidu {
  private String verzia;
  private String nazov;
  private String api;

  public VerzieAndroidu(String verzia, String nazov, String api) {
    this.verzia = verzia;
    this.nazov = nazov;
    this.api = api;
  }

  public String getVerzia() {
    return verzia;
  }

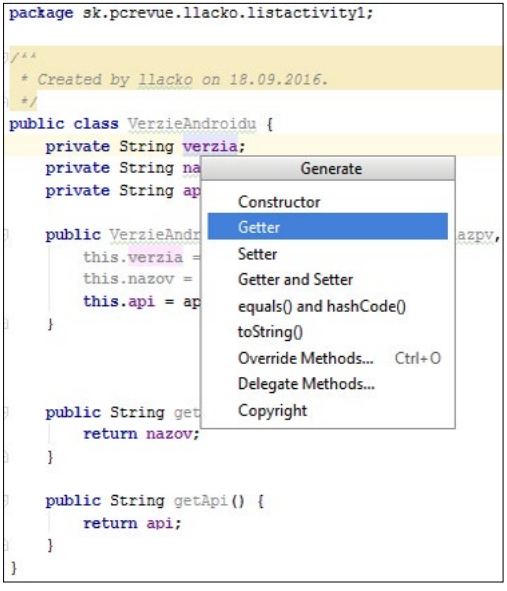
  public String getNazov()  {
    return nazov;
  }

  public String getApi() {
    return api;
  }
}
```

TIP

Na vytvoření metod na přístup k atributům třídy stačí z místní nabídky názvu atributu vybrat položku **Generate**.

Jelikož se hodnoty nastavují pouze v konstruktoru, v tomto příkladu potřebujete jen metody `getNazevAtributu`.



Obrázek 9.7: Vytvoření metod „getter" a „setter" na přístup k atributům třídy

Aplikační kód

Datový model (tedy údaje, které se zobrazí jako položky) je spravován v takzvaném adaptéru, což je třída odvozená od třídy `BaseAdapter`.

POZNÁMKA

Na úrovni adaptéru se realizují i případné operace s údaji, například seřazení či filtrování.

V předchozím příkladu byla jako zdroj údajů pro `ListActivity` použita třída `ArrayAdapter`. Tento adaptér dokáže odevzdat k zobrazení jen jeden řetězec pro položku. Můžete sice zkombinovat více atributů – dokonce i různého typu – do jednoho textového řetězce, jelikož `ArrayAdapter` přebírá textový řetězec z metody `toString()`. V našem příkladu potřebujeme v rámci každé položky zobrazit více atributů, každý z nich bude napojený na jiný prvek `TextView`.

Řešením je adaptér typu `SimpleAdapter`, který však vyžaduje údaje ve formě seznamu map, přičemž každá instance objektu `Map` obsahuje údaje pro jednu položku.

```
Map<String, String> polozkyMap = new HashMap<String, String>();
polozkyMap.put("verzia", ver.getVerzia());
polozkyMap.put("nazov", ver.getNazov());
polozkyMap.put("api", ver.getApi());
```

Následně je potřeba mapovat atributy na prvky TextView. K tomuto účelu je potřeba vytvořit dvě pole, první s názvy atributů a druhé s identifikátory prvků. Atribut na příslušné pozici v prvním poli odpovídá identifikátoru na stejné pozici v druhém poli.

```
String[] nazvyAtributu = {"verzia", "nazov","api"};
int[] idAtributu = { R.id.verzia, R.id.nazov, R.id.api};
SimpleAdapter adapter = new SimpleAdapter
  (this, getListAdapterData(verze),
  R.layout.list_items, nazvyAtributu, idAtributu);
```

Kompletní kód hlavní aktivity:

```
public class MainActivity extends ListActivity
{
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    List<VerzieAndroidu> verzie = Arrays.asList(
      new VerzieAndroidu("Android 1.5", "Cupcake", "API 3"),
      new VerzieAndroidu("Android 1.6", "Donut", "API 4"),
      new VerzieAndroidu("Android 2.0", "Eclair", "API 5"),
      new VerzieAndroidu("Android 2.2", "Froyo", "API 8"),
      new VerzieAndroidu("Android 2.3", "Gingerbread", "API 9"),
      new VerzieAndroidu("Android 3.0", "Honeycomb", "API 11"),
      new VerzieAndroidu("Android 4.0", "Ice Cream Sandwich", "API 15"),
      new VerzieAndroidu("Android 4.1", "Jelly Bean", "API 16"),
      new VerzieAndroidu("Android 4.4", "KitKat", "API 19"),
      new VerzieAndroidu("Android 5.0", "Lollipop", "API 21"),
      new VerzieAndroidu("Android 6.0", "Marshmallow", "API 23"),
      new VerzieAndroidu("Android 7.0", "Nougat", "API 24")
    );
    setListAdapter(getListAdapter(verzie));
  }

  private ListAdapter getListAdapter(List<VerzieAndroidu> verzie)
  {
    String[] nazvyAtributu = {"verzia", "nazov","api"};
    int[] idAtributu = { R.id.verzia, R.id.nazov, R.id.api};
    SimpleAdapter adapter = new SimpleAdapter
      (this, getListAdapterData(verzie),
      R.layout.list_item, nazvyAtributu, idAtributu);
    return adapter;
  }

  private List<Map<String, ?>> getListAdapterData(List<VerzieAndroidu> verze)
  {
    List<Map<String, ?>> list = new ArrayList<Map<String, ?>>(verze.size());
    for (VerzieAndroidu ver : verze)
    {
      Map<String, String> polozkyMap = new HashMap<String, String>();
      polozkyMap.put("verzia", ver.getVerzia());
```

```
        polozkyMap.put("nazov", ver.getNazov());
        polozkyMap.put("api", ver.getApi());
        list.add(polozkyMap);
      }
      return list;
    }
  }
```

Aktivita odvozená od `ListActivity`, která zobrazuje jen seznam objektů bez dalších prvků, ne vždy vyhovuje. Proto v dalších příkladech ukážeme řešení s využitím prvku `ListView`, který je možné zapouzdřit do layoutu aktivity spolu s dalšími prvky.

Příklad – ListActivity s kontextovým menu položky

V předchozím příkladu jsme ukázali jednoduchou obsluhu události „klepnutí a přidržení". Často je potřeba realizovat nad některou položkou více druhů operací. Zaběhlá konvence je, že po podržení položky se zobrazí kontextové menu a uživatel si vybere požadovanou operaci.

V aktivitě je potřeba překrýt metodu `onCreateContextMenu()` a naplnit menu položkami v souvislosti s vybranou položkou záznamu. Když si uživatel vybere položku menu, zavolá se metoda `onContextItemSelected()`.

Metoda `getListView()` vrátí interní prvek `ListView` se seznamem.

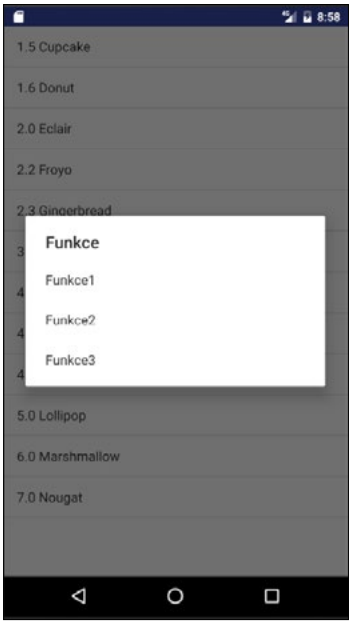
Metoda `onCreateContextMenu()` na vytvoření položek menu je společná pro všechny prvky, které budou zobrazovat menu, proto je v reálné aplikaci potřeba zkontrolovat, zda se jedná o menu pro předmětný prvek. Provedeme to pro ilustraci i ve cvičném příkladu, kde je takovýto prvek pouze jeden. Pomocí metody `add()` následně naplníme kontextové menu položkami.

```
if (view.getId() == this.getListView().getId())
{
  menu.setHeaderTitle("Funkce");
  menu.add(Menu.NONE, MENU_ITEM_ID_FUNKCE1, Menu.NONE, "Funkce1");
  ...
}
```

Položku menu identifikuje jednoznačný identifikátor položky deklarovaný jako konstanta. Pomocí metody `setHeaderTitle()` nastavíte nadpis nad kontextovým menu.

```
menu.setHeaderTitle("Funkce");
```

Když vyzkoušíte aplikaci v této fázi, zobrazí se menu s nadpisem i položkami, po klepnutí na vybranou položku se ale zatím nestane nic.



Obrázek 9.8: Zobrazení kontextového menu pro položku

I obsluha výběru položky kontextového menu pomocí metody `onContextItemSelected(MenuItem item)` se odehrává na úrovni aktivity. Parametr obsahuje objekt `MenuItem`, konkrétně položku menu vybranou uživatelem. Pomocí metody `getItemId()` zjistíte identifikátor vybrané položky a pomocí metody `getMenuInfo()` dokážete z vráceného objektu `ContextMenuInfo` zjistit podrobnější informace o vybrané položce, konkrétně:

- `id` – obsahuje identifikátor položky: Pokud položka pochází ze záznamu databázové tabulky, obsahuje proměnná hodnotu atributu `_id`, tj. unikátní identifikátor záznamu.
- `position` – udává pozici prvku v seznamu, takže první položka má hodnotu `0`.
- `targetView` – `view`, pro který se zobrazuje kontextové menu. V případě seznamů `ListView` to bude konkrétní `view` pro jednotlivou položku.

V `onContextItemSelected(MenuItem item)` můžete vytvořit rozhodovací logiku pro jednotlivé položky menu.

Dodržujte kontrakt metody. Po obsloužení výběru je potřeba vrátit hodnotu `true`.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends ListActivity
{
  private static final int ITEM_ID_FUNKCE1 = Menu.FIRST + 1;
  private static final int ITEM_ID_FUNKCE2 = Menu.FIRST + 2;
  private static final int ITEM_ID_FUNKCE3 = Menu.FIRST + 3;

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
```

```
    List<String> verze = Arrays.asList("1.5 Cupcake", "1.6 Donut",
      "2.0 Eclair", "2.2 Froyo", "2.3 Gingerbread", "3.0 Honeycomb",
      "4.0 Ice Cream Sandwich", "4.1 Jelly Bean", "4.4 KitKat",
      "5.0 Lollipop", "6.0 Marshmallow", "7.0 Nougat");

    ArrayAdapter<String> listAdapter =
      new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
      verze );
    setListAdapter(listAdapter);
    registerForContextMenu(getListView());
  }

  @Override
  public void onCreateContextMenu(ContextMenu menu, View view,
    ContextMenu.ContextMenuInfo menuInfo)
  {
    if (view.getId() == this.getListView().getId())
    {
      menu.setHeaderTitle("Funkce");
      menu.add(Menu.NONE, ITEM_ID_FUNKCE1, Menu.NONE, "Funkce1");
      menu.add(Menu.NONE, ITEM_ID_FUNKCE2, Menu.NONE, "Funkce2");
      menu.add(Menu.NONE, ITEM_ID_FUNKCE3, Menu.NONE, "Funkce3");
    }
  }

  @Override
  public boolean onContextItemSelected(MenuItem item)
  {

    switch(item.getItemId())
    {
      case ITEM_ID_FUNKCE1:
        Toast.makeText(getApplicationContext(), "Funkce1",
          Toast.LENGTH_SHORT).show();
        return true;
      case ITEM_ID_FUNKCE2:
        Toast.makeText(getApplicationContext(), "Funkce2",
          Toast.LENGTH_SHORT).show();
        return true;
      case ITEM_ID_FUNKCE3:
        Toast.makeText(getApplicationContext(), "Funkce3",
          Toast.LENGTH_SHORT).show();
        return true;
      default:
        return super.onContextItemSelected(item);
    }
  }
}
```

Příklad – ListView se statickými údaji v poli řetězců

V první a nejjednodušší variantě využijeme pole řetězů v elementu `<string-array>` uložené ve zdrojích ve složce *res/values/string.xml*. Vypíšeme zatím jen majoritní verze Androidu bez čísla API. Tento soubor postupně použijeme pro víc příkladů.

```
<resources>
  ...

  <string-array name="verze_array">
    <item>Android 1.5 Cupcake</item>
    <item>Android 1.6 Donut</item>
    <item>Android 2.0 Eclair</item>
    <item>Android 2.2 Froyo</item>
    <item>Android 2.3 Gingerbread</item>
    <item>Android 4.0 Ice Cream Sandwich</item>
    <item>Android 4.1 Jelly Bean</item>
    <item>Android 4.4 KitKat</item>
    <item>Android 5.0 Lollipop</item>
    <item>Android 6.0 Marshmallow</item>
    <item>Android 7.0 Nougat</item>
  </string-array>
</resources>
```

Návrh uživatelského rozhraní

Vzhledem k jednoduchému zadání bude jednoduchý i návrh layoutu hlavní aktivity. Bude obsahovat jediný prvek typu `ListView` na zobrazení rolovatelného seznamu položek. `ListView` je přes atribut `android:entries` napojen na pole řetězců obsahující položky, které chceme přes `ListView` vypsat.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.listvievstatic.MainActivity">

  <ListView
    android:id="@+id/verzeList"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/verze_array"/>

</RelativeLayout>
```

Aplikační kód

Stejně jednoduchý bude i kód hlavní aktivity. V metodě `onCreate()` přiřadíme identifikátor prvku `ListView` do proměnné `listView` a definujeme metodu `onItemClick()` na obsluhu události – klepnutí na položku. V tomto příkladu jen vypíšeme název prvku prostřednictvím objektu `Toast`.

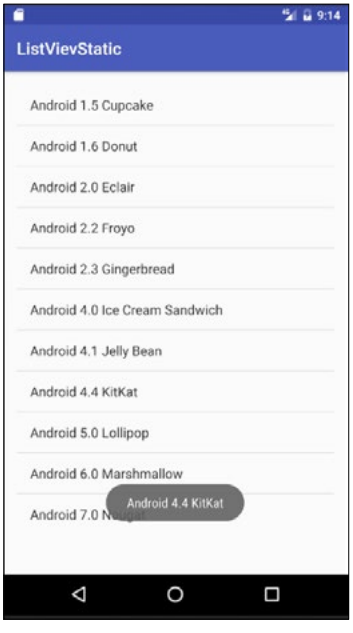
```
public class MainActivity extends Activity
  implements OnItemClickListener {
  ListView listView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    listView = (ListView) findViewById(R.id.verzeList);
    listView.setOnItemClickListener(this);

  }
  @Override
  public void onItemClick(AdapterView<?> adapter, View view, int position,
    long id) {
    Toast.makeText(getApplicationContext(), ((TextView) view).getText(),
    Toast.LENGTH_SHORT).show();
  }
}
```

Všimněte si, že v deklaraci třídy je implementovaný `OnItemClickListener` využívající metodu `setOnItemClickListener`. Po klepnutí na položku se aktivuje metoda `onItemClick()`.



Obrázek 9.9: Test fungování aplikace. V dolní části je toastový výpis, který se zobrazí po klepnutí na položku.

Příklad – ListView s údaji v kódu

V předchozí aplikaci jsme napojili prvek `ListView` na zdroj údajů ve zdrojích přímo v kódu XML návrhu layoutu přes atribut `android:entries`. V tomto příkladu naplníme prvek `ListView` v kódu aktivity. Výhody jsou zřejmé – údaje je možné různým způsobem předzpracovat, prvek naplnit z různých zdrojů údajů, například z databázové tabulky SQLite, z JSON, XML... Pro jednoduchost použijeme stejný zdroj údajů jako v předchozím příkladu – pole řetězců v elementu `<string-array>` uložené ve zdrojích ve složce *res/values/string.xml*.

Návrh uživatelského rozhraní

I v tomto příkladu bude layout hlavní aktivity obsahovat jediný prvek typu `ListView`. Jeho definice bude ještě o jeden řádek kratší – nepotřebujeme definovat atribut `android:entries`.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.listvievstatic.MainActivity">

  <ListView
    android:id="@+id/verzeList"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
</RelativeLayout>
```

Můžete využít implicitní layout pro jednotlivé řádky seznamu nebo si navrhnout vlastní. V tomto příkladu vytvoříme nový layout v souboru *list_item.xml*. Soubor bude ve složce *res/layout*. Jako kořenový element bude obsahovat prvek `TextView`.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dp">
</TextView>
```

Aplikační kód

V kódu hlavní aktivity načítáme hodnoty z pole řetězců a odevzdáme ho objektu `ListView` přes objekt typu `Adapter`. `Adapter` funguje tak, že převezme údaje ve vhodném formátu, nejčastěji jako pole objektů, nebo třeba objekt `Cursor` u databázové aplikace. Položek může být libovolné množství, jak jsme však už uvedli, musí být stejného typu. Klíčovou funkcionalitou objektu `Adapter` je schopnost transformovat každou položku na objekt typu `View`. V případě objektu `ListView` jsou to jednotlivé položky seznamu. Každé `View` si vyžádá tu položku, která odpovídá pořadí jejího zobrazení.

V našem příkladu využijeme objekt typu `ArrayAdapter<T>`. Je to odvozený adaptér určený k práci s polem objektů, například s textovými řetězci nebo jinými objekty, které se dají konvertovat na textový řetězec. Konstruktor má tři parametry: `Context`, odkaz na layout a pole instancí třídy T.

```
listView.setAdapter(new ArrayAdapter<String>(this,
  android.R.layout.simple_list_item_1, verze));
```

Layout pro zobrazení záznamů si můžete definovat sami nebo můžete využít některý z layoutů, které jsou definované v Android Frameworku. V našem případě jsme použili nejjednodušší předdefinovaný layout `android.R.layout.simple_list_item_1`, který obsahuje jediný prvek `TextView` a nic víc.

Obsluha události klepnutí na položku zůstává stejná jako v předchozím příkladu.

```
public class MainActivity extends AppCompatActivity
  implements AdapterView.OnItemClickListener {

  ListView listView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    listView = (ListView) findViewById(R.id.verzeList);

    String[] verze = getResources().getStringArray(R.array.verze_array);
    listView.setAdapter(new ArrayAdapter<String>(this,
      android.R.layout.simple_list_item_1, verze));
      listView.setOnItemClickListener(this);
  }

  public void onItemClick(AdapterView<?> adapter, View view,
    int position, long id) {
    Toast.makeText(getApplicationContext(), ((TextView) view).getText(),
      Toast.LENGTH_SHORT).show();
  }
}
```

Namísto načítání hodnot ze zdrojů byste mohli statické hodnoty definovat přímo v kódu.

```
static final String[] verze = {"Android 1.5 Cupcake",
  "Android 1.6 Donut",
  ...
};
```

Znovu zdůrazňujeme, že účelem příkladu bylo ukázat, jak se zobrazí položky seznamu vygenerované kódem. V reálné aplikaci nebudou údaje statické, ale vypočítané nebo získané z nějakého zdroje.

Příklad – Výběr více položek ze seznamu

Velmi často se v souvislosti se seznamem využívá scénáře, kdy si uživatel vybere jeden nebo několik prvků a tyto prvky se následně nějakým způsobem zpracují, zobrazí a podobně. Proto vytvoříme aplikaci se dvěma aktivitami. V hlavní aktivitě si uživatel vybere položky, označí je a seznam vybraných položek se po potvrzení zobrazí v jiné aktivitě pomocí prvku `ListView`. Použijeme stejný zdroj údajů jako v předchozích příkladech – pole řetězců v souboru *res/values/string.xml*.

```
<resources>
  <string name="app_name">Seznam3</string>
  <string name="vysledek_activity">Vysledek Activity</string>
  <string name="ok">OK</string>
  <string-array name="verze_array">
    <item>Android 1.5 Cupcake</item>
    ...
  </string-array>
</resources>
```

Návrh uživatelského rozhraní

Layout hlavní aktivity obsahuje dva prvky: `ListView` zarovnaný s horním okrajem layoutu a prvek `Button` zarovnaný s dolním okrajem layoutu typu **Relative Layout**. Také je definováno, že prvek `ListView` je nad prvkem `Button`.

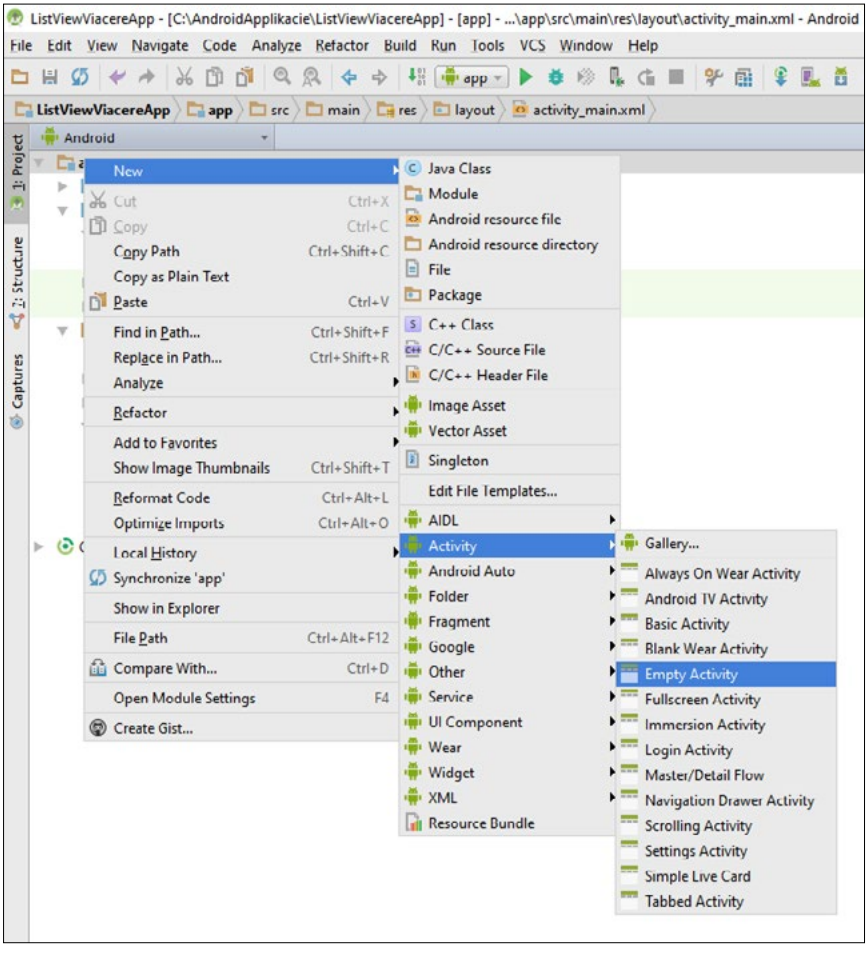
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.listviewviacereapp.MainActivity">

  <Button
    android:id="@+id/okbutton"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text="@string/ok" />
  <ListView
    android:id="@+id/list"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_above="@id/okbutton"
    android:layout_alignParentTop="true"/>
</RelativeLayout>
```

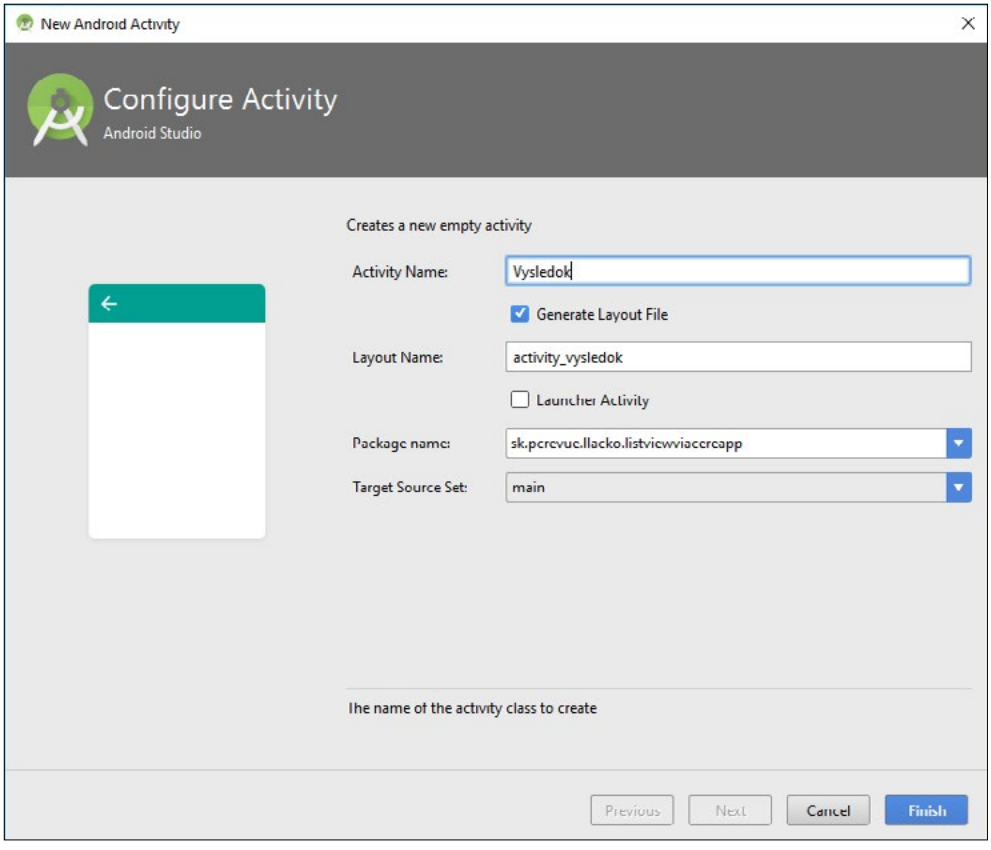
Layout druhé aktivity *vysledek.xml* na zobrazení výsledků obsahuje pouze prvek `ListView` jako kořenový element.

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/vysledekList"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
</ListView>
```

Další aktivitu přidáte do projektu pomocí místní nabídky složky projektu **App** → **New** → **Activity** → **EmptyActivity**.



Obrázek 9.10: Přidání další aktivity do projektu



Obrázek 9.11: Dialog na přidání aktivity do projektu

Aplikační kód

V kódu hlavní aktivity načítáme hodnoty z pole řetězců a odevzdáme ho objektu `listView` přes `ArrayAdapter`. V obsluze události klepnutí na tlačítko vybereme označené položky a uložíme je do pole řetězců `vysledky`. Toto pole odevzdáme přes objekt `Bundle` aktivitě na zobrazení výsledků.

```
public class MainActivity extends AppCompatActivity
  implements View.OnClickListener {

  Button button;
  ListView listView;
  ArrayAdapter<String> adapter;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    listView = (ListView) findViewById(R.id.list);
```

```
    button = (Button) findViewById(R.id.okbutton);

    String[] verze = getResources().getStringArray(R.array.verze_array);
    adapter = new ArrayAdapter<String>(this,
      android.R.layout.simple_list_item_multiple_choice, verze);
    listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
    listView.setAdapter(adapter);
    button.setOnClickListener(this);
  }


  public void onClick(View v)  {
    SparseBooleanArray checked = listView.getCheckedItemPositions();
    ArrayList<String> vybrane = new ArrayList<String>();
    for (int i = 0; i < checked.size(); i++) {
      // Pozice v rámci adapteru
      int position = checked.keyAt(i);
      // pridej oznacene polozky
      if (checked.valueAt(i))
        vybrane.add(adapter.getItem(position));
    }

    String[] vysledky = new String[vybrane.size()];
    for (int i = 0; i < vybrane.size(); i++) {
      vysledky[i] = vybrane.get(i);
    }

    Intent intent = new Intent(getApplicationContext(),
      Vysledok.class);

    Bundle b = new Bundle();
    b.putStringArray("selectedItems", vysledky);
    intent.putExtras(b);
    startActivity(intent);
  }
}
```

Kód aktivity na zobrazení výsledku, tedy položek označených v hlavní aktivitě, bude uložen v souboru *VysledekActivity.java*.

```
public class Vysledok extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout. activity_vysledok);

    Bundle b = getIntent().getExtras();
    String[] resultArr = b.getStringArray("selectedItems");
    ListView lv = (ListView) findViewById(R.id.vysledekList);
```

```
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
          android.R.layout.simple_list_item_1, resultArr);
        lv.setAdapter(adapter);
      }
    }
```

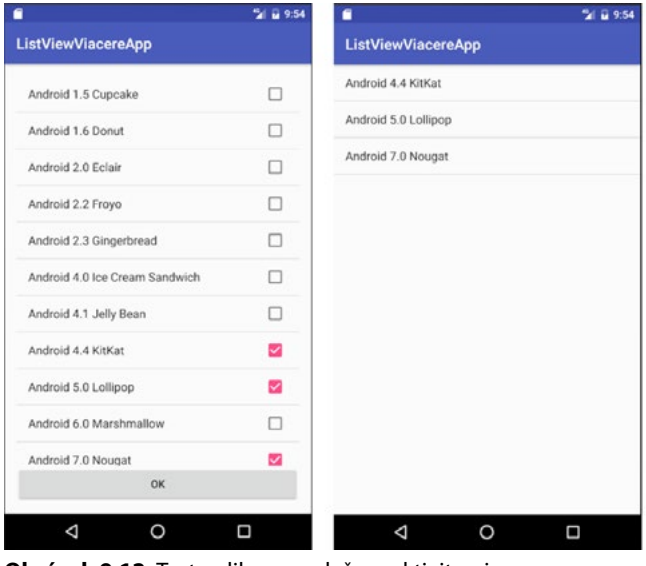
Posledním, avšak velmi důležitým krokem, bez kterého by vám aplikace nefungovala, je přidání aktivity na zobrazení výsledků do manifestu v souboru *AndroidManifest.xml*. Pokud jste aktivitu přidávali přes místní nabídku, vývojové prostředí Android Studio už zařídilo vše potřebné.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.listviewviacereapp">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".Vysledok"></activity>
  </application>
</manifest>
```

Nyní můžete aplikaci spustit a otestovat její fungování.



Obrázek 9.12: Test aplikace se dvěma aktivitami

Příklad – Zobrazení hierarchické struktury

K rychlejší a pohodlnější navigaci ve větším množství položek se často využívá hierarchická struktura. Objekty reálného světa se zpravidla dají různým způsobem kategorizovat, ať už jsou to živočišné druhy, čeledě rostlin, zboží, územní členění a podobně. Proto aplikace uživateli zobrazí seznam na nejvyšší úrovni hierarchické struktury, například kategorie zboží. Po výběru některé z kategorií se hierarchická struktura rozvine a zobrazí se položky patřící do dané kategorie.

Hierarchických úrovní může být více podle toho, o jaké objekty se jedná. V příkladu využijeme `ExpandableListView` připojený k údajům přes `BaseExpandableListAdapter`. Abychom ukázali další funkcionalitu, mazání položek, po rozvinutí kategorie se zobrazí nejen název vnořené položky, ale i ikona s červeným křížkem k vymazání. Jelikož vymazání položky omylem může představovat v závislosti na typu aplikace vážný problém, před vlastním vymazáním si necháme tuto operaci uživatelem potvrdit.

I v tomto příkladu využijeme jako zdroj údajů seznam verzí Androidu. Na nejvyšší úrovni hierarchie budou kategorizované podle první číslice označení verze. Na nejnižší úrovni hierarchie budou zobrazeny položky, v tomto případě verze za tečkou.

Verze 2.x

- 2.0 Eclair
- 2.2 Froyo
- 2.3 GingerBread

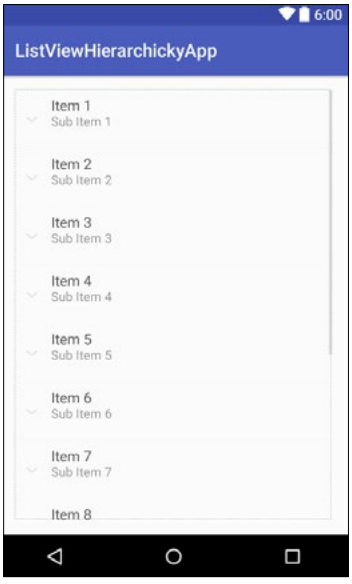
Návrh uživatelského rozhraní

Hlavní aktivita v souboru *activity_main.xml* obsahuje pouze jeden prvek typu `ExpandableListView`.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.listviewhierarchickyapp.MainActivity">

  <ExpandableListView
    android:id="@+id/verze_list"
    android:layout_width="match_parent"
    android:layout_height="fill_parent" >
  </ExpandableListView>
</RelativeLayout>
```

Pro zajímavost si prohlédněte návrh layoutu. V levé části jsou šipky směřující dolů na rozbalení skupin položek.



Obrázek 9.13: V návrhu layoutu si všimněte levé části šipky směřující dolů na rozbalení skupin položek

Na rozdíl od `ListView` je pro rozbalitelný seznam položek potřeba navrhnout oba dva layouty: layout pro zobrazení skupiny položek a layout pro zobrazení vnořené položky.

Layout pro skupinu položek je definován v souboru *skupina_item.xml*.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  android:padding="10dp" >

  <TextView
    android:id="@+id/skupina"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingLeft="25dp"/>
</LinearLayout>
```

Layout pro položku je definován v souboru *polozka_item.xml*. Každá položka obsahuje textové pole s názvem a v pravé části ikonu na vymazání položky.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:padding="10dp" >

  <TextView
    android:id="@+id/polozka"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
```

```
    android:layout_centerVertical="true"
    android:paddingLeft="25dp" />

  <ImageView
    android:id="@+id/vymaz"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:src="@drawable/del"
    android:contentDescription="@string/vymaz"/>

</RelativeLayout>
```

Ikona (červený křížek) na vymazání položky je umístěna ve složce *res/drawable*. V kódu se budeme odkazovat jen na název obrázku bez přípony.

TIP

Obrázky do příslušné složky *drawable* nejsnáze umístíte tak, že soubory s obrázky zkopírujete do schránky v souborovém manažeru (**Ctrl+C**), přepnete se do Android Studia, označíte složku, do které je chcete zkopírovat, a použijete kombinaci kláves **Ctrl+V**.

Aplikační kód

Nejprve vytvoříme třídu `ExpandableListAdapter`, odvozenou od `BaseExpandableListAdapter`, ve které definujeme metody na práci s hierarchickou strukturou.

V obsluze události klepnutí na mazací ikonu se nejprve pomocí dialogového okna typu `AlertDialog` dotážeme uživatele, zda skutečně chce položku vymazat.

Soubor *ExpandableListAdapter.java* (kompletní kód i s importy):

```
import android.app.Activity;
import android.content.Context;
import android.content.DialogInterface;
import android.graphics.Typeface;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseExpandableListAdapter;
import android.widget.ImageView;
import android.widget.TextView;
import android.app.AlertDialog;

import java.util.List;
import java.util.Map;

public class ExpandableListAdapter extends BaseExpandableListAdapter {
  private Activity context;
  private Map<String, List<String>> skupinyOS;
  private List<String> verzeOS;
```

```
public ExpandableListAdapter(Activity context, List<String> verzeOS,
  Map<String, List<String>> skupinyOS) {
  this.context = context;
  this.skupinyOS = skupinyOS;
  this.verzeOS = verzeOS;
}

public Object getChild(int poziceSkupiny, int pozicePolozky)
{
  return skupinyOS.get(verzeOS.get(poziceSkupiny)).get(pozicePolozky);
}

public long getChildId(int poziceSkupiny, int pozicePolozky)
{
  return pozicePolozky;
}


public View getChildView(final int poziceSkupiny, final int pozicePolozky,
  boolean posledniPolozka, View convertView, ViewGroup parent)
{
  final String verze = (String) getChild(poziceSkupiny, pozicePolozky);
  LayoutInflater inflater = context.getLayoutInflater();

  if (convertView == null)
    convertView = inflater.inflate(R.layout.polozka_item, null);
  TextView item = (TextView) convertView.findViewById(R.id.polozka);

  ImageView delete = (ImageView) convertView.findViewById(R.id.vymaz);
  delete.setOnClickListener(new View.OnClickListener()
  {
    public void onClick(View v)
    {
      AlertDialog.Builder builder = new AlertDialog.Builder(context);
      builder.setMessage("Skutečne smazat?");
      builder.setCancelable(false);
      builder.setPositiveButton("Ano",
        new DialogInterface.OnClickListener()
      {
        public void onClick(DialogInterface dialog, int id)
        {
          List<String> child =
          skupinyOS.get(verzeOS.get(poziceSkupiny));
          child.remove(pozicePolozky);
          notifyDataSetChanged();
        }
      });

      builder.setNegativeButton("Ne",
        new DialogInterface.OnClickListener()
      {
```

```
        public void onClick(DialogInterface dialog, int id)
        {
          dialog.cancel();
        }
      });
      AlertDialog alertDialog = builder.create();
      alertDialog.show();
    }
  });
  item.setText(verze);
  return convertView;
}

public Object getGroup(int poziceSkupiny)  {
  return verzeOS.get(poziceSkupiny);
}

public long getGroupId(int poziceSkupiny) {
  return poziceSkupiny;
}

public int getGroupCount() {
  return verzeOS.size();
}

public int getChildrenCount(int poziceSkupiny) {
  return skupinyOS.get(verzeOS.get(poziceSkupiny)).size();
}

public View getGroupView(int poziceSkupiny, boolean jeRozbalena,
  View convertView, ViewGroup parent) {
  String laptopName = (String) getGroup(poziceSkupiny);
  if (convertView == null)  {
    LayoutInflater infalInflater = (LayoutInflater) context
      .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    convertView = infalInflater.inflate(R.layout.skupina_item,
      null);
  }
  TextView item = (TextView) convertView.findViewById(R.id.skupina);
  item.setTypeface(null, Typeface.BOLD);
  item.setText(laptopName);
  return convertView;
}

public boolean hasStableIds()   {
  return true;
}

public boolean isChildSelectable(int poziceSkupiny, int pozicePolozky) {
  return true;
}
```

```
}
```

Kód hlavní aktivity v souboru *MainActivity.java*:

```
public class MainActivity extends AppCompatActivity {

  List<String> skupinyList;
  List<String> polozkyList;
  Map<String, List<String>> skupinyOS;
  ExpandableListView expandListView;

  @Override
  protected void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    vytvorSkupiny();
    vytvorPolozky();

    expandListView = (ExpandableListView) findViewById(R.id.verze_list);
    final ExpandableListAdapter expListAdapter = new ExpandableListAdapter(
      this, skupinyList, skupinyOS);
    expandListView.setAdapter(expListAdapter);

    expandListView.setOnChildClickListener(new
      ExpandableListView.OnChildClickListener() {
      public boolean onChildClick(ExpandableListView parent, View v,
        int groupPosition, int childPosition, long id) {
        final String selected = (String) expListAdapter.getChild(
        groupPosition, childPosition);
        Toast.makeText(getBaseContext(), selected, Toast.LENGTH_LONG)
          .show();
        return true;
      }
    });
  }

  private void vytvorSkupiny() {
    skupinyList = new ArrayList<String>();
    skupinyList.add("Verze 1");
    skupinyList.add("Verze 2");
    skupinyList.add("Verze 3");
    skupinyList.add("Verze 4");
    skupinyList.add("Verze 5");
    skupinyList.add("Verze 6");
    skupinyList.add("Verze 7");
  }

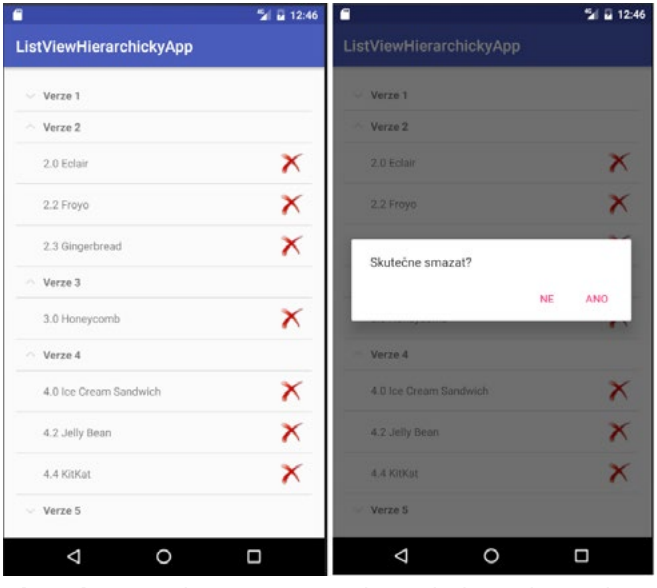
  private void vytvorPolozky() {
    String[] v1 = { "1.5 Cupcake","1.6 Donut"};
    String[] v2 = { "2.0 Eclair", "2.2 Froyo", "2.3 Gingerbread " };
    String[] v3 = { "3.0 Honeycomb"};
    String[] v4 = { "4.0 Ice Cream Sandwich", "4.2 Jelly Bean",
      "4.4 KitKat"};
```

```
    String[] v5 = { "5.0 Lollipop"};
    String[] v6 = { "6.0 Marshmallow"};
    String[] v7 = { "7.0 Nougat"};

    skupinyOS = new LinkedHashMap<String, List<String>>();

    for (String skupina : skupinyList)    {
      if (skupina.equals("Verze 1")) naplnPolozky(v1);
      else if (skupina.equals("Verze 2")) naplnPolozky(v2);
      else if (skupina.equals("Verze 3")) naplnPolozky(v3);
      else if (skupina.equals("Verze 4")) naplnPolozky(v4);
      else if (skupina.equals("Verze 5")) naplnPolozky(v5);
      else if (skupina.equals("Verze 6")) naplnPolozky(v6);
      else if (skupina.equals("Verze 7")) naplnPolozky(v7);
      skupinyOS.put(skupina, polozkyList);
    }
  }

  private void naplnPolozky(String[] ver)  {
    polozkyList = new ArrayList<String>();
    for (String verzia : ver)
      polozkyList.add(verzia);
  }
}
```



Obrázek 9.14: Zobrazení rozvinuté hierarchické struktury (vlevo) a dialog na potvrzení vymazání položky (vpravo)

Seznam objektů s obrázky

V úvodu jsme slíbili, že po vyzkoušení základních principů a funkcí se zaměříme více na design. V reálných aplikacích se pro lepší názornost v položkách vedle textu velmi často zobrazuj i obrázky, přičemž text obsahuje více řádků, například nadpis a podnadpis, přičemž tyto texty jsou vypsány jiným fontem. Také naše cvičné údaje o verzích Androidu si o takovýto způsob přímo říkají – ke každému kódovému označení s názvem zákusku lze přece přiřadit i jeho obrázek, jakési neformální logo verze.

Návrh uživatelského rozhraní

Layout hlavní aktivity už snad ani nemůže být jednodušší – obsahuje pouze prvek `ListView`.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.listviewobrazkyapp.MainActivity">

  <ListView
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
</RelativeLayout>
```

Naproti tomu layout položky v souboru *list_item.xml* je komplikovanější, jelikož je potřeba v každé položce zobrazit obrázek, nadpis a podnadpis. Vzájemnou polohu prvků udržuje vizuální kontejner `RelativeLayout`. Abyste měli představu o poměru velikosti obrázku, který je zadaný v dp, tuto jednotku jsme použili i pro velikost textu, a to přes upozornění vývojového prostředí, abychom pro velikost textu použili jednotky sp.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >

  <ImageView
    android:id="@+id/obrazek"
    android:layout_width="80dp"
    android:layout_height="80dp"
    android:contentDescription="@string/obrazek"
    android:paddingLeft="10dp"
    android:paddingRight="10dp" />

  <TextView
    android:id="@+id/nazev"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@+id/obrazek"
    android:paddingBottom="10dp"
    android:textColor="#0033FF"
    android:textSize="16dp" />
```

```
    <TextView
      android:id="@+id/popis"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@+id/nazev"
      android:layout_toRightOf="@+id/obrazek"
      android:paddingLeft="10dp"
      android:textColor="#3399FF"
      android:textSize="14dp" />

</RelativeLayout>
```

Aplikační kód

Vytvořte třídu `Polozka` zapouzdřující datovou strukturu pro zobrazované položky. Má jen tři atributy: identifikátor obrázku, název položky a její popis a překrytí metody `toString()`.

```
public class Polozka {
  private int obrId;
  private String nazev;
  private String popis;

  public Polozka(int imageId, String title, String desc) {
    this.obrId = imageId;
    this.nazev = title;
    this.popis = desc;
  }
  public int getImageId() {
    return obrId;
  }

  public void setImageId(int imageId) {
    this.obrId = imageId;
  }

  public String getDesc() {
    return popis;
  }

  public void setDesc(String desc) {
    this.popis = desc;
  }

  public String getTitle() {
    return nazev;
  }

  public void setTitle(String title) {
    this.nazev = title;
  }

  @Override
```

```
  public String toString() {
    return nazev + "\n" + popis;
  }
}
```

Vytvořte třídu na `MujBaseAdapter` odvozenou od třídy `android.widget.BaseAdapter`. Bude sloužit k přiřazení údajů k layoutu řádků prvku `ListView`.

```
public class MujBaseAdapter extends BaseAdapter {
  Context context;
  List<Polozka> rowItems;

  public MujBaseAdapter(Context context, List<Polozka> items) {
    this.context = context;
    this.rowItems = items;
  }

  private class ViewHolder  {
    ImageView imageView;
    TextView txtTitle;
    TextView txtDesc;
  }

  public View getView(int position, View convertView, ViewGroup parent)  {
    ViewHolder hld = null;

    LayoutInflater lInflater = (LayoutInflater)
      context.getSystemService(Activity.LAYOUT_INFLATER_SERVICE);
    if (convertView == null) {
      convertView = lInflater.inflate(R.layout.list_item, null);
      hld = new ViewHolder();
      hld.txtDesc = (TextView) convertView.findViewById(R.id.popis);
      hld.txtTitle = (TextView) convertView.findViewById(R.id.nazev);
      hld.imageView = (ImageView) convertView.findViewById(R.id.obrazek);
      convertView.setTag(hld);
    }
    else {
      hld = (ViewHolder) convertView.getTag();
    }
    Polozka polozka = (Polozka) getItem(position);
    hld.txtDesc.setText(polozka.getDesc());
    hld.txtTitle.setText(polozka.getTitle());
    hld.imageView.setImageResource(polozka.getImageId());
    return convertView;
  }

  @Override
  public int getCount() {
    return rowItems.size();
  }

  @Override
```

```
    public Object getItem(int position) {
      return rowItems.get(position);
    }

    @Override
    public long getItemId(int position)  {
      return rowItems.indexOf(getItem(position));
    }
  }
```

Nakonec jsme se dostali ke kódu hlavní aktivity. Aby byla aplikace co nejjednodušší, jsou v hlavní aktivitě definovány i statické textové řetězce pro jednotlivé položky (tedy pro název verze a informaci o kódovém označení a číslování API), která budou v položce popis. Identifikátory obrázků jsou v poli hodnot typu `integer`.

Obrázky jsou umístěny ve složce *res/drawable*. V našem případě se jedná o ilustrační ikony, takže jsme obrázky umístili do složky pro jedno rozlišení. V kódu se budeme odkazovat jen na název obrázku bez přípony. Název obrázku musí začínat písmenem.

Kompletní kód v souboru *MainActivity.java*.

```
public class MainActivity extends AppCompatActivity implements
  AdapterView.OnItemClickListener {

  public static final String[] verze = new String[] {
    "1.5 Cupcake", "1.6 Donut", "2.0 Eclair", "2.2 Froyo", "2.3 Gingerbread",
    "3.0 Honeycomb","4.0 Ice Cream Sandwich", "4.1 Jelly Bean", "4.4 KitKat",
    "5.0 Lollipop", "6.0 Marshmallow", "7.0 Nougat"
  };

  public static final String[] api = new String[]{
    "Cupcake, API 3", "Donut, API 4", "Eclair, API 5", "Froyo, API 8",
    "Gingerbread, API 9", "Honeycomb, API 11", "Ice Cream Sandwich, API 15",
    "Jelly Bean, API 16", "KitKat, API 19", "Lollipop, API 21",
    "Marshmallow, API 23", "Nougat, API 24"
  };

  public static final Integer[] images =  {
    R.drawable.cupcake, R.drawable.donut, R.drawable.eclair,
    R.drawable.froyo, R.drawable.gingerbread, R.drawable.honeycomb,
    R.drawable.icecream, R.drawable.jellybean, R.drawable.kitkat,
    R.drawable.lollipop, R.drawable.marshmallow, R.drawable.nougat,
  };

  ListView listView;
  List<Polozka> polozkySeznamu;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

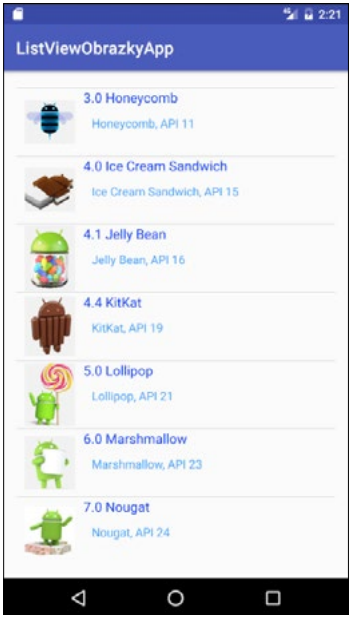
    polozkySeznamu = new ArrayList<Polozka>();
```

```
    for (int i = 0; i < verze.length; i++) {
      Polozka item = new Polozka(images[i], verze[i], api[i]);
      polozkySeznamu.add(item);
    }

    listView = (ListView) findViewById(R.id.list);
    MujBaseAdapter adapter = new MujBaseAdapter(this, polozkySeznamu);
    listView.setAdapter(adapter);
    listView.setOnItemClickListener(this);
  }

  @Override
  public void onItemClick(AdapterView<?> parent, View view, int pozice,
    long id)  {
    Toast toast = Toast.makeText(getApplicationContext(),
    (pozice + 1) + ": " + polozkySeznamu.get(pozice),
    Toast.LENGTH_SHORT);
    toast.setGravity(Gravity.BOTTOM|Gravity.CENTER_HORIZONTAL, 0, 0);
    toast.show();
  }
}
```

Nakonec aplikaci vyzkoušejte. Porovnejte, jak působí na uživatele v porovnání s technokratickým pojetím seznamu položek v prvním příkladu této kapitoly.



Obrázek 9.15: Test fungování aplikace na zobrazení seznamu s obrázky a strukturovaným textem

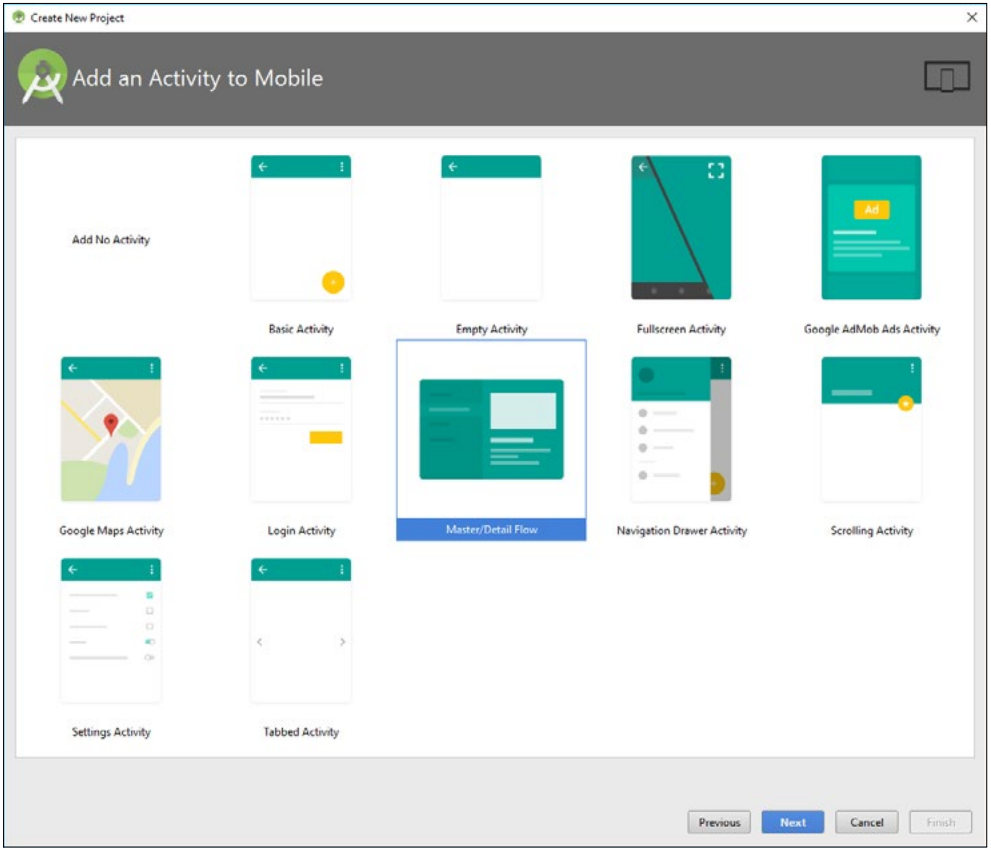
Příklad – Aktivita Master/Detail Flow

Hodně aplikací využívá zobrazení typu master-detail, tedy seznam položek spolu s detaily o vybrané položce. Dobrým příkladem je e-mailový klient, který zpravidla vlevo obsahuje seznam zpráv a vpravo zobrazí detaily k vybrané zprávě. Takto se struktura master-detail zobrazuje tehdy, pokud je k tomu dostatek místa, například na tabletu či telefonu s displejem na šířku. Pokud dostatek místa není, například na displeji telefonu na výšku, zobrazí se nejprve seznam a po klepnutí na položku se zobrazí nová aktivita s detaily o vybrané položce.

Takovýto seznam jsme vytvářeli i v kapitole věnované fragmentům. Při vytváření aplikací v Android Studiu jste si určitě všimli, že toto vývojové prostředí nabízí více typů hlavní aktivity a mezi nimi i typ **Master/Detail Flow**.

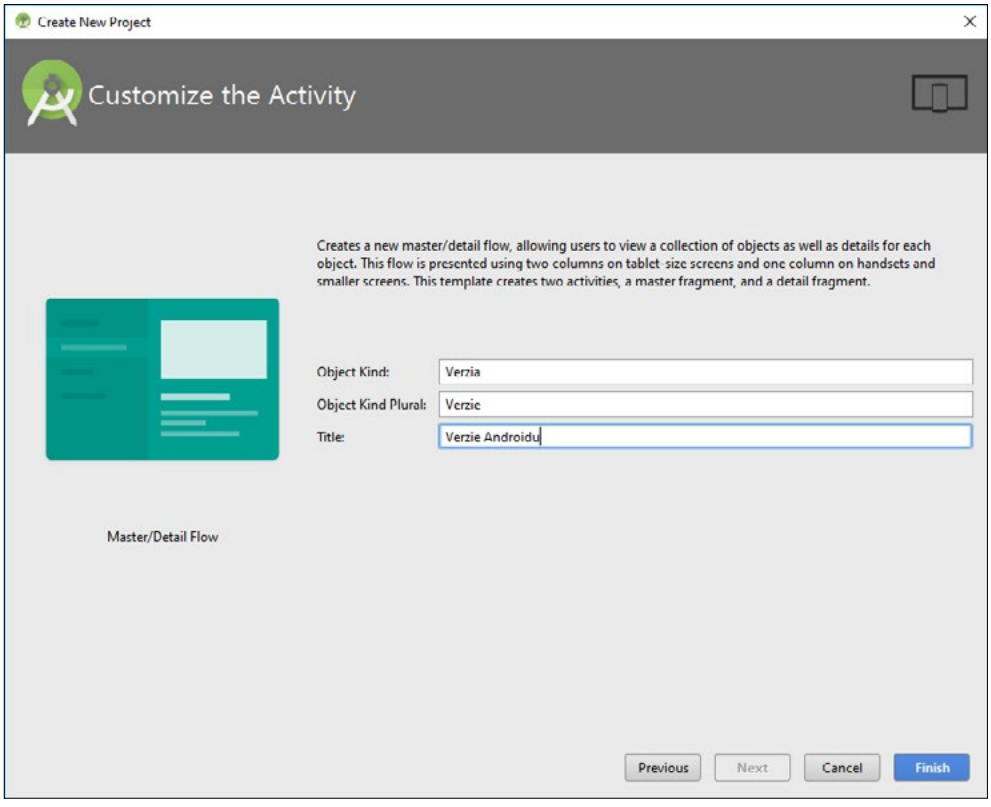
Vytvoření projektu

Vytvořte projekt nové aplikace a vyberte tento typ hlavní aktivity.



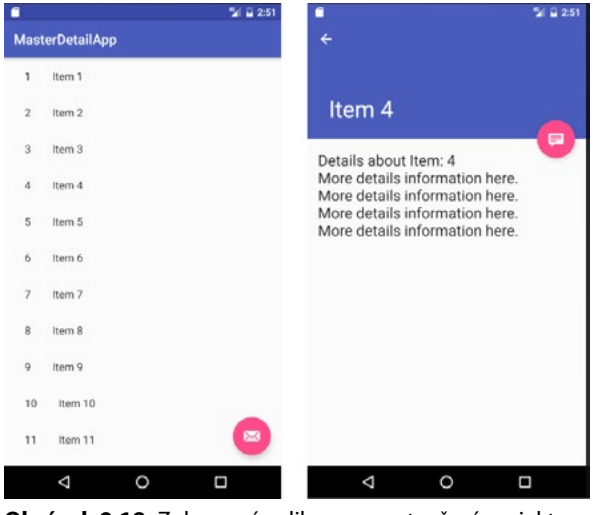
Obrázek 9.16: Výběr hlavní aktivity typu Master/Detail Flow

Následně zadejte názvy položek ve tvaru jednotného i množného čísla. Například Zpráva – zprávy, Položka – položky, Kniha – knihy, Skladba – skladby... a název seznamu.

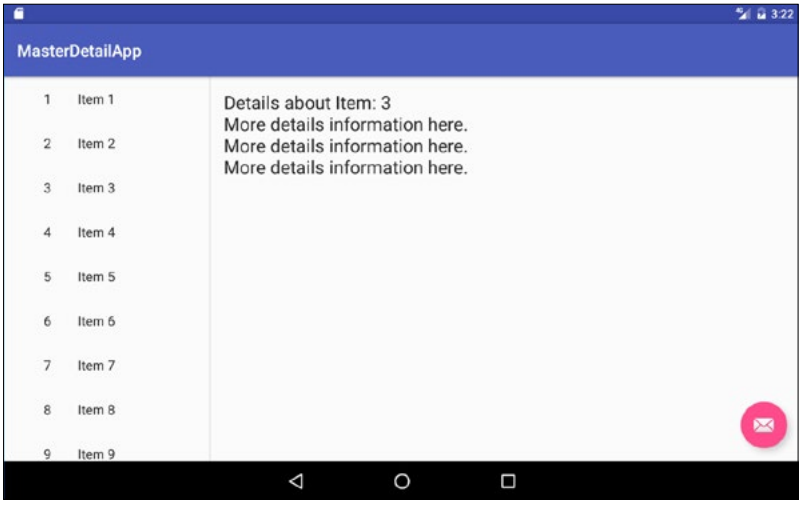


Obrázek 9.17: Zadání názvu položek

Po vytvoření projektu doporučujeme aplikaci spustit a prohlédnout si, jak funguje s demo údaji.



Obrázek 9.18: Zobrazení aplikace po vytvoření projektu na telefonu



Obrázek 9.19: Zobrazení aplikace po vytvoření projektu na telefonu

Struktura projektu

Následně je užitečné seznámit se s jeho strukturou a účelem klíčových souborů, které projekt tvoří. V našich názvech `<položka>` je název objektu, tedy textový řetězec, který jste při vytváření projektu v dialogu **Customize the Activity** zadali do pole **Object Kind**. V našem příkladu jsme zadali název `Verzia` – `verzie`:

- *<položka>ListActivity.java* obsahuje kód hlavní aktivity, jejíž úlohou je zobrazit seznam (master) položek. Způsob zobrazení je definován v souboru *activity_<položka>_list.xml*. Pokud kód v třídě identifikuje, že displej zařízení, na kterém je aplikace spuštěná, je dostatečně velký a má dostatečné rozlišení na zobrazení dvou panelů, tedy seznamu i detailů o vybrané položce vedle sebe, vytvoří se a zobrazí instance `<položka>_list fragment` v rámci aktivity *activity_<položka>_twopane.xml*. Pokud displej zařízení není dostatečně velký, zobrazí se aktivita *activity_<položka>_list.xml*. Tato třída také implementuje `onItemSelected()` callback metodu, která se bude volat, pokud si uživatel vybere položku ze seznamu master. Také zobrazí panel s detaily. V případě zobrazení na dvou panelech vedle sebe se vytvoří instance třídy `<položka>DetailFragment class` a fragment se přidá do aktivity. Na menších displejích se zobrazí sekundární aktivita `<položka>DetailActivity`.
- Složka *<položka>_list.xml* obsahuje dva soubory definice layoutů pro úzký a široký displej nad 900 dp, které obsahují dva, resp. jeden fragment.
- *<položka>ListFragment.java* obsahuje třídu, jejíž metody zpracovávají výběr, včetně zvýraznění vybrané položky.
- *<položka>DetailActivity.java* obsahuje kód na zobrazování sekundární aktivity na úzkých displejích.
- *<položka>DetailFragment.java* – metody třídy načítají údaje pro zobrazení podrobností o vybraném objektu.

- *DummyContent.java* – třída generuje demo data na zobrazení. Nejjednodušší cesta k přizpůsobení projektu vede přes nahrazení této třídy třídou, která zprostředkuje reálné údaje k zobrazení v aplikaci.

Přizpůsobení

Jak jsme avizovali přizpůsobení šablony projektů, začneme třídou *DummyContent.java*, která zprostředkuje údaje. Naším cílem je zobrazit seznam verzí Androidu a k nim webové stránky portálu *developer.android.com* s popisy novinek příslušné verze.

Původní obsah třídy *DummyContent.java* po vytvoření projektu:

```
public class DummyContent {

  /**
   * An array of sample (dummy) items.
   */
  public static final List<DummyItem> ITEMS = new ArrayList<DummyItem>();

  /**
   * A map of sample (dummy) items, by ID.
   */
  public static final Map<String, DummyItem> ITEM_MAP =
    new HashMap<String, DummyItem>();

  private static final int COUNT = 25;

  static {
    // Add some sample items.
    for (int i = 1; i <= COUNT; i++) {
      addItem(createDummyItem(i));
    }
  }

  private static void addItem(DummyItem item) {
    ITEMS.add(item);
    ITEM_MAP.put(item.id, item);
  }

  private static DummyItem createDummyItem(int position) {
    return new DummyItem(String.valueOf(position), "Item " + position,
    makeDetails(position));
  }

  private static String makeDetails(int position) {
    StringBuilder builder = new StringBuilder();
    builder.append("Details about Item: ").append(position);
    for (int i = 0; i < position; i++) {
      builder.append("\nMore details information here.");
    }
    return builder.toString();
  }
```

```
  /**
   * A dummy item representing a piece of content.
   */
  public static class DummyItem {
    public final String id;
    public final String content;
    public final String details;

    public DummyItem(String id, String content, String details) {
      this.id = id;
      this.content = content;
      this.details = details;
    }

    @Override
    public String toString() {
      return content;
    }
  }
}
```

Začneme s úpravami. Potřebujeme údaje ve tvaru název verze – ten se zobrazí v části Master – a URL webové stránky, jejíž obsah se zobrazí v části Slave.

```
public class DummyContent {
  public static final List<DummyItem> ITEMS = new ArrayList<DummyItem>();

  public static Map<String, DummyItem> ITEM_MAP =
    new HashMap<String, DummyItem>();

  static {
    addItem(new DummyItem("1", "Nougat",
    "https://developer.android.com/about/versions/nougat/index.html"));
    addItem(new DummyItem("2", "Marshmallow",
    "https://developer.android.com/about/versions/marshmallow/index.htmlm"));
    addItem(new DummyItem("3", "Lollipop",
    "https://developer.android.com/about/versions/lollipop.html"));
    addItem(new DummyItem("4", "KitKat",
    "https://developer.android.com/about/versions/kitkat.html"));
    addItem(new DummyItem("5", "JellyBean",
    "https://developer.android.com/about/versions/jelly-bean.html"));
  }

  private static void addItem(DummyItem item) {
    ITEMS.add(item);
    ITEM_MAP.put(item.id, item);
  }

  private static DummyItem createDummyItem(int position) {
    return new DummyItem(String.valueOf(position), "Item " + position,
      makeDetails(position));
```

```
    }

    private static String makeDetails(int position) {
      StringBuilder builder = new StringBuilder();
      builder.append("Details about Item: ").append(position);
      for (int i = 0; i < position; i++) {
        builder.append("\nMore details information here.");
      }
      return builder.toString();
    }

    public static class DummyItem {
      public String id;
      public String website_name;
      public String website_url;

      public DummyItem(String id, String website_name,
        String website_url)
      {
        this.id = id;
        this.website_name = website_name;
        this.website_url = website_url;
      }

      @Override
      public String toString() {
        return website_name;
      }
    }
  }
```

V souboru *verzia_detail.xml* navrhneme jednoduchý layout na zobrazení webové stránky. Prvek `TextView` nahradíme prvkem `WebView`.

```
<WebView
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/website_detail"></WebView>
```

Také je potřeba modifikovat kód třídy fragmentu v souboru `VerziaDetailFragment.java`.

```
public class VerziaDetailFragment extends Fragment {

  public static final String ARG_ITEM_ID = "item_id";

  private DummyContent.DummyItem mItem;

  public VerziaDetailFragment() {
  }

  @Override
  public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);

    if (getArguments().containsKey(ARG_ITEM_ID)) {
      mItem =
        DummyContent.ITEM_MAP.get(getArguments().getString(ARG_ITEM_ID));

      Activity activity = this.getActivity();
      CollapsingToolbarLayout appBarLayout = (CollapsingToolbarLayout)
        activity.findViewById(R.id.toolbar_layout);
      if (appBarLayout != null) {
        appBarLayout.setTitle(mItem.website_name);
      }
    }
  }

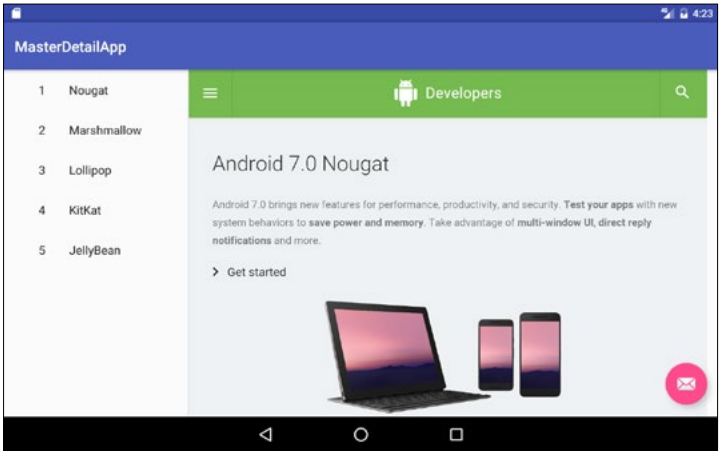
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View rootView = inflater.inflate(R.layout.verzia_detail,
    container, false);

    if (mItem != null) {
      ((WebView) rootView.findViewById(R.id.website_detail))
        .loadUrl(mItem.website_url);
    }
    return rootView;
  }
}
```

Aplikace přistupuje k internetu, nezapomeňte proto přidat do aplikačního manifestu příslušné oprávnění.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Nyní můžete aplikaci vyzkoušet.



Obrázek 9.20: Zobrazení aplikace na tabletu

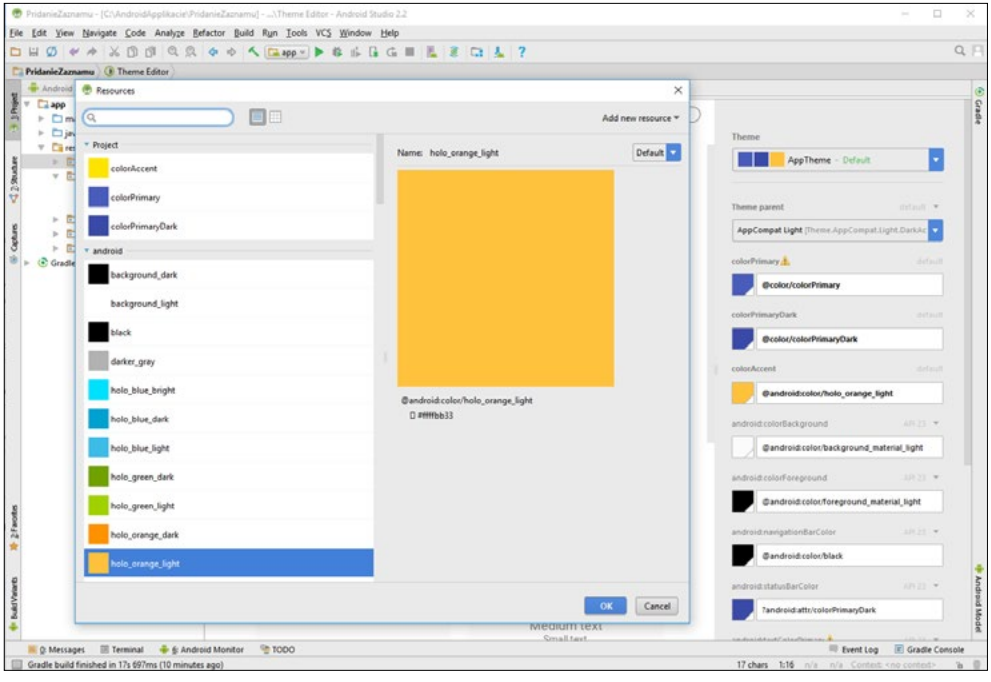
Přidání objektu do seznamu

V předchozí kapitole najdete popis využití tlačítka Floating Action Button a spodní lišty Snackbar, které jsou součástí šablony aktivity **Basic Activity**. Toto tlačítko využijeme jako ovládací prvek na přidání záznamu. Proto při vytvoření projektu využijte tento typ hlavní aktivity. Do složky *drawable* překopírujte vhodnou ikonu pro tlačítko; v tomto případě se pro přidání záznamu zaběhla ikona se symbolem „+“ (plus). S námětem si těžkou hlavu dělat nebudeme, budeme přidávat náhodná čísla v intervalu 0–6.

V souboru *activity_main* změňte přiřazení obrázku.

```
<android.support.design.widget.FloatingActionButton
  android:id="@+id/fab"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="bottom|end"
  android:layout_margin="@dimen/fab_margin"
  app:srcCompat="@drawable/ico_add" />
```

Po přepnutí do návrhového zobrazení zjistíte, že ikona má stále pro tento úkon netypickou růžovou barvu pozadí. Lepší bude sytě žlutá nebo oranžová. Pomocí nabídky **Tools → Android → Theme Editor** otevřete editor barevného schématu aplikace a změňte barvu **colorAccent** například na **holo_orange_light**.

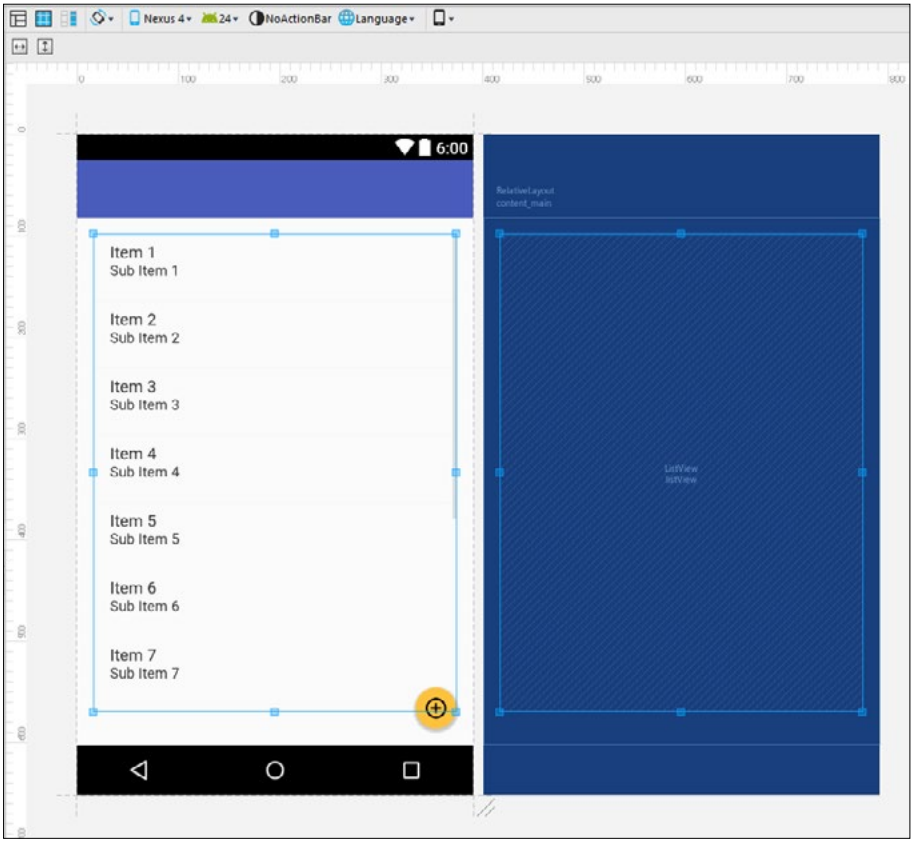


Obrázek 9.21: Změna barvy v editoru barevného schématu aplikace

V tomto stadiu máte pěknou ikonu na přidání záznamu, která ale zatím nic užitečného nedělá. Zatím v aplikaci nemáme ani seznam záznamů. Ten, přesněji návrh jeho zobrazení, přidáme do souboru *content_main.xml*.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/content_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  app:layout_behavior="@string/appbar_scrolling_view_behavior"
  tools:context="sk.pcrevue.llacko.pridaniezaznamu.MainActivity"
  tools:showIn="@layout/activity_main">

  <ListView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/listView"
    android:layout_marginBottom="16dp" />
</RelativeLayout>
```



Obrázek 9.22: Uživatelské rozhraní aplikace v návrhovém zobrazení

V kódu hlavní aktivity vytvoříme seznam a kód, který po stisknutí tlačítka přidá novou položku. Ponecháme možnost nově přidanou položku zrušit. K tomuto účelu využijeme tlačítko na liště Snackbar.

```
public class MainActivity extends AppCompatActivity {

  ArrayList<String> listItems = new ArrayList<String>();
  ArrayAdapter<String> adapter;
  private ListView lvZoznam;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    lvZoznam = (ListView) findViewById(R.id.listView);

    adapter = new ArrayAdapter<String>(this,
      android.R.layout.simple_list_item_1,
      listItems);
    lvZoznam.setAdapter(adapter);

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        addListItem();
        Snackbar.make(view, "Položka pridaná", Snackbar.LENGTH_LONG)
          .setAction("Zrušené", undoOnClickListener).show();
      }
    });
  }

  View.OnClickListener undoOnClickListener = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
      listItems.remove(listItems.size() -1);
      adapter.notifyDataSetChanged();
      Snackbar.make(view, "Odobrané", Snackbar.LENGTH_LONG)
        .setAction("Akcia", null).show();
    }
  };

  private void addListItem() {

    Random rnd = new Random();
    int nCislo = rnd.nextInt(6);
    listItems.add(String.valueOf(nCislo));
    adapter.notifyDataSetChanged();
```

```
    }

    ...
}
```



Obrázek 9.23: Test aplikace

Seznamy objektů využívající RecyclerView a CardView

Hodně mobilních aplikací potřebuje poutavou grafickou formou zobrazovat různé objekty, ať už je to nabídka zboží, seznam restaurací, seznam událostí a podobně. K tomuto účelu se využívají – a nyní si půjčíme terminologii z konkurenčního operačního systému Windows – takzvané dlaždice nebo kartičky stejné či různé velikosti, přičemž každá z nich obsahuje textové a obrazové informace o jednom objektu. Objektů je téměř vždy více, než je prostoru na displeji, takže kartičky – či dlaždice – jsou umístěny jakoby na virtuální ploše a displej tvoří jen výřez – nebo pohled na tuto plochu.

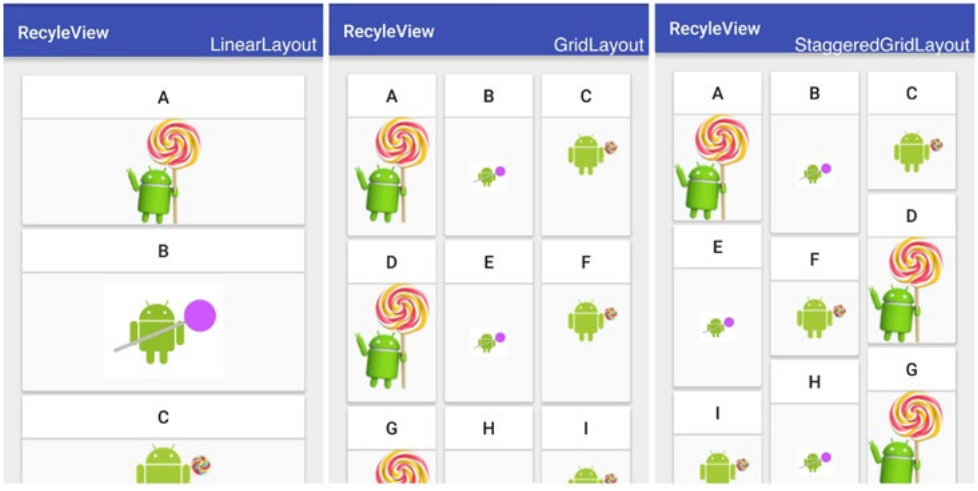
RecyclerView

Prvek `ListView` na zobrazení seznamů objektů má některá omezení, takže se k tomuto účelu nehodí. `RecyclerView`, jak vyplývá z názvu, opakovaně používá jednotlivé prvky, které přesouvá na obrazovku nebo mimo ni, namísto aby se vytvářely nové, takže mnohem lépe hospodaří s pamětí zařízení.

`RecyclerView` umožňuje využít tři způsoby standardního rozmístění prvků:

- `LinearLayoutManager` – klasický seznam objektů, který se dá posouvat vodorovně nebo svisle

- `GridLayoutManager` – matice nebo mřížka objektů stejné velikosti
- `StaggeredGridLayoutManager` – mřížka objektů, které mohou mít různou velikost



Obrázek 9.24: Zleva doprava: LinearLayoutManager, GridLayoutManager, StaggeredGridLayoutManager

Pokud nevyhovuje žádný ze způsobů standardního rozmístění prvků, dá se využít rozvržení na míru s využitím třídy `RecyclerView.LayoutManager`.

Položky seznamu zobrazované pomocí `RecyclerView` jsou implementovány jako instance třídy `ViewHolder`. Obsahují informace o tom, co a jak se má zobrazit. Údaje jsou napojeny přes adaptér `RecyclerView.Adapter`. Ten musí mít implementované tyto metody:

- `getItemCount()` – vrátí počet objektů, které se zobrazí v seznamu.
- `onCreateViewHolder()` – metoda vytvoří a vrátí objekt `ViewHolder`. Rozložení entit je definováno pomocí XML layout souboru.
- `onBindViewHolder()` – naplní prvek definovaným rozvržením textu a grafiky tak, jak se zobrazí uživateli.

CardView

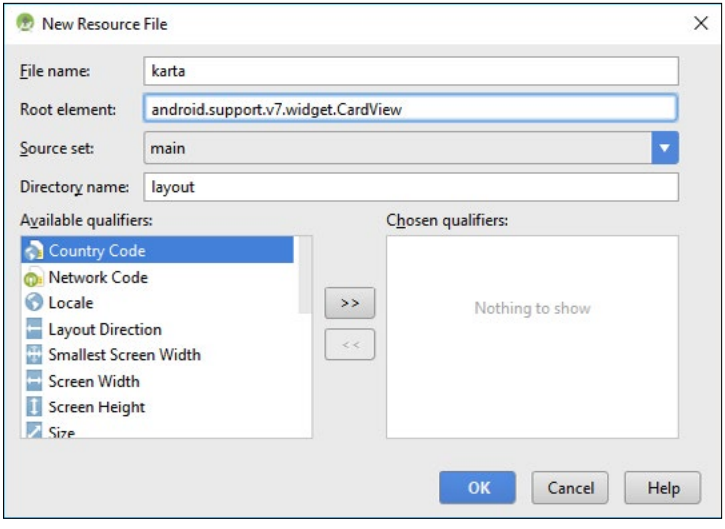
Třída `CardView` umožňuje prezentovat informace na kartách. Tyto karty – nebo pro migrující z Windows dlaždice – se potom zobrazují v seznamech pomocí instance `RecyclerView`. `CardView` umožňuje zatraktivnit zobrazování na kartách stínováním, zaoblenými rohy a dalšími vizuálními efekty.

Příklad RecyclerView a CardView

Vytvořte projekt aplikace s hlavní aktivitou typu **Basic Activity**. Tlačítko Floating Action Button můžete odstranit, protože v aplikaci nebude zapotřebí, a kód tak bude přehlednější. Prvním krokem je přidání knihoven, které se mají k projektu přilinkovat, do souboru *build.gradle* (**module:app**) do sekce `dependencies`. Aby byly knihovny kompatibilní, všechny musí mít stejné číslo verze.

```
dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2',
{
    exclude group: 'com.android.support', module: 'support-annotations'
  })
  compile 'com.android.support:appcompat-v7:24.2.1'
  compile 'com.android.support:design:24.2.1'
  compile 'com.android.support:recyclerview-v7:24.1.1'
  compile 'com.android.support:cardview-v7:24.1.1'
  testCompile 'junit:junit:4.12'
}
```

Pokračujeme návrhem layoutu karty. Soubor s layoutem ve složce *res* nejrychleji a nejjednodušeji vytvoříte pomocí položky místní nabídky **Layout resource file**. Do pole **Root element** zadejte `android.support.v7.widget.CardView`.



Obrázek 9.25: Vytvoření layoutu karty

Kořenovým elementem layoutu je `<android.support.v7.widget.CardView>`. Pomocí parametrů definujete barvu pozadí, zaoblení rohů kartiček se zobrazenými informacemi, 3D efekt a podobně. Uvnitř `CardView` jsou prvky, v tomto případě nadpis, text informace a obrázek, zapouzdřené ve vizuálním kontejneru `RelativeLayout`.

Soubor *karta.xml*:

```
<android.support.v7.widget.CardView
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:card_view="http://schemas.android.com/apk/res-auto"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:id="@+id/card_view"
  android:layout_margin="5dp"
  card_view:cardBackgroundColor="@color/cardview_light_background"
```

```
    card_view:cardCornerRadius="8dp"
    card_view:cardElevation="4dp"
    card_view:contentPadding="5dp"
    android:foreground="?selectableItemBackground"
    android:clickable="true">

    <RelativeLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:padding="16dp" >

      <ImageView
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:id="@+id/item_obr"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginEnd="16dp"
      />

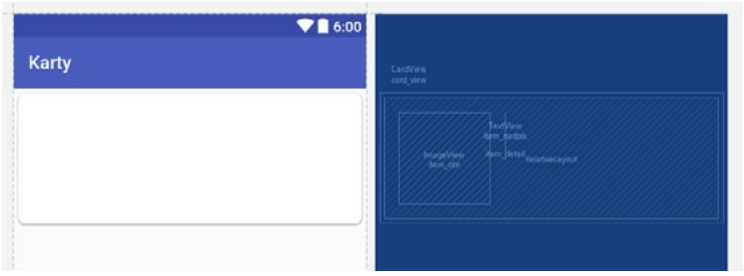
      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/item_nadpis"
        android:layout_toEndOf="@+id/item_obr"
        android:layout_alignParentTop="true"
        android:textSize="30sp"
      />

      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/item_detail"
        android:layout_toEndOf="@+id/item_obr"
        android:layout_below="@+id/item_nadpis"
      />
    </RelativeLayout>
```

Do souboru *activity_main.xml* přidáme prvek `RecyclerView`.

```
<android.support.v7.widget.RecyclerView
  android:id="@+id/recycler_view"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  app:layout_behavior="@string/appbar_scrolling_view_behavior" />
```

Pokračujeme vytvořením adaptéru, který by v reálné aplikaci zprostředkoval údaje nejčastěji z webové databáze, synchronizované SQLite lokální databáze či jiných zdrojů. V tomto případě adaptér poskytuje statické údaje, které má ve svém kódu. Jak už bylo zmíněno, v kódu adaptéru je potřeba vytvořit objekt `ViewHolder` obsahující informace, co a jak se má zobrazit. `ViewHolder` musí mít implementované metody `onCreateViewHolder()`, `onBindViewHolder()` a `getItemCount()`.



Obrázek 9.26: Návrh layoutu karty

Soubor *RecyclerAdapter.java* (seznamy údajů a obrázků ve výpisu jsou zkrácené):

```
public class RecyclerAdapter extends
  RecyclerView.Adapter<RecyclerAdapter.ViewHolder> {

  private String[] verzie = new String[] {
    "1.5 Cupcake",
    "1.6 Donut"...};

  private String[] popis = new String[]{
    "malý muffin v košíčku so šľahačkou – 30.4.2009",
    "šiška s otvorom – 1.9.2009",... };

  private int[] obrazky = { R.drawable.beta,
    R.drawable.cupcake,
    R.drawable.donut,... };

  @Override
  public ViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
    View v = LayoutInflater.from(viewGroup.getContext())
      .inflate(R.layout.karta, viewGroup, false);
    ViewHolder viewHolder = new ViewHolder(v);
    return viewHolder;
  }

  @Override
  public void onBindViewHolder(ViewHolder viewHolder, int i) {
    viewHolder.tit.setText(verzie[i]);
    viewHolder.det.setText(popis[i]);
    viewHolder.obr.setImageResource(obrazky[i]);
  }

  @Override
  public int getItemCount() {
    return verzie.length;
  }

  class ViewHolder extends RecyclerView.ViewHolder {

    public ImageView obr;
```

```
    public TextView tit, det;

    public ViewHolder(View itemView) {
      super(itemView);
      obr = (ImageView)itemView.findViewById(R.id.item_obr);
      tit = (TextView)itemView.findViewById(R.id.item_nadpis);
      det = (TextView)itemView.findViewById(R.id.item_detail);
      itemView.setOnClickListener(new View.OnClickListener() {
        @Override public void onClick(View v) {
          int position = getAdapterPosition();

          Snackbar.make(v, "Vybraná položka " + position,
            Snackbar.LENGTH_LONG)
            .setAction("Action", null).show();

        }
      });
    }
  }
}
```

V kódu hlavní aktivity se inicializuje `RecyclerView`.

```
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;

public class MainActivity extends AppCompatActivity {

  RecyclerView recyclerView;
  RecyclerView.LayoutManager layoutManager;
  RecyclerView.Adapter adapter;

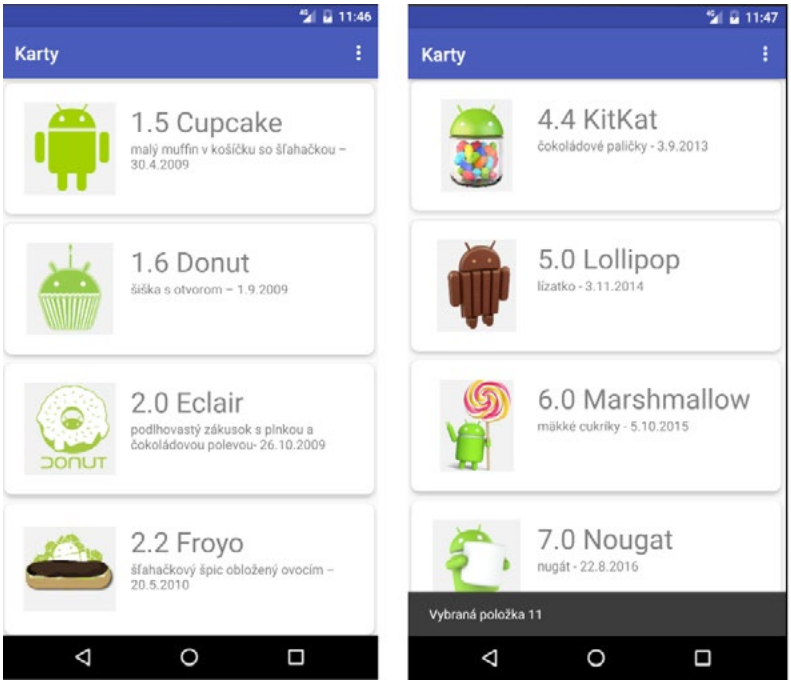
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    recyclerView =
      (RecyclerView) findViewById(R.id.recycler_view);

    layoutManager = new LinearLayoutManager(this);
    recyclerView.setLayoutManager(layoutManager);

    adapter = new RecyclerAdapter();
    recyclerView.setAdapter(adapter);
  }

  ...
}
```



Obrázek 9.27: Aplikace využívající RecyclerView a CardView

KAPITOLA

10

Databáze a práce s údaji

Témata kapitoly:

- Ukládání údajů
- Ukládání údajů do souboru
- Ukládání dočasných souborů
- Databáze SQLite
- Pokusy se zabudovanou databází SQLite v emulátoru
- Interakce aplikace s databází
- Příklad – Čtenářský deník
- Údaje z webu ve formátech JSON a XML
- Statická data ve formátu XML

Ukládání údajů

I mobilní aplikace mohou pracovat s relativně velkými objemy dat, které mohou být uloženy lokálně nebo na internetových serverech. Proto Android obsahuje několik tříd na ukládání menšího množství údajů a má i implementovanou lokální databázi SQLite. V druhé části kapitoly ukážeme práci s formáty JSON a XML. Do této problematiky patří i cloudová služba Firebase, kterou představíme v samostatné kapitole.

Android poskytuje několik základních možností ukládání údajů:

- **Sdílené vlastnosti** – (SharedPreferences) uložení soukromých údajů základních (primitive) typů ve formě párů klíč-hodnota
- **Interní úložiště** – uložení soukromých údajů do vnitřní paměti zařízení (včetně uložení do tzv. dočasných cache)
- **Externí úložiště** – uložení veřejných údajů do přídavné paměti (pevně zabudované v zařízení, resp. na SD kartu)
- **SQLite** – uložení údajů do databáze přímo v zařízení s Androidem
- **Síťové úložiště** – uložení údajů na webový server
- **Poskytovatelé obsahu** – určené na zpřístupnění soukromých údajů jiným aplikacím

Třída SharedPreferences

Třída `SharedPreferences` je nejjednodušší formou uchovávání údajů a vlastností aplikací. Umožňuje aplikacím ukládání a správu malého množství jednoduše strukturovaných údajů. Je implementovaná jako perzistentní mapa k ukládání párových hodnot typu klíč-hodnota. Podporované jsou základní datové typy jako `string`, `boolean`, `int`, `long` a `float`.

Inicializační metoda `Context.getSharedPreferences()`, která je určená k získání instance objektu SP, pracuje se dvěma argumenty. Prvním je název souboru XML, do kterého se budou ukládat páry vlastností, druhým je tzv. operační mód:

- `MODE_APPEND` – nové vlastnosti doplní původní vlastnosti
- `MODE_ENABLE_WRITE_AHEAD_LOGGING` – značka pro otevření databáze
- `MODE_MULTI_PROCESS` – kontrola změn údajů po předešlém načítání SP
- `MODE_PRIVATE` – soubor XML bude přístupný pouze pro konkrétní aplikaci
- `MODE_WORLD_READABLE/WRITEABLE` – z/do souboru XML mohou číst/zapisovat i jiné aplikace

Údaje uložené přes `SharedPreferences` jsou perzistentní, takže údaje uložené během spuštění aplikace budou dostupné i po jejím ukončení a restartu. Typicky je tato třída využívaná k dlouhodobému ukládání dat aplikací, například konfigurace, nejvyššího dosaženého skóre hry a podobně.

Pokud aktivita potřebuje získat objekt `SharedPreferences`, využije metodu `Activity.getPreferences(int mode)`, kde parametr definuje přístupový mód, například `MODE_PRIVATE` pro privátní údaje aplikace.

Pokud chcete získat přístup k objektu `SharedPreferences`, který není spojený s konkrétní aktivitou, je k dispozici metoda `Context.getSharedPreferences(String name, int mode)`, kde parametr `name` je název souboru `SharedPreferences`.

Pro zápis je potřeba pomocí metody `SharedPreferences.edit()` získat instanci objektu `SharedPreferences.Editor`. Přes tento objekt potom můžete přidat nebo změnit hodnoty prostřednictvím metod:

- `putInt(String key, int value)`
- `putString(String key, String value)`
- `remove(String key)`

Následně je potřeba změny potvrdit, v databázové terminologii provést commit, aby se natrvalo zapsaly pomocí metody `SharedPreferences.Editor.commit()`. Po potvrzení údajů je možné aplikaci ukončit a změny budou natrvalo zapsány.

Ke čtení údajů jsou k dispozici metody:

- `getAll()`
- `getBoolean(String key, …)`
- `getString(String key, …)`

Příklad – Uložení nejvyššího dosaženého skóre hry

Námětem příkladu je ukládání údajů, a ne hra samotná, proto bude aplikace simulovat hru tak, že po stisknutí tlačítka vygeneruje skóre jako náhodné číslo v intervalu 1–1 000. Pokud bude toto číslo větší než předtím uložená hodnota, přepíše ji. Po restartu aplikace se zobrazí nejvyšší dosud dosažené skóre.

Vzájemná poloha prvků uživatelského rozhraní hlavní aktivity je udržovaná pomocí kontejneru `RelativeLayout`.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="sk.pcrevue.llacko.ukladanieskore.MainActivity">

  <TextView
    android:id="@+id/top_label"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Top skóre:"
    android:textSize="30sp" />

  <TextView
    android:id="@+id/top_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@id/top_label"
    android:layout_toRightOf="@id/top_label"
    android:textSize="30sp" />

  <TextView
    android:id="@+id/skore_text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:gravity="center"
    android:textSize="60sp" />

  <Button
    android:id="@+id/nuluj_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_gravity="bottom"
    android:text="Nuluj" >
  </Button>

  <Button
    android:id="@+id/hra_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_above="@id/nuluj_button"
    android:text="Hra" >
  </Button>
</RelativeLayout>
```

Definice řetězců je v souboru *strings.xml*. Když uživatel klepne na tlačítko **HRA**, obslužný kód vygeneruje nové skóre, které se ukládá do proměnné `skore`. Pokud je tato hodnota vyšší než top skóre uložené v `SharedPreferences`, musíme uloženou hodnotu přepsat aktuální. Zavolá se metoda `edit()`, která vrátí objekt `SharedPreferences.Editor`. Pomocí metody `putInt()` se hodnota uloženého top skóre aktualizuje.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  private static String TOP_SKORE = "top_skore";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final SharedPreferences sharedPreferences = getPreferences(MODE_PRIVATE);
    final TextView topSkore = (TextView) findViewById(R.id.top_text);
    topSkore.setText(String.valueOf(sharedPreferences.getInt(TOP_SKORE, 0)));

    final TextView aktualSkore = (TextView) findViewById(R.id.skore_text);
    aktualSkore.setText(String.valueOf("0"));

    final Button playButton = (Button) findViewById(R.id.hra_button);
    playButton.setOnClickListener(new View.OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        Random r = new Random();
        int skore = r.nextInt(1000);
        aktualSkore.setText(String.valueOf(skore));
        if (skore > sharedPreferences.getInt(TOP_SKORE, 0))
        {
          SharedPreferences.Editor editor = sharedPreferences.edit();
          editor.putInt(TOP_SKORE, skore);
          editor.commit();
          topSkore.setText(String.valueOf(skore));
        }
      }
    });

    final Button resetButton = (Button) findViewById(R.id.nuluj_button);
    resetButton.setOnClickListener(new View.OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putInt(TOP_SKORE, 0);
        editor.commit();
        topSkore.setText(String.valueOf("0"));
```

```
            aktualSkore.setText(String.valueOf("0"));
        }
      });
    }
}
```



Obrázek 10.1: Test aplikace. Zkuste po nastavení skóre aplikaci restartovat. Měla by se zobrazit naposledy uložená hodnota top skóre

Příklad – PreferenceFragment

Ukládání údajů prostřednictvím třídy `SharedPreferences` se používá velmi často k ukládání konfiguračních údajů. Aby nebylo nutné řešit v každé aplikaci znovu a znovu uživatelské rozhraní na zobrazování a nastavování takto uložených hodnot, nabízí Android pro tento účel třídu s názvem `PreferenceFragment`. Podobně jako jiné fragmenty dokážeme tento fragment zakomponovat do uživatelského rozhraní aplikace a zobrazit ho buď samostatně na telefonech s menším displejem, nebo spolu s jiným fragmentem na tabletech s větším displejem, případně na telefonech s vyšším rozlišením a orientací na šířku. Fungování třídy nejlépe ukážeme na praktickém příkladu.

Abychom pokračovali v linii předchozího příkladu, který je užitečný k uložení nejvyššího dosaženého skóre ve hře, je v tomto případě námětem uložení jména nebo přezdívky hráče, takzvaného „nickname“.

POZNÁMKA

Pozornější čtenář vnímá rozdíl mezi těmito náměty. Ukládání skóre nevyžadovalo uživatelské rozhraní. Skóre se ukládalo po dohrání hry (simulované tlačítkem) automaticky. Naproti tomu nickname si uživatel nastavuje, případně mění, sám a potřebuje pro tento účel uživatelské rozhraní.

Uživatelské rozhraní hlavní aktivity obsahuje jediné tlačítko k nastavení nickname.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <Button
    android:id="@+id/nick_button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="NICKNAME" >
  </Button>

</LinearLayout>
```

Layout fragmentu v souboru *pref_fragment.xml*:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  class="sk.pcrevue.llacko.ukladaniefragment.PrefFragment"
  android:orientation="vertical"
  android:id="@+id/mujPreferenceFragment">
</fragment>
```

Aplikace bude potřebovat ještě návrh layoutu `PreferenceScreen`, který de facto definuje vzhled dialogového okna na editování nickname v souboru *nick_prefs.xml*. Soubor umístěte do podsložky *xml* vytvořené ve složce *res*.

```
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android"
  android:key="@+id/pref_screen" >

  <EditTextPreference
    android:dialogMessage="Nick Name"
    android:dialogTitle="Edituj Nick Name"
    android:key="nickname"
    android:negativeButtonText="Zrus"
    android:positiveButtonText="OK"
    android:title="Nick Name">
  </EditTextPreference>
</PreferenceScreen>
```

Vytvořte třídu `PrefFragment`:

```
public class PrefFragment extends PreferenceFragment {
  @Override
  public void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.nick_prefs);
  }
}
```

V kódu hlavní aktivity vytvoříme obsluhu události od tlačítka. Po klepnutí na tlačítko vytvoříme záměr k spuštění aktivity obsahující `PreferenceFragment`:

```
public class MainActivity extends Activity
{
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void onClick(View v) {
    startActivity(new Intent(MainActivity.this, PrefActivity.class));
  }
}
```

Vytvořte novou aktivitu `PrefActivity`. V kódu aktivity *PrefFragment.java* vytvoříme třídu `MujPreferenceFragment` rozšiřující bázovou třídu `PreferenceFragment`. Úlohou fragmentu je zobrazit aktuální hodnotu jména hráče a umožnit její změnu.

```
public class PrefActivity extends Activity {
  private static final String NICKNAME = "nickname";

  @Override
  protected void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.pref_fragment);
  }

  public static class MujPreferenceFragment extends PreferenceFragment {
    protected static final String TAG = "MojPrefsFragment";

    private OnSharedPreferenceChangeListener listener;
    private Preference preference;

    @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.nick_prefs);
      preference = (Preference) getPreferenceManager()
        .findPreference(NICKNAME);

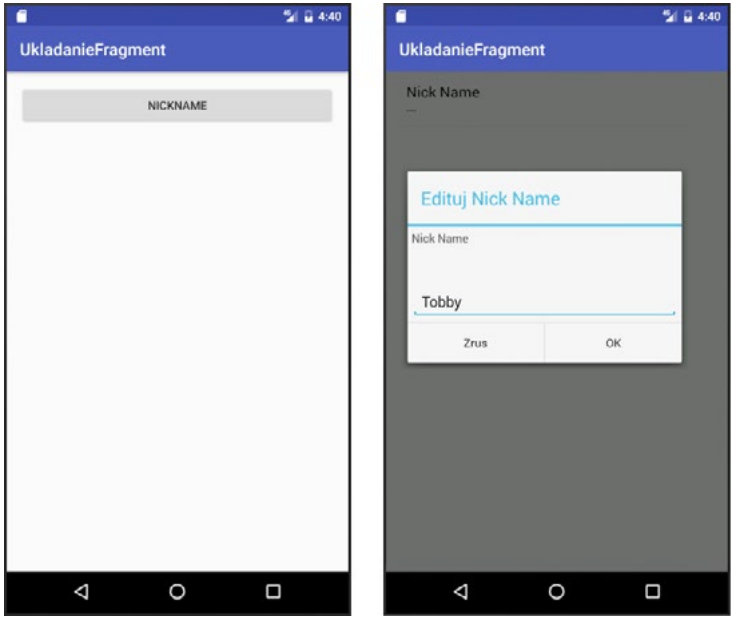
      listener = new OnSharedPreferenceChangeListener() {
        @Override
        public void onSharedPreferenceChanged
          (SharedPreferences sp, String key) {
          preference.setSummary(sp.getString(NICKNAME, "---"));
        }
      };

      SharedPreferences prefs =
        getPreferenceManager().getSharedPreferences();
```

```
            prefs.registerOnSharedPreferenceChangeListener(listener);
            listener.onSharedPreferenceChanged(prefs, NICKNAME);
        }
    }
}
```

Nezapomeňte aktivity s `PreferenceFragment` deklarovat v *AndroidManifest.xml*:

```
<activity android:name=".PrefActivity" >
        </activity>
```



Obrázek 10.2: Test aplikace

Ukládání údajů do souboru

Operační systém Android využívá linuxové jádro a z toho vyplývá i struktura jeho souborového systému a hierarchie složek. Základem hierarchie je jednoduchý integrovaný strom složek (souborů) začínající složkou „/“ (kořen – root stromu). Do kořenové složky jsou zahrnuty všechny ostatní složky (soubory), včetně složek reprezentujících ostatní média (interní či externí).

Cestou (path) se rozumí část integrované stromové struktury vedoucí ke konkrétní složce (souboru), resp. datovému médiu. Absolutní cestou je cesta začínající složkou „/“. Android používá koncept připojování (mount) a odpojování (umount) složek (médií) známý z prostředí Linuxu. Stromovou strukturu dělí na interní (internal flash drive) a externí (external drive) část (prostor). Stále přitom hovoříme o jediném integrovaném stromu, jehož jednotlivé složky tvoří tzv. přípojné body (mount points), ať už pro souborové systémy, nebo pro ostatní média (např. SD karty). Všechny souborové systémy jsou tedy vždy součástí společné stromové hierarchie.

K dispozici jsou dva typy úložných prostorů, kam mohu aplikace ukládat údaje:

- **Interní úložiště** – Fyzicky je ve složce */data/data/<jméno aplikace>/files/*. Aplikace by ho měly používat na malé a střední objemy dat, pro interní potřeby aplikací, privátní údaje, dočasné údaje a podobně.
- **Externí úložiště** – Jedná se o vyměnitelné úložiště, nejčastěji o paměťovou kartu. Fyzicky je externí úložiště mapované jako složka */sdcard/Android/data/<jméno aplikace>/files/*. Využívá se na velké objemy údajů, které jsou všeobecného charakteru. Typickým příkladem jsou obrázky, hudba, videa a dokumenty. Musí se počítat s tím, že paměťová karta může být přemístěna do jiného zařízení, případně i do zařízení jiné platformy, například do počítače s operačním systémem Windows či macOS.

Při práci se soubory budeme v principu využívat třídu `FileInputStream (BufferedInputStream)`, pomocí které načítáme údaje (bajty) ze souborů, a třídu `FileOutputStream (BufferedOutputStream)`, pomocí které ukládáme údaje (bajty) do souborů. Používáme přitom konkrétně metody `read()` a `write()`.

Příklad – Ukládání do souboru v interní paměti

Aplikace po spuštění zkontroluje, zda existuje textový soubor s daným jménem. Pokud ne, tento soubor vytvoří a zapíše do něj textovou informaci.

Na práci se soubory v interním úložišti jsou k dispozici metody:

- `FileOutputStream.openFileOutput(String name, int mode)` – otevření privátního souboru pro zápis. Pokud soubor s daným jménem neexistuje, bude vytvořen.
- `FileInputStream.openFileInput(String name)` – otevření privátního souboru pro čtení.

Layout hlavní aktivity využívá `ScrollView`, aby bylo možné zobrazit vertikálně rolovatelný text v komponentech uspořádaných v kontejneru `LinearLayout` vertikálně.

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/scrollView1"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <LinearLayout
    android:id="@+id/linearLayout1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >
  </LinearLayout>

</ScrollView>
```

Kód hlavní aktivity obsahuje metody pro zápis údajů do textového souboru a jejich čtení.

```
public class MainActivity extends AppCompatActivity {

  private String TAG = "SouborInterny";
  private final static String soubor = "Soubor1.txt";

  @Override
```

```
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    LinearLayout layout = (LinearLayout) findViewById(R.id.linearLayout1);

    if (!getFileStreamPath(soubor).exists())  {
      try {souborZapis();}
      catch (FileNotFoundException e)  {
        Log.i(TAG, "FileNotFoundException");}
    }

    try {souborCteni(layout);}
    catch (IOException e) {Log.i(TAG, "IOException");}
  }

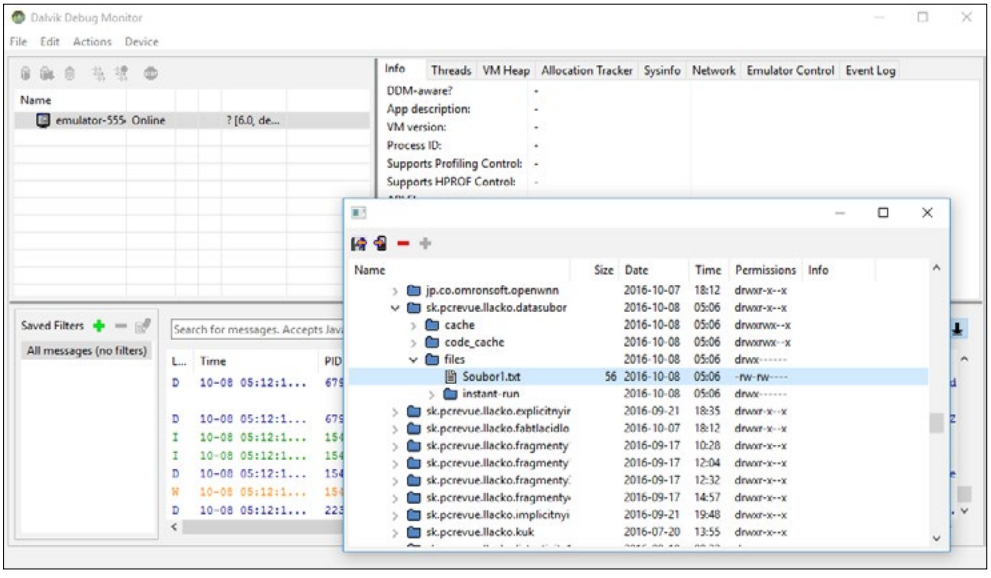
  private void souborZapis() throws FileNotFoundException   {
    FileOutputStream fos = openFileOutput(soubor, MODE_PRIVATE);
    PrintWriter pw = new PrintWriter(new BufferedWriter(
      new OutputStreamWriter(fos)));
    pw.println("Lorem ipsum dolor sit amet");
    pw.println("consectetur adipisicing elit");
    pw.close();
  }

  private void souborCteni(LinearLayout lay) throws IOException  {
    FileInputStream fis = openFileInput(soubor);
    BufferedReader reader = new BufferedReader(new InputStreamReader(fis));
    String line = "";

    while ((line = reader.readLine())!=null)  {
      TextView tv = new TextView(this);
      tv.setText(line);
      lay.addView(tv);
    }
    reader.close();
  }
}
```

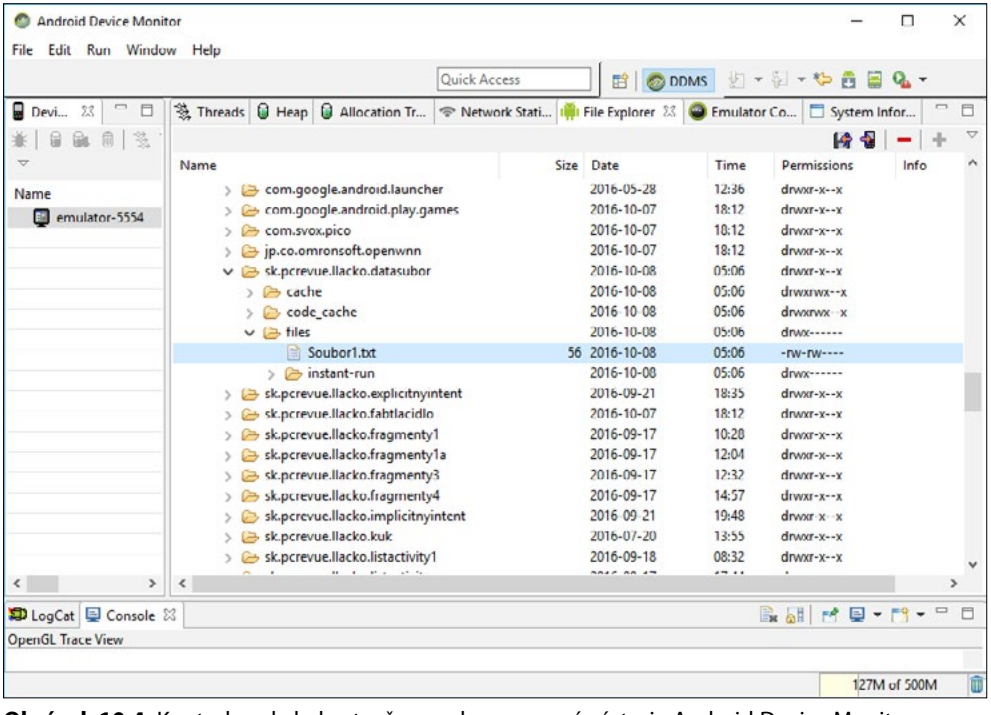
Pomocí průzkumníku souborů na reálném zařízení můžete zkontrolovat, že soubor byl skutečně vytvořen. Na emulátoru můžete využít starší, ale i Android Studiem podporovanou metodu. Dalvik Debug Monitor spustíte přes dávkový soubor *ddms.bat*, který najdete v adresáři, kam jste umístili Android Development Tools ve složce *sdk/tools*. V nabídce **Device** zvolte položku **File Explorer**.

V nástroji **File Explorer** najděte soubor ve složce */data/data/<jméno aplikace>/files/*, v našem případě */data/data/SouborInterni/files/Soubor1.txt*. Pokud si chcete prohlédnout jeho obsah, pomocí tlačítka **Pull File from Device** (první tlačítko zleva na panelu nástrojů) ho zkopírujte do vybrané složky hostitelského vývojářského počítače.



Obrázek 10.3: Kontrola, zda byl vytvořen soubor, pomocí nástroje Dalvik Debug Monitor

Lepší a doporučenou metodou je **Android Device Monitor**. Spustíte ho z Android Studia pomocí nabídky **Tools** → **Android** → **Android Device Monitor**.



Obrázek 10.4: Kontrola, zda byl vytvořen soubor, pomocí nástroje Android Device Monitor

Příklad – Ukládání do souboru v externí paměti

Android umožňuje aplikacím i ukládání souborů do externí paměti, která je nejčastěji fyzicky realizovaná jako paměťová karta. Námětem aplikace je zápis obrázku do externí paměti, tj. fyzicky na paměťovou kartu, jeho následné načtení a zobrazení.

POZNÁMKA

Při práci s externí pamětí musíte mít neustále na zřeteli, že tato paměť může být uživatelem kdykoliv odebrána.

V takovém případě soubory a média z pohledu aplikace de facto zmizí bez varování. Proto pokud aplikace potřebuje zapisovat do externí paměti, musí si nejprve zjistit její stav pomocí metody `String Environment.getExternalStorageState()`.

Metoda vrací hodnoty vyjadřující stav externí paměti:

- `MEDIA_MOUNTED` – médium je vloženo a připraveno k zápisu a čtení
- `MEDIA_MOUNTED_READ_ONLY` – médium je vloženo a připraveno jen k zápisu
- `MEDIA_REMOVED` – médium není vloženo

Další možné stavy jsou `MEDIA_BAD_REMOVAL`, `MEDIA_CHECKING`, `MEDIA_NOFS` (neznámý souborový systém), `MEDIA_UNMOUNTABLE`.

UPOZORNĚNÍ

Pokud chce aplikace zapisovat do externí paměti, musí mít odpovídající oprávnění. Do manifestu aplikace v souboru *AndroidManifest.xml* budete muset přidat oprávnění `WRITE_EXTERNAL_STORAGE`:`<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>`

V projektu aplikace ve složce *res* vytvořte vnořenou složku *raw* a do ní překopírujte vhodný obrázek.

Layout hlavní aktivity obsahuje prvek `ImageView`.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.datasuborkarta.MainActivity">

  <ImageView
    android:id="@+id/image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:contentDescription="Popis obrázka"
    android:layout_centerInParent="true"/>

</RelativeLayout>
```

Aplikace načítá soubor s obrázkem z adresáře *res/raw* a zkopíruje ho do externího úložiště. Následně obrázek načítá z tohoto externího souboru a zobrazí ho na obrazovce. V metodě `onCreate` kód kontroluje externí úložiště, zda je přítomno a přístupné. Kopírování začíná vytvořením nového výstupního streamu, který bude použit k zápisu do souboru v externí paměti. Následně se vytváří vstupní stream pro načítání obrazových dat z adresáře *res/raw*. Potom proběhne kopírování údajů ze vstupního do výstupního streamu. Na načítání obrázku z externí paměti a jeho přidání do `ImageView` se využívá metoda `setImageURI()`.

Kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  private String TAG = "SuborExterny";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

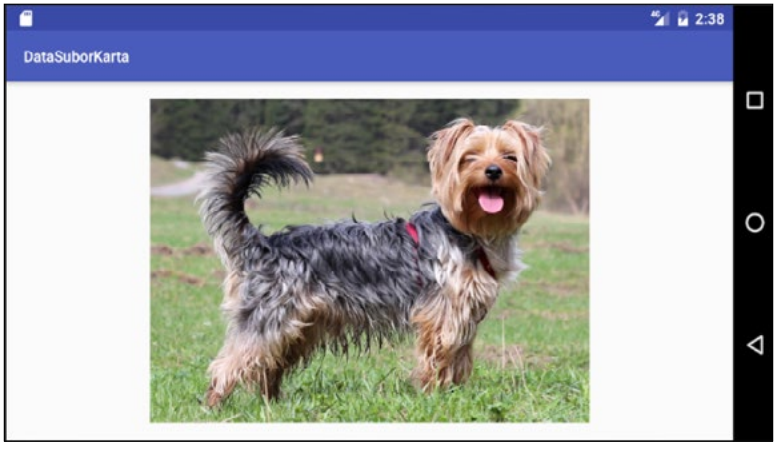
    if (Environment.MEDIA_MOUNTED.equals(Environment
      .getExternalStorageState()))   {
      File soubor = new
        File(getExternalFilesDir(Environment.DIRECTORY_PICTURES),
        "tobby.jpg");

      if (!soubor.exists()) kopirujNaKartu(soubor);
      ImageView iw = (ImageView) findViewById(R.id.image);
      iw.setImageURI(Uri.parse("file://" + soubor.getAbsolutePath()));
    }
  }

  private void kopirujNaKartu(File soubor) {
    try   {
      BufferedOutputStream bos = new BufferedOutputStream(
        new FileOutputStream(soubor));
      BufferedInputStream bis = new BufferedInputStream(getResources()
        .openRawResource(R.raw.tobby));
      kopiruj(bis, bos);
    }
    catch (FileNotFoundException e)  {
      Log.e(TAG, "FileNotFoundException");}
  }

  private void kopiruj(InputStream is, OutputStream os)  {
    final byte[] buf = new byte[1024];
    int nPocet;
    try   {
      while (-1 != (nPocet = is.read(buf)))  {
        os.write(buf, 0, nPocet);
      }
    } catch (IOException e) {e.printStackTrace();}
    finally  {
```

```
        try  {
          is.close();
          os.close();
        }
        catch (IOException e) {Log.e(TAG, "IOException");}
      }
    }
  }
```



Obrázek 10.5: Test aplikace

Pomocí vhodného souborového manažeru můžete zkontrolovat, zda se soubor skutečně zkopíroval do externího úložiště.

Od verze Android 4.4 je podporovaných více externích úložišť k ukládání dat na jednom zařízení. Tablet může mít například 16GB zabudovanou paměť a slot na paměťovou kartu. Na práci s více úložnými zařízeními jsou k dispozici metody Context.getExternalFilesDirs(), Context.getExternalCacheDirs() a Context.getObbDirs().

Externí zařízení přístupná pomocí těchto rozhraní API musí být částečně součástí zařízení. Typicky se to řeší pamětí uvnitř nebo slotem pro kartu SD v prostoru pro baterie, takže ji uživatel nemůže odebrat za běhu zařízení. Proto mohou vývojáři očekávat, že data uložená v těchto úložištích tam budou dostupná po dlouhou dobu.

Povolení WRITE_EXTERNAL_STORAGE udělí pouze přístup pro zápis do primárního úložiště na zařízení. Aplikace nesmí psát na sekundární externí zařízení pro ukládání dat, s výjimkou specifických adresářů. To umožňuje, že systém může vyčistit soubory po odinstalování aplikací.

Ukládání dočasných souborů

Pokud aplikace potřebuje uložit soubory, jejichž smazáním nepřijde o žádná důležitá data, používá se tzv. cache. Dočasné soubory budou umístěny ve složce */data/data/<jmeno aplikace>/cache/* nebo ve složce */sdcard/Android/data/<jmeno aplikace>/cache/*. Pokud bude mít systém v úložišti málo místa, automaticky tyto soubory smaže.

Příklad kódu:

```
Filedir= getCacheDir(); // popr. getExternalCacheDir();
File soubor = newFile(dir, "souborCache");
os = newFileOutputStream(soubor);
is= newFileInputStream(soubor);
```

Databáze SQLite

Embedded databáze SQLite (*SQLite.org*) má široké použití na různých platformách.

POZNÁMKA

Do zařízení s Androidem není nutné instalovat databázový engine ani ho přibalovat k aplikacím, SQLite je už součástí operačního systému.

SQLite je malá, ale výkonná relační databáze, která pracuje bez serveru, nemusí se instalovat a nemá žádné konfigurační soubory. Na rozdíl od SQL Serveru, Oracle, MySQL a jiných řešení klient-server používá SQLite pouze klientskou část. Data jsou uložena v souboru, ke kterému přistupuje přímo klientská aplikace. Zjednodušeně řečeno: samotná aplikace zastává funkci klienta i serveru současně. Jinak řečeno: server je součástí klienta. I když SQLite vyřizuje vždy pouze jeden požadavek, samotný výkon databáze je relativně velmi vysoký. Čtení dat má přitom prioritu před zápisem.

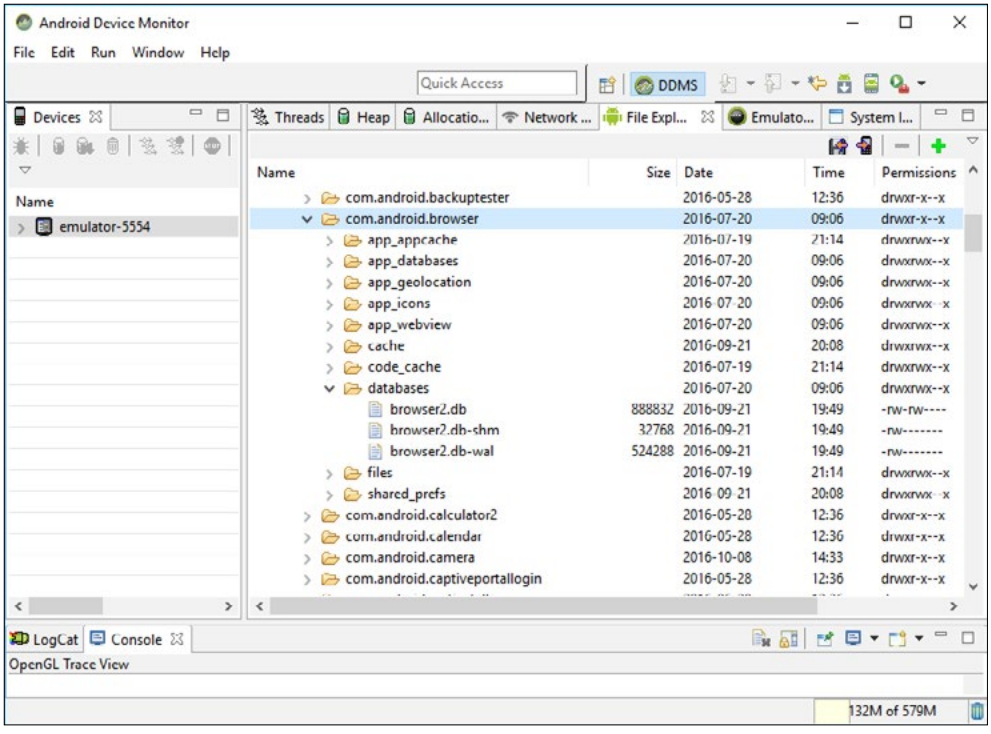
POZNÁMKA

Databázi SQLite je možné směle přiřadit přívlastek „in memory", jelikož u poměrně malých objemů dat, s kterými zpravidla operuje aplikace pro Android, jsou tyto uloženy v paměti. Jelikož mají moderní tablety a ultrabooky polovodičové disky SSD, i operace ukládání údajů do databázového souboru jsou velmi rychlé.

Databázový engine čte a zapisuje data do souborů. Z funkcionalit klasických databázových serverů je tu podpora více tabulek, indexování, podpora triggerů, můžete vytvářet pohledy. SQLite podporuje transakce ACID (Atomicity, Consistency, Isolation, Durability).

Abyste mohli vyvíjet aplikaci s databází SQLite, potřebujete znát aspoň základy práce s databázemi, databázovými tabulkami, a především jazyk SQL.

Jak jsme už zmiňovali, SQLite je souborová databáze. Databázový soubor pro příslušnou aplikaci, která využívá SQLite engine, je ve složce */data/data/<package name>/databases/<database filename>.db*.



Obrázek 10.6: Databázové soubory pro aplikaci browser, která je součástí Androidu

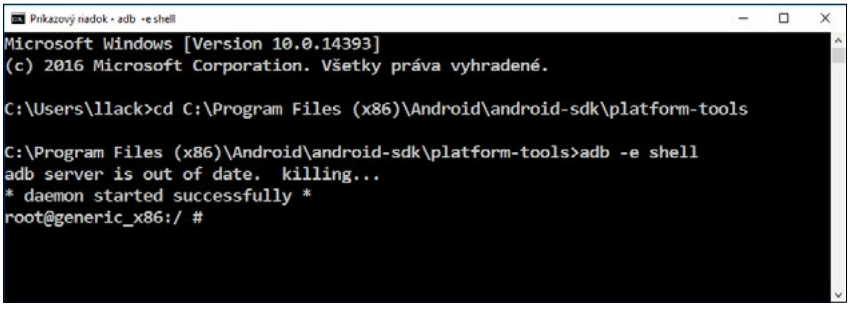
Pokusy se zabudovanou databází SQLite v emulátoru

Dříve než začnete ladit zbytek aplikace, doporučujeme odladit databázovou vrstvu, tedy příkazy na vytvoření databáze, databázových tabulek, jejich naplnění, výběr údajů a podobně. Můžete využít dvě metody: buď si nainstalovat SQLite i do vývojářského počítače (jelikož se jedná o multiplatformní databázi, je k dispozici i pro Windows a macOS), nebo využít toho, že databáze SQLite je součástí každého Androidu, tedy i toho v emulátoru.

K tomuto účelu vám poslouží nástroj ADB. U standardní instalace ho najdete ve složce *C:\Program Files (x86)\Android\android-sdk\platform-tools*

Ve Windows spusťte příkazový řádek a přejděte do složky s aplikací ADB. Následně zadejte příkaz:

```
adb -e shell
```



Obrázek 10.7: Spuštění ADB pomocí příkazového řádku Windows

Databázový soubor pro příslušnou aplikaci, která využívá SQLite engine, je ve složce */data/data/<package name>/databases/<database filename>.db*. Přejděte do složky */data/data* příkazem:

```
cd /data/data
```

Zatím nemáme vlastní aplikaci, která by využívala databázi, do adresářů jiných či systémových aplikací zasahovat nebudeme, a tak si vytvoříme „falešný" adresář aplikace *pokus*. Bude mít název *databases*.

```
mkdir com.example.pokus
cd com.example.pokus
mkdir databases
cd databases
```

Odezva příkazového řádku bude:

```
C:\Program Files (x86)\Android\android-sdk\platform-tools>adb -e shell
root@generic_x86:/ # cd /data/data
root@generic_x86:/data/data # mkdir com.example.pokus
root@generic_x86:/data/data # cd com.example.pokus
root@generic_x86:/data/data/com.example.pokus # mkdir databases
root@generic_x86:/data/data/com.example.pokus # cd databases
root@generic_x86:/data/data/com.example.pokus/databases #
```

Abychom v adresáři vytvořili databázi, spustíme interaktivní nástroj SQLite:

```
sqlite3 ./mydatabase.db
```

Odezva konzolové aplikace bude:

```
sqlite3 ./mydatabase.db                                                <
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite>
```

Všimněte si, že příkazem `.help` můžete vypsat seznam všech příkazů databázové konzoly včetně jejich syntaxe.

Nyní můžete oprášit svoje znalosti databázového jazyka SQL a vytvořit databázi, v ní databázovou tabulku a tuto naplnit údaji. Všimněte si, že každý záznam bude mít atribut `_id` jako unikátní identifikátor, který se automaticky inkrementuje.

```
create table knihy
(
  _id integer primary key autoincrement,
  autor text,
  nazov text,
  zaner text
);
```

Po potvrzení příkazu se nestalo nic. To je ten lepší případ, kdy se příkaz provedl. V horším případě by konzole vypsala příslušné chybové hlášení. O tom, zda se tabulka vytvořila, dostanete odezvu příkazem, který vypíše seznam tabulek v databázi.

```
sqlite> .tables
knihy
```

Pokračujte vložením několika záznamů.

```
insert into knihy (autor, nazov, zaner)
values ('Jo Nesbo', 'Snehuliak', 'detektivka');
insert into knihy (autor, nazov, zaner)
values ('Lars Kepler', 'Písečný muž', 'detektivní thriller');
```

Obsah databázové tabulky vypíšete příkazem:

```
select * from knihy;
```

Konzolová aplikace vrátí výpis.

```
sqlite> select * from knihy;
1|Jo Nesbo|Snehuliak|detektivka
2|Lars Kepler|Písečný muž|detektivní thriller
```

Interakce aplikace s databází

Databázová aplikace využívá architekturu klient-server, přičemž SQLite plní funkci serveru a vaše aplikace k němu přistupuje přes vhodné rozhraní. Zjednodušeně řečeno, aplikace posílá databázovému serveru příkazy SQL a databázový server je vykoná. Pokud se jedná o dotaz, typicky příkaz SQL SELECT, databáze vrátí množinu výsledků, které dotazu vyhovují.

Při ladění databázových aplikací na klasických platformách – Windows, Windows Server, Linux, macOS – je dobrým zvykem nejprve příkazy SQL, s kterými bude aplikace pracovat, samostatně otestovat. V praxi to probíhá tak, že se přes klientskou konzolovou aplikaci zadávají dotazy a přes tu samou aplikaci se kontroluje správnost vrácených výsledků. Databázová vrstva se testuje komplexně. Zpravidla se testují tyto operace:

- Vytvoření databáze
- Vytvoření objektů v databázi, tedy tabulek, pohledů, uložených procedur, triggerů...
- Vkládání údajů do databázových tabulek
- Výběr údajů pomocí příkazů SELECT
- Změny a mazání údajů

V předchozí části jsme vám ukázali, jak vyzkoušet příkazy na práci s databází pomocí konzolové aplikace vývojářského počítače, který se připojí k databázovému enginu emulátoru zaří-

zení s Androidem. V další části ukážeme jiný postup, ve kterém nainstalujeme verzi Windows databáze SQLite na vývojářský počítač.

Java třídy na přístup k databázi SQLite a jejím údajům

Třída `SQLiteDatabase` tvoří rozhraní mezi aplikačním kódem a databází. Nejčastěji budete používat tyto její metody:

- `insert()` – vloží nový záznam do databázové tabulky
- `delete()` – vymaže záznamy z tabulky
- `query()` – spustí dotaz do databáze a zpřístupní výsledky pomocí objektu `Cursor`
- `execSQL()` – spustí SQL příkaz, který nevrací údaje; typickým příkladem je příkaz na vytvoření tabulky, indexu a podobně
- `rawQuery()` – spustí SQL příkaz a zpřístupní výsledky pomocí objektu `Cursor`

Třída `Cursor` umožňuje aplikačnímu kódu přístup k výsledkům databázového dotazu. Například operace SQL `SELECT` provedená nad databází může vrátit desítky, tisíce, někdy i miliony záznamů. Pomocí metod třídy `Cursor` je může aplikace zobrazit. Zjednodušeně si můžeme kurzor představit jako ukazatel na aktuálně zobrazovaný záznam.

- `close()` – zavře objekt `cursor` a uvolní jeho zdroje
- `getCount()` – vrátí počet záznamů, které jsou výsledkem databázového dotazu
- `get<type>()` – vrátí hodnotu specifikovaného typu daného indexem sloupce záznamu na pozici kurzoru (`getString()`, `getInt()`, `getShort()`, `getFloat()` a `getDouble()`)
- `moveToFirst()` – přesun na první záznam
- `moveToLast()` – přesun na poslední zázna
- `moveToNext()` – přesun na další záznam
- `move()` – posun o daný počet záznamů od aktuálního záznamu

Ke komunikaci aplikace pro Android s databází SQLite ohledně řízení životního cyklu databáze slouží třída `SQLiteOpenHelper`. Pod řízením životního cyklu databáze rozumíme vytváření a případnou aktualizaci databázových struktur. K aktualizaci databáze dochází typicky tehdy, když aktualizovaná, vylepšená verze aplikace využívá další, případně jiné atributy než verze předchozí. `SQLiteOpenHelper` obsahuje metody:

- `onCreate(SQLiteDatabase)` – metoda se volá v případě, že databázové objekty ještě neexistují a je potřeba je vytvořit.
- `onUpgrade(SQLiteDatabase, int, int)` – metoda se volá při aktualizaci aplikace, která mění i návrhovou strukturu databázových objektů. Ve cvičném příkladu nebudeme s aktualizací počítat a v metodě vytvoříme strukturu databáze nanovo.

Při implementaci datového modelu budete v aplikaci de facto využívat jen konstruktor a metodu `onCreate(SQLiteDatabase)`.

Příklad – Čtenářský deník

Vytvořte jednoduchou aplikaci čtenářský deník, do které si budete zaznamenávat knihy, které jste přečetli. Do deníku se bude zaznamenávat datum, jméno autora, název knihy a její hodnocení.

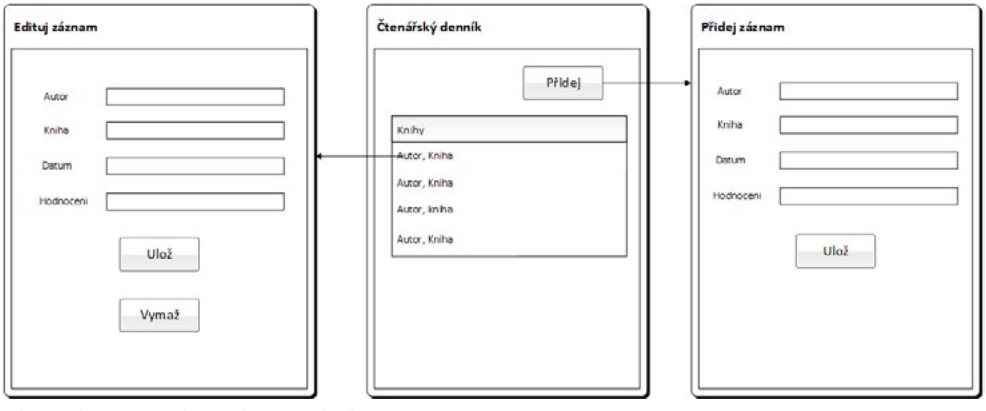
Rozbor řešení

Údaje se budou ukládat do databáze SQLite, která je součástí platformy Android a je k ní k dispozici zabudované rozhraní API. Čtenářský deník v podobě, v jaké ho chceme v tomto jednoduchém příkladu vytvořit, potřebuje jen čtyři atributy: datum, jméno autora, název knihy a její hodnocení.

Jelikož hodnota každého atributu může být duplicitní, pro správné fungování aplikace je zapotřebí i pátý atribut, unikátní identifikátor každé položky. Starosti s unikátním identifikátorem naštěstí mít nebudete, protože databáze SQLite umožňuje jeho automatické generování. Aby bylo na aplikaci co vylepšovat, včetně procvičení změny návrhové struktury databázové tabulky formou aktualizace aplikace, přestože SQLite disponuje datovým typem na uložení data, budeme v první verzi datum ukládat jako textový řetězec.

Z hlediska uživatelského rozhraní vytvoříme postupně tři aktivity.

- **Hlavní aktivita** – (na obrázku uprostřed) bude obsahovat seznam knih a tlačítko na přidání nového záznamu
- **Přidej záznam** (vpravo) – aktivita na přidání nového záznamu
- **Edituj záznam** (vlevo) – aktivita na editování, případně vymazání záznamu



Obrázek 10.8: Nákres obrazovek aktivit

Klepnutím na položku na obrazovce hlavní aktivity se zobrazí aktivita na editování záznamu. V polích `EditView` budou při inicializaci aktivity vyplněné hodnoty atributů editovaného záznamu.

Naše aplikace bude využívat databázi SQLite, která je součástí Androidu, nebudeme ale zadávat příkazy SQL na zařízeních s Androidem. Hlavně na telefonech by to nebylo pohodlné a ani se nebudeme přes konzoli připojovat ke spuštěnému emulátoru. Ani tento postup není pohodlný. Využijeme skutečnosti, že SQLite je multiplatformní databáze, a nainstalujete si ji na vývojářský počítač s Windows, případně macOS. Instalační soubor získáte na webu *http://sqlite.org*.

Doporučujeme nainstalovat konzolovou aplikaci **Command Line Shell For SQLite** ze sekce **Precompiled Binaries for Windows**, případně **Precompiled Binaries for Mac OS X (x86)** s přibalenou databází v souboru *sqlite-shell-win32-x86-XXXX.zip*. Abychom byli přesnější, aplikace se neinstaluje, stačí ji vybalit z archívu a spustit – a můžete zadávat příkazy SQL.

Z archívu vybalte soubor *sqlite.exe* a spusťte ho. Na rozdíl od aplikace nemusíte vytvářet novou databázi, protože konzolová aplikace má už databázi, se kterou můžete pracovat, vytvořenou.

Nejprve vytvořte databázovou tabulku příkazem `CREATE TABLE`. Atribut `id` bude automaticky se inkrementující unikátní identifikátor záznamu:

```
CREATE TABLE knihy
(
  _id INTEGER PRIMARY KEY AUTOINCREMENT,
  autor TEXT,
  kniha TEXT,
  datum TEXT,
  hodnoceni TEXT
);
```

Pokud konzolová aplikace neoznámí chybu, příkaz na vytvoření nové databázové tabulky byl úspěšně proveden.

Automaticky generovaný unikátní identifikátor, případně i další atributy, které využívají automaticky se inkrementující hodnoty, je potřeba deklarovat jako `INTEGER PRIMARY KEY`. Pokud v příkazu `INSERT` vložíte do příslušného atributu hodnotu `NULL`, automaticky se vytvoří nová hodnota získaná inkrementováním naposledy vložené hodnoty.

TIP

Pokud potřebujete v databázové tabulce datový typ `BOOLEAN`, doporučujeme ho simulovat datovým typem `INTEGER` využívajícím jen hodnoty `0` pro `FALSE` a `1` pro `TRUE`.

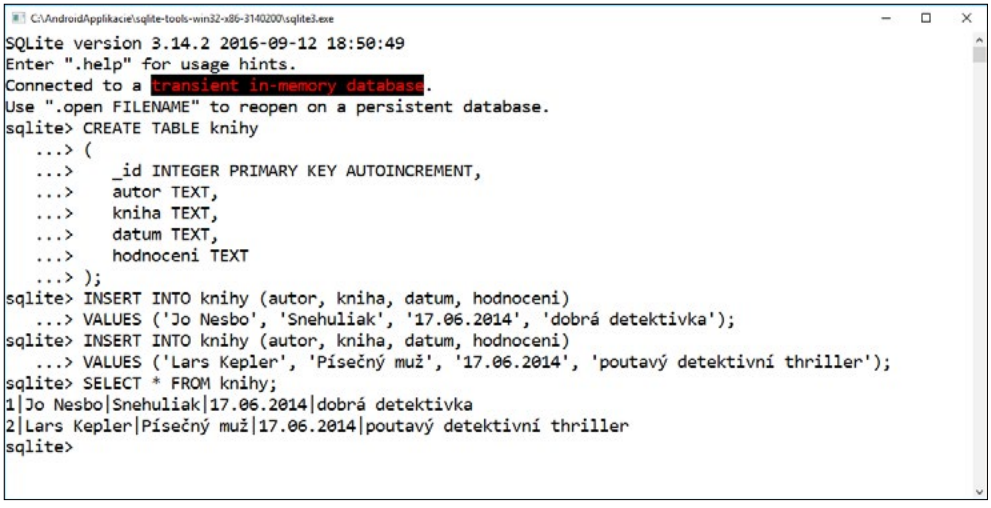
Následně otestujte příkaz `INSERT` na vložení nového záznamu do právě vytvořené tabulky. Vložíme dva záznamy:

```
INSERT INTO knihy (autor, kniha, datum, hodnoceni)
VALUES ('Jo Nesbo', 'Snehuliak', '17.06.2014',
  'dobrá detektivka');

INSERT INTO knihy (autor, kniha, datum, hodnoceni)
VALUES ('Lars Kepler', 'Písečný muž', '17.06.2014',
  'poutavý detektivní thriller');
```

Správnost vytvoření databázové tabulky a vložení záznamů si ověříme výpisem pomocí příkazu `SELECT`.

```
SELECT * FROM knihy;
```



Obrázek 10.9: Testování příkazů SQL v konzolové aplikaci Sqlite3.exe

Ve výpisu si všimněte, že pro vámi vkládané záznamy se vygenerovaly hodnoty `id`, tedy automaticky se inkrementujícího unikátního identifikátoru, `1` a `2`.

Můžete vyzkoušet složitější dotaz:

```
SELECT _id, kniha, autor, hodnoceni
FROM knihy
ORDER BY kniha;
```

V testování pokračujeme příkazy na změnu a vymazání údajů:

```
UPDATE knihy
SET datum = '18.06.2014'
WHERE _id = 2;

DELETE FROM knihy
WHERE _id = 1;
```

Nakonec vymažeme i databázovou tabulku. I tuto funkcionalitu budete ve své aplikaci potřebovat, pokud budete v rámci aktualizace měnit návrhovou strukturu databázové tabulky.

```
DROP TABLE IF EXISTS knihy;
```

POZNÁMKA

Pokud vámi zformulované a odladěné dotazy správně fungují v konzolové aplikaci, není důvod, proč by neměly fungovat ve vaší aplikaci pro Android.

Pokud budou v aplikaci problémy, budete vědět, že je způsobuje něco jiného, a ne syntaktické chyby v příkazech SQL.

Datový model

V aplikaci potřebujete vytvořit vrstvu, která bude řešit ukládání a čtení údajů. Nazveme ji `DataModel`. Jejím úkolem bude zjistit, zda je už pro aplikaci vytvořena databáze a tabulka. Pokud ne, tak tyto objekty při prvním spuštění aplikace vytvoří. Třída `DataModel` bude také číst a ukládat údaje.

Ve složce zdrojových kódů Javy vytvořte novou třídu `DataModel`. Třída bude odvozená od třídy `SQLiteOpenHelper`.

Vytvořte vlastní podtřídu třídy `SQLiteOpenHelper`, například s názvem `DataModel`, ve které překryjete konstruktor a metody `onCreate()` a `onUpdate()`. V konstruktoru zavoláte rodičovský konstruktor se známými parametry. V metodě `onCreate()` je zformulovaný řetězec příkazu SQL na vytvoření nové databázové tabulky.

Nejprve definujte textové konstanty. Název databáze, databázové tabulky, názvy jejích atributů, případně i klauzuli `ORDER BY`, pokud chcete funkcionalitu aplikace rozšířit o řazení.

Atribut `_id` je unikátní identifikátor záznamu, primární klíč tabulky.

TIP

Doporučujeme použít pro primární klíč zaběhlý identifikátor _id, jelikož tento název předpokládají kurzorové adaptéry.

V našem případě se bude jednat o automaticky generovaný atribut deklarovaný klauzulí `INTEGER PRIMARY KEY`. V praxi to funguje tak, že když je v příkazu `INSERT` hodnota primárního klíče nastavena na `NULL`, databáze inkrementováním naposledy vložené hodnoty vytvoří nový identifikátor.

```
protected static final String DB_DATABAZE = "dbKknihy";
protected static final int DB_VERZE = 1;
protected static final String DB_TABULKA = "tblKnihy";

//atributy sloupce tabulky
public static final String ATR_ID = "_id";
public static final String ATR_AUTOR = "autor";
public static final String ATR_KNIHA = "kniha";
public static final String ATR_DATUM = "datum";
public static final String ATR_HODNOCENI = "hodnoceni";
```

Konstruktor třídy `DataModel` má jako parametr objekt `Context`. Připomínáme, že třída `DataModel` je odvozená od třídy `SQLiteOpenHelper`. Kontextem bývá zpravidla aktivita, která helper využívá. V konstruktoru musíte zavolat rodičovský konstruktor se známými parametry:

```
public DataModel(Context ctx) {
  super(ctx, "DB_DATABAZE", null, DB_VERZE);
}
```

Význam parametrů konstruktoru rodičovské třídy `SQLiteOpenHelper:public SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version)`:

- `context` – zpravidla aktivita, která bude helper využívat.
- `name` – název souboru. Pokud má tento parametr hodnotu `NULL`, vytvoří se databáze in-memory.

- `factory` – parametr `NULL` indikuje standardně používanou továrnu na kurzory (CursorFactory).
- `version` – verze vaší databáze, přesněji její návrhové struktury. Hodnota začíná na `1`, a pokud je verze databáze v zařízení starší než tato hodnota, volá se metoda `onUpdate()`.

Konstruktor vámi vytvořené třídy bude mít jediný parametr typu `Context`. Jako hodnota parametru se odevzdá aktivita, která bude tento helper využívat.

Těla metod `onCreate()` a `onUpdate()` nejjednodušeji přidáte, pokud umístíte kurzor na název třídy `DataModel`. Ve žlutém obdélníku se zobrazí místní nabídka s volbou **Add unimplemented methods**. Po klepnutí na nabídku budou doplněny kostry kódu těchto metod.

V metodě `onCreate()` je zformulovaný řetězec příkazu SQL na vytvoření nové databázové tabulky. Odladili jsme ho v části Interakce aplikace s databází.

```
CREATE TABLE knihy
(
  _id INTEGER PRIMARY KEY AUTOINCREMENT,
  autor TEXT,
  kniha TEXT,
  datum TEXT,
  hodnoceni TEXT
);
```

Vytvoření databázové tabulky

Na vytvoření databázové tabulky jsme využili metodu `execSQL()`. Jejím parametrem je příkaz SQL.

```
@Override
public void onCreate(SQLiteDatabase db) {
  db.execSQL("CREATE TABLE " + DB_TABULKA
    + " (" + ATR_ID   + " INTEGER PRIMARY KEY,"
    + ATR_AUTOR + " TEXT,"
    + ATR_KNIHA + " TEXT,"
    + ATR_DATUM + " TEXT,"
    + ATR_HODNOCENI + " TEXT"
    + ");");
}
```

POZNÁMKA

Metoda `execSQL()` se dá využít jen pro příkazy SQL, které nevracejí údaje.

Pokud byste potřebovali do databázové tabulky vložit explicitně údaje, například hodnoty pro inicializaci a podobně, můžete využít programovou konstrukci:

```
public class MujHelper extends SQLiteOpenHelper {
  public MujHelper (Context context) {
    super(context, "poznamky", null, 1);
  }

  @Override
  public void onCreate(SQLiteDatabase database)  {
```

```
    String sSql = "CREATE TABLE poznamky
      (_id INTEGER PRIMARY KEY, nazev TEXT, poznamka TEXT)";
    database.execSQL(sSql);

    String sInsert = "INSERT INTO poznamky VALUES(null, ?, ?)";
    database.execSQL(sInsert, new Object[] {"Nakup", "text poznamky..."});
    database.execSQL(sInsert, new Object[] {"Oznam", "text poznamky..."});
    database.execSQL(sInsert, new Object[] {"Info", "text poznamky..."});
  }

  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    ...
  }
}
```

Pomocí metody `execSQL()` dokážete zadat z aplikace do databáze libovolný dotaz. Všimněte si v dotazu `INSERT` otazníků reprezentujících parametry, které jsou při volání metody `execSQL()` nahrazeny hodnotami pole hodnot.

UPOZORNĚNÍ

Výhodou takovéto konvence předávání parametrů je jednak typová bezpečnost, jednak zabránění potenciálnímu narušení – bezpečnostní díry SQL Injection.

V příkladu tuto funkcionalitu nevyužijeme. Po nainstalování aplikace se v okamžiku jejího prvního spuštění vytvoří prázdná databázová tabulka a prázdná zůstane, dokud do ní nevloží uživatel data.

V metodě `onUpdate()` vymažeme databázovou tabulku, samozřejmě pouze v případě, že existuje, a zavoláme metodu `onCreate()` na vytvoření nové aktualizované tabulky.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
  String query = "DROP TABLE IF EXISTS " + DB_TABULKA;
  db.execSQL(query);
  onCreate(db);
}
```

Metoda se volá, pokud je verze databáze v zařízení starší než hodnota v parametrech konstruktoru rodičovské třídy `SQLiteOpenHelper`.

V reálné aplikaci by z takovéhoto řešení při změně návrhové struktury databáze uživatelé nadšení nebyli. Odstranění tabulky samozřejmě znamená vymazání všech údajů, které v ní byly uloženy.

Vkládání záznamů do databáze

K získání objektu `SQLiteDatabase` se v závislosti na typu operace využívají metody:

- `SQLiteOpenHelper.getReadableDatabase()` – objekt `SQLiteDatabase` je otevřen jen ke čtení
- `SQLiteOpenHelper.getWritableDatabase()` – objekt `SQLiteDatabase` je otevřen pro zápis i čtení

Metoda SQLiteOpenHelper.close() zavře všechna propojení na databázi.

Ke vkládání údajů slouží metoda třídy SQLiteDatabase.long insert (String table, String nullColumnHack, ContentValues values).

V proceduře vlozZaznam je implementovaná následovně:

```
public long vlozZaznam(HashMap<String, String> atributy) {
  SQLiteDatabase db = this.getWritableDatabase();
  ContentValues val = new ContentValues();
  val.put(ATR_AUTOR, atributy.get(ATR_AUTOR));
  val.put(ATR_KNIHA, atributy.get(ATR_KNIHA));
  val.put(ATR_DATUM, atributy.get(ATR_DATUM));
  val.put(ATR_HODNOCENI, atributy.get(ATR_HODNOCENI));
  long id = db.insert(DB_TABULKA, null, val);
  db.close();
  return id;
}
```

Jako vstupní parametr je použita struktura HashMap, která je implementací datové struktury Map. Vkládané údaje jsou zapouzdřeny v objektu typu ContentValues.

Metoda v případě úspěchu vrátí vygenerovaný identifikátor vloženého záznamu. Pokud se vložení záznamu nepodařilo, vrátí metoda hodnotu -1. Objekt ContentValues obsahuje hodnoty atributů vkládaného záznamu. Tyto vkládáme pomocí metody put().

Aktualizace záznamů

Tato metoda je podobná předchozí. K vymazání údajů je aplikovaná metoda update:

```
int update(String table, ContentValues values,
  String whereClause, String[] whereArgs)
```

Pomocí parametrů je potřeba specifikovat podmínku na výběr záznamu nebo množiny záznamů, které budete aktualizovat.

```
public int aktualizujZaznam(HashMap<String, String> atributy) {
  SQLiteDatabase db = this.getWritableDatabase();
  ContentValues val = new ContentValues();
  val.put(ATR_AUTOR, atributy.get(ATR_AUTOR));
  val.put(ATR_KNIHA, atributy.get(ATR_KNIHA));
  val.put(ATR_DATUM, atributy.get(ATR_DATUM));
  val.put(ATR_HODNOCENI, atributy.get(ATR_HODNOCENI));
  return db.update(DB_TABULKA, val, ATR_ID + " = ?",
    new String[] {atributy.get(ATR_ID)});
}
```

Metoda aktualizujZaznam() vrátí počet aktualizovaných záznamů.

Vymazání záznamů

Na vymazání záznamů z databázové tabulky je aplikovaná metoda:

```
public int delete (String table, String whereClause, String[] whereArgs)
```

Nejprve ukážeme nejjednodušší verzi s přímo zadaným řetězcem příkazu SQL DELETE.

```
public void vymazZaznam(String id) {
  SQLiteDatabase db = this.getWritableDatabase();
  String deleteQuery = "DELETE FROM " + DB_TABULKA + " WHERE _id='"+ id +"'";
  db.execSQL(deleteQuery);
}
```

Pokud byste potřebovali zjistit počet vymazaných řádků, můžete použít následující variantu:

```
public int vymazZaznam2(long id) {
  SQLiteDatabase db = getWritableDatabase();
  String[] argumenty = { String.valueOf(id) };
  int nVymazane = db.delete(DB_TABULKA, ATR_ID + "= ?", argumenty);
  db.close();
  return nVymazane;
}
```

Metoda vrátí počet vymazaných řádků.

Objekt `SQLiteDatabase` se využívá i na dotazování aplikace na údaje v databázi.

Výběr údajů z databázové tabulky

K výběru údajů z databázové tabulky jsme využili metodu:

```
Cursor query (String table, String[] columns, String selection,
  String[] selectionArgs, String groupBy, String having, String orderBy)
```

V třídě `DataModel`, která je odvozená od třídy `SQLiteOpenHelper`, implementujeme dvě metody.

Metoda `dajZaznamy()` vrátí všechny záznamy ve formě pole `ArrayList`.

```
public ArrayList<HashMap<String, String>> dajZaznamy() {
  ArrayList<HashMap<String, String>> alVysledky;
  alVysledky = new ArrayList<HashMap<String, String>>();
  String sSQL = "SELECT * FROM " + DB_TABULKA;
  SQLiteDatabase db = this.getWritableDatabase();
  Cursor cursor = db.rawQuery(sSQL, null);

  //Presun na prvy zaznam
  if (cursor.moveToFirst())     {
    do {
      HashMap<String, String> hm = new HashMap<String, String>();
      hm.put(ATR_ID, cursor.getString(0));
      hm.put(ATR_AUTOR, cursor.getString(1));
      hm.put(ATR_KNIHA, cursor.getString(2));
      hm.put(ATR_DATUM, cursor.getString(3));
      hm.put(ATR_HODNOCENI, cursor.getString(4));
      alVysledky.add(hm);
    } while (cursor.moveToNext()); // kurzor na dalsi zaznam
  }
  return alVysledky;
}
```

Metodu můžete jednoduše modifikovat, aby byly vybrány pouze záznamy, které vyhovují specifické podmínce. Stačí do příkazu SQL SELECT doplnit klauzuli WHERE.

Metoda `dajZaznam()` vrátí konkrétní záznam podle zadaného identifikátoru `_id`. Záznam je vrácen jako objekt typu `HashMap`.

```
public HashMap<String, String> dajZaznam(String id) {
  HashMap<String, String> hm = new HashMap<String, String>();
  SQLiteDatabase db = this.getReadableDatabase();
  String sSQL = "SELECT * FROM " + DB_TABULKA + " WHERE _id='"+id+"'";
  Cursor cursor = db.rawQuery(sSQL, null);
  if (cursor.moveToFirst()) {
    do {
      hm.put(ATR_AUTOR, cursor.getString(1));
      hm.put(ATR_KNIHA, cursor.getString(2));
      hm.put(ATR_DATUM, cursor.getString(3));
      hm.put(ATR_HODNOCENI, cursor.getString(4));
    } while (cursor.moveToNext());
  }
  return hm;
}
```

Kompletní kód třídy `DataModel`:

```
public class DataModel extends SQLiteOpenHelper {

  protected static final String DB_DATABAZE = "dbKknihy";
  protected static final int DB_VERZE = 1;
  protected static final String DB_TABULKA = "tblKnihy";

  //atributy sloupce tabulky
  public static final String ATR_ID = "_id";
  public static final String ATR_AUTOR = "autor";
  public static final String ATR_KNIHA = "kniha";
  public static final String ATR_DATUM = "datum";
  public static final String ATR_HODNOCENI = "hodnoceni";

  public DataModel(Context ctx) {
    super(ctx, "DB_DATABAZE", null, DB_VERZE);
  }

  @Override
  public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + DB_TABULKA
      + " (" + ATR_ID + " INTEGER PRIMARY KEY,"
      + ATR_AUTOR  + " TEXT,"
      + ATR_KNIHA + " TEXT,"
      + ATR_DATUM + " TEXT,"
      + ATR_HODNOCENI + " TEXT"
      + ");");
  }

  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    String query = "DROP TABLE IF EXISTS " + DB_TABULKA;
    db.execSQL(query);
```

```
  onCreate(db);
}

public long vlozZaznam(HashMap<String, String> atributy) {
  SQLiteDatabase db = this.getWritableDatabase();
  ContentValues val = new ContentValues();
  val.put(ATR_AUTOR, atributy.get(ATR_AUTOR));
  val.put(ATR_KNIHA, atributy.get(ATR_KNIHA));
  val.put(ATR_DATUM, atributy.get(ATR_DATUM));
  val.put(ATR_HODNOCENI, atributy.get(ATR_HODNOCENI));
  long id = db.insert(DB_TABULKA, null, val);
  db.close();
  return id;
}

public int aktualizujZaznam(HashMap<String, String> atributy) {
  SQLiteDatabase db = this.getWritableDatabase();
  ContentValues val = new ContentValues();
  val.put(ATR_AUTOR, atributy.get(ATR_AUTOR));
  val.put(ATR_KNIHA, atributy.get(ATR_KNIHA));
  val.put(ATR_DATUM, atributy.get(ATR_DATUM));
  val.put(ATR_HODNOCENI, atributy.get(ATR_HODNOCENI));
  return db.update(DB_TABULKA, val, ATR_ID + " = ?",
  new String[] {atributy.get(ATR_ID)});
}

public void vymazZaznam(String id)  {
  SQLiteDatabase db = this.getWritableDatabase();
  String deleteQuery =
    "DELETE FROM " + DB_TABULKA + " WHERE _id='"+ id +"'";
  db.execSQL(deleteQuery);
}

public ArrayList<HashMap<String, String>> dajZaznamy()  {
  ArrayList<HashMap<String, String>> alVysledky;
  alVysledky = new ArrayList<HashMap<String, String>>();
  String sSQL = "SELECT * FROM " + DB_TABULKA;
  SQLiteDatabase db = this.getWritableDatabase();
  Cursor cursor = db.rawQuery(sSQL, null);

  //Presun na prvy zaznam
  if (cursor.moveToFirst()) {
    do {
      HashMap<String, String> hm = new HashMap<String, String>();
      hm.put(ATR_ID, cursor.getString(0));
      hm.put(ATR_AUTOR, cursor.getString(1));
      hm.put(ATR_KNIHA, cursor.getString(2));
      hm.put(ATR_DATUM, cursor.getString(3));
      hm.put(ATR_HODNOCENI, cursor.getString(4));
      alVysledky.add(hm);
    } while (cursor.moveToNext()); // kurzor na dalsi zaznam
```

```
    }
    return alVysledky;
  }

  public HashMap<String, String> dajZaznam(String id)
  {
    HashMap<String, String> hm = new HashMap<String, String>();
    SQLiteDatabase db = this.getReadableDatabase();
    String sSQL = "SELECT * FROM " + DB_TABULKA + " WHERE _id='"+id+"'";
    Cursor cursor = db.rawQuery(sSQL, null);
    if (cursor.moveToFirst()) {
      do {
        hm.put(ATR_AUTOR, cursor.getString(1));
        hm.put(ATR_KNIHA, cursor.getString(2));
        hm.put(ATR_DATUM, cursor.getString(3));
        hm.put(ATR_HODNOCENI, cursor.getString(4));

      } while (cursor.moveToNext());
    }
    return hm;
  }
}
```

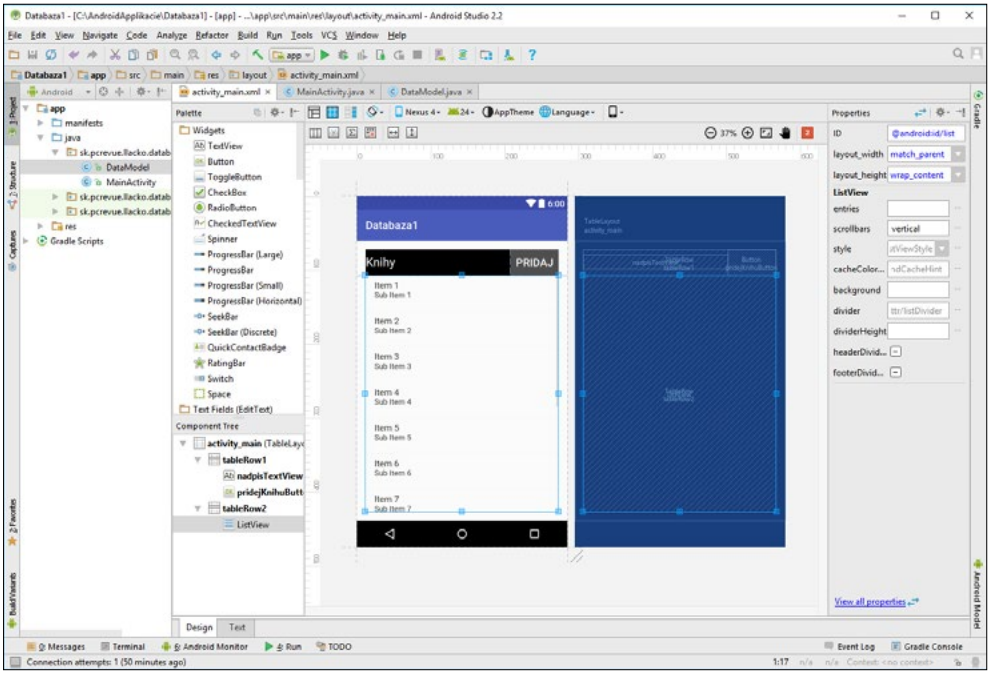
Návrh uživatelského rozhraní

Podobně jako v předchozích kapitolách můžete uživatelské rozhraní vytvořit buď v návrhovém prostředí Android Studia, nebo přímo v definičním souboru XML, případně kombinací obou metod. V tomto příkladu budete pravděpodobně kopírovat bloky kódu návrhu formuláře pro zadávání a editování atributů záznamu, jelikož v aktivitě přidání nového záznamu jsou stejné jako v aktivitě editování.

Hlavní aktivita

V hlavní aktivitě zobrazíme pomocí prvku `ListView` záznamy a v záhlaví zobrazíme tlačítko na přidání nového záznamu. Jelikož potřebujeme obrazovku aktivity vertikálně rozdělit na dvě části (záhlaví a seznam položek), jednou z alternativ je layout `TableLayout`. Tabulka je rozdělena na dvě sekce `TableRow`.

V souboru *activity_main.xml* změňte typ layoutu z `RelativeLayout` na `TableLayout`. Nemusíte editovat přímo soubor. I tuto změnu můžete realizovat ve vývojovém prostředí Android Studio. V prvním úzkém řádku je název seznamu, v tomto případě „Knihy“, realizovaný jako prvek `TextView` a tlačítko (prvek `Button`) na přidání nového záznamu do čtenářského deníku. V druhém řádku, který vyplňuje zbytek obrazovky, je prvek `ListView` určen na výpis jednotlivých položek.



Obrázek 10.10: Návrh uživatelského rozhraní hlavní aktivity

Výsledek vizuálního návrhu si můžete prohlédnout v souboru *aktivity_main.xml*:

```
TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="sk.pcrevue.llacko.databaza1.MainActivity">

  <TableRow
    android:id="@+id/tableRow1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#000000" >
    <TextView
      android:id="@+id/nadpisTextView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Knihy"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:textColor="#FFFFFF"
      android:layout_weight="1"/>
```

```
    <Button
      android:id="@+id/pridejKnihuButton"
      android:background="#444444"
      android:onClick="pridajKnihu"
      android:text="Pridaj"
      android:textColor="#FFFFFF"
      android:textSize="20sp" />
  </TableRow>

  <TableRow
    android:id="@+id/tableRow2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <ListView
      android:id="@android:id/list"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1">
    </ListView>
  </TableRow>
</TableLayout>
```

K zobrazení jednotlivých záznamů slouží prvek `ListView`. K zobrazení každé z položek je využit layout *polozka.xml*. Tento layout využívá kořenový element typu `TableRow`.

Prvek `TextView` na zobrazení identifikátoru se nezobrazuje, ale musí být součástí položky, aby po klepnutí na některý záznam bylo možné určit jeho identifikátor.

```
<TableRow xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical" >
  <TextView
    android:id="@+id/knihaId"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:visibility="gone" />

  <TextView
    android:id="@+id/knihaAutor"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#444444"
    android:textSize="20sp"
    android:textStyle="bold" />
  <TextView
    android:id="@+id/knihaNazev"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#444444"
```

```
    android:textSize="20sp"
    android:textStyle="bold" />
</TableRow>
```

Aktivita na přidání záznamu

Obrazovka aktivity `PridajZaznam`, kterou je potřeba vytvořit, obsahuje prvky `EditView` na zadávání hodnot atributů a tlačítko, kterým se potvrzuje uložení zadaných údajů.

Kód v souboru *pridej_zaznam.xml* (kvůli úspoře místa je v kódu uveden jen první z atributů):

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <TableRow
    android:id="@+id/tableRow1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#000000" >

    <TextView
      android:id="@+id/pridejTextView"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/nadpis_pridej"
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:textColor="#FFFFFF" />
  </TableRow>

  <TableRow
    android:id="@+id/tableRow2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
      android:id="@+id/autorTV"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
       android:text="@string/s_autor" />

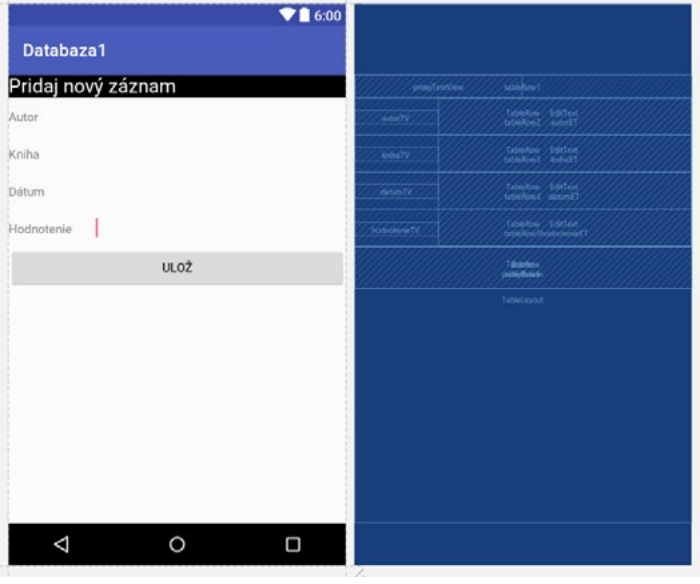
    <EditText
      android:id="@+id/autorET"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:ems="20"
      android:inputType="textPersonName">
      <requestFocus />
    </EditText>

  </TableRow>
```

```
    ...

    <TableRow
      android:id="@+id/tableRow6"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <Button
        android:id="@+id/pridejButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bt_ulož"
        android:onClick="pridajZaznam"
        android:layout_weight="1"/>
    </TableRow>
  </TableLayout>
```



Obrázek 10.11: Obrazovka aktivity na přidání záznamu

Aktivita na editování existujícího záznamu

Podobně jako aktivita na přidání záznamu i obrazovka této aktivity obsahuje prvky `EditView` na editování hodnot atributů. Při inicializaci aktivity jsou v těchto prvcích hodnoty atributů editovaného záznamu. Aktivita obsahuje dvě tlačítka: jedno, kterým se potvrzuje uložení zadaných údajů, a druhé na vymazání aktuálního záznamu.

Kód v souboru *edituj_zaznam.xml*. Kvůli úspoře místa je v kódu uveden jen první z atributů.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
```

```
<TableRow
  android:id="@+id/tableRow1"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:background="#000000" >

  <TextView
    android:id="@+id/editujTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/nadpis_edituj"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textColor="#FFFFFF" />
</TableRow>

<TableRow
  android:id="@+id/tableRow2"
  android:layout_width="match_parent"
  android:layout_height="wrap_content">

  <TextView
    android:id="@+id/autorTV"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="@string/s_autor" />

  <EditText
    android:id="@+id/autorET"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:ems="20"
    android:inputType="textPersonName">
    <requestFocus />
  </EditText>

</TableRow>

...

<TableRow
  android:id="@+id/tableRow6"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" >

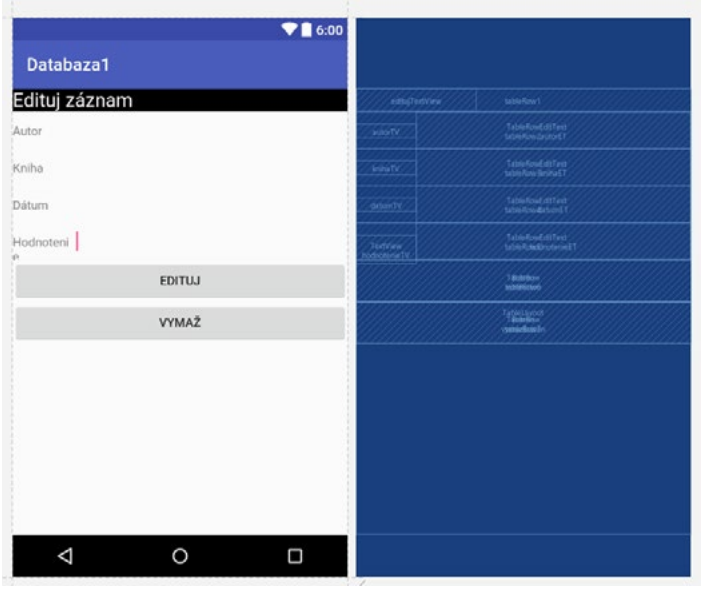
  <Button
    android:id="@+id/editButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="editujZaznam"
```

```
        android:text="@string/bt_edituj"
        android:layout_weight="1"/>
    </TableRow>

    <TableRow
      android:id="@+id/tableRow7"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >

      <Button
        android:id="@+id/vymazButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="vymazZaznam"
        android:text="@string/bt_vymaz"
        android:layout_weight="1"/>

    </TableRow>
  </TableLayout>
```



Obrázek 10.12: Obrazovka aktivity na editování, případně vymazání záznamu

Programování aplikační logiky

Aplikační logiku tvoří v tomto případě kód tří aktivit:

- *MainActivity.java*
- *PridajZaznam.java*
- *EditujZaznam.java*

Všechny tři aktivity úzce spolupracují s datovým modelem v třídě `DataModel`.

Hlavní aktivita

Původně je v kódu vygenerovaném při vytváření projektu hlavní třída aplikace, v našem případě třída `MainActivity`, odvozená od třídy `Activity`. V tomto příkladu je potřeba změnit definici třídy a odvodit ji od třídy `ListActivity`, jelikož na stránce hlavní aktivity se bude vypisovat seznam položek pomocí prvku `ListView`.

```
public class MainActivity extends ListActivity
```

Údaje z databáze budou uloženy v objektu typu `ArrayList`.

```
ArrayList<HashMap<String, String>> knihyHM = dm.dejZaznamy();
```

`ArrayList` se naplní metodou třídy datového modelu `dejZaznamy()`.

K obsluze klepnutí na položku seznamu vytvořte listener typu `setOnItemClickListener` a přidejte neimplementovanou metodu `onItemClick()`.

Po klepnutí na položku se z prvku `TextView knihaId` zjistí unikátní identifikátor vybraného záznamu, tedy atribut `_id`. Tento parametr se pomocí objektu `Intent` odevzdá aktivitě `EditujZaznam`. Tlačítkem **Přidej** se aktivuje metoda `pridejKnihu`, která zobrazí aktivitu `PridejZaznam`.

Kompletní kód v souboru *MainActivity.Java*:

```
public class MainActivity extends ListActivity {

  Intent intent;
  TextView knihaId;
  DataModel dm = new DataModel(this);

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ArrayList<HashMap<String, String>> knihyHM = dm.dajZaznamy();

    if(knihyHM.size()!=0) {
      ListView lw = getListView();
      lw.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        //kliknuti na polozku zoznamu
        public void onItemClick(AdapterView<?> parent,
          View view, int position, long id)  {
          knihaId = (TextView) view.findViewById(R.id.knihaId);
          String sKnihaId = knihaId.getText().toString();
          Intent  theIndent = new
            Intent(getApplication(),EditujZaznam.class);
          theIndent.putExtra("knihaId", sKnihaId);
          startActivity(theIndent);
        }
      });
      ListAdapter adapter = new SimpleAdapter( MainActivity.this,knihyHM,
        R.layout.polozka,
      new String[] { "_id","autor", "kniha"}, new int[] {R.id.knihaId,
        R.id.knihaAutor, R.id.knihaNazov});
      setListAdapter(adapter);
    }
```

```
  }

  public void pridajKnihu(View view) {
    Intent theIntent = new Intent(getApplication(), PridajZaznam.class);
    startActivity(theIntent);
  }
}
```

Aktivita na přidání záznamu

Po potvrzení zadaných údajů se vytvoří objekt `HashMap` obsahující hodnoty atributů a tento objekt se odevzdá metodě datového modelu `vlozZaznam()`. Po přidání záznamu se děj přenese zpět na hlavní obrazovku.

Kompletní kód v souboru *PridajZaznam.Java*:

```
public class PridajZaznam extends Activity {
  EditText autorET;
  EditText knihaET;
  EditText datumET;
  EditText hodnotenieET;

  DataModel dm = new DataModel(this);

  @Override
  public void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_pridaj_zaznam);

    autorET = (EditText) findViewById(R.id.autorET);
    knihaET = (EditText) findViewById(R.id.knihaET);
    datumET = (EditText) findViewById(R.id.datumET);
    hodnotenieET = (EditText) findViewById(R.id.hodnotenieET);
  }

  public void pridejZaznam(View view)  {
    HashMap<String, String> dotazHM =  new  HashMap<String, String>();

    dotazHM.put("autor", autorET.getText().toString());
    dotazHM.put("kniha", knihaET.getText().toString());
    dotazHM.put("datum", datumET.getText().toString());
    dotazHM.put("hodnoceni", hodnotenieET.getText().toString());

    dm.vlozZaznam(dotazHM);
    // vola MainActivity
    this.aktivujMainActivity(view);
  }

  public void aktivujMainActivity(View view)  {
    Intent ii = new Intent(getApplication(), MainActivity.class);
    startActivity(ii);
  }
}
```

Aktivita na editování záznamu

Aktivita prostřednictvím objektu typu `Intent` dostane od hlavní aktivity jako parametr identifikátor záznamu, který chce uživatel editovat, případně vymazat. Atributy záznamu se naplní prvky `EditView`. Aktivita obsahuje i metody na obsluhu událostí klepnutí na tlačítka **Změň** a **Vymaž**.

Kompletní kód v souboru *EditujZaznam.java*:

```
public class EditujZaznam extends Activity {
  EditText autorET;
  EditText knihaET;
  EditText datumET;
  EditText hodnoceniET;

  DataModel dm = new DataModel(this);

  public void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_edituj_zaznam);

    autorET = (EditText) findViewById(R.id.autorET);
    knihaET = (EditText) findViewById(R.id.knihaET);
    datumET = (EditText) findViewById(R.id.datumET);
    hodnoceniET = (EditText) findViewById(R.id.hodnotenieET);

    Intent ii = getIntent();
    String knihaId = ii.getStringExtra("knihaId");
    HashMap<String, String> knihaHM = dm.dajZaznam(knihaId);

    if(knihaHM.size()!=0) {
      autorET.setText(knihaHM.get("autor"));
      knihaET.setText(knihaHM.get("kniha"));
      datumET.setText(knihaHM.get("datum"));
      hodnoceniET.setText(knihaHM.get("hodnoceni"));
    }
  }

  public void editujZaznam(View view)  {
    HashMap<String, String> dotazHM =  new  HashMap<String, String>();

    autorET = (EditText) findViewById(R.id.autorET);
    knihaET = (EditText) findViewById(R.id.knihaET);
    datumET = (EditText) findViewById(R.id.datumET);
    hodnoceniET = (EditText) findViewById(R.id.hodnotenieET);

    Intent ii = getIntent();
    String sID = ii.getStringExtra("knihaId");

    dotazHM.put("id", sID);
    dotazHM.put("autor", autorET.getText().toString());
    dotazHM.put("kniha", knihaET.getText().toString());
    dotazHM.put("datum", datumET.getText().toString());
    dotazHM.put("hodnoceni", hodnoceniET.getText().toString());
```

```
        dm.aktualizujZaznam(dotazHM);
        this.aktivujMainActivity(view);
    }

    public void vymazZaznam(View view)   {
        Intent ii = getIntent();
        String sID = ii.getStringExtra("knihaId");
        dm.vymazZaznam(sID);
        this.aktivujMainActivity(view);
    }


    public void aktivujMainActivity(View view)   {
        Intent ii = new Intent(getApplication(), MainActivity.class);
        startActivity(ii);
    }

}
```

Nezapomeňte obě nové aktivity přidat v aplikačním manifestu *AndroidManifest.xml* do seznamu aktivit.

Nyní můžete aplikaci otestovat. Přidejte několik záznamů, zkuste je editovat a mazat.



Obrázek 10.13: Spuštění aplikace – obrazovka aktivity na přidání nového záznamu

Údaje z webu ve formátech JSON a XML

Vytvořte aplikaci, která bude zobrazovat údaje z vámi vybraného webového zdroje údajů.

Rozbor zadání

Na první pohled by se mohlo zdát, že zařádil tiskařský šotek a vymazal zbytek zadání. Ne, příklad v této části má skutečně takto krátké, a navíc všeobecné zadání. Při zadání konkrétní aplikace od konkrétního zákazníka samozřejmě dostanete adresu serveru nebo webové služby, která poskytuje mobilní aplikaci údaje. Případně bude součástí zadání i nutnost vytvořit webovou aplikaci nebo službu poskytující údaje.

Druhou neméně důležitou otázkou je formát poskytovaných údajů. U existující služby se budete muset přizpůsobit a aplikace bude „konzumovat" údaje v takovém formátu, v jakém je služba poskytuje. Pokud budete službu, serverovou či webovou aplikaci na generování údajů vytvářet sami, můžete si formát zvolit podle svých preferencí. Nejčastěji se údaje generují ve formátech JSON a XML. Tyto formáty jsou populární hlavně kvůli tomu, že jsou nezávislé na platformě.

Námětem kapitoly proto bude vytvoření dvou aplikací, které budou zpracovávat a zobrazovat údaje. Jedna aplikace bude pracovat s údaji ve formátu formátu JSON a druhá s údaji ve formátu XML. Využijeme stejný zdroj údajů – populární portál Yahoo Developer Network, který umožňuje přepínat formát generovaných údajů.

Zdroj údajů pro aplikaci

Jako zdroj údajů ve formátu JSON pro cvičnou aplikaci můžete využít populární portál Yahoo Developer Network. Údaje můžete vybírat dotazem podobným SQL přes webovou konzoli *https://developer.yahoo.com/yql/console/*.

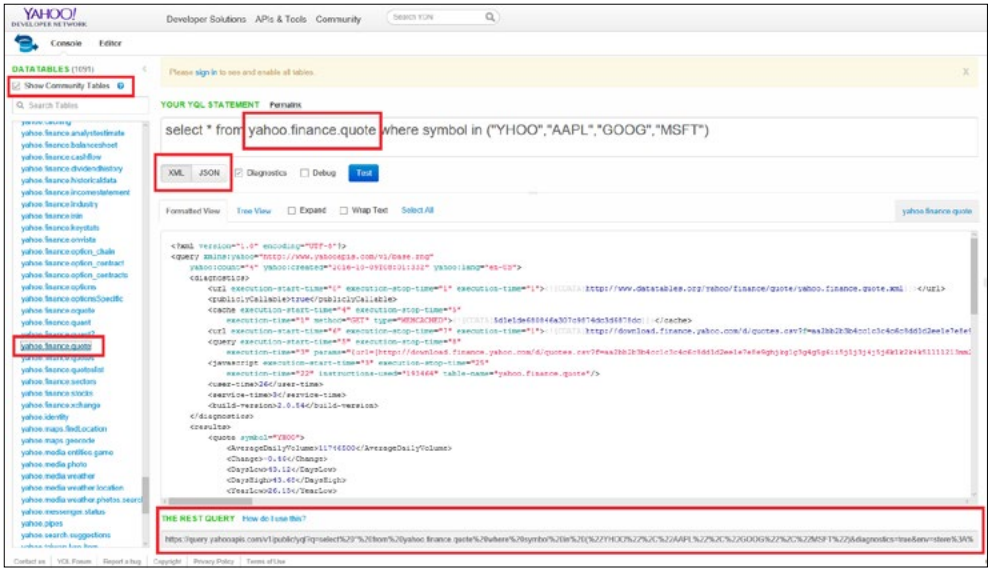
První, co vás napadne, bude pravděpodobně nejpopulárnější služba Yahoo Weather, která poskytuje informace o počasí. Jako parametr je potřeba specifikovat město a jednotky zobrazení teploty. Město se zadává nikoliv pomocí názvu, ale využívá se parametr WOEID (Where On Earth IDentifier). Pro zajímavost: WOEID Prahy je 796597 a Bratislavy 818717. Takže pokud potřebujete údaje o počasí pro Prahu, použili byste dotaz:

```
select * from weather.forecast where woeid=796597
```

Pro jednoduchou cvičnou aplikaci, ve které se chceme zaměřit na způsob načítání údajů ve formátech JSON a XML, a ne na jejich zpracování a obrazování, však mají údaje ze služby Yahoo Weather poměrně složitou strukturu. Doporučujeme proto něco jednoduššího, například údaje o aktuálním stavu a trendu akcií firem ze služby yahoo.finance. Abyste měli přístup k této komunitní tabulce, musíte na webové stránce, ze které zadáváte dotaz, zaškrtnout položku **Show Community Tables**. Stav a trend akcií firmy Google zobrazíte pomocí dotazu:

```
select * from yahoo.finance.quote where symbol in("GOOG")
```

Aby se negenerovaly i diagnostické údaje, pole **Diagnostics** nesmí být označeno.



Obrázek 10.14: Webová stránka zdroje údajů

Podle toho, zda stisknete tlačítko **JSON** nebo **XML**, vygeneruje aplikace údaje v požadovaném formátu.

Příklad údajů ve formátu JSON:

```
{
  "query": {
  "count": 1,
  "created": "2016-10-09T05:34:31Z",
  "lang": "en-US",
  "results": {
    "symbol": "AAPL",
    "AverageDailyVolume": "35386200",
    "Change": "+0.17",
    "DaysLow": "113.51",
    "DaysHigh": "114.56",
    "YearLow": "89.47",
    "YearHigh": "123.82",
    "MarketCapitalization": "614.61B",
    "LastTradePriceOnly": "114.06",
    "DaysRange": "113.51 - 114.56",
    "Name": "Apple Inc.",
    "Symbol": "AAPL",
    "Volume": "24358443",
    "StockExchange": "NMS"
  },
  }
}
```

Příklad údajů ve formátu XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng"
  yahoo:count="4" yahoo:created="2016-10-09T08:07:48Z" yahoo:lang="en-US">
...
  <results>
    <quote symbol="AAPL">
      <AverageDailyVolume>35386200</AverageDailyVolume>
      <Change>+0.17</Change>
      <DaysLow>113.51</DaysLow>
      <DaysHigh>114.56</DaysHigh>
      <YearLow>89.47</YearLow>
      <YearHigh>123.82</YearHigh>
      <MarketCapitalization>614.61B</MarketCapitalization>
      <LastTradePriceOnly>114.06</LastTradePriceOnly>
      <DaysRange>113.51 - 114.56</DaysRange>
      <Name>Apple Inc.</Name>
      <Symbol>AAPL</Symbol>
      <Volume>24358443</Volume>
      <StockExchange>NMS</StockExchange>
    </quote>
  </results>
</query>
```

Pro vás bude důležité pole **THE REST QUERY** v dolní části formuláře webového portálu, které obsahuje kompletní adresu URL zdroje údajů pro vaši aplikaci. Tuto adresu zkopírujete do schránky a přidáte jako textový řetězec do kódu aplikace. Pokud chcete zobrazit údaje o akciích více firem, můžete jich v dotazu nebo i uvozovkách na příslušném místě URL adresy vložit více a oddělit je čárkou:

```
select * from yahoo.finance.quote where symbol in("GOOG,MSFT")
```

Pokud chcete možnosti aplikace rozšířit, můžete měnit textový řetězec s označením firmy uvnitř adresy URL: Řetězec s adresou URL můžete rozdělit na tři řetězce, přičemž prostřední z nich bude proměnná typu `string`, která bude obsahovat zkratku aktuálně vybrané firmy.

Návrh uživatelského rozhraní

Jelikož budou obě aplikace využívat stejný zdroj údajů a lišit se bude pouze jejich formát, návrh uživatelského rozhraní a definice textových řetězců budou stejné pro obě aplikace.

Abychom návrh co nejvíc zjednodušili, budeme zobrazovat jen dvousloupcovou tabulku hodnot popis-hodnota. Na hlavní aktivitu můžeme využít vizuální kontejner `GridLayout`. Pro jednoduchost zobrazíme jen tři údaje: změnu hodnoty akcií a minimální a maximální denní hodnotu.

Uživatelské rozhraní hlavní aktivity v souboru *Activity_main.xml*:

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:useDefaultMargins="true"
  android:alignmentMode="alignBounds"
```

```
    android:columnOrderPreserved="false"
    android:columnCount="3"
    tools:context="sk.pcrevue.llacko.prikladjson.MainActivity" >

    <TextView
      android:layout_gravity="right"
      android:text="Zmena" />

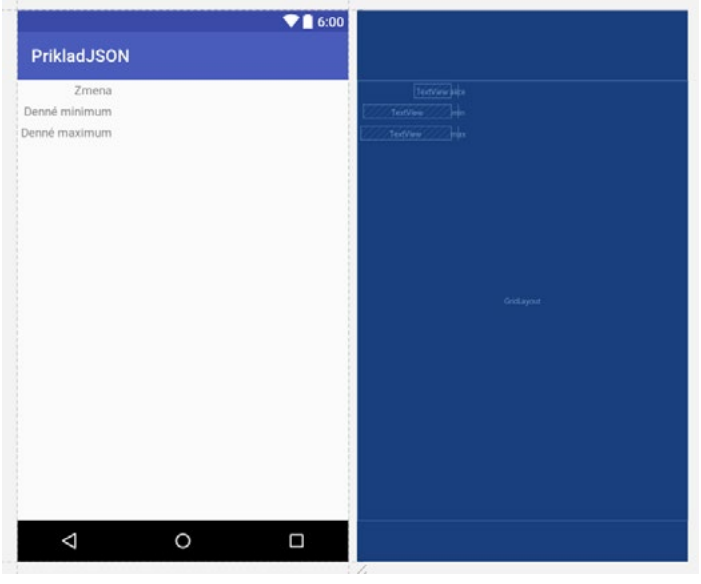
    <TextView
      android:layout_column="1"
      android:id="@+id/akce"/>

    <TextView
      android:layout_column="0"
      android:layout_gravity="right"
      android:text="Denné minimum" />

    <TextView
      android:layout_column="1"
      android:id="@+id/min"/>

    <TextView
      android:layout_column="0"
      android:layout_gravity="right"
      android:text="Denné maximum" />

    <TextView
      android:layout_column="1"
      android:id="@+id/max"/>
</GridLayout>
```



Obrázek 10.15: Návrh uživatelského rozhraní aplikace na zobrazování údajů z webu

Aplikace na načítání údajů ve formátu JSON

Hodně webových služeb poskytuje údaje v populárním formátu JSON (JavaScript Object Notation). Referenci formátu najdete na *http://www.json.org*. Dokument JSON je textový a obsahuje přísně strukturované údaje, které jsou dobře čitelné pro člověka i počítač. Formát JSON podporuje většina programovacích jazyků. Údaje ve formátu JSON mohou obsahovat atributy typu `String`, `Boolean`, `Array`, `Object`, `Number` a `NULL`.

Definice formátu v Backus-Naurově formátu zabere jen několik řádků.

```
<objekt> ::= {} | { <páry> }
<páry> ::= <pár> | <pár> , <páry>
<pár> ::= <reťazec> : <hodnota>
<pole> ::= [] | [ <prvky> ]
<prvky> ::= <hodnota> | <hodnota> , <prvky>
<hodnota> ::= <reťazec> | <číslo> | <objekt> | <pole> | true | false | null
```

Syntaxe je vybudovaná na dvou základních strukturách:

- množina párových údajů typu klíč-hodnota
- seznam hodnot

Činnost aplikačního kódu můžeme rozdělit do čtyř na sebe navazujících fází:

- Načítání údajů z REST API přes HTTP řetězec
- Převod načítaných údajů na objekt JSON
- Parsování objektu JSON za účelem zjištění hodnoty sledovaných atributů
- Zobrazení načítaných údajů

První tři fáze budou realizované na pozadí v asynchronní třídě, čtvrtá fáze nastoupí po úspěšném načtení a zpracování údajů.

Na analýzu údajů uložených ve formátu JSON použijeme příslušné objekty třídy `JsonReader`. Ta je určena ke čtení (analýze) údajů uložených (zakódovaných) v proudu JSON (stream). Údaji přitom mohou být řetězce, čísla, logické stavy, resp. hodnoty `NULL`. Údaje nejsou směrovány přímo koncovým proměnným, ale mezistupeň je zformovaný pomocí třídy `BufferedReader`, která zvyšuje výkon (performance) čtení údajů.

Příprava na odesílání a příjem údajů na serveru, resp. ze serveru, se realizuje spuštěním „proudového zapisovače" (StreamWriter) a „proudového čítače" (StreamReader), jehož výstup je směřován právě na zmíněný `BufferedReader`. Na zpracování proudu obsahujícího pole používáme metody `beginArray()`, `hasNext()` a `endArray()`. Naopak při zpracování proudu obsahujícího objekty používáme metody `beginObject()`, `hasNext()` a `endObject()`.

Důležitou úlohou je implementace asynchronní práce (AsyncTask) se zdrojem údajů nacházejícím se v externí síti (na internetu). AsyncTask vytváří samostatné vlákno (thread) určené k vykonání (execute) časově náročnějších úloh na pozadí.

Načítání údajů z webové služby bude proto probíhat v asynchronní třídě:

```
NactiUdajeAsyncTask extends AsyncTask<String, String, String>.
```

Po vytvoření „prázdného" kódu třídy přidejte přes položku místní nabídky **Add unimplemented methods** metodu:

```
protected String doInBackground(String... params)
```

V kódu metody vytvořte objekt `DefaultHttpClient` a přes `HttpPost` otevřete propojení na adresu URL webové služby. Abyste dostali údaje ve formátu JSON, musíte v hlavičce specifikovat řetězec.

```
httppost.setHeader("Content-type", "application/json");
```

Parser formátu JSON bohužel neumí přímo načítat z InputStreamu, proto je potřeba obsah nejprve uložit do textového řetězce přes `BufferedReader` a tento řetězec následně poslat jako parametr do konstruktoru `JsonObject`.

```
jsonObject = new JSONObject(sJSON);
```

Údaje ze služby budou načítány prostřednictvím objektu typu `InputStream` přes `HttpResponse`. Jelikož se jedná o textový formát, na jeho načítání po řádcích využijeme objekt `BufferedReader` a na uložení objekt typu `StringBuilder`.

```
HttpResponse response = klientHTTP.execute(httppost);
HttpEntity entity = response.getEntity();
inputStream = entity.getContent();
BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
  "UTF-8"), 8);
StringBuilder sb = new StringBuilder();
String radek = null;

while ((radek = reader.readLine()) != null)
{
  sb.append(radek + "\n");
}
sJSON = sb.toString();
```

V této fázi jsou údaje v textové podobě v řetězci JSON. Vaším úkolem je získat ze struktury hierarchického formátu JSON hodnoty atributů, které bude aplikace vypisovat.

Pokud se podíváte na údaje JSON, zjistíte, že vámi požadované atributy `symbol`, `DaysLow` a `DaysHigh` jsou až na třetí úrovni hierarchie.

```
"query": {
  "results": {
  "quote": {
    "symbol": "GOOG",
    "Change": "+10.36",
    "DaysLow": "544.00",
    "DaysHigh": "553.56",
```

Musíte tedy každý záznam dokumentu JSON parsovat postupně přes uzly:

```
- query
- - results
- - quote
```

```
JSONObject queryJSONObject = jsonObject.getJSONObject("query");
```

```
JSONObject resultsJSONObject = queryJSONObject.getJSONObject("results");
JSONObject quoteJSONObject = resultsJSONObject.getJSONObject("quote");
```

Až na této úrovni hierarchie v sekci quote můžete načítat hodnoty atributů pomocí příkazu:

```
sDenniMinimum = quoteJSONObject.getString("DaysLow");
```

Po načítání údajů a ukončení metody asynchronní třídy můžete tyto hodnoty vložit do prvků typu TextView v metodě onPostExecute(String result):

```
protected void onPostExecute(String result)
{
  TextView min = (TextView)findViewById(R.id.min);
  min.setText(sDenniMinimum);
  ...
}
```

Kompletní kód aplikace v souboru *MainActivity.java* (URL adresa je pro úsporu místa zkrácená), všimněte si ošetření potenciálních výjimek:

```
public class MainActivity extends AppCompatActivity {

  static String sURL = "https://query.yahooapis.com/v1/public/yql?q= ";
  static String sFirma = "";
  static String sDenniMinimum = "";
  static String sDenniMaximum = "";
  static String sZmena = "";

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new NacitajUdajeAsyncTask().execute();
  }

  private static final String TAG = "JSON priklad";

  private class NacitajUdajeAsyncTask extends
    AsyncTask<String, String, String> {

    protected String doInBackground(String... arg0) {
      String response = null;
      String sJSON = null;
      InputStream in = null;
      try {
        URL url = new URL(sURL);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");

        in = new BufferedInputStream(conn.getInputStream());
        BufferedReader reader = new BufferedReader(new
          InputStreamReader(in, "UTF-8"), 8);
        StringBuilder sb = new StringBuilder();
        String radek = null;
```

```
            while ((radek = reader.readLine()) != null) {
              sb.append(radek + "\n");
            }
            sJSON = sb.toString();

          } catch (MalformedURLException e) {
            Log.e(TAG, "MalformedURLException: " + e.getMessage());
          } catch (ProtocolException e) {
            Log.e(TAG, "ProtocolException: " + e.getMessage());
          } catch (IOException e) {
            Log.e(TAG, "IOException: " + e.getMessage());
          } catch (Exception e) {
            Log.e(TAG, "Exception: " + e.getMessage());
          }
          finally  {
            try{if(in != null)in.close();}
            catch(Exception e){}
          }

          JSONObject jsonObject;
          try  {
            jsonObject = new JSONObject(sJSON);
            JSONObject queryJSONObject = jsonObject.getJSONObject("query");
            JSONObject resultsJSONObject =
              queryJSONObject.getJSONObject("results");
            JSONObject quoteJSONObject =
              resultsJSONObject.getJSONObject("quote");

            sFirma = quoteJSONObject.getString("symbol");
            sDenniMinimum = quoteJSONObject.getString("DaysLow");
            sDenniMinimum = quoteJSONObject.getString("DaysHigh");
            sZmena = quoteJSONObject.getString("Change");
          } catch (JSONException e) {
            e.printStackTrace();
          }
          return sJSON;
        }


        protected void onPostExecute(String result) {
          TextView akce = (TextView)findViewById(R.id.akce);
          TextView min = (TextView)findViewById(R.id.min);
          TextView max = (TextView)findViewById(R.id.max);

          akce.setText(sFirma + " : " + sZmena);
          min.setText(sDenniMinimum);
          max.setText(sDenniMinimum);
        }
      }
    }
```

Aby se vaše aplikace mohla připojovat k údajům z internetu, musíte v aplikačním manifestu v souboru *AndroidManifest.xml* deklarovat příslušné oprávnění.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Po doplnění oprávnění do aplikačního manifestu můžete aplikaci vyzkoušet.



Obrázek 10.16: Zobrazování údajů z webu ve formátu JSON

Aplikace na načítání údajů ve formátu XML

Kromě srozumitelnosti je hlavní výhodou dokumentů XML univerzálnost a multiplatformovost, proto se velmi často využívají k výměně informací v heterogenním prostředí.

Na parsování údajů z dokumentu XML se využívá buď DOM (Document Object Model), nebo XmlPullParser. Proto i náš příklad bude mít dvě varianty, které se budou lišit použitou technologií parsování XML. Uživatelské rozhraní obou aplikací bude stejné – shodné s aplikací pro načítání údajů v JSON.

Varianta využívající Document Object Model

Document Object Model (*http://www.w3.org/DOM*) je platformově i jazykově neutrální rozhraní, které aplikacím umožňuje dynamický přístup k objektům v dokumentu XML. Umožňuje načítat dokument XML do objektové stromové struktury, se kterou je potom možné pohodlně manipulovat podle potřeby. Každý dokument podle DOM má hierarchickou stromovou strukturu.

K údajům XML přistupuje aplikace přes připojení realizované objektem typu `HttpURLConnection`. Objekt vznikl přetypováním objektu typu `URLConnection`. Dříve než začnete načítat údaje, je potřeba otestovat, zda se podařilo spojení se zdrojem údajů vytvořit. Stav spojení zjistíte pomocí metody `getResponseCode()`. Metoda musí vrátit hodnotu `200 - Good Connection`. V objektu `HttpURLConnection` je to konstanta `HttpURLConnection`.

```
URL url = new URL(sURL);
URLConnection connection;
connection = url.openConnection();
HttpURLConnection httpConnection = (HttpURLConnection)connection;
int kod = httpConnection.getResponseCode();

if (kod == HttpURLConnection.HTTP_OK)
{
  ...
}
```

Na parsování dokumentu XML je použit objekt `DocumentBuilder` z `DocumentBuilderFactory`. Instance `DocumentBuilderFactory` slouží ke správnému formátování stažených údajů při vytvá-

ření instance objektu `DocumentBuilder`. Tento objekt umožňuje jednoduše stáhnout údaje ze serveru, přičemž tyto údaje se přímo zformátují do požadovaného formátu `Document`.

V příkladu předpokládáme jen jednu hodnotu (jeden element `<quote>`), jelikož nás zajímají jen akcie jediné firmy. Pokud bychom chtěli zjistit akcie více firem, příslušná služba použitá jako zdroj údajů by vrátila více elementů `<quote>`. Jejich hodnoty byste načítali v cyklu typu:

```
for (int i = 0 ; i < nl.getLength(); i++)
{
  ...
}
```

Objekt typu `Document` vznikne parsováním objektu `InputStream`:

```
InputStream in = httpConnection.getInputStream();
DocumentBuilderFactory dbf =
  DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document dom = db.parse(in);
```

Hodnoty parametrů, které nás zajímají, přiřadíme do proměnných v metodě `hodnotaElementu()`, která vrátí hodnotu zapouzdřenou ve vybraném elementu ve formě textového řetězce. Metoda využívá objekt `NodeList`, který uchovává pořadí uzlů, jejich potomků a rodičů. Na něm jsou implementovány metody jako například `firstChild`, která vrátí prvního potomka, nebo `GetElementByTagName`, která vrátí element s daným `TagName`.

```
private String hodnotaElementu(Element entry, String tagName)
{
  String sHodnota = null;
  NodeList nl = entry.getElementsByTagName(tagName);
  if(nl != null && nl.getLength() > 0)
  {
    Element el = (Element) nl.item(0);
    sHodnota = el.getFirstChild().getNodeValue();
  }
  return sHodnota;
}
```

Kompletní kód aplikace v souboru *MainActivity.java* (URL adresa je pro úsporu místa zkrácená):

```
public class MainActivity extends Activity
{
  static String sURL = "https://query.yahooapis.com/v1/public/...";

  static String sFirma = "";
  static String sDenniMinimum = "";
  static String sDenniMaximum = "";
  static String sZmena = "";

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
  new NactiUdajeAsyncTask().execute();
}

private class NactiUdajeAsyncTask extends AsyncTask<String, String, String>
{

  @Override
  protected String doInBackground(String... params)
  {
    try
    {
      URL url = new URL(sURL);
      URLConnection connection;
      connection = url.openConnection();
      HttpURLConnection httpConnection = (HttpURLConnection)connection;
      int kod = httpConnection.getResponseCode();

      // bylo pripojeni uspesne?
      if (kod == HttpURLConnection.HTTP_OK)
      {
        InputStream in = httpConnection.getInputStream();
        DocumentBuilderFactory dbf =
          DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();
        Document dom = db.parse(in);

        //root element query
        Element rootEl = dom.getDocumentElement();

        //element quote
        NodeList nl = rootEl.getElementsByTagName("quote");
        if (nl != null && nl.getLength() > 0)
        {
          sFirma = hodnotaElementu(rootEl, "Symbol");
          sZmena = hodnotaElementu(rootEl, "Change");
          sDenniMinimum = hodnotaElementu(rootEl, "DaysLow");
          sDenniMaximum = hodnotaElementu(rootEl, "DaysHigh");
        }
      }
    }
    catch (MalformedURLException e)
    {
      Log.d("Chyba", "Malformed URL Exception", e);
    } catch (IOException e)
    {
      Log.d("Chyba", "IOException", e);
    } catch (ParserConfigurationException e)
    {
      Log.d("Chyba", "Parser Configuration Exception", e);
    } catch (SAXException e)
    {
```

```
            Log.d("Chyba", "SAX Exception", e);
          }
          finally {}

          return null;
        }

        protected void onPostExecute(String result)
        {
          TextView akce = (TextView)findViewById(R.id.akce);
          TextView min = (TextView)findViewById(R.id.min);
          TextView max = (TextView)findViewById(R.id.max);

          akce.setText(sFirma + " : " + sZmena);
          min.setText(sDenniMinimum);
          max.setText(sDenniMaximum);
        }

        private String hodnotaElementu(Element entry, String tagName)
        {
          String sHodnota = null;
          NodeList nl = entry.getElementsByTagName(tagName);
          if(nl != null && nl.getLength() > 0)
          {
            Element el = (Element) nl.item(0);
            sHodnota = el.getFirstChild().getNodeValue();
          }
          return sHodnota;
        }
      }
    }
```

Aby se vaše aplikace mohla připojovat k údajům z internetu, musíte v aplikačním manifestu v souboru *AndroidManifest.xml* deklarovat příslušné oprávnění:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Po doplnění oprávnění do aplikačního manifestu můžete aplikaci vyzkoušet.

Varianta využívající XmlPullParser

Načítání údajů z XML bude v této variantě realizované prostřednictvím objektu `XmlPullParser`. V porovnání s technologií Document Object Model (DOM), použitou v předchozím případě, je jednodušší, funguje efektivněji a spotřebuje méně operační paměti.

Klíčovou metodou je `getEventType()`, která vrací typ objektu, jímž právě parser prochází. Metoda může vrátit následující hodnoty (fyzicky se jedná o celočíselnou hodnotu):

- `START_DOCUMENT` – počáteční stav parseru
- `START_TAG` – počáteční tag elementu XML
- `TEXT` – indikuje, že parser je nastavený textový obsah elementu; hodnotu textového řetězce můžete načítat pomocí metody `getText()`

- END_TAG – koncový tag XML dokumentu
- END_DOCUMENT – konec dokumentu

Pomocí metody next() posunete parser na další objekt.

Pro ilustraci uvedeme fragment kódu nejjednodušší aplikace využívající XmlPullParser na parsování nejjednoduššího dokumentu XML <Pozdrav>Ahoj</Pozdrav>, uloženého v textovém řetězci. Tento fragment kódu je jen ilustrační, netýká se cvičného příkladu.

```
public static void main (String args[])
  throws XmlPullParserException, IOException
{
  XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
  factory.setNamespaceAware(true);
  XmlPullParser xpp = factory.newPullParser();

  xpp.setInput( new StringReader ( "<Pozdrav>Ahoj</Pozdrav>" ) );
  int typ = xpp.getEventType();
  while (typ != XmlPullParser.END_DOCUMENT)
  {
    if(typ == XmlPullParser.START_DOCUMENT)
    {
      System.out.println("START_DOCUMENT");
    }
    else if(typ == XmlPullParser.START_TAG)
    {
      System.out.println("START_TAG "+xpp.getName());
    }
    else if(typ == XmlPullParser.END_TAG)
    {
      System.out.println("END_TAG "+xpp.getName());
    }
    else if(typ == XmlPullParser.TEXT)
    {
      System.out.println("TEXT "+xpp.getText());
    }
    typ = xpp.next();
  }
  System.out.println("END_DOCUMENT");
}
```

Výsledek parsování pro dokument <Pozdrav>Ahoj</Pozdrav> bude:

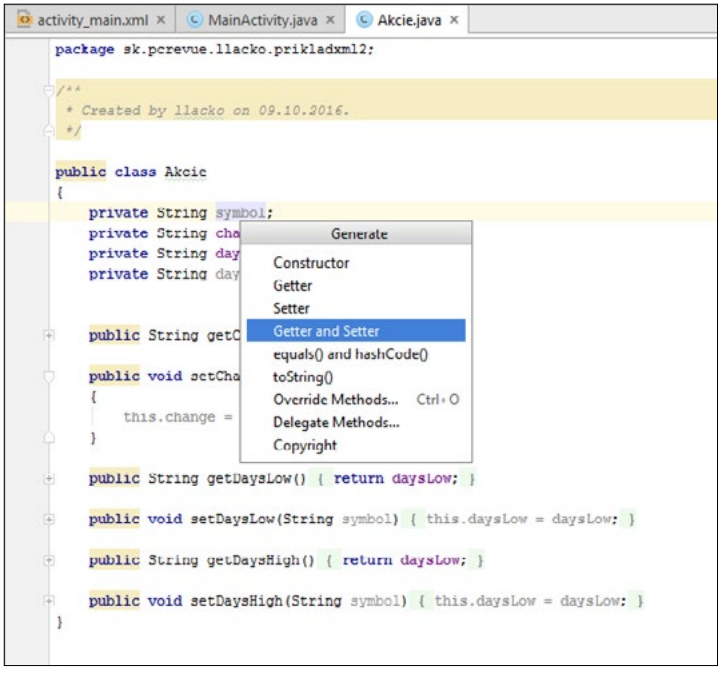
```
START_DOCUMENT
START_TAG Pozdrav
TEXT Ahoj
END_TAG Pozdrav
END_DOCUMENT
```

Pokud by bylo nutné načítat hodnotu atributu, použili byste metodu getAttributeValue():

```
sHodnotaAtributu = parser.getAttributeValue(null,"value");
```

Vraťme se však k cvičnému příkladu na zobrazení akcií vybrané firmy podle údajů ve formátu XML ze služby yahoo.finance.

Přestože v tomto nejjednodušším příkladu budeme zobrazovat jen údaje pro jeden objekt, v tomto případě informace o akciích jedné firmy, budeme postupovat systematicky a vytvoříme na zapouzdření načítaných údajů třídu. V kódu třídy `Akcie` budou jen atributy pro hodnoty, které potřebujeme načítat z XML, a metody pro nastavení atributu (setter) a získání jeho hodnoty (getter). Na obrázku je postup, jak tyto metody ve vývojovém prostředí Android Studio vygenerovat pro jednotlivé atributy automaticky. V místní nabídce atributu aktivujte funkci **Generate** nebo použijte klávesovou zkratku **Alt+Insert**.



Obrázek 10.17: Pomůcka pro vytvoření getteru a setteru

Pro úsporu místa jsou ve výpisu metody pro nastavení atributu (setter) a získání jeho hodnoty (getter) pouze pro jeden atribut.

```
public class Akcie
{
  private String symbol;
  private String change;
  private String daysLow;
  private String daysHigh;

  public String getSymbol()
  {
    return symbol;
  }
```

```
  public void setSymbol(String symbol)
  {
    this.symbol = symbol;
  }
  ...
}
```

I v této variantě se načítání dokumentu XML a parsování údajů odehrává v asynchronní třídě `NactiUdajeAsyncTask()`.

Při parsování elementů načítáme nejprve textový řetězec uvnitř elementu, a když parser načítá koncový tag elementu, podle jeho názvu přiřadíme textovou hodnotu příslušné třídy.

```
int typ = parser.getEventType();
while (typ != XmlPullParser.END_DOCUMENT)
{
  String sTag = parser.getName();
  switch (typ)
  {
    case XmlPullParser.END_TAG:
    if (tagname.equalsIgnoreCase("Change"))
    {
      akcie.setChange(text);
    }
...
```

Kompletní kód aplikace v souboru *MainActivity.java* (URL adresa je pro úsporu místa zkrácená):

```
public class MainActivity extends AppCompatActivity {

  static String sURL =
    "https://query.yahooapis.com/v1/public/yql?q=select%20*%20from%20yahoo.
    finance.quote%20where%20symbol%20in%20(%22AAPL%22)&env=
    store%3A%2F%2Fdatatables.org%2Falltableswithkeys";

  private Akcie akcie;
  private String text;

  @Override
  protected void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new NactiUdajeAsyncTask().execute();
  }

  public void parsujXML(XmlPullParser parser)  {
    try {
      int typ = parser.getEventType();
      while (typ != XmlPullParser.END_DOCUMENT)  {
        String sTag = parser.getName();
        switch (typ) {
          case XmlPullParser.START_TAG:
            if (sTag.equalsIgnoreCase("quote"))  {
```

```
            akcie = new Akcie();
          }
          break;

        case XmlPullParser.TEXT:
          text = parser.getText();
          break;

        case XmlPullParser.END_TAG:
          if (sTag.equalsIgnoreCase("Change")) {
            akcie.setChange(text);
          }
          else if (sTag.equalsIgnoreCase("DaysLow"))  {
            akcie.setDaysLow(text);
          }
          else if (sTag.equalsIgnoreCase("DaysHigh"))  {
            akcie.setDaysHigh(text);
          } else if (sTag.equalsIgnoreCase("Symbol"))  {
            akcie.setSymbol(text);
          }
          break;
        default: break;
      }
      typ = parser.next();
    }
  }
  catch (Exception e)  {
    e.printStackTrace();
  }
}

private class NactiUdajeAsyncTask extends AsyncTask<String,
  String, String> {
  @Override
  protected String doInBackground(String... params)
  {
    try
    {
      XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
      factory.setNamespaceAware(true);
      XmlPullParser parser = factory.newPullParser();
      parser.setInput(new InputStreamReader(getUrlData(sURL)));
      parsujXML(parser);
    }

    catch (XmlPullParserException e)  {
      e.printStackTrace();
    } catch (URISyntaxException e)    {
      e.printStackTrace();
    } catch (IOException e)           {
      e.printStackTrace();
```

```
        }
        finally {}
        return null;
      }

      public InputStream getUrlData(String surl) throws URISyntaxException,
      IOException
      {
        URL url = new URL(surl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setRequestMethod("GET");
        return conn.getInputStream();
      }

      protected void onPostExecute(String result) {
        TextView akce = (TextView)findViewById(R.id.akce);
        TextView min = (TextView)findViewById(R.id.min);
        TextView max = (TextView)findViewById(R.id.max);

        akce.setText(akcie.getSymbol() + " : " + akcie.getChange());
        min.setText(akcie.getDaysLow());
        max.setText(akcie.getDaysHigh());
      }
    }
}
```

Aby se vaše aplikace mohla připojovat k údajům z internetu, musíte v aplikačním manifestu v souboru *AndroidManifest.xml* deklarovat příslušné oprávnění:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Po doplnění oprávnění do aplikačního manifestu můžete aplikaci vyzkoušet.

Statická data ve formátu XML

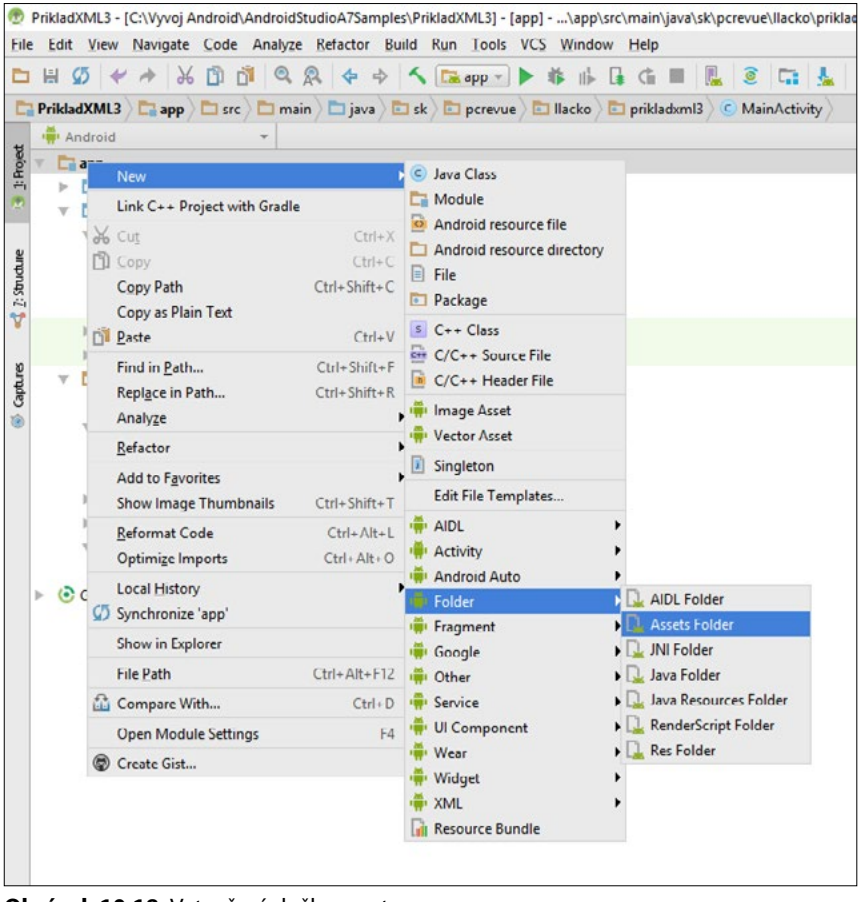
Pokud potřebujete využívat v aplikaci statická data, například aby aplikace fungovala i v režimu offline, jednou z možností je uložit tyto statické údaje do dokumentu XML. Dokument je potřeba umístit do složky *assets*, případně *res/raw*. Pokud nastane potřeba aktualizace údajů, realizuje se to v rámci aktualizace aplikace.

Aplikační kód přistupuje k údajům v dokumentu XML prostřednictvím objektu `Resources` voláním metod `getResources()` a `openRawResources()`. Poslední ze jmenovaných metod vrátí objekt typu `InputStream`, který umožňuje údaje z dokumentu XML načítat, ale neumožňuje je měnit.

Vytvořte projekt aplikace pro Android a ve složce *assets* vytvořte soubor `data.xml`. Složku vytvoříte pomocí místní nabídky a volby **New** → **Folder** → **Assets Folder**.

```
<?xml version="1.0" encoding="UTF-8"?>
  <prezidenti>
    <prezident>
      <id>1</id>
      <meno>George Washington</meno>
      <od>1789</od>
```

```
    <do>1797</do>
  </prezident>
  <prezident>
    <id>2</id>
    <meno>John Adams</meno>
    <od>1797</od>
    <do>1801</do>
  </prezident>
  ...
</prezidenti>
```



Obrázek 10.18: Vytvoření složky assets

Na zobrazení údajů z dokumentu XML využijeme prvek `ListView`.

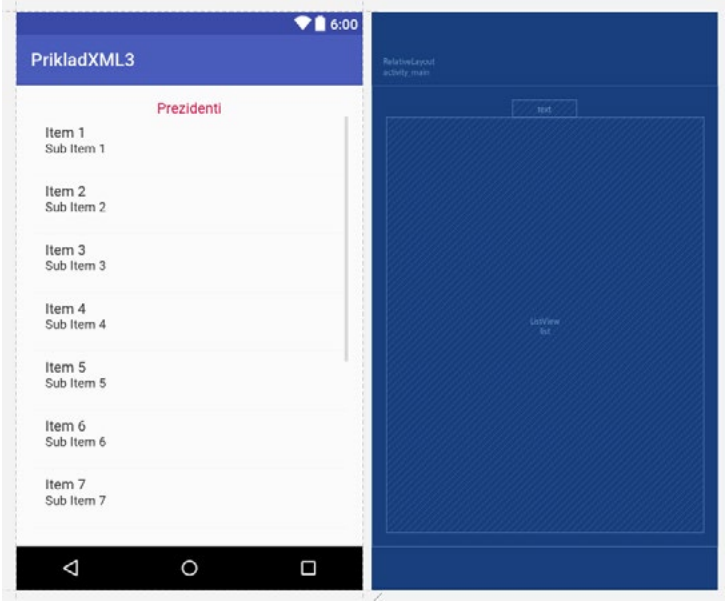
Návrh uživatelského rozhraní v souboru *main_activity.xml*:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
```

```
    tools:context="${packageName}.${activityClass}" >

    <TextView
      android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:text="@string/prezidenti"
      android:textColor="#CC0033"
      android:textSize="16sp" />

    <ListView
      android:id="@+id/list"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_below="@id/text" />
</RelativeLayout>
```



Obrázek 10.19: Návrh uživatelského rozhraní

Ještě je potřeba navrhnout layout položek prvku `ListView`, v kterém definujete, jak budou načítané údaje pro každou položku zobrazeny.

Návrh Layoutu je v souboru *list_item.xml*:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dp"
  android:textSize="16sp" />
```

Aplikační kód

Podobně jako v předchozím příkladu vytvoříme i v tomto na zapouzdření načítaných údajů třídu. Instance třídy budou v objektu typu `ArrayList`. V kódu třídy `Prezidenti` budou jen atributy pro hodnoty, které potřebujeme načítat z XML, a metody k nastavení atributu a získání jeho hodnoty. Postup, jak tyto metody ve vývojovém prostředí Android Studio vygenerovat pro jednotlivé atributy automaticky, je popsán v předchozím příkladu.

Vytvořte třídu *PrezidentiUSA.java*. Pro úsporu místa jsou ve výpisu metody pro nastavení atributu (setter) a získání jeho hodnoty (getter) pouze pro jeden atribut.

Na formátování údajů pro výpis dobře poslouží překrytí metody `toString()` kódem, který vytvoří textový řetězec z hodnot atributů včetně kódu `\n`, což je kód pro LF (Line Feed), tedy přechod na nový řádek.

```
public class PrezidentiUSA {
  private String id;
  private String meno;
  private String odRok;
  private String doRok;

  public String getId(){
    return id;
  }
  ...

  @Override
  public String toString() {
     return id + ": " + meno + "\n" + odRok + "-"  + doRok;
  }
}
```

Jelikož v tomto případě budeme načítat lokální soubor, který si aplikace „nese s sebou" ve složce *assets*, a známe i velikost dokumentu, jelikož prezidentů USA bylo méně než 50, nepotřebujeme realizovat čtení dokumentu ani jeho parsování v asynchronní třídě. Můžeme to pokojně realizovat v hlavním vlákně. Aby byl kód aplikace přehledně strukturovaný, budeme parsování realizovat v třídě `XMLPullParserHandler`.

Kód třídy *XMLPullParserHandler.java*:

```
public class XMLPullParserHandler {
  List<PrezidentiUSA> prezidenti;
  private PrezidentiUSA prezident;
  private String text;

  public XMLPullParserHandler()  {
    prezidenti = new ArrayList<PrezidentiUSA>();
  }

  public List<PrezidentiUSA> dajPrezidentov()  {
    return prezidenti;
  }

  public List<PrezidentiUSA> parse(InputStream is)  {
```

```
    XmlPullParserFactory factory = null;
    XmlPullParser parser = null;
    try {
      factory = XmlPullParserFactory.newInstance();
      factory.setNamespaceAware(true);
      parser = factory.newPullParser();
      parser.setInput(is, null);

      int typ = parser.getEventType();
      while (typ != XmlPullParser.END_DOCUMENT)    {
        String tagname = parser.getName();
        switch (typ) {
          case XmlPullParser.START_TAG:
            if (tagname.equalsIgnoreCase("prezident")) {
              prezident = new PrezidentiUSA();
            }
            break;

          case XmlPullParser.TEXT:
            text = parser.getText();
            break;

          case XmlPullParser.END_TAG:
            if (tagname.equalsIgnoreCase("prezident")) {
              prezidenti.add(prezident);
            }
            else if (tagname.equalsIgnoreCase("id"))   {
              prezident.setId(text); ;
            }
            else if (tagname.equalsIgnoreCase("meno")) {
              prezident.setMeno(text);
            }
            else if (tagname.equalsIgnoreCase("od"))   {
              prezident.setOdRok(text);
            } else if (tagname.equalsIgnoreCase("do"))  {
              prezident.setDoRok(text);
            }
            break;

          default: break;
        }
        typ = parser.next();
      }
    }
    catch (XmlPullParserException e)   {
      e.printStackTrace();
    }
    catch (IOException e)   {
      e.printStackTrace();
    }
    return prezidenti;
  }
}
```

Jelikož jsme na parsování dokumentu XML vytvořili samostatnou třídu, kód hlavní aktivity bude jednoduchý a přehledný.

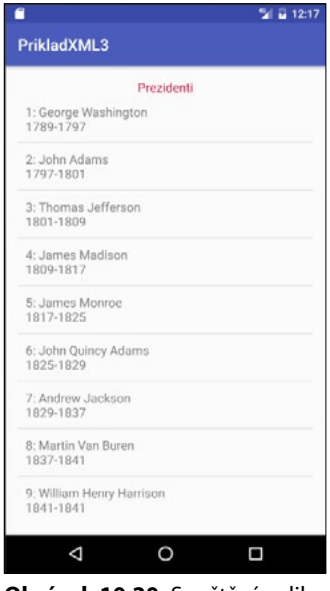
Kód v souboru *main_activity.xml*:

```
public class MainActivity extends Activity
{
  ListView listView;

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    listView = (ListView) findViewById(R.id.list);

    List<PrezidentUSA> prezidenti = null;
    try
    {
      XMLPullParserHandler parser = new XMLPullParserHandler();
      prezidenti = parser.parse(getAssets().open("data.xml"));
      ArrayAdapter<PrezidentUSA> adapter =
        new ArrayAdapter<PrezidentUSA>(this,R.layout.list_item, prezidenti);
      listView.setAdapter(adapter);
    } catch (IOException e)
    {
      e.printStackTrace();
    }
  }
}
```



Obrázek 10.20: Spuštění aplikace

KAPITOLA

11

Grafika

Grafika je jedním z nejdůležitějších designových i výrazových prostředků mobilních aplikací. Moderní zařízení využívající platformu Android disponují výkonnými procesory, hardwarově akcelerovanou grafikou a displeji s vysokým rozlišením, takže ani náročná 2D, a dokonce ani 3D grafika pro ně nepředstavuje problém.

Na vykreslení grafiky má aplikace pro Android několik možností na zobrazení statických nebo dynamicky se měnících grafických prvků.

Grafické prvky je možné vykreslovat buď přes `View`, přes nebo `Canvas`. Vykreslování grafických objektů prostřednictvím objektu `View` je jednodušší, ale méně flexibilní. Využívá se tehdy, když se nakreslené grafické prvky aktualizují zřídka, případně vůbec. Vykreslování s využitím objektu `Canvas` je složitější, ovšem mnohem flexibilnější a výkonnější. Dá se využít i k časté aktualizaci grafických prvků.

Jak vyplývá z názvu složky *Drawable*, je určená pro objekty, které je možné vykreslit, tedy obrázky, tvary a podobně.

Statickou grafiku definovanou v souborech XML můžete vytvářet pomocí tříd `ColorDrawable`, `BitmapDrawable`, `ShapeDrawable` a `GradientDrawable`. Soubory XML s definicí statické grafiky je potřeba umístit do složky *res/drawable*. Soubory se identifikují pomocí jejich názvu před příponou xml. Tento název musí být tvořen malými písmeny. Aby bylo možné pokrýt co nejvíc rozlišení displeje, využívají se zpravidla složky se specifikací rozlišení v modifikátoru, který je součástí názvu.

- *res/drawable-hdpi*
- *res/drawable-ldpi*
- *res/drawable-mdpi*
- *res/drawable-xhdpi*
- *res/drawable-xxhdpi*

Analogicky jako při vykreslování prvků uživatelského rozhraní je možné i grafické prvky vykreslovat přes definice v souborech XML nebo programově.

Příklady v této kapitole budou postupně ilustrovat možnosti zobrazení obrázků a grafických tvarů jednak prostřednictvím definice XML, jednak programově v kódu Java. Ukážeme načítání a zobrazení obrázku z internetu.

Témata kapitoly:

- **Zobrazení obrázku přes návrh XML**
- **Zobrazení obrázku programově**
- **Zobrazení obrázku z internetu**
- **Základní grafické tvary – staticky**
- **Základní grafické tvary – dynamicky**
- **Vykreslování na Canvas**
- **Kreslení dotykem na Canvas**
- **Dynamické vykreslování na Canvas s využitím View**
- **Dynamické vykreslování na Canvas s využitím SurfaceView**
- **Animace**
- **Animace prvků uživatelského rozhraní**

Zobrazení obrázku přes návrh XML

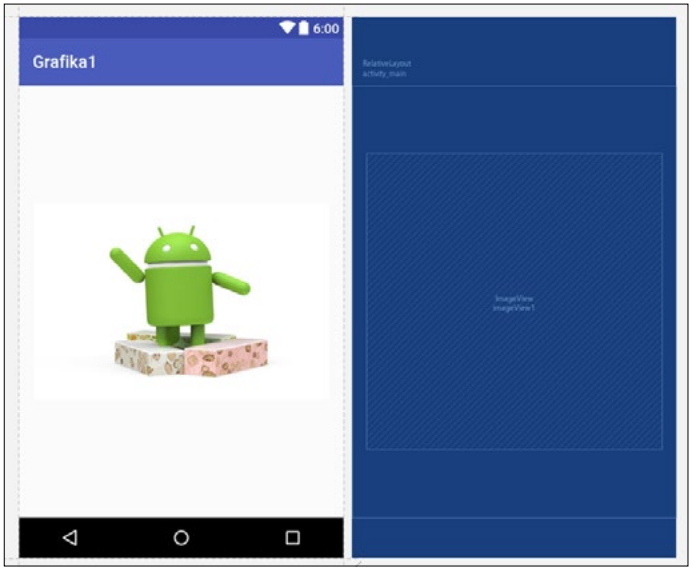
V prvním triviálním příkladu ukážeme zobrazení obrázku. Příklad má dvě varianty. V první zobrazíme obrázek pomocí prvku `View` definovaného v souboru XML. V druhé variantě zobrazíme obrázek pomocí kódu Java.

K zobrazení obrázku přidáme v návrhovém zobrazení do vizuálního kontejneru `RelativeLayout` prvek `ImageView`.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.example.llacko.grafika1.MainActivity">

  <ImageView
    android:id="@+id/imageView1"
    android:layout_width="350dp"
    android:layout_height="350dp"
    android:layout_centerInParent="true"
    android:contentDescription="Logo Android 7.0"
    android:src="@drawable/nougat" />
</RelativeLayout>
```

Obrázek přidejte do složky *res/drawable*.



Obrázek 11.1: Přidání obrázku do layoutu hlavní aktivity

Podobným jednoduchým postupem můžete zobrazit obrázek na pozadí. Obrázky je potřeba umístit také do adresáře *res/drawable*. Odkaz v souboru *main.xml* bude ve stejném tvaru jako odkaz na xml soubor s definicí grafiky, tedy název souboru s obrázkem bez přípony malými písmeny:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
  <View
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/obrazok"/>
</LinearLayout>
```

Zobrazení obrázku programově

Pomocí metody `setContentView()` definujeme připojení na definiční soubor XML, který však v tomto případě obsahuje jen kontejnerový prvek `RelativeLayout` a nic v něm. Aby bylo možné do `RelativeLayout` přidávat prvky programově, musíte mu přiřadit identifikátor. Android Studio mu přiřadí identifikátor `android:id="@+id/activity_main"`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.grafika1.MainActivity">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!" />

</RelativeLayout>
```

Pro prvek `ImageView` nastavíme bitmapový obrázek, který se bude vykreslovat. V našem případě je uložen ve formátu PNG. Kód pokračuje nastavením vlastností – těch samých, jaké jsme nastavovali v definici XML.

Výška a šířka obrázku nebude uložena přímo v kódu, ale v souboru *dimens.xml* ve složce *res\values*. Je to doporučený postup. Kdybyste se rozhodli později změnit velikost zobrazeného obrázku, nemusíte zasahovat do kódu.

```
<resources>
  <!-- Default screen margins, per the Android Design guidelines. -->
  <dimen name="activity_horizontal_margin">16dp</dimen>
  <dimen name="activity_vertical_margin">16dp</dimen>
  <dimen name="image_width">250dp</dimen>
  <dimen name="image_height">250dp</dimen>
</resources>
```

Výška, šířka a informace o vycentrování obrázku se nastaví přes objekt `RelativeLayout.LayoutParams`. Nakonec přidáme objekt `View` do vizuálního kontejneru `RelativeLayout`.

```
public class MainActivity extends AppCompatActivity {

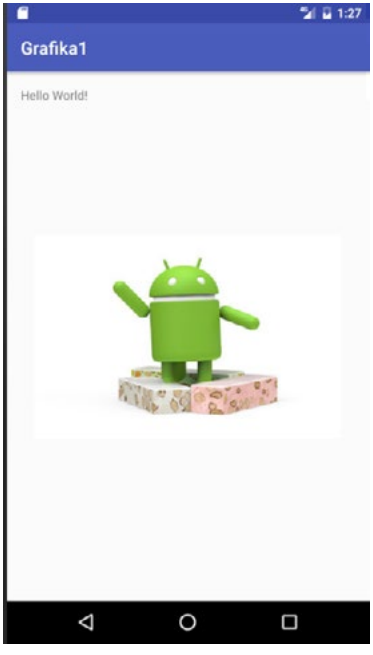
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    RelativeLayout relativeLayout = (RelativeLayout)
      findViewById(R.id.activity_main);

    ImageView imgView = new ImageView(getApplicationContext());
    imgView.setImageDrawable(getResources().getDrawable(R.drawable.nougat));

    int width = (int) getResources().getDimension(R.dimen.image_width);
    int height = (int) getResources().getDimension(R.dimen.image_height);

    RelativeLayout.LayoutParams params =
      new RelativeLayout.LayoutParams(width, height);
    params.addRule(RelativeLayout.CENTER_IN_PARENT);
    imgView.setLayoutParams(params);
    relativeLayout.addView(imgView);
  }
}
```



Obrázek 11.2: Zobrazení obrázku pomocí kódu

Zobrazení obrázku z internetu

Zobrazování obrázků je jednou z nejčastěji implementovaných funkcí v mobilních aplikacích, ať už jsou to aplikace s nabídkou zboží, aplikace zobrazující fotografie a podobně. Jak takovýto obrázek na vhodném místě a vhodným způsobem zobrazit v aplikaci pro Android?

Rozbor řešení

Naší úlohou je zobrazení obrázku z webu. Vlastní uložení obrázku na webovém serveru není námětem této publikace, proto předpokládáme, že obrázek už na webu uložený je. Pro cvičnou aplikaci postačí získat adresu URL libovolného obrázku z libovolné webové stránky. Na obrázek v prohlížeči klepněte prvým tlačítkem myši a zvolte **Vlastnosti**. Ze zobrazeného dialogu zkopírujte adresu URL a použijte ji v příkladu.

Povolení přístupu k internetu

Po vytvoření projektu, dříve než se pustíte do návrhu uživatelského rozhraní, doporučujeme nastavit aplikaci oprávnění přistupovat k internetu v souboru *AndroidManifest.xml*. Pokud na tento krok zapomenete, budete se při spuštění aplikace zbytečně divit, proč aplikace nefunguje, přestože její kód vypadá úplně v pořádku. Do manifestu přidejte řádek se sekcí `uses-permission`:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Návrh uživatelského rozhraní

Uživatelské rozhraní miniaplikace tvoří jediný prvek `ImageView`:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.example.llacko.grafika3.MainActivity">

  <ImageView
    android:id="@+id/imageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentTop="true"
    android:contentDescription="N-logo"/>
</RelativeLayout>
```

Pokud byste potřebovali zobrazit obrázek ze zdrojů, nastavili byste hodnotu atributu:

```
android:src="@drawable/obr"
```

Aplikační kód

Načítání čehokoliv z webu může trvat déle a v našem případě může nastat u některého uživatele aplikace kombinace pomalého internetového připojení a obrázků s vysokým rozlišením. Proto se tyto záležitosti neřeší v hlavním vlákně, ale v asynchronní třídě `NactiObrazek` v metodě `doInBackground()` na pozadí, aby během nahrávání obrázku nebylo blokované uživatelské rozhraní.

Po úspěšném načítání se obrázek přiřadí k prvku `ImageView` v metodě `onPostExecute()`:

```
public class MainActivity extends AppCompatActivity {

  public static final String sURL = "http://adresa.../image.png";
  ImageView imageView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    imageView = (ImageView) findViewById(R.id.imageView1);
    WebovyObrazek uloha = new WebovyObrazek ();
    uloha.execute(new String[] { sURL });
  }

  private class WebovyObrazek extends AsyncTask<String, Void, Bitmap> {
    @Override
    protected Bitmap doInBackground(String... urls) {
      Bitmap bmp = null;
      for (String url : urls) {
        bmp = nactiBmp(url);
      }
      return bmp;
    }

    @Override
    protected void onPostExecute(Bitmap result){
      imageView.setImageBitmap(result);
    }

    private Bitmap nactiBmp(String url){
      Bitmap bmp = null;
      InputStream stream = null;
      BitmapFactory.Options bmOptions = new BitmapFactory.Options();
      bmOptions.inSampleSize = 1;

      try {
        stream = vytvorStream(url);
        bmp = BitmapFactory.decodeStream(stream, null, bmOptions);
        stream.close();
      }
      catch (IOException e1) {
        e1.printStackTrace();
      }
```

```
        return bmp;
      }

      private InputStream vytvorStream(String urlString) throws IOException {
        InputStream stream = null;
        URL url = new URL(urlString);
        URLConnection connection = url.openConnection();

        try {
          HttpURLConnection httpCon = (HttpURLConnection) connection;
          httpCon.setRequestMethod("GET");
          httpCon.connect();

          if (httpCon.getResponseCode() == HttpURLConnection.HTTP_OK) {
            stream = httpCon.getInputStream();
          }
        } catch (Exception ex){
          ex.printStackTrace();
        }
        return stream;
      }
    }
  }
```

Základní grafické tvary – staticky

Základní grafické tvary jsou reprezentovány třídami odvozenými od třídy `Shape`. Určitě využijete `PathShape` na vykreslení segmentů a křivek, `RectShape` na vykreslení obdélníků a `OvalShape` na vykreslení kruhů a elips.

Postup vytvoření „grafiky XML" je jednoduchý. Do adresáře *drawable* umístěte soubor XML a vhodně jej pojmenujte. V našem jednoduchém příkladu chceme vykreslit zaoblené orámování displeje, proto jsme vybrali názvy *rect1.xml* a *rect2.xml*. Kořenovým (root) elementem bude `shape`. V souborech jsou definovány grafické prvky, v tomto případě prvek `rectangle` (obdélník) – barevný se zaoblenými rohy. Uvádíme jen obsah definičního souboru prvního obdélníka, druhý se liší jen barvou plochy a orámování:

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle">
  <solid
    android:color="#0000ff"/>
  <stroke
    android:width="5dp"
    android:color="#00FF00"/>
  <corners
    android:radius="30dp" />
</shape>
```

V definičním souboru XML hlavní aktivity vložíme grafické prvky do layoutu jako `ImageView`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/main_window"
  android:layout_width="match_parent"
```

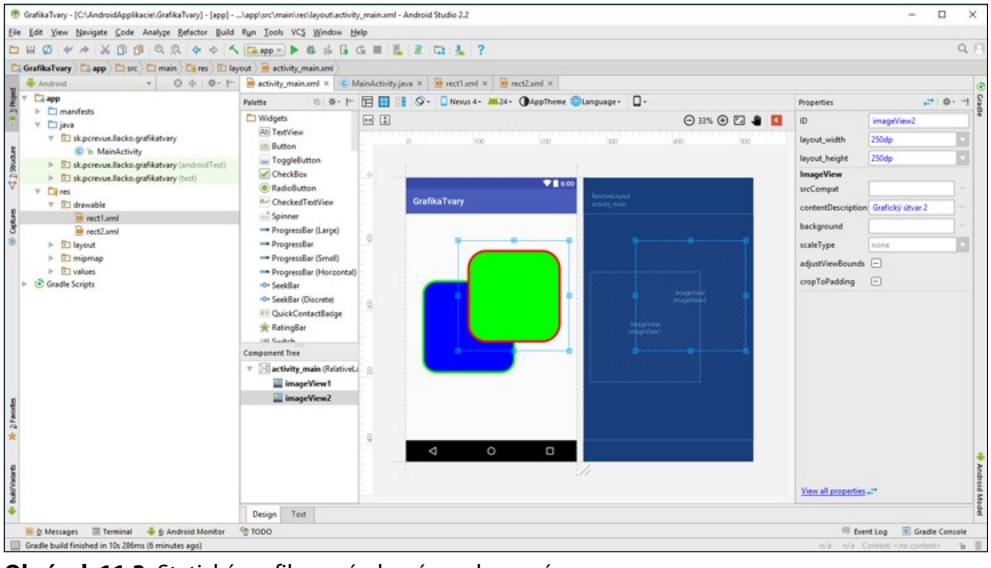
```
    android:layout_height="match_parent" >

    <ImageView
      android:id="@+id/imageView1"
      android:layout_width="250dp"
      android:layout_height="250dp"
      android:layout_alignParentLeft="true"
      android:layout_centerVertical="true"
      android:contentDescription="@string/rect1_desc"
      android:padding="20dp"
      android:src="@drawable/rect1" />

    <ImageView
      android:id="@+id/imageView2"
      android:layout_width="250dp"
      android:layout_height="250dp"
      android:layout_alignParentRight="true"
      android:layout_alignParentTop="true"
      android:layout_marginTop="45dp"
      android:contentDescription="@string/rect2_desc"
      android:padding="20dp"
      android:src="@drawable/rect2" />

</RelativeLayout>
```

V takto jednoduchém ilustračním příkladu nemusíte aplikaci ani spouštět, stačí se přepnout v definičním souboru XML hlavní aktivity do návrhového zobrazení a přesvědčit se, že se na ploše aktivity zobrazily dva částečně se překrývající obdélníky orámované čarou jiné barvy se zaoblenými rohy.



Obrázek 11.3: Statická grafika v návrhovém zobrazení

Základní grafické tvary – dynamicky

Zatímco se statická grafika definovaná v kódu XML hodí spíše na vykreslování pozadí, na vykreslování dynamické grafiky, ať už jednoduchých her, diagramů, schémat procesů a podobně, se používá aplikační kód. V druhé části příkladu tedy vykreslíme grafické tvary programově. Abychom představili více tvarů, namísto obdélníků vykreslíme barevné kruhy, tj. objekty `OvalShape`. Definiční soubor XML layoutu hlavní aktivity obsahuje jen kontejnerový prvek `RelativeLayout`. Aby bylo možné do něj přidávat prvky, musíte mu přiřadit identifikátor.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/activity_main"
  tools:context="${relativePackage}.${activityClass}" >

</RelativeLayout>
```

V souboru *dimens.xml* ve složce *res\values* je definovaná výška a šířka obrázků:

```
<resources>
  <dimen name="image_width">300dp</dimen>
  <dimen name="image_height">300dp</dimen>
  <dimen name="padding">3dp</dimen>
</resources>
```

Celé vykreslení obou prvků se odehrává v kódu hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  int alpha = 127;

  @Override
  public void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    int width = (int) getResources().getDimension(R.dimen.image_width);
    int height = (int) getResources().getDimension(R.dimen.image_height);
    int padding = (int) getResources().getDimension(R.dimen.padding);

    RelativeLayout rl = (RelativeLayout) findViewById(R.id.kontejner);

    //prvny modry kruh...
    ShapeDrawable blueShape = new ShapeDrawable(new OvalShape());
    blueShape.getPaint().setColor(Color.BLUE);
    blueShape.setIntrinsicHeight(height);
    blueShape.setIntrinsicWidth(width);
    blueShape.setAlpha(alpha);

    //... do ImageView
    ImageView blueView = new ImageView(getApplicationContext());
    blueView.setImageDrawable(blueShape);
```

```
        blueView.setPadding(padding, padding, padding, padding);

        // ... do RelativeLayout
        RelativeLayout.LayoutParams blueParams =
        new RelativeLayout.LayoutParams(height, width);
        blueParams.addRule(RelativeLayout.CENTER_VERTICAL);
        blueParams.addRule(RelativeLayout.ALIGN_PARENT_LEFT);
        blueView.setLayoutParams(blueParams);
        rl.addView(blueView);

        // druhy zeleny kruh ...
        ShapeDrawable greenShape = new ShapeDrawable(new OvalShape());
        greenShape.getPaint().setColor(Color.GREEN);
        greenShape.setIntrinsicHeight(height);
        greenShape.setIntrinsicWidth(width);
        greenShape.setAlpha(alpha);

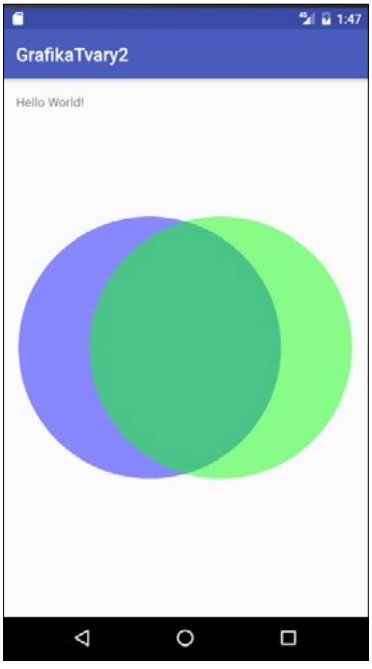
        // ... do ImageView
        ImageView greenView = new ImageView(getApplicationContext());
        greenView.setImageDrawable(greenShape);
        greenView.setPadding(padding, padding, padding, padding);

        // ... do RelativeLayout
        RelativeLayout.LayoutParams greenParams =
          new RelativeLayout.LayoutParams(height, width);
        greenParams.addRule(RelativeLayout.CENTER_VERTICAL);
        greenParams.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);

        greenView.setLayoutParams(greenParams);

        rl.addView(greenView);
      }
    }
```

V kódu získáme referenci na kontejner `RelativeLayout`, umožňující relativní uspořádání prvků. V proceduře `onCreate()` se postupně vytvoří dva objekty `OvalShape` včetně nastavení barvy, výšky a šířky. Abychom ilustrovali možnost překrývání a s ním spojeného míchání barev v místě, kde se obrazce překrývají, definujeme i parametr `alpha` – transparentnost. Hodnotu transparentnosti nastavíme na 127, tj. přibližně na polovinu.



Obrázek 11.4: Vykreslení grafických tvarů programovým kódem

Vykreslování na Canvas

Pokud chcete kreslit složitější grafiku, případně grafiku, která se často aktualizuje, využijete s výhodou kreslení na `Canvas`. Objekt `Canvas` znamená v praxi přesně to, co jeho překlad do češtiny – malířské plátno, které slouží jako podklad na vykreslování.

POZNÁMKA

`Canvas` umožňuje vykreslování objektů do souřadnicového systému.

Kreslení na plátno (canvas) s následným zobrazením grafiky na obrazovce zařízení je principiálně možné realizovat dvěma způsoby:

- **Využitím existujícího objektu View** – v případě, že chcete doplnit statické grafické prvky k už existujícím prvkům prostřednictvím existujícího plátna.
- **Prostřednictvím nově vytvořeného plátna** – v případě dynamického obsahu, který má být pravidelně měněn a překreslován, přičemž v tom samém vlákně voláme metodu `invalidate()` a v samostatném vlákně používáme management cestou třídy `SurfaceView`, která zpřístupňuje aktuální plochu cestou třídy `SurfaceHolder` (`getHolder()`, `SurfaceHolder.Callback`). Zpřístupnění plátna v rámci samostatného vlákna je v tomto případě řešeno metodou `lockCanvas()` a jeho odblokování po vykreslení všech požadovaných prvků metodou `unlockCanvasAndPost()`.

Nejčastěji budete v souvislosti s kreslením na `Canvas` využívat metody `drawText()`, `drawPoints()`, `drawColor()`, `drawOval()` a `drawBitmap()`.

Canvas umožňuje pokročilejší vykreslování. Můžete nastavit různé barvy a styly, tloušťku čar, velikost textu. Můžete nastavit různé metody grafické optimalizace, například antialiasing na vyhlazení obrazu, aby se odstranily zubaté šikmé čáry, a podobně. Pro naznačené účely budete používat metody setStrokeWidth(), setTextSize(), setColor(), či setAntiAlias().

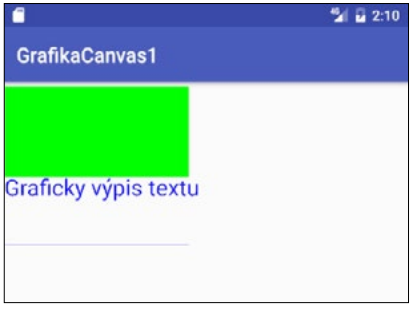
Vykreslování na Canvas budeme demonstrovat na triviálním příkladu, ve kterém vykreslíme obdélník, čáru a text.

Nejprve vytvoříme třídu Kreslenie.

```
public class Kreslenie extends View {
  public Kreslenie(Context context){super(context);}
  @SuppressLint("DrawAllocation") @Override
  protected void onDraw(Canvas canvas)  {
    Paint paint = new Paint();
    paint.setColor(Color.GREEN);
    canvas.drawRect(10, 10, 500, 250, paint);
    paint.setColor(Color.BLUE);
    paint.setTextSize((float) 60.0);
    canvas.drawText("Graficky výpis textu", 10, 300, paint);
    canvas.drawLine(10, 430, 500, 430, paint);
  }
}
```

V kódu hlavní aktivity nastavíme tuto třídu jako kontextové View pro vykreslování v metodě SetContentView(). Z uvedeného vyplývá, že návrh XML hlavní aktivity se vůbec nepoužije.

```
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(new Kreslenie(this));
  }
}
```



Obrázek 11.5: Vykreslení jednoduchých grafických tvarů na Canvas

Kreslení dotykem na Canvas

U přístrojů s dotykovým displejem se námět na další grafickou aplikaci plnou dynamiky nabízí sám – kreslení prstem po obrazovce. K tomuto účelu trochu přepracujeme třídu Kreslenie z předchozího příkladu.

Princip je jednoduchý. V obslužné proceduře onTouchEvent budeme snímat jako události dotyky (MotionEvent.ACTION_DOWN) a pohyby prstu (MotionEvent.ACTION_MOVE) a ukládat souřadnice, kde jste se displeje dotkli, do pole. Pro jednoduchost budeme serializovat, to znamená ukládat souřadnice x a y sériově do jednorozměrného pole. Následně vykreslíme čáru jako spojnici mezi posledními dvěma body. Všimněte si nastavení tloušťky čáry.

Nejprve přidejte referenci na snímání pohybu na dotykové obrazovce a pole objektů k ukládání souřadnic:

```
android.view.MotionEvent;
import java.util.ArrayList;
```

Po identifikaci události dotyku nebo posunu prstu se explicitně vyvolá událost invalidate(). V obsluze této události se zavolá metoda onDraw na překreslení obrazovky:

```
public class Kreslenie extends View {
  public Kreslenie(Context context)  {
    super(context);
  }

  ArrayList<Float> pt = new ArrayList<Float>();

  @Override
  public boolean onTouchEvent(MotionEvent event) {
    int e = event.getAction();
    if (e == MotionEvent.ACTION_DOWN || e == MotionEvent.ACTION_MOVE)  {
      pt.add(event.getX());  pt.add(event.getY());
      invalidate();
    }
    return true;
  }


  @Override
  protected void onDraw(Canvas canvas)  {
    Paint paint = new Paint();
    paint.setColor(Color.BLACK);

    paint.setStrokeWidth(10);
    for(int i =2; i < pt.size(); i+=2)  {
      canvas.drawLine(pt.get(i-2), pt.get(i-1),
        pt.get(i), pt.get(i+1) , paint);
    }
  }
}
```

V kódu hlavní aktivity nastavíme tuto třídu jako kontextové View pro vykreslování v metodě SetContentView(). Návrh XML hlavní aktivity se vůbec nepoužije.

```
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
```

```
        setContentView(new Kreslenie(this));
    }
}
```



Obrázek 11.6: Příklad aplikace na kreslení prstem

Samozřejmě i zde platí, že za málo námahy dostanete málo efektu. Aplikace má „drobné" vady, například pokud kreslíte více tvarů, spojí body, kde jste prst zdvihli a znovu položili.

Dynamické vykreslování na Canvas s využitím View

Vykreslování na `Canvas` můžete realizovat přes `View` nebo třídu odvozenou od `View` jako v předchozích příkladech. To v případě, že aktualizace grafiky budou méně časté. U často aktualizované grafiky je lepší použít specializovanou třídu `SurfaceView`.

Plátno je rozhraním mezi aktivitou a obrazovkou, resp. plochou (surface) zařízení, na kterou budou vykresleny grafické prvky. Prostřednictvím plátna jsou prováděny konkrétní příkazy určené na kreslení – tzv. rendering pixelů. V konečném důsledku jde o změnu barev pixelů obrazovky, čímž vytváříme požadované grafické tvary. Informace o barvě pixelů konkrétních grafických prvků jsou umísťovány do paměťového bloku objektu `Bitmap`, který musíme vytvořit spolu s objektem `Canvas`.

Námětem příkladu bude pohyb figurky po ploše. Vzhledem k tomu, že vykreslování přes `View` se hodí pro méně často aktualizovanou grafiku, mohli bychom takto realizovat například jednoduchou hru typu šachy. V reálné aplikaci by se figurka pohybovala po ploše znázorňující herní scénu, například po šachovnici.

Dříve než přistoupíte k tvorbě kódu na vykreslování, je potřeba realizovat dva přípravné kroky:

- Do adresáře *drawable* vložte obrázek nějaké figurky. Obrázek snadno získáte pomocí Googlu v sekci obrázky.
- V souboru *dimens.xml* ve složce *res\values* definujte výšku a šířku obrázku:

```
<resources>
  <dimen name="image_width">50dp</dimen>
  <dimen name="image_height">50dp</dimen>
</resources>
```

Vykreslování se realizuje v kódu hlavní aktivity. Figurka změní svou polohu v sekundových intervalech.

V metodě `onCreate()` vytvoříme instanci objektu `Bitmap` na zobrazení obrázku ze zdrojů. Vytvoříme třídu `HraView`, odvozenou od `View`, a přidáme ji do layoutu hlavní aktivity.

```
rl.addView(hraView);
```

POZNÁMKA

Všimněte si v konstruktoru třídy `HraView` způsobu získání informace o rozlišení obrazovky v horizontálním i vertikálním směru.

Pohyb figurky bude realizován jako postupné vykreslování, přičemž jednotlivé iterace se realizují ve vlákně. V každé fázi pohybu se předtím vykreslený obrázek vymaže a nakreslí na nové pozici. Pro jednoduchost měníme souřadnice vykreslování přičtením konstanty. Figurka se samozřejmě vykreslí pouze v případě, že její souřadnice nejsou mimo zobrazovací plochu. Taktování po sekundách se realizuje pomocí metody:

```
Thread.sleep(1000);
```

Na vykreslování se využívá překrytá metoda `onDraw()` a v ní `canvas.drawBitmap()`.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  @Override
  public void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    final RelativeLayout rl = (RelativeLayout)
      findViewById(R.id.activity_main);
    final Bitmap bmp = BitmapFactory.decodeResource(getResources(),
      R.drawable.droid);
    final HraView hraView = new HraView(getApplicationContext(),bmp);
      rl.addView(hraView);

    new Thread(new Runnable()  {
      @Override
      public void run()  {
        while (hraView.posun())   {
          hraView.postInvalidate();
          try {Thread.sleep(1000);}
          catch (InterruptedException e)  {
            Log.i("KreslenieCanvas", "CHYBA Sleep");}
        }
      }
    }).start();
  }

  private class Poloha  {
    float sX, sY;

    public Poloha(float x, float y)   {
```

```
    sX = x;   sY = y;
  }
  synchronized Poloha posun(Poloha dxdy) {
    return new Poloha(sX + dxdy.sX , sY + dxdy.sY);
  }
  synchronized Poloha dejSouradnice()  {
    return new Poloha(sX, sY);
  }
}

private class HraView extends View   {
  final private Bitmap mBmp;
  private Poloha mAktualni;
  final private Poloha mDelta;

  final private DisplayMetrics mMetrika;
  final private int mWidth;
  final private int mHeight;
  final private int mOld, mNew;
  final private Paint mPainter = new Paint();

  public HraView(Context context, Bitmap bitmap)  {
    super(context);

    mOld = (int) getResources().getDimension(R.dimen.image_height);
    this.mBmp = Bitmap.createScaledBitmap(bitmap, mOld, mOld, false);
    mNew = mOld + 10;

    mMetrika = new DisplayMetrics();
    MainActivity.this.getWindowManager().getDefaultDisplay()
      .getMetrics(mMetrika);
    mWidth = mMetrika.widthPixels;
    mHeight = mMetrika.heightPixels;

    float x = 10; float y = 10;
    mAktualni = new Poloha(x, y);

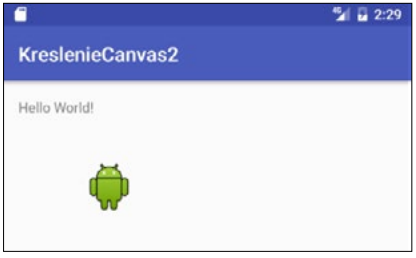
    float dy = y+1;
    float dx = x+1;
    mDelta = new Poloha(dx, dy);

    mPainter.setAntiAlias(true);

  }

  @Override
  protected void onDraw(Canvas canvas)  {
    Poloha pp = mAktualni.dejSouradnice();
    canvas.drawBitmap(mBmp, pp.sX, pp.sY, mPainter);
  }
```

```
    protected boolean posun()  {
      mAktualni = mAktualni.posun(mDelta);
      if (mAktualni.sY < 0 - mNew
        || mAktualni.sY > mHeight + mNew
        || mAktualni.sX < 0 - mNew
        || mAktualni.sX > mWidth + mNew)
        return false;
      else return true;
    }
  }
}
```



Obrázek 11.7: Dynamické vykreslování

Dynamické vykreslování na Canvas s využitím SurfaceView

Třída `SurfaceView` poskytuje specializovanou kreslicí plochu umožňující i zobrazení hierarchie či překrývání objektů, tzv. z-order. Spravuje vykreslování na úrovni vrstvy `Surface`. `SurfaceView` využívá vlákno mimo hlavního vlákna uživatelského rozhraní.

Podobně jako v předchozím příkladu je námětem příkladu pohyb figurky po ploše. Vzhledem k tomu, že se vykreslování přes `SurfaceView` hodí i pro dynamičtěji aktualizovanou grafiku, mohli bychom takto realizovat například hru typu Sokoban. V příkladu budeme posouvat figurku a zároveň ji rotovat, čímž vytvoříme jakousi manuálně překreslovanou animaci.

POZNÁMKA

V následující části se budeme animaci věnovat podrobněji a ukážeme jednodušší metody, jak pohybovat grafickým objektem.

Pokud chcete definovat `SurfaceView`, budete muset vytvořit vlastní třídu, která je podtřídou `SurfaceView`, a implementovat rozhraní `SurfaceHolder.Callback`.

Ve fázi nastavení `SurfaceView` je potřeba pomocí metody `SurfaceViews.getHolder()` získat objekt `SurfaceHolder` a pomocí metody `SurfaceHolder.addCallback` registrovat `SurvaceView` pro zpětné volání.

```
private class HraView extends SurfaceView implements SurfaceHolder.Callback {
  mSurfaceHolder = getHolder();
  mSurfaceHolder.addCallback(this);
}
```

Po vytvoření objektu SurfaceView se volá metoda surfaceCreated(), ve které je potřeba definovat vlákno. V průběhu vykonávání metody surfaceCreated() nelze kreslit.

```
public void surfaceCreated(SurfaceHolder holder) {
  mThread = new Thread(new Runnable() {
    public void run() {
    ...
    });
    mThread.start();
  }
}
```

Metoda surfaceChanged() se volá při každé strukturální změně Surface pro zobrazování. Metoda surfaceDestroyed() se volá před odstraněním objektu Surface. Po návratu z této metody už není možné realizovat operace vykreslování.

POZNÁMKA

SurfaceView callback metody jsou volány z hlavního vlákna aplikace, ne z vlákna SurfaceView. Proto musíte zabezpečit synchronizaci údajů využívaných oběma vlákny.

K tomu slouží zámek – metoda lockCanvas(). Následně realizujete vykreslování třeba pomocí metody drawBitmap(). Nakonec pomocí metody unlockCanvasAndPost() odemkněte Canvas a dojde k aktualizaci zobrazení.

```
canvas = mSurfaceHolder.lockCanvas();
if (null != canvas) {
  kresliFigurku(canvas);
  mSurfaceHolder.unlockCanvasAndPost(canvas);
}
```

Kompletní kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    RelativeLayout rl = (RelativeLayout) findViewById(R.id.activity_main);
    final HraView bubbleView = new HraView(getApplicationContext(),
      BitmapFactory.decodeResource(getResources(), R.drawable.droid));
      rl.addView(bubbleView);
  }

  private class HraView extends SurfaceView implements
    SurfaceHolder.Callback   {
    private final Bitmap mBmp;
    private final int  mWidth, mHeight, mRozmer, mRozmerUpr;
    private final DisplayMetrics mDM;
    private float sX, sY, sDx, sDy, mRot;
    private final SurfaceHolder mSurfaceHolder;
```

```
private final Paint mPaint = new Paint();
private Thread mThread;

private static final int KROK = 1;
private static final float ROTACIA = 1.0f;

public HraView(Context context, Bitmap bitmap)     {
  super(context);

  mRozmer = (int) getResources().getDimension(
    R.dimen.image_height);
  this.mBmp = Bitmap.createScaledBitmap(bitmap, mRozmer,
    mRozmer, false);

  mRozmerUpr = mRozmer / 2;

  mDM = new DisplayMetrics();
  MainActivity.this.getWindowManager().getDefaultDisplay()
    .getMetrics(mDM);
  mWidth = mDM.widthPixels;
  mHeight = mDM.heightPixels;

  Random r = new Random();
  sX = (float) r.nextInt(mHeight);
  sY = (float) r.nextInt(mWidth);
  sDx = (float) r.nextInt(mHeight) / mHeight;
  sDx *= r.nextInt(2) == 1 ? KROK : -1 * KROK;
  sDy = (float) r.nextInt(mWidth) / mWidth;
  sDy *= r.nextInt(2) == 1 ? KROK : -1 * KROK;
  mRot = 1.0f;

  mPaint.setAntiAlias(true);
  mSurfaceHolder = getHolder();
  mSurfaceHolder.addCallback(this);
}

private void kresliFigurku(Canvas canvas)  {
  canvas.drawColor(Color.YELLOW);
  mRot += ROTACIA;
  canvas.rotate(mRot, sY + mRozmerUpr, sX
    + mRozmerUpr);
  canvas.drawBitmap(mBmp, sY, sX, mPaint);
}

private boolean posun()  {
  sX += sDx;
  sY += sDy;
  if (sX < 0 - mRozmer
    || sX > mHeight + mRozmer
    || sY < 0 - mRozmer
    || sY > mWidth + mRozmer)
```

```
        return false;
      else return true;
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
      int height) {}

    @Override
    public void surfaceCreated(SurfaceHolder holder) {
      mThread = new Thread(new Runnable()  {
        public void run() {
          Canvas canvas = null;
          while (!Thread.currentThread().isInterrupted() && posun())  {
            canvas = mSurfaceHolder.lockCanvas();
            if (null != canvas)  {
              kresliFigurku(canvas);
              mSurfaceHolder.unlockCanvasAndPost(canvas);
            }
          }
        }
      });
      mThread.start();
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
      if (null != mThread) mThread.interrupt();
    }
  }
}
```



Obrázek 11.8: Dynamické vykreslování s využitím SurfaceView

Animace

Animace umožňuje měnit grafický objekt, konkrétně parametry vykreslování, například velikost, barvu, přehlednost, umístění, orientaci a podobně. Změny se realizují dynamicky, postupně, v závislosti na čase. Android poskytuje několik různých metod animace implementovaných jako třídy.

Metody klasické animace

Prakticky od prvních verzí API podporuje Android jednoduché animace:

- `TransitionDrawable` – animace přechodu mezi dvěma zobrazeními. Realizuje se postupným stmíváním prvního a postupným rozjasňováním druhého zobrazení.
- `AnimationDrawable` – animace pomocí rychlého zobrazování série obrázků.
- `TweenAnimation` – vytváření animace změnou parametrů zobrazení.

POZNÁMKA

Tyto animace je možné aplikovat jen na objekty, které jsou potomky třídy `View`.

Příklad animace TransitionDrawable

V příkladu využijeme stejné objekty jako v příkladu na vykreslování základních grafických tvarů, tedy dvou obdélníků definovaných v souborech XML *rect1.xml* a *rect2.xml* umístěných ve složce *drawable*.

Uvádíme jen obsah definičního souboru prvního obdélníka, druhý se liší jen barvou plochy (`#ff00ff`).

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
  android:shape="rectangle" >
  <solid android:color="#0000ff" />
</shape>
```

Přechod mezi zobrazeními je definován v souboru *transformace.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:drawable="@drawable/rect1"
    android:right="100dp"/>
  <item android:drawable="@drawable/rect2"
    android:left="100dp"/>
</transition>
```

V definičním souboru XML hlavní aktivity vložíme grafické prvky do layoutu jako `ImageView`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/main_window"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <ImageView
    android:id="@+id/image_view"
    android:layout_width="match_parent"
    android:layout_height="250dp"
    android:contentDescription="@string/obrazek_popis"
    android:layout_centerVertical="true"/>
</RelativeLayout>
```

Animace v trvání dvou sekund je vytvořena a spuštěna v kódu hlavní aktivity. V metodě `onCreate` se načítá definiční soubor transformace *shape_transition.xml*:

```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState)    {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TransitionDrawable transition = (TransitionDrawable) getResources()
      .getDrawable(R.drawable.transformace);
    transition.setCrossFadeEnabled(true);
    ((ImageView) findViewById(R.id.image_view)).setImageDrawable(transition);
    transition.startTransition(2000);
  }
}
```

Po spuštění aplikace se vlevo zobrazí modrý obdélník, který postupně mizí, a vpravo se začíná zobrazovat fialový obdélník.

Příklad animace AnimationDrawable

Tato metoda animace se nejvíc podobá tvorbě klasických animovaných filmů, které se před érou pokročilé počítačové grafiky skládaly z jednotlivých obrázků, jež se potom snímaly na jednotlivá políčka filmového pásu.



Obrázek 11.9: Princip snímkové animace

`AnimationDrawable` zobrazuje postupně sérii objektů `drawable`, přičemž každý z objektů se zobrazuje ve stanovený čas.

Námětem příkladu bude zobrazení jednoduché animace. V našem případězachycuje, jak roste květ. Obrázky pro váš příklad můžete najít na webu nebo si je můžete nakreslit. V layoutu hlavní aktivity je prvek `ImageView` na zobrazení animace jako série obrázků.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/frame"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
```

```
  <ImageView
    android:id="@+id/animovany_obrazek"
    android:layout_width="300dp"
    android:layout_height="250dip"
    android:layout_gravity="center"
    android:layout_marginBottom="20dp"
    android:layout_marginTop="20dp"
    android:contentDescription="@string/obrazek_popis"
    android:scaleType="centerCrop" />
</LinearLayout>
```

Uložte snímky, které budou tvořit animaci, do adresáře *drawable*. V souboru *animace.xml* je ve složce *drawable* definice průběhu animace. Každý snímek tvořící animaci bude zobrazen po dobu 300 milisekund (výpis je zkrácený):

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
  android:oneshot="true" >
  <item
    android:drawable="@drawable/anim1"
    android:duration="300"/>
    ...
    android:drawable="@drawable/anim7"
    android:duration="300"/>
</animation-list>
```

Kód hlavní aktivity:

```
public class MainActivity extends Activity {
  private AnimationDrawable anim;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ImageView imageView = (ImageView) findViewById(R.id.animovany_obrazek);
    imageView.setBackgroundResource(R.drawable.animace);
    anim = (AnimationDrawable) imageView.getBackground();
  }

  @Override
  protected void onPause()  {
     super.onPause();
    if (anim.isRunning()) anim.stop();
  }

  @Override
  public void onWindowFocusChanged(boolean hasFocus)        {
    super.onWindowFocusChanged(hasFocus);
    if (hasFocus) anim.start();
    }
}
```

Příklad animace Tween

Animace tohoto typu umožňuje zobrazit `ImageView` a animovat jednu nebo několik jeho vlastností současně. Na příkladu grafického objektu budeme demonstrovat jeho postupné snižování průsvitnosti, které vizuálně působí, jako kdyby se obrázek vynořil z pozadí. Následně ukážeme pohyb objektu, změnu jeho velikosti. Animaci ukončí zmizení objektu, tedy zvyšování průsvitnosti. Budeme měnit i tempo časového průběhu animace.

V kódu XML layoutu hlavní aktivity je jen jeden prvek `ImageView`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.animaciatween.MainActivity">

  <ImageView
    android:id="@+id/obrazek"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:src="@drawable/droid"
    android:contentDescription="Androidko"
    android:visibility="invisible"/>
</RelativeLayout>
```

V souboru *dimens.xml* je uvedena definice posunu:

```
<resources>
  <dimen name="posun">100dp</dimen>
</resources>
```

Ve složce res vytvořte podsložku *anim* a v ní soubor *animacia.xml*. V krátkosti vysvětlíme jednotlivé fáze námi definované animace:

- V sekci `<alpha>` se postupně zmenšuje transparentnost objektu z hodnoty `0.0` – úplně průsvitné, na hodnotu `1.0` – neprůsvitné. Fáze trvá 2 000 milisekund.
- V sekci `<rotate>` se objekt otočí třikrát dokola, tj. třikrát o 360 stupňů kolem osy z, v průběhu tří sekund.
- V sekci `<translate>` se objekt plynule posune na novou pozici.
- V sekci `<scale>` dojde k zvětšení objektu na dvojnásobek.
- V (poslední) sekci `<alpha>` se objekt znovu stává průsvitným.

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
  android:shareInterpolator="false" >

  <alpha
    android:duration="2000"
    android:fromAlpha="0.0"
    android:interpolator="@android:anim/linear_interpolator"
    android:toAlpha="1.0" />

  <rotate
```

```
    android:duration="3000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/accelerate_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="3000"
    android:toDegrees="1080" />

  <translate
    android:duration="3000"
    android:fromXDelta="0"
    android:fromYDelta="0"
    android:interpolator="@android:anim/overshoot_interpolator"
    android:startOffset="7000"
    android:toXDelta="100"
    android:toYDelta="100" />

  <scale
    android:duration="3000"
    android:fromXScale="1"
    android:fromYScale="1"
    android:interpolator="@android:anim/anticipate_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="10000"
    android:toXScale="2"
    android:toYScale="2" />

  <alpha
    android:duration="2000"
    android:fromAlpha="1.0"
    android:interpolator="@android:anim/decelerate_interpolator"
    android:startOffset="13000"
    android:toAlpha="0.0" />

</set>
```

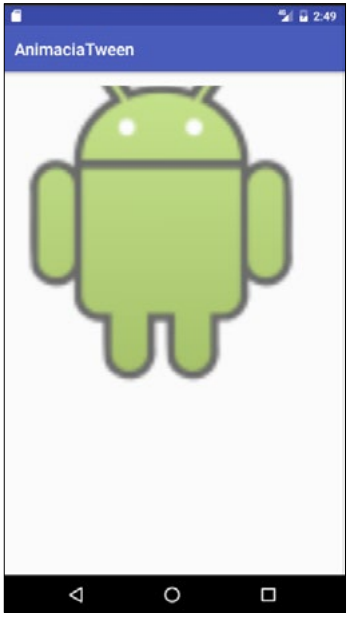
Animace je definovaná v kódu hlavní aktivity a spuštěna v metodě `onWindowFocusChanged()`.

```
public class MainActivity extends AppCompatActivity {

  private ImageView iw;
  private Animation anim;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    iw = (ImageView) findViewById(R.id.obrazek);
    anim = AnimationUtils.loadAnimation(this, R.anim.animacia);
  }
```

```
  @Override
  public void onWindowFocusChanged(boolean hasFocus) {
    super.onWindowFocusChanged(hasFocus);
    if (hasFocus) iw.startAnimation(anim);
  }
}
```



Obrázek 11.10: Test aplikace – fáze zvětšování objektu

Property Animation

Dosud prezentované animace jsou velmi jednoduše realizovatelné, mají ale svá omezení. Především jsou určeny jen pro 2D grafiku. Jedinou funkcionalitou, která mírně zasahuje do působnosti 3D, je rotace, jelikož objekt rotuje okolo osy z. Klasické animace ovlivňují pouze to, jak je prvek vykreslen.

Jednu z hlavních nevýhod klasické animace byste objevili, kdybyste chtěli snímat dotyky na pohybující se objekt, například byste takto klepáním prstu chtěli sestřelovat objekty. Neuspěli byste, jelikož během animace se jen průběžně vykresluje posun z původní pozice do nové. Skutečné místo, kde můžete na objekt klepnout, se nezmění, takže byste museli implementovat vlastní logiku na ovládání dotykem.

Od verze API 11 (Android 3.0 Honeycomb) je k dispozici nový typ animace – Property Animation, který je mnohem flexibilnější.

POZNÁMKA

Pro starší verze Androidu je k dispozici open-source knihovna `NineOldAndroids` (*http://nineoldandroids.com*), která zpřístupňuje funkcionalitu Property Animation.

Property Animation umožňuje animovat prakticky jakýkoliv objekt.

POZNÁMKA

Můžete definovat animaci, která dokáže měnit jakoukoliv vlastnost objektu v průběhu času, bez ohledu na to, zda se objekt kreslí na obrazovku nebo ne.

Fungování animace vysvětluje její název. Animace zavolá setter objektu a nastaví mu novou hodnotu zvoleného parametru získanou interpolací.

Můžete definovat následující vlastnosti animace:

- **Doba trvání** – můžete zadat dobu trvání animace. Předvolená délka je 300 ms.
- **Čas interpolace** – můžete definovat, jakým způsobem budou hodnoty parametru vypočítány. Algoritmus výpočtu musíte navrhnout jako funkci, kde vstupní veličinou je aktuálně uplynulý čas a výsledná hodnota je v intervalu 0–1, tedy matematicky zapsáno y = f(t), y<0;1>. Funkce mohou být lineární i nelineární.
- **Chování a počet opakování** – můžete určit, zda se animace bude nebo nebude opakovat, když dosáhne konce, a pokud ano, kolikrát. Můžete také určit, zda chcete animaci přehrát v opačném směru.
- **Seskupování** – můžete seskupit animace do logických množin, které jsou spouštěny společně nebo postupně nebo po uplynutí zadaného zpoždění.
- **Zpoždění obnovování** – můžete určit, jak často se bude aktualizovat zobrazení. Implicitně je nastavena aktualizace každých 10 ms, ale rychlost, s jakou vaše aplikace dokáže aktualizovat zobrazení, závisí v konečném důsledku na tom, jak rychle dokáže zařízení pracovat, tj. jak rychle jeho systém může obsluhovat základní časovač.

PropertyAnimation funguje na čtyřech pilířích:

- `ValueAnimator` je v principu časovač, který „tiká“ s nastavenou frekvencí.
- `TimeInterpolator` vypočítává hodnotu parametru podle vstupního parametru času y = f(t), y<0;1>.
- `TypeEvaluator` mapuje výsledek z interpolátoru na rozsah animace, matematicky vyjádřeno X0 + t*(x1 – x0). Pokud je v průběhu potřeba měnit hodnotu parametru například v rozsahu od 200 do 600, evaluátor mapuje hodnoty z intervalu <0,1> na hodnoty v intervalu <200, 600>.
- `AnimatorUpdateListener` – volá se při každé změně zobrazení v důsledku animace.

S objektem manipuluje třída `ObjectAnimator` odvozená od třídy `ValueAnimator`. Typicky se volá nad objektem odvozeným od `View`. K dispozici jsou metody `setRotationX()`, `setRotationY()`, `setTranslationX()`, `setTranslationY()`, `setScaleX()`, `setScaleY()` a `setAlpha()`.

POZNÁMKA

Všimněte si, že rotovat je možné podle osy X i Y, což znamená, že Property Animation nás posouvá z 2D do 3D.

Možnosti a princip fungování Property Animation budeme demonstrovat na dvou příkladech. V prvním ukážeme využití objektů popsaných v této části. Druhý příklad bude zaměřen na posloupnost na sebe navazujících různých typů animací.

Příklad na ilustraci principu fungování Property Animation

Nejprve ukážeme nejjednodušší příklad, v kterém pomocí animace změníme barvu pozadí objektu `ImageView`. Stejně dobře by posloužil příklad na změnu jiného parametru, například velikosti objektu, pro ilustraci fungování objektu `TypeEvaluator`, ale bude lepší, když ukážeme změnu barvy jako komplexnějšího parametru, než je jednoduchá změna velikosti. V kódu XML layoutu hlavní aktivity je jen prvek `ImageView` a tlačítko na spuštění animace.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">

  <ImageView
    android:id="@+id/image_view1"
    android:layout_width="match_parent"
    android:layout_height="300dp"
    android:contentDescription="@string/obrazek_popis" />

  <Button
    android:id="@+id/start_button"
    android:layout_width="match_parent"
    android:layout_height="100dp"
    android:padding="30dp"
    android:text="@string/tlacidlo_popis"/>

</LinearLayout>
```

Vše se odehrává v obsluze události stisknutí tlačítka, kterým uživatel spustí animaci. Vytvoříme objekt `ValueAnimator`, jehož úlohou bude měnit hodnotu barvy ze žluté na zelenou. Mapování změny hodnoty kódu barvy definujeme pomocí objektu `TypeEvaluator`. V tomto případě, jelikož se jedná o hodnotu ARGB, se využije `ArgbEvaluator`.

```
ValueAnimator anim = ValueAnimator.ofObject(new ArgbEvaluator(),
  Color.YELLOW, Color.GREEN);
```

Evaluátor v tomto případě nemůže provést lineární mapování, jelikož hexadecimální hodnoty přiřazené použitým barvám jsou:

```
Color.YELLOW = #ffff00
Color.GREEN = #00ff00
```

Pokud by interpolátor začal hodnotu mapovat lineárně, začala by se projevovat modrá složka barvy, jelikož pozice barevných složek RGB v hexadecimální konstantě čísla barvy je `#RRGGBB`.

TIP

K dispozici jsou evaluátory `ArgbEvaluator`, `FloatEvaluator`, `IntEvaluator` a `RectEvaluator`.

V listeneru, který se volá při každé změně animace, je možné realizovat nějakou akci. V našem příkladu pro ilustraci fungování nastavíme pro `ImageView` novou hodnotu pozadí.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState)        {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button startButton = (Button) findViewById(R.id.start_button);
    startButton.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
        spustAnimaciu();
      }
    });
  }

  public void spustAnimaciu() {
    final ImageView imageView = (ImageView) findViewById(R.id.image_view1);
    ValueAnimator anim = ValueAnimator.ofObject(new ArgbEvaluator(),
      Color.YELLOW, Color.GREEN);

    anim.addUpdateListener(new AnimatorUpdateListener() {
      @Override
      public void onAnimationUpdate(ValueAnimator animation) {
        imageView.setBackgroundColor((Integer) animation
          .getAnimatedValue());
      }
    });
    anim.setDuration(3000);
    anim.start();
  }
}
```

Příklad komplexnějšího využití Property Animation

Tento příklad ukáže, jak je možné vytvořit posloupnost na sebe navazujících různých typů animací. Abyste měli možnost porovnávat, bude v následujícím příkladu využití Property Animation podobná animace jako v části *Příklad animace Tween*, tedy snižování průsvitnosti, pohyb objektu a změna velikosti. K posunu objektu je metodické a účelné definovat rozměr posunutí ve zdrojích v souboru *dimens.xml*.

```
<resources>
  <dimen name="posun">200dp</dimen>
</resources>
```

Obrázek, který bude předmětem animace, umístěte do složky *drawable*. V tomto příkladu je to soubor *et.png*.

Návrhový kód XML hlavní aktivity aplikace obsahuje pouze jeden prvek typu `ImageView`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  tools:context="sk.pcrevue.llacko.animacia2.MainActivity">

  <ImageView
    android:id="@+id/obrazok"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:src="@drawable/droid"
    android:alpha="0"
    android:contentDescription="Androidko"/>

</RelativeLayout>
```

Animace se spustí automaticky v obsluze události `onWindowFocusChanged()`, kdy se spustí první úloha jako objekt `Runnble`. V tomto případě je to změna průsvitnosti, tedy změna parametru `Alpha`. Po jejím ukončení se prostřednictvím metody `withEndAction()` spustí další úloha, v tomto případě rotace obrázku:

```
Runnable pruhlednost = new Runnable()
{
  public void run()
  {
    iv.animate().setDuration(2000)
      .setInterpolator(new LinearInterpolator()).alpha(1.0f)
      .withEndAction(rotace);
  }
};
```

Rotace stejným mechanizmem spustí další část animace – posun.

UPOZORNĚNÍ

Metoda `withEndAction()` je dostupná až od API 16 (Android 4.1 Jelly Bean).

Kompletní kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  private ImageView iv;
  @Override
  public void onCreate(Bundle savedInstanceState)  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
```

```
  @Override
  public void onWindowFocusChanged(boolean hasFocus)  {
    super.onWindowFocusChanged(hasFocus);
    iv = (ImageView) findViewById(R.id.obrazok);
    if (hasFocus) priehladnost.run();
  }

  Runnable priehladnost = new Runnable() {
    public void run() {
      iv.animate().setDuration(2000)
        .setInterpolator(new LinearInterpolator()).alpha(1.0f)
        .withEndAction(rotacia);
    }
  };

  Runnable rotacia = new Runnable() {
    public void run() {
      iv.animate().setDuration(3000)
        .setInterpolator(new AccelerateInterpolator())
        .rotationBy(1080.0f).withEndAction(posun);
    }
  };

  Runnable posun = new Runnable() {
    public void run()  {
      float transl = getResources().getDimension(R.dimen.posun);
      iv.animate().setDuration(2000)
        .setInterpolator(new OvershootInterpolator())
        .translationXBy(transl).translationYBy(transl)
        .withEndAction(zvetsenie);
    }
  };

  Runnable zvetsenie = new Runnable() {
    public void run()  {
      iv.animate().setDuration(2000)
        .setInterpolator(new AnticipateInterpolator())
        .scaleXBy(1.0f).scaleYBy(1.0f);
    }
  };
}
```


Animace prvků uživatelského rozhraní

Framework Transitions umožňující vývojářům provádět přechodové animační efekty na obrazovce uživatelského rozhraní přibyl do Androidu od verze Android 4.4 KitKat. Prvky uživatelského rozhraní můžete zeslabit, posunout na nové pozice, případně je můžete nechat vyklouznout z obrazovky a podobně.

Nejúčinnější způsob, jak implementovat přechody uvnitř aplikačního kódu, jsou scény. Scéna představuje buď kompletní rozvržení obrazovky uživatelského rozhraní, nebo podmnožinu prvků reprezentovanou `ViewGroup`. Scény jsou definované tak, že znázorňují dva různé stavy, v tomto případě na začátku a na konci transformace. Například můžete posunout tlačítko dolů, aby se nad něj dal umístit objekt, který aktivovalo. Přechody lze realizovat i dynamicky zevnitř kódu aplikace.

Nejjednodušší přechodová transformace využívá metodu `beginDelayedTransition()` ze třídy `TransitionManager`.

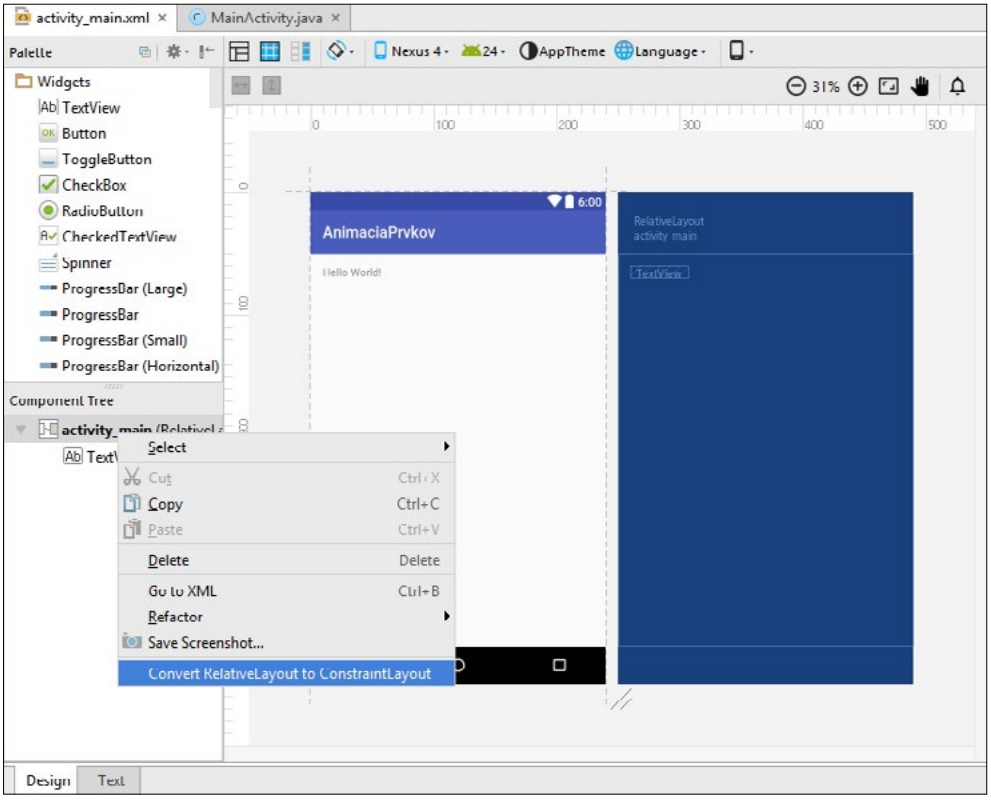
Framework Transitions využívá objekty nazývané interpolátory:

- `LinearInterpolator` – animace se realizuje při konstantní rychlosti.
- `AccelerateDecelerateInterpolator` – standardně se animace provede konstantní rychlostí. Tento interpolátor definuje, že animace začne pomalu, postupně se zrychluje, až se dosáhne středu animačního cyklu. Potom se animace znovu začne zpomalovat.
- `AccelerateInterpolator` – od předchozího interpolátoru se liší tím, že od začátku až do konce se animace jen zrychluje.
- `DecelerateInterpolator` – animace začne rychle a postupně zpomalí.
- `AnticipateInterpolator` – dobrou analogií této animace je hod a nejlepší prak ve hře Angry Birds. Nejprve se ruka s objektem pohybuje dozadu a potom se objekt, který házíme, začne pohybovat správným směrem. Objekty se tedy nejprve pohybují na krátkou vzdálenost v opačném směru, než je nakonfigurovaná animace, a až potom se začnou pohybovat v správném směru.
- `AnticipateOvershootInterpolator` – kombinuje efekt animace s interpolátorem `AnticipateInterpolator` s návratem do cílové polohy na obrazovce.
- `BounceInterpolator` – animovaný objekt se po přesunutí do cílového umístění odrazí.
- `CycleInterpolator` – animace se bude opakovat zadaný počet opakování.
- `OvershootInterpolator` – nastřelování na zadanou cílovou pozici před návratem.

Příklad – Animace přesunu tlačítek

Námět tohoto příkladu by se dal popsat jako žonglování s tlačítky. Jak už bylo zmíněno, tlačítka je často potřeba po stisknutí přesunout jinam, aby uvolnila místo na zobrazení prvku či jinému ději, který sama vyvolala. Na začátku animace budou dvě tlačítka v horní části obrazovky, na konci je přesunete do spodní části obrazovky a zároveň přehodíte horizontálně jejich pořadí. Tlačítko A, které bylo původně vlevo nahoře, bude tedy vpravo dole a tlačítko B se z pozice vpravo nahoře posune do pozice vlevo dole. Interpolátor `bounce_interpolator` zařídí, že tlačítka při „nárazu" na spodní okraj trochu poskočí, jako kdyby se od něj odrazila.

V příkladu využijeme vizuální kontejner `ConstraintLayout`, který je dostupný od verze Android Studio 2.2 a umožňuje mnohem jednodušší vizuální návrh prvků uživatelského rozhraní. Po vytvoření projektu s hlavní aktivitou typu **Empty Activity** definiční soubor hlavní aktivity využívá na zapouzdření prvků kontejner `RelativeLayout`. Typ kontejneru změníte buď přepsáním počátečního elementu, jelikož se ale v tomto případě bavíme o vizuálním návrhu, přepněte se do záložky **Design** a v okně **Component Tree** klepněte na volbu **Convert Relativelayout to ConstraintLayout**.



Obrázek 11.11: Změna vizuálního kontejneru na ConstraintLayout

Všimněte si v návrhovém kódu hlavní aktivity, jak se při této konverzi změnily vlastnosti prvků, v tomto případě `TextView`.

```
<android.support.constraint.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.animaciaprvkov.MainActivity">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    tools:layout_constraintTop_creator="1"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    tools:layout_constraintLeft_creator="1"
    app:layout_constraintLeft_toLeftOf="parent"
```

```
    app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Nyní vytvoříte dva definiční soubory scén uživatelského rozhraní také využijete `ConstraintLayout`.

Počáteční stav *Scena1.xml*:

```
<android.support.constraint.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/constraintLayout">

  <Button
    android:text="Tlačidlo A"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
    app:layout_constraintLeft_toLeftOf="@+id/constraintLayout"
    android:layout_marginStart="16dp"
    app:layout_constraintTop_toTopOf="@+id/constraintLayout"
    android:layout_marginTop="16dp"
    android:onClick="NaScenu2" />

  <Button
    android:text="Tlačidlo B"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    app:layout_constraintTop_toTopOf="@+id/constraintLayout"
    android:layout_marginTop="16dp"
    app:layout_constraintRight_toRightOf="@+id/constraintLayout"
    android:layout_marginEnd="16dp"
    android:onClick="NaScenu1" />

</android.support.constraint.ConstraintLayout>
```

Koncový stav *Scena2.xml*:

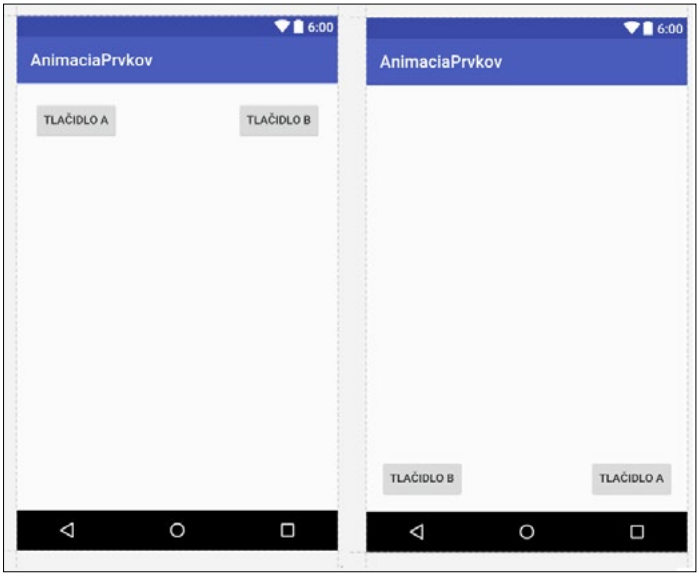
```
<android.support.constraint.ConstraintLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/constraintLayout">

  <Button
    android:text="Tlačidlo A"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button"
```

```
    android:onClick="NaScenu2"
    app:layout_constraintRight_toRightOf="@+id/constraintLayout"
    android:layout_marginEnd="20dp"
    app:layout_constraintBottom_toBottomOf="@+id/constraintLayout"
    android:layout_marginBottom="20dp" />

  <Button
    android:text="Tlačidlo B"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:onClick="NaScenu1"
    app:layout_constraintLeft_toLeftOf="@+id/constraintLayout"
    android:layout_marginStart="20dp"
    app:layout_constraintBottom_toBottomOf="@+id/constraintLayout"
    android:layout_marginBottom="20dp" />

</android.support.constraint.ConstraintLayout>
```



Obrázek 11.12: Počáteční a finální stav animace

Do složky *res* přidejte podsložku *transition*, ve které bude definiční soubor transformace *transition.xml*.

```
<transitionSet
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:transitionOrdering="sequential" >
  <changeBounds
    android:duration="3000"
    android:interpolator="@android:anim/linear_interpolator"
  </changeBounds>
</transitionSet>
```

Animace se odehrává v kódu hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  ViewGroup rootContainer;
  Scene s1, s2;
  Transition transitionMgr;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    rootContainer = (ViewGroup) findViewById(R.id.rootContainer);

    transitionMgr = TransitionInflater.from(this)
      .inflateTransition(R.transition.transition);

    s1 = Scene.getSceneForLayout(rootContainer, R.layout.scena1, this);
    s2 = Scene.getSceneForLayout(rootContainer, R.layout.scena2, this);
    s1.enter();
  }

  public void NaScenu2 (View view)
  {
    TransitionManager.go(s2, transitionMgr);
  }
  public void NaScenu1 (View view)
  {
    TransitionManager.go(s1, transitionMgr);
  }

}
```

KAPITOLA

12

Multimédia

Témata kapitoly:

- Příklad – Přehrávání zvuku
- Příklad – Přehrávání videa
- Příklad – Nahrávání zvuku
- Příklad – Snímání fotografie I
- Příklad – Snímání fotografie II
- Příklad – Snímání fotografie s využitím externí aplikace
- Příklad – Nahrávání videa

Moderní telefony a tablety mají kvalitní hardware nejen na zobrazování fotografií, přehrávání zvuku a videa, ale i na jejich snímání a zpracování. V přenosných zařízeních se často využívá možnost nahrávat hlasové poznámky. Vestavěné fotoaparáty a kamery jsou stále kvalitnější, takže běžně zvládají natáčet video v rozlišení full HD 1 920 × 1 080 pixelů (1080p), ale i v rozlišení 4K (3 840 × 2 160). Navíc multimédia – tedy možnost přehrávat v aplikaci zvuk, případně video, například instruktážní – jsou jedním z hlavních faktorů prospívajících atraktivnosti aplikace.

Na podporu práce s multimédii jsou na platformě Android vývojářům aplikací k dispozici třídy `AudioManager` – třída umožňuje aplikacím přehrávat zvukové efekty a zvukové soubory či ovládat hardwarová zařízení související s přehráváním zvuku, například reproduktory a bezdrátová sluchátka. Aplikace získá referenci na třídu `AudioManager` voláním metody `getSystemService(Context.AUDIO_SERVICE)`. Následně je možné načítat zvukové soubory a ovládat hardwarová zařízení.

- `SoundPool` – třída obsahuje kolekci zvukových vzorků nebo streamů. `SoundPool` umožňuje míchat a hrát více vzorků současně.
- `RingtoneManager`, `Ringtone` – na ovládání zvuků vyzvánění a alarmů.
- `MediaPlayer` – třída umožňuje aplikacím přehrávat audio- a video- soubory. Z třídy `MediaPlayer` budete nejčastěji využívat metody:
  - `setDataSource()` – pomocí této metody oznámíte třídě `MediaPlayer`, jaké streamy se budou přehrávat.
  - `prepare()` – v metodě se inicializuje `MediaPlayer` a načítají se potřebné multimediální streamy. Metoda je synchronní, což znamená, že ji budete používat tehdy, když je mediální obsah uložen v souboru v zařízení. K dispozici je i asynchronní verze této metody, kterou použijete, když je mediální obsah streamovaný z internetu.
  - `start()` – umožní spustit přehrávání nebo pokračovat v pozastaveném přehrávání.
  - `pause()` – umožní pozastavit přehrávání.

- `seekTo()` – metoda umožňuje přejít na konkrétní pozici ve streamu.
- `stop()` – zastaví přehrávání.
- `release()` – uvolňuje aktuálně používané prostředky na přehrávání multimediálního obsahu.

- `MediaRecorder` – umožňuje aplikacím záznam zvuku a obrazu.
- `Camera` – umožňuje aplikacím ovládání hardwaru na snímání fotografií.

Příklad – Přehrávání zvuku

Zadání příkladu spočívá kompletně v jeho názvu. Vytvořte projekt aplikace pro Android, která bude přehrávat soubor s krátkým zvukovým efektem, například sirénou. Soubor získáte z internetu. Ve složce *res* vytvořte vnořenou složku *raw* a do ní překopírujte zvukový soubor.

V layoutu hlavní aktivity bude tlačítko na přehrání zvukového efektu a dvě tlačítka na ovládání hlasitosti. Úroveň hlasitosti ve škále od 0 do 10 také zobrazíme na displeji.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context="sk.pcrevue.llacko.multimedia1.MainActivity">

  <Button
    android:id="@+id/play"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dip"
    android:enabled="false"
    android:text="Play" >
  </Button>

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="10dip"
    android:textSize="32sp" >
  </TextView>

  <LinearLayout
    android:id="@+id/linearLayout1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="25dp" >

    <Button
      android:id="@+id/down"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="-" >
      </Button>

      <Button
        android:id="@+id/up"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="+" >
      </Button>
    </LinearLayout>
  </LinearLayout>
```

V kódu hlavní aktivity, v metodě `OnCreate`, získá aplikace odkaz na třídu `AudioManager` a nastaví implicitní hodnotu hlasitosti.

Po získání reference na tlačítko `Play` se toto tlačítko zakáže doté doby, než se vytvoří třída `SoundPool` a naplní se obsahem, tedy zvukem ze souboru. Po naplnění bude `SoundPool` obsahovat jeden audiostream. Pokud by se to nepodařilo, nebylo by co přehrávat. Tlačítko Play povolíme po nahrání zvuku v metodě `setOnLoadCompleteListener`. Pro jednoduchost načítáme zvukový soubor v hlavním vlákně, jelikož je soubor krátký a je součástí aplikace. Není nutné ho nahrávat z internetu ani z paměťové karty.

Aplikace může přehrávat zvuk až poté, co získá fokus na příslušný hardware.

```
int result = audioManager.requestAudioFocus(afChangeListener,
  AudioManager.STREAM_MUSIC, AudioManager.AUDIOFOCUS_GAIN);
```

V aplikaci na přehrávání zvuku by se hodilo pomocí zvukového efektu signalizovat i klepnutí na tlačítka regulace hlasitosti. Proto přehrajeme zvuk:

```
audioManager.playSoundEffect(AudioManager.FX_KEY_CLICK);
```

Kompletní kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {
  private int hlasitost = 5, max = 10, min = 0;
  private SoundPool soundPool;
  private int id;
  private AudioManager audioManager;
  private boolean bPlay;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final TextView tv = (TextView) findViewById(R.id.textView1);
    tv.setText(String.valueOf(hlasitost));

    audioManager = (AudioManager) getSystemService(AUDIO_SERVICE);
```

```
final Button upButton = (Button) findViewById(R.id.up);
upButton.setOnClickListener(new View.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    audioManager.playSoundEffect(AudioManager.FX_KEY_CLICK);
    if (hlasitost < max) {
      hlasitost ++;
      tv.setText(String.valueOf(hlasitost));
    }
  }
});

final Button downButton = (Button) findViewById(R.id.down);
downButton.setOnClickListener(new View.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    audioManager.playSoundEffect(AudioManager.FX_KEY_CLICK);
    if (hlasitost > min) {
      hlasitost --;
      tv.setText(String.valueOf(hlasitost));
    }
  }
});

final Button playButton = (Button) findViewById(R.id.play);
playButton.setEnabled(false);

soundPool = new SoundPool(1, AudioManager.STREAM_MUSIC, 0);
id = soundPool.load(this, R.raw.zvuk, 1);

soundPool.setOnLoadCompleteListener(
  new SoundPool.OnLoadCompleteListener()
{
  @Override
  public void onLoadComplete(SoundPool soundPool, int sampleId,
    int status)
  {
    if (status ==0 ) playButton.setEnabled(true);
  }
});

playButton.setOnClickListener(new View.OnClickListener()
{
  @Override
  public void onClick(View v)
  {
    if (bPlay)
```

```
        soundPool.play(id, (float) hlasitost / max,
        (float) hlasitost / max, 1, 0, 1.0f);
      }

    });

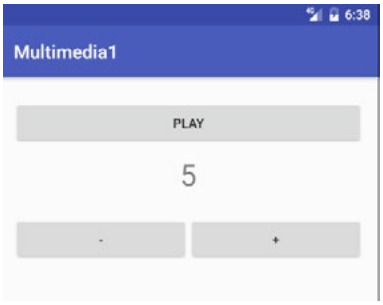
    int result = audioManager.requestAudioFocus(afChangeListener,
      AudioManager.STREAM_MUSIC, AudioManager.AUDIOFOCUS_GAIN);

    bPlay = AudioManager.AUDIOFOCUS_REQUEST_GRANTED == result;
  }

  @Override
  protected void onResume() {
    super.onResume();
    audioManager.setSpeakerphoneOn(true);
    audioManager.loadSoundEffects();
  }

  @Override
  protected void onPause()  {
    if (soundPool != null)  {
      soundPool.unload(id);
      soundPool.release();
      soundPool = null;
    }
    audioManager.setSpeakerphoneOn(false);
    audioManager.unloadSoundEffects();
    super.onPause();
  }


  AudioManager.OnAudioFocusChangeListener afChangeListener =
    new AudioManager.OnAudioFocusChangeListener()
  {
    public void onAudioFocusChange(int focusChange)
    {
      if (focusChange == AudioManager.AUDIOFOCUS_LOSS)   {
        audioManager.abandonAudioFocus(afChangeListener);
        bPlay = false;
      }
    }
  };
}
```



Obrázek 12.1: Jednoduchá aplikace na přehrání zvukového souboru a nastavování hlasitosti

Příklad – Přehrávání videa

Na zobrazování videa se využívá třída `VideoView` odvozená od třídy `SurfaceView`, která vnitřně využívá `MediaPlayer`. Třída `VideoView` dokáže načítat videoobsah z různých zdrojů a řídit jeho přehrávání. V aplikaci zobrazíme okno na přehrávání videa a integrované ovládací prvky na řízení přehrávání.

Vytvořte projekt aplikace pro Android, která bude přehrávat krátké video. Soubor získáte z internetu nebo z příkladů, které jsou součástí Android SDK. Ve složce *res* vytvořte vnořenou složku *raw* a do ní překopírujte soubor s videem. Jelikož bude video součástí aplikace, vybírejte krátké video, aby velikost souboru nepřesáhla několik megabajtů.

V layoutu hlavní aktivity bude pouze prvek `VideoView` na přehrávání videa.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/LinearLayout1"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <VideoView
    android:id="@+id/prehravac"
    android:layout_width="480dp"
    android:layout_height="640dp"
    android:layout_centerInParent="true"
    android:gravity="center" >
  </VideoView>

</RelativeLayout>
```

V metodě `OnCreate` získá aplikace odkaz na prvek `VideoView`. Následně vytváří instanci třídy `MediaController`, která obsahuje ovládací prvky na ovládání přehrávače.

V projektech tohoto typu můžete využít metody třídy `VideoView`:

- `setVideoPath(String path)` – cesta k videosouboru. Může to byť adresa URL videa na webu nebo cesta k lokálnímu souboru.
- `setVideoUri(Uri uri)` – adresa videa na webu jako objekt URI.
- `start()` – spustí přehrávání.
- `stopPlayback()` – zastaví přehrávání.

- `pause()` – přeruší přehrávání.
- `isPlaying()` – vrátí hodnotu `Boolean`, zda se video právě přehrává.
- `setMediaController(MediaController)` – vytvoří instanci objektu `MediaController` na zobrazení ovládacích prvků přehrávání.
- `setOnCompletionListener(MediaPlayer.OnCompletionListener)` – callback metoda; volá se, když je přehrávání videa na konci.
- `setOnErrorListener(MediaPlayer.OnErrorListener)` – callback metoda; volá se, pokud při přehrávání videa dojde k chybě.
- `setOnPreparedListener(MediaPlayer.OnPreparedListener)` – callback metoda; volá se, když je video připravené k přehrávání.
- `getDuration()` – čas trvání videa. Hodnotu `-1` vrátí, pokud se neaktivuje callback `OnPreparedListener()`.
- `getCurrentPosition()` – pozice přehrávání (`integer`).

Video se nahrává přes URI, který odkazuje na soubor uložený ve složce *res/raw*. Metoda `OnPreparedListener` se bude volat, jakmile je soubor s videem načten a připraven k přehrání.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {
  VideoView videoView = null;

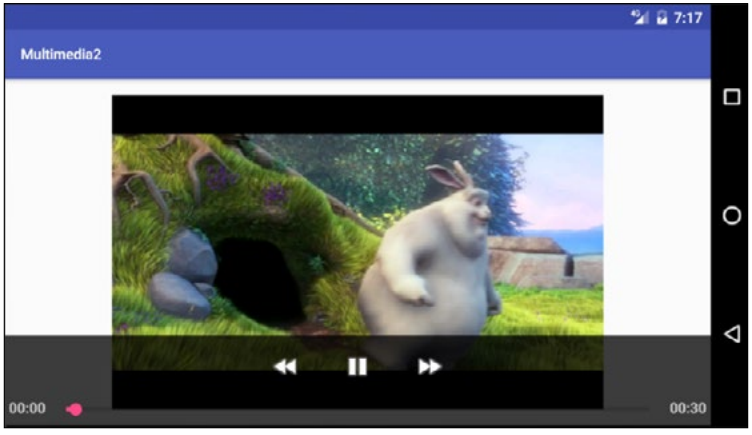
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    videoView = (VideoView) findViewById(R.id.prehravac);

    final MediaController mediaController =
      new MediaController(MainActivity.this, true);
    mediaController.setEnabled(false);
    videoView.setMediaController(mediaController);
    videoView.setVideoURI(Uri.parse(
      "android.resource://sk.pcrevue.llacko.multimedia2/raw/demovideo"));

    videoView.setOnPreparedListener(new MediaPlayer.OnPreparedListener()  {
      @Override
      public void onPrepared(MediaPlayer mp)  {
        mediaController.setEnabled(true);
      }
    });
  }

  @Override
  protected void onPause()    {
    if (videoView != null && videoView.isPlaying())     {
      videoView.stopPlayback();
      videoView = null;
```

```
        }
        super.onPause();
      }
    }
```



Obrázek 12.2: Test aplikace

Pokud přehráváte video z internetu, nezapomeňte tuto skutečnost deklarovat v aplikačním manifestu:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Příklad – Nahrávání zvuku

Třídu `MediaRecorder` lze použít také k záznamu zvuku i videa. V aplikacích budete nejčastěji využívat metody:

- `setAudioSource()` – výběr zdroje, odkud se bude nahrávat zvuk
- `setVideoSource()` – výběr zdroje, odkud se bude nahrávat video
- `setOutputFormat()` – nastavení formátu souboru, do kterého se nahrávaný zvuk nebo video uloží (například MP3, MP4)

Význam metod `prepare()`, `start()`, `stop()` a `release()` není potřeba vysvětlovat.

V layoutu hlavní aktivity takto jednoduché aplikace není moc co řešit. Uživatelské rozhraní aplikace sestává jen ze dvou tlačítek. První aktivuje nahrávání a druhé přehrávání nahraného záznamu. Třetí tlačítko, které by mělo funkci stop, není zapotřebí, protože tlačítka na nahrávání a přehrávání fungují systémem start/stop.

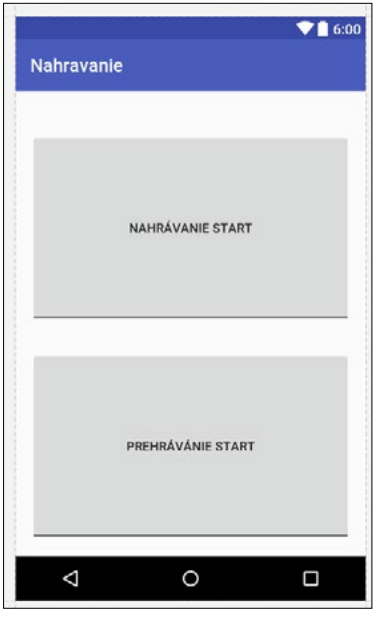
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <ToggleButton
    android:id="@+id/record_button"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:layout_marginTop="30dip"
    android:layout_weight="1"
    android:textOn="Nahrávánie STOP"
    android:textOff="Nahrávanie START"/>

  <ToggleButton
    android:id="@+id/play_button"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="30dip"
    android:layout_weight="1"
    android:textOff="Prehrávánie START"
    android:textOn="Prehrávánie STOP"/>

</LinearLayout>
```



Obrázek 12.3: Uživatelské rozhraní se dvěma Start/Stop tlačítky

Tato aplikace bude potřebovat v *AndroidManifest.xml* povolení k nahrávání zvuku, jelikož tato aktivita může znamenat zásah do soukromí uživatele a přístup k externí paměti, kam bude ukládat soubor s nahrávkou.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Po stisknutí tlačítka **Start nahrávání** začne aplikace nahrávat zvuk z vestavěného nebo externího mikrofonu. Popis tlačítka, kterým jsme spustili nahrávání, se změní na **Stop nahrávání**. Po zastavení nahrávání je možné nahraný zvuk uložený v souboru *record1.3gp* přehrát. Formát nahrávání je nastaven jako `MediaRecorder.OutputFormat.THREE_GPP`.

```
public class MainActivity extends AppCompatActivity {

  private static final String soubor = Environment
    .getExternalStorageDirectory().getAbsolutePath()
    + "/record1.3gp";
  private AudioManager audioManager;
  private MediaRecorder mRecorder;
  private MediaPlayer mPlayer;


  @Override
  public void onCreate(Bundle icicle)  {
    super.onCreate(icicle);
    setContentView(R.layout.activity_main);

    final ToggleButton btRecord = (ToggleButton)
      findViewById(R.id.record_button);
    final ToggleButton btPlay = (ToggleButton)
      findViewById(R.id.play_button);

    btRecord.setOnCheckedChangeListener(new
      CompoundButton.OnCheckedChangeListener() {
      @Override
      public void onCheckedChanged(CompoundButton buttonView,
        boolean isChecked) {
        btPlay.setEnabled(!isChecked);
        onRecordPressed(isChecked);
      }
    });

    btPlay.setOnCheckedChangeListener(new
      CompoundButton.OnCheckedChangeListener() {
      @Override
      public void onCheckedChanged(CompoundButton buttonView,
        boolean isChecked)  {
        btRecord.setEnabled(!isChecked);
        onPlayPressed(isChecked);
      }
    });
    audioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
    audioManager.requestAudioFocus(afcListener,
      AudioManager.STREAM_MUSIC, AudioManager.AUDIOFOCUS_GAIN);
  }

  private void onRecordPressed(boolean bReady)  {
    if (bReady) startRecording();
    else stopRecording();
  }

  private void startRecording()  {
    mRecorder = new MediaRecorder();
```

```
  mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
  mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
  mRecorder.setOutputFile(soubor);
  mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

  try {
    mRecorder.prepare();
  } catch (IOException e) {
    Log.e("ZvukZaznam", "CHYBA MediaRecorder");
  }

  mRecorder.start();
}

private void stopRecording()  {
  if (mRecorder != null)  {
    mRecorder.stop();
    mRecorder.release();
    mRecorder = null;
  }
}

private void onPlayPressed(boolean bReady)   {
  if (bReady) startPlaying();
  else stopPlaying();
}

private void startPlaying() {
  mPlayer = new MediaPlayer();
  try  {
    mPlayer.setDataSource(soubor);
    mPlayer.prepare();
    mPlayer.start();
  } catch (IOException e) {
    Log.e("ZvukZaznam", "CHYBA MediaPlayer");
  }
}

private void stopPlaying()  {
  if (mPlayer  != null)  {
    if (mPlayer.isPlaying()) mPlayer.stop();
    mPlayer.release();
    mPlayer = null;
  }
}

AudioManager.OnAudioFocusChangeListener afcListener =
  new AudioManager.OnAudioFocusChangeListener()  {
  public void onAudioFocusChange(int fokus)
  {
    if (fokus == AudioManager.AUDIOFOCUS_LOSS)   {
```

```
            audioManager.abandonAudioFocus(afcListener);
            if (mPlayer.isPlaying()) stopPlaying();
          }
        }
      };

      @Override
      public void onPause() {
        super.onPause();
        if (mRecorder  != null)   {
          mRecorder.release();
          mRecorder = null;
        }

        if (mPlayer  != null)  {
          mPlayer.release();
          mPlayer = null;
        }
      }
    }
```

Aplikaci vyzkoušejte na reálném zařízení.

Příklad – Snímání fotografie I

V příkladu využijeme možností třídy `Camera`, která aplikacím zprostředkovává přístup ke službě, která řídí vestavěnou kameru v zařízení. Pomocí metod třídy `Camera` můžete nastavovat parametry pro vestavěnou kameru, spustit a zastavit funkci náhledu, zaostřit a podobně. V dalším příkladu ukážeme modernější řešení využívající funkce novějších verzí Androidu. Toto řešení se používalo ve starších verzích.

POZNÁMKA

Aby příklad demonstrující využití kamery nebyl příliš rozsáhlý, rozdělili jsme ho na dvě části. V první části ukážeme zobrazení náhledu a snímání fotografie dotykem na displej, avšak fotografii nikam neuložíme. V druhém příkladu budeme aktivovat snímání obrázku tlačítkem a obrázek uložíme do souboru.

Tato aplikace bude potřebovat v *AndroidManifest.xml* povolení použití kamery, jelikož tato aktivita může znamenat zásah do soukromí uživatele. Možná budete chtít nastavit, aby aplikace mohla využít další funkce, například autofokus či blesk.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
```

Uživatelské rozhraní aplikace je nejjednodušší, jaké může být. Na celou plochu displeje se zobrazí náhled scény snímané kamerou. Sejmutí obrázku se zrealizuje klepnutím na displej. Snímání je indikováno dvojsekundovým „zmrazením“ náhledu. Layout hlavní aktivity tvoří pouze prvek `SurfaceView`, jehož úlohou je indikovat klepnutí na displej.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.fotoaparat1.MainActivity">

  <SurfaceView
    android:id="@+id/cameraView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
  </SurfaceView>
</RelativeLayout>
```

V kódu hlavní aktivity nastavíme zobrazování na celou obrazovku.

```
getWindow().setFormat(PixelFormat.TRANSLUCENT);
requestWindowFeature(Window.FEATURE_NO_TITLE);
getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
  WindowManager.LayoutParams.FLAG_FULLSCREEN);
```

Následně je potřeba získat instanci třídy `Camera` a nastavit všechny potřebné parametry na snímání. Určitě budete chtít nastavit náhled snímané scény na displeji. Po sejmutí obrázku jej může aplikace zpracovat a uložit, případně odeslat. Jakmile vaše aplikace už nepotřebuje hardware na snímání obrázků, uvolní ho, aby byl dostupný pro ostatní aplikace.

```
public class MainActivity extends Activity {
  private Camera camera;
  private RelativeLayout layout;
  private SurfaceHolder surfaceHolder;
  private static final int FREEZE = 2000;
  private boolean bSpusten;
}
```

V metodě `OnCreate` získá aplikace instanci třídy `Camera`. Pro jednoduchost využijeme metodu `open`, která vrací odkaz na první kameru v zařízení, na kterém je aplikace spuštěna. Zpravidla je to kamera na zadní straně zařízení. Pokud má přístroj více kamer, aplikace mohou získat instanci konkrétní kamery. V metodě `OnCreate` také inicializujeme snímání dotyku. Pokud se uživatel dotkne obrazovky, zavolá se metoda `onTouch` a v ní se sejme obrázek. Také nastavíme zobrazení náhledu, který ukazuje uživateli, na co je fotoaparát v daném okamžiku zaměřen.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  //na celú obrazovku
  getWindow().setFormat(PixelFormat.TRANSLUCENT);
  requestWindowFeature(Window.FEATURE_NO_TITLE);
  getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
    WindowManager.LayoutParams.FLAG_FULLSCREEN);

  setContentView(R.layout.activity_main);
```

```
    bSpusten = false;
    camera = instanceCamera();
    if (camera == null) finish();

    //nastavenie dotyku
    layout = (RelativeLayout) findViewById(R.id.layout_main);
    layout.setEnabled(false);
    layout.setOnTouchListener(new View.OnTouchListener() {
      @Override
      public boolean onTouch(View v, MotionEvent event) {
        if (event.getActionMasked() == (MotionEvent.ACTION_UP)) {
          camera.takePicture(shutterCallback, null,
            pictureCallback);
        }
        return true;
      }
    });

    //nastavenie pre nahlsd
    SurfaceView surfaceView = (SurfaceView) findViewById(R.id.cameraView);
    surfaceHolder = surfaceView.getHolder();
    surfaceHolder.addCallback(holderCallback);
  }

  private Camera instanceCamera() {
    try {camera = Camera.open();}
    catch (RuntimeException e) {
    Log.e("FOTO1", "ERR> instancia Camera");}
    return camera;
  }
```

SurfaceHolder.Callback využívá tři metody SurfaceCreated, surfaceChanged a surfaceDestroyed. V metodě surfaceCreated se nastaví zobrazení náhledu. Pokud se změní parametry objektu Surface, například při změně orientace zařízení, zavolá se metoda surfaceChanged. V této metodě zastavíme zobrazování náhledu a vzhledem k aktuálním rozměrům zařízení najdeme vhodnou velikost náhledu kamery. Pomocí metody setPreviewDisplay restartujeme náhled.

```
  SurfaceHolder.Callback holderCallback = new SurfaceHolder.Callback()   {
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
      try {
        camera.setPreviewDisplay(holder);
        camera.startPreview();
        bSpusten = true;
      }
      catch (IOException e) {
        Log.e("FOTO1", "ERR> preview");
      }
    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width,
```

```
      int height)  {
      if (surfaceHolder.getSurface() == null) return;
      layout.setEnabled(false);

      if (bSpusten == true)  {
        try  {
          camera.stopPreview();
          bSpusten = false;
        } catch (Exception e) {}
      }

      Camera.Parameters p = camera.getParameters();
      Camera.Size bestSize = findBestSize(p, width, height);
      p.setPreviewSize(bestSize.width, bestSize.height);
      camera.setParameters(p);

      try {camera.setPreviewDisplay(holder);}
      catch (IOException e) {Log.e("FOTO1", "ERR> preview display");}
      try  {
        camera.startPreview();
        bSpusten = true;
        layout.setEnabled(true);
      } catch (RuntimeException e) {Log.e("FOTO1", "ERR> preview");}
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {}
  };
```

V metodě `camera.takePicture(shutterCallback, null, pictureCallback)` jsou definované dva objekty: `CameraCallback` – `ShutterCallback` a `CameraCallback`. `ShutterCallback` se aktivuje v okamžiku, kdy uživatel vytvoří snímek, aby aplikace věděla, že došlo k sejmutí.

```
Camera.ShutterCallback shutterCallback = new Camera.ShutterCallback()
{
  @Override
  public void onShutter()
  {
    bSpusten = false;
  }
};
```

Metoda `CameraCallback` se zavolá po sejmutí obrázku vždy, když už je obrázek k dispozici. Klasické aplikace v této metodě ukládají do souboru. V tomto případě pouze zmrazíme náhled na dobu dvou sekund.

```
Camera.PictureCallback pictureCallback = new Camera.PictureCallback()
{
  @Override
  public void onPictureTaken(byte[] data, Camera camera)
  {
    try {Thread.sleep(FREEZE);}
    catch (InterruptedException e) {e.printStackTrace();}
```

```
    // ukladani obrazku

    // obnov preview
    try
    {
      camera.startPreview();
      bSpusten = true;
    } catch (Exception e) { Log.e("FOTO1", "ERR> preview");}
  }
};

@Override
protected void onPause()
{
  super.onPause();
  layout.setEnabled(false);
  if (camera != null)
  {
    if (bSpusten == true)
    {
      try
      {
        camera.stopPreview();
        bSpusten = false;
      }
      catch (Exception e) {Log.e("FOTO1", "ERR> onPause");   }
    }
    camera.release();
    camera = null;
  }
}

@Override
protected void onResume()
{
  super.onResume();
  if (null == camera) camera = instanceCamera();
  if (null == camera) finish();
}

private Camera.Size findBestSize(Camera.Parameters parameters, int width,
  int height)
{
  List<Camera.Size> sizes = parameters.getSupportedPreviewSizes();
  Camera.Size bestSize = sizes.remove(0);
  for (Camera.Size size : sizes)
  {
    if ((size.width * size.height) > (bestSize.width *
      bestSize.height))
      bestSize = size;
```

```
    }
    return bestSize;
  }
```

Příklad – Snímání fotografie II

V druhém příkladu budeme aktivovat snímání obrázku tlačítkem a obrázek uložíme do souboru. Uživatelské rozhraní bude obsahovat pouze jedno tlačítko.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${relativePackage}.${activityClass}" >

  <Button
    android:id="@+id/captureFront"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:onClick="onClick"
    android:text="FOTO" />
</RelativeLayout>
```

Tato aplikace bude potřebovat v *AndroidManifest.xml* povolení k použití kamery, jelikož tato aktivita může znamenat zásah do soukromí uživatele a přístup k externí paměti, kam se uloží sejmutý obrázek.

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Nejprve vytvoříme samostatnou třídu `UlozFoto`, která bude mít na starosti uložení sejmutého obrázku. Názvy obrázků se budou generovat s využitím data a času. Třída bude implementací `PictureCallback`.

```
public class UlozFoto implements PictureCallback
{
  private final Context context;

  public UlozFoto(Context context)
  {
    this.context = context;
  }

  @SuppressLint("SimpleDateFormat")
  @Override
  public void onPictureTaken(byte[] data, Camera camera)
  {
    File pictureFileDir = dejAdresar();
    if (!pictureFileDir.exists() && !pictureFileDir.mkdirs())
    {
      Toast.makeText(context, "Nemohu vytvořit adresář.",
```

```
        Toast.LENGTH_LONG).show();
      return;
    }

    SimpleDateFormat format = new SimpleDateFormat("yyyymmddhhmmss");
    String date = format.format(new Date());
    String photoFile = "Obrazek_" + date + ".jpg";
    String filename = pictureFileDir.getPath() + File.separator + photoFile;
    File pictureFile = new File(filename);

    try
    {
      FileOutputStream fos = new FileOutputStream(pictureFile);
      fos.write(data);
      fos.close();
      Toast.makeText(context, "Uložen obrázek:" + photoFile,
        Toast.LENGTH_LONG).show();
    } catch (Exception error)
    {
      Toast.makeText(context, "Chyba ukládání obrázku.",
        Toast.LENGTH_LONG).show();
    }
  }

  private File dejAdresar()
  {
    File sdDir = Environment
      .getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
    return new File(sdDir, "FotoMeAplikace");
  }
}
```

V kódu hlavní aktivity v obsluze stisknutí tlačítka voláme metodu `takePicture()`. Ta definuje pouze jeden callback – `PictureCallback`, který má na starosti uložení sejmutého obrázku.

```
public class MainActivity extends Activity
{
  private Camera camera;
  private int id = 0;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (!getPackageManager()
      .hasSystemFeature(PackageManager.FEATURE_CAMERA))
    {
      Toast.makeText(this, "Kamera není k dispozici",
      Toast.LENGTH_LONG).show();
    } else
```

```
    {
      id = predniKamera();
      if (id < 0)
      {
        Toast.makeText(this, "Přední kamera není k dispozici.",
          Toast.LENGTH_LONG).show();
      } else camera = Camera.open(id);
    }
  }

  public void onClick(View view)
  {
     camera.takePicture(null, null, new UlozFoto(getApplicationContext()));
  }

  private int predniKamera()
  {
    int camid = -1;
    int pocetKamer = Camera.getNumberOfCameras();
    for (int i = 0; i < pocetKamer; i++)
    {
      CameraInfo info = new CameraInfo();
      Camera.getCameraInfo(i, info);
      if (info.facing == CameraInfo.CAMERA_FACING_FRONT)
      {
        camid = i;
        break;
      }
    }
    return camid;
  }

  @Override
  protected void onPause()
  {
    if (camera != null)
    {
      camera.release();
      camera = null;
    }
    super.onPause();
  }
}
```

Příklad – Snímání fotografie s využitím externí aplikace

V tomto příkladu nebudeme řešit fyzické snímání obrázku z hardwaru. Aplikace vytvoří záměr, pomocí něhož oznámí operačnímu systému, že potřebuje sejmout obrázek. Operační systém vybere vhodnou aplikaci pro tento účel. Pokud je vhodných aplikací schopných splnit požadavek deklarovaný v záměru více, vybere vhodnou aplikaci uživatel. Uživatelské rozhraní bude obsahovat tlačítko na snímání a prvek `ImageView` na zobrazení obrázku.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="FOTO" >
  </Button>
  <ImageView
    android:id="@+id/obrázek"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:src="@drawable/ic_launcher" >
  </ImageView>
</LinearLayout>
```

Jelikož aplikace přímo nepřistupuje k hardwaru kamery, nepotřebuje povolení v manifestu. Toto povolení samozřejmě musí mít aplikace, která přebere záměr a provede sejmutí obrázku.

```
public class MainActivity extends Activity
{
  private static final int REQUEST_CODE = 1;
  private Bitmap bitmap;
  private ImageView imageView;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    imageView = (ImageView) findViewById(R.id.obrazek);
  }

  public void onClick(View View)
  {
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    intent.addCategory(Intent.CATEGORY_OPENABLE);
```

```
    startActivityForResult(intent, REQUEST_CODE);
  }

  @Override
  protected void onActivityResult(int requestCode,
    int resultCode, Intent data)
  {
    InputStream stream = null;
    if (requestCode == REQUEST_CODE && resultCode == Activity.RESULT_OK)
    try
    {
      if (bitmap != null) bitmap.recycle();
      stream = getContentResolver().openInputStream(data.getData());
      bitmap = BitmapFactory.decodeStream(stream);
      imageView.setImageBitmap(bitmap);
    } catch (FileNotFoundException e)
    {
      e.printStackTrace();
    }
    finally
    {
      if (stream != null)
        try {stream.close();}
        catch (IOException e) {e.printStackTrace();}
    }
  }
}
```

Příklad – Nahrávání videa

V příkladu budeme aktivovat nahrávání videa tlačítkem a záznam uložíme do souboru. Uživatelské rozhraní bude i v tomto případě obsahovat pouze jedno tlačítko.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/btZaznam"
    android:onClick="onClick"
    android:text="Záznam videa" >
  </Button>

</LinearLayout>
```

V kódu hlavní aktivity pomocí metody PackageManager.hasSystemFeature() zkontrolujeme, zda je zvolená kamera dostupná. Pro hlavní kameru na zadní straně telefonu je parametr konstanta PackageManager.FEATURE_CAMERA, pro přední „selfie“ kameru PackageManager.FEATURE_CAMERA_FRONT. Pokud chcete jen zjistit, zda telefon nějakou kameru má, použijete parametr PackageManager.FEATURE_CAMERA_ANY. Po stisknutí tlačítka nahrávání aplikace odešle záměr.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {

  private static final int VIDEO_CAPTURE = 101;
  private Uri fileUri;
  private static final String AUTHORITY="sk.pcrevue.llacko.kamera";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button recordButton =
      (Button) findViewById(R.id.btZaznam);

    if (!jeKamera())
      recordButton.setEnabled(false);
  }

  private boolean jeKamera() {
    return (getPackageManager().hasSystemFeature(
      PackageManager.FEATURE_CAMERA_ANY));
  }

  public void onClick(View view)
  {
    Intent intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    startActivityForResult(intent, VIDEO_CAPTURE);
  }

  protected void onActivityResult(int requestCode,
    int resultCode, Intent data) {

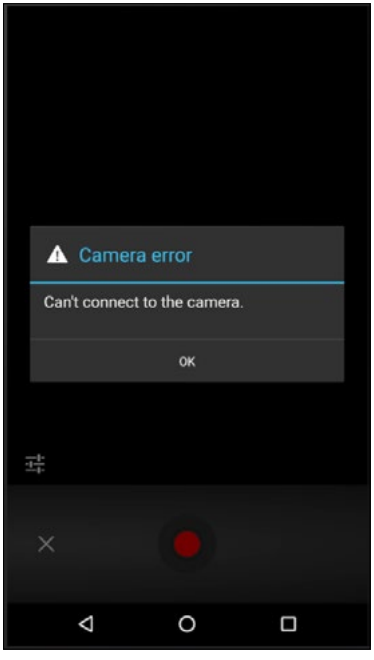
    Uri videoUri = data.getData();

    if (requestCode == VIDEO_CAPTURE) {
      if (resultCode == RESULT_OK) {
        Toast.makeText(this, "Video uložené do:\n" +
          videoUri, Toast.LENGTH_LONG).show();
      } else if (resultCode == RESULT_CANCELED) {
        Toast.makeText(this, "Nahrávanie prerušené.",
        Toast.LENGTH_LONG).show();
      } else {
        Toast.makeText(this, "Chyba",
          Toast.LENGTH_LONG).show();
```

```
            }
        }

    }
}
```

Aplikaci je potřeba spustit na reálném zařízení disponujícím fotoaparátem. Emulátor, který kameru nemá, je vynikajícím prostředkem na otestování situace, kdy není kamera dostupná.



Obrázek 12.4: Spuštění na emulátoru bez kamery

KAPITOLA

13

Senzory a komunikace

Témata kapitoly:

- Integrované senzory smartphonů a tabletů
- Snímač otisků prstů
- Komunikace přes Bluetooth

Integrované senzory smartphonů a tabletů

Moderní chytré telefony a tablety disponují bohatou senzorovou výbavou – zpravidla kompasem, akcelerometrem, gyroskopem, měřičem intenzity osvětlení, náklonoměrem a systémem k určování polohy. Náklonoměr a akcelerometr mají ve smartphonech a tabletech velmi důležitou funkci. Umožňuje nejen zjištění aktuální orientace přístroje (na výšku či na šířku), ale jelikož smartphone ani tablet nemá žádná kurzorová tlačítka ani žádný ekvivalent joysticku, využívá se naklápění přístroje velmi často i k ovládání her.

Senzory jsou specializované komponenty určené k měření konkrétních fyzikálních veličin. Telefony obsahují tyto komponenty zabudované ve svém hardwaru jako samostatné nebo komplexní integrované součástky. Android API kromě zpřístupnění hardwarových snímačů definuje i tzv. syntetické, respektive virtuální senzory, které zpracovávají údaje z jednoho nebo více hardwarových senzorů a po provedení vlastních výpočtů poskytují vlastní derivované údaje.

Základní kategorizace senzorů:

- **pohybové** (motion) – měří velikost akceleračních a rotačních sil, nejčastěji zrychlení, gravitaci a rotační vektory na gyroskopickém principu.
- **polohové** (position) – poskytují informace o poloze a orientaci. Využívají geomagnetické údaje a údaje z družic GPS.
- **environmentální** – měří fyzikální parametry okolního prostředí, nejčastěji intenzitu osvětlení, teplotu, tlak, vlhkost a podobně.

Ne všechna zařízení mají všechny zmíněné senzory a ne všechny verze Androidu podporují všechny typy senzorů.

Pro ilustraci možností představíme senzorovou výbavu běžného moderního telefonu střední třídy.

- **Proximity Sensor** – v telefonech měří vzdálenost od ucha. Využívá se k vypnutí dotykové obrazovky během přiložení telefonu k uchu a u některých přístrojů i na „Direct Call", tedy funkci, která umožní jednoduchým přiložením k uchu iniciovat hovor na zobrazované číslo.
- **Gyroskop** – měří pootočení zařízení v trojrozměrném prostoru.
- **Akcelerometr** – měří pohyb zařízení v trojrozměrném prostoru.
- **Geomagnetický senzor** – umožňuje funkci digitálního kompasu, případně vytvoření aplikace na vyhledávání kovových předmětů ve stěně.
- **Barometr** – měří atmosférický tlak v aktuálním místě, takže umožňuje vytvářet aplikace typu výškoměr.
- **Hall Sensor** – slouží u některých modelů na zjištění otevření či zavření krytu přístroje. Pravděpodobně se jedná o klasický magnetický senzor.
- **Light Sensor** – měří intenzitu okolního světla. Slouží k automatickému přizpůsobení jasu displeje okolním světelným podmínkám. Některé přístroje mají RGB Light Sensor, který měří intenzitu jednotlivých barevných složek světla – tedy červené, zelené, modré a bílé složky.
- **Gesture Sensor** – infračervené pole snímá gesta uživatele nad displejem.

Kromě izolovaných údajů z jednotlivých senzorů poskytuje firmware telefonu nebo tabletu i komplexní informace o dynamických změnách polohy telefonu. Existují totiž fyzická omezení jednotlivých senzorů, pro které může být těžké určit skutečnou orientaci a pohyb zařízení ze surových dat z jednotlivých senzorů. Například údaje z akcelerometru jsou odvozeny ze setrvačnosti vyplývající z pohybu zařízení. Gyroskop měří rychlost otáčení, ne však polohu. Proto jsou v operačním systému implementovány složité geometrické výpočty, které jsou nutné ke zjištění polohy orientace a pohybu ze surových dat z jednotlivých senzorů. Na platformě Android se jedná o `TYPE_ORIENTATION`. Takovéto informace jsou důležité pro aplikace pracující s rozšířenou realitou (augmented reality); jedná se hlavně o polohu, rotační zrychlení a lineární zrychlení.

Získávání údajů ze senzorů

Android sensor framework obsahuje třídy, rozhraní a metody umožňující aplikacím využít údaje ze senzorů. Aplikace načítá buď surové údaje přímo ze senzorů (raw sensor data), nebo komplexní údaje, které poskytuje operační systém na základě zpracování údajů z více senzorů.

Potřebujete znát hlavně tyto objekty „senzorového" frameworku:

- `SensorManager` – třída určená na vytvoření instancí služeb poskytovaných senzory, zpřístupnění, konfiguraci, kalibraci senzorů a registraci snímačů událostí, v původní terminologii event listeners.
- `Sensor` – třída na vytvoření instance specifického senzoru a získání/nastavení jeho parametrů.
- `SensorEvent` – objekt snímače událostí senzoru, pomocí kterého lze získat typ senzoru, u kterého došlo ke vzniku události, údaje poskytované senzorem, přesnost údajů a časovou známku vzniku události.

- `SensorEventListener` – rozhraní na implementaci metod vykonávaných v případě vzniku události senzoru – při změně údajů poskytovaných senzorem `onSensorChanged()` a při změně přesnosti těchto údajů `onAccuracyChanged()`.

Instanci objektu `SensorManager` získáte pomocí metody:

```
getSystemService(Context.SENSOR_SERVICE)
```

Identifikace senzoru a jeho možností (capabilities)

Pokud v aplikaci potřebujete získat údaje poskytované konkrétním senzorem, musíte tento senzor v první řadě zpřístupnit. Pokud zařízení, na kterém je aplikace spuštěná, daný senzor obsahuje, můžete případně upravit jeho vlastnosti (rozsah poskytovaných údajů, jejich rozlišení, nároky na spotřebu energie) a pokračovat v jejich využívání. V opačném případě, pokud je to možné, se aplikace může pokusit využít jiný typ senzoru. Pokud to možné není, aplikace uživatele informuje o tom, že pro příslušnou funkcionalitu je nezbytný senzor, který se v zařízení nenachází, a danou aplikaci není možné použít.

O potřebě existence konkrétního senzoru je vhodné potenciální uživatele vaší aplikace informovat přímo v jejím manifestu pomocí výrazu `android:required="true"` v rámci elementu `<uses-feature>`. Aplikace tak bude dostupná pouze pro ta zařízení, která požadovaný senzor obsahují.

Dostupnost senzorů zjistíte programovou konstrukcí:

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (mSensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null){
  // Senzor, v tomto prípade magnetometer je dostupný.
}
else {
  // Senzor je nedostupný.
}
```

Na přístup ke konkrétnímu senzoru využijete metodu:

```
SensorManager.getDefaultSensor(int typ)
```

Senzor se vybírá pomocí parametru `typ`:

- `TYPE_ACCELEROMETER` – zrychlení v m/s^2 ve třech fyzických osách (x, y a z)
- `TYPE_AMBIENT_TEMPERATURE` – teplota vzduchu místnosti v °C
- `TYPE_GRAVITY` – gravitační zrychlení v m/s^2 ve třech osách (x, y, z)
- `TYPE_GYROSCOPE` – měří zrychlení rotace v rad/s kolem tří os (x, y, a z)
- `TYPE_LIGHT` – měří osvětlení v luxech (lx)
- `TYPE_LINEAR_ACCELERATION` – měří zrychlení v m/s^2 bez vlivu gravitace v osách (x, y, z)
- `TYPE_MAGNETIC_FIELD` – hodnota geomagnetického pole ve třech osách (x, y, z) v jednotkách μT
- `TYPE_ORIENTATION` – komplexní informace z více senzorů, poloha rotace zařízení kolem tří os (x, y, z)
- `TYPE_PRESSURE` – měří okolní atmosférický tlak v jednotkách hPa nebo mbar

- TYPE_PROXIMITY – měří vzdálenost objektu v cm od displeje zařízení
- TYPE_RELATIVE_HUMIDITY – vlhkost okolního vzduchu v procentech (%)
- TYPE_ROTATION_VECTOR – orientace zařízení vztažená ke třem vektorům
- TYPE_TEMPERATURE – teplota ve °C

Tyto typy senzorů jsou podporovány od verze Android 4.0 (API 16).

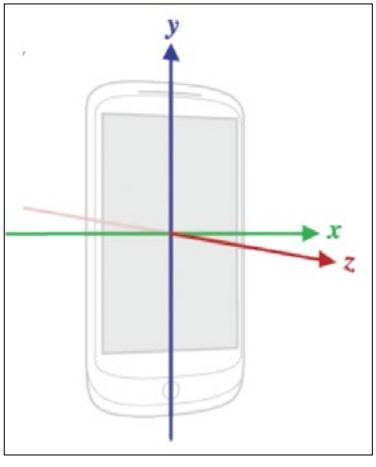
Aplikace získává údaje z vybraného senzoru přes SensorEventListener, který volá metodu onSensorChanged() po každé změně hodnoty naměřené senzorem. Při změně přesnosti měření se volá procedura onAccuracyChanged().

Událost senzoru vznikne vždy v okamžiku, kdy senzor detekuje změnu měřené veličiny. Od Android v2.3 (API Level 9) přitom rozeznáváme dva druhy senzorů související s periodicitou poskytování údajů:

- non-streaming sensor – poskytuje údaje pouze v případě změny měřených veličin.
- streaming sensor – poskytuje údaje v pravidelných intervalech, přičemž délku intervalu získáme pomocí metody getMinDelay(). Touto metodou zároveň získáme informaci o maximální rychlosti obnovy údajů (maximum data rate).

POZNÁMKA

Poloha zařízení se určuje vůči třem osám: x, y a z. Mapování os na zařízení se po změně jeho polohy nemění.



Obrázek 13.1: Poloha zařízení se určuje vůči třem osám: x, y a z

Příklad – Zobrazení údajů z akcelerometru

Většina zařízení s Androidem disponuje akcelerometrem schopným měřit zrychlení ve třech osách. V aplikacích se dá využít například ke snímání dynamiky pohybu při ovládání. V jednoduchém příkladu vypíšeme hodnoty zrychlení v osách x, y a z.

V aplikaci použijeme připojení na senzor TYPE_ACCELEROMETER, takže se budou načítat absolutní hodnoty zrychlení, v ose y se tedy i v klidovém stavu zařízení projeví hodnota gravitačního zrych-

lení. Návrh uživatelského rozhraní sestává ze šesti prvků `TextView`. Tři z nich v levém sloupci zobrazují názvy os. Prvky v pravém sloupci budou zobrazovat hodnoty zrychlení v příslušné ose. Na rozmístění prvků jsme využili kontejner `RelativeLayout`.

TIP

Pokud byste vypisovali více než tři dvojice údajů, například pokud byste chtěli aplikaci rozšířit o výpis filtrovaných hodnot, bude výhodnější kontejner `TableLayout`.

Text legendy jsme v tomto případě umístili přímo do návrhu prvků. Názvy os jsou v zemích využívajících latinku mezinárodní. Pro zjednodušení jsou ve výpisu jen prvky týkající se osy x.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/RelativeLayout1"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/x_popis"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:padding="5dp"
    android:text="X: "
    android:textSize="20sp" />

  <TextView
    android:id="@+id/x_hodnota"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"
    android:layout_toRightOf="@+id/x_popis"
    android:padding="5dp"
    android:textSize="20sp" />

...

</RelativeLayout>
```

Prvním krokem při získávání údajů poskytovaných konkrétním senzorem je vytvoření instance (reference) služby:

```
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

Následně zjistíme přítomnost konkrétního senzoru v zařízení (v kladném případě můžeme následně získat/upravit jeho parametry):

```
if ((akcelerometr = sensorManager
  .getDefaultSensor(Sensor.TYPE_ACCELEROMETER))==null)
  finish();
```

Třetím krokem je registrace snímače událostí:

```
sensorManager.registerListener(this, akcelerometr,
  SensorManager.SENSOR_DELAY_UI);
  posledniAktualizace = System.currentTimeMillis();
```

Posledním krokem algoritmu je povinná implementace dvou metod rozhraní `SensorEventListener`:

```
@Override
public void onSensorChanged(SensorEvent event)
{}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy)
{}
```

Uvedená metoda je prováděna při změně přesnosti (`LOW`, `MEDIUM`, `HIGH`, `UNRELIABLE`) senzoru. Metoda se provádí v okamžiku, kdy senzor poskytne nové údaje, a to prostřednictvím objektu `SensorEvent`.

V kódu hlavní aktivity si všimněte, že listener pro senzor akcelerometru se pokaždé deaktivuje, aby se šetřila energie baterie. Hodnota zrychlení se načítá v metodě `onSensorChanged()`. Aby se zobrazené hodnoty daly přečíst, aktualizuje se hodnota každých 500 ms, tj. každou půlsekundu. V této jednoduché aplikaci změnu přesnosti, která volá metodu `onAccuracyChanged()`, ignorujeme.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends Activity implements SensorEventListener
{
  private static final int INTERVAL = 500;
  private SensorManager sensorManager;
  private Sensor akcelerometr;

  private TextView xHodnota, yHodnota, zHodnota;
  private long posledniAktualizace;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    xHodnota = (TextView) findViewById(R.id.x_hodnota);
    yHodnota = (TextView) findViewById(R.id.y_hodnota);
    zHodnota = (TextView) findViewById(R.id.z_hodnota);

    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    if ((akcelerometr = sensorManager
      .getDefaultSensor(Sensor.TYPE_ACCELEROMETER))==null)
      finish();
  }

  @Override
```

```
  protected void onResume()
  {
    super.onResume();

    sensorManager.registerListener(this, akcelerometr,
      SensorManager.SENSOR_DELAY_UI);
    posledniAktualizace = System.currentTimeMillis();
  }

  @Override
  protected void onPause()
  {
    sensorManager.unregisterListener(this);
    super.onPause();
  }

  @Override
  public void onSensorChanged(SensorEvent event)
  {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
    {
      long actualTime = System.currentTimeMillis();
      if (actualTime - posledniAktualizace > INTERVAL)
      {
        posledniAktualizace = actualTime;
        float x = event.values[0], y = event.values[1], z = event.values[2];
        xHodnota.setText(String.valueOf(x));
        yHodnota.setText(String.valueOf(y));
        zHodnota.setText(String.valueOf(z));
      }
    }
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy)
  {}
}
```

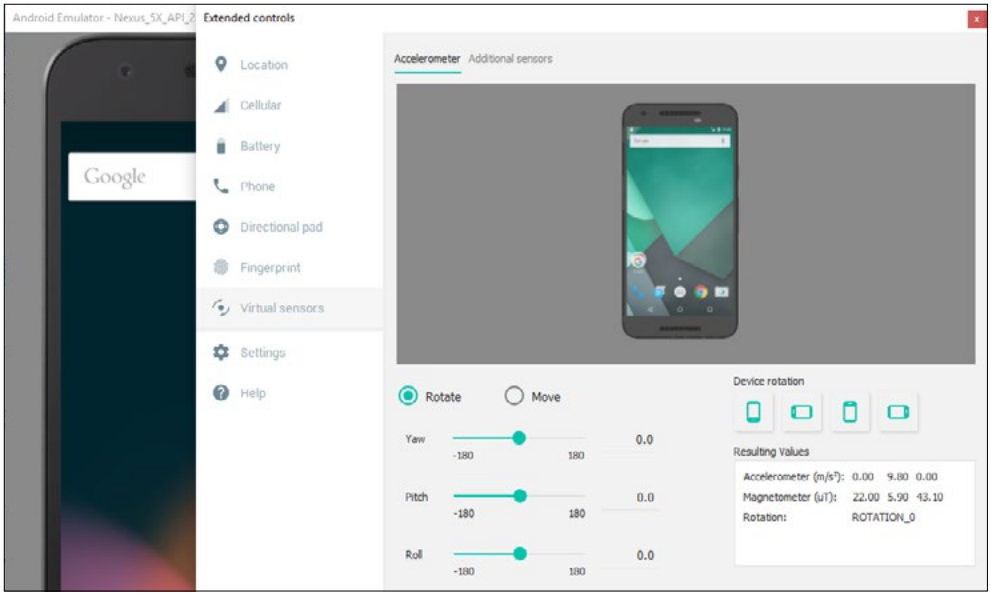
Nyní můžete aplikaci spustit. Zkuste nejprve pohybovat zařízením v poloze podle obrázku, na kterém je souřadnicový systém. Následně otáčejte zařízením a pozorujte, jak se mění hodnoty v jednotlivých osách a hlavně jejich znaménka.

POZNÁMKA

Poklud je zařízení v klidu, naměříte při použití senzoru `TYPE_ACCELEROMETER` v ose y hodnotu 9,81, což odpovídá gravitačnímu zrychlení.



Obrázek 13.2: Zobrazení výstupů z akcelerometru



Obrázek 13.3: Údaje ze senzorů je možné simulovat i na emulátoru, nejlépe se však aplikace tohoto typu testují na reálném zařízení

Příklad – Zobrazení filtrovaných údajů z akcelerometru

U předchozího příkladu platí modifikované úsloví, že za málo úsilí málo užitné hodnoty. Na tabletu by si pozornější čtenáři určitě na obrázku běhu aplikace všimli nesrovnalosti v hodnotách os x a y. Nesrovnalost vzniká, když je některý tablet, například Lenovo Yoga, obrácen na výšku. Na některých tabletech se totiž předpokládá základní orientace na šířku, například kvůli zabudovanému stojanu. U těchto přístrojů je souřadnicová soustava oproti telefonům natočená o 90 stupňů, to znamená, že osa y je ve směru kratší hrany. Proto hodnota blízká 9,81 bude na takovémto tabletu v ose x. Pokud bychom telefon nebo tablet pootočili o 45 stupňů, gravitační zrychlení se rovnoměrně rozdělí do os x a y.

Jak řešit tento problém, aby akcelerometr fungoval v každé poloze zařízení? U pokusů s aplikací v předchozí části jste asi zjistili, že se projevují dvě složky zrychlení: malé změny způsobené dynamickými otřesy a dlouhodobé pomalé změny, případně hodnoty posunuté o konstantu gravitačního zrychlení. Abychom mohli tyto hodnoty separovat, je účelné aplikovat na údaje z akcelerometru dva filtry: filtr typu horní propust, který přenese jen krátkodobé změny, například otřesy, a dolní propust, který naopak tyto otřesy zadrží.

Gravitační složka zrychlení se v jednotlivých osách vypočítá porovnáním aktuální a předchozí hodnoty.

Uživatelské rozhraní bude modifikací předchozího, jenom kromě „surových" hodnot zrychlení pro jednotlivé osy snímaných přímo z akcelerometru zobrazíme i hodnoty po aplikaci filtrů LowPass a HighPass. Použijeme kontejner `TableLayout`. Ve výpisu jsou kvůli úspoře místa jen prvky na zobrazení legendy a hodnot pro osu x.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <TableLayout
    android:id="@+id/tableLayout1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <TableRow
      android:id="@+id/tableRow1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" >

      <TextView
        android:text="X:"
        android:textSize="20sp" />
      <TextView
        android:id="@+id/x_hodnota"
        android:textSize="20sp" />
    </TableRow>

    <TableRow
      android:id="@+id/tableRow4"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" >
      <TextView
        android:text="LowPass X"
        android:textSize="20sp" />
      <TextView
        android:id="@+id/x_lowpass"
        android:textSize="20sp" />
    </TableRow>

    <TableRow
      android:id="@+id/tableRow7"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" >
      <TextView
        android:text="HighPass X"
        android:textSize="20sp" />
      <TextView
        android:id="@+id/x_highpass"
```

```
        android:textSize="20sp" />
    </TableRow>

</LinearLayout>
```

V kódu hlavní aktivity budeme do polí `aAkcelerace` a `aGravitace` ukládat pro jednotlivé osy hodnoty zrychlení a gravitačního zrychlení.

Metody na aplikaci filtru horní a dolní propusti jsou poměrně jednoduché. Filtr LowPass vrací takzvaný vážený průměr, kde je 80 procent dlouhodobé hodnoty a 20 procent aktuálně načítané hodnoty:

```
private float lowPass(float aktualni, float gravitace)
{
  return gravitace * 0.8f + aktualni * (1 - 0.8f);
}

private float highPass(float aktualni, float gravitace)
{
  return aktualni - gravitace;
}
```

Kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity
  implements SensorEventListener {

  private SensorManager sensorManager;
  private Sensor akcelerometer;

  private float[] aAkcelerace = new float[3];
  private float[] aGravitace = new float[3];

  private TextView tvX, tvY,tvZ, tvXhi, tvYhi,tvZhi, tvXlo, tvYlo,tvZlo ;

  private long aktualizace;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    tvX = (TextView) findViewById(R.id.x_hodnota);
    tvY = (TextView) findViewById(R.id.y_hodnota);
    tvZ = (TextView) findViewById(R.id.z_hodnota);

    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    if ((akcelerometer = sensorManager
      .getDefaultSensor(Sensor.TYPE_ACCELEROMETER))==null)
      finish();
    aktualizace = System.currentTimeMillis();
  }
```

```
@Override
protected void onResume()
{
  super.onResume();
  sensorManager.registerListener(this, akcelerometer,
    SensorManager.SENSOR_DELAY_UI);
}

@Override
protected void onPause()
{
  super.onPause();
  sensorManager.unregisterListener(this);
}

@Override
public void onSensorChanged(SensorEvent event)
{
  if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
  {
    long actualTime = System.currentTimeMillis();
    if (actualTime - aktualizace > 500)
    {
      aktualizace = actualTime;
      float rawX = event.values[0];
      float rawY = event.values[1];
      float rawZ = event.values[2];

      //low-pass filter
      aGravitace[0] = lowPass(rawX, aGravitace[0]);
      aGravitace[1] = lowPass(rawY, aGravitace[1]);
      aGravitace[2] = lowPass(rawZ, aGravitace[2]);

      //high-pass filter
      aAkcelerace[0] = highPass(rawX, aGravitace[0]);
      aAkcelerace[1] = highPass(rawY, aGravitace[1]);
      aAkcelerace[2] = highPass(rawZ, aGravitace[2]);

      tvX.setText(String.valueOf(rawX));
      tvY.setText(String.valueOf(rawY));
      tvZ.setText(String.valueOf(rawZ));

      tvXhi.setText(String.valueOf(aGravitace[0]));
      tvYhi.setText(String.valueOf(aGravitace[1]));
      tvZhi.setText(String.valueOf(aGravitace[2]));

      tvXlo.setText(String.valueOf(aAkcelerace[0]));
      tvYlo.setText(String.valueOf(aAkcelerace[1]));
      tvZlo.setText(String.valueOf(aAkcelerace[2]));
    }
```

```
    }
  }

  private float lowPass(float aktualni, float gravitace)
  {
    return gravitace * 0.8f + aktualni * (1 - 0.8f);
  }

  private float highPass(float aktualni, float gravitace)
  {
    return aktualni - gravitace;
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy)
  {}
}
```

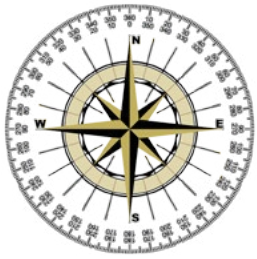
Jako ilustraci netradičního využití akcelerometru můžete vytvořit aplikaci, která promění tablet na pomůcku fitness, například na procvičení ztuhlých rukou při práci s počítačem. Stačí držet tablet v předpažených rukách displejem k sobě a pohybovat jím od sebe a k sobě, a to co nejrychleji. Na základě údajů z akcelerometru (osa z) se na jednoduchých stupnicích zobrazuje aktuální zrychlení, akumulované zrychlení a časový postup.

Příklad – Magnetický kompas

Aplikace typu „kompas" využívají polovodičový senzor na měření magnetického pole. Pomocí tohoto senzoru je možné určit nejen orientaci zařízení vůči severu, tedy severnímu magnetickému pólu. Na prezentování se nejčastěji používá kruhová stupnice, uvnitř které je obrázek tzv. magnetického kříže, který přehledně ukazuje hlavní světové strany. Na otáčení kruhu můžete využít například animaci – rotaci obrázku tak, aby stupnice ukazovala správnou hodnotu vůči pevnému bodu.

Samotný snímač geomagnetického pole poskytuje naměřenou hodnotu fyzikální veličiny v jednotkách μT (mikro Tesla) pro každou ze tří os souřadnicové soustavy. S výjimkou přímého měření intenzity magnetického pole nemůžete použít tento snímač přímo. Pro aplikace typu kompas můžete využít vektor rotace na určení rotačního pohybu nebo můžete použít údaje z akcelerometru a snímače geomagnetického pole s využitím metody `getRotationMatrix()` na získání matice rotace, případně i sklonu. Následně můžete použít tyto matice a pomocí metod `getOrientation()` a `getInclination()` získat azimut a údaje o sklonu.

Do složky *drawable* překopírujte vhodný obrázek kompasové růžice.



Obrázek 13.4: Obrázek kompasové růžice

Layout hlavní aktivity obsahuje jen prázdný kontejner `RelativeLayout` s bílým pozadím.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:background="#ffffffff"
  android:id="@+id/frame">
</RelativeLayout>
```

Vytvořte třídu na zobrazení ciferníku kompasu. Její kód můžete kvůli přehlednosti umístit do souboru *MainActivity.java*.

```
public class CifernikView extends View
{
  int sirka, vyska, stredX, stredY, topX, leftY;

  Bitmap bmp = BitmapFactory.decodeResource(getResources(),
    R.drawable.kompas1);
  int bmpSirka = bmp.getWidth();
  int bmpVyska = bmp.getHeight();

  public CifernikView(Context context)
  {
    super(context);
  };

  @Override
  protected void onSizeChanged(int w, int h, int oldw, int oldh)
  {
    sirka = layout.getWidth();
    vyska = layout.getHeight();
    stredX = sirka / 2;
    stredY = vyska / 2;
    leftY = stredX - bmpSirka / 2;
    topX = stredY - bmpVyska / 2;
  }

  @Override
  protected void onDraw(Canvas canvas)
  {
    canvas.save();
    canvas.rotate((float) -natoceniStupne, stredX, stredY);
```

```
      canvas.drawBitmap(bmp, leftY, topX, null);
      canvas.restore();
    }
  }
```

V kódu hlavní aktivity je potřeba zaregistrovat dva senzory: snímač intenzity magnetického pole a akcelerometr. V metodě `onSensorChanged` snímáme zrychlení a magnetické pole. Z naměřených hodnot se vypočítá takzvaná rotační matice.

```
SensorManager.getRotationMatrix(rotMatrix, null, zrychleni, magnet);
```

Z rotační matice zjistíme orientaci a následně úhel natočení v radiánech. K rotaci bitmapy je potřeba konvertovat hodnotu úhlu na stupně.

```
float orientMatrix[] = new float[3];
SensorManager.getOrientation(rotMatrix, orientMatrix);
float radiany = orientMatrix[0];
natoceniStupne = Math.toDegrees(radiany);
```

Kompletní kód hlavní aktivity:

```
public class MainActivity extends Activity implements SensorEventListener
{
  private RelativeLayout layout;
  private CifernikView cifernik;

  private SensorManager sm;
  private Sensor aMeter;
  private Sensor mMeter;

  private float[] zrychleni = null;
  private float[] magnet = null;
  private double natoceniStupne;

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    layout = (RelativeLayout) findViewById(R.id.frame);
    cifernik = new CifernikView(getApplicationContext());
    layout.addView(cifernik);

    sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    aMeter = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    mMeter = sm.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
    if (aMeter==null || mMeter==null) finish();
  }

  @Override
  protected void onResume()
  {
    super.onResume();
    sm.registerListener(this, aMeter, SensorManager.SENSOR_DELAY_NORMAL);
```

```
    sm.registerListener(this, mMeter, SensorManager.SENSOR_DELAY_NORMAL);
  }

  @Override
  protected void onPause()
  {
    super.onPause();
    sm.unregisterListener(this);
  }

  @Override
  public void onSensorChanged(SensorEvent event)
  {
    if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER)
    {
      zrychleni = new float[3];
      System.arraycopy(event.values, 0, zrychleni, 0, 3);
    }

    else if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD)
    {
      magnet = new float[3];
      System.arraycopy(event.values, 0, magnet, 0, 3);
    }

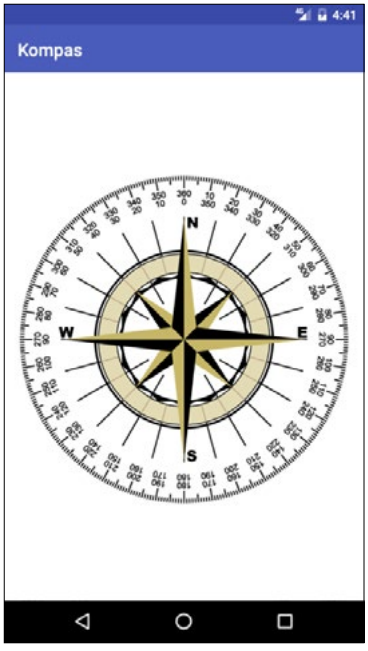
    if (zrychleni != null && magnet != null)
    {
      float rotMatrix[] = new float[9];

      boolean bOK = SensorManager.getRotationMatrix(rotMatrix,
        null, zrychleni, magnet);

      if (bOK)
      {
        float orientMatrix[] = new float[3];
        SensorManager.getOrientation(rotMatrix, orientMatrix);
        float radiany = orientMatrix[0];
        natoceniStupne = Math.toDegrees(radiany);
        cifernik.invalidate();
        zrychleni = magnet = null;
      }
    }
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy)
  {}

}
```



Obrázek 13.5: Test aplikace

Senzor osvětlení

Pokud tablet disponuje senzorem měřícím intenzitu osvětlení, které implicitně využívá k automatické regulaci jasu displeje, dá se využít i v aplikacích. Například může reagovat na to, že uživatel vyšel z místnosti ven na sluneční světlo nebo opačně, a přizpůsobit tomu barevný motiv aplikace, případně vhodně zvýšit jas nebo kontrast mezi pozadím a písmem, podobně jako to dělají třeba navigace GPS po setmění.

Jako nejjednodušší aplikace se nabízí luxmetr, tedy měřič intenzity osvětlení. Dá se použít například při sledování ergonometrie pracovního prostředí, zda má žák nebo student v pokoji dostatek světla na čtení a podobně. Princip aplikace je triviální – stačí periodicky snímat údaje ze snímače a zobrazit je v číselné, případně i analogové podobě.

Princip lze dále rozšiřovat, například o automatické zhotovení fotografie, když někdo v místnosti rozsvítí, případně jako dočasný optický alarm pro vystavený exemplář, který bude chránit tablet nenápadně umístěný vedle předmětu, přičemž opodstatněnost tabletu se automaticky odůvodní tím, že je na něm popis exponátu.

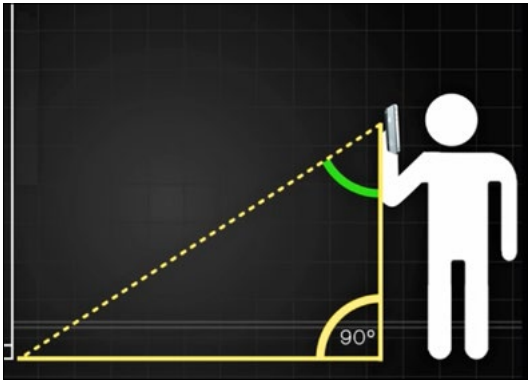
Náklonoměr

Triviální je využití náklonoměru buď v aplikaci typu úhloměr, nebo se dá tablet s grafickým znázorněním proměnit v dvouosou vodováhu. S vhodnou aplikací se dá tablet využít i na měření vzdálenosti, přičemž výhodou není přesnost, ale operativnost. Jak to provést? Tablet disponuje kamerou a náklonoměrem. Princip názorně vysvětluje obrázek.

Předpokládáme, že znáte svou výšku. Od ní odečtěte přibližně 10 cm, jelikož se předpokládá, že budete zařízení držet ve výšce očí. Následně zaměřte kameru (pomocí zaměřovacího kříže)

na měřený objekt vodorovně, například na stěnu budovy ve výšce vašich očí. Vodorovnost si zařízení zjistí pomocí náklonoměru a vhodně ji na obrazovce indikuje. Proveďte první měření.

Následně skloňte zařízení tak, abyste záměrným bodem mířili na spodní okraj objektu, a proveďte druhé měření. Zařízení si změří úhel sklonu. Zbytek už je jen školní geometrie o pravoúhlém trojúhelníku, kdy je potřeba vypočítat jednu odvěsnu, když znáte druhou odvěsnu a úhel mezi odvěsnou a přeponou.



Obrázek 13.6: Princip měření vzdálenosti tabletem

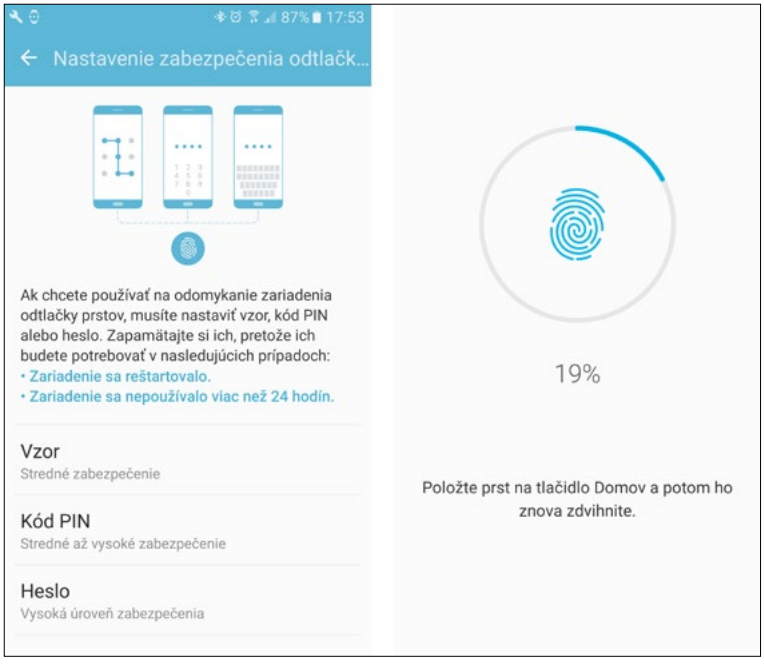
Rozměry menších předmětů můžete změřit tak, že je položíte na displej tabletu a ve vhodné aplikaci přemístíte kurzory tak, aby lícovaly s hranami měřeného předmětu.

Zní to na první pohled neuvěřitelně, ale tablet s vhodnou aplikací dokáže změřit i tepovou frekvenci. Princip snímání je stejný jako ten, který používají „kolíčkové" snímače na stacionárních bicyklech a jiných fitness zařízeních. Při pumpování krve do vlásečnic se během každého tepu mění průsvitnost vašeho prstu. Položíte prst na kameru na zadní straně tabletu – aplikace snímá a vyhodnocuje změnu průsvitnosti vašeho prstu pro vnější světlo.

Snímač otisků prstů

Ověřování otisků prstů využívá dotykový snímač moderních zařízení s Androidem. Podporované je od verze API 23: Android 6.0 (Marshmallow). Nejčastěji je snímač umístěn pod displejem, kde může, ale nemusí být integrován do hardwarového tlačítka, případně na zadní straně pod objektivem hlavního fotoaparátu. Také může být tento snímač integrovaný s tlačítkem nebo plochou, po které můžete posouvat prst.

Identifikace pomocí otisků prstů je vícestupňový proces, který je implementovaný na úrovni operačního systému. Údaje o vzorech na porovnávání otisků jsou uloženy šifrovaně. Ověření pomocí otisku prstu je samozřejmě dostupné jen v zařízeních, která obsahují snímač a na kterých byla nastavena konfigurace zabezpečení a nasnímaný a uložený aspoň jeden otisk prstu.



Obrázek 13.7: Konfigurace zabezpečení a snímání vzoru otisku

Pokud chcete nakonfigurovat ověřování otisků prstů na fyzickém zařízení, spusťte **Nastavení** a vyberte možnost přihlašování pomocí otisků prstu. Nejprve je potřeba nakonfigurovat alternativní autentizaci, například pomocí kódu PIN. Následně nasnímejte jeden nebo více vzorů otisků vašich prstů.

Do aplikace, která bude využívat tento způsob autentizace, je potřeba do aplikačního manifestu přidat řádek:

```
<uses-permission android:name="android.permission.USE_FINGERPRINT" />
```

Aplikace by měly akceptovat standardy a jedním z nich je doporučená ikona na autentizaci pomocí otisku prstu. Vhodný obrázek vložte do složky *Drawable*.

UPOZORNĚNÍ

Při vytváření projektu musíte nastavit verzi Android 6.0 Marshmallow nebo vyšší.



Obrázek 13.8: Doporučená ikona na autentizaci pomocí otisku prstu

Naše aplikace bude mít v layoutu hlavní aktivity pouze tuto ikonu.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.odtlacokprsta.MainActivity">

  <ImageView
    android:layout_width="150dp"
    android:layout_height="150dp"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:scaleType="fitCenter"
    android:src="@drawable/ico_fp_40px"/>
</RelativeLayout>
```

Ověřování otisku využívá dvě systémové služby: `KeyguardManager` a `FingerprintManager`.

```
keyguardManager =
  (KeyguardManager) getSystemService(KEYGUARD_SERVICE);
fingerprintManager =
  (FingerprintManager) getSystemService(FINGERPRINT_SERVICE);
```

Následně je potřeba ověřit, zda je autentizace pomocí otisků prstů povolená a zda je nastaven alespoň jeden otisk. Ověřování otisků prstů zahrnuje generování šifrovacího klíče, který je bezpečně uložen v zařízení v systémovém úložišti klíčů. Aplikace potřebuje povolení k přístupu k úložišti klíčů.

```
protected void generateKey() {
  try {
    keyStore = KeyStore.getInstance("AndroidKeyStore");
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

Následuje generování klíče pomocí služby `KeyGenerator`. Před generováním je potřeba získat odkaz na instanci `KeyGenerator`.

```
try {
  keyGenerator = KeyGenerator.getInstance(
    KeyProperties.KEY_ALGORITHM_AES,
    "AndroidKeyStore");
```

```
} catch (NoSuchAlgorithmException | NoSuchProviderException e) {
  throw new RuntimeException(
    "Probblém inštancie KeyGenerator", e);
}
```

Nyní může aplikace vygenerovat klíč.

```
try {
  keyStore.load(null);
  SecretKey key = (SecretKey) keyStore.getKey(KEY_NAME, null);
  cipher.init(Cipher.ENCRYPT_MODE, key);
  return true;
} catch (KeyPermanentlyInvalidatedException e) {
  return false;
} catch (KeyStoreException | CertificateException
  | UnrecoverableKeyException | IOException
  | NoSuchAlgorithmException | InvalidKeyException e) {
  throw new RuntimeException("Cipher ERR", e);
}
```

Do řetězcové proměnné `"my_key"` se zapíše klíč, který je v kontejneru úložiště klíčů. Tento klíč se použije k šifrování a dešifrování a nastavení různých parametrů šifrování. Šifrování řeší `CryptoObject` v metodě `cipherInit()`.

Pokud to shrneme, zatím jsme popisovali přípravné fáze, získávání a vytváření objektů na šifrování a s ním související klíče. Na první pohled to vypadá složitě, ale je to poměrně jednoduchá, a navíc logická posloupnost kroků. Nyní však vstupujeme do finále, kterým je autentizace pomocí volání metody `authenticate` instance `FingerprintManager`. Toto volání spustí ještě složitější posloupnost kroků, kterou však nemusíme řešit, protože se realizuje na úrovni operačního systému.

Výsledkem v případě úspěchu i neúspěchu je callback. Metody je potřeba implementovat v třídě, která rozšiřuje třídu `FingerprintManager.AuthenticationCallback`. V našem případě jsme respektovali tradici, přesněji popis autentizace na *https://developer.android.com*, a třídu jsme nazvali `FingerprintHandler`.

Kompletní kód třídy `FingerprintHandler`:

```
public class FingerprintHandler extends
  FingerprintManager.AuthenticationCallback {

  private CancellationSignal cs;
  private Context aContext;

  public FingerprintHandler(Context context) {
    aContext = context;
  }

  public void startAuth(FingerprintManager fm,
    FingerprintManager.CryptoObject cryptoObject) {

    cs = new CancellationSignal();

    if (ActivityCompat.checkSelfPermission(aContext,
```

```
        Manifest.permission.USE_FINGERPRINT) !=
        PackageManager.PERMISSION_GRANTED) {
        return;
      }
      fm.authenticate(cryptoObject, cs, 0, this, null);
    }

    @Override
    public void onAuthenticationError(int errMsgId,
      CharSequence errString) {
      Toast.makeText(aContext,
        "Chyba autentifikácie\n" + errString,
        Toast.LENGTH_LONG).show();
    }

    @Override
    public void onAuthenticationHelp(int helpMsgId,
      CharSequence helpString) {
      Toast.makeText(aContext,
        "Nápoveda\n" + helpString,
        Toast.LENGTH_LONG).show();
    }

    @Override
    public void onAuthenticationFailed() {
      Toast.makeText(aContext,
        "Chyba autentifikácie.",
        Toast.LENGTH_LONG).show();
    }

    @Override
    public void onAuthenticationSucceeded(
      FingerprintManager.AuthenticationResult result) {

      Toast.makeText(aContext,
        "Autentifikácia OK.",
        Toast.LENGTH_LONG).show();
    }
  }
```

A nyní kód hlavní aktivity. Pro úplnost uvádíme nejprve i seznam referencí.

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.app.KeyguardManager;
import android.hardware.fingerprint.FingerprintManager;
import android.widget.Toast;
import android.Manifest;
import android.content.pm.PackageManager;
import android.support.v4.app.ActivityCompat;
import android.security.keystore.KeyProperties;
import android.security.keystore.KeyGenParameterSpec;
import android.security.keystore.KeyPermanentlyInvalidatedException;
```

```
import java.security.KeyStore;
import java.security.NoSuchAlgorithmException;
import java.security.NoSuchProviderException;
import java.security.cert.CertificateException;
import java.security.InvalidAlgorithmParameterException;
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.KeyStoreException;
import java.security.UnrecoverableKeyException;

import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.Cipher;
```

Kompletní kód hlavní aktivity. Vzhledem k ne úplně triviální souvislosti a posloupnosti kroků je výpis kompletní, přestože je trochu delší.

```
public class MainActivity extends AppCompatActivity {
  private static final String KEY_NAME = "my_key";
  private FingerprintManager fm;
  private KeyguardManager km;
  private KeyStore keyStore;
  private KeyGenerator kg;
  private Cipher cipher;
  private FingerprintManager.CryptoObject co;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    km = (KeyguardManager) getSystemService(KEYGUARD_SERVICE);
    fm = (FingerprintManager) getSystemService(FINGERPRINT_SERVICE);

    if (!km.isKeyguardSecure()) {
      Toast.makeText(this,
        "Nakonfigurujte odtlačok prsta v Nastavení",
        Toast.LENGTH_LONG).show();
      return;
    }

    if (ActivityCompat.checkSelfPermission(this,
      Manifest.permission.USE_FINGERPRINT) !=
        PackageManager.PERMISSION_GRANTED) {
      Toast.makeText(this,
        "Autentifikácia odtlačkom nie je povolená",
        Toast.LENGTH_LONG).show();
      return;
    }

    if (!fm.hasEnrolledFingerprints()) {
```

```
    Toast.makeText(this,
      "Zaregistrujte aspoň jeden odtlačok",
      Toast.LENGTH_LONG).show();
    return;
  }

  generateKey();

  if (cipherInit()) {
    co = new FingerprintManager.CryptoObject(cipher);
    FingerprintHandler helper = new FingerprintHandler(this);
    helper.startAuth(fm, co);
  }
}

public boolean cipherInit() {
  try {
    cipher = Cipher.getInstance(
      KeyProperties.KEY_ALGORITHM_AES + "/"
      + KeyProperties.BLOCK_MODE_CBC + "/"
      + KeyProperties.ENCRYPTION_PADDING_PKCS7);
  } catch (NoSuchAlgorithmException |
    NoSuchPaddingException e) {
      throw new RuntimeException("Cipher ERR", e);
  }

  try {
    keyStore.load(null);
    SecretKey key = (SecretKey) keyStore.getKey(KEY_NAME, null);
    cipher.init(Cipher.ENCRYPT_MODE, key);
    return true;
  } catch (KeyPermanentlyInvalidatedException e) {
    return false;
  } catch (KeyStoreException | CertificateException
    | UnrecoverableKeyException | IOException
    | NoSuchAlgorithmException | InvalidKeyException e) {
    throw new RuntimeException("Cipher ERR", e);
  }
}

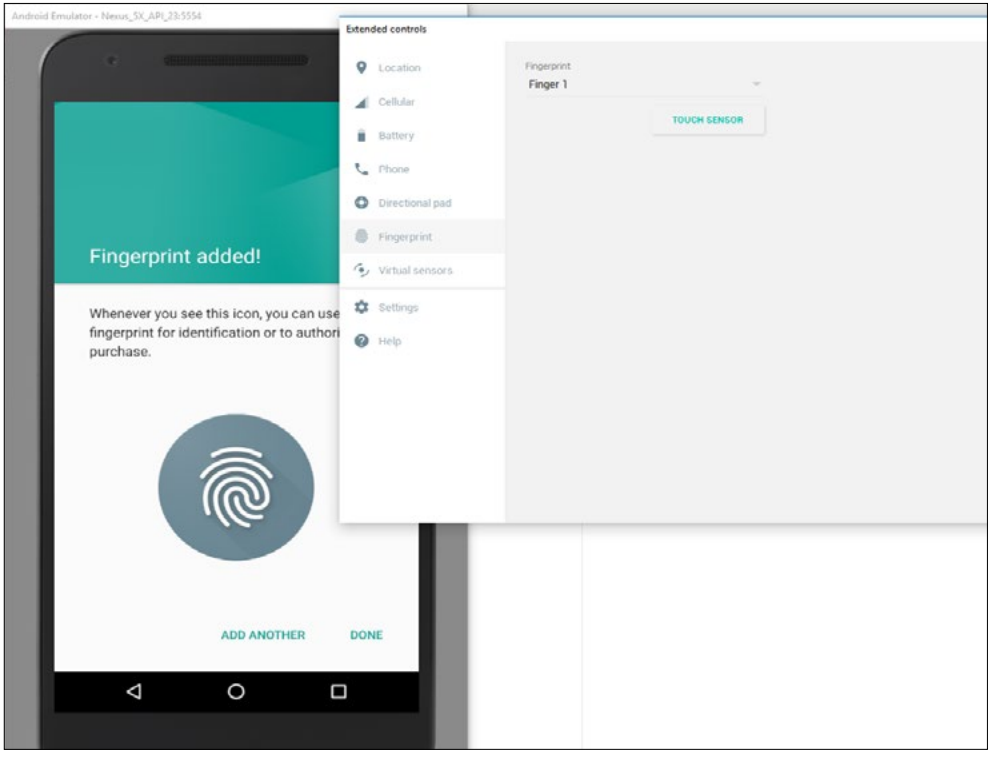
protected void generateKey() {
  try {
    keyStore = KeyStore.getInstance("AndroidKeyStore");
  } catch (Exception e) {e.printStackTrace();}

  try {
    kg = KeyGenerator.getInstance(
      KeyProperties.KEY_ALGORITHM_AES,
      "AndroidKeyStore");
  } catch (NoSuchAlgorithmException |
    NoSuchProviderException e) {
    throw new RuntimeException(
```

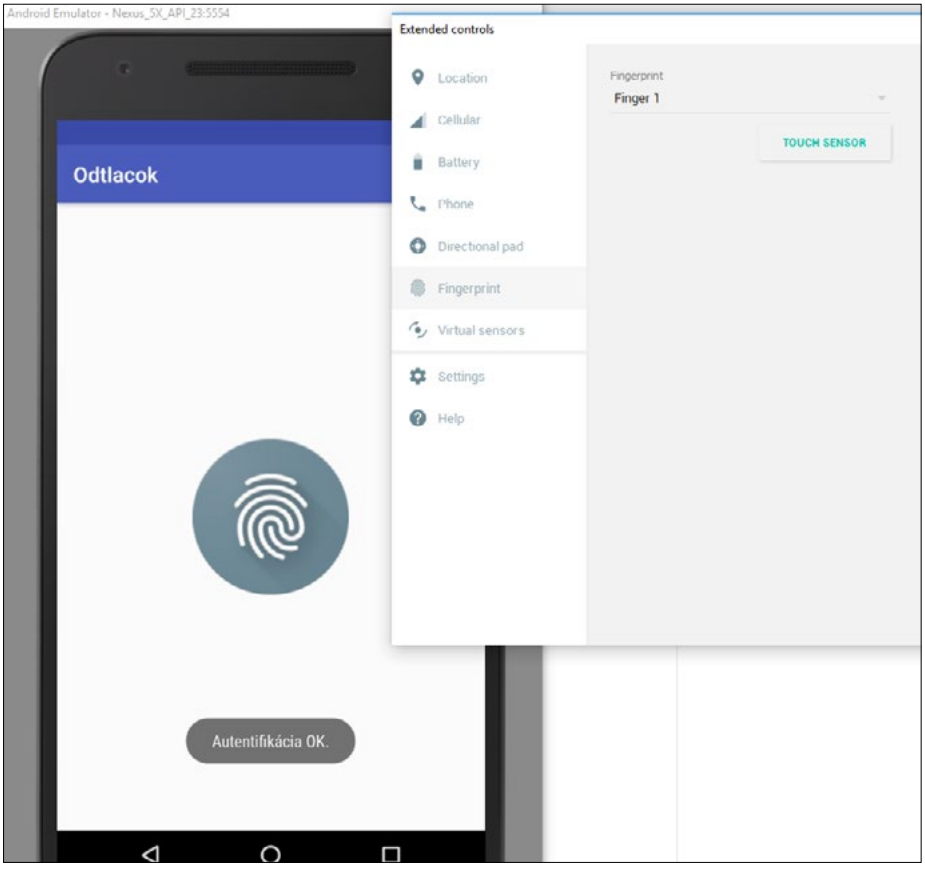
```
            "Problém inštancie KeyGenerator", e);
        }

        try {
          keyStore.load(null);
          kg.init(new
            KeyGenParameterSpec.Builder(KEY_NAME,
            KeyProperties.PURPOSE_ENCRYPT |
            KeyProperties.PURPOSE_DECRYPT)
            .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
            .setUserAuthenticationRequired(true)
            .setEncryptionPaddings(
            KeyProperties.ENCRYPTION_PADDING_PKCS7)
            .build());
            kg.generateKey();
        } catch (NoSuchAlgorithmException |
          InvalidAlgorithmParameterException
          | CertificateException | IOException e) {
          throw new RuntimeException(e);
        }
      }
    }
```

Nyní můžete aplikaci vyzkoušet. Dá se to i na emulátoru, avšak i zde musíte autentizaci nastavit.



Obrázek 13.9: Nastavení autentizace pomocí otisku prstů na emulátoru



Obrázek 13.10: Testování autentizace v aplikaci pomocí otisku prstu na emulátoru

Komunikace přes Bluetooth

V současnosti, především v souvislosti s internetem, roste potřeba komunikace zařízení s Androidem, ať už jsou to telefony, tablety, nebo wearables, s různými externími zařízeními, ať už jsou to komerčně vyráběné produkty, nebo různé amatérské prototypy zařízení, například roboti v rámci studentských projektů, hobby aktivit a podobně. Jednou z metod komunikace je rozhraní Bluetooth. Bluetooth zařízení se musí nejprve najít a spárovat se zařízením s Androidem přes nastavení operačního systému.

Android proto obsahuje podporu pro framework Bluetooth, který umožňuje aplikacím pro Android na zařízeních s podporou Bluetooth bezdrátově vyměňovat data s jinými zařízeními s Bluetooth.

API Bluetooth umožňuje provádět několik typů operací:

- Vyhledávat zařízení Bluetooth
- Získat seznam spárovaných zařízení
- Připojit se k dalším zařízením pomocí služby discovery

Na zařízení musí být samozřejmě povolena komunikace přes Bluetooth. Aplikace proto může umožnit zapnout Bluetooth přes intent s konstantou `ACTION_REQUEST_ENABLE`. Jeho syntaxe je následující:

```
Intent turnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(turnOn, 0);
```

Android poskytuje ke komunikaci přes toto rozhraní třídu `BluetoothAdapter`. Její instance se vytváří voláním statické metody `getDefaultAdapter()`. Syntaxe volání metody je následující:

```
private BluetoothAdapter BA;
BA = BluetoothAdapter.getDefaultAdapter();
```

Nejčastěji budete využívat metody:

- `enable()` – metoda povolí adaptér Bluetooth, pokud není povolen
- `isEnabled()` – metoda vrací hodnotu `true`, pokud je adaptér Bluetooth povolen
- `disable()` – metoda zakáže adaptér Bluetooth
- `getName()` – metoda vrátí název adaptéru Bluetooth
- `setName(String name)` – metoda změní název Bluetooth
- `getState()` – metoda vrací aktuální stav adaptéru Bluetooth
- `startDiscovery()` – metoda aktivuje proces zjišťování zařízení Bluetooth během 120 sekund

Vytvořte projekt aplikace pro Android s hlavní aktivitou typu **Empty Activity**. Pro názornost jsou textové řetězce přímo v návrhovém kódu. V reálné aplikaci by byly v souboru *strings.xml*. Definice uživatelského rozhraní aktivity:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="com.example.llacko.bluetooth.MainActivity">

  <TextView android:text="Bluetooth test" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textview"
    android:textSize="35dp"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />

  <ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/bt"
```

```
    android:layout_below="@+id/textview"
    android:layout_centerHorizontal="true"
    android:theme="@style/Base.TextAppearance.AppCompat" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Zapni"
    android:id="@+id/button"
    android:layout_below="@+id/imageView"
    android:layout_toStartOf="@+id/imageView"
    android:layout_toLeftOf="@+id/imageView"
    android:clickable="true"
    android:onClick="on" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Viditeľné"
    android:onClick="visible"
    android:id="@+id/button2"
    android:layout_alignBottom="@+id/button"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Zar"
    android:onClick="list"
    android:id="@+id/button3"
    android:layout_below="@+id/imageView"
    android:layout_toRightOf="@+id/imageView"
    android:layout_toEndOf="@+id/imageView" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Off"
    android:onClick="off"
    android:id="@+id/button4"
    android:layout_below="@+id/button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

<ListView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/listView"
    android:layout_alignParentBottom="true"
    android:layout_alignLeft="@+id/button"
    android:layout_alignStart="@+id/button"
```

```
    android:layout_below="@+id/textView2" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Párované zariadenia:"
    android:id="@+id/textView2"
    android:textSize="25dp"
    android:layout_below="@+id/button4"
    android:layout_alignLeft="@+id/listView"
    android:layout_alignStart="@+id/listView" />

</RelativeLayout>
```



Obrázek 13.11: Uživatelské rozhraní aplikace

V manifestu Android je potřeba deklarovat příslušná povolení:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.llacko.bluetooth">
  <uses-permission android:name="android.permission.BLUETOOTH"/>
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
```

```
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Nejdůležitější je samozřejmě kód hlavní aktivity:

```
public class MainActivity extends AppCompatActivity {
  Button b1,b2,b3,b4;
  private BluetoothAdapter BA;
  private Set<BluetoothDevice> pairedDevices;
  ListView lv;


  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    b1 =(Button) findViewById(R.id.button);
    b2=(Button)findViewById(R.id.button2);
    b3=(Button)findViewById(R.id.button3);
    b4=(Button)findViewById(R.id.button4);

    BA = BluetoothAdapter.getDefaultAdapter();
    lv = (ListView)findViewById(R.id.listView);
  }

  public void on(View v){
    if (!BA.isEnabled()) {
      Intent turnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      startActivityForResult(turnOn, 0);
      Toast.makeText(getApplicationContext(),"Zapnutý",
        Toast.LENGTH_LONG).show();
    }
    else
    {
      Toast.makeText(getApplicationContext(),"Už je zapnutý",
        Toast.LENGTH_LONG).show();
    }
  }

  public void off(View v){
    BA.disable();
    Toast.makeText(getApplicationContext(),"Vypnutý",
      Toast.LENGTH_LONG).show();
```

```
    }

    public  void visible(View v){
      Intent getVisible = new
        Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
      startActivityForResult(getVisible, 0);
    }

    public void list(View v){
      pairedDevices = BA.getBondedDevices();
      ArrayList list = new ArrayList();

      for(BluetoothDevice bt : pairedDevices)
      list.add(bt.getName());
      Toast.makeText(getApplicationContext(),"Zobrazujem párované zariadenia",
        Toast.LENGTH_SHORT).show();

      final ArrayAdapter adapter = new
        ArrayAdapter(this,android.R.layout.simple_list_item_1, list);
      lv.setAdapter(adapter);
    }

}
```

Nyní můžete aplikaci s vhodným zařízením Bluetooth vyzkoušet.

KAPITOLA

14

Mapy a navigace

Témata kapitoly:

- Lokalizační a mapové služby
- Zobrazení polohy na mapě

Lokalizační a mapové služby

Přístroje platformy Android, hlavně telefony, disponují obvody pro příjem signálu GPS z družic, takže po propojení s mapovými službami není problém vytvářet aplikace typu navigace, tracker, geofence (na sledování pohybu dětí a starších lidí) a podobně.

POZNÁMKA

Lokalizační služby dodávají mobilním aplikacím novou dimenzi, jelikož aplikace může poskytovat uživateli informace vzhledem k jeho momentální poloze.

Nejsou to jen aplikace na navigaci, ale například aplikace s přehledem jídelních lístků zobrazí nabídku restaurací nabízejících vybrané jídlo seřazenou od nejbližší vzhledem k aktuální poloze uživatele.

Android nabízí třídu `Location`, která reprezentuje polohu. Instance třídy `Location` obsahuje údaje o zeměpisné délce a šířce a aktuálním čase. Také obsahuje informaci o aktuální přesnosti polohy, nadmořské výšce, rychlosti, azimutu a podobně.

- `LocationProvider` – zařízení s Androidem mohou určovat aktuální polohu nejen ze signálu GPS, ale i triangulací stanic GSM či z přístupových bodů Wi-Fi. Aplikace, které používají lokalizační služby, musí požádat o povolení poskytnutí údajů o poloze. Android má dvě úrovně oprávnění:
  - `android.permission.ACCESS_COARSE_LOCATION` – přibližná poloha, zhruba na úrovni městského bloku
  - `android.permission.ACCESS_FINE_LOCATION` – přesná poloha
- `LocationManager` – systémová služba na získání informací o poloze. Umožňuje určit poslední známou polohu uživatele, případně informaci, když se zařízení blíží

nebo vzdaluje od určité geografické oblasti. Instanci objektu `LocationManager` získáte pomocí metody `getSystemService(Context.LOCATION_SERVICE)`.

- `LocationListener` volá metody callback, když se změní stav objektů typu `Location` nebo `LocationProvider`:
  - `onLocationChanged()` se volá v případě, že se změní poloha. Údaje o aktuální poloze jsou objekty typu Location. Můžete zjistit zeměpisnou šířku (latitude), zeměpisnou délku (longitude), vzdálenost od jiné lokace (distance), výšku (altitude) a azimut (bearing), tj. úhel od severu.
  - `onProviderEnabled()` – když uživatel zapne/povolí vyžadovaný provider
  - `onProviderDisabled()` – když uživatel vypne/zakáže vyžadovaný provider
  - `onStatusChanged()` – když se změní stav poskytovatele

Příklad – Určení polohy zařízení

V prvním triviálním příkladu zobrazíme pouze zeměpisné souřadnice zjištěné přes GPS. Proto návrh layoutu hlavní aktivity obsahuje jen dva prvky `TextView`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <TextView
    android:id="@+id/latitude"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
  </TextView>
  <TextView
    android:id="@+id/longitude"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
  </TextView>

</LinearLayout>
```

V *AndroidManifest.xml* deklarujte oprávnění aplikace přistupovat k geografické poloze a k internetu.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

V kódu hlavní aktivity budeme periodicky zjišťovat a vypisovat zeměpisné souřadnice, samozřejmě pouze v případě, že to nastavení zařízení umožní.

Informace o poloze bude aplikace zjišťovat přes metodu `requestLocationUpdates()`. První parametr, jehož hodnotu nastavíme na `GPS_PROVIDER`, definuje využívání přesné polohy GPS. Druhý parametr udává periodu aktualizace v milisekundách. Jako kompromis jsme v aplikaci nastavili 3 sekundy, abychom byli trochu přátelští k baterii zařízení. Druhý parametr udává vzdálenost v metrech, která vyvolá aktualizaci. Pokud nepotřebujete změnu po metrech, která je náročná na výpočetní výkon, a tedy i spotřebu baterie, nastavte tento parametr na nulu.

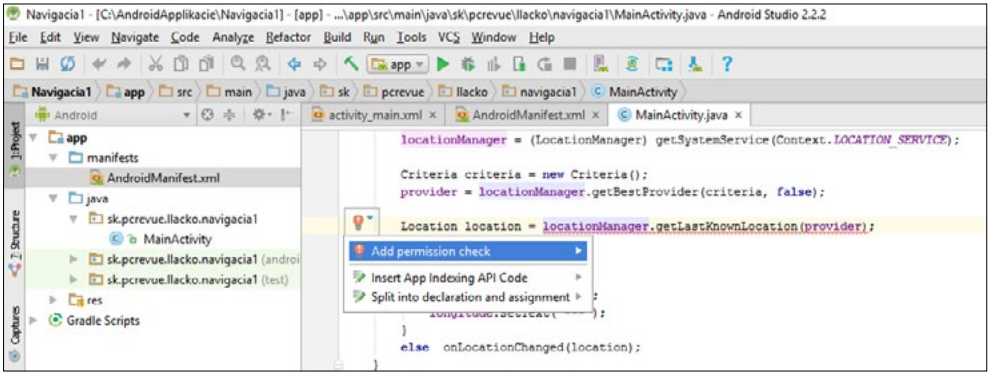
Pomocí metody `getBestProvider` vybíráte operátory podle definovaných kritérií. Kritériem bude zpravidla najít nejlepšího operátora, který nevyžaduje zpoplatnění.

Zjištění zeměpisných souřadnic je poměrně triviální:

```
locationManager = (LocationManager)
  getSystemService(Context.LOCATION_SERVICE);

Criteria criteria = new Criteria();
provider = locationManager.getBestProvider(criteria, false);
Location location = locationManager.getLastKnownLocation(provider);
```

Po napsání tohoto kódu však zjistíte, že Android Studio v metodě `locationManager.getLastKnown Location()` deklaruje chybu a nabídne její vyřešení tak, že před řádek s voláním procedury zařadí programovou konstrukci na zkontrolování oprávnění.



Obrázek 14.1: Přidání kontroly oprávnění

Některá oprávnění, včetně možnosti identifikovat aktuální polohu uživatele, jsou klasifikována jako nebezpečná a pro Android 6.0 nebo novější vyžadují speciální oprávnění. V manifestu už takovéto oprávnění máme.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

To v případě, že aplikace je nainstalovaná na starších verzích Androidu, zabezpečí vyžádání povolení od uživatele při instalaci aplikace. Ve verzi Android 6.0 nebo novější musí aplikace povolení žádat za běhu. Proto ta chybová hlášení a nabídka ošetření v Android Studiu.

Nabízená podmínka na ošetření kontroly oprávnění se automaticky vloží i s komentářem ve tvaru:

```
if (ActivityCompat.checkSelfPermission(this,
  Manifest.permission.ACCESS_FINE_LOCATION)
  != PackageManager.PERMISSION_GRANTED &&
  ActivityCompat.checkSelfPermission(this,
  Manifest.permission.ACCESS_COARSE_LOCATION)
  != PackageManager.PERMISSION_GRANTED) {
  // TODO: Consider calling
  // ActivityCompat#requestPermissions
  // here to request the missing permissions, and then overriding
  // public void onRequestPermissionsResult(int requestCode,
  // String[] permissions, int[] grantResults)
```

```
    // to handle the case where the user grants the permission.
    // See the documentation
    // for ActivityCompat#requestPermissions for more details.
    return;
  }
```

Pojďme na to pěkně postupně. V metodě `OnCreate()` vyžádáme povolení:

```
  protected void requestPermission(String permissionType, int requestCode) {
    int permission = ContextCompat.checkSelfPermission(this,
      permissionType);

    if (permission != PackageManager.PERMISSION_GRANTED) {
      ActivityCompat.requestPermissions(this,
        new String[]{permissionType}, requestCode
      );
    }
  }
```

V komentáři je návrh přetížit metodu `onRequestPermissionsResult()`.

```
  public void onRequestPermissionsResult(int requestCode,
    String permissions[], int[] grantResults) {
    switch (requestCode) {
      case LOCATION_REQUEST_CODE: {

        if (grantResults.length == 0
          || grantResults[0] != PackageManager.PERMISSION_GRANTED) {
          Toast.makeText(this,
            "Vyžaduje sa prístup k polohe", Toast.LENGTH_LONG).show();
        }
        return;
      }
    }
  }
```

Kód hlavní aktivity – blok na kontrolu oprávnění v manifestu, který vkládá Android Studio – je podtržen a uveden jen jednou. Před ostatními voláními metod třídy `locationManager` je jen komentář `//kontrola oprávnení`, taktéž podtržený:

```
  public class MainActivity extends AppCompatActivity
    implements LocationListener {

    private TextView latitute;
    private TextView longitude;
    private LocationManager locationManager;
    private String provider;


    private static final int LOCATION_REQUEST_CODE = 101;
    private String TAG = "Navigacia1";


    @Override
```

```
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  requestPermission(Manifest.permission.ACCESS_FINE_LOCATION,
    LOCATION_REQUEST_CODE);

  latitute = (TextView) findViewById(R.id.latitude);
  longitude = (TextView) findViewById(R.id.longitude);
  locationManager = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);

  Criteria criteria = new Criteria();
  provider = locationManager.getBestProvider(criteria, false);

  if (ActivityCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_FINE_LOCATION)
    != PackageManager.PERMISSION_GRANTED &&
    ActivityCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_COARSE_LOCATION) !=
    PackageManager.PERMISSION_GRANTED) {
    return;
  }
  Location location = locationManager.getLastKnownLocation(provider);

  if (location == null)
  {
    latitute.setText("---");
    longitude.setText("---");
  }
  else  onLocationChanged(location);

}

protected void requestPermission(String permissionType, int requestCode) {
  int permission = ContextCompat.checkSelfPermission(this,
    permissionType);

  if (permission != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this,
      new String[]{permissionType}, requestCode
    );
  }
}

@Override
public void onRequestPermissionsResult(int requestCode,
  String permissions[], int[] grantResults) {
  switch (requestCode) {
    case LOCATION_REQUEST_CODE: {
```

```
      if (grantResults.length == 0
        || grantResults[0] != PackageManager.PERMISSION_GRANTED) {
        Toast.makeText(this,
          "Vyžaduje sa prístup k polohe", Toast.LENGTH_LONG).show();
      }
      return;
    }
  }
}

@Override
protected void onResume()
{
  super.onResume();

  //kontrola oprávnení
  locationManager.requestLocationUpdates(provider, 3000, 1, this);
}

@Override
protected void onPause()
{
  super.onPause();
  //kontrola oprávnení
  locationManager.removeUpdates(this);
}

@Override
public void onLocationChanged(Location location)
{
  double lat = location.getLatitude();
  double lng = location.getLongitude();
  latitute.setText("Zemepisná šírka: " + String.valueOf(lat));
  longitude.setText("Zemepisná dĺžka" + String.valueOf(lng));
}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {}

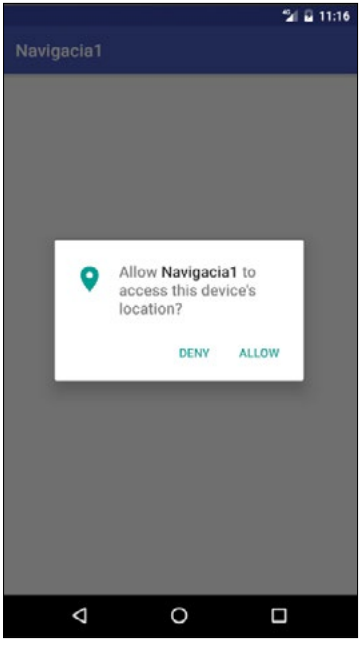
@Override
public void onProviderEnabled(String provider)
{
  Toast.makeText(this, "Provider Enabled" + provider,
    Toast.LENGTH_SHORT).show();

}

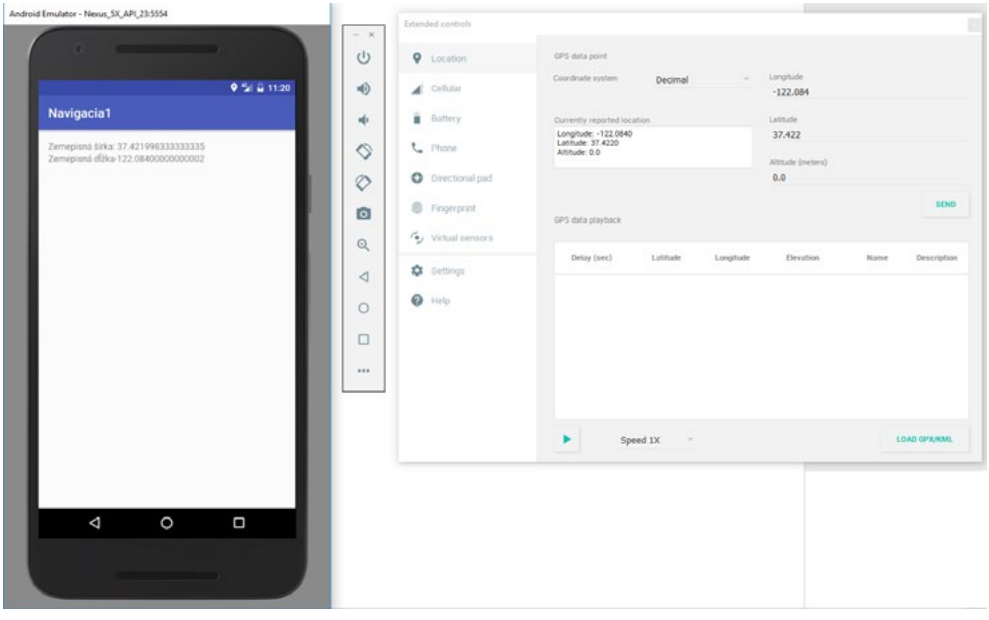
@Override
public void onProviderDisabled(String provider)
{
  Toast.makeText(this, "Provider Disabled  " + provider,
```

```
            Toast.LENGTH_SHORT).show();
    }
}
```

Po spuštění žádá aplikace uživatele o povolení přístupu k poloze.



Obrázek 14.2: Po spuštění žádá aplikace o povolení přístupu k poloze



Obrázek 14.3: Testování aplikace v emulátoru, kde můžete zeměpisnou polohu simulovat

Zjistíte, že aplikace v dosahu GPS funguje, a po čase zjistíte další věc: že se zařízení zahřívá více, než je obvyklé, a že rychle klesá kapacita baterie. Zjišťování souřadnic ze signálu GPS je totiž velmi výpočtově náročné, a proto je potřeba polohu zjišťovat jen tehdy, když je to nezbytné. Přesvědčíte se o tom i v nastavení operačního systému v položce **Poloha**. Vaše aplikace bude s největší pravděpodobností v sekci **Nedávné žádosti o určení polohy** charakterizovaná jako aplikace s vysokou spotřebou energie z baterie.

Příklad – Určení polohy zařízení

Proto v druhém příkladu ukážeme sofistikovanější aplikaci, která bude mnohem šetrnější k baterii vašeho zařízení. V příkladu zobrazíme nejprve poslední známou polohu zařízení, kterou, protože není aktuální, zobrazíme červeným písmem. Po načtení aktuálních hodnot je zobrazíme černým fontem. V aplikaci zobrazíme přesnost.

Abychom na to později nezapomněli, nejprve v *AndroidManifest.xml* deklarujte oprávnění aplikace přistupovat ke geografické poloze.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

Návrh layoutu hlavní aktivity obsahuje i výpis času a přesnosti či aktuálnosti údajů:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <TextView
    android:id="@+id/latitude"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FFFF0000">
  </TextView>
  <TextView
    android:id="@+id/longitude"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FFFF0000">
  </TextView>
  <TextView
    android:id="@+id/time"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FFFF0000">
  </TextView>
  <TextView
    android:id="@+id/accuracy"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FFFF0000">
  </TextView>
</LinearLayout>
```

Po spuštění aplikace nejprve zjišťuje, zda má systém k dispozici údaje o poloze, které nejsou starší než 5 minut (300 sekund = 300 000 milisekund). K tomuto účelu byla vytvořena metoda, která jako parametr dostane požadovanou přesnost a čas, jak dlouho akceptujeme neaktualizované údaje. Kontrola oprávnění v předchozím příkladu je nahrazena pro úsporu místa komentářem.

```
private Location posledniPozice(float minAccuracy, long cas)
{
  Location vysledek = null;
  float topPresnost = Float.MAX_VALUE;
  long topCas = Long.MIN_VALUE;

  List<String> matchingProviders = locManager.getAllProviders();
  for (String provider : matchingProviders)
  {
    //kontrola oprávnení
    Location location = locManager.getLastKnownLocation(provider);
    if (location != null)
    {
      float accuracy = location.getAccuracy();
      long time = location.getTime();

      if (accuracy < topPresnost)
      {
        vysledek = location;
        topPresnost = accuracy;
        topCas = time;
      }
    }
  }

  if (topPresnost > minAccuracy
    || (System.currentTimeMillis() - topCas) > cas)return null;
  else return vysledek;
}
```

Pokud jsou k dispozici údaje, které nejsou starší, než jsme ochotní akceptovat, vypíše aplikace tyto údaje okamžitě, ale červenou barvou. Zároveň začne zjišťovat aktuální polohu, a když se jí to podaří, změní se barva textu z červené na černou. Údaje o poloze se zjišťují z GPS, z GSM či Wi-Fi.

Kompletní kód hlavní aktivity:

```
public class MainActivity extends Activity
{
  private static final long VZORKOVANI = 1000 * 30;
  private static final long POLLING = 1000 * 10;
  private static final float MIN_PRESNOST = 25.0f;
  private static final float LAST_PRESNOST = 500.0f;
  private static final float MIN_VZDALENOST = 10.0f;

  private boolean bPrvni = true;
  private TextView lat, lng, time, accuracy;
```

```
private int barva = Color.BLACK;

private Location location;
private LocationManager locManager;
private LocationListener locListener;

@Override
public void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  lat = (TextView) findViewById(R.id.latitude);
  lng = (TextView) findViewById(R.id.longitude);
  time = (TextView) findViewById(R.id.time);
  accuracy = (TextView) findViewById(R.id.accuracy);

  if ((locManager = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE))==null)
    finish();

  location = posledniPozice(LAST_PRESNOST, 300000);
  if (null != location) Zobraz(location);
  else accuracy.setText("Poslední poloha starší než 5 minut");

  locListener = new LocationListener()
  {
    public void onLocationChanged(Location loc)
    {
      if (bPrvni)
      {
        barvyTextu(barva);
        bPrvni = false;
      }

      if (location == null || loc.getAccuracy() <
        location.getAccuracy())
      {
        location = loc;
        Zobraz(loc);
        //kontrola oprávnení
        if (location.getAccuracy() < MIN_PRESNOST)
        locManager.removeUpdates(locListener);
      }
    }

    public void onStatusChanged(String provider, int status,
      Bundle extras) {}
    public void onProviderEnabled(String provider) {}
    public void onProviderDisabled(String provider) {}
  };
```

```
}

@Override
protected void onResume()
{
  super.onResume();
  float presnost = location.getAccuracy();
  long cas = location.getTime();

  if (presnost > LAST_PRESNOST || cas < System.currentTimeMillis()-
    120000)
  {

    //kontrola oprávnení
    locManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
      POLLING, MIN_VZDALENOST, locListener);

    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
      POLLING, MIN_VZDALENOST, locListener);

    //naplanuj unregister listenerov po 30 sekundach
    Executors.newScheduledThreadPool(1).schedule(new Runnable()
    {
      @Override
      public void run()
      {
        locManager.removeUpdates(locListener);
      }
    }, VZORKOVANI, TimeUnit.MILLISECONDS);
  }
}

@Override
protected void onPause()
{
  super.onPause();
  locManager.removeUpdates(locListener);
}

private void Zobraz(Location location)
{
  lat.setText("Zem. šírka:" + location.getLongitude());
  lng.setText("Zem. délka:" + location.getLatitude());
  accuracy.setText("Přesnost:" + location.getAccuracy());
  time.setText("Čas:"  + new SimpleDateFormat("MM/dd/yyyy HH:mm:ss",
    Locale.getDefault()).format(new Date(location.getTime())));
}

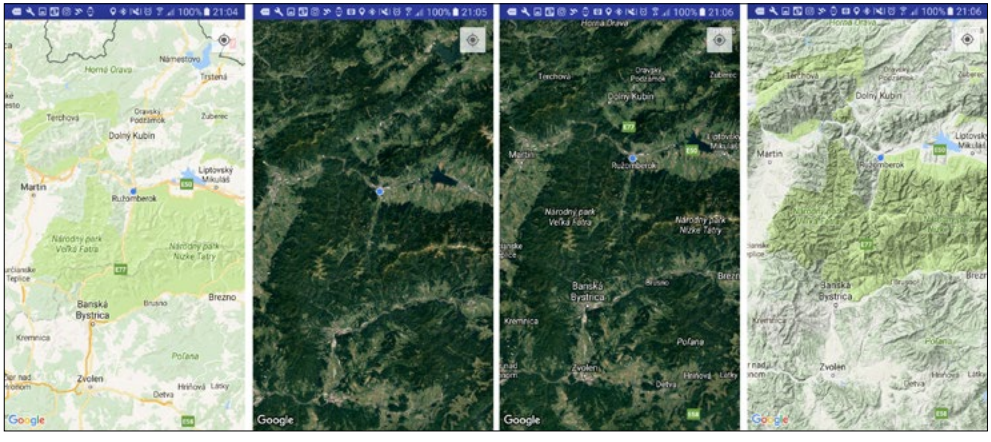
private void barvyTextu(int color)
{
  accuracy.setTextColor(color);
```

```
        time.setTextColor(color);
        lat.setTextColor(color);
        lng.setTextColor(color);
    }
}
```

Zobrazení polohy na mapě

Zobrazení polohy na mapě je vizuální reprezentací polohy. Google poskytuje mapovou službu Google Maps Android API. Služba poskytuje několik typů map:

- **Normal** – klasické cestovní mapy
- **Satellite** – snímky zemského povrchu
- **Hybrid** – kombinace snímků a klasické cestovní mapy
- **Terrain** – zobrazení topografických detailů, například výškových rozdílů a podobně



Obrázek 14.4: Příklad zobrazení Normal, Satellite Hybrid a Terain mapového podkladu

Aplikace pro Android dokáže přizpůsobit zobrazení mapy požadavkům uživatele. Umožňuje přidat ikony, tzv. markery, na konkrétní místa na mapě nebo kreslit na popředí, přičemž mapový podklad je na pozadí. Mapa může reagovat na gesta, nejčastěji změnou měřítka na gesto přiblížení nebo pootočením na gesto pootočení dvou prstů. API také umožňuje zobrazit aktuální polohu zařízení.

Google Maps Android API využívá základní třídy:

- `GoogleMap` – třída, která zastupuje a spravuje mapu. Řeší stahování a zobrazování mapových podkladů a reaguje na ovládací prvky na mapě. Objekt `GoogleMap` není vytvořen přímo aplikací, ale vytváří se samostatně při vytvoření instancí objektů typu `MapView` nebo `MapFragment`. Odkaz na objekt získáte pomocí metody `GetMap()` v instancích tříd `MapView`, `MapFragment` nebo `SupportMapFragment`.
- `MapFragment` – zobrazení mapy ve fragmentech.

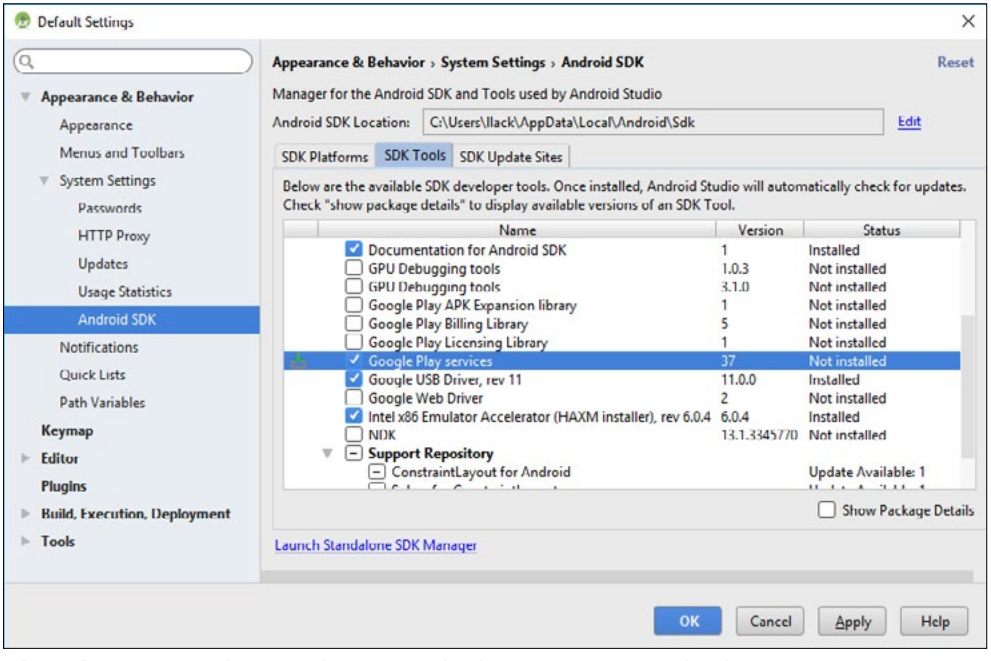
- `Camera` – definuje část mapy, která se zobrazí na obrazovce, a bod, z kterého by tento pohled snímala fiktivní kamera.
- `Marker` – třída reprezentuje ikony, někdy i vyskakovací okna, které ukazují na konkrétní lokality.

Přípravné kroky

Před začátkem vývoje aplikace využívající mapové služby je potřeba realizovat několik přípravných kroků. Nejprve je potřeba nakonfigurovat Google Play Services SDK. Následně je potřeba získat klíč API, který identifikuje vaši aplikaci. Parametry pro nastavení je potřeba vložit do manifestu aplikace.

Instalace Google Play Services

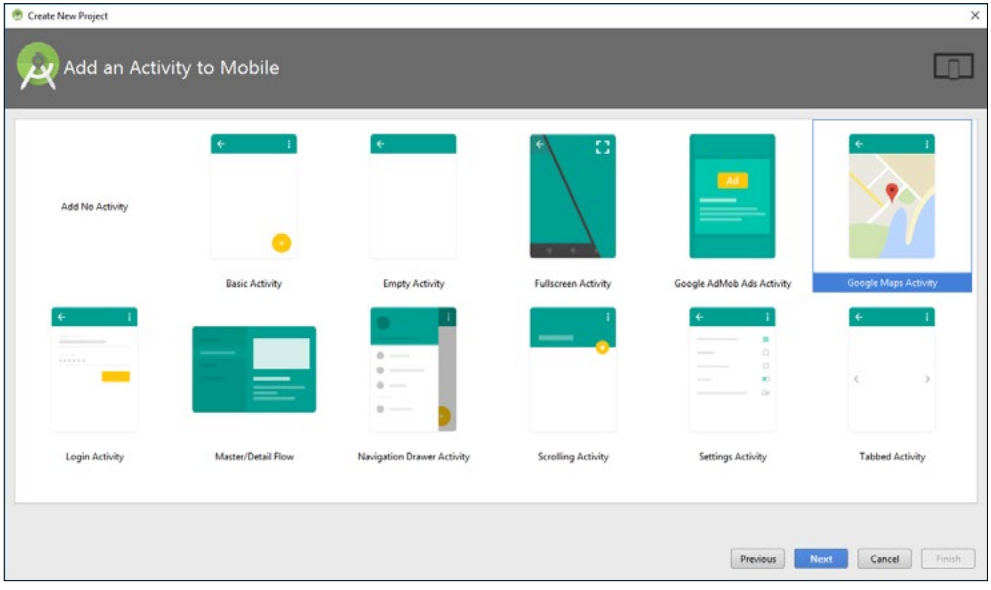
Pokud nemáte nainstalované Google Play Services, je potřeba doinstalovat toto SDK přes Android SDK Manager.



Obrázek 14.5: Kontrola nainstalování Google Play Services přes Android SDK Manager

Úvodní příklad

Vytvořte projekt aplikace a v dialogu na výběr hlavní aktivity vyberte typ **Google Maps Activity**. Minimální verze SDK musí být nastavena na API 19: Android 4.4 (KitKat) nebo novější verzi.

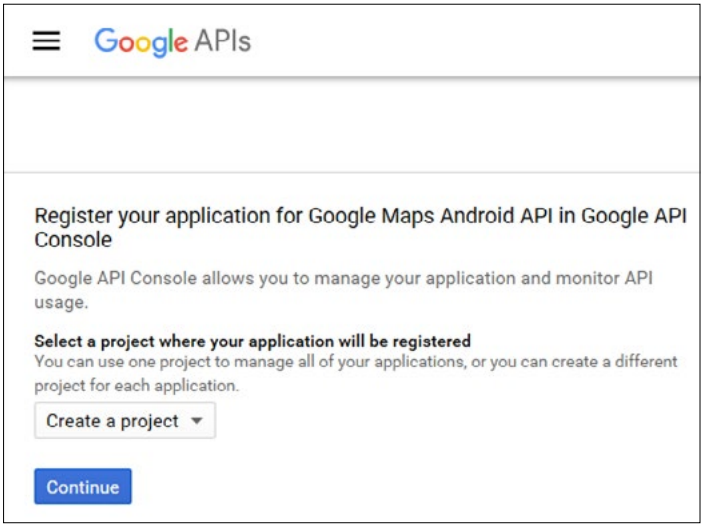


Obrázek 14.6: Výběr typu hlavní aktivity Google Maps Activity

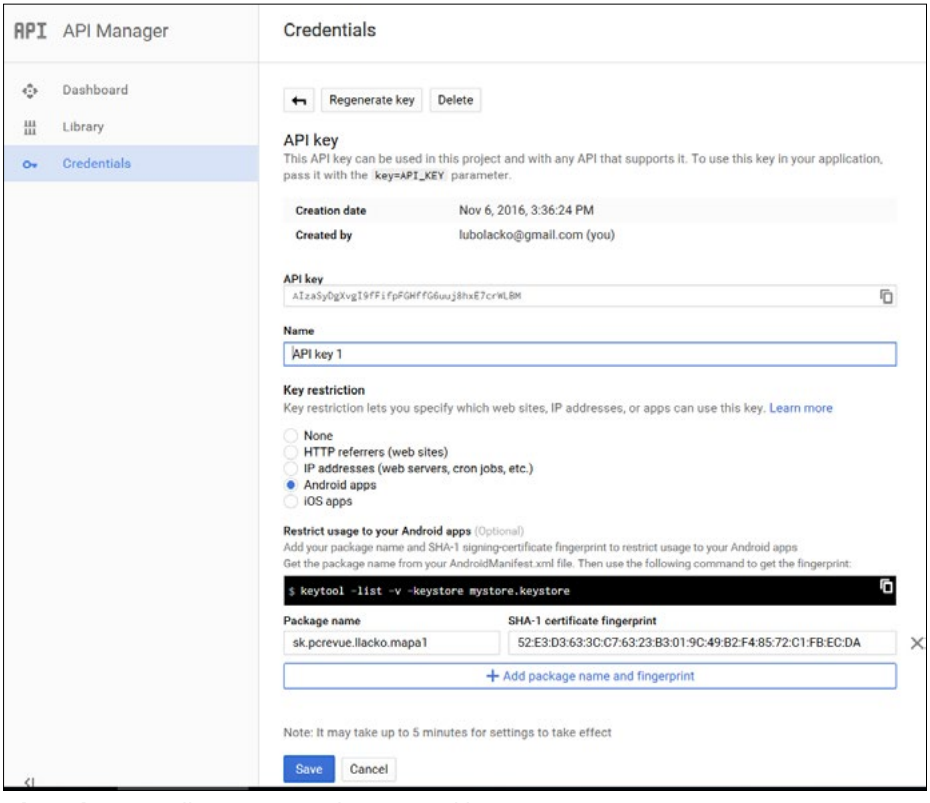
Vytvoření Google Maps API klíče

Předtím, než aplikace může využívat rozhraní API Google Maps pro Android, musí být nejprve zaregistrovaná v konzole Google API. K tomu je potřeba unikátní klíč označovaný i jako otisk SHA-1. Po vytvoření projektu se v souboru *google_maps_api.xml* zobrazí upozornění, že vaše aplikace potřebuje klíč Google Maps API. V upozornění je i odkaz na vývojářskou konzoli Google i s klíčem vloženým do URL. Zkopírujte a vložte tento odkaz do okna prohlížeče. Alternativně můžete spustit konzoli Google API přes adresu URL *https://code.google.com/apis/console*.

Pokračujte tlačítkem **Continue**. Zkontrolujte, zda je vybraná volba **Create a project**. Na další stránce pokračujte tlačítkem **Create API key**. Pro jistotu můžete zkontrolovat, zda se otisk SHA-1 shoduje s tím, co je uvedeno v souboru *google_maps_api.xml*. Pokračujte tlačítkem **Create**.



Obrázek 14.7: Konzole Google API spuštěná přes adresu URL připravenou v souboru google_maps_api.xml



Obrázek 14.8: Volby a nastavení k vytvoření klíče

Vraťte se do projektu v Android Studiu a v souboru *google_maps_api.xml* zkopírujte klíč namísto žolíka, čili zástupného textu `YOUR_KEY_HERE`.

```
<string name="google_maps_key" templateMergeStrategy="preserve"
translatable="false">AIzaSyDgXvgI9fFifpFGHffG6uuj8hxE7crWLBM</string>
```

Google Maps API klíč je zadarmo, můžete ho použít ve vašich aplikacích, které vyžadují rozhraní služby Maps API. Klíč podporuje neomezený počet uživatelů. Klíč potřebujete na publikování aplikace. Během vývoje vám automaticky vytvoří ladicí klíč.

Všimněte si, že klíč, který jste vložili do souboru *google_maps_api.xml*, se automaticky uložil i do aplikačního manifestu v souboru *AndroidManifest.xml*. Všimněte si poznámky v komentářích:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.mapa1">

  <!--
    The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
    Google Maps Android API v2, but you must specify either coarse or fine
    location permissions for the 'MyLocation' functionality.
  -->
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

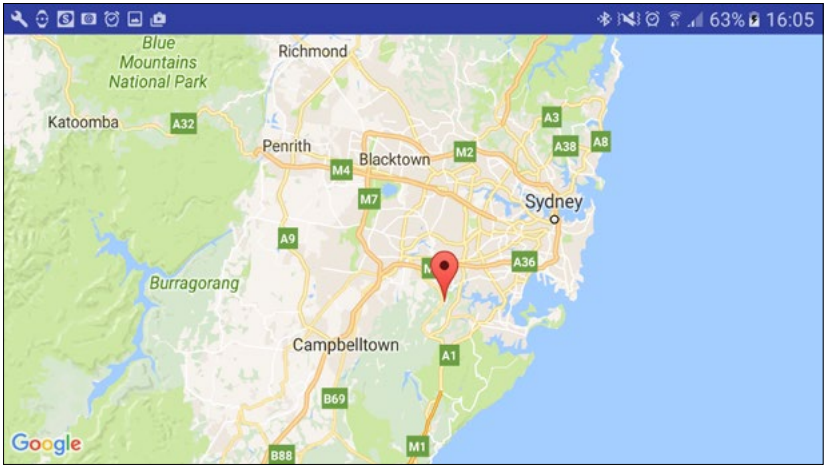
    <!--
      The API key for Google Maps-based APIs is defined as a string resource.
      (See the file "res/values/google_maps_api.xml").
      Note that the API key is linked to the encryption key used
      to sign the APK.
      You need a different API key for each encryption key,
      including the release
      key that is used to sign the APK for publishing.
      You can define the keys for the debug and release targets
      in src/debug/ and src/release/.
    -->
    <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="@string/google_maps_key" />

    <activity
      android:name=".MapsActivity"
      android:label="@string/title_activity_maps">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
```

```
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Nyní můžete aplikaci spustit. Na emulátoru nebo i reálném zařízení se vám může zobrazit oznámení, že aplikace se bude dát spustit až po upgradu Google Play Services. V takovém případě potvrďte oznámení tlačítkem **Update**.



Obrázek 14.9: Test aplikace projektu s hlavní aktivitou typu Google Maps Activity

Zobrazí se mapa s vlaječkou v australském Sydney. Je to v pořádku, protože přesně tato poloha je po vytvoření projektu s hlavní aktivitou typu Google Maps Activity v kódu hlavní aktivity definovaná.

```
public class MapsActivity extends FragmentActivity implements
  OnMapReadyCallback {

  private GoogleMap mMap;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);
    // Obtain the SupportMapFragment and get notified
    // when the map is ready to be used.
    SupportMapFragment mapFragment = (SupportMapFragment)
      getSupportFragmentManager()
      .findFragmentById(R.id.map);
      mapFragment.getMapAsync(this);
  }

  @Override
  public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
```

```
    // Add a marker in Sydney and move the camera
    LatLng sydney = new LatLng(-34, 151);
    mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in
      Sydney"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
  }
}
```

Pro úplnost ještě návrh layoutu hlavní aktivity v souboru *activity_maps.xml*:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:map="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.SupportMapFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.mapa1.MapsActivity" />
```

Příklad zobrazení polohy na mapě

Po seznámení se s aplikací s hlavní aktivitou typu **Google Maps Activity** ji můžete vylepšit. Připomínáme, že aplikace zatím jen zobrazila místo se zadanými souřadnicemi, takže nezjišťovala aktuální polohu zařízení.

Aplikace využívající mapové služby vyžaduje v manifestu několik typů oprávnění. Google Maps Android API v2 oprávnění `ACCESS_FINE_LOCATION` nevyžaduje, avšak určení polohy vašeho zařízení ano. Proto do aplikačního manifestu nezapomeňte přidat oprávnění:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Nejprve změňte implicitně přidaný marker označující australské město Sydney a přidejte další marker i vyskakovací okno s textem a obrázkem, v tomto případě městského erbu měst Ružomberok a Liptovský Mikuláš. Text se zobrazí po klepnutí na ikonu.

```
public class MapsActivity extends FragmentActivity implements
  OnMapReadyCallback {

  private GoogleMap mMap;
  static final LatLng LIPTOVSKY_MIKULAS = new LatLng(49.084167, 19.602222);
  static final LatLng RUZOMBEROK = new LatLng(49.078611, 19.308333);

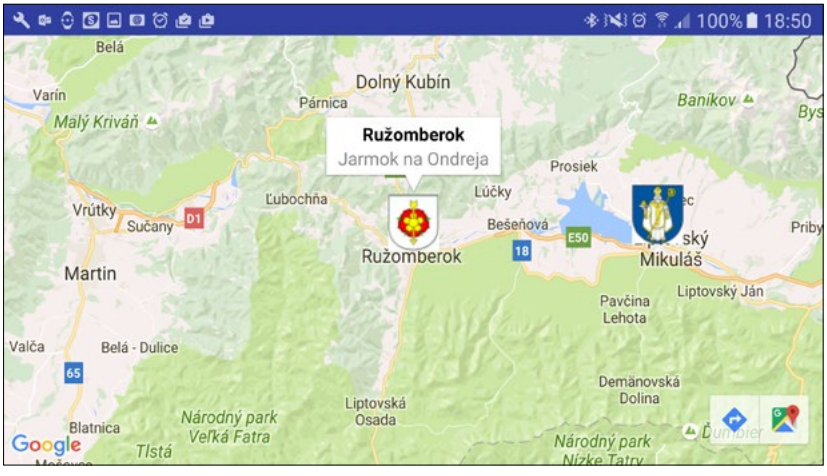
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);
    // Obtain the SupportMapFragment and get notified
    // when the map is ready to be used.
    SupportMapFragment mapFragment = (SupportMapFragment)
      getSupportFragmentManager()
      .findFragmentById(map);
      mapFragment.getMapAsync(this);
  }
```

```
    @Override
    public void onMapReady(GoogleMap googleMap) {
      mMap = googleMap;

      Marker rbk = mMap.addMarker(new MarkerOptions()
        .position(RUZOMBEROK)
        .title("Ružomberok")
        .snippet("Jarmok na Ondreja")
        .icon(BitmapDescriptorFactory
        .fromResource(R.drawable.ruzomberok)));

      Marker lm = mMap.addMarker(new MarkerOptions()
        .position(LIPTOVSKY_MIKULAS)
        .title("Lipt. Mikuláš")
        .snippet("Jarmok na Mikuláša")
        .icon(BitmapDescriptorFactory
        .fromResource(R.drawable.mikulas)));

      mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(LIPTOVSKY_MIKULAS,
        15));
      mMap.animateCamera(CameraUpdateFactory.zoomTo(10), 2000, null);
    }
}
```



Obrázek 14.10: Test aplikace zobrazující mapu a markery

Příklad – Zobrazení polohy v jiné aplikaci schopné zobrazit mapy

V předchozí aplikaci jsme zobrazovali mapu v samotné aplikaci. Často je v souladu s koncepcí Androidu účelné zobrazit polohu na mapě v jiné aplikaci, která je k tomu určena. Pokud je takových aplikací více, vybere si nejvhodnější uživatel. Proto v naší aplikaci zadáme jen název, případně poštovní adresu lokality, kterou chceme zobrazit. A tuto odešleme jako záměr. V layoutu

hlavní aktivity bude jen prvek TextView a tlačítko. V prvku TextView si všimněte definice typu textu android:inputType="textPostalAddress".

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${relativePackage}.${activityClass}" >

  <EditText
    android:id="@+id/lokalita"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:hint="@string/lokalita_txt"
    android:inputType="textPostalAddress" />

  <Button
    android:id="@+id/mapaButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/lokalita"
    android:layout_centerHorizontal="true"
    android:text="@string/mapa_tlacidlo" />
</RelativeLayout>
```

Pro úplnost uvádíme i definice textových řetězců.

```
<resources>
  <string name="app_name">Najdi lokalitu</string>
  <string name="lokalita_txt">Zadej název nebo adresu lokality</string>
  <string name="mapa_tlacidlo">Zobraz mapu</string>
</resources>
```

V kódu hlavní aktivity se na základě zadané adresy připraví řetězec s parametry, který se přes záměr odevzdá aktivitě aplikace schopné zobrazit mapu. Jelikož je to vhodný příklad na demonstraci životního cyklu aktivity, v metodách souvisejících se životním cyklem aktivity jsme přidali výpis aktuálního stavu do protokolu.

```
public class MainActivity extends Activity
{
  private final String TAG = "NajdiLokalitu";
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    final EditText addrText = (EditText) findViewById(R.id.lokalita);
    final Button button = (Button) findViewById(R.id.mapaButton);

    button.setOnClickListener(new Button.OnClickListener()
    {
```

```
    @Override
    public void onClick(View v)
    {
      try {
        String address = addrText.getText().toString();
        address = address.replace(' ', '+');
        Intent geoIntent = new Intent(
          android.content.Intent.ACTION_VIEW,
          Uri.parse("geo:0,0?q=" + address));
        startActivity(geoIntent);
      } catch (Exception e) {
        Log.e(TAG, e.toString());
      }
    }
  });
}

@Override
protected void onStart()
{
  super.onStart();
  Log.i(TAG, "Aktivita spustená.");
}

@Override
protected void onRestart()
{
  super.onRestart();
  Log.i(TAG, "Aktivita restartovaná.");
}

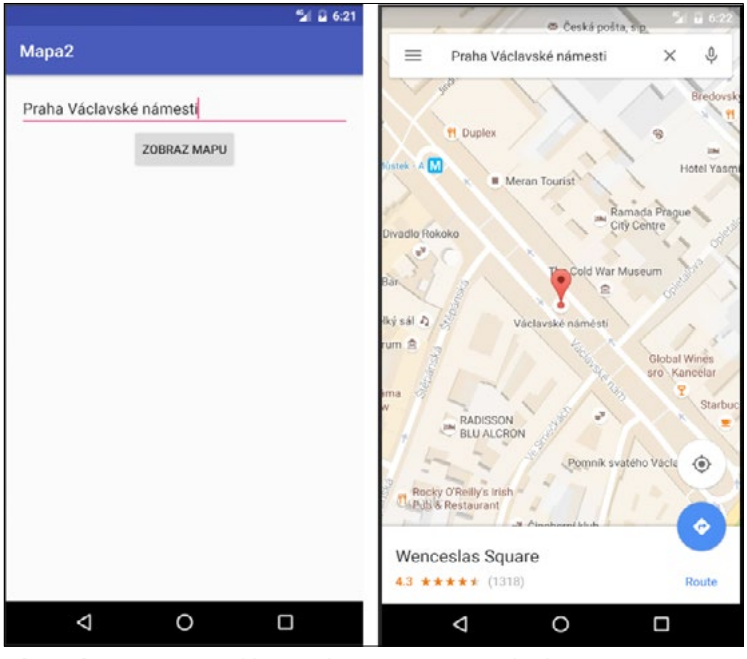
@Override
protected void onResume()
{
  super.onResume();
  Log.i(TAG, "Aktivita má fokus a je v stave Resume");
}

@Override
protected void onPause()
{
  super.onPause();
  Log.i(TAG, "Iná aktivita získala fokus");
}

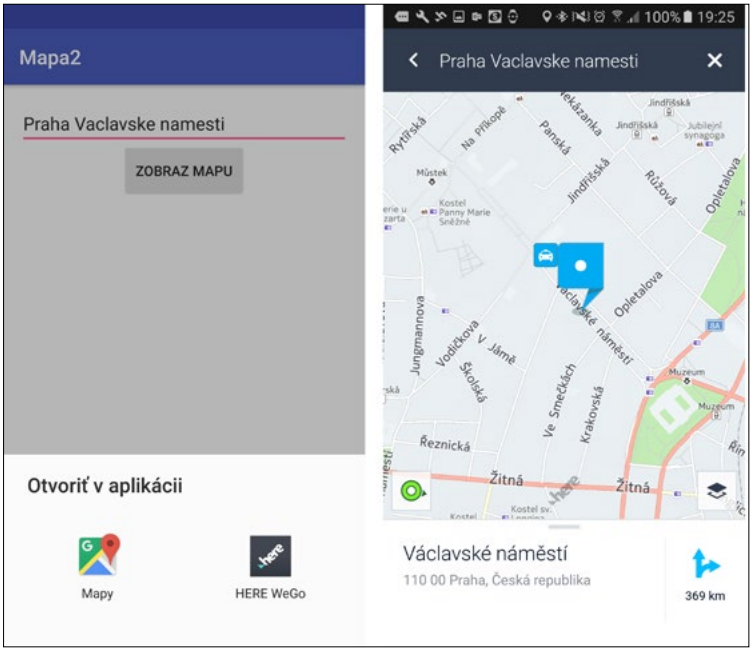
@Override
protected void onStop()
{
  super.onStop();
  Log.i(TAG, "Aktivita zastavená");
}
```

```
    @Override
    protected void onDestroy()
    {
      super.onDestroy();
      Log.i(TAG, "Aktivita odstránená.");
    }
}
```

Po zadání názvu nebo poštovní adresy lokality mohou nastat dvě situace. Pokud je na zařízení jen jediná aplikace schopná zobrazit mapu, zobrazí se rovnou poloha zadaného místa. Toto je typický případ emulátoru, kde je jen aplikace Google Maps. Na reálném zařízení je zpravidla více aplikací schopných zobrazit mapové informace. Proto se v takovém případě zobrazí nabídka aplikací, které jsou schopné zobrazit polohu lokality na mapě.



Obrázek 14.11: Test aplikace zobrazující mapu, pokud je na zařízení dostupná pouze jedna taková aplikace



Obrázek 14.12: Test aplikace zobrazující mapu, pokud je na zařízení více aplikací schopných zobrazit mapu. Uživatel si vybral aplikaci HERE WeGo

Příklad – Zobrazení aktuální polohy na mapě

V posledním příkladu této kapitoly spojíme poznatky s určováním polohy a Google Maps API s cílem zobrazit na mapě aktuální polohu. I v tomto příkladu vytvořte projekt aplikace a v dialogu na výběr hlavní aktivity vyberte typ **Google Maps Activity**. Minimální verze SDK musí být nastavena na API 19: Android 4.4 (KitKat) nebo novější verzi.

I v tomto příkladu se nejprve otevře soubor *google_maps_api.xml*, kde je odkaz k získání klíče Google Maps API. V upozornění je i odkaz na vývojářskou konzoli Google s klíčem vloženým do URL. Zkopírujte a vložte tento odkaz do okna prohlížeče. Postup získání klíče je popsán v prvním příkladu využívajícím mapy. Po získání zkopírujte klíč na místo žolíka, tj. zástupného textu `YOUR_KEY_HERE`.

```
<string name="google_maps_key" templateMergeStrategy="preserve"
translatable="false">AIzaSyDgXvgI9fFifpFGHffG6uuj8hxE7crWLBM</string>
```

Do aplikačného manifestu se automaticky přidá oprávnění na určení přesné polohy. Doporučujeme i přidání oprávnění na určení přibližné polohy podle GSM nebo Wi-Fi, pokud signál GPS není k dispozici.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Layout hlavní aktivity necháme bez úprav:

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:map="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.SupportMapFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.mapa3.MapsActivity" />
```

Všechno podstatné se bude odehrávat v kódu hlavní aktivity:

```
public class MapsActivity extends FragmentActivity implements
  OnMapReadyCallback {

  private static final int LOCATION_REQUEST_CODE = 101;

  private GoogleMap mMap;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);

    requestPermission(Manifest.permission.ACCESS_FINE_LOCATION,
      LOCATION_REQUEST_CODE);

    // Indikuje že mapa je pripravená
    SupportMapFragment mapFragment = (SupportMapFragment)
      getSupportFragmentManager()
      .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
  }

  protected void requestPermission(String permissionType, int requestCode) {
    int perm = ContextCompat.checkSelfPermission(this,
      permissionType);

    if (perm != PackageManager.PERMISSION_GRANTED) {
      ActivityCompat.requestPermissions(this,
        new String[]{permissionType}, requestCode
      );
    }
  }

  @Override
  public void onRequestPermissionsResult(int requestCode,
    String perm[], int[] grantResults) {
      switch (requestCode) {
        case LOCATION_REQUEST_CODE: {

        if (grantResults.length == 0
```

```
            || grantResults[0] != PackageManager.PERMISSION_GRANTED) {
            Toast.makeText(this, "Potrebujete oprávnenie na určenie polohy",
              Toast.LENGTH_LONG).show();
          }
          return;
        }
      }
    }

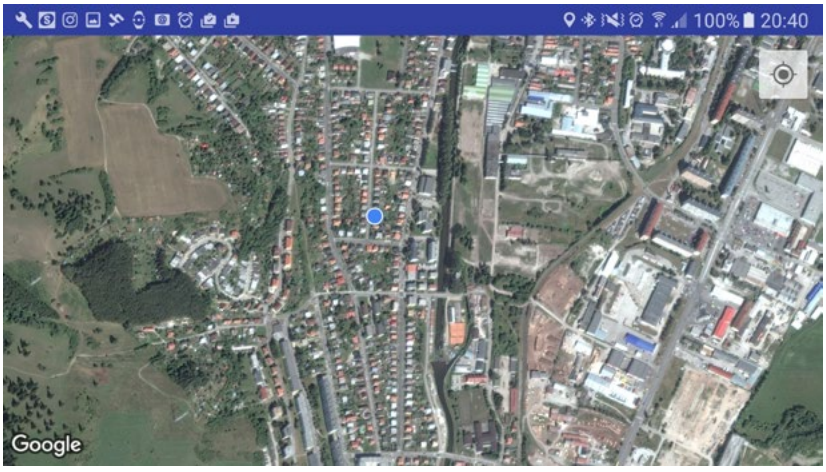
    @Override
    public void onMapReady(GoogleMap googleMap) {
      mMap = googleMap;

      if (mMap != null) {
        if (ActivityCompat.checkSelfPermission(this,
          Manifest.permission.ACCESS_FINE_LOCATION) !=
          PackageManager.PERMISSION_GRANTED &&
          ActivityCompat.checkSelfPermission(this,
          Manifest.permission.ACCESS_COARSE_LOCATION) !=
          PackageManager.PERMISSION_GRANTED) {
            return;
        }
        mMap.setMyLocationEnabled(true);
        mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
      }

    }
}
```

V tomto případě jsme použili typ zobrazení mapy `MAP_TYPE_SATELLITE`. Jak už bylo v úvodu této části zmíněno, k dispozici jsou tyto typy zobrazení:

- `GoogleMap.MAP_TYPE_NONE` – mapový podklad se nezobrazí
- `GoogleMap.MAP_TYPE_NORMAL` – zobrazí se mapa
- `GoogleMap.MAP_TYPE_SATELLITE` – zobrazí se satelitní záběr vybrané oblasti
- `GoogleMap.MAP_TYPE_HYBRID` – kombinace satelitního záběru s některými atributy mapy jako jsou cesty, popisy měst, ulic a podobně
- `GoogleMap.MAP_TYPE_TERRAIN` – topografické informace



Obrázek 14.13: Zobrazení aktuální polohy na mapě

KAPITOLA

15

Sociální sítě

Témata kapitoly:

- Vytvoření aplikace využívající Facebook
- Příklad aplikace na posílání statusů a obrázků

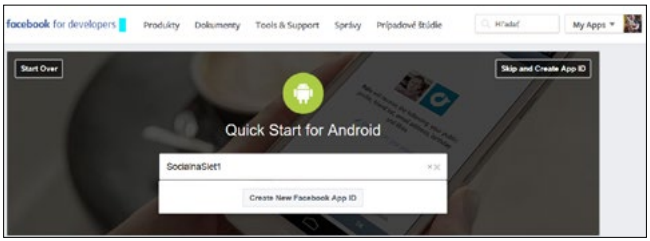
Vytvoření aplikace využívající Facebook

Mobilní aplikace stále častěji komunikují se sociálními sítěmi, v našich končinách především s Facebookem. Umožňují sdílet polohu, různé události, skóre ve hrách a podobně. Námětem kapitoly je umožnit aplikaci posílat statusy na Facebook – jinak řečeno: jak v aplikaci využít Facebook SDK. Tento framework je podporovaný od verze Android 4.0.3 (API 15:). Uživatelé musí mít na svém zařízení s Androidem nainstalovanou nativní verzi aplikace Facebook ve verzi 7.0 a novější.

Facebook for developers

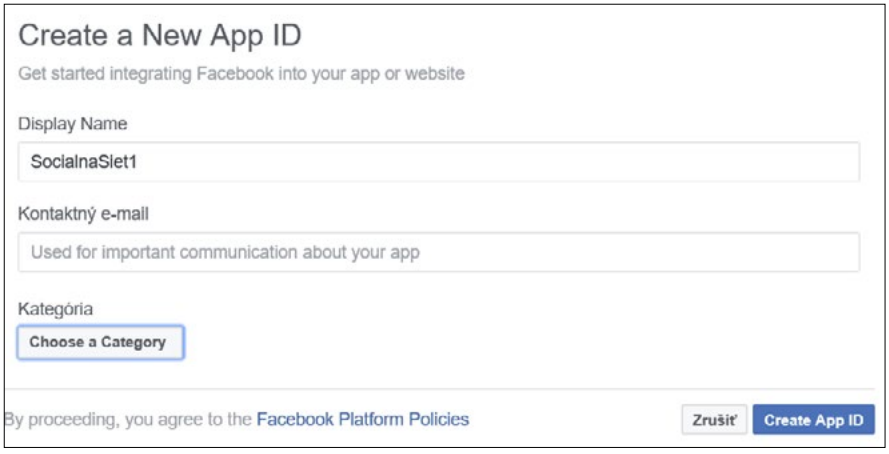
Začněte na stránce Facebook pro vývojáře – *https://developers.facebook.com*. Nejste-li registrovaní, musíte se nejprve zaregistrovat jako vývojář. Následně proběhne zpřísněná kontrola vaší identity přes SMS na vaše telefonní číslo. Po přihlášení najděte na stránce odkaz **Getting Started Android** a následně klepněte na odkaz **Quick Start**. Můžete použít i přímý odkaz: *https://developers.facebook.com/quickstarts/?platform=android*.

Na vývojářském portálu zadejte název nové aplikace.



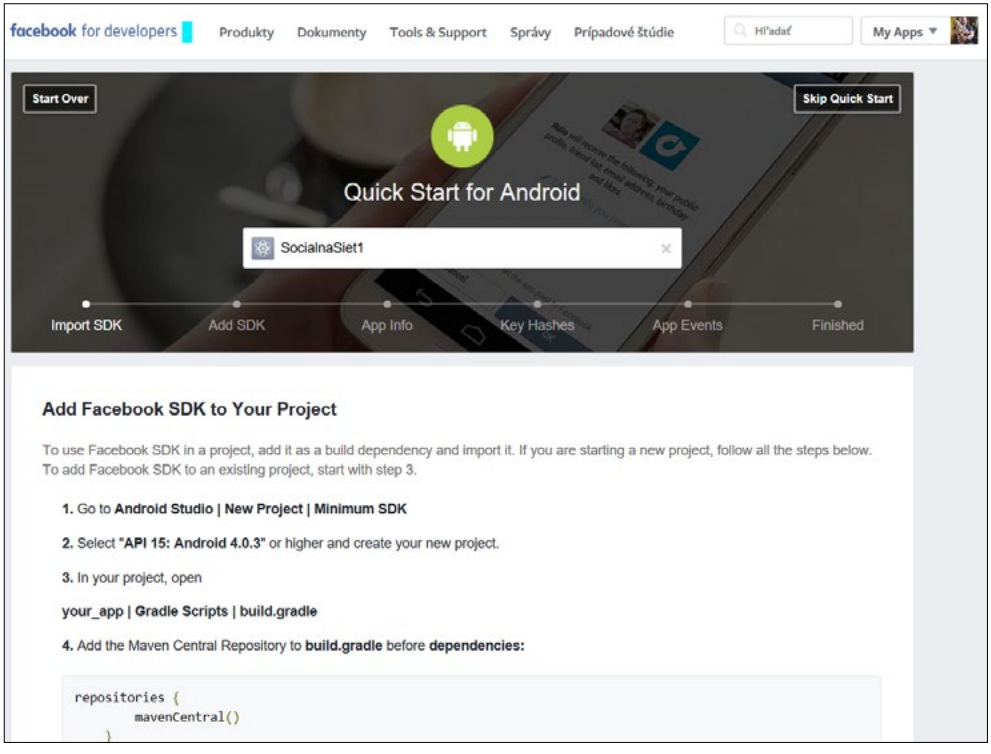
Obrázek 15.1: Vytvoření aplikace na vývojářském portálu Facebooku

Upozorňujeme, že na základě pravidel pro vývoj aplikací nesmí název aplikace obsahovat textové řetězce „face“, „FB“, „Book“, „Insta“, „Gram“ a „Rift“ (kvůli ochraně značek, které patří Facebooku). Budete vyzváni, abyste identifikovali textový řetězec, aby se potvrdilo, že nejste robot.



Obrázek 15.2: Upřesnění údajů pro aplikaci

Následně se zobrazí stránka s rámcovým postupem, který má 7 kroků.



Obrázek 15.3: Quick Start for Android

Můžete postupovat podle nich nebo podle našeho návodu.

Sdílení obsahu z aplikace na Facebooku

Z aplikace pro Android můžete na Facebook sdílet různé zajímavosti, například polohu, dosažené skóre ve hře, různé zajímavosti a podobně.

Do aplikačního manifestu je potřeba přidat nastavení `ContentProvider`. Parametr `{APP_ID}` je ID vaší aplikace.

```
<provider
  android:authorities="com.facebook.app.FacebookContentProvider{APP_ID}"
  android:name="com.facebook.FacebookContentProvider"
  android:exported="true"/>
```

V našem případě je to:

```
<provider
  android:authorities="com.facebook.app.
FacebookContentProvider1334950249883434"
  android:name="com.facebook.FacebookContentProvider"
  android:exported="true"/>
```

Post může sestávat z těchto atributů:

- `contentURL` – adresa URL sdíleného obsahu na webu
- `contentTitle` – název příspěvku
- `imageURL` – URL obrázku, který může být součástí příspěvku
- `contentDescription` – text příspěvku, zpravidla 2–4 věty

Pro každý atribut využijete model `ShareLinkContent`. Například:

```
ShareLinkContent content = new ShareLinkContent.Builder()
  .setContentUrl(Uri.parse("https://developers.facebook.com"))
  .build();
```

Při sdílení obrázků je potřeba mít na paměti maximální velikost souboru s obrázkem 12 MB a uživatelé musí mít na svém zařízení s Androidem nainstalovanou nativní verzi aplikace Facebook ve verzi 7.0 a novější.

```
Bitmap image = ...
SharePhoto photo = new SharePhoto.Builder()
  .setBitmap(image)
  .build();
SharePhotoContent content = new SharePhotoContent.Builder()
  .addPhoto(photo)
  .build();
```

Podobně je možné sdílet i video, i zde je omezení maximální velikosti souboru s videem 12 MB.

```
Uri videoFileUri = ...
ShareVideo = new ShareVideo.Builder()
  .setLocalUrl(videoUrl)
  .build();
ShareVideoContent content = new ShareVideoContent.Builder()
  .setVideo(video)
  .build();
```

Můžete sdílet i kombinovaný multimediální obsah, například fotografie a videa, maximálně 6 fotek nebo videí v jednom postu.

```
SharePhoto sharePhoto1 = new SharePhoto.Builder()
    .setBitmap(...)
    .build();
SharePhoto sharePhoto2 = new SharePhoto.Builder()
  .setBitmap(...)
  .build();
ShareVideo shareVideo1 = new ShareVideo.Builder()
  .setLocalUrl(...)
  .build();
ShareVideo shareVideo2 = new ShareVideo.Builder()
  .setLocalUrl(...)
  .build();

ShareContent shareContent = new ShareMediaContent.Builder()
  .addMedium(sharePhoto1)
  .addMedium(sharePhoto2)
  .addMedium(shareVideo1)
  .addMedium(shareVideo2)
  .build();

ShareDialog shareDialog = new ShareDialog(...);
shareDialog.show(shareContent, Mode.AUTOMATIC);
```

Uživatelské rozhraní na sdílení obsahu na Facebook

Facebook SDK nabízí i grafické prvky uživatelského rozhraní. Začneme tlačítkem **Like**.

```
LikeView likeView = (LikeView) findViewById(R.id.like_view);
likeView.setObjectIdAndType(
  "https://www.facebook.com/FacebookDevelopers",
  LikeView.ObjectType.PAGE);
```

Tlačítko **Share** otevře dialog ke sdílení obsahu:

```
ShareButton shareButton = (ShareButton)findViewById(R.id.fb_share_button);
shareButton.setShareContent(content);
```

Dialog ke sdílení přepne uživatele do nativní aplikace Facebook a po odeslání sdíleného obsahu sa fokus vrátí do vaší aplikace.

```
ShareDialog.show(activityOrFragment, content);
```

Dialog můžete vytvořit v metodě `OnCreate`:

```
public class MainActivity extends FragmentActivity {
  CallbackManager callbackManager;
  ShareDialog shareDialog;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    FacebookSdk.sdkInitialize(getApplicationContext());
    callbackManager = CallbackManager.Factory.create();
```

```
    shareDialog = new ShareDialog(this);
    // voliteľná registrácia callbacku
    shareDialog.registerCallback(callbackManager, new
      FacebookCallback<Sharer.Result>() { ... });
  }
}
```

Následně dialog zobrazíte:

```
if (ShareDialog.canShow(ShareLinkContent.class)) {
  ShareLinkContent linkContent = new ShareLinkContent.Builder()
    .setContentTitle("Hello Facebook")
    .setContentDescription(
    "Aplikácia 'SocialnaSieť1' zdieľa")
    .setContentUrl(Uri.parse("http://developers.facebook.com/android"))
    .build();
  shareDialog.show(linkContent);
}
```

Navíc volání callback manageru z SDK:

```
@Override
protected void onActivityResult(final int requestCode, final int resultCode,
  final Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
  callbackManager.onActivityResult(requestCode, resultCode, data);
}
```

Tlačítko **Send** aktivuje **Message Dialog**, který odešle post z aplikace na sociální síť:

```
SendButton sendButton = (SendButton)findViewById(R.id.fb_send_button);
sendButton.setShareContent(shareContent);
sendButton.registerCallback(callbackManager, new
  FacebookCallback<Sharer.Result
```

Vytvoření aplikace pro Android přihlašující se k Facebooku

Vytvořte projekt aplikace pro Android. Nemusíte používat stejný název aplikace a jmenného prostoru, jako jste zadali, když jste registrovali aplikaci na vývojářském portálu Facebooku, je to však vhodné kvůli konzistenci.

Vytvořte tedy projekt se stejným názvem, jaký jste si zaregistrovali na vývojářském portálu Facebooku. Nejprve upravte soubor *build.gradle* (Module:app). Najdete ho ve složce *Gradle Scripts*. Do souboru přidejte před blok `dependencies` `Maven Central Repository`.

```
repositories {
  mavenCentral()
}
```

Do sekce `dependencies` je potřeba přidat následující řádek:

```
compile 'com.facebook.android:facebook-android-sdk:[4,5)'
```

Soubor *build.gradle* (Module:app):

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 24
  buildToolsVersion "24.0.0"
  defaultConfig {
    applicationId "sk.pcrevue.llacko.socialnasiet1"
    minSdkVersion 22
    targetSdkVersion 24
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner
      "android.support.test.runner.AndroidJUnitRunner"
  }
  buildTypes {
    release {
      minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'),
        'proguard-rules.pro'
    }
  }
}

repositories {
  mavenCentral()
}

dependencies {
  compile fileTree(dir: 'libs', include: ['*.jar'])
  androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2',{
    exclude group: 'com.android.support', module: 'support-annotations'
  })
  compile 'com.android.support:appcompat-v7:24.2.1'
  compile 'com.facebook.android:facebook-android-sdk:[4,5)'
  testCompile 'junit:junit:4.12'
}
```

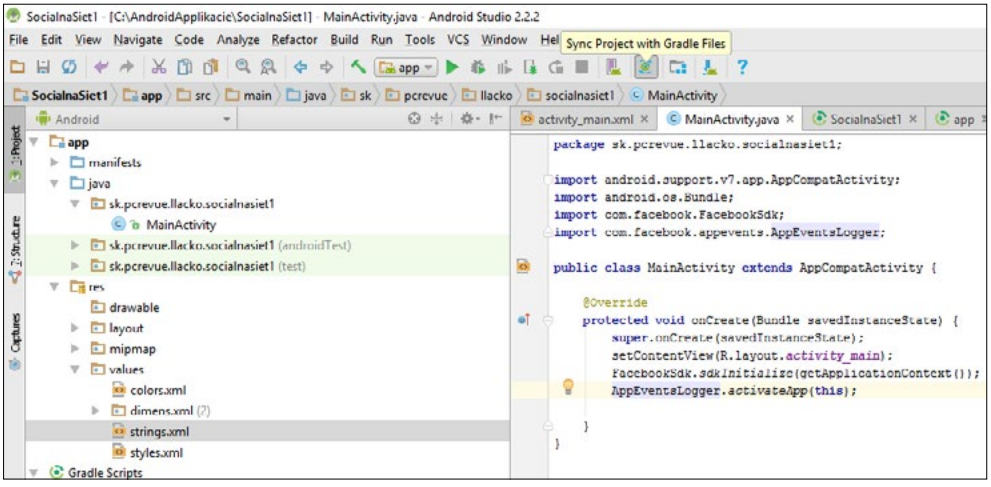
V této fázi je potřeba projekt sestavit, aby se změny v souboru *build.gradle* synchronizovaly s projektem. Na liště použijte tlačítko **Sync Project with Gradle Files** (u implicitní konfigurace čtvrté zleva).

Do kódu aplikace přidejte import:

```
import com.facebook.FacebookSdk;
```

Parametr App ID budete potřebovat vložit do projektu vaší aplikace pro Android. Do souboru *strings.xml* přidejte řetězec Facebook ID vaší aplikace:

```
<string name="facebook_app_id">1334950249883434</string>
```



Obrázek 15.4: Quick Start for Android

V aplikačním manifestu je potřeba přidat povolení přístupu k internetu:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

A také metadata element:

```
<application android:label="@string/app_name" ...>
  ...
  <meta-data android:name="com.facebook.sdk.ApplicationId"
    android:value="@string/facebook_app_id"/>
  ...
</application>
```

Na stránce přibude sekce pro aplikaci, do které je potřeba přidat do pole KEY HASHES takzvaný podpisový klíč aplikace. Tento klíč získáte pomocí nástroje keytool z balíčku Java.

Ladicí klíč vygenerovaný vývojovým prostředím Android Studio na podepisování vašich aplikací najdete ve složce *user/.android/debug.keystore*; v tomto případě konkrétně ve složce *C:\Users\Luboslav\.android*. Jedná se o binární soubor. Z něj se generuje takzvaný digitální otisk pomocí aplikace *Keytool.exe*.

```
keytool -exportcert -alias androiddebugkey -keystore
  %HOMEPATH%\.android\debug.keystore | openssl sha1 -binary
```

Aplikaci *keytool.exe* najdete v adresáři, do kterého se nainstalovala Java. V našem případě to byl adresář *C:\Program Files\Java\jdk1.8.0_91\bin*.

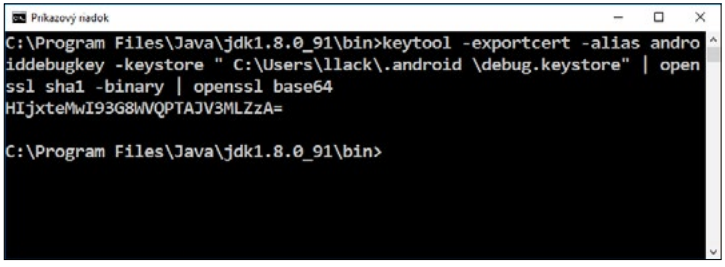
V našem případě byl příkaz ve tvaru:

```
keytool -exportcert -alias androiddebugkey -keystore "
C:\Users\llack\.android \debug.keystore" | openssl sha1 -binary |
  openssl base64
```

Po spouštění příkazu v konzolové aplikaci dostanete pravděpodobně následující chybové hlášení:

```
C:\Program Files\Java\jdk1.8.0_91\bin>keytool -exportcert -alias
  androiddebugkey -keystore " C:\Users\llack\.android \debug.keystore" |
  openssl sha1 -binary | openssl base64
'openssl' is not recognized as an internal or external command,
  operable program or batch file.
```

Abyste mohli příkaz spustit, musíte mít nainstalované openssl for windows, například z adresy *https://www.openssl.org/related/binaries.html*, nebo *http://gnuwin32.sourceforge.net/packages/openssl.htm*.



Obrázek 15.5: Generování ladicího klíče

Na vygenerování definitivního kódu použijte příkaz ve tvaru:

```
keytool -exportcert -alias YOUR_RELEASE_KEY_ALIAS -keystore
  YOUR_RELEASE_KEY_PATH | openssl sha1 -binary | openssl base64
```

Vygenerovaný 28znakový řetězec zadáte do pole **Key Hashes** na vývojářském portálu Facebooku.

Znovu přesuneme pozornost na projekt v Android Studiu. Do kódu hlavní aktivity do metody `Create` přidejte inicializaci Facebook SDK.

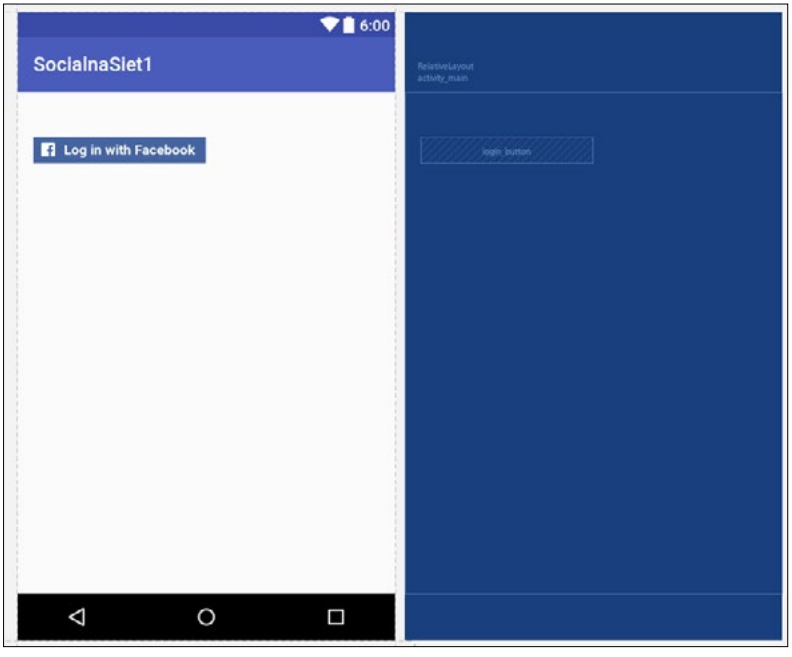
```
@Override
public void onCreate() {
  super.onCreate();
  FacebookSdk.sdkInitialize(getApplicationContext());
  AppEventsLogger.activateApp(this);
}
```

Layout hlavní aktivity bude obsahovat tlačítko na přihlášení se k sociální síti Facebook. Nemusíte ho vytvářet, k dispozici je widget `com.facebook.widget.LoginButton`.

```
RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.socialnasiet1.MainActivity">

  <com.facebook.login.widget.LoginButton
    android:id="@+id/login_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
```

```
    android:layout_marginTop="30dp"
    android:layout_marginBottom="30dp" />
</RelativeLayout>
```



Obrázek 15.6: Widget na přihlášení se k Facebooku v návrhovém zobrazení

V metodě `onCreateView()` je potřeba vytvořit callback, ve kterém můžete v případě potřeby nastavovat vlastnosti `LoginBehavior`, `DefaultAudience` či `ToolTipPopup.Style`. Například:

```
Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  FacebookSdk.sdkInitialize(this.getApplicationContext());

  callbackManager = CallbackManager.Factory.create();

  LoginManager.getInstance().registerCallback(callbackManager,
    new FacebookCallback<LoginResult>() {
    @Override
    public void onSuccess(LoginResult loginResult) {
      // App code
    }

    @Override
    public void onCancel() {
      // App code
    }

    @Override
```

```
    public void onError(FacebookException exception) {
      // App code
    }
  });
}
```

V kódu hlavní aktivity v souboru *MainActivity.java* je potřeba zaregistrovat callback objektu `CallbackManager` a ošetřit aplikačním kódem všechny tři varianty, tedy úspěšné přihlášení, zrušení přihlašování a chybu.

```
public class MainActivity extends FragmentActivity {

  LoginButton loginButton;
  CallbackManager callbackManager;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    FacebookSdk.sdkInitialize(this.getApplicationContext());

    callbackManager = CallbackManager.Factory.create();

    LoginManager.getInstance().registerCallback(callbackManager,
    new FacebookCallback<LoginResult>() {
      @Override
      public void onSuccess(LoginResult loginResult) {
        Context context = getApplicationContext();
        Toast.makeText(context, "Uspešne prihlásený",
          Toast.LENGTH_LONG).show();
      }

      @Override
      public void onCancel() {
        Context context = getApplicationContext();
        Toast.makeText(context, "Zrušené", Toast.LENGTH_LONG).show();
      }

      @Override
      public void onError(FacebookException exception) {
        Context context = getApplicationContext();
        Toast.makeText(context, "chyba", Toast.LENGTH_LONG).show();
      }
    });
  }

  @Override
  protected void onActivityResult(int requestCode,
    int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    callbackManager.onActivityResult(requestCode, resultCode, data);
  }
}
```

Pokud je přihlášení úspěšné, parametr `LoginResult` obsahuje `AccessToken` obsahující informace o přidělených a zamítnutých oprávněních. V naší aplikaci vypíšeme stav pomocí toastové notifikace. Metoda `onActivityResult()` přepošle výsledek přihlášení objektu `callbackManager` vytvořeného v metodě `onCreate()`.

Příklad aplikace na posílání statusů a obrázků

Uživatelské rozhraní aplikace obsahuje tlačítko na přihlášení se k účtu Facebooku a dvě tlačítka na poslání statusu a fotky. Všimněte si jmenného prostoru `xmlns:facebook="http://schemas.android.com/apk/res-auto"`.

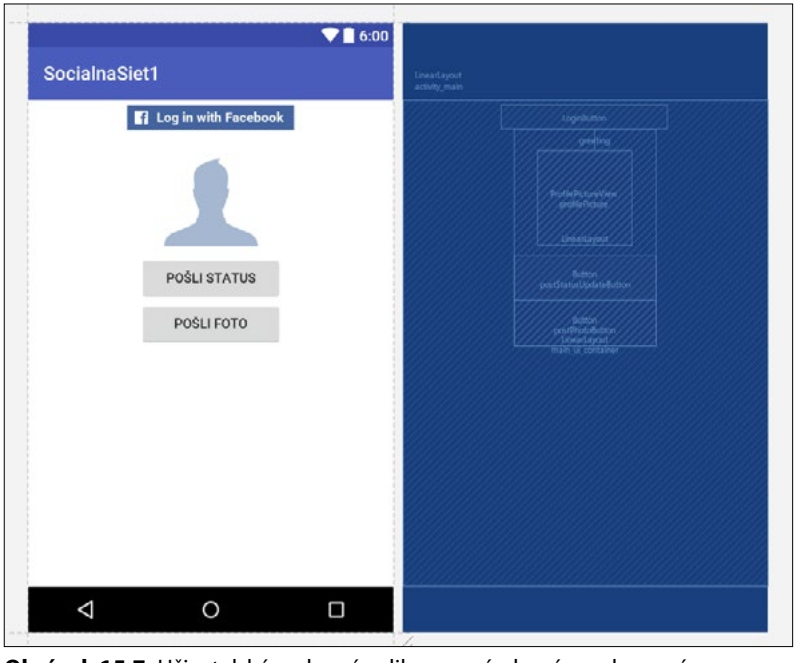
Kompletní kód uživatelského rozhraní hlavní aktivity:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:facebook="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="#FFF"
  tools:context="sk.pcrevue.llacko.socialnasiet1.MainActivity">

  <LinearLayout android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFF"
    android:id="@+id/main_ui_container">
    <com.facebook.login.widget.LoginButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginTop="5dp"
      android:layout_gravity="center_horizontal"
      facebook:com_facebook_confirm_logout="false"
      facebook:com_facebook_tooltip_mode="never_display"
    />
    <LinearLayout
      android:layout_width="150dp"
      android:layout_height="wrap_content"
      android:layout_gravity="center_horizontal"
      android:gravity="center_horizontal"
      android:orientation="vertical"
    >
      <TextView
        android:id="@+id/greeting"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="10dp"
        android:layout_gravity="center"
```

```
        android:textColor="#333"
        android:textSize="18sp"/>
      <com.facebook.login.widget.ProfilePictureView
        android:id="@+id/profilePicture"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:gravity="center_horizontal"
        android:layout_marginBottom="10dp"
        facebook:com_facebook_preset_size="normal"/>
      <Button
        android:id="@+id/postStatusUpdateButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Pošli status"
      />
      <Button
        android:id="@+id/postPhotoButton"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Pošli foto"
      />

    </LinearLayout>
  </LinearLayout>
</LinearLayout>
```



Obrázek 15.7: Uživatelské rozhraní aplikace v návrhovém zobrazení

Vytvořte třídu `MojFacebookBroadcastReceiver`. V tomto jednoduchém příkladu poslouží k ladění:

```
import android.os.Bundle;
import android.util.Log;
import com.facebook.FacebookBroadcastReceiver;

public class MojFacebookBroadcastReceiver extends FacebookBroadcastReceiver {
  @Override
  protected void onSuccessfulAppCall(String appCallId, String action,
    Bundle extras) {
    Log.d("SocialnaSiet1", String.format("Obrázok z aplikácie " +
    appCallId + " nahratý."));
  }

  @Override
  protected void onFailedAppCall(String appCallId, String action,
    Bundle extras) {
    Log.d("SocialnaSiet1", String.format("Obrázok z aplikácie " +
    appCallId + " CHYBA."));
  }
}
```

U takovéto komplexní aplikace uvádíme kompletní kód aplikačního manifestu:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.socialnasiet1">

  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <activity
      android:name="com.facebook.CustomTabActivity">
      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <category android:name="android.intent.category.BROWSABLE" />
        <data android:scheme="@string/fb_login_protocol_scheme" />
      </intent-filter>
    </activity>
```

```
    <meta-data android:name="com.facebook.sdk.ApplicationId"
      android:value="@string/facebook_app_id"/>

    <meta-data
      android:name="com.facebook.sdk.ApplicationName"
      android:value="@string/facebook_app_name" />

    <provider
      android:name="com.facebook.FacebookContentProvider"
      android:authorities=
        "com.facebook.app.FacebookContentProvider1334950249883434"
      android:exported="true" />

    <receiver android:name=
      "sk.pcrevue.llacko.socialnasiet1.MojFacebookBroadcastReceiver">
      <intent-filter>
        <action android:name="com.facebook.platform.AppCallResultBroadcast"/>
      </intent-filter>
    </receiver>
  </application>
</manifest>
```

Kód hlavní aktivity budeme vysvětlovat postupně. Začneme importy. Přesvědčte se, že v kódu hlavní aktivity máte všechny potřebné importy pro integraci funkcí Facebook SDK.

```
import android.app.AlertDialog;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.location.Location;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.FragmentActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.facebook.*;
import com.facebook.appevents.AppEventsLogger;
import com.facebook.login.LoginManager;
import com.facebook.login.LoginResult;
import com.facebook.login.widget.LoginButton;
import com.facebook.login.widget.ProfilePictureView;
import com.facebook.share.ShareApi;
import com.facebook.share.Sharer;
import com.facebook.share.model.SharePhoto;
import com.facebook.share.model.SharePhotoContent;
import com.facebook.share.model.ShareLinkContent;
```

```
import com.facebook.share.widget.ShareDialog;

import java.util.ArrayList;
import java.util.Arrays;
```

Pokračujeme deklarací proměnných. Aby bylo možné poslat i zeměpisné souřadnice, jsou deklarovány staticky pro město Ružomberok.

```
private static final String PERMISSION = "publish_actions";
  private static final Location RUZOMBEROK_LOCATION = new Location("") {
  {
    setLatitude(49.0748);
    setLongitude(19.30751);
  }
};

private final String PENDING_ACTION_BUNDLE_KEY =
  "sk.pcrevue.llacko.socialnasiet2";

private Button postStatusUpdateButton;
private Button postPhotoButton;
private ProfilePictureView profilePictureView;
private TextView greeting;
private PendingAction pendingAction = PendingAction.NONE;
private boolean canPresentShareDialog;
private boolean canPresentShareDialogWithPhotos;
private CallbackManager callbackManager;
private ProfileTracker profileTracker;
private ShareDialog shareDialog;
```

Metoda `FacebookCallback` vypisuje při ladění stavy jako úspěšné přihlášení, chyby či přerušení akce uživatelem.

```
private FacebookCallback<Sharer.Result> shareCallback = new
  FacebookCallback<Sharer.Result>() {
  @Override
  public void onCancel() {
    Log.d("SocialnaSiet1", "Zrusene");
  }

  @Override
  public void onError(FacebookException error) {
    Log.d("SocialnaSiet1", String.format("Chyba: %s", error.toString()));
    String title = "Chyba";
    String alertMessage = error.getMessage();
    showResult(title, alertMessage);
  }

  @Override
  public void onSuccess(Sharer.Result result) {
    Log.d("SocialnaSiet1", "O.K.!");
    if (result.getPostId() != null) {
      String title = "O.K";
```

```
        String id = result.getPostId();
        String alertMessage = String.format("Post ID: %1$s", id);
        showResult(title, alertMessage);
      }
    }

    private void showResult(String title, String alertMessage) {
      new AlertDialog.Builder(MainActivity.this)
      .setTitle(title)
      .setMessage(alertMessage)
      .setPositiveButton("O.K", null)
      .show();
    }
  };

  private enum PendingAction {
    NONE,
    POST_PHOTO,
    POST_STATUS_UPDATE
  }
```

V metodě `OnCreate` jsou ošetřené stavy jako úspěšné přihlášení, chyby či přerušení akce uživatelem a inicializované objekty Facebook SDK pro uživatelské rozhraní, jako jsou tlačítka či profilový obrázek přihlášeného uživatele.

```
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    FacebookSdk.sdkInitialize(this.getApplicationContext());

    callbackManager = CallbackManager.Factory.create();

    LoginManager.getInstance().registerCallback(callbackManager,
      new FacebookCallback<LoginResult>() {
      @Override
      public void onSuccess(LoginResult loginResult) {
        handlePendingAction();
        updateUI();
      }

      @Override
      public void onCancel() {
        if (pendingAction != PendingAction.NONE) {
          showAlert();
          pendingAction = PendingAction.NONE;
        }
        updateUI();
      }

      @Override
      public void onError(FacebookException exception) {
        if (pendingAction != PendingAction.NONE
```

```
      && exception instanceof FacebookAuthorizationException) {
      showAlert();
      pendingAction = PendingAction.NONE;
    }
    updateUI();
  }

  private void showAlert() {
    new AlertDialog.Builder(MainActivity.this)
    .setTitle("Zrušené")
    .setMessage("Privilégiá neboli udelené")
    .setPositiveButton("OK", null)
    .show();
  }
});

shareDialog = new ShareDialog(this);
shareDialog.registerCallback(
  callbackManager,
  shareCallback);

if (savedInstanceState != null) {
  String name = savedInstanceState.getString(PENDING_ACTION_BUNDLE_KEY);
  pendingAction = PendingAction.valueOf(name);
}

setContentView(R.layout.activity_main);

profileTracker = new ProfileTracker() {
  @Override
  protected void onCurrentProfileChanged(Profile oldProfile,
    Profile currentProfile) {
    updateUI();
    // It's possible that we were waiting for Profile
    // to be populated in order to
    // post a status update.
    handlePendingAction();
  }
};

profilePictureView = (ProfilePictureView)
  findViewById(R.id.profilePicture);
greeting = (TextView) findViewById(R.id.greeting);

postStatusUpdateButton = (Button)
  findViewById(R.id.postStatusUpdateButton);
postStatusUpdateButton.setOnClickListener(new View.OnClickListener() {
  public void onClick(View view) {
    onClickPostStatusUpdate();
  }
});
```

```
    postPhotoButton = (Button) findViewById(R.id.postPhotoButton);
    postPhotoButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        onClickPostPhoto();
      }
    });

    // Can we present the share dialog for regular links?
    canPresentShareDialog = ShareDialog.canShow(
      ShareLinkContent.class);

    // Can we present the share dialog for photos?
    canPresentShareDialogWithPhotos = ShareDialog.canShow(
      SharePhotoContent.class);
  }
```

Následují metody na obsluhu událostí:

```
  @Override
  protected void onResume() {
    super.onResume();
    updateUI();
  }

  @Override
  protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);

    outState.putString(PENDING_ACTION_BUNDLE_KEY, pendingAction.name());
  }

  @Override
  protected void onActivityResult(int requestCode, int resultCode, Intent data){
    super.onActivityResult(requestCode, resultCode, data);
    callbackManager.onActivityResult(requestCode, resultCode, data);
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    profileTracker.stopTracking();
  }
```

A metody na poslání statusu a obrázků:

```
  private void updateUI() {
    boolean enableButtons = AccessToken.getCurrentAccessToken() != null;

    postStatusUpdateButton.setEnabled(enableButtons || canPresentShareDialog);
    postPhotoButton.setEnabled(enableButtons ||
      canPresentShareDialogWithPhotos);
```

```
  Profile profile = Profile.getCurrentProfile();
  if (enableButtons && profile != null) {
    profilePictureView.setProfileId(profile.getId());
    greeting.setText(String.format("Hello %1$s!", profile.getFirstName()));
  } else {
    profilePictureView.setProfileId(null);
    greeting.setText(null);
  }
}

private void handlePendingAction() {
  PendingAction previouslyPendingAction = pendingAction;
  // These actions may re-set pendingAction if they are still pending,
  // but we assume they
  // will succeed.
  pendingAction = PendingAction.NONE;

  switch (previouslyPendingAction) {
    case NONE:
      break;
    case POST_PHOTO:
      postPhoto();
      break;
    case POST_STATUS_UPDATE:
      postStatusUpdate();
      break;
  }
}

private void onClickPostStatusUpdate() {
  performPublish(PendingAction.POST_STATUS_UPDATE, canPresentShareDialog);
}

private void postStatusUpdate() {
  Profile profile = Profile.getCurrentProfile();
  ShareLinkContent linkContent = new ShareLinkContent.Builder()
    .setContentTitle("SocialnaSiet1")
    .setContentDescription(
    "SocialnaSiet1 príklad integrácie s Facebookom")
    .setContentUrl(Uri.parse("http://developers.facebook.com/docs/android"))
    .build();
  if (canPresentShareDialog) {
    shareDialog.show(linkContent);
  } else if (profile != null && hasPublishPermission()) {
    ShareApi.share(linkContent, shareCallback);
  } else {
    pendingAction = PendingAction.POST_STATUS_UPDATE;
  }
}

private void onClickPostPhoto() {
```

```
    performPublish(PendingAction.POST_PHOTO, canPresentShareDialogWithPhotos);
  }

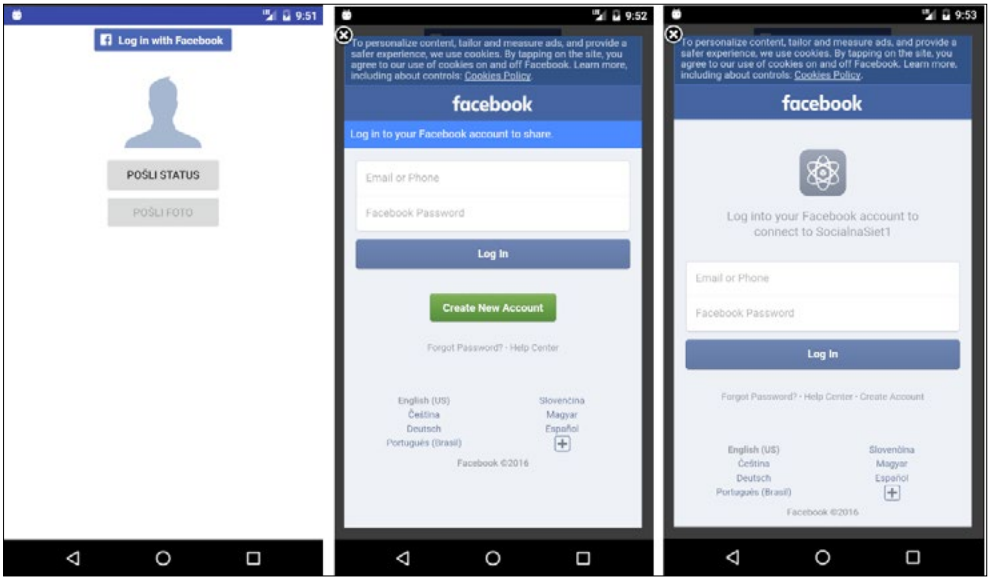
  private void postPhoto() {
    Bitmap image = BitmapFactory.decodeResource(this.getResources(),
      R.drawable.icon);
    SharePhoto sharePhoto = new SharePhoto.Builder().setBitmap(image).build();
    ArrayList<SharePhoto> photos = new ArrayList<>();
    photos.add(sharePhoto);

    SharePhotoContent sharePhotoContent =
      new SharePhotoContent.Builder().setPhotos(photos).build();
    if (canPresentShareDialogWithPhotos) {
      shareDialog.show(sharePhotoContent);
    } else if (hasPublishPermission()) {
      ShareApi.share(sharePhotoContent, shareCallback);
    } else {
      pendingAction = PendingAction.POST_PHOTO;
      // We need to get new permissions, then complete the action
      // when we get called back.
      LoginManager.getInstance().logInWithPublishPermissions(
        this,
        Arrays.asList(PERMISSION));
    }
  }

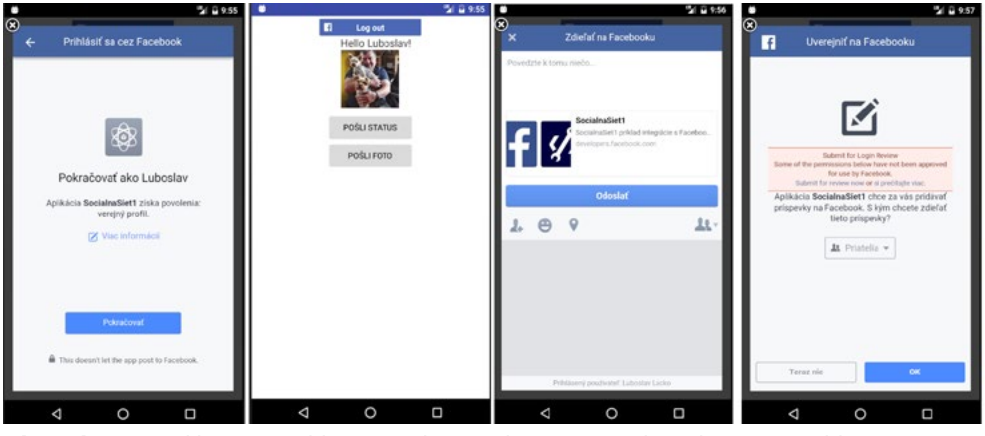
  private boolean hasPublishPermission() {
    AccessToken accessToken = AccessToken.getCurrentAccessToken();
    return accessToken != null &&
      accessToken.getPermissions().contains("publish_actions");
  }

  private void performPublish(PendingAction action, boolean allowNoToken) {
    AccessToken accessToken = AccessToken.getCurrentAccessToken();
    if (accessToken != null || allowNoToken) {
      pendingAction = action;
      handlePendingAction();
    }
  }
```

Nyní můžete aplikaci spustit. Některé funkce na emulátoru nebudou fungovat, protože aplikace není publikovaná do aplikačního obchodu Google Play.



Obrázek 15.8: Uprostřed pokus o poslání statusu bez přihlášení, vpravo přihlášení se k účtu Facebooku



Obrázek 15.9: Aplikace po přihlášení, 3. zleva poslání statusu, obrázek vpravo problém s oprávněními

KAPITOLA

16

Cloudové služby

Témata kapitoly:

- **Google Cloud Messaging**
- **Firebase Cloud Messaging**
- **Přijímání notifikací na popředí**
- **Posílání zpráv přes Azure Notification Hub**
- **Připojení aplikace pro Android k mobilní službě**

Z hlediska principu fungování můžeme mobilní aplikace rozdělit na dvě základní skupiny.

- Aplikace fungující autonomně – aplikace, zpravidla jednodušší, které se nepotřebují připojovat k externím zdrojům údajů, službám ani serverovým aplikacím.
- Aplikace využívající cloudové či mobilní služby, externí databáze, případně aplikace připojující se k middlewarové aplikační vrstvě.

TIP

Chystáte se tuto kapitolu přeskočit, protože neplánujete vyvíjet podnikové aplikace? Zamyslete se nad triviálním scénářem. Vytvořili jste hru a chcete, aby její součástí byl i žebříček top skóre nejúspěšnějších hráčů. Jak byste bez připojení ke cloudovým službám nebo databázi tuto funkcionalitu implementovali?

Vzhledem k rozvoji cloudových platforem a jejich stále častějšímu využívání se většina moderních mobilních aplikací v blízké budoucnosti bez využívání cloudových služeb neobejde.

V první cvičné aplikaci využívající Firebase Cloud Messaging (FCM) ukážeme, jak posílat notifikaci (oznámení) z aplikačního serveru do aplikace na Androidu. V druhé cvičné aplikaci ukážeme připojení k mobilní službě s databází a odesílání a přijímání údajů z této služby. Takto je možné například implementovat už zmíněný žebříček top skóre nejúspěšnějších hráčů.

Google Cloud Messaging

Google Cloud Messaging (GCM) je bezplatná služba, která pomáhá vývojářům odesílat údaje ze serverů do jejich aplikace pro systém Android. Může se jednat o jednoduchou zprávu o tom, že daná aplikace pro Android má na serveru nové údaje připravené ke stažení (například video nahrané přítelem). Také to může být zpráva obsahující až 4 kB údajů, takže například chatovací aplikace mohou zprávu využívat přímo. Služba Google Cloud Messaging, respektive její notifikační systém, je

součástí aplikace Google Play Services. Po posledních vylepšeních umožňuje trvalé připojení, upstream, odesílání zpráv a synchronizaci notifikací.

Funkce trvalého připojení (Persistent connections) poskytuje vývojářům jednoduchou možnost, jak odesílat velké množství zpráv velkému množství zařízení. Upstream messaging umožňuje vývojářům posílat zprávy v opačném směru, tedy z aplikace na vlastní servery. Poslední a možná nejzajímavější novinkou pro uživatele je synchronizace notifikací. V praxi funguje tak, že když uživatel vymaže notifikaci na jednom zařízení, automaticky se vymaže i z ostatních.

TIP

Pro nové aplikace doporučuje Google použít místo Google Cloud Messaging novou službu Firebase Cloud Messaging (FCM). Také u inovací existujících aplikací se doporučuje upgradovat GCM na FCM.

Firebase Cloud Messaging

Společnost Google získala v roce 2014 akvizicí společnosti Firebase stejnojmennou cloudovou databázi a množinu cloudových služeb na ukládání a správu údajů, notifikace, analytiku, autentizaci a další služby. Ke všem službám je k dispozici rozhraní na programování a soubor knihoven určený aplikacím pro Android. Služby se spravují přes webové rozhraní.

Cloudové služby jsou vynikající možností nejen na ukládání údajů, ale i na posílání oznámení do aplikací. Pomocí webové konzole můžete poslat všem uživatelům své aplikace oznámení například o školení, nové verzi a podobně. Posílání a přijímání oznámení bude i námětem ukázkového příkladu.

POZNÁMKA

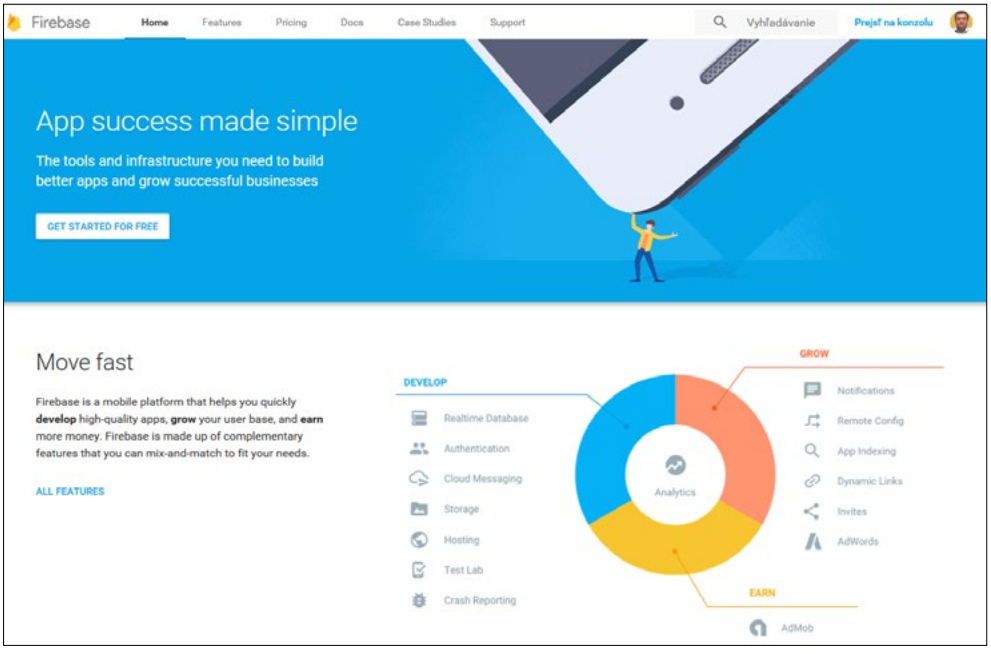
Podpora Firebase je integrovaná přímo do vývojového prostředí Android Studio.

Služba Firebase Cloud Messaging (FCM) umožňuje bezplatně posílat zprávy mezi systémy Android, iOS, Chrome a webovými aplikacemi využívajícími JavaScript. Server může například odesílat zprávy přímo na jednotlivá zařízení, skupiny zařízení nebo na zařízení přihlášená k odběru témat. Aplikace na zařízení může také odesílat zprávy přímo na server a na zařízení ze stejné skupiny.

POZNÁMKA

Cloudová platforma Firebase má mnohem více funkcí včetně databáze real time, storage, hostingu, testovacích a dalších funkcí. Firebase Cloud Messaging je jen jednou ze služeb platformy.

Vstupním bodem do služby Firebase je adresa URL *https://firebase.google.com/*. Jako první krok klepněte na odkaz **Get Started for Free**. Rozsáhlejší balíčky jsou zpoplatněny, pro účely aplikace hobby ale stačí bezplatný balíček SPARK:



Obrázek 16.1: Domovská stránka Firebase https://firebase.google.com/

		SPARK Free Generous limits for hobbyists	FLAME $25 per month Predictable pricing for growing apps	BLAZE Pay as you go Commodity pricing for apps at scale
Included Free Analytics, App Indexing, Authentication, Cloud Messaging, Crash Reporting, Dynamic Links, Invites, Notifications & Remote Config		✓	✓	✓
Realtime Database	Simultaneous connections	100	Unlimited[1]	Unlimited[1]
	GB stored	1 GB	2.5 GB	$5/GB
	GB downloaded	10 GB/month	20 GB/month	$1/GB
	Automated backups	✗	✗	✓
Storage	GB stored	5 GB	50 GB	$0.026/GB[2]
	GB downloaded	1 GB/day	50 GB/month	$0.12/GB[2]
	Upload operations	20,000/day	100,000/day	$0.10/thousand[2]
	Download operations	50,000/day	250,000/day	$0.01/thousand[2]
Hosting	GB stored	1 GB	10 GB	$0.026/GB
	GB transferred	10 GB/month	50 GB/month	$0.15/GB
	Custom domain hosting & SSL	✓	✓	✓
Test Lab	Daily quota or hourly rate	15 tests/day[3]	15 tests/day[3]	hourly, per device[4]
Google Cloud Platform	Use BigQuery & other IaaS[5]	✗	✗	✓

Obrázek 16.2: Porovnání možností a limitů bezplatného balíčku SPARK a komerčních balíčků FLAME a BLAZE služby Firebase

Jak funguje Firebase Cloud Messaging? Klientské zařízení, v tomto případě chytrý telefon nebo tablet s Androidem, se přihlásí k serveru FCM, to znamená k serverům v datovém centru, na kterých běží cloudová služba Firebase. Server FCM přiřadí zařízení takzvaný token FCM. Jak aplikace využívající FCM komunikuje na pozadí se serverem, odešle na něj token. Jednodušší (například chatovací), aplikace server nepotřebují, vystačí s posíláním zpráv. Služba Firebase Cloud Messaging má samozřejmě vlastní databázi, do které se všechny zprávy ukládají.

Zprávy FCM

Zprávy je možné odeslat buď na konkrétní zařízení, případně na zařízení, která jsou přihlášená k odběru nějakého tématu, nebo na skupinu zařízení, která sdílejí stejný identifikační kód skupiny. V praxi jsou to většinou různá zařízení (telefon, tablet...) toho samého uživatele. Zprávy mohou být:

- **notifikační**, které zařízení zobrazí jako notifikaci
- **datové**, pomocí kterých aplikace získává údaje, jež následně vhodným způsobem zpracuje a zobrazí

Na posílání zpráv se využívají protokoly HTTP a XMPP. Údaje pro oba dva typy zpráv jsou ve formátu JSON. Notifikační zpráva může obsahovat maximálně 2 kB údajů a datová zpráva maximálně 4 kB údajů, takže podobně jako u GCM mohou některé (například chatovací) aplikace využívat spojení přímo.

Příklad notifikační zprávy, která bude mít titulek „Česko vs. Slovensko“ a v těle zprávy text „výborný zápas!“:

```
{
  "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification" : {
    "body" : "výborný zápas!",
    "title" : "Česko vs. Slovensko",
    "icon" : "myicon"
  }
}
```

Aplikace po přijetí zprávy dostane událost `onMessageReceived()`, kterou v případě, že se jí zpráva týká, může obsloužit.

Příklad datové zprávy:

```
{
  "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "data" : {
    "Nick" : "lubolacko",
    "body" : "výbornýzápas!",
    "Room" : "ČeskoVsSlovensko "
  },
}
```

Zpráva může obsahovat zároveň notifikační i datovou část:

```
{
  "to" : "APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...",
    "notification" : {
```

```
    "body" : "great match!",
      "title" : "Portugal vs. Denmark",
      "icon" : "myicon"
  },
  "data" : {
    "Nick" : "Mario",
    "Room" : "PortugalVSDenmark"
  }
}
```

Chování aplikace při příjmu zprávy, která obsahuje zároveň notifikační i datovou část, závisí na tom, zda je aplikace na pozadí či popředí (tj. zda je nebo není aktivní v čase přijetí). Pokud je aplikace na pozadí, zobrazí se jen notifikace a údaje v datové části se zpracují tehdy, když uživatel klepne na notifikaci. Pokud je aplikace v popředí, má aplikace k dispozici zprávu i s údaji a může ji zpracovat, zobrazit a podobně.

Zprávy mohou být typu non-collapsible a collapsible; původní terminologii jsme použili úmyslně. Zpráva non-collapsible funguje tak, že každá jednotlivá zpráva se doručí do zařízení. Každá zpráva tedy obsahuje další nebo doplňkovou informaci. Například v aplikaci IM chcete doručit každou zprávu, protože každá zpráva má jiný obsah. FCM nezaručuje pořadí jejich doručení. Limit je 100 uložených zpráv pro zařízení, po dosažení tohoto limitu, například když je telefon či tablet vypnutý nebo offline, se všechny uložené zprávy zahodí. Pokud se přístroj znovu připojí do režimu online, dostane speciální zprávu oznamující, že došlo k dosažení limitu. Aplikace potom může vhodně zvládnout situaci – obvykle požaduje úplnou synchronizaci aplikace a serveru.

Naproti tomu zpráva collapsible může bát nahrazena novou. Typickým příkladem je aktualizace notifikace o stavu probíhajícího sportovního zápasu. Pouze nejnovější zpráva je relevantní. Zprávy mohou mít normální a vysokou prioritu. Vysokou prioritu mají zpravidla notifikační a nižší datové zprávy. Priorita se definuje atributem `priority`:

```
{
  "to" : "bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
  "priority" : "normal",
  "notification" : {
    "body" : "Nové číslo časopisu v predaji.",
    "title" : "PC REVUE",
    "icon" : "new"
  },
  "data" : {
    "volume" : "3.21.16",
    "contents" : "http://www.pcrevue.sk"
  }
}
```

Životnost zprávy se udává parametrem `time_to_live`, který může být v intervalu 0 až 2 419 200 sekund.

```
{
  "collapse_key" : "demo",
  "to" : "aaazzz",
  "data" : {
    "key1" : "hodnota1",
```

```
      "key2" : "hodnota2"
    },
    "time_to_live" : 3
}
```

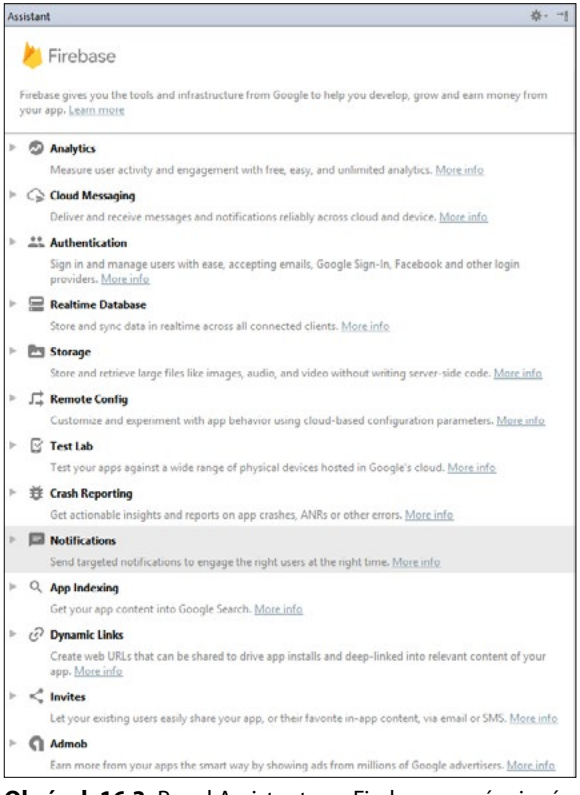
FCM umožňuje aplikaci přijímat zprávy od více odesílatelů. Předpokládejme například, že aplikace je agregátorem článků nebo blogových příspěvků od několika přispěvatelů a každý z nich by měl být schopen odeslat zprávu, když zveřejní nový článek. Tato zpráva může obsahovat adresu URL, aby aplikace mohla stáhnout článek. Limit je 100 odesílatelů.

Příklad na posílání zpráv

Na uvítací stránce cloudové služby Firebase najděte odkaz **Create New Project**. Ale nepředbíhejme, vraťme se do vývojového prostředí Android Studio, ve kterém vytvořte nový projekt. Projekt v tomto vývojovém prostředí může využívat Firebase prostřednictvím integrovaného pluginu. Aktivuje se pomocí nabídky **Tools → Firebase**. Zobrazí se panel **Assistant** s nabídkou nástrojů.

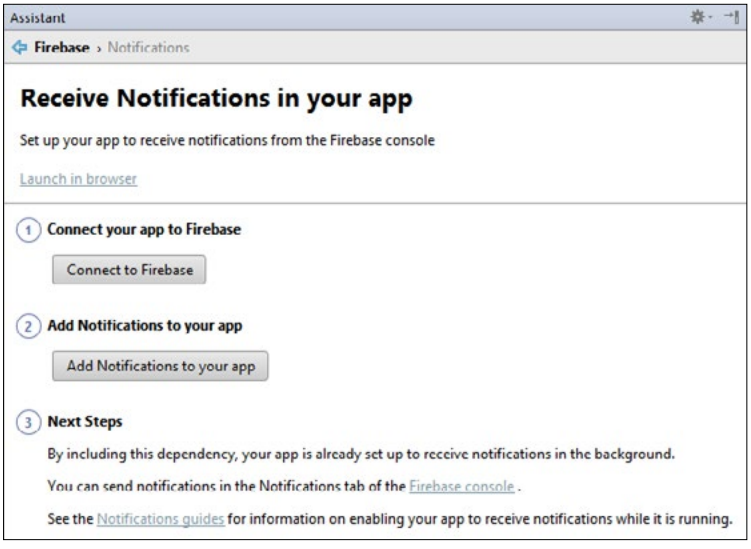
TIP

Doporučujeme přezkoumat nabídku nástrojů a zamyslet se, zda by se daly využít ve vašich aplikacích.

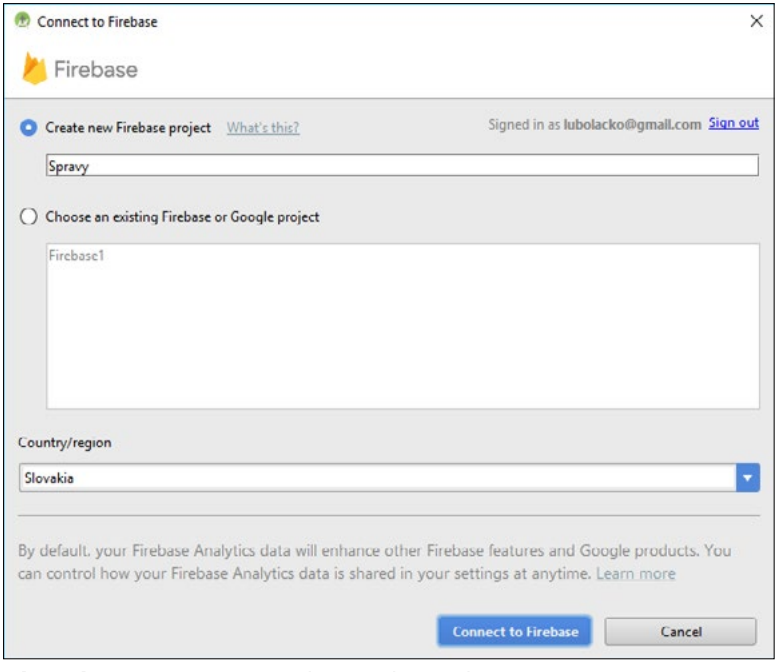


Obrázek 16.3: Panel Assistant pro Firebase ve vývojovém prostředí Android Studio

Vyberte položku **Notifications**, klepnutím na šipku zobrazíte odkaz **Receive Notifications in your app**. Zobrazí se stejnojmenný panel s postupem pro notifikace. Abyste mohli vytvořit nový projekt Firebase a napojit ho na projekt v Android Studiu, musí být aplikace připojená k Firebase. Proto postup jako první krok obsahuje tlačítko **Connect to Firebase**.



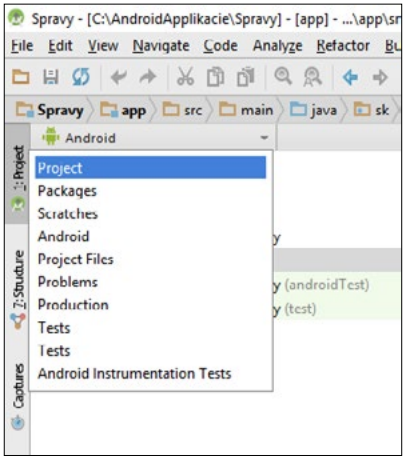
Obrázek 16.4: Panel s postupem pro notifikace



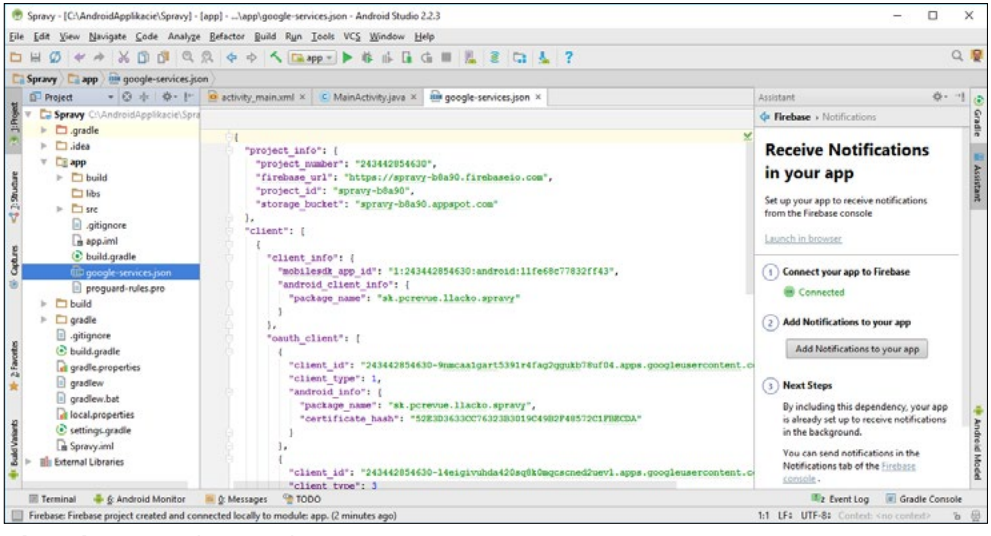
Obrázek 16.5: Vytvoření nového projektu Firebase

Při prvním připojení se zobrazí okno prohlížeče, které vyžaduje povolení přístupu k účtu Google. Povolení musíte akceptovat. Následuje vytvoření nového projektu Firebase. Pod pojmem projekt nemáme nyní na mysli projekt aplikace pro Android, ten už máme vytvořený, ale vytvoření a konfiguraci projektu v příslušné cloudové službě.

Po potvrzení vytvoření připojení se dialog zavře a tlačítko **Connect to Firebase** na panelu v Android Studiu se po pár sekundách změní na zelené oznámení **Connected**. Zároveň se do projektu přidal soubor *google-services.json*. Abyste ho zobrazili a prozkoumali, přepněte levý panel **Project** z implicitní volby **Android** na **Project**.

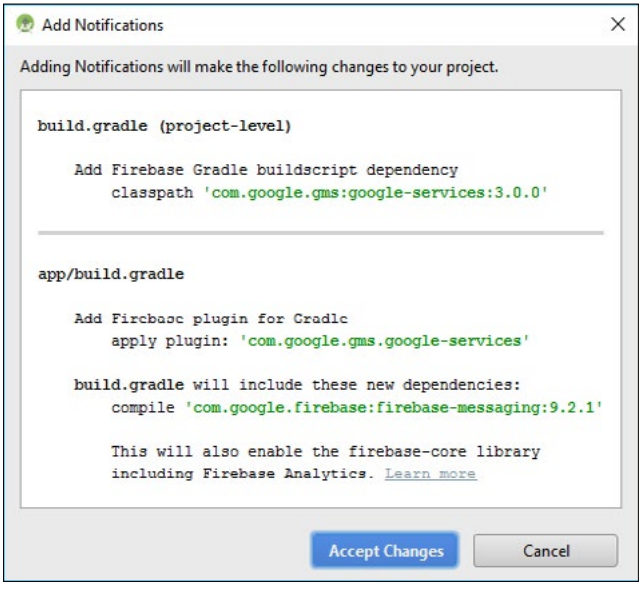


Obrázek 16.6: Přepnutí levého panelu Project v Android Studiu



Obrázek 16.7: Soubor google-services.json

Pokračujeme krokem 2. Klepněte na tlačítko **Add Notifications to your App**. Zobrazí se dialog se změnami, které se po potvrzení provedou v projektu. Změny se týkají dvou verzí definičního souboru k sestavení *build.gradle*.



Obrázek 16.8: Potvrzení změn

Soubor *build.gradle* (Project):

```
buildscript {
  repositories {
    jcenter()
  }
  dependencies {
    classpath 'com.android.tools.build:gradle:2.2.3'
    classpath 'com.google.gms:google-services:3.0.0'
  }
}
```

Soubor *build.gradle* (Module:app):

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 24
  buildToolsVersion "24.0.0"
  defaultConfig {
    applicationId "sk.pcrevue.llacko.spravy"
    minSdkVersion 22
    targetSdkVersion 24
    versionCode 1
    versionName "1.0"
    testInstrumentationRunner
```

```
        "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
      release {
        minifyEnabled false
          proguardFiles getDefaultProguardFile('proguard-android.txt'),
          'proguard-rules.pro'
      }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2',
{
      exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.google.firebase:firebase-messaging:10.0.1'
    testCompile 'junit:junit:4.12'
}

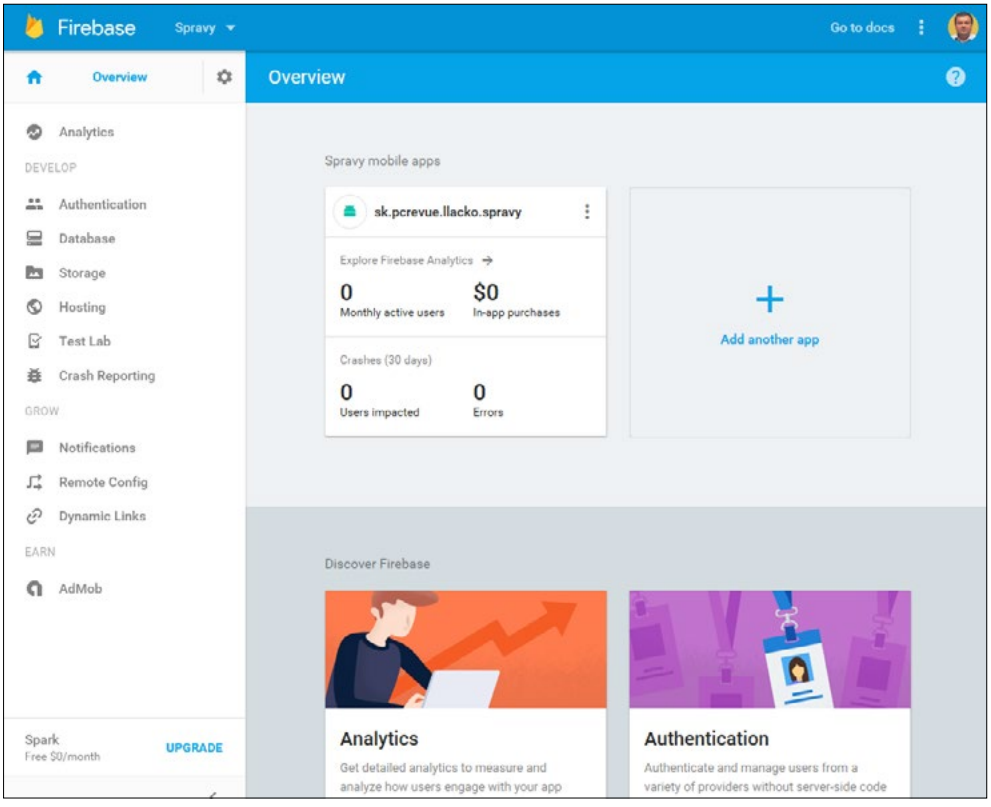
apply plugin: 'com.google.gms.google-services'
```

Konfigurace FCM

Abyste mohli ladit přijímání notifikací, je potřeba poslat notifikaci pro konkrétní aplikaci – tu, jejíž projekt vytváříte v Android Studiu. Ve webovém prohlížeči zadejte adresu URL Firebase konzole *https://console.firebase.google.com/*. Vyberte projekt aplikace, které chcete posílat zprávy. Na levém panelu vyberte položku **Notification**. Doporučujeme prozkoumat i možnosti dalších služeb.

Tlačítkem **Send Your First Message** zobrazíte obrazovku, na které zadáte textovou zprávu. Do pole **Message label (optional)** můžete zadat identifikátor zprávy. Ten se uživateli v aplikaci nezobrazí, slouží k identifikaci zprávy v systému Firebase. V sekci **Target** si všimněte možnosti zacílení notifikace na různé skupiny uživatelů aplikací. Implicitně nastavená volba **User Segment** definuje, že oznámení bude zacíleno na všechny uživatele vaší aplikace. Pokud byste měli více verzí či jazykových mutací vaší aplikace, můžete oznámení zacílit přesněji, například jen ve španělštině.

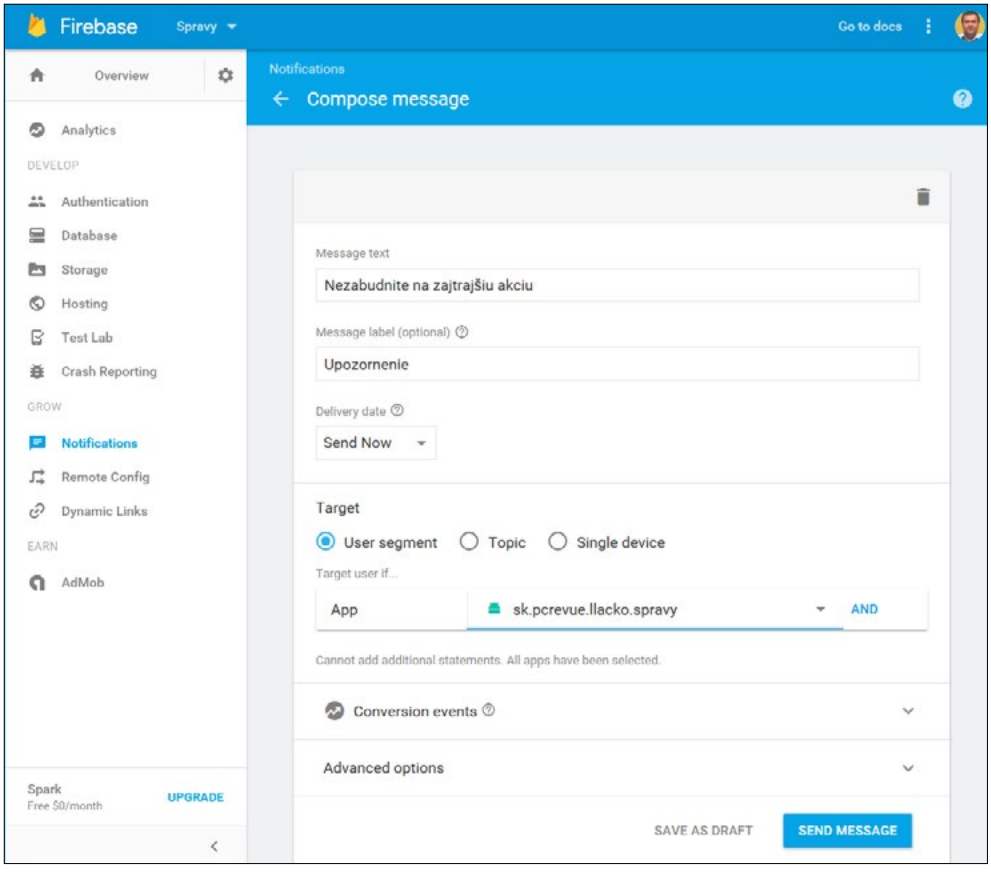
V předvoleném nastavení se název shoduje s názvem aplikace. V sekci **Advanced options** to můžete změnit. Také můžete nastavit prioritu, zvuk notifikace, dobu expirace oznámení a podobně. Standardně nastavená doba expirace je 4 týdny.



Obrázek 16.9: Webová konzole služby Firebase pro vybranou aplikaci

Po potvrzení následuje dialog s rekapitulací zadaných údajů notifikace, ve kterém můžete notifikaci odeslat.

Notifikace se odešle, zatím ji však nemá kdo přijmout. Momentálně je, jak se populárně říká, v cloudu a čeká na spuštění aplikace, jíž je tato notifikace určena. Nejprve je ale potřeba takovouto aplikaci dokončit a spustit. Notifikace má životnost 4 týdny, takže na vytvoření aplikace máte dost času. Zabere to jen několik minut. Vraťte se do projektu aplikace v Android Studiu, odkud jste proces notifikace inicializovali, a spusťte aplikaci na reálném zařízení připojeném k internetu. Notifikační lištu otevřete gestem posunu prstu od horní hrany displeje směrem dolů.



Obrázek 16.10: Dialog na zacílení zprávy

Přijímání notifikací na popředí

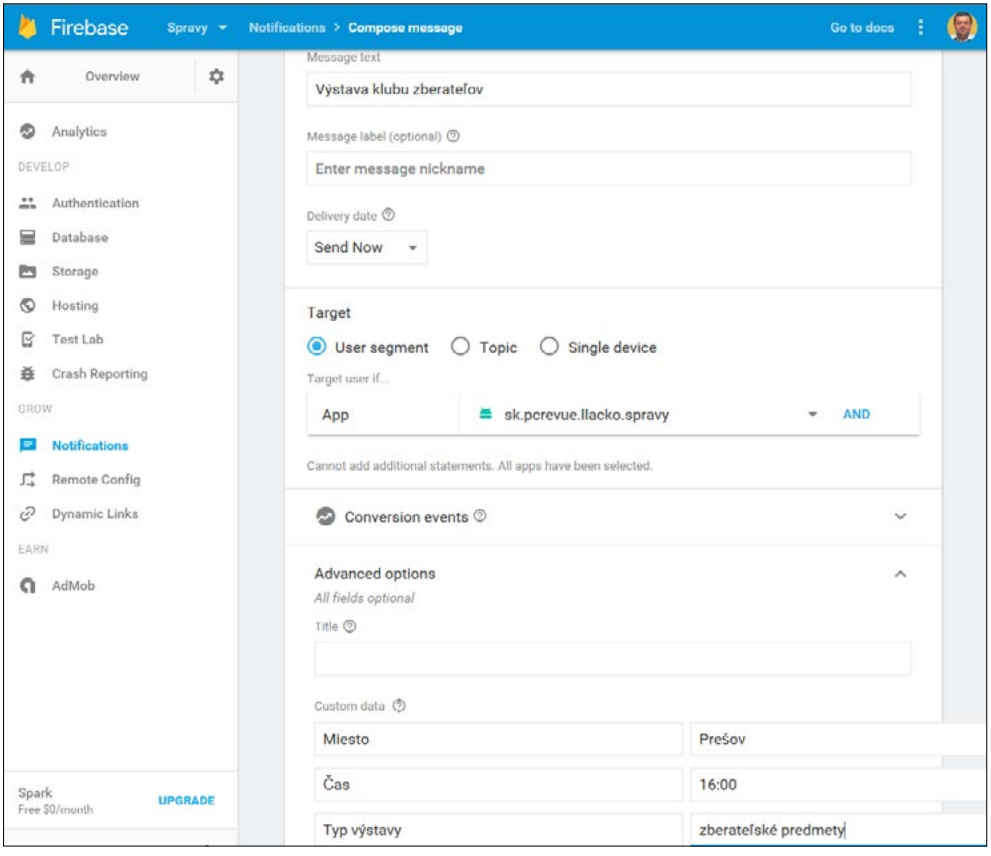
Dosud byla naše aplikace schopná přijímat notifikace posílané přes konzoli Firebase na pozadí. Víc možností poskytuje přijímání a zpracování notifikací přímo v aplikaci. Notifikace může kromě textu obsahovat i údaje ve tvaru: název parametru – hodnota. Tyto parametry se zadávají v sekci **Advanced options** a dají se zobrazit v uživatelském rozhraní aplikace.

V návrhu uživatelského rozhraní máme prvek `TextView` s identifikátorem `tw`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
```

```
    tools:context="sk.pcrevue.llacko.spravy.MainActivity">

    <TextView
      android:id="@+id/tw"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!" />
</RelativeLayout>
```



Obrázek 16.11: Parametry zprávy

Využijeme ho na zobrazení notifikace za běhu aplikace na popředí. Do souboru hlavní aktivity doplňte kód:

```
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Bundle bData = getIntent().getExtras();
```

```
    if (bData != null) {
      TextView textView = (TextView) findViewById(R.id.tw);
      textView.setText(bData.getString("Miesto"));
    }
  }
}
```

Ve složce *Java* vytvořte novou službu pomocí nabídky **New** → **Service** → **Service**. Pojmenujte ji například `FcmMessageService`. Volby **Exported** a **Enabled** nechejte označené. Ve službě se obslouží událost `onMessageReceived()`.

Abyste mohli fungování posílání notifikací sledovat v Android Studiu (konkrétně na dolním panelu **Android Monitor**) a filtrovat zprávy, přidejte do kódu služby textový řetězec `TAG`. Monitorovat budeme událost, přesněji její obsluhu `onMessageReceive()`.

Kód služby *FbMessageService.java*:

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import com.google.firebase.messaging.FirebaseMessagingService;
import android.util.Log;
import com.google.firebase.messaging.RemoteMessage;

public class FcmMessageService extends FirebaseMessagingService {
  public FcmMessageService() {
  }

  String TAG = "firebaseoznam";

  @Override
  public void onMessageReceived(RemoteMessage remoteMessage) {
    Log.d(TAG, "Nazov: " + remoteMessage.getNotification().getTitle());
    Log.d(TAG, "správa: " + remoteMessage.getNotification().getBody());
  }
}
```

Do manifestu aplikace je potřeba doplnit sekci `<intent-filter>`. Sekce `<service>` se v manifestu vytvořila automaticky při vytváření služby pomocí nabídky **New** → **Service** → **Service**.

```
<service
  android:name=".FbMessageService"
  android:enabled="true"
  android:exported="true">
  <intent-filter>
    <action android:name="com.google.firebase.MESSAGING_EVENT"/>
  </intent-filter>
</service>
```

UPOZORNĚNÍ

Pokud se v okně **Android monitor** vypíše hlášení: `Service connection failed: ConnectionResult{statusCode=SERVICE_VERSION_UPDATE_REQUIRED`, je potřeba aktualizovat verze knihoven Firebase a Google Play Services na vašem zařízení.

Výpis v okně **Android Monitor** bude mít po přijetí zprávy tvar:

```
D/FA: Connected to remote service
V/FA: Processing queued up service tasks: 2
V/FA: Inactivity, disconnecting from the service
I/FA: Tag Manager is not found and thus will not be used
D/FA: Logging event (FE): _nr, Bundle[{_o=fcm, _sc=MainActivity,
  _si=-5349226475463111052, _ndt=0, _nmt=1482682645,
  _nmid=5676010998671448212}]
D/firebaseoznam: Nazov: null
D/firebaseoznam: správa: Výstava klubu zberateľov
V/FA: Using measurement service
V/FA: Connecting to remote service
D/FA: Logging event (FE): _nf, Bundle[{_o=fcm, _sc=MainActivity,
  _si=-5349226475463111052, _ndt=0, _nmt=1482682645,
  _nmid=5676010998671448212}]
V/FA: Using measurement service
V/FA: Connection attempt already in progress
D/FA: Connected to remote service
V/FA: Processing queued up service tasks: 2
V/FA: Inactivity, disconnecting from the service
```

Posílání zpráv přes Azure Notification Hub

Abychom mohli vytvořit aplikaci pro Android, která pracuje s cloudovými notifikacemi, potřebujeme aplikační server třetí strany, případně cloudovou platformu, kde běží aplikace na pozadí, jež poskytuje údaje pro mobilní aplikace, a tato aplikace potřebuje posílat notifikace do všech připojených aplikací, případně do aplikace konkrétního uživatele. Kromě služby Firebase Cloud Messaging můžete využít i cloudové služby třetích stran. V příkladu využijeme službu Azure Notification Hub, která je součástí cloudové platformy Microsoft Azure. Ukážeme připojení na mobilní službu s databází, odesílání a přijímání údajů z této služby.

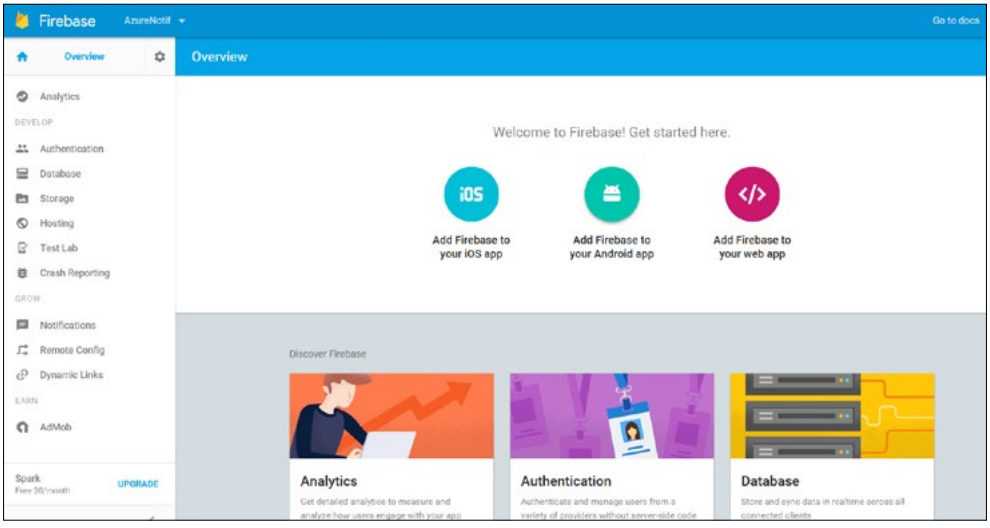
Možná si položíte otázku, proč jsme vybrali platformu Microsoftu a nevytvářeli například serverovou aplikaci v PHP. Hledali jsme nejjednodušší řešení, kterým je 30denní zkušební verze dostupná všem. Nepotřebujete zřizovat žádný serverhosting ani nic podobného. Jednoduše si vytvoříte účet v cloudové službě. Navíc na platformě Windows se využívá programovací jazyk C#, který vychází ze syntaxe Javy a jehož syntaxe je na 95 procent shodná s Javou.

Vytvoření projektu v Android Studiu

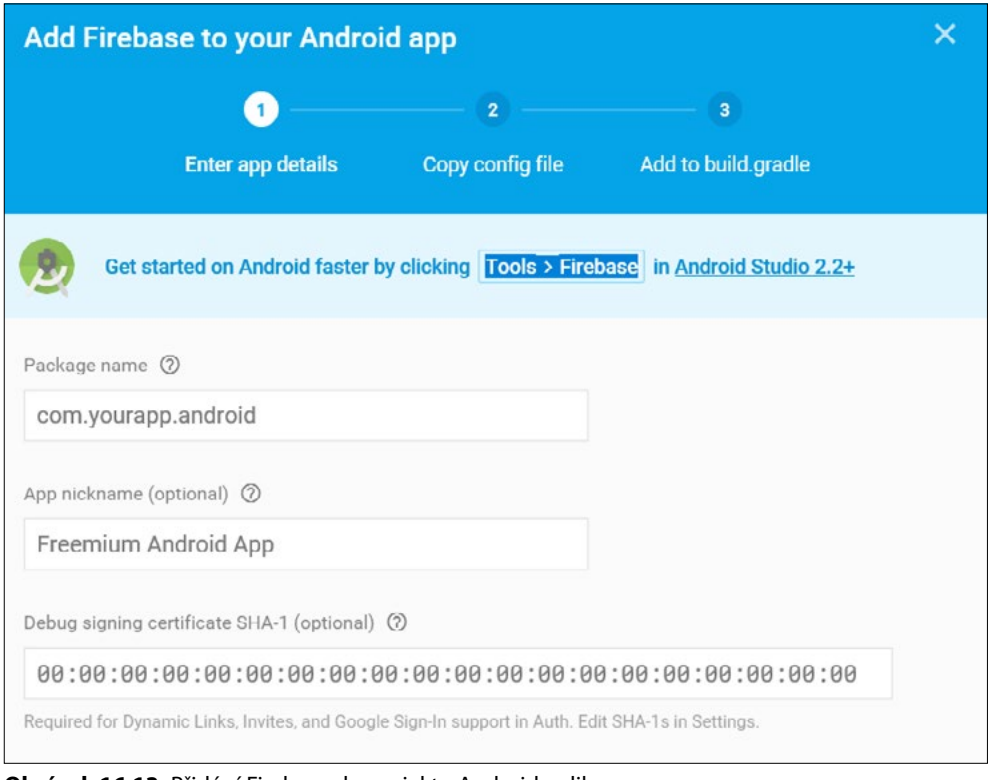
Tento krok je předběžný a dá se popsat jedinou větou. Vytvořte projekt a jako typ hlavní aktivity nastavte **Empty Activity**.

Vytvoření FCM

Podobně jako v předchozím příkladu zadejte ve webovém prohlížeči adresu URL konzole: Firebase *https://console.firebase.google.com*. V předchozím příkladu jsme řešili integraci Firebase do projektu aplikace pro Android v Android Studiu. To je nejjednodušší způsob. Pro úplnost ukážeme, jak vytvořit projekt ve webové konzolei Firebase a propojit ho s projektem aplikace pro Android. Klepněte na odkaz **Add Firebase to your Android App** a postupujte podle pokynů.

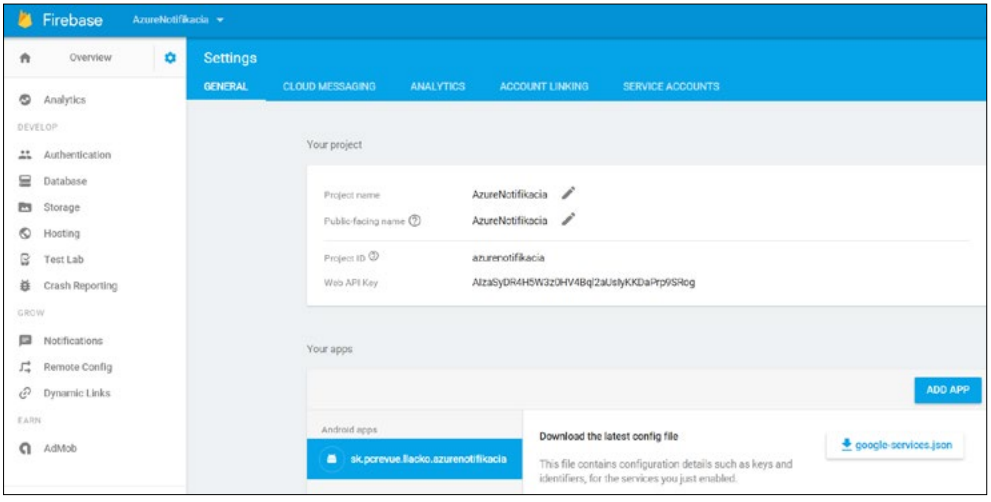


Obrázek 16.12: Projekt v konzoli služby Firebase



Obrázek 16.13: Přidání Firebase do projektu Android aplikace

Na propojení Firebase Cloud Messaging s Azure Notification Hub budete potřebovat Web API Key, který zkopírujete z nastavení projektu Firebase.



Obrázek 16.14: Získání Web API Key

Vytvoření Azure Notification Hub

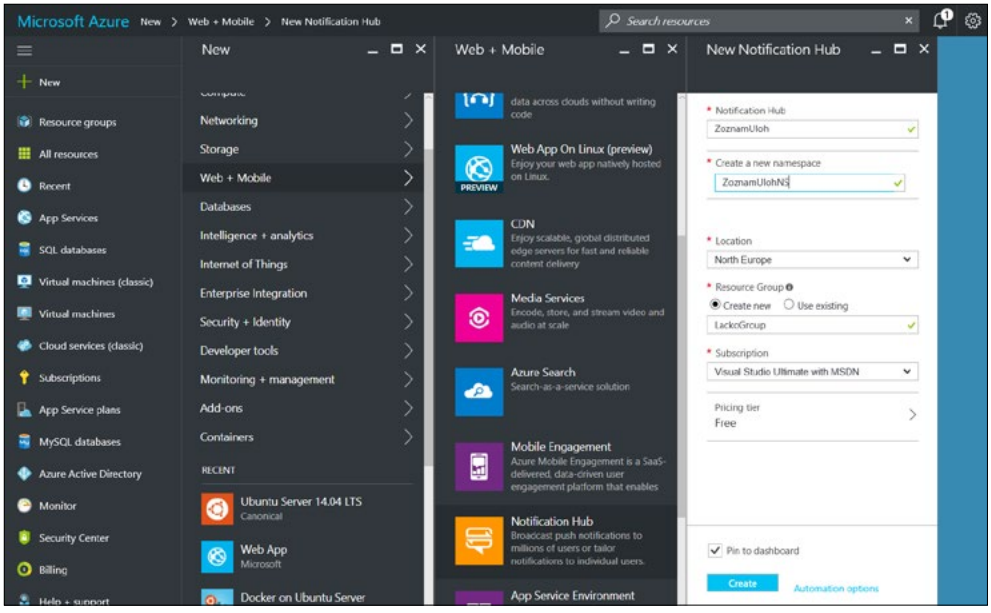
V této fázi se dějství znovu přenese do vašeho konta Microsoft Azure, které musíte mít v tomto okamžiku vytvořené a aktivované. Můžete použít:

- 30denní zkušební konto
- Konto, které jste získali jako předplatitel MSDN
- Konto studenta přes DreamsPark
- Komerční konto

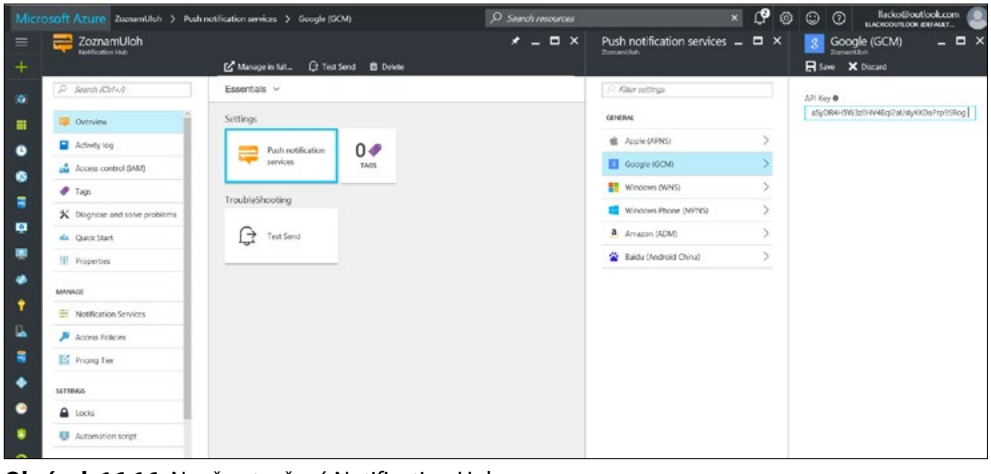
Vstupní bod do platformy představuje URL *www.windowsazure.com*. Můžete si vytvořit zkušební konto a během jednoho měsíce máte k dispozici na vyzkoušení libovolné služby v objemu 150 EUR.

Notification Hub budete vytvářet přes **Azure Portal**: *https://portal.sazure.com/*. Nový Notification Hub vytvoříte tlačítkem **+ New**, které najdete vlevo nahoře. Zvolte položku **Web + Mobile** → **Notification Hub** → **Quick Create**.

Zadejte vhodný název, jmenný prostor, vyberte umístění datového centra, ve kterém bude služba provozována, a skupinu prostředků a vše potvrďte klepnutím na tlačítko **Create.** Po vytvoření, které trvá několik sekund, najdete vámi vytvořený Notification Hub jako dlaždici na ploše portálu.



Obrázek 16.15: Vytvoření Notification Hub

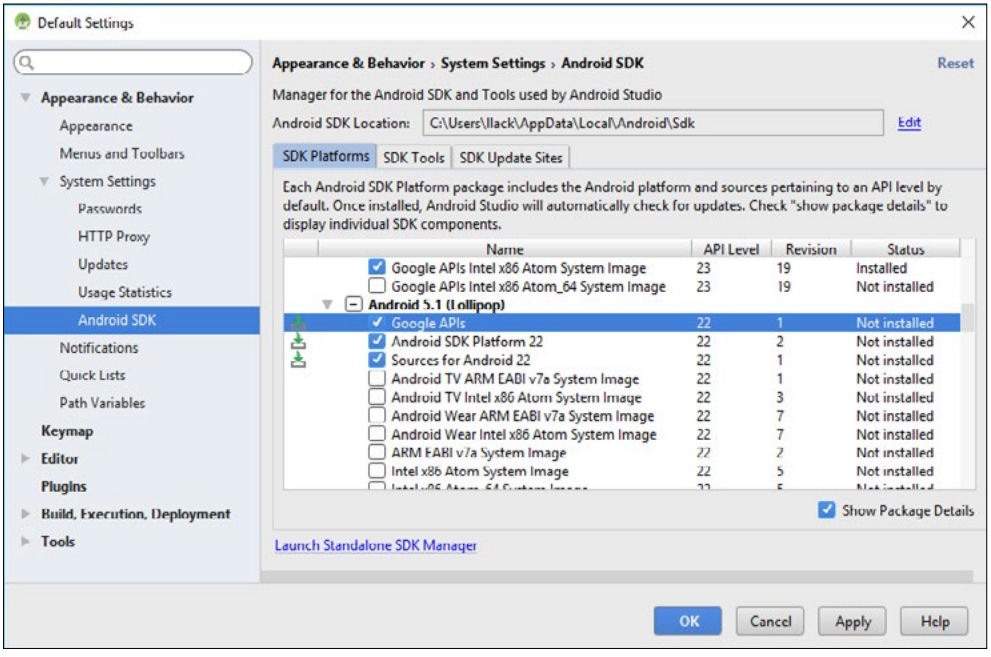


Obrázek 16.16: Nově vytvořený Notification Hub

Klepněte na tlačítko **Push Notifications Services** a v panelu, který se zobrazí vpravo, vyberte **Google (GCM)**. Zadejte Web API Key, který jste si zkopírovali z projektu Firebase z webové konzole Firebase. Zadaný údaj potvrďte tlačítkem **Save**.

Přidání služby Google Play do projektu aplikace pro Android

V Android Studiu zvolte položku nabídky **Tools** → **Android** → **SDK Manager**. V okně Android SDK Manager vyhledejte cílovou verzi SDK pro Android, kterou používá váš projekt (v našem případě 5.1), a otevřete ji klepnutím na **Show Package Details**. Pokud jste ještě nenainstalovali služby Google Play, klepněte na položku **Google Play Services**.



Obrázek 16.17: Instalace API pro Google Play Services

Otevřete soubor *build.gradle* v adresáři aplikace a do sekce `dependencies` přidejte řádky:

```
compile 'com.google.android.gms:play-services-gcm:10.0.1'
compile 'com.microsoft.azure:notification-hubs-android-sdk:0.4@aar'
compile 'com.microsoft.azure:azure-notifications-handler:1.0.1@aar'
```

Potom klepněte na ikonu **Sync Project with Gradle Files**.

Za sekci `dependencies` přidejte sekci:

```
repositories {
   maven {
    url "http://dl.bintray.com/microsoftazuremobile/SDK"
  }
}
```

Abyste implementovali podporu FCM do aplikace, potřebujete vytvořit službu na získání registračních tokenů pomocí rozhraní Firebase InstanceId API od Googlu. V tomto příkladu třídu pojmenujeme `MyInstanceIDService`. Do aplikačního manifestu v souboru *AndroidManifest.xml* přidejte do sekce `application` následující řádky:

```
<meta-data android:name="com.google.android.gms.version"
  android:value="@integer/google_play_services_version" />
<service android:name=".MyInstanceIDService">
  <intent-filter>
    <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
  </intent-filter>
</service>
```

Po získání registračního tokenu FCM z rozhraní Firebase InstanceId API ho použijeme na registraci do Azure Notification Hub. Registrace se realizuje pomocí `IntentService` s názvem `RegistrationIntentService`. Tato služba bude také zodpovědná za aktualizaci registračního tokenu FCM. Do aplikačního manifestu přidejte do sekce `<application>` následující:

```
<service
  android:name=".RegistrationIntentService"
  android:exported="false">
</service>
```

Také je v manifestu zapotřebí definovat receiver na příjem oznámení:

```
<service
  android:name=".RegistrationIntentService"
  android:exported="false">
</service>
```

Nakonec v manifestu přidejte potřebná oprávnění:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE" />
```

Kompletní kód v aplikačním manifestu v souboru *AndroidManifest.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.azurenotifikacia">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <meta-data android:name="com.google.android.gms.version"
      android:value="@integer/google_play_services_version" />

    <service android:name=".MyInstanceIDService">
      <intent-filter>
```

```
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
      </intent-filter>
    </service>

    <service
      android:name=".RegistrationIntentService"
      android:exported="false">
    </service>

    <receiver android:name="com.microsoft.windowsazure.notifications.
      NotificationsBroadcastReceiver"
      android:permission="com.google.android.c2dm.permission.SEND">
      <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE" />
        <category android:name="<your package name>" />
      </intent-filter>
    </receiver>

  </application>

  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
  <uses-permission
    android:name="com.google.android.c2dm.permission.RECEIVE" />

</manifest>
```

Aplikační logika v kódu v Javě

Ve složce kódu vytvořte novou třídu `NotificationSettings`. V kódu třídy vytvořte proměnné:

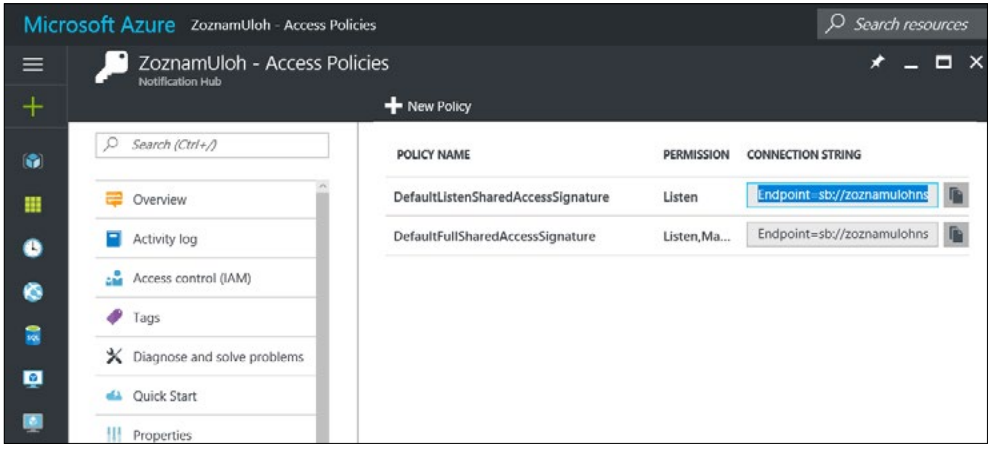
- `SenderId` – ID odesílatele, které jste získali v konzoli Firebase v nastavení projektu na kartě **Cloud Messaging**.
- `HubListenConnectionString` – připojovací řetězec `DefaultListenAccessSignature` pro rozbočovač. Tento připojovací řetězec můžete zkopírovat klepnutím na položku **Access Policies** v okně **Settings** hubu na Azure Portal.
- `HubName` – název notifikačního centra oznámení, který se zobrazí na Azure Portal.

```
public static String SenderId = "<project number>";
public static String HubName = "<HubName>";
public static String HubListenConnectionString =
  "<pripojovací reťazec DefaultListenSharedAccessSignature connection>";
```

Kód třídy s hodnotami pro náš příklad:

```
public class NotificationSettings {
  public static String SenderId = " 509757516665";
  public static String HubName = "ZoznamUloh";
  public static String HubListenConnectionString = "
    Endpoint=sb://zoznamulohns.servicebus.windows.net/;
  SharedAccessKeyName=
    DefaultListenSharedAccessSignature;
```

```
    SharedAccessKey=IneSB++…";
}
```



Obrázek 16.18: Připojovací řetězec DefaultListenAccessSignature na portálu Azure

Další třída, kterou je potřeba vytvořit, je `MyInstanceIDService`. Je to implementace služby procesu získání `InstanceID`.

```
import com.google.firebase.iid.FirebaseInstanceIdService;
import android.content.Intent;
import android.util.Log;

public class MyInstanceIDService extends FirebaseInstanceIdService {
  private static final String TAG = "MyInstanceIDService";

  @Override
  public void onTokenRefresh() {

    Log.d(TAG, "Obnova GCM registračného tokenu");

    Intent intent = new Intent(this, RegistrationIntentService.class);
    startService(intent);
  }
}
```

Další třídou, kterou přidáte do projektu, je `RegistrationIntentService`.

```
import android.app.IntentService;
import android.content.Intent;
import android.content.SharedPreferences;
import android.preference.PreferenceManager;
import android.util.Log;
import com.google.firebase.iid.FirebaseInstanceId;
import com.microsoft.windowsazure.messaging.NotificationHub;

public class RegistrationIntentService extends IntentService  {
  private static final String TAG = "RegIntentService";
```

```
private NotificationHub hub;

public RegistrationIntentService() {
  super(TAG);
}

@Override
protected void onHandleIntent(Intent intent) {

  SharedPreferences sharedPreferences =
    PreferenceManager.getDefaultSharedPreferences(this);
  String resultString = null;
  String regID = null;
  String storedToken = null;

  try {
    String FCM_token = FirebaseInstanceId.getInstance().getToken();
    Log.d(TAG, "FCM Registration Token: " + FCM_token);

    // či bol token poslaný na sever
    if (((regID=sharedPreferences.getString("registrationID", null)) ==
      null)){

      NotificationHub hub = new
        NotificationHub(NotificationSettings.HubName,
      NotificationSettings.HubListenConnectionString, this);
      Log.d(TAG, "Nová registrácia FCM tokenu : " + FCM_token);
      regID = hub.register(FCM_token).getRegistrationId();

      resultString = "Úspešne zaregistrovaný - RegId : " + regID;
      Log.d(TAG, resultString);

      sharedPreferences.edit().putString("registrationID", regID ).apply();
      sharedPreferences.edit().putString("FCMtoken", FCM_token ).apply();
    }

    // Je potrebný refresh tokenu?
    else if ((storedToken=sharedPreferences.getString("FCMtoken", "")) !=
      FCM_token) {

      NotificationHub hub = new
        NotificationHub(NotificationSettings.HubName,
        NotificationSettings.HubListenConnectionString, this);
      Log.d(TAG, "NH registrácia refresh s tokenom: " + FCM_token);
      regID = hub.register(FCM_token).getRegistrationId();

      resultString = "Úspešne zaregistrovanýy - RegId : " + regID;
      Log.d(TAG, resultString);

      sharedPreferences.edit().putString("registrationID", regID ).apply();
```

```
            sharedPreferences.edit().putString("FCMtoken", FCM_token ).apply();
          }

          else {
            resultString = "Predchádzajúca registrácia úspešná - RegId : " +
              regID;
          }
        } catch (Exception e) {
          Log.e(TAG, resultString="Problém s registráciou", e);
        }

        if (MainActivity.isVisible) {
          MainActivity.mainActivity.ToastNotify(resultString);
        }
      }
    }
```

Ve třídě `MainActivity` přidejte importy:

```
import com.google.android.gms.common.ConnectionResult;
import com.google.android.gms.common.GoogleApiAvailability;
import com.microsoft.windowsazure.notifications.NotificationsManager;
import android.content.Intent;
import android.util.Log;
```

Dále přidejte proměnné:

```
public static MainActivity mainActivity;
public static Boolean isVisible = false;
private static final String TAG = "MainActivity";
private static final int PLAY_SERVICES_RESOLUTION_REQUEST = 9000;
```

Přidejte metodu na zjišťování dostupnosti služeb Google Play.

```
private boolean checkPlayServices() {
  GoogleApiAvailability apiAvailability =
    GoogleApiAvailability.getInstance();
  int resultCode = apiAvailability.isGooglePlayServicesAvailable(this);
  if (resultCode != ConnectionResult.SUCCESS) {
    if (apiAvailability.isUserResolvableError(resultCode)) {
      apiAvailability.getErrorDialog(this, resultCode,
        PLAY_SERVICES_RESOLUTION_REQUEST)
      .show();
    } else {
      Log.i(TAG, "Zariadenie je podporované Google Play Services.");
      Toast.makeText(this, "Zariadenie nie je podporované Google Play
        Services.", Toast.LENGTH_LONG).show();
      ToastNotify("Zariadenie nie je podporované Google Play Services.");
      finish();
    }
    return false;
  }
  return true;
}
```

Všimněte si, že kromě klasické toastové notifikace vypíšeme oznámení i pomocí threadově bezpečné metody `ToastNotify`, která vypíše trvalé oznámení do okna aplikace. Prvku `TextView`, ve kterém se po vytvoření projektu vypíše řetězec `Hello World`, jsme přiřadili identifikátor `tw`:

```
public void ToastNotify(final String notificationMessage) {
  runOnUiThread(new Runnable() {
    @Override
    public void run() {
      Toast.makeText(MainActivity.this, notificationMessage,
        Toast.LENGTH_LONG).show();
      TextView helloText = (TextView) findViewById(R.id.tw);
      helloText.setText(notificationMessage);
    }
  });
}
```

Přidejte metodu na registraci Notification Hub.

```
public void registerWithNotificationHubs()
{
  if (checkPlayServices()) {
    // Start IntentService na zaregistrovanie aplikácie do FCM.
    Intent intent = new Intent(this, RegistrationIntentService.class);
    startService(intent);
  }
}
```

Dále přidejte metody na ověření stavu aplikace.

```
@Override
  protected void onStart() {
    super.onStart();
    isVisible = true;
  }

@Override
  protected void onPause() {
    super.onPause();
    isVisible = false;
  }

@Override
  protected void onResume() {
    super.onResume();
    isVisible = true;
  }

@Override
  protected void onStop() {
    super.onStop();
    isVisible = false;
  }
```

Do metody Create přidejte kód na spuštění procesu registrace při vytvoření aktivity:

```
@Override
  protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);

  mainActivity = this;
  NotificationsManager.handleNotifications(this,
    NotificationSettings.SenderId, MyHandler.class);
  registerWithNotificationHubs();
}
```

Nakonec vytvoříme třídu MyHandler na příjem notifikací. Je to podtřída třídy com.microsoft.windowsazure.notifications.NotificationsHandler. Její kód přepíše metodu OnReceive, aby obslužná rutina nahlásila přijaté notifikace. Obslužná rutina také posílá oznámení správci oznámení Android pomocí metody sendNotification(). Metoda sendNotification() musí být spuštěná, když není spuštěná aplikace.

```
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.media.RingtoneManager;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.NotificationCompat;
import com.microsoft.windowsazure.notifications.NotificationsHandler;

public class MyHandler extends NotificationsHandler  {
  public static final int NOTIFICATION_ID = 1;
  private NotificationManager mNotificationManager;
  NotificationCompat.Builder builder;
  Context ctx;

  @Override
  public void onReceive(Context context, Bundle bundle) {
    ctx = context;
    String nhMessage = bundle.getString("message");
    sendNotification(nhMessage);
    if (MainActivity.isVisible) {
      MainActivity.mainActivity.ToastNotify(nhMessage);
    }
  }

  private void sendNotification(String msg) {

    Intent intent = new Intent(ctx, MainActivity.class);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

    mNotificationManager = (NotificationManager)
      ctx.getSystemService(Context.NOTIFICATION_SERVICE);
```

```
        PendingIntent contentIntent = PendingIntent.getActivity(ctx, 0,
          intent, PendingIntent.FLAG_ONE_SHOT);

        Uri defaultSoundUri =
          RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
        NotificationCompat.Builder mBuilder =
          new NotificationCompat.Builder(ctx)
          .setSmallIcon(R.mipmap.ic_launcher)
          .setContentTitle("Notification Hub Demo")
          .setStyle(new NotificationCompat.BigTextStyle()
          .bigText(msg))
          .setSound(defaultSoundUri)
          .setContentText(msg);

        mBuilder.setContentIntent(contentIntent);
        mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
      }
    }
```

Nyní můžete aplikaci spustit a ověřit, zda se úspěšně zaregistrovala v centru oznámení. Aplikaci doporučujeme spustit na reálném zařízení. Pokud spouštíte aplikaci na emulátoru, musíte přidat po jeho spuštění přístupové parametry vašeho účtu Google. U reálného zařízení předpokládáme, že máte přístupové parametry nastavené.

POZNÁMKA

Aplikace sama o sobě nedělá nic užitečného, jelikož očekává oznámení od serveru, které je potřeba poslat.

Zprávy můžete posílat z portálu Azure pomocí tlačítka **Test Send**.

Připojení aplikace pro Android k mobilní službě

V úvodu jsme slíbili, že ukážeme, jak může aplikace ukládat údaje v cloudové službě. Takto je možné například implementovat už zmíněný žebříček top skóre nejúspěšnějších hráčů. Zatím jsme ukázali jen posílání notifikace, takže už byste uměli hráči poslat oznámení, že dosáhl absolutně nejvyššího skóre v dané hře. Ještě ale nemáme implementované ukládání údajů na straně serveru.

Abychom nestřídali platformy, ukážeme připojení aplikace pro Android k Microsoft Azure Mobile Apps. V reálné aplikaci můžete využít tuto nebo jinou komerční službu nebo si můžete napsat serverovou aplikaci sami.

POZNÁMKA

Výhodou Azure Mobile Apps je, že hned po vytvoření mobilní služby v cloudu si můžete stáhnout hotový projekt ukázkové aplikace pro všechny nejpoužívanější mobilní platformy včetně Androidu.

Microsoft Azure Mobile Apps

Aplikační služby se využívají pro specifický okruh často se opakujících úloh v mobilních aplikacích, typicky k ukládání dat, autentizaci, posílání notifikací a podobně. Použitím aplikační služby si ulehčíte a zrychlíte vývoj, jelikož funkcionalitu, kterou přesunete na mobilní službu, byste museli pracně vytvářet a provozovat ji na vlastním nebo pronajatém serverovém prostředí.

Aplikační službu vytvoříte rychle, doslova na několik klepnutí. Aplikační služby mohou využívat aplikace pro nejpoužívanější tabletové a mobilní platformy Windows, iOS a Android. Provoz aplikační služby je levný na bázi měsíčních poplatků. Nároky na správu jsou prakticky nulové, jelikož je provozována v cloudovém prostředí platformy Microsoft Azure, takže neřešíte provozní aspekty, jako je zálohování, monitorování, obnova a podobně.

Typickými scénáři pro využití cloudových aplikační služeb jsou:

- Ukládání údajů v cloudu
- Autentizace uživatelů
- Notifikace

Služba Mobile Apps, která je součástí Azure App Service, nabízí vysoce škálovatelnou a globálně dostupnou platformu pro vývoj mobilních aplikací určenou pro vývojáře a integrátory systémů. Přináší bohatou sadu funkcí pro vývojáře aplikací pro mobilní zařízení. S Mobile Apps můžete:

- **Sestavovat nativní a multiplatformní aplikace** – ať už vytváříte nativní aplikace pro iOS, Android a Windows, nebo multiplatformní aplikace pomocí technologií Xamarin či Cordova (PhoneGap), můžete využívat výhody App Service a nativních sad SDK.
- **Nastavit připojení k firemním systémům** – s Mobile Apps můžete jednoduše a rychle přidat firemní přihlašování a připojovat se k lokálním nebo cloudovým firemním zdrojům.
- **Vytvářet aplikace připravené pro offline režim se synchronizací dat** – produktivitu mobilních pracovníků zajistíte sestavením aplikací, které pracují v režimu offline a využívají Mobile Apps na synchronizaci údajů na pozadí, když se obnoví připojení, se zdroji firemních dat nebo s rozhraními API SaaS.
- **Odesílat oznámení milionům příjemců** – uživatele aplikace oslovíte pomocí rychlých oznámení, které můžete odesílat na jakékoliv zařízení, personalizovat je podle jejich potřeb a zasílat v nejvhodnější čas.

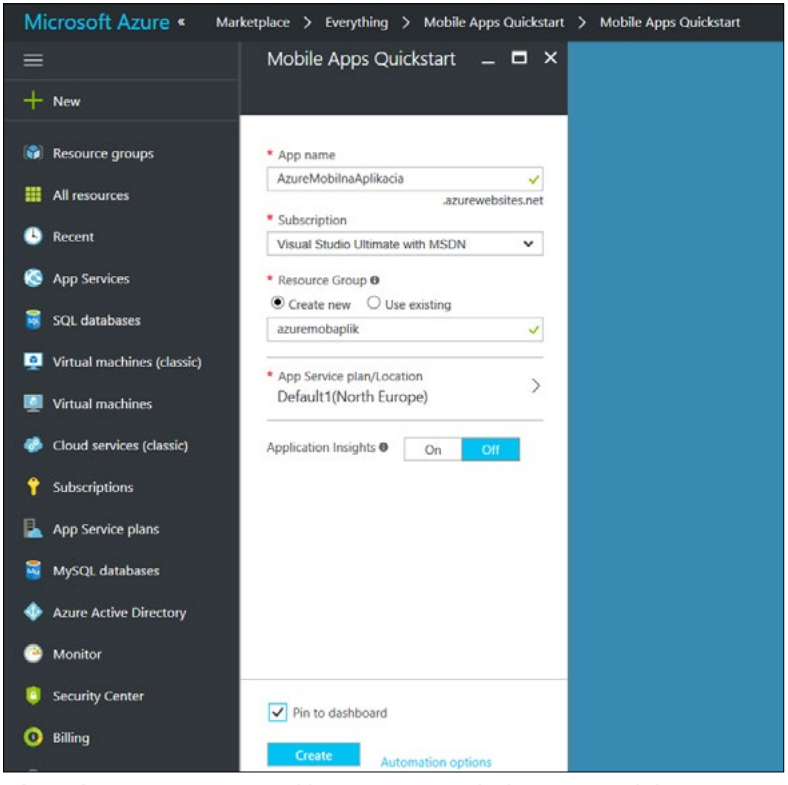
Pro vývoj mobilních řešení využívajících cloud jsou důležité následující funkce:

- **Ověřování a autorizace** – můžete si vybrat ze stále se rozšiřujícího seznamu zprostředkovatelů identity, mezi které patří nejen Azure Active Directory pro podnikové ověřování, ale i služby sociálních sítí, jako je Facebook, Google, Twitter a účet Microsoft. Služba Azure Mobile Apps poskytuje službu OAuth 2.0 pro každého zprostředkovatele.

- **Přístup k údajům** – služba Azure Mobile Apps poskytuje zdroj dat Odate verze 3, který funguje s mobilními zařízeními a je propojený se službou SQL Azure nebo místním systémem SQL Server. Tato služba může být založena na Entity Framework, což vám umožní jednoduchou integraci s jinými zprostředkovateli dat NoSQL i SQL, včetně Azure Table Storage, MongoDB, DocumentDB a rozhraní API pro SaaS, jako je Office 365 a Salesforce.com.
- **Synchronizace offline dat** – klientské sady SDK ulehčují vytváření robustních a rychle reagujících mobilních aplikací, které budou pracovat s datovou množinou offline. Tu potom lze automaticky synchronizovat s backendovými daty, a to i s podporou řešení konfliktů.
- **Oznámení** (push Notification) – klientské sady SDK se snadno integrují s registračními funkcemi Azure Notification Hubs, což vám umožní současně zasílat notifikace milionům uživatelů.

Vytvoření aplikace využívající službu Azure Mobile Apps

Námětem ukázkové aplikace bude seznam úkolů, které přes mobilní službu můžete sdílet mezi více zařízeními jednoho uživatele, a to i napříč platformami. K vytvoření Azure backendu pro aplikaci pro Android použijte webový nástroj Azure Portal na adrese *https://portal.azure.com*.



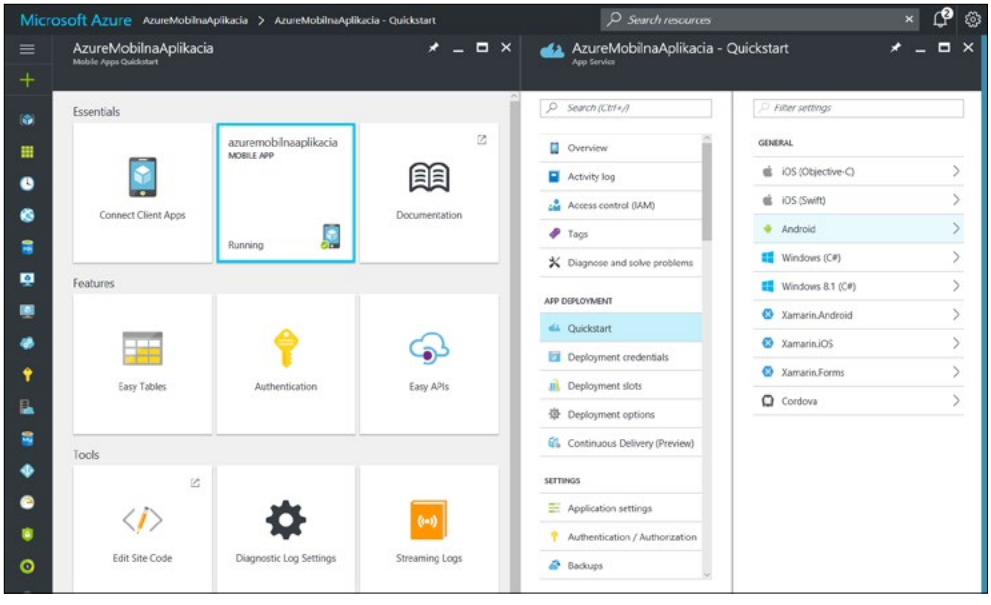
Obrázek 16.19: Vytvoření aplikace využívající službu Azure Mobile Apps pomocí šablony Mobile Apps Quickstart

Nový backend pro mobilní aplikaci Hub vytvoříte tlačítkem **+ New**, které najdete vlevo nahoře. Aktivujte vyhledávání a vyhledejte položku **Mobile Apps Quickstart**. Při vytváření nezapomeňte označit pole **Pin to dashboard**, aby se podpůrná služba pro mobilní aplikaci zobrazila jako dlaždice na ploše portálu Azure.

Klepněte na tlačítko **Create**. Vytvoření a nasazení služby trvá několik minut.

Konfigurace služby

Na portálu Azure aktivujte dlaždici vytvořené služby a klepněte na **Settings** → **Quick Start** → **Android**.



Obrázek 16.20: Výběr platformy

Většina mobilních služeb se neobejde bez ukládání údajů, proto bude prvním krokem vytvoření databáze. Pokud vám stačí 20 MB, můžete si vytvořit bezplatnou 20MB databázi SQL. V dialogovém okně je i výběr regionu globálního datového centra, kde bude umístěna vaše mobilní služba. S výjimkou specifických odůvodněných případů je potřeba mít ve stejném centru umístěnu i databázi, kterou bude tato služba využívat.

Vytvoření databáze v jiné oblasti se nedoporučuje, protože takováto konfigurace bude vyžadovat náklady na rozšíření šířky pásma a samozřejmě přístup služby k databázi bude mít vyšší latenci. Tuto bezplatnou databázi můžete použít i na mobilní služby, případně na jiné účely. Pokud už máte v rámci Azure ve stejném datovém centru vytvořenou jinou databázi, můžete ji samozřejmě použít i na právě vytvářenou mobilní službu.

V oblasti **Create a table API** vyberte jako jazyk backendu možnost **Node.js**. Potvrďte a klepněte na **Create TodoItem table**. Tím se ve vaší databázi vytvoří nová tabulka `TodoItem`. Následně stáhněte projekt aplikace pro Android. Je to funkční projekt Android pro aplikaci přednastavenou na připojení k vašemu prostředí backend. Otevřete projekt v Android Studiu pomocí možnosti

Import Project. Nezapomeňte si vybrat tuto možnost importu, abyste se vyhnuli případným chybám JDK. Stisknutím tlačítka Spustiť aplikáciu sestavíte projekt a spustíte aplikaci v simulátoru Androidu. Možná budete muset doinstalovat správnou verzi Android SDK Build-Tools, v našem případě verzi 23.

V aplikaci zadejte smysluplný text, například „Nakŕmiť rybičky", a klepněte na tlačítko **Add**. Tím odešlete do prostředí backend v Azure, které jste předtím vytvořili, požadavek POST. Prostředí backend vloží data z požadavku do tabulky SQL `TodoItem` a vrátí informace o nově uložených položkách do mobilní aplikace. Mobilní aplikace zobrazí tato data v seznamu.

Prohlédněte si dobře komentovaný kód hlavní aktivity v souboru *ToDoActivity.java*. V aplikaci je využitý i prvek `ProgressBar` na indikaci probíhajících operací, jelikož interakce s cloudovou službou nějaký čas trvá. Uvádíme jen deklaraci proměnných, kód metody `onCreate()` a kód metody na přidání nové položky do seznamu.

```
private MobileServiceClient mClient;
private MobileServiceTable<ToDoItem> mToDoTable;
private ToDoItemAdapter mAdapter;
private EditText mTextNewToDo;
private ProgressBar mProgressBar;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_to_do);

  mProgressBar = (ProgressBar) findViewById(R.id.loadingProgressBar);

  // Initialize the progress bar
  mProgressBar.setVisibility(ProgressBar.GONE);

  try {
    // Create the Mobile Service Client instance, using the provided

    // Mobile Service URL and key
    mClient = new MobileServiceClient(
      "https://azuremobilnaaplikacia.azurewebsites.net",
      this).withFilter(new ProgressFilter());

    // Extend timeout from default of 10s to 20s
    mClient.setAndroidHttpClientFactory(new OkHttpClientFactory() {
      @Override
      public OkHttpClient createOkHttpClient() {
        OkHttpClient client = new OkHttpClient();
        client.setReadTimeout(20, TimeUnit.SECONDS);
        client.setWriteTimeout(20, TimeUnit.SECONDS);
        return client;
      }
    });

    // Get the Mobile Service Table instance to use
```

```
        mToDoTable = mClient.getTable(ToDoItem.class);

        // Offline Sync
        //mToDoTable = mClient.getSyncTable("ToDoItem", ToDoItem.class);

        //Init local storage
        initLocalStore().get();

        mTextNewToDo = (EditText) findViewById(R.id.textNewToDo);

        // Create an adapter to bind the items with the view
        mAdapter = new ToDoItemAdapter(this, R.layout.row_list_to_do);
        ListView listViewToDo = (ListView) findViewById(R.id.listViewToDo);
        listViewToDo.setAdapter(mAdapter);

        // Load the items from the Mobile Service
        refreshItemsFromTable();

      } catch (MalformedURLException e) {
        createAndShowDialog(new Exception("There was an error creating the
          Mobile Service. Verify the URL"), "Error");
      } catch (Exception e){
        createAndShowDialog(e, "Error");
      }
    }

    public void addItem(View view) {
      if (mClient == null) {
        return;
      }

      // Create a new item
      final ToDoItem item = new ToDoItem();

      item.setText(mTextNewToDo.getText().toString());
      item.setComplete(false);

      // Insert the new item
      AsyncTask<Void, Void, Void> task = new AsyncTask<Void, Void, Void>(){
        @Override
        protected Void doInBackground(Void... params) {
          try {
            final ToDoItem entity = addItemInTable(item);

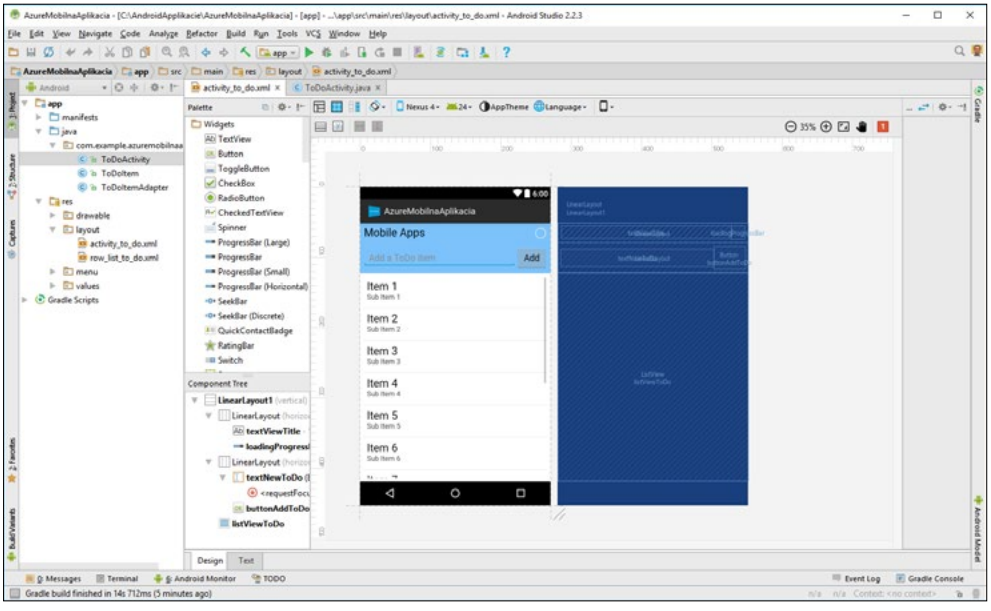
            runOnUiThread(new Runnable() {
              @Override
              public void run() {
                if(!entity.isComplete()){
                  mAdapter.add(entity);
                }
              }
```

```
            });
        } catch (final Exception e) {
            createAndShowDialogFromTask(e, "Error");
        }
        return null;
      }
    };

    runAsyncTask(task);

    mTextNewToDo.setText("");
  }
```

Ve vývojovém prostředí můžete případně projekt přizpůsobit svým požadavkům.



Obrázek 16.21: Projekt aplikace pro Android ve vývojovém prostředí Android Studio

KAPITOLA

17

Služby a broadcasty

Témata kapitoly:

- **Služba a její životní cyklus**

V dosud prezentovaných tématech jsme se zaměřovali většinou na činnosti, jež uživatelé vidí. Avšak v mobilních zařízeních často potřebujeme i kód běžící na pozadí, a to buď trvale, nebo delší dobu. A to je úlohou služeb.

Na rozdíl od broadcast receiverů, které jsou určeny k rychlému provedení nějaké akce, po níž skončí, jsou služby navrženy tak, aby mohly plnit úlohy, jež potřebují na dokončení dlouhý čas (například stahování souboru prostřednictvím připojení k internetu nebo streamování hudby k uživateli), ale nevyžadují uživatelské rozhraní.

POZNÁMKA

Služby (Services) se používají na pravidelné a nepřetržitě prováděné akce, které nevyžadují uživatelské rozhraní.

Typickým příkladem je stahování souborů, synchronizace informací v zařízení s cloudovým serverem, sledování komunikace či přehrávání hudby na pozadí. Služby se spouštějí, zastavují a kontrolují z jiných komponent aplikace, například z aktivit, broadcast receiverů či jiných služeb.

Služby mají dvě hlavní výhody:

- Umožňují provádět činnosti na pozadí i v případě, že systém aplikaci služby ukončí.
- Umožňují, aby kód služby v jednom procesu mohl komunikovat s kódem v jiném procesu.

Výhodou služeb je i to, že jsou méně náchylné k násilnému ukončení systémem (například při nedostatku paměti) než aktivity na pozadí. Při případném ukončení je systém po získání potřebných prostředků restartuje.

POZNÁMKA

Na rozdíl od aktivity má služba vždy maximálně jednu instanci.

Přestože služba běží na pozadí, neběží ve svém vlákně, jak tomu je typicky u jiných operačních systémů. U platformy Android je služba spuštěna v hlavním aplikačním vlákně, její blokování má tak stejné následky jako blokování aktivity, tedy špatnou odezvu uživatelského rozhraní. Pro dlouhotrvající akce je potřeba vytvořit samostatné vlákno, ve kterém může služba realizovat sovou práci.

Služba zůstává spuštěna, dokud není explicitně zastavena z jiné komponenty pomocí metody `stopService()`, dokud sama služba nezavolá metodu `stopSelf()` nebo není ukončena systémem. Po případném opětovném zavolání metody `startService()` se jen opětovně zavolá metoda `onStartCommand()` v už běžící službě.

Služba a její životní cyklus

Služba je část aplikace, která běží na pozadí, a nemá uživatelské rozhraní. Různé části aplikace mohou spouštět služby, které budou nadále běžet na pozadí.

UPOZORNĚNÍ

Podobně jako aktivity je potřeba i služby definovat v souboru *AndroidManifest.xml*.

Služby mají dva základní stavy, přičemž služba se může nacházet i v obou zároveň:

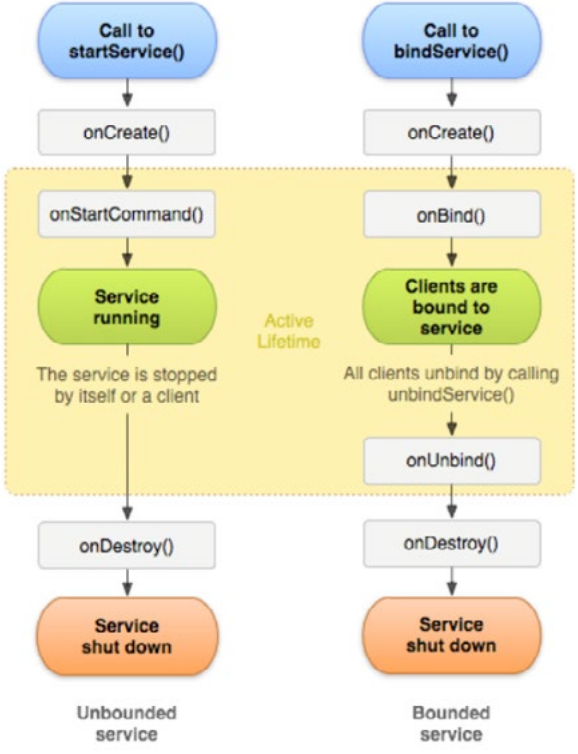
- **Spuštěná služba (started)** – služba běží na pozadí, dokonce i tehdy, když aktivita, která službu pustila, je už zastavená. Služba by se měla automaticky ukončit tehdy, když provede vše, co bylo zapotřebí, nebo ji násilně ukončí operační systém, aby se uvolnily systémové prostředky. Znovu zdůrazňujeme, že pokud má operační systém nedostatek prostředků, ukončuje nejprve aplikace na pozadí a až potom služby, takže ukončení služby operačním systémem je ojedinělé. Spuštěné služby se aktivují přes volání metody `StartService()`. Když spuštěná služba dokončí svoje úlohy, měla by se ukončit přes volání `stopSelf()`. Alternativně může být běžící služba zastavena jiným komponentem přes volání metody `stopService()`.
- **Vázaná služba (bound)** – služba poskytuje rozhraní klient-server, přes které mohou součásti spolupracovat se službou, posílat požadavky, získávat výsledky a komunikovat. Vázaná služba běží pouze tehdy, pokud je k ní připojena součást, přičemž k jedné službě může být připojeno více součástí. Když se odpojí i poslední, služba se ukončí.

Aktivita například spustí službu přehrávání zvuku. Aktivita, která službu spustila, má uživatelské rozhraní poskytující ovládací prvky pro uživatele za účelem pozastavení přehrávání nebo přeskočení na další stopu. Stejně tak bude služba pravděpodobně potřebovat oznamovat informace volající aktivitě, například že přehrávání aktuální zvukové stopy bylo dokončeno a služba potřebuje poskytnout informace o další stopě, kterou má začít přehrávat.

Komponent se na vázanou službu připojí prostřednictvím volání metody `bindService()`. Pokud vazba na službu už není zapotřebí, komponent by měl volat metodu `unbindService()`. K jedné službě může být připojeno více komponentů. Když se poslední vázaný komponent odpojí, službu ukončí operační systém Android. Vázanou službu lze spustit i prostřednictvím metody `StartService()`. Po spuštění komponentů, například aktivit, se tyto potom mohou vázat na spuštěnou službu přes volání metody `bindService()`. Když je vázaná služba spuštěna pomocí volání

`StartService()`, bude pokračovat v činnosti i po odpojení posledního komponentu. Vázaná služba zahrnuje implementaci metody `onBind()`.

Metody navázané na stavy životního cyklu služby názorně vysvětluje obrázek.



Obrázek 17.1: Životní cyklus služby

`onCreate()` – metoda se volá při prvním vytvoření služby. Když už služba běží, volání metody se nerealizuje.

`onBind()` – metoda se volá v případě, že některá aktivita nebo některý komponent požaduje spuštění služby voláním metody `startService()`.

`onUnbind()` – metoda se volá, když se odpojí všechny aktivity a komponenty, které se službou komunikovaly.

`onDestroy()` – metoda se volá při zrušení služby. Služba by prostřednictvím této metody měla uvolnit všechny používané prostředky.

`onStartCommand()` – metoda vrací konstanty, které určují, jak se bude služba chovat, když ji systém ukončí kvůli nedostatku prostředků:

- `START_STICKY` – služba se vždy restartuje, ale záměr v `onStartCommand()` bude mít hodnotu `NULL`. Toto chování je typické pro služby, které jsou explicitně spouštěny a zastavovány pomocí `startService()` a `stopService()`.

- START_NOT_STICKY – služba se restartuje, pouze pokud čekají na vyřízení další požadavky. Jinak zůstane služba zastavená. Toto chování je typické pro služby, které se spouštějí periodicky a po skončení požadovaného úkonu se samy ukončí pomocí metody stopSelf().
- START_REDELIVER_INTENT – služba se restartuje, pouze pokud čekají na vyřízení další požadavky nebo pokud služba před ukončením nedokončila svou práci. Po restartu je do metody onStartCommand() předán původní záměr. Toto chování je typické pro služby, u kterých je potřeba zabezpečit provedení operace.

Ke zjištění, jak se služba spustila, slouží atribut flags, který může nabývat tyto hodnoty:

- START_FLAG_REDELIVERY – indikuje, že byl znovu doručen předchozí záměr, ale služba byla ukončena před zavoláním metody stopSelf().
- START_FLAG_RETRY – indikuje, že služba byla restartována systémem kvůli nedostatku prostředků a měla nastavený parametr START_STICKY.

Ke spuštění služby z aktivity je určena metoda startService(), která jako parametr dostane záměry. Volání je asynchronní. Nejprve se volá metoda OnCreate(). Následně je záměr odevzdán metodě onStartCommand(). Po spuštění služba běží, dokud není explicitně zastavena.

Ke zpřístupnění služby API ostatním komponentám (klientům) je určena metoda bindService(). I volání této metody je asynchronní:

```
Context.bindService (Intent service, ServiceConnection conn, int flags)
```

Vazba na služby umožňuje komponentám posílat požadavky a přijímat odpovědi ze služby běžící ve stejném nebo jiném procesu. Pokud není služba spuštěná, spustí se a bude pokračovat v běhu, dokud je na ni připojen aspoň jeden klient. Služba vrací tzv. IBinder, který reprezentuje určitý objekt ze služby. V klientovi je potřeba vytvořit ServiceConnection. Pokud je jako parametr odevzdaná hodnota BIND_AUTO_CREATE, vytvoří se nová služba, v opačném případě je vrácena hodnota false.

Služba musí být deklarována v aplikačním manifestu v sekci application:

```
<application>
  <service android:name="MojaSluzba> </service>
</application>
```

Příklad – Přehrávání hudby na pozadí

Námětem příkladu je vytvoření aplikace, která bude přehrávat zvukový soubor ve formátu MP3. Soubor bude de facto přehrávat služba spuštěná aplikací, takže hudba bude hrát dále, i když uživatel činnost aplikace na popředí, která službu spustila, pozastaví nebo ukončí.

Pro jednoduchost bude hudba součástí projektu aplikace. Po vytvoření projektu vytvořte ve složce *res* vnořenou složku *raw* a do ní umístěte soubor MP3, který bude služba přehrávat. Stačí nějaké delší zvonění stažené například z *http://ringtones.mob.org*.

Uživatelské rozhraní layoutu hlavní aktivity obsahuje jen dvě tlačítka: **START** a **STOP** – na spuštění a zastavení služby.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="sk.pcrevue.llacko.sluzbahudba.MainActivity">

    <Button
      android:id="@+id/start_button"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="START">
    </Button>

    <Button
      android:id="@+id/stop_button"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="STOP">
    </Button>

</LinearLayout>
```

Službu je potřeba deklarovat v *AndroidManifest.xml*.

```
<service android:name=".PrehravacService " />
```

Kód hlavní aktivity v metodě `onCreate()` vytváří záměr na službu `PrehravacService`. V obsluze události klepnutí na příslušné tlačítko aplikace spouští, respektive zastavuje službu.

Přerušení přehrávání hudby na pozadí, zatímco uživatel provádí něco jiného, například kontroluje zprávy elektronické pošty, je velmi rušivé. Proto je realizováno jako služba Foreground. V principu je to obyčejná služba, ale s vysokou prioritou, takovou, která odpovídá zobrazené aktivitě. Službu Foreground spustíte voláním metody `startForeground` v metodě `OnCreate`.

UPOZORNĚNÍ

Nutná podmínka pro službu Foreground je vytvoření notifikace do panelu oznámení, jelikož uživatel musí být informován, že tato služba je spuštěna a nebude vypnuta, dokud tak neučiní ručně.


```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    final Intent hudbaIntent = new Intent(getApplicationContext(),
      PrehravacService.class);

    final Button startButton = (Button) findViewById(R.id.start_button);
    startButton.setOnClickListener(new View.OnClickListener()  {
      public void onClick(View src) {
```

```
        startService(hudbaIntent);
      }
    });

    final Button stopButton = (Button) findViewById(R.id.stop_button);
    stopButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View src) {
        stopService(hudbaIntent);
      }
    });
  }
}
```

Kód služby v souboru *PrehravacService.java* přehraje soubor MP3. Objekt `MediaPlayer` je nastaven tak, aby skladbu přehrál pouze jednou, a po přehrání skladby zavolá metodu `onCompletion()`, ve které služba zastaví sama sebe. Po spuštění služba pošle notifikaci notifikačnímu centru, odkud i po ukončení aplikace spouštějící službu lze tuto aplikaci znovu spustit a přehrávání ukončit. Vytvořte třídu:

```
import android.app.Notification;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;

public class PrehravacService extends Service
{
  private static final int ID = 1;
  private MediaPlayer mediaPlayer;
  private int startID;

  @Override
  public void onCreate()  {
    super.onCreate();

    mediaPlayer = MediaPlayer.create(this, R.raw.hudba);
    if (mediaPlayer != null)  {
      mediaPlayer.setLooping(false);
      mediaPlayer.setOnCompletionListener(new
        MediaPlayer.OnCompletionListener() {
        @Override
        public void onCompletion(MediaPlayer mp) {
          stopSelf(startID);
        }
      });
    }

    //Notifikace
    final Intent notifIntent = new Intent(getApplicationContext(),
      MainActivity.class);
    final PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
```

```
        notifIntent, 0);

      final Notification notification = new Notification.Builder(
        getApplicationContext())
        .setSmallIcon(android.R.drawable.ic_media_play)
        .setOngoing(true).setContentTitle("Prehrávání hudby")
        .setContentText("Klikni na spuštení aplikace Hudba1")
        .setContentIntent(pendingIntent).build();
      startForeground(ID, notification);
    }


    @Override
    public int onStartCommand(Intent intent, int flags, int startid) {
      if (mediaPlayer != null) {
        startID = startid;
        if (mediaPlayer.isPlaying()) mediaPlayer.seekTo(0);
        else mediaPlayer.start();
      }
      return START_NOT_STICKY;
    }

    @Override
    public void onDestroy() {
      if (mediaPlayer != null)   {
        mediaPlayer.stop();
        mediaPlayer.release();
      }
    }

    @Override
    public IBinder onBind(Intent intent) {return null;}

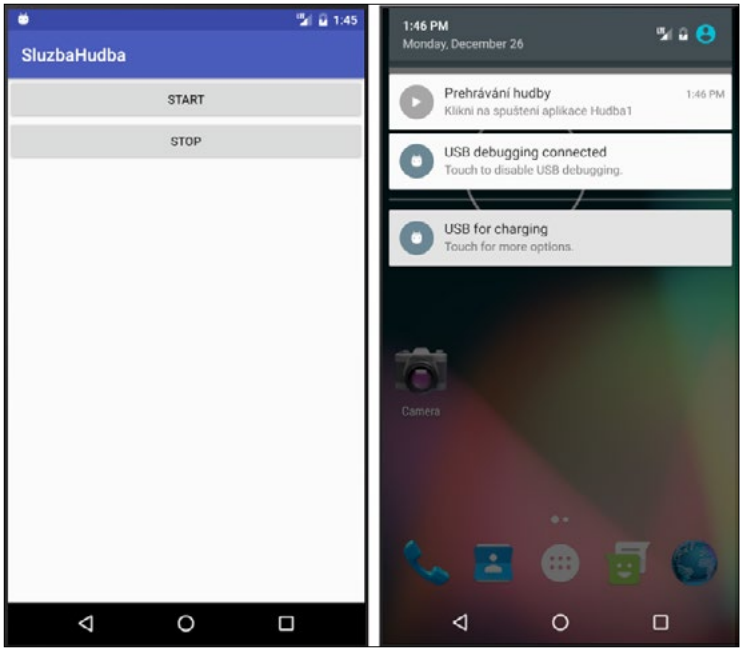
}
```

Ikona ve stavovém řádku oznámení signalizuje, že služba přehrává hudbu i po ukončení aplikace, která službu spustila. Klepnutím na položku v notifikačním centru může uživatel aplikaci znovu spustit.

Pomocí metody `startForeground` deklarujeme spuštění služby v popředí, takže je méně pravděpodobné, že bude ukončena, když bude systém potřebovat více zdrojů.

V metodě `onStartCommand` se kontroluje, zda objekt `mediaPlayer` existuje a zda něco přehrává. Pokud přehrává, potom se provede posun na začátek skladby, pokud ne, spustí se přehrávání. Hodnota `START_NOT_STICKY` znamená, že systém by neměl automaticky restartovat službu, když byla ukončena.

Některé služby pracují nezávisle na jiných komponentech. Jsou spuštěné, provedou svou práci a zastaví se. Jiné služby naproti tomu přijímají žádosti a poskytují odpovědi pro další komponenty, přičemž tyto nemusí být z aplikace, která službu spustila.



Obrázek 17.2: Po ukončení aplikace, která službu spustila, se k ní můžete vrátit přes notifikační centrum (vpravo)

Tyto komponenty budou muset být napojené na službu. Musí otevřít a udržovat připojení ke službě, aby s ní mohly komunikovat. Pokud komponenty a služby běží v různých v procesech, potom existují dvě možnosti, jak to provést:

- Použít třídu `Messenger`
- Definovat vzdálené rozhraní ke službě pomocí interface AIDL (Android Interface Definition Language)

Pomocí třídy Messenger posíláme zprávy do handleru aktivity z jedné komponenty do druhé přes hranice procesu. Po odeslání se zprávy ve frontě postupně zpracovávají přijímacími komponentami. Tento přístup byste měli používat, pokud vaše služba může tolerovat sekvenční přístupy.

Na straně služby je potřeba vytvořit obslužnou rutinu na příjem a zpracování konkrétní zprávy. Služba bude muset vytvořit i handler poskytující možnost spojení do libovolného klienta, který se váže k této službě.

KAPITOLA

18

Poskytování obsahu

Témata kapitoly:

- Třída ContentProvider
- Ukládání kontaktů v zařízení s Androidem
- Příklad – Přístup ke kontaktům v zařízení
- Příklad – Přístup ke kontaktům, načítání údajů na pozadí
- Příklad – Aktualizace údajů
- Příklad – Vytvoření aplikace, která bude poskytovat údaje
- Příklad – Vytvoření aplikace, která bude poskytovat údaje z databáze

Mnohé aplikace si údaje, se kterými pracují, ukládají do lokální databáze SQLite. Tato data jsou dostupná jen pro příslušnou aplikaci. Avšak některé údaje je užitečné poskytnout i jiným aplikacím, samozřejmě jen těm, které k tomu mají oprávnění, a se souhlasem uživatele. Typickým příkladem jsou kontakty. Kromě aplikace Kontakty, která je spravuje, je mohou potřebovat e-mailoví klienti třetích stran, aplikace sociálních sítí a podobně. Proto je zapotřebí mechanizmus na poskytnutí údajů do jiných aplikací běžících na stejném zařízení. Sdílení údajů mezi aplikacemi pro Android se dosahuje prostřednictvím poskytovatelů obsahu, v původní anglické terminologii Content Providers.

Třída ContentProvider

Třída `ContentProvider` je na platformě Android centralizovaným úložištěm pro strukturované údaje. Zapouzdřuje různé sady dat (data sets) a umožňuje prosazovat požadovaná oprávnění na přístup k těmto údajům. Do určité míry připomíná `ContentProvider` databázi, avšak tato třída byla speciálně navržena pro údaje, které mají být sdíleny napříč různými aplikacemi.

`ContentProvider` poskytuje přístup k strukturovaným údajům sdíleným mezi různými aplikacemi. Tato data jsou aplikaci poskytnuta buď jako databázové tabulky, tedy v podstatě stejně, jako jsou přístupné tabulky databáze SQLite, nebo jako přístup do souboru. Jedná se tedy o přístup typu klient-server, přičemž klient žádá o přístup k údajům, a serverem, který tento požadavek splní (samozřejmě jen pokud může), je `ContentProvider`, tj. poskytovatel obsahu.

Aplikace, které chtějí získat přístup ke sdíleným údajům, využívají třídu `ContentResolver`, která umožňuje aplikacím číst a zapisovat data, která jsou uložena v poskytovatelích obsahu. Podporuje metody `query`, `insert`, `update`, `delete` a také další služby, například informování registrovaných pozorovatelů o změnách.

Údaje mohou být fyzicky uloženy v databázi SQLite, v souborech, v cloudu a podobně, závisí to na konkrétní implementaci.

TIP

`ContentProvider` je možné využívat nejen na sdílení údajů mezi aplikacemi, ale i na přístup k údajům z vlastní aplikace. Výhodou je zkrácení kódu oproti klasickému přístupu do databáze.

Na platformě Android jsou i systémoví poskytovatelé obsahu. Typickým příkladem jsou kontakty, SMS, údaje o zmeškaných hovorech, historie prohlížení webu či multimediální knihovny.

Aplikace potřebuje nejprve získat odkaz na třídu `ContentResolver` pomocí metody `Context.getContentResolver()`. `ContentProvider` a `ContentResolver` společně umožňují, aby kód běžící v jednom procesu přistupoval k údajům spravovaným v jiném procesu.

`ContentProvider` je reprezentován jako databázová tabulka, která má záznamy (řádky) a atributy (sloupce). Aplikace přistupuje k požadovaným údajům přes URI.

Formát řetězce URI je `content://authority/path/id`.

- `content:` – standardní předpona indikující, že data mají být zpracována přes `ContentProvider`.
- `authority` – identifikátor providera, který musí být deklarován v *AndroidManifest.xml*. Řetězec by měl obsahovat jméno balíčku, aby se zabezpečila jedinečnost.
- `path` – slouží ke strukturování obsahu. Může mít několik segmentů.
- `id` – identifikátor požadované položky.

Povinné jsou jen první dvě části. Například URI na přístup ke kontaktům:

```
ContactsContract.Contacts.CONTENT_URI =
  "content://com.android.contacts/contacts/"
```

V tomto případě se jedná o URI pro přístup k celé tabulce, nikoliv ke specifickému záznamu.

Pro `ContentProvider` je potřeba implementovat 6 základních metod:

- `onCreate` – inicializace providerů, nastavení přístupu ke zdroji údajů, například k databázi. Tato metoda se volá, pokud se poskytovatel obsahu vytvoří poprvé a využívá se k provedení nějaké inicializační úlohy vyžadované poskytovatelem obsahu.
- `getType()` – vrácení dat MIME pro konkrétní URI. Formát MIME je definován jako `vnd.X.cursor.dir/Y` nebo `vnd.X.cursor.item/Y`
- `Query()` – výběr požadovaných údajů. Tato metoda se volá, když klient požaduje, aby se údaje získaly od poskytovatele obsahu. Metoda je zodpovědná za identifikaci a načítání údajů. Množinu údajů může tvořit jeden nebo více řádků. Následně se provede extrakce dat a vrácení výsledků zabalených v objektu `Cursor`.
- `Insert()` – přidání údajů. Tato metoda se volá, když klient potřebuje uložit do databáze poskytovatele. Metoda musí určit cíl pro data, realizovat vložení a vrátit úplnou URI nově přidaného řádku.
- `Update()` – aktualizace údajů. Tato metoda se volá, když je zapotřebí aktualizovat existující záznamy v databázi poskytovatele obsahu. Metoda používá argumenty umožňující

aktualizovat příslušné řádky tabulky a vrátí počet řádků aktualizovaných v důsledku operace update.

- `Delete()` – smazání údajů. Metoda se volá, když je potřeba, aby se jeden nebo více záznamů odstranilo z tabulky. Metoda `delete()` odstraní označené řádky a vrátí počet odstraněných řádků.

Je potřeba mít na zřeteli, že metody kromě `onCreate()` nelze volat z více procesů současně, a proto jejich kód musí být vláknově bezpečný.

Na přístup k údajům je určena metoda:

```
Cursor query (Uri uri, String[] projection, String selection,
  String[] selectionArgs, String sortOrder)
```

Připomínáme, že projekce znamená výběr atributů (sloupců) databázové tabulky a selekce je výběr podmnožiny záznamů na základě zadaného kritéria. Metoda `query()` vrací objekt typu `Cursor`.

Ukládání kontaktů v zařízení s Androidem

Několik příkladů z této kapitoly věnované poskytování obsahu přes `ContentProvider` bude využívat databázi kontaktů v zařízení, proto vysvětlíme, jak jsou kontakty v zařízení organizovány a jak se k nim ze strany aplikace dá přistupovat.

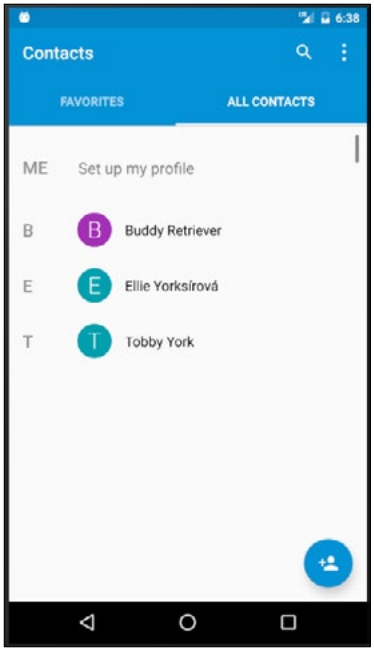
Kontakty se na platformě Android ukládají v centrálním úložišti údajů, v databázi, ve které jsou uloženy systémové údaje telefonu. Aplikace k tomuto úložišti nemůže přistupovat přímo, takže údaje jsou zabezpečené proti zneužití, změně nebo smazání. K přístupu k údajům v této databázi se na straně aplikace používají objekty typu `ContentResolver` a na straně databáze objekty typu `ContentProvider`. V databázi jsou uloženy nejen místní kontakty, tj. kontakty přímo v zařízení, ale i kontakty synchronizované s cloudovými službami či Facebookem. Databázi kontaktů využívá aplikace Telefon, ale i další aplikace, například e-mailový klient, aplikace sociálních sítí, správci kontaktů, aplikace na slučování a synchronizaci kontaktů a podobně.

Příklad – Přístup ke kontaktům v zařízení

První příklad na demonstraci výměny údajů mezi procesy přes `ContentProvider` bude klasický – načíst seznam kontaktů v zařízení a zobrazit je. Aby byla aplikace komplexnější, zobrazíme i obrázky přiřazené ke kontaktům.

Nejprve ukážeme klasickou deklaraci povolení přístupu ke kontaktům, které funguje do API 22, tj. Android 5.1 Lollipop, včetně této verze. Pro tento způsob povolení přístupu vytvořte projekt pro API 22. Aplikace bude potřebovat v souboru *AndroidManifest.xml* deklaraci povolení přístupu k údajům v databázi kontaktů.

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```



Obrázek 18.1: Seznam cvičných kontaktů v emulátoru, ve kterém budeme aplikaci testovat

Když vytvoříte projekt pro API 23, tj. Android 6.0 Marshmallow a novější, musíte přidat do kódu hlavní aktivity blok, který si od uživatele vyžádá povolení při prvním spuštění aplikace.

```
if (ContextCompat.checkSelfPermission(thisActivity,
  Manifest.permission.READ_CONTACTS)
  != PackageManager.PERMISSION_GRANTED) {

  // Vysvetlenie?
  if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
    Manifest.permission.READ_CONTACTS)) {

    // Asynchrónne zobrazenie vysvetlenia
  } else {

    // Vyžiadanie oprávnenia.

    ActivityCompat.requestPermissions(thisActivity,
      new String[]{Manifest.permission.READ_CONTACTS},
      MY_PERMISSIONS_REQUEST_READ_CONTACTS);

  }
}
```

Pokračovat budeme jednodušším kódem, který bude fungovat pro API 22 včetně.

Layout hlavní aktivity nebudeme potřebovat, jelikož tato bude rozšířením třídy `ListActivity`. Budeme potřebovat návrh zobrazení jedné položky seznamu, kde vedle sebe zobrazíme fotografii a jméno kontaktu. Na vodorovné uspořádání položek využijeme kontejnerový prvek

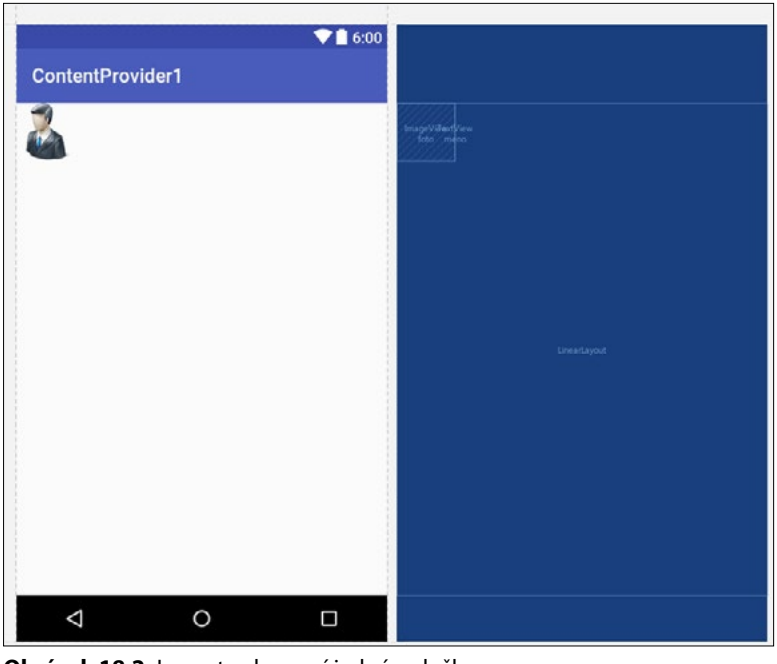
`LinearLayout` s horizontální orientací uspořádání prvků. Návrh layoutu položky seznamu je v souboru *kontakt.xml*. Pro jednoduchost můžete jako zástupný obrázek kontaktu využít ikonu aplikace nebo libovolnou jinou vhodnou ikonu umístěnou například ve složce *drawable*.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal" >

  <ImageView
    android:id="@+id/foto"
    android:layout_width="60dp"
    android:layout_height="60dp"
    android:contentDescription="..."
    android:background="@drawable/user" >
  </ImageView>

  <TextView
    android:id="@+id/meno"
    android:layout_width="wrap_content"
    android:layout_height="60dp"
    android:textSize="30sp" >
  </TextView>

</LinearLayout>
```



Obrázek 18.2: Layout zobrazení jedné položky seznamu

V kódu hlavní aktivity v proměnné `atributy` definujeme dva atributy třídy `ContactsContract.ContactsColumns` – jméno kontaktu a adresu URI obrázku kontaktu. Následně kód hlavní aktivity získá odkaz na `ContentResolver` kontextu. Pro metodu `query` je potřeba upravit řetězec `where` definující kritéria výběru záznamů a řetězec `sort` s definicí pravidel pro seřazení výsledků. Pomocí podmínky v řetězci `where` jsme odfiltrovali kontakty s chybějícími nebo prázdnými názvy. V řetězci `sort` definujeme, že záznamy budou seřazeny v sestupném pořadí podle atributu `_id`. Metoda `query()` vrátí objekt `Cursor` – ukazatel, který umožní aplikaci přistupovat k výsledkům. Nakonec vytvoříme a nastavíme nový adaptér, který se použije k zobrazení seznamu.

```
public class MainActivity extends ListActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    String atributy[] = new String[] { ContactsContract.Contacts._ID,
      ContactsContract.Contacts.DISPLAY_NAME,
      ContactsContract.Contacts.PHOTO_THUMBNAIL_URI};

    ContentResolver contentResolver = getContentResolver();
    String where = "((" + ContactsContract.Contacts.DISPLAY_NAME + "
      NOTNULL) AND ("
      + ContactsContract.Contacts.DISPLAY_NAME + " != '' ))";
    String sort = ContactsContract.Contacts._ID + " ASC";
    Cursor cursor =
      contentResolver.query(ContactsContract.Contacts.CONTENT_URI,
      atributy, where, null, sort);
    setListAdapter(new KontaktyAdapter(this, R.layout.kontakt, cursor, 0));

  }
}
```

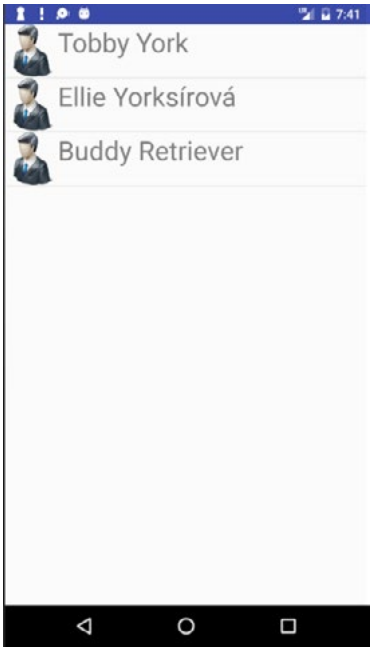
Adaptér na zobrazení seznamu kontaktů je v souboru *KontaktyAdapter.java*. V konstruktoru této třídy je definován předvolený obrázek pro kontakty, které nemají fotografii. Abychom nemuseli vkládat do aplikace další obrázek, například ikonu postavičky, použili jsme ikonu aplikace. V metodě `newView()` vytvoříme přes `LayoutInflater` pohled s využitím layoutu *kontakt.xml*, ve kterém je definován vzhled zobrazení záznamu, v tomto případě fotografie a jméno kontaktu.

V metodě `bindView` přidáme do `View` data. Zobrazí se jméno kontaktu a uloží odkaz na předvolený obrázek. Následně kód kontroluje, zda je pro daný kontakt k dispozici reálná fotografie. Pokud ano, použije se URI tohoto obrázku a otevře se vstupní stream na načítání údajů fotografie do paměti.

```
public class KontaktyAdapter extends ResourceCursorAdapter {
  private Bitmap fakeBmp;

  public KontaktyAdapter(Context ctx, int layout, Cursor c, int flags)  {
    super(ctx, layout, c, flags);
    fakeBmp = BitmapFactory.decodeResource(ctx.getResources(),
      R.drawable.user);
  }

  @Override
```



Obrázek 18.3: Zobrazení kontaktů získaných přes ContentProvider

```
public View newView(Context ctx, Cursor cursor, ViewGroup parent) {
  LayoutInflater inflater = (LayoutInflater) ctx
    .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
  return inflater.inflate(R.layout.kontakt, parent, false);
}

@Override
public void bindView(View view, Context context, Cursor cursor)  {
  TextView textView = (TextView) view.findViewById(R.id.meno);
  textView.setText(cursor.getString(cursor
    .getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME)));

  ImageView imageView = (ImageView) view.findViewById(R.id.foto);
  Bitmap bmp = fakeBmp;

  String obrUri = cursor.getString(cursor
    .getColumnIndex(ContactsContract.Contacts.PHOTO_THUMBNAIL_URI));

  if (obrUri != null)  {
    InputStream is = null;
    try   {
      is = context.getContentResolver().openInputStream(
        Uri.parse(obrUri));
      if (is != null) bmp = BitmapFactory.decodeStream(is);
    } catch (FileNotFoundException e)  {
      Log.i("ERR", "FileNotFoundException");
    }
```

```
      }
      imageView.setImageBitmap(bmp);
    }
  }
```

Nyní můžete aplikaci spustit.

Příklad – Přístup ke kontaktům, načítání údajů na pozadí

Možná jste v předchozím příkladu zpozorněli u věty o načítání fotografie do paměti. Jelikož se v tomto případě jedná jen o malé náhledy, mohli jsme si dovolit načítat je v hlavním vlákně aplikace. Pokud by se jednalo o větší objemy údajů, museli byste zabránit provádění intenzivní práce v hlavním vlákně, protože to může negativně ovlivnit schopnost aplikace reagovat na podněty uživatele a zhoršit uživatelskou zkušenost. Pro tento účel poskytuje Android třídu `CursorLoader`, která využívá `AsyncTask` tak, aby se déle trvající činnosti prováděly ve vlákně na pozadí.

Tato aplikace realizuje dotaz Content Provideru ve vlákně na pozadí pomocí objektu `CursorLoader`. Aplikace vytvoří a nastaví prázdný adaptér, protože v tomto okamžiku ještě nejsou k dispozici žádné údaje.

Pokud chcete v aplikaci použít `CursorLoader`, nejprve je nutné vytvořit objekt, který implementuje rozhraní `LoaderCallback` z třídy `LoaderManager`. Potom za běhu bude muset aplikace vytvořit a inicializovat objekt `CursorLoader` pomocí metody `initLoader()`. První zpětné volání bude `OnCreate` a po načtení údajů se zavolá metoda `onLoadFinished()`.

V metodě `onCreateLoader` připravíme pro `CursorLoader` řetězec `where` definující kritéria výběru záznamů a řetězec `sort` s definicí pravidel na seřazení výsledků.

```
public class MainActivity extends ListActivity implements
LoaderManager.LoaderCallbacks<Cursor>
{
  private KontaktyAdapter kontaktyAdapter;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    kontaktyAdapter = new KontaktyAdapter(this, R.layout.kontakt, null, 0);
    setListAdapter(kontaktyAdapter);
    getLoaderManager().initLoader(0, null, this);
  }

  static final String[] CONTACTS_ROWS = new String[]
    { Contacts._ID, Contacts.DISPLAY_NAME, Contacts.PHOTO_THUMBNAIL_URI };

  @Override
  public Loader<Cursor> onCreateLoader(int id, Bundle args)
  {
    String where = "((" + Contacts.DISPLAY_NAME + " NOTNULL) AND ("
      + Contacts.DISPLAY_NAME + " != '' ))";
```

```
    String sort = Contacts._ID + " ASC";

    return new CursorLoader(this, Contacts.CONTENT_URI, CONTACTS_ROWS,
      where, null, sort);
  }

  @Override
  public void onLoadFinished(Loader<Cursor> loader, Cursor data)
  {
    kontaktyAdapter.swapCursor(data);
  }

  @Override
  public void onLoaderReset(Loader<Cursor> loader)
  {
    kontaktyAdapter.swapCursor(null);
  }
}
```

Adaptér na zobrazení seznamu kontaktů v souboru *KontaktyAdapter.java* je stejný jako v předchozím příkladu.

Příklad – Aktualizace údajů

Objekt `ContentResolver` obsahuje kromě metody `query()` další užitečné metody na práci s údaji.

```
int delete (Uri url, String where, String[] selectArgs)
```

Tato metoda vrací počet řádků, které byly odstraněny.

```
Uri insert (Uri url, ContentValues values)
```

Metoda `insert` má jako parametr URI označující data set, do kterého chcete vložit nový záznam a hodnoty v objektu `ContentValues`. Metoda vrací identifikátor URI pro aktuálně vložený záznam:

```
int update(Uri url, ContentValues values, String where, String[]
  selectionArgs)
```

Metoda na aktualizaci údajů obsahuje i pole pro výběr argumentů a podmínku, která definuje, které konkrétní řádky se budou aktualizovat. Metoda vrací počet řádků, které byly aktualizovány.

Námětem příkladu bude přidání nového kontaktu do seznamu. Aplikace bude potřebovat oprávnění nejen ke čtení seznamu kontaktů, ale i k zápisu do tohoto seznamu a k přístupu k seznamu účtů.

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
```

Jelikož v této aplikaci vypíšeme pouze jména, v layoutu definujícím rozložení prvků seznamu hlavní aktivity bude pouze prvek `TextView`:

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/meno"
  android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_marginLeft="10dp"
    android:textSize="30sp" >
</TextView>
```

Pro jednoduchost jsou údaje o přidávaném kontaktu zadané jako konstanty přímo v kódu hlavní aktivity. V metodě `onCreate` nejprve aplikace získá seznam cloudových účtů, ke kterým je zařízení připojeno, jelikož kontakty v zařízení jsou vázány s uživatelským účtem, zpravidla s účtem Google. Kontakty se vkládají ze seznamu v zařízení v metodě `vlozKontakty()` dávkovým způsobem. Metoda nastaví operace, ve kterých se všechny nové kontakty vloží naráz. Samotné vkládání se realizuje v metodě `pridajZaznamDoDavky()`. Nejprve definujeme atributy, které pro nové záznamy chceme vložit, a následně hodnoty pro konkrétní záznam.

```
import java.util.ArrayList;
import java.util.List;
import android.support.v4.app.LoaderManager.LoaderCallbacks;

public class MainActivity extends ListActivity implements
  LoaderManager.LoaderCallbacks<Cursor> {

  public final static String[] kontakty = new String[] { "Tobby Yorkshir",
    "Filip Bigel"};

  private static final String atributy[] = new String[]
    {ContactsContract.Contacts._ID, ContactsContract.Contacts.DISPLAY_NAME};
  private static final String atributy1[] = new String[] {
    ContactsContract.Contacts.DISPLAY_NAME };
  private static final int[] idRes = new int[] { R.id.meno };

  private Account[] konto;
  private String typ;
  private String meno;
  private SimpleCursorAdapter cursorAdapter;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    konto = AccountManager.get(this).getAccountsByType("com.google");
    typ = konto[0].type;
    meno = konto[0].name;
    vlozKontakty();

    cursorAdapter = new SimpleCursorAdapter(this, R.layout.activity_main,
      null,
      atributy1, idRes, 0);
    setListAdapter(cursorAdapter);
    getLoaderManager().initLoader(0, null, this);
  }

  private void vlozKontakty() {
    ArrayList<ContentProviderOperation> davka = new
```

```
    ArrayList<ContentProviderOperation>();

  for (String name : kontakty) pridejZaznamDoDavky(name, davka);
  try  {
    getContentResolver().applyBatch(ContactsContract.AUTHORITY, davka);
  } catch (RemoteException e) {
    Log.i("ERR", "RemoteException");
  } catch (OperationApplicationException e) { Log.i("ERR",
    "RemoteException");}
  }
}

private void pridejZaznamDoDavky(String name,
  List<ContentProviderOperation> ops) {
  int position = ops.size();

  ops.add(ContentProviderOperation.newInsert(
    ContactsContract.RawContacts.CONTENT_URI)
    .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, typ)
    .withValue(ContactsContract.RawContacts.ACCOUNT_NAME, meno)
    .withValue(ContactsContract.Contacts.STARRED, 1).build());

  ops.add(ContentProviderOperation.newInsert(
    ContactsContract.Data.CONTENT_URI)
    .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID,
    position)
    .withValue(ContactsContract.Data.MIMETYPE,
    ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)
    .withValue(ContactsContract.CommonDataKinds.
    StructuredName.DISPLAY_NAME, name).build());

}

private void deleteContact(String name) {
  getContentResolver().delete(ContactsContract.RawContacts.CONTENT_URI,
    ContactsContract.Contacts.DISPLAY_NAME + "=?",
    new String[] { name });

}

public Loader<Cursor> onCreateLoader(int id, Bundle args) {

  String select = "((" + Contacts.DISPLAY_NAME + " NOTNULL) AND ("
    + ContactsContract.Contacts.DISPLAY_NAME + " != '' ) AND (" +
    ContactsContract.Contacts.STARRED
    + "== 1))";

  return new CursorLoader(this, Contacts.CONTENT_URI, atributy,
    select, null, Contacts._ID + " ASC");
}
```

```
    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
      cursorAdapter.swapCursor(data);
    }

    public void onLoaderReset(Loader<Cursor> loader) {
      cursorAdapter.swapCursor(null);
    }

  }
```

Takto jednoduchá aplikace, která vloží kontakty z konstantních řetězců v metodě `onCreate`, je samozřejmě v praxi nepoužitelná, stačí změnit orientaci zařízení, metoda se zavolá znovu a znovu přidá ty samé kontakty, jako demonstrace použití metody `insert()` však postačí.

Příklad – Vytvoření aplikace, která bude poskytovat údaje

V předchozích příkladech jsme ukázali, jak přistupovat k údajům z jiných procesů. V tomto příkladu vytvoříme aplikaci, která bude poskytovatelem obsahu. Aplikace bude potřebovat úložný systém pro údaje. K tomuto účelu můžete využít databázi SQLite, která je součástí operačního systému Android. To bude námětem dalšího příkladu. U tohoto příkladu vystačíte s jednodušším řešením. Údaje budou uloženy v jednoduché datové struktuře typu `SparseArray`.

Aplikace neobsahuje aktivity, v souboru *AndroidManifest.xml* je ale potřeba definovat provider:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.contentprovider4"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="19" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <provider
      android:name="com.example.contentprovider4.SeznamContentProvider"
      android:authorities="com.example.contentprovider4"
      android:exported="true" >
    </provider>
  </application>

</manifest>
```

Implementujeme třídu `DbZaznam`, která definuje strukturu údajů v databázové tabulce. Hodnota unikátního identifikátoru `_id` se bude po vložení záznamu automaticky inkrementovat.

```
class DbZaznam
{
  private static int id;
  private final int _id;
  private final String _hodnota;

  DbZaznam(String _hodnota)
  {
    this._hodnota = _hodnota;
    this._id = id++;
  }

  String getData() {return _hodnota;}

  int getID() {return _id;}
}
```

V aplikaci je potřeba vytvořit třídu `Contract` a implementovat podtřídu `ContentProvider`.

```
public final class DataContract
{
  private static final Uri BASE_URI =
    Uri.parse("content://com.example.contentprovider4/");

  public static final Uri CONTENT_URI = Uri.withAppendedPath(BASE_URI,
    "data_table");

  public static final String CONTENT_DIR_TYPE =
    ContentResolver.CURSOR_DIR_BASE_TYPE
     + "/SeznamContentProvider.hodnota.text";

  public static final String CONTENT_ITEM_TYPE =
    ContentResolver.CURSOR_ITEM_BASE_TYPE
    + "/SeznamContentProvider.hodnota.text";

  public static final String[] ALL_COLUMNS = {"_id", "hodnota" };
}
```

V souboru *SeznamContentProvider.java* jsou implementovány metody `query`, `insert`, `delete`, `update` a `getType`.

```
public class SeznamContentProvider extends ContentProvider
{
  private static final SparseArray<DbZaznam> db = new
    SparseArray<DbZaznam>();

  @Override
    public synchronized Cursor query(Uri uri, String[] projection,
    String selection, String[] selectionArgs, String sortOrder)
  {
    MatrixCursor cursor = new MatrixCursor(DataContract.ALL_COLUMNS);
    if (isTableUri(uri))
    {
      for (int nc = 0; nc < db.size(); nc++)
```

```
    {
      DbZaznam dbz = db.get(db.keyAt(nc));
      cursor.addRow(new Object[] { dbz.getID(), dbz.getData() });
    }
  } else if (isItemUri(uri))
  {
    Integer id = Integer.parseInt(uri.getLastPathSegment());
    if (null != db.get(id))
    {
      DbZaznam dbz = db.get(id);
      cursor.addRow(new Object[] { dbz.getID(), dbz.getData() });
    }
  }
  return cursor;
}

@Override
  public synchronized Uri insert(Uri uri, ContentValues value)
{
  if (value.containsKey(DataContract.HODNOTA))
  {
    DbZaznam dbz = new DbZaznam(value.getAsString(DataContract.HODNOTA));
    db.put(dbz.getID(), dbz);
    return Uri.withAppendedPath(DataContract.CONTENT_URI,
      String.valueOf(dbz.getID()));
  }
  return null;
}

@Override
  public synchronized int update(Uri uri, ContentValues values,
  String selection, String[] selectionArgs)
{
  return 0;
}

@Override
  public synchronized int delete(Uri uri, String selection, String[]
  selectionArgs)
{
  int nOdstranene = 0;
  if (isTableUri(uri))
  {
    nOdstranene = db.size();
    db.clear();
  }
  else if (isItemUri(uri))
  {
    Integer reqId = Integer.parseInt(uri.getLastPathSegment());

    if (null != db.get(reqId))
```

```
        {
          db.remove(reqId);
          nOdstranene++;
        }
      }
      return nOdstranene;
    }

    @Override
      public synchronized String getType(Uri uri)
    {
      String typ = DataContract.CONTENT_ITEM_TYPE;
      if (isTableUri(uri)) typ = DataContract.CONTENT_DIR_TYPE;
      return typ;
    }

    @Override
      public boolean onCreate()
    {
      return true;
    }

    private boolean isItemUri(Uri uri)
    {
      return uri.getLastPathSegment().matches("\\d+");
    }

    private boolean isTableUri(Uri uri)
    {
      return uri.getLastPathSegment().equals(DataContract.DATA_TABLE);
    }

  }
```

Příklad – Vytvoření aplikace, která bude poskytovat údaje z databáze

Jak bylo avizováno v předchozím příkladu, ukážeme vytvoření aplikace, která bude poskytovatelem obsahu a bude poskytovat údaje z databáze SQL Lite, jež je součástí operačního systému Android na každém zařízení. Námětem aplikace bude jednoduchá databáze studentů, přesněji absolventů, která obsahuje jen tři atributy: První je unikátní identifikátor ID, který se nebude zadávat, ale při vkládání nového záznamu se metodou automatické inkrementace vytvoří tak, že k identifikátoru předcházejícího záznamu se přičte 1. Další dva atributy budou jméno studenta a nejvyšší dosažené vzdělání. Atribut `meno` bude pro jednoduchost obsahovat jméno i příjmení.

Hlavní aktivita obsahuje dva prvky typu `EditText` na zadávání údajů a dvě tlačítka: Jedno na vložení záznamu a jedno na výpis. Výpis bude realizován pro jednoduchost pomocí toastové notifikace postupně záznam po záznamu. Prvky jsou uspořádány vertikálně uvnitř vizuálního kontejneru `LinearLayout`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context="sk.pcrevue.llacko.contentprovider4.MainActivity">

  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText2"
    android:hint="Meno"
    android:textColorHint="@android:color/holo_blue_light" />
  <EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/editText3"
    android:hint="Stupeň vzdelania"
    android:textColorHint="@android:color/holo_blue_bright" />
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:text="Pridaj študenta"
    android:onClick="onClickPridaj"/>
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Daj záznam"
    android:id="@+id/button"
    android:onClick="onClickZaznamy"/>

</LinearLayout>
```

Dříve než se dostaneme ke tvorbě kódu hlavní aktivity a třídy poskytovatele obsahu z databáze, je potřeba v souboru *AndroidManifest.xml* definovat provider.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.contentprovider4">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
```

```
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <provider android:name="StudentProvider"
      android:authorities=
      "sk.pcrevue.llacko.contentprovider4.StudentProvider"/>
  </application>
</manifest>
```

Jádrem kódu hlavní aktivity jsou metody na obsluhu události stisknutí tlačítka na přidání záznamu a výpis seznamu studentů. Všimněte si, jak je realizován postupný výpis záznamů v databázi pomocí toastové notifikace.

```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void onClickPridaj(View view) {
    // Add a new student record
    ContentValues values = new ContentValues();
    values.put(StudentProvider.MENO,
    ((EditText)findViewById(R.id.editText2)).getText().toString());

    values.put(StudentProvider.VZDELANIE,
    ((EditText)findViewById(R.id.editText3)).getText().toString());

    Uri uri = getContentResolver().insert(
      StudentProvider.CONTENT_URI, values);

    Toast.makeText(getBaseContext(),
      uri.toString(), Toast.LENGTH_LONG).show();
  }

  public void onClickZaznamy(View view) {
    String URL =
      "content://sk.pcrevue.llacko.contentprovider4.StudentProvider";

    Uri students = Uri.parse(URL);
    Cursor c = managedQuery(students, null, null, null, "meno");

    if (c.moveToFirst()) {
      do{
        Toast.makeText(this,
          c.getString(c.getColumnIndex(StudentProvider._ID)) +
          ", " +  c.getString(c.getColumnIndex( StudentProvider.MENO)) +
          ", " + c.getString(c.getColumnIndex( StudentProvider.VZDELANIE)),
          Toast.LENGTH_SHORT).show();
      } while (c.moveToNext());
```

```
    }
  }
}
```

Těžiště práce s databází je na třídě `StudentProvider`. Nejprve ukážeme konstanty na vytvoření databáze `DbAbsolventi` a v ní vytvoření databázové tabulky `studenti`.

```
private SQLiteDatabase db;
static final String DATABASE_NAME = "DBAbsolventi";
static final String TABLE_NAME = "studenti";
static final int DATABASE_VERSION = 1;
static final String CREATE_DB_TABLE =
  " CREATE TABLE " + TABLE_NAME +
  " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
  " meno TEXT NOT NULL, " +
  " vzdelanie TEXT NOT NULL);";
```

Samotná práce s údaji se realizuje ve třídě `DatabaseHelper`. Všimněte si, že záznamy budou ve výpisu seřazeny podle jména studenta.

```
if (sortOrder == null || sortOrder == "") sortOrder = MENO;
```

Kompletní kód třídy `StudentProvider` bude:

```
public class StudentProvider extends ContentProvider {
  static final String PROVIDER_NAME =
    "sk.pcrevue.llacko.contentprovider4.StudentProvider";
  static final String URL = "content://" + PROVIDER_NAME + "/student";
  static final Uri CONTENT_URI = Uri.parse(URL);

  static final String _ID = "_id";
  static final String MENO = "meno";
  static final String VZDELANIE = "vzdelanie";

  private static HashMap<String, String> STUDENTS_PROJECTION_MAP;

  static final int STUDENT = 1;
  static final int STUDENT_ID = 2;

  static final UriMatcher uriMatcher;
  static{
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI(PROVIDER_NAME, "student", STUDENT);
    uriMatcher.addURI(PROVIDER_NAME, "student/#", STUDENT_ID);
  }


  private SQLiteDatabase db;
  static final String DATABASE_NAME = "DBAbsolventi";
  static final String TABLE_NAME = "studenti";
  static final int DATABASE_VERSION = 1;
  static final String CREATE_DB_TABLE =
    " CREATE TABLE " + TABLE_NAME +
    " (_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
```

```
    " meno TEXT NOT NULL, " +
    " vzdelanie TEXT NOT NULL);";

  private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context){
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
      db.execSQL(CREATE_DB_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){
      db.execSQL("DROP TABLE IF EXISTS " +  TABLE_NAME);
      onCreate(db);
    }
  }

  @Override
  public boolean onCreate() {
    Context context = getContext();
    DatabaseHelper dbHelper = new DatabaseHelper(context);
    db = dbHelper.getWritableDatabase();
    return (db == null)? false:true;
  }

  @Override
  public Uri insert(Uri uri, ContentValues values) {
    long rowID = db.insert( TABLE_NAME, "", values);
    if (rowID > 0) {
      Uri _uri = ContentUris.withAppendedId(CONTENT_URI, rowID);
      getContext().getContentResolver().notifyChange(_uri, null);
      return _uri;
    }

    throw new SQLException("chyba pridania záznamu " + uri);
  }

  @Override
  public Cursor query(Uri uri, String[] projection,
    String selection,String[] selectionArgs, String sortOrder) {
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    qb.setTables(TABLE_NAME);

    switch (uriMatcher.match(uri)) {
      case STUDENT:
        qb.setProjectionMap(STUDENTS_PROJECTION_MAP);
        break;
```

```
    case STUDENT_ID:
      qb.appendWhere( _ID + "=" + uri.getPathSegments().get(1));
      break;

    default:
  }

  if (sortOrder == null || sortOrder == "") sortOrder = MENO;

  Cursor c = qb.query(db, projection, selection,
    selectionArgs,null, null, sortOrder);
  c.setNotificationUri(getContext().getContentResolver(), uri);
  return c;
}

@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
  int count = 0;
  switch (uriMatcher.match(uri)){
    case STUDENT:
      count = db.delete(TABLE_NAME, selection, selectionArgs);
      break;

    case STUDENT_ID:
      String id = uri.getPathSegments().get(1);
      count = db.delete( TABLE_NAME, _ID +  " = " + id +
        (!TextUtils.isEmpty(selection) ? "AND (" + selection + ')' : ""),
        selectionArgs);
      break;
    default:
      throw new IllegalArgumentException("Unknown URI " + uri);
  }

  getContext().getContentResolver().notifyChange(uri, null);
  return count;
}

@Override
public int update(Uri uri, ContentValues values,
  String selection, String[] selectionArgs) {
  int count = 0;
  switch (uriMatcher.match(uri)) {
    case STUDENT:
      count = db.update(TABLE_NAME, values, selection, selectionArgs);
      break;

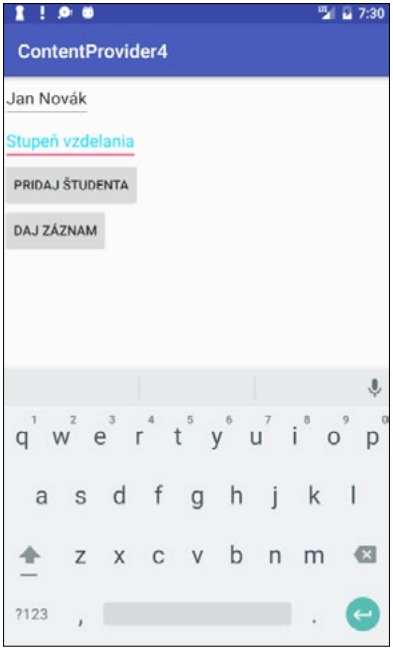
    case STUDENT_ID:
      count = db.update(TABLE_NAME, values,
        _ID + " = " + uri.getPathSegments().get(1) +
        (!TextUtils.isEmpty(selection) ? "AND (" +selection + ')' : ""),
        selectionArgs);
```

```
            break;
          default:
            throw new IllegalArgumentException("Unknown URI " + uri );
        }

        getContext().getContentResolver().notifyChange(uri, null);
        return count;
      }

      @Override
      public String getType(Uri uri) {
        switch (uriMatcher.match(uri)){
          case STUDENT:
            return "vnd.android.cursor.dir/vnd.example.students";
          case STUDENT_ID:
            return "vnd.android.cursor.item/vnd.example.students";
          default:
            throw new IllegalArgumentException("Unsupported URI: " + uri);
        }
      }
    }
```

V tomto stadiu můžete aplikaci vyzkoušet. Postupně vložte několik záznamů a následně je nechejte vypsat.



Obrázek 18.4: Zadávání údajů do databáze



Obrázek 18.5: Výpis údajů z databáze přes toastovou notifikaci

KAPITOLA

19

Vývoj pro nositelná zařízení

Témata kapitoly:

- **Zaměření na účel**
- **Vytvoření emulátoru Android Wear**
- **Projekt aplikace pro Android Wear**
- **Posílání notifikace do hodinek**
- **Hospodaření s plochou displeje**
- **Aplikace na popředí**

V současnosti je na trhu několik typů hodinek od různých výrobců, ne všechny jsou však kompatibilní s platformou Android Wear. Například Samsung využívá pro svoje hodinky vlastní operační systém Tizen. Dříve než se pustíte do vývoje aplikace pro inteligentní hodinky, zamyslete se nad účelem a především filozofií jejich používání, aby vaše aplikace měla smysl a pro uživatele byla přínosem.

Hodně lidí přestalo nosit hodinky a na zjišťování času používají telefon, takže značný podíl interakce s telefonem představuje tento účel. Podle výsledků průzkumů se uživatelé telefonů s Androidem podívají na jeho displej 125krát za den. Pokaždé jej musí vyndat z kapsy, případně i z pouzdra. Vývoj však jde po spirále a operativnost hodinek ke zjišťování informací upoutala pozornost tvůrců mobilních platforem. Tak vznikla platforma Android Wear. Základem filozofie nové platformy je operativní poskytnutí relevantních informací při co nejjednodušším a nejintuitivnějším ovládání.

Zaměření na účel

Účelem hodinek je poskytnout uživateli množství drobných, ale v daném okamžiku velmi potřebných informací. Všimněte si, že hodinky vnímáme jako komplexní celek, a ne jako místo na zapouzdření aplikací, jinak bychom napsali „cílem aplikace v hodinkách". Někteří výrobci hodinek s proprietárním operačním systémem a následně i vývojáři pro tyto platformy se pomocí hodinek snaží duplikovat chytrý telefon. Na malém displeji tak najdete ikony aplikací, které lze posléze spouštět. Málokdy je to účelné. Pro Android Wear sice můžete vytvořit samostatnou hru, která bude fungovat na hodinkách, ale kromě výjimek potvrzujících pravidlo takováto hra uživatele hodinek rychle omrzí.

Neuděláte chybu, když budete Android Wear považovat za notifikační lištu pro svůj chytrý telefon s Androidem, jakousi prodlouženou ruku běžného smartphonu. Hodinky nebo náramek mají v první řadě podávat užitečné informace, například

upozornění na příchozí zprávy, výpis zajímavých notifikací, upozornění na aktuální problémy, například dopravní omezení a podobně. Na hodinky se uživatel rychle podívá a případně pouze něco rychle potvrdí. Není to smartphone, s kterým při vyhledávání pracujete o něco déle, ale také ne příliš dlouho typicky několik sekund, případně desítky sekund.

Android Wear proto vyžaduje odlišný přístup k navrhování designu a fungování celých aplikací. Na malém displeji by se měla zobrazovat jen skutečně důležitá oznámení a aplikace by se měly dát pohodlně ovládat velkými tlačítky. Google pro tyto účely v rámci Wear vytvořil systém kartiček, mezi kterými je možné jednoduchým gestem přepínat. Na hodinkách je potřeba lidem nabídnout informace a minimum interakcí, ne možnost spouštění a používání aplikací. To jejich konstrukční formát neumožňuje, případně jen za cenu soustředění, a tím i uživatelského nepohodlí. Nemá smysl duplikovat funkce aplikací v telefonu.

Jako příklad účelného využití inteligentních hodinek popíšeme typický scénář při cestě letadlem. Děj začíná upozorněním na nutnost odchodu na letiště a sejmutím kódu QR při nástupu do letadla (samozřejmě z displeje hodinek). Hodinky vám pak poskytnou informaci o hotelu v cílové destinaci, kterou můžete na hodinkách operativně ukázat taxikáři. Oceníte to obzvlášť v asijských zemích, kde mají jinou abecedu. Stejný scénář v těchto zemích můžete použít i jako překladač – jednoduše ukážete člověku, se kterým se pokoušíte komunikovat, přeloženou frázi na displeji hodinek. Scénář pokračuje operativním zjištěním předpovědi počasí. A můžete pokračovat: Zjistíte, ve které restauraci v nejbližším okolí mají nejlepší pizzu, a v předstihu si ji objednáte. Hodinky vás pak navigují k restauraci, kde vám přinesou objednané jídlo, za které jediným klepnutím na displej hodinek zaplatíte.

TIP

Pokud si do tabletu nebo telefonu uživatel nainstaluje aplikaci využívající součinnost s hodinkami, část aplikace běžící v hodinkách se do nich nainstaluje automaticky.

Návrh a vývoj aplikace pro zařízení, která nosíme na sobě, například hodinky, brýle a podobně, vzhledem ke specifikám těchto zařízení vyžaduje jinou filozofii než návrh aplikací pro mobilní telefony. Jiná konstrukce a filozofie ovládání vyžaduje jiný přístup k návrhu uživatelského rozhraní. Zařízení poskytují informace buď na bázi nabídky (ekvivalent oznámení), nebo dotazu.



Obrázek 19.1: Uživatelské rozhraní platformy Android Wear. V tomto případě se konkrétně jedná o konstrukční faktor čtvercových hodinek

Nabídka: Kontextový stream

Nabídka informací je realizována jako vertikální seznam karet, z nichž každá obsahuje užitečné informace. Uživatelé se podobně jako u telefonů a tabletů přesouvají gesty mezi kartami obsahujícími relevantní informace. Vzhledem k malým rozměrům displeje jsou informace do maximální míry zhuštěny. Na zobrazení podrobností slouží další karty.

Zařízení se ovládá pomocí několika základních gest. Vertikální pohyb prstu slouží k přepínání mezi kartami. Horizontální gesto směrem doleva umožní zobrazení podrobnějších detailů. Posunem doprava informaci potvrdíte, případně zrušíte notifikaci. Klepnutím na tlačítko nebo jiný ovládací prvek potvrdíte akci.

U klasického telefonu máte na různé akce různé aplikace a ty mají zpravidla více různých karet, takže získat požadovanou informaci nebývá vždy jednoduché. Model uživatelského rozhraní platformy Android Wear zajišťuje, že uživatelé nemusí spustit více aplikací kvůli kontrole aktualizací, například zpráv, statusů ze sociálních sítí a podobně. Místo toho mohou jednoduše přepínat karty se stručnými aktualizacemi informací, které jsou pro ně důležité.

POZNÁMKA

Aplikace pro platformu Android Wear může vytvářet karty a aplikovat je do kontextového streamu vždy, když mohou být s vysokou pravděpodobností uživateli užitečné.

Karty ve streamu jsou více než obyčejnými notifikacemi. Vodorovným gestem může uživatel zobrazit další související karty, případně ovládací prvky.

Dotaz: Cue cards

V případech, kdy zařízení neposkytuje notifikace a odpovědi z kontextového streamu, může získávat informace z Googlu. Aktivuje se hlasovým povelem „OK Google“ nebo poklepáním na pozadí domovské obrazovky. Gestem, tedy posouváním prstu po obrazovce hodinek, se zobrazí seznam navrhovaných hlasových příkazů, které mohou být aktivovány hlasem nebo poklepáním.

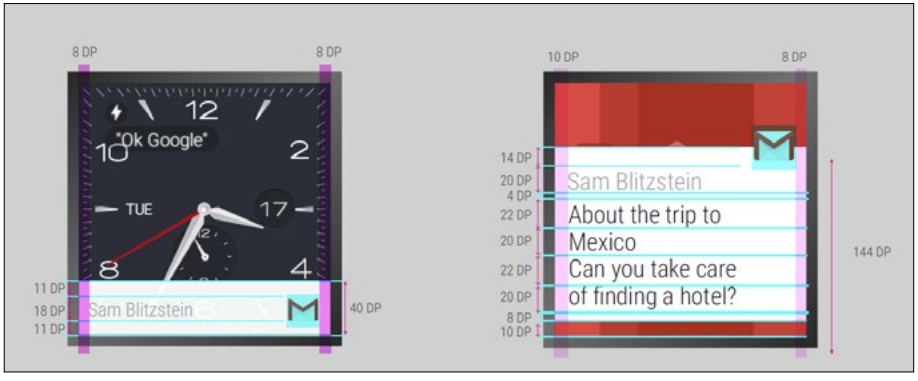
Z hlediska vývojáře takovéto gesto nebo hlasový příkaz aktivují specifický typ záměru, tedy vytvoří objekt typu `Intent`. V aplikacích tedy mohou vývojáři některé z těchto záměrů realizovat. Z hlediska uživatele vaše aplikace zrealizuje úlohu zadanou pomocí hlasových příkazů. Pokud je na příslušný typ záměru zaregistrovaných více aplikací, uživatel má možnost si vybrat, kterou aplikaci na danou úlohu upřednostní. Aplikace tedy mohou reagovat na hlasový příkaz stejným způsobem, jako klasické aplikace reagují na klasické podněty, například klepnutí na tlačítko.

Výchozím stavem zařízení je domovská obrazovka, která zobrazuje buď obsah vztahující se k první kartě, nebo hodinky ukazující čas s takovým designem ciferníku, jaký si uživatel zvolil. Klepnutím kamkoliv na displej nebo hlasovým povelem „Ok Google“ se hodinky aktivují a jsou připraveny přijmout hlasový povel pro požadovanou funkci nebo si tuto funkci může uživatel vybrat na dotykové obrazovce. Zařízení zobrazuje také stavové indikátory, které ukazují konektivitu, stav baterie, režim v letadle, případně celkový počet nepřečtených položek.

Zařízení se mohou přepnout do režimu Ambient Mode s nízkou spotřebou, který realizuje stmívání obrazovky. Proto karty s obsahem k nahlédnutí budou automaticky optimalizovány k zobrazení v tomto stavu. Uživatelé mohou opustit tento režim klepnutím na obrazovku nebo nakloněním zařízení.



Obrázek 19.2: Doporučení designu aplikace pro Android Wear – ikony pro jednoduché a intuitivní ovládání

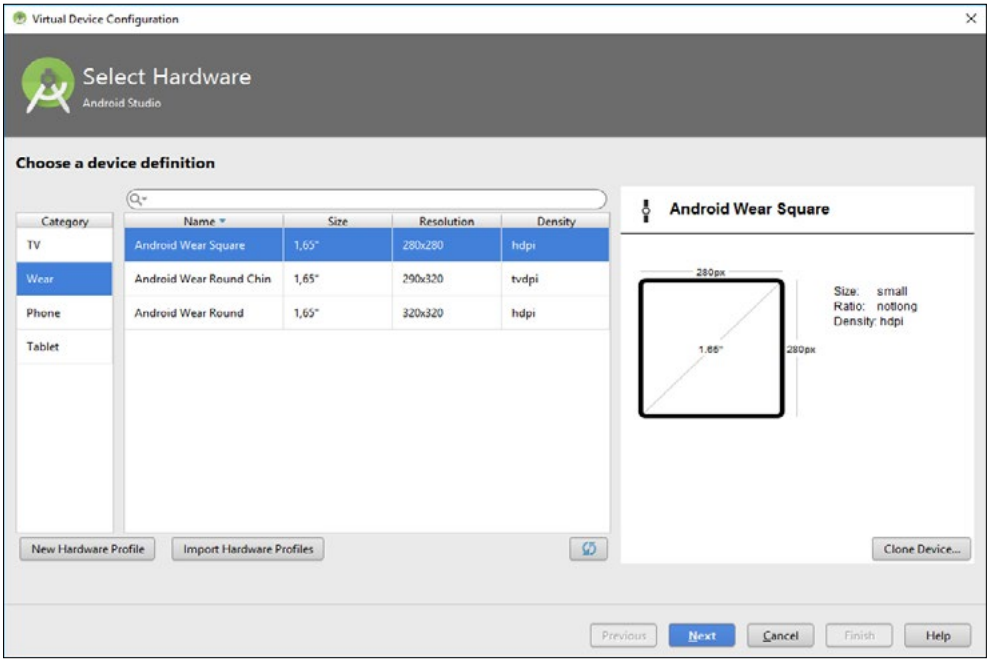


Obrázek 19.3: Doporučení designu aplikace pro Android Wear – notifikace

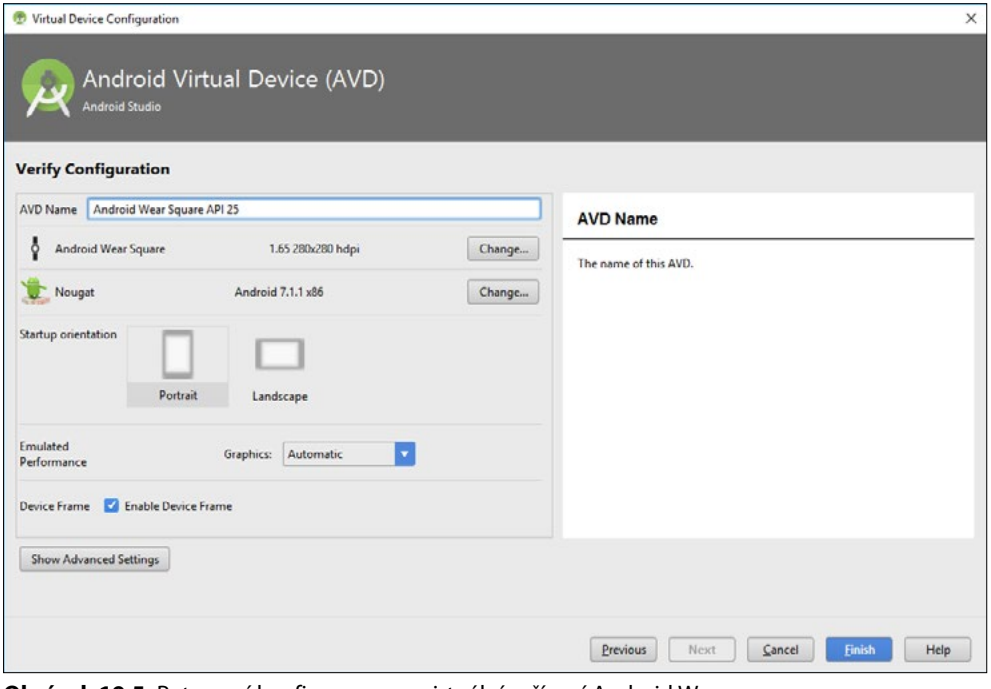
Vytvoření emulátoru Android Wear

Pro platformu Android Wear můžete začít vyvíjet aplikace i v případě, že nemáte reálné zařízení k dispozici. Do značné míry ho nahradí virtuální zařízení. Na vytvoření emulátoru spusťte **AVD Manager**. V prvním kroku zvolte tvar displeje emulátoru: čtvercový, elipsovitý nebo kruhový.

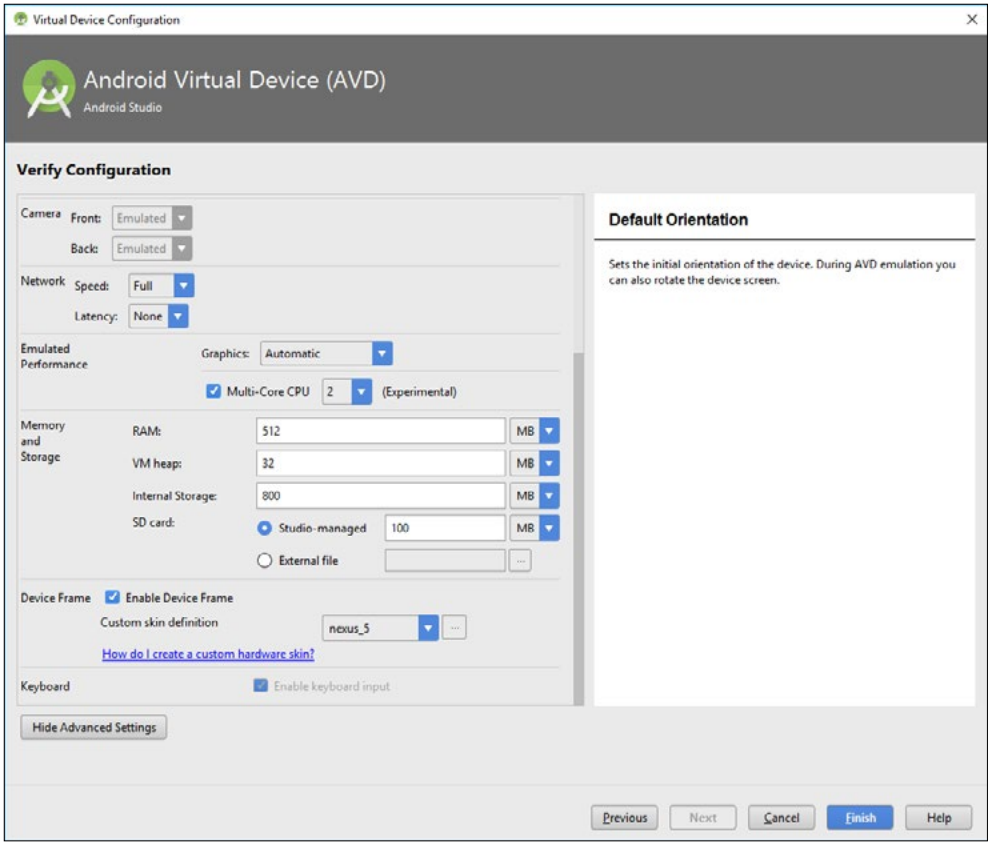
Následuje volba verze platformy. Pokud potřebujete speciální nastavení parametrů, například omezenou kapacitu paměti, aktivujte tlačítko **Show Advances Settings**.



Obrázek 19.4: Výběr formátu pro virtuální zařízení Android Wear



Obrázek 19.5: Potvrzení konfigurace pro virtuální zařízení Android Wear



Obrázek 19.6: Možnosti pokročilého nastavení

Následně je potřeba spárovat zařízení s Androidem (telefon nebo tablet) s emulátorem. Pomocí tohoto zařízení připojeného přes USB budete moct operativně do emulátoru posílat notifikace. Na zařízení nainstalujte z Google Play aplikaci Android Wear. Po náběhu emulátoru připojte zařízení Android přes port USB. Aby propojení fungovalo, musíte přesměrovat komunikační port AVD na připojené zařízení.

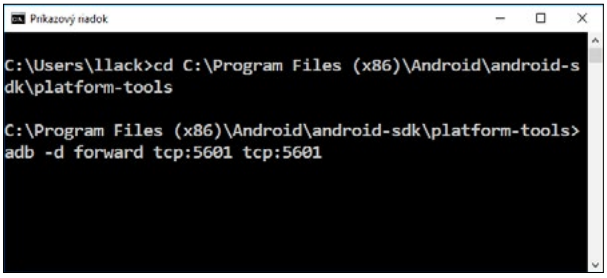
Z konzolové aplikace operačního systému spusťte příkaz:

```
adb -d forward tcp:5601 tcp:5601
```

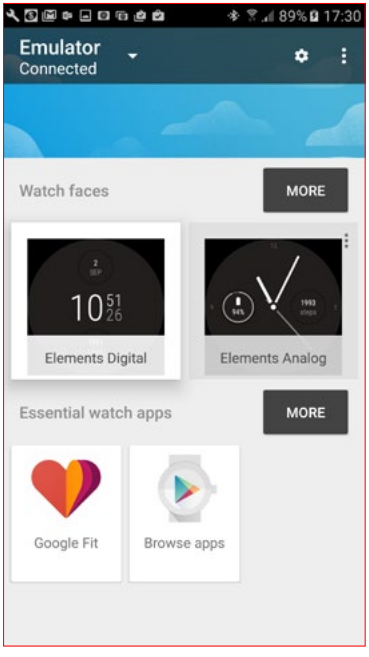
Aplikaci *adb.exe* najdete v adresáři, kam jste nainstalovali Android SDK, v podadresáři *\sdk\platform-tools*.

Adresář v konzolové aplikaci změníte příkazem:

```
cd C:\Program Files (x86)\Android\android-sdk\platform-tools
```



Obrázek 19.7: Přesměrování komunikačního portu



Obrázek 19.8: Aplikace Android Wear na smartphonu

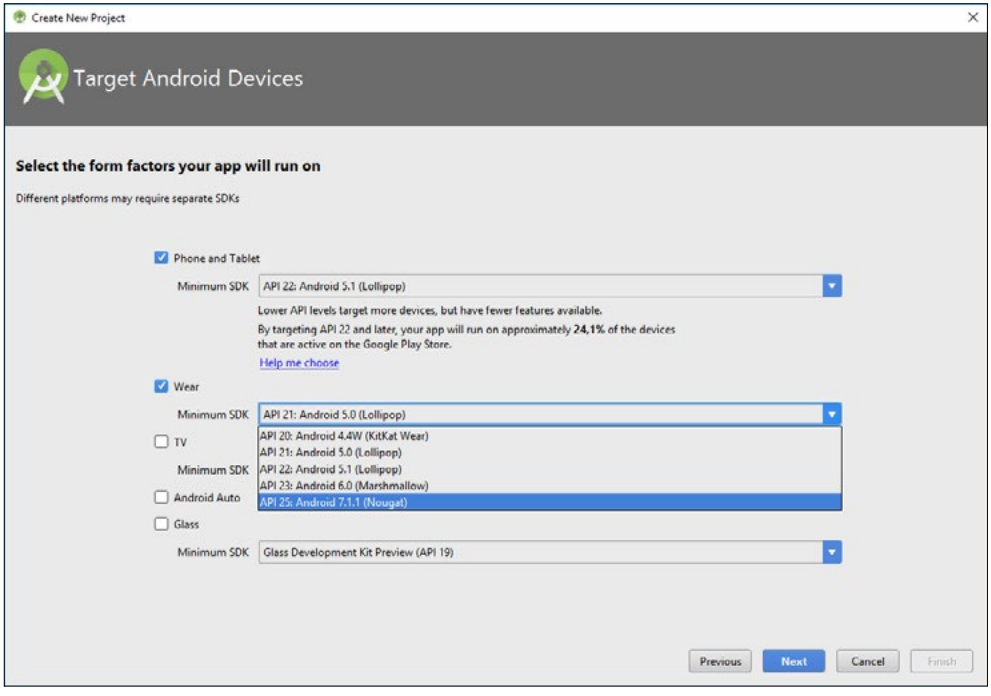
Spusťte na připojeném zařízení aplikaci Android Wear a připojte se z ní k emulátoru. V nabídce aplikace zvolte **Pair with emulator**. Úspěšné připojení vyzkoušejte například tak, že z telefonu změníte ciferník v emulátoru.

Projekt aplikace pro Android Wear

V Android Studiu vytvořte projekt standardním postupem, jako cílové API nastavte jednu z voleb:

- API 20: Android 4.4 (KitKat Wear),
- API 21: Android 5.0 (Lollipop)
- API 22: Android 51 (Lollipop)
- API 23: Android 6.0 (Marshmallow)
- API 25: Android 7.1.1 (Nougat)

Při návrhu uživatelského rozhraní přepněte zobrazování layoutu na **Android Wear** (**Square**, nebo **Round**).



Obrázek 19.9: Výběr verze Androidu na hodinkách

Všimněte si, že dialog má momentálně sekce:

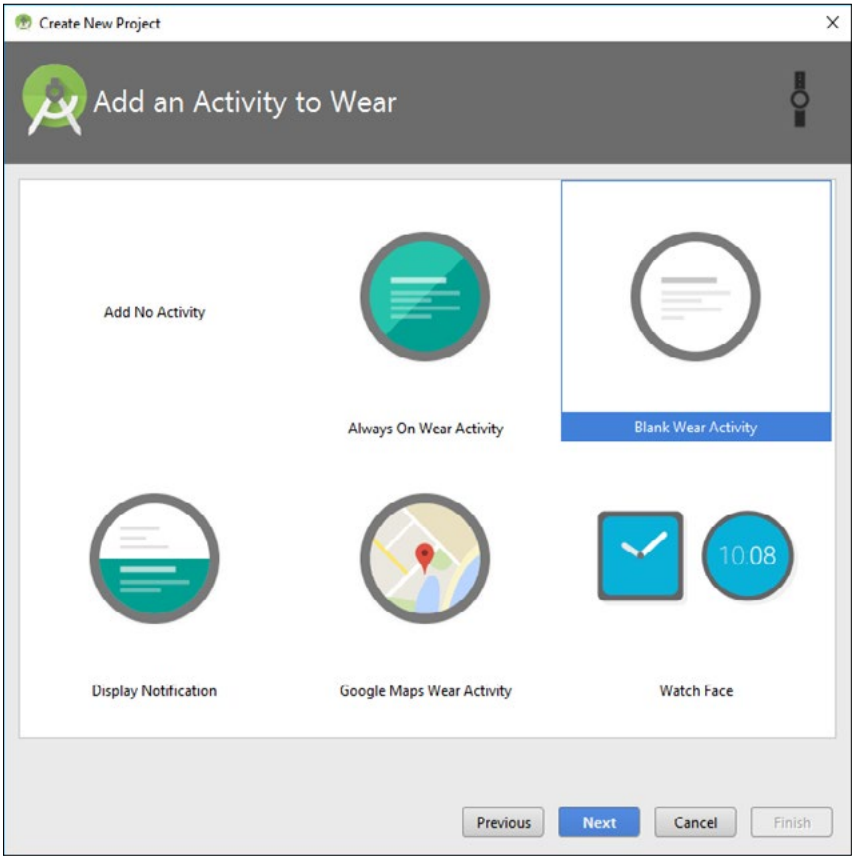
- Phone and Tablet
- Wear
- TV
- Android Auto
- Glass

Z toho logicky vyplývá, že z projektu můžete vytvořit aplikace pro několik sesterských platforem. My vytvoříme aplikaci na hodinkách nebo jiném zařízení platformy Android Wear, ale i na klasickém telefonu či tabletu.

POZNÁMKA

Vytvoření tandemu dvou projektů je logické. Uživatel bude mít s sebou zařízení wearable zároveň s telefonem a tato zařízení budou mezi sebou komunikovat.

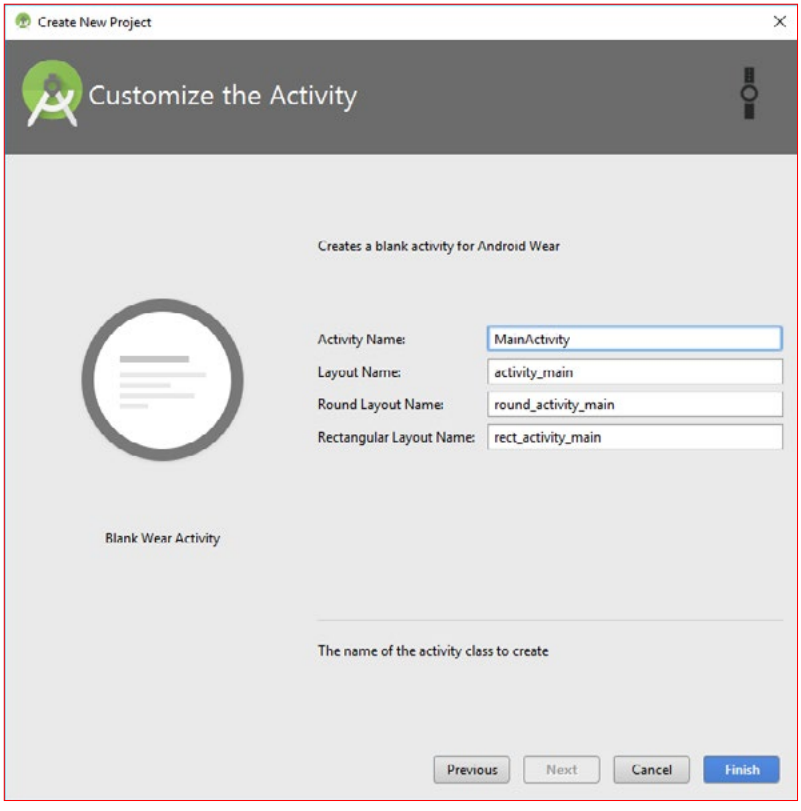
V dialogu **Add an Activity** přidejte pro telefon a tablet aktivitu typu **Blank Activity** a následně pro hodinky aktivitu **Blank Wear Activity**.



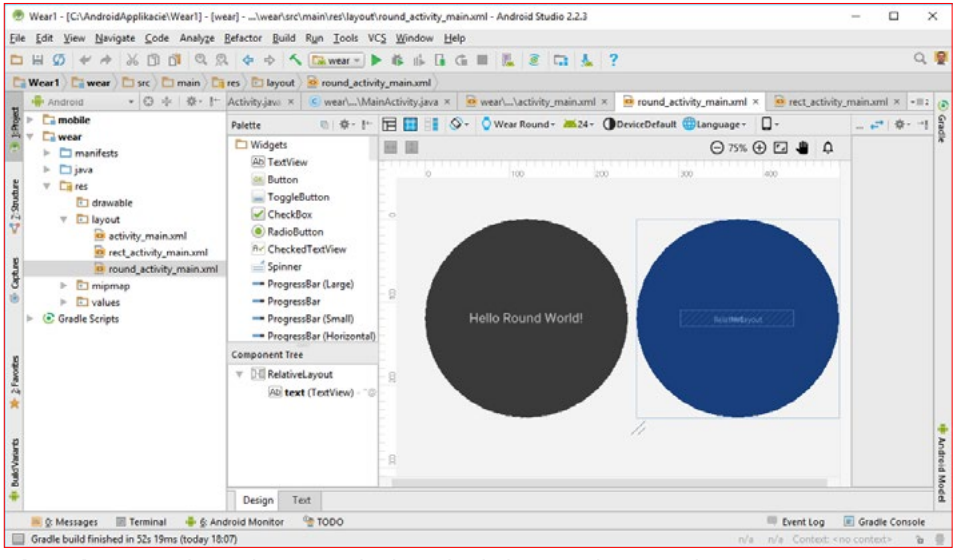
Obrázek 19.10: Výběr typu aktivity pro hodinky

Po dokončení zadávání údajů v dialozích průvodce vytvoří Android Studio nový projekt se dvěma moduly: mobile a wear. Nyní máte vytvořený projekt pro svůj telefon či tablet i pro zařízení wearable. Jak rozdělit úlohy pro tento spolupracující tandem? Aplikace na telefonu bude řešit náročnější úlohy, například síťové komunikace nebo náročnější zpracování údajů, nebo také úlohy, které vyžadují intenzivní interakci s uživatelem. Na zařízení wearable se budou posílat aktuální notifikace o ukončení činnosti, připomínky a podobně.

Modul wear obsahuje po vytvoření projektu dvě hlavní aktivity, pro kruhové i čtvercové hodinky, které vypíší text „Hello World“. Po vytvoření projektu se vám možná zobrazí obrazovka pro návrh hlavní aktivity na typickém velkém obdélníkovém displeji telefonu. Proto v záhlaví přepněte náhled **Wear Round** nebo **Wear Square**, podle toho, pro který tvar displeje právě navrhujete hlavní aktivitu aplikace pro hodinky. Všimněte si, že už v takovéto jednoduché aplikaci vypisující jeden textový řetězec je návrh jiný pro čtvercové hodinky, kde je text u horního okraje okraji displeje, a jiný pro kruhové hodinky, kde je text ve středu displeje, kde může být textové oznámení nejširší.



Obrázek 19.11: Všimněte si nastavení dvou typů hlavní aktivity pro čtvercové a kruhové hodinky



Obrázek 19.12: Hlavní aktivita pro kruhové hodinky v návrhovém zobrazení

Návrhový kód hlavní aktivity je prakticky identický jako u aktivity pro telefon či tablet:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.wear1.MainActivity"
  tools:deviceIds="wear_round">

  <TextView
    android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_round" />
</RelativeLayout>
```

Kód hlavní aktivity pro wearable po vytvoření projektu je:

```
import android.app.Activity;
import android.os.Bundle;
import android.support.wearable.view.WatchViewStub;
import android.widget.TextView;

public class MainActivity extends Activity {

  private TextView mTextView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final WatchViewStub stub = (WatchViewStub)
      findViewById(R.id.watch_view_stub);
    stub.setOnLayoutInflatedListener(new
      WatchViewStub.OnLayoutInflatedListener() {
      @Override
      public void onLayoutInflated(WatchViewStub stub) {
        mTextView = (TextView) stub.findViewById(R.id.text);
      }
    });
  }
}
```

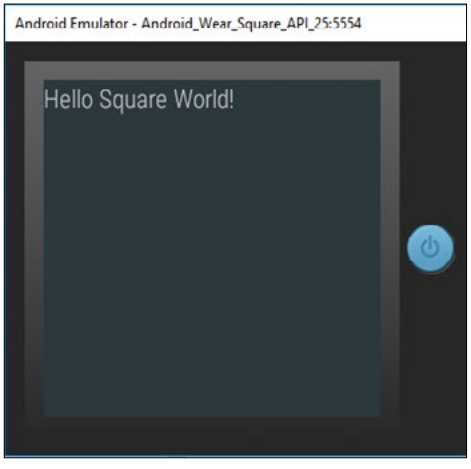
V souboru *build.gradle* (Module: wear) si všimněte sekce `dependencies`, ve které je definovaná verze SDK pro nositelné zařízení.

```
apply plugin: 'com.android.application'

android {
  compileSdkVersion 24
```

```
    buildToolsVersion "24.0.0"
    defaultConfig {
      applicationId "sk.pcrevue.llacko.wear1"
      minSdkVersion 21
      targetSdkVersion 24
      versionCode 1
      versionName "1.0"
    }
    buildTypes {
      release {
        minifyEnabled false
          proguardFiles getDefaultProguardFile('proguard-android.txt'),
          'proguard-rules.pro'
      }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.google.android.support:wearable:2.0.0-beta2'
    compile 'com.google.android.gms:play-services-wearable:10.0.1'
}
```



Obrázek 19.13: Spuštění aplikace v emulátoru

Posílání notifikace do hodinek

Námětem první aplikace bude poslat notifikaci z telefonu, k němuž jsou hodinky – nebo v našem případě emulátor platformy Android Wear – připojené.

Zdůrazňujeme, že pro část projektu mobile, tedy pro mobilní telefon, bude hlavní aktivita obsahovat jen jeden prvek `EditText` a jedno tlačítko na poslání notifikace.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:orientation="vertical"
  tools:context="sk.pcrevue.llacko.wear2.MainActivity">

  <EditText
    android:id="@+id/editText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp"/>
  <Button
    android:id="@+id/actionButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:text="Pošli notifikáciu"
    android:onClick="sendNotification"/>
</LinearLayout>
```

Jádrem aplikačního kódu je poslání notifikace.

```
public class MainActivity extends Activity {

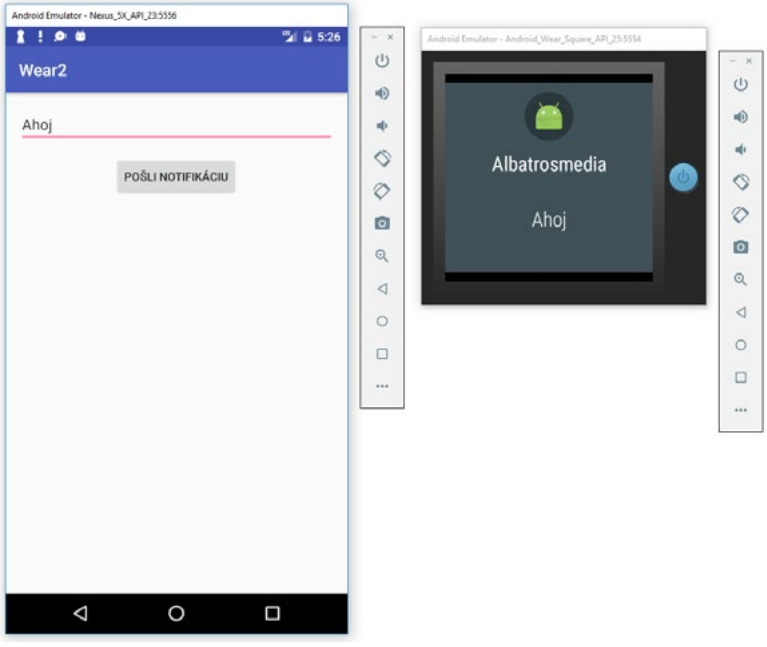
  private EditText editText;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final WatchViewStub stub = (WatchViewStub)
      findViewById(R.id.watch_view_stub);
    stub.setOnLayoutInflatedListener(new
      WatchViewStub.OnLayoutInflatedListener() {
      @Override
      public void onLayoutInflated(WatchViewStub stub) {
        editText = (EditText) stub.findViewById(R.id.editText);
      }
    });
  }

  public void sendNotification(View view) {
    String toSend = editText.getText().toString();
    if(toSend.isEmpty())
      toSend = "prazdna notifikácia";
    Notification notification = new
```

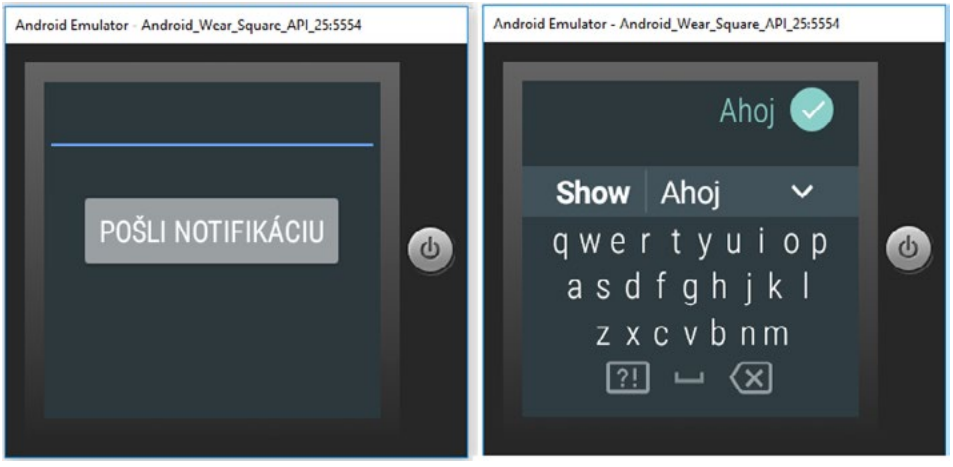
```
        NotificationCompat.Builder(getApplication())
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentTitle("Albatrosmedia")
        .setContentText(toSend)
        .extend(new
        NotificationCompat.WearableExtender().setHintShowBackgroundOnly(true))
        .build();
      NotificationManagerCompat notificationManager =
        NotificationManagerCompat.from(getApplication());
      int notificationId = 1;
      notificationManager.notify(notificationId, notification);
    }
  }
```

Nyní můžete aplikaci spustit a poslat notifikaci na hodinky připojené k zařízení s Androidem.



Obrázek 19.14: Spuštění aplikace v emulátoru

Můžete zkusit zajímavý pokus a návrhový kód hlavní aktivity pro mobilní zařízení a kód Java překopírovat do modulu wear, například do aktivity pro hodinky čtvercového tvaru. Po spuštění aplikace hodinky pošlou notifikaci samy sobě.



Obrázek 19.15: Spuštění aplikace na emulátoru hodinek pošle notifikaci na ty samé hodinky

Hospodaření s plochou displeje

Displej hodinek je – co se týká rozměrů i počtu pixelů – mnohem menší než u telefonu, o tabletu ani nemluvě. Proto musí aplikace s jeho plochou dobře hospodařit. Ukážeme jednoduchý příklad, kdy aplikace po spuštění zobrazí tlačítko. Jím se aktivuje požadovaná akce. V tomto případě spojíme dva náměty dohromady a ukážeme, jak se řeší čekání na dokončení nějaké akce. Během zobrazení kruhového ukazatele průběhu, prvku `DelayedConfirmationView`, bude tlačítko skryté.

V tomto příkladu budeme řešit návrh uživatelského rozhraní i aplikační kód v modulu wear, tedy pro hodinky. V našem případě ukážeme kód pro čtvercové hodinky. Návrhový kód aktivity bude využívat vizuální kontejner `Frame Layout`, který umožňuje zobrazit jen jeden prvek (view), v našem případě buď tlačítko, nebo prvek `DelayedConfirmationView`.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context="sk.pcrevue.llacko.wear2.MainActivity"
  tools:deviceIds="wear_square">
  <Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="Stlač"
    android:onClick="btAkcia" />
  <android.support.wearable.view.DelayedConfirmationView
    android:id="@+id/delayedView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:layout_gravity="center_horizontal"
    app:circle_border_color="@color/green"
    app:circle_border_width="20dp"
    app:circle_color="@color/white"
    app:circle_radius="60dp"
    app:circle_radius_pressed="60dp"
    app:circle_padding="16dp"
    app:update_interval="100"/>
</FrameLayout>
```

V kódu aktivity se v obsluze stisknutí tlačítka, tj. v metodě `btAkcia`, zobrazí prvek promítající průběh čekání. V tomto případě čekáme na doběhnutí časovače nastaveného na 3 sekundy. Všimněte si, jak se nastavuje viditelnost obou prvků.

```
public class MainActivity extends Activity {

  private Button button;
  private DelayedConfirmationView delayedView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final WatchViewStub stub = (WatchViewStub)
      findViewById(R.id.watch_view_stub);
    stub.setOnLayoutInflatedListener(new
      WatchViewStub.OnLayoutInflatedListener() {
      @Override
      public void onLayoutInflated(WatchViewStub stub) {
        button = (Button) stub.findViewById(R.id.button);
        delayedView = (DelayedConfirmationView)
          stub.findViewById(R.id.delayedView);
        delayedView.setTotalTimeMs(3000);
        showOnlyButton();

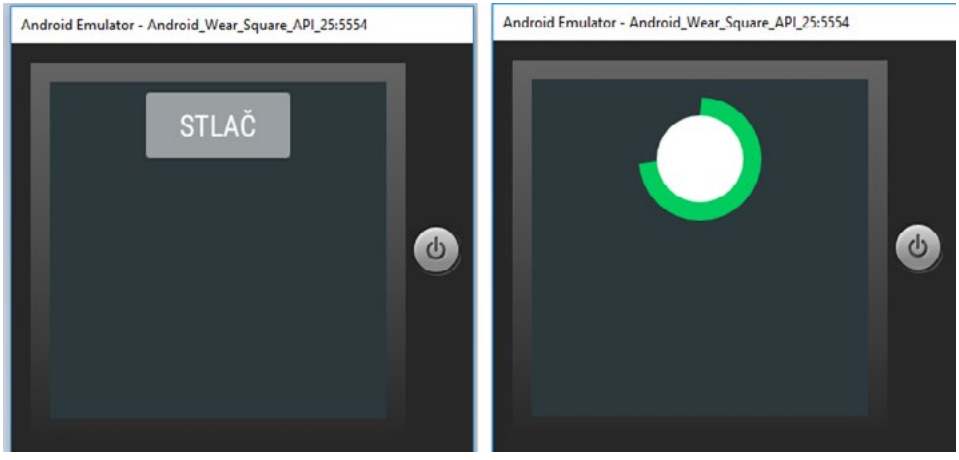
      }
    });
  }

  public void btAkcia(View view) {
    button.setVisibility(View.GONE);
    delayedView.setVisibility(View.VISIBLE);
    delayedView.setListener(new
      DelayedConfirmationView.DelayedConfirmationListener() {
      @Override
      public void onTimerFinished(View view) {
        showOnlyButton();
      }
      @Override
      public void onTimerSelected(View view) {
      }
    });
```

```
      delayedView.start();
    }

    public void showOnlyButton() {
      button.setVisibility(View.VISIBLE);
      delayedView.setVisibility(View.GONE);
    }
  }
```

V tomto případě pro hodinky čtvercového tvaru.



Obrázek 19.16: Skrytí tlačítka během zobrazování jiného prvku

Aplikace na popředí

Energie v hodinkách a náramcích je nedostatková komodita, a tak je většina aplikací navržena tak, aby zobrazila požadovanou informaci a následně se hodinky přepnuly do režimu s nízkou spotřebou. Některé aplikace Wear však potřebují být viditelné, dokud je uživatel neukončí, například když si běžec monitoruje pulz nebo když nakupujeme v obchodě podle seznamu v hodinkách a podobně. Znovu zdůrazňujeme, že stále viditelné aplikace mají negativní vliv na výdrž baterie, takže byste měli bedlivě zvážit, zda tuto funkcionalitu do své aplikace přidat.

Zařízení Android Wear se systémem Android verze 5.1 nebo vyšší umožňují aplikacím zůstat na popředí a zároveň šetřit energii baterie. Zařízení se i během zobrazení aplikace nachází v režimu s nízkou spotřebou. Aplikace, které běží v obou režimech, tedy v ambient i interaktivním režimu, se nazývají always-on aplikace.

Do aplikačního manifestu přidejte do sekce `application` sdílenou knihovnu wearable a povolení `WAKE_LOCK`.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.wear1">

  <uses-feature android:name="android.hardware.type.watch" />
```

```
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@android:style/Theme.DeviceDefault">

    <uses-library android:name="com.google.android.wearable"
      android:required="false" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />

    <activity
      android:name=".MainActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Hlavní aktivita bude odvozená od třídy `WarableActivity` a do metody `OnCreate` přidejte volání metody `setAmbientEnabled()`.

```
public class MainActivity extends WearableActivity {
  private TextView mTextView;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final WatchViewStub stub = (WatchViewStub)
      findViewById(R.id.watch_view_stub);
    stub.setOnLayoutInflatedListener(new
      WatchViewStub.OnLayoutInflatedListener() {
        @Override
        public void onLayoutInflated(WatchViewStub stub) {
          mTextView = (TextView) stub.findViewById(R.id.text);
        }
    });
    setAmbientEnabled();
  }
}
```

Pointou aplikace je řídit přechody mezi jednotlivými režimy. Pokud uživatel nemá žádnou interakci s vaší aplikací po určitý čas nebo pokud se dotkne obrazovky dlaní, aby hodinky „uspal“, systém přejde do módu ambient. V takovémto typu aplikace byste měli používat černé pozadí s minimálním množstvím bílého textu a grafiky. Pro ulehčení přechodu z interaktivního režimu do režimu ambient je třeba zachovat podobné umístění objektů na obrazovce.

Když aktivita přepne do režimu ambient, systém volá metodu `onEnterAmbient()`. Následující fragment kódu ukazuje, jak změnit barvu textu na bílou a vypnout antialiasing potom, co se systém přepne do tohoto režimu.

```
@Override
public void onEnterAmbient(Bundle ambientDetails) {
  super.onEnterAmbient(ambientDetails);

  mStateTextView.setTextColor(Color.WHITE);
  mStateTextView.getPaint().setAntiAlias(false);
}
```

Po přepnutí do interaktivního režimu se vrátí původní barvy, například:

```
@Override
public void onExitAmbient() {
  super.onExitAmbient();

  mStateTextView.setTextColor(Color.GREEN);
  mStateTextView.getPaint().setAntiAlias(true);
}
```

Klíčovou funkcionalitou aplikace je obnovovat kontext v módu ambient. Je třeba řešit kompromis aktuálnost versus výdrž baterie. Doporučujeme využít metodu `onUpdateAmbient()`, která umožňuje aktualizovat obrazovku v režimu ambient jednou za minutu. Pokud vaše aplikace vyžaduje častější aktualizace a smíříte se se zkrácením provozu na baterii, můžete intervaly obnovování zkrátit, pokud ale nechcete spotřebovat příliš mnoho energie, aktualizace by neměla být častější než jednou za 10 sekund.

Pokud stačí aktualizovat jednou za minutu, systém poskytuje volání callback metody `onUpdateAmbient()`, která umožňuje aktualizovat obrazovku s touto doporučenou frekvencí. Pokud chcete aktualizovat obsah aplikace, přepište metodu `onUpdateAmbient()`:

```
@Override
public void onUpdateAmbient() {
  super.onUpdateAmbient();

  // Update the content
}
```

Když potřebujete zobrazený kontext na hodinkách aktualizovat častěji, typickým příkladem je fitness, měření času či navigace, můžete pomocí objektu `AlarmManager` probudit procesor z hibernace a aktualizovat obrazovku častěji. Postup je možné shrnout do několika bodů:

- Vytvořte objekt `AlarmManager`
- Nastavte frekvenci aktualizací
- Naplánujte další aktualizaci, když aktivita přepne do ambient režimu nebo je v současné době v tomto režimu
- Zrušte alarm, když se aktivita přepne do interaktivního režimu nebo pokud je aktivita aplikace zastavená

`AlarmManager` může vytvořit nové instance vaší aktivity, když je spuštěná. Aby se předešlo této situaci, je zapotřebí, aby byla aktivita v manifestu deklarovaná `android: launchMode = parameter "Singleinstance"`.

V následující ukázce kódu je objekt `AlarmManager` vytvořen v metodě `OnCreate()`. Objekt bude po vytvoření ve stavu čekání na záměr:

```
private AlarmManager mAmbientStateAlarmManager;
private PendingIntent mAmbientStatePendingIntent;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  setAmbientEnabled();

  mAmbientStateAlarmManager =
    (AlarmManager) getSystemService(Context.ALARM_SERVICE);
  Intent ambientStateIntent =
    new Intent(getApplicationContext(), MainActivity.class);

  mAmbientStatePendingIntent = PendingIntent.getActivity(
    getApplicationContext(),
    0,
    ambientStateIntent,
    PendingIntent.FLAG_UPDATE_CURRENT);
  ...
}
```

Když se alarm spustí a aktivuje čekající záměr, je nutné aktualizovat obrazovku a naplánovat další alarm pomocí metody `onNewIntent()`:

```
@Override
public void onNewIntent(Intent intent) {
  super.onNewIntent(intent);

  setIntent(intent);
  refreshDisplayAndSetNextUpdate();
}
```

V tomto příkladu `AlarmManager` spouští alarm v režimu ambient každých 20 sekund. Je potřeba aktualizovat obrazovku a nastavit zpoždění pro budoucí aktualizaci.

```
// interval prebúdzania procesora treba premeniť na milisekundy
private static final long AMBIENT_INTERVAL_MS =
  TimeUnit.SECONDS.toMillis(20);

private void refreshDisplayAndSetNextUpdate() {

  if (isAmbient()) {
    // spracuj údaje a refresh displeja v ambient mode
  } else {
    // spracuj údaje a refresh displeja v interaktivnom mode
  }
```

```
    long timeMs = System.currentTimeMillis();

    // naplánovanie nového alarmu
    if (isAmbient()) {
      // výpočet času pre trigger
      long delayMs = AMBIENT_INTERVAL_MS - (timeMs % AMBIENT_INTERVAL_MS);
      long triggerTimeMs = timeMs + delayMs;

      mAmbientStateAlarmManager.setExact(
        AlarmManager.RTC_WAKEUP,
        triggerTimeMs,
        mAmbientStatePendingIntent);

    } else {
      // výpočet času pre trigger pre interktívny mod
    }
  }
```

Naplánování alarmu, aby se aktualizovala obrazovka, když je aktivita ve stavu přechodu do režimu ambient nebo už v tomto režimu je, dosáhneme přepsáním metod `onEnterAmbient()` a `onUpdateAmbient()`:

```
@Override
public void onEnterAmbient(Bundle ambientDetails) {
  super.onEnterAmbient(ambientDetails);

  refreshDisplayAndSetNextUpdate();
}

@Override
public void onUpdateAmbient() {
  super.onUpdateAmbient();

  refreshDisplayAndSetNextUpdate();
}
```

Po přechodu do interaktivního módu je potřeba zrušit alarm.

```
@Override
public void onExitAmbient() {
  super.onExitAmbient();

  mAmbientStateAlarmManager.cancel(mAmbientStatePendingIntent);
}
```

Když uživatel aktivitu vaší aplikace zastaví, je potřeba zrušit alarm v metodě `onDestroy()` příslušné aktivity:

```
@Override
public void onDestroy() {
  mAmbientStateAlarmManager.cancel(mAmbientStatePendingIntent);
  super.onDestroy();
}
```

KAPITOLA

20

Novinky verze 7.0 Nougat

Témata kapitoly:

- **Vylepšené notifikace**
- **Podpora více oken (Multi-Window)**
- **Vývoj aplikace pro Android TV**
- **Spuštění aplikace pro Android TV na emulátoru**

V době dokončení rukopisu bylo procentuální zastoupení tehdy nejnovější verze Android 7.0 Nougat podle přehledu portálu Google *http://developer.android.com/about/dashboards/index.html* 0,7 %, v době vydání publikace to už bude podstatně více. Je tedy vhodné osvojit si co nejvíce funkcí nové verze a využívat je ve svých aplikacích. Ukážeme využití některých nejzajímavějších funkcionalit.

Vylepšené notifikace

Významnou novinkou uživatelského rozhraní, kterou uživatel velmi rychle objeví, je nová notifikační lišta. Notifikace se zobrazují v komprimovanější a zároveň přehlednější podobě. Oznamovací panel zabírá celou šířku obrazovky, takže dokáže zobrazit více informací. Ikony aplikací v oznámení jsou mnohem menší, neodděluje je orámování, ale jen tenká lišta. Pokud chcete zobrazit celý obsah delšího oznámení nebo více oznámení některé aplikace, stačí na nejnovější zobrazení položit prst a posunout ho dolů. Oznámení můžete po přečtení vymazat nebo na něj reagovat, například odpovědět na zprávu v Messengeru nebo jiné aplikaci. Samozřejmě jen v aplikaci, která bude tuto vlastnost podporovat.

Vytvořte projekt s minimálním nastavením verze SDK Android 7.0 Nougat. Uživatelské rozhraní aktivity obsahuje jen jediné tlačítko, pomocí kterého je možné poslat notifikaci.

```
<RelativeLayout xmlns:android="http://schemas.
android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.notifikacie1.
MainActivity">

  <Button
    android:text="Pošli"
    android:layout_width="100dp"
```

```
    android:layout_height="50dp"
    android:id="@+id/button"
    android:onClick="btPosli" />
</RelativeLayout>
```

V kódu hlavní aktivity je obsluha události stisknutí tlačítka, ve které se pošle notifikace. Notifikace jsou tvořeny pomocí třídy `NotificationCompat.Builder`, která umožňuje nastavit ikonu, název a text notifikace. Zatím jsme neprovedli nic nového. Android 7.0 však umožňuje reagovat na notifikaci. V tomto příkladu se spustí druhá aktivita. Kód hlavní aktivity vyšle záměr, který spustí druhou aktivitu.

```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  protected void btPosli(View view) {

    Uri defaultSoundUri= RingtoneManager.getDefaultUri(
      RingtoneManager.TYPE_NOTIFICATION);

    NotificationCompat.Builder builder =
      new NotificationCompat.Builder(this)
      .setSmallIcon(android.R.drawable.ic_dialog_info)
      .setContentTitle("Notifikácia")
      .setContentText("text notifikácie")
      .setSound(defaultSoundUri);

    Intent resultIntent = new Intent(this, Druha.class);

    PendingIntent pendingIntent =
      PendingIntent.getActivity(this, 0,resultIntent,
      PendingIntent.FLAG_UPDATE_CURRENT
      );

    NotificationCompat.Action action = new NotificationCompat.Action.Builder(
      android.R.drawable.sym_action_chat,
      "Open", pendingIntent).build();

    builder.addAction(action);
    builder.setContentIntent(pendingIntent);
    int notificationId = 101;

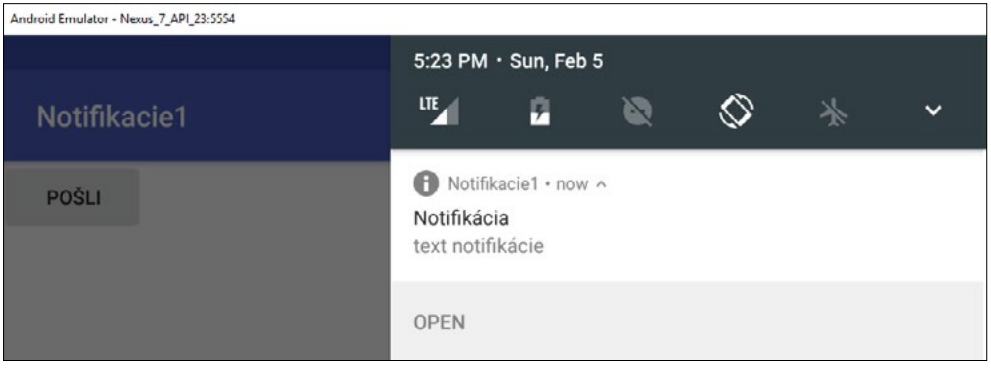
    NotificationManager notifyMgr = (NotificationManager)
      getSystemService(NOTIFICATION_SERVICE);
    notifyMgr.notify(notificationId, builder.build());
  }
}
```

Pomocí místní nabídky **New** → **Activity** → **Empty Activity** složky **App** vytvořte druhou aktivitu, která bude obsahovat jen jediný prvek `TextView`:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_druha"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.notifikacie1.Druha">

  <TextView
    android:text="DRUHÁ AKTIVITA"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"/>
</RelativeLayout>
```


Nyní můžete aplikaci otestovat. Tlačítkem pošlete notifikaci, prohlédněte si ji na notifikační liště a tlačítkem **Open** pod notifikací aktivujte druhou aktivitu aplikace. Ta by v reálné aplikaci vypsala například podrobnosti o události, které se notifikace týkala.



Obrázek 20.1: Zobrazení notifikace

Další novinkou, kterou Android 7.0 v oblasti notifikací přináší, je sdružování notifikací. Když přijde více notifikací k té samé události, zabraly by u klasického zobrazení hodně místa. Každé oznámení musí být označeno, že patří do příslušné skupiny, pomocí metody `setGroup()`.

```
public class MainActivity extends AppCompatActivity {

  final static String GROUP_KEY_NOTIFY = "group_key_notify";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  protected void btPosli(View view) {
```

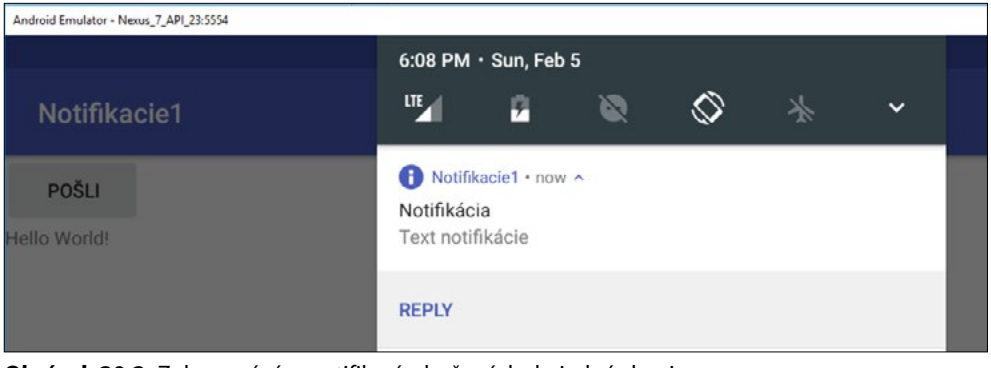
```
        NotificationCompat.Builder builder1 =
          new NotificationCompat.Builder(this)
          .setSmallIcon(android.R.drawable.ic_dialog_info)
          .setContentTitle("Notifikácia1")
          .setContentText("text notifikácie 1")
          .setGroup(GROUP_KEY_NOTIFY);


        NotificationCompat.Builder builder2 =
          new NotificationCompat.Builder(this)
          .setSmallIcon(android.R.drawable.ic_dialog_info)
          .setContentTitle("Notifikácia2")
          .setContentText("text notifikácie 2")
          .setGroup(GROUP_KEY_NOTIFY);

        NotificationCompat.Builder builderSum =
          new NotificationCompat.Builder(this)
          .setSmallIcon(android.R.drawable.ic_dialog_info)
          .setContentTitle("Sumárna notifikácia")
          .setContentText("text sumárnej notifikácie")
          .setGroup(GROUP_KEY_NOTIFY)
          .setGroupSummary(true);

        int notificationId1 = 101;
        int notificationId2 = 102;
        int notificationId3 = 103;

        NotificationManager notifyMgr = (NotificationManager)
          getSystemService(NOTIFICATION_SERVICE);
        notifyMgr.notify(notificationId1, builder1.build());
        notifyMgr.notify(notificationId2, builder2.build());
        notifyMgr.notify(notificationId3, builderSum.build());
      }
    }
```



Obrázek 20.2: Zobrazení více notifikací sdružených do jedné skupiny

V posledním příkladu ukážeme, jak můžeme z notifikace poslat zprávu do aplikace. Hlavní aktivita bude obsahovat i prvek TextView na zobrazení zprávy, kterou napíšeme jako odpověď na notifikaci.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="sk.pcrevue.llacko.notifikacie1.MainActivity"
  android:orientation="vertical">

  <Button
    android:text="Pošli"
    android:layout_width="100dp"
    android:layout_height="50dp"
    android:id="@+id/button"
    android:onClick="btPosli" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:id="@+id/textView" />

</LinearLayout>
```

V kódu je potřeba v metodě OnCreate vytvořit objekt RemoteInput. Intent od oznámení směřuje do hlavní aktivity.

```
public class MainActivity extends AppCompatActivity {

  private static int notificationId = 101;
  private static String KEY_REPLY = "key_reply";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Intent intent = this.getIntent();
    Bundle remoteInput = RemoteInput.getResultsFromIntent(intent);

    if (remoteInput != null) {
      TextView myTextView = (TextView) findViewById(R.id.textView);
      String inputString = remoteInput.getCharSequence(
        KEY_REPLY).toString();

      myTextView.setText(inputString);

      Notification repliedNotification =
        new Notification.Builder(this)
        .setSmallIcon(android.R.drawable.ic_dialog_info)
```

```
        .setContentText("Odpoveď prijatá")
        .build();

      NotificationManager notificationManager = (NotificationManager)
        getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(notificationId,
        repliedNotification);
    }
  }

  protected void btPosli(View view) {

    String replyLabel = "Napíšte odpoveď";
    RemoteInput remoteInput =
      new RemoteInput.Builder(KEY_REPLY)
      .setLabel(replyLabel)
      .build();

    Intent resultIntent = new Intent(this, MainActivity.class);

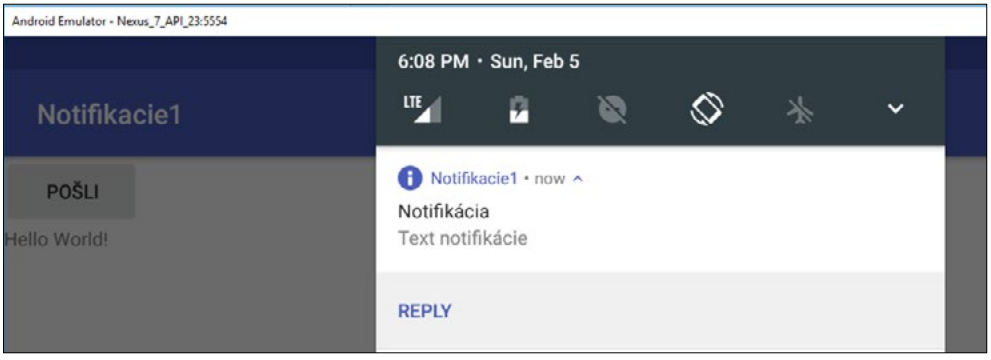
    PendingIntent resultPendingIntent =
      PendingIntent.getActivity(
      this,
      0,
      resultIntent,
      PendingIntent.FLAG_UPDATE_CURRENT
    );

    NotificationCompat.Action replyAction =
      new NotificationCompat.Action.Builder(
      android.R.drawable.ic_dialog_info,
      "Reply", resultPendingIntent)
      .addRemoteInput(remoteInput)
      .build();

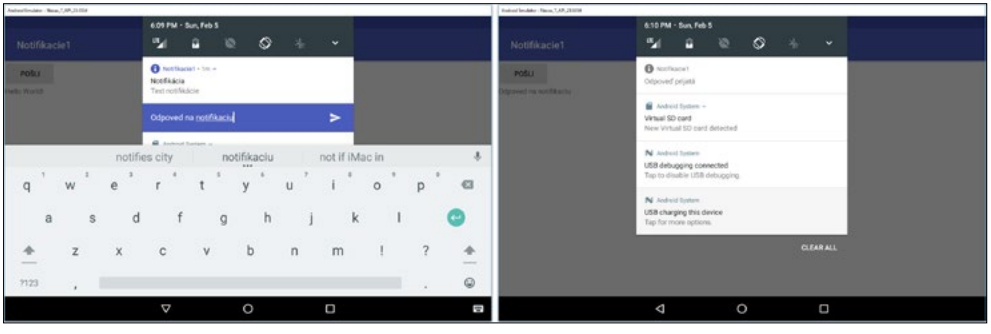
    Notification notif =
      new NotificationCompat.Builder(this)
      .setColor(ContextCompat.getColor(this, R.color.colorPrimary))
      .setSmallIcon(android.R.drawable.ic_dialog_info)
      .setContentTitle("Notifikácia")
      .setContentText("Text notifikácie")
      .addAction(replyAction).build();

    NotificationManager notificationManager =  (NotificationManager)
      getSystemService(Context.NOTIFICATION_SERVICE);

    notificationManager.notify(notificationId, notif);
  }
}
```



Obrázek 20.3: Notifikace umožňující odpověď textovou zprávou



Obrázek 20.4: Aplikace přijala odpověď z notifikace

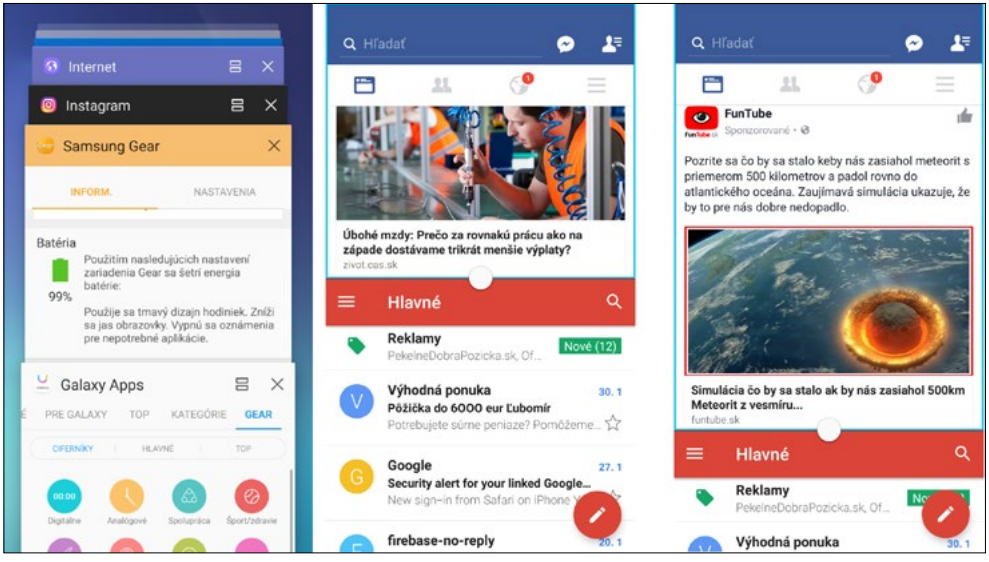
Podpora více oken (Multi-Window)

Jednou z nejzajímavějších funkcí, které přinesl Android 7.0, je podpora více oken. V praxi to znamená, že na obrazovce může být zobrazeno více aktivit současně. Využít se dají tři formy:

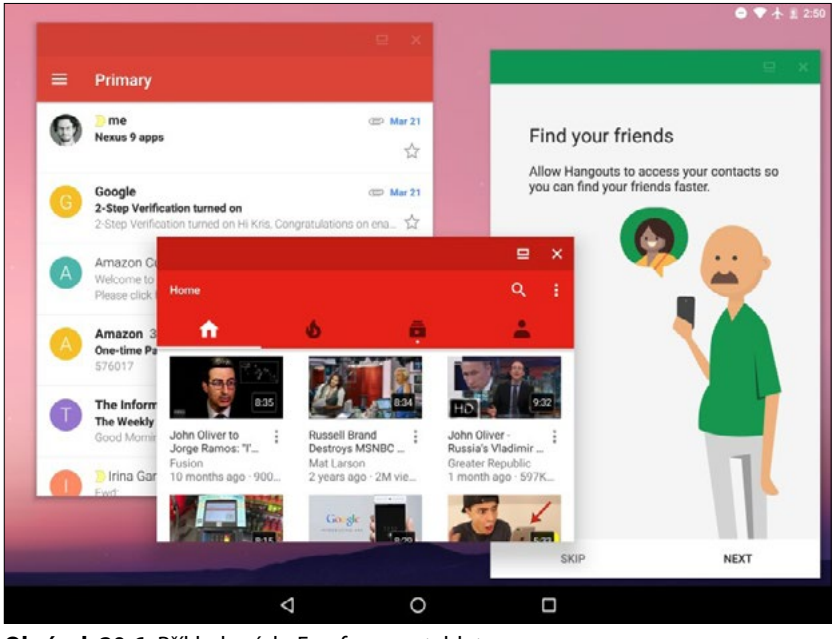
- **Rozdělení obrazovky mezi dvě aktivity** horizontálně nebo vertikálně, poměr plochy určuje uživatel pomocí posouvací lišty, kterou může posouvat vodorovně nebo svisle.
- **Volný režim,** ve kterém může být okno aktivity umístěno kdekoliv. Tento režim má praktický smysl jen na zařízeních s větší úhlopříčkou obrazovky.
- **Obraz v obraze** – nejčastěji se tento režim využívá na přehrávání videa v malém okně, přičemž aktivita druhé aplikace, například e-mailový klient, textový editor, sociální síť a podobně, zabírá celou plochu obrazovky. Okno aktivity s videem překrývá část plochy aktivity, která běží na celé obrazovce.

Z uživatelského hlediska je situace jednoduchá. Režim dvou oken aktivujete delším podržením tlačítka, kterým zobrazíte seznam naposledy spuštěných aplikací. Dvojitým rychlým stisknutím tlačítka provedete další novou funkci – rychlé přepínání mezi aktivní a poslední používanou aplikací. I při normálním stisknutí tlačítka na zobrazení seznamu naposledy spuštěných aplikací se zobrazí seznam a aplikace, které Multi-Window podporují, mají v záhlaví ikonu složenou ze dvou vodorovně nebo svisle uspořádaných tenkých obdélníků. Klepnutím na tuto ikonu

se aplikace zobrazí na polovině obrazovky. Další aplikace, kterou spustíte, pokud podporuje Multi Window, se zobrazí na volné polovině obrazovky. Pomocí lišty mezi okny můžete upravit jejich procentuální podíl plochy displeje.



Obrázek 20.5: Režim Multi-Window z pohledu uživatele. Na obrázku vlevo si v záhlaví některých aplikací všimněte ikony složené ze dvou vodorovně uspořádaných tenkých obdélníků. Tyto aplikace podporují režim Multi-Window

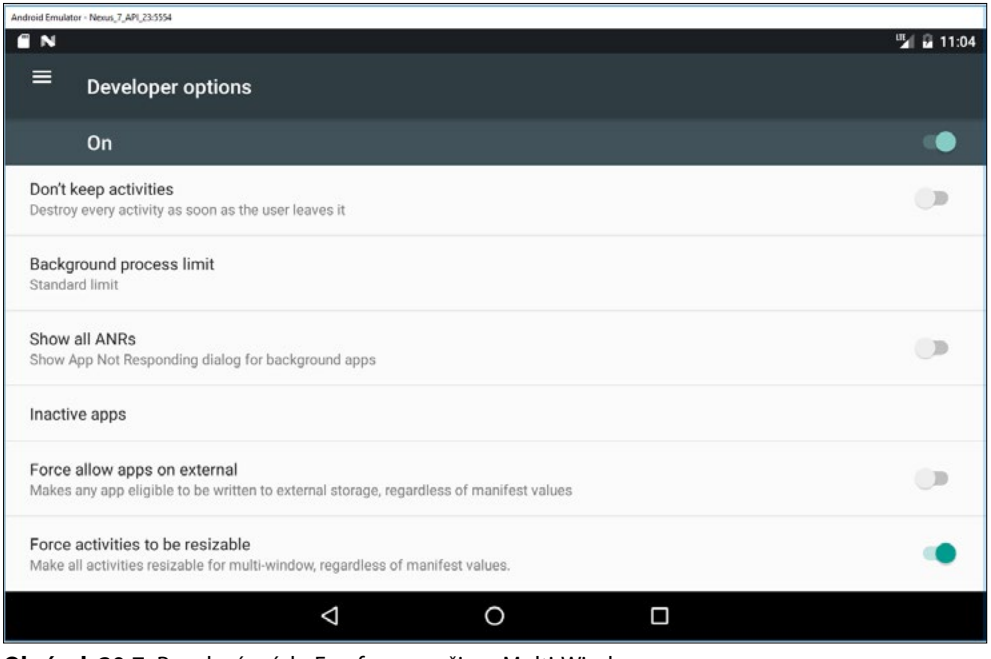


Obrázek 20.6: Příklad módu Freeform na tabletu

Volný (Freeform) režim je v podstatě plnohodnotný režim zobrazení aplikace pro Android v okně, které může mít proměnlivou velikost a může být uzavřené nebo maximalizované stejně jako na vašem PC, Macu, případně Chromebooku. Ne každé zařízení tento režim podporuje. Výrobci větších zařízení se mohou rozhodnout tento režim povolit, takže většinou v předvoleném nastavení není povolený. Google nespecifikoval, pro jaká zařízení tento režim může být povolen, závisí tedy na uvážení výrobce, zda považuje displej svého zařízení za dostatečně velký a jeho rozlišení dostatečné na podporu tohoto režimu. Pod pojmem „velký přístroj" proto může být vnímán 7palcový „phablet", tablet, televizor s Androidem nebo – pokud to výrobce uzná za vhodné – i telefon.

Jak povolit režim Freeform na zařízení nebo emulátoru s Androidem 7.0

Jako první krok musíte zařízení přepnout do vývojářského módu, pokud se už v tomto módu nenachází. Aktivujte **Nastavení** a přejděte na obrazovku, na které se zobrazují informace o operačním systému. Najděte položku **Číslo sestavy** (**Build Number**) a začněte na tuto položku rychle klepat. Musíte klepnout až 7krát. Pokud například po šestém klepnutí přestanete, Android vás povzbudí, že se jedná o seriózní postup, aby se z vás stal vývojář, a zobrazí vám oznámení „Jste pouhé 1 klepnutí od toho, abyste se stali vývojářem". To avizuje, kolikrát je potřeba ještě klepnout. Po posledním klepnutí se zobrazí oznámení „Nyní jste vývojářem" a v **Nastavení** přibude položka **Možnosti pro vývojáře (Developer options)**. Najděte položku **Force activities to be resizable** a povolte tuto možnost.



Obrázek 20.7: Povolení módu Freeform v režimu Multi-Window

UPOZORNĚNÍ

Následně je potřeba zařízení nebo emulátor restartovat. Na reálné zařízení doporučujeme nainstalovat free aplikaci Taskbar (*https://play.google.com/store/apps/details?id=com.farmerbb.taskbar*).

Zda je aplikace, přesněji její aktivita, aktuálně ve módu Freeform, můžete zkontrolovat pomocí kódu:

```
public Boolean jeFreeform() {
  return getPackageManager().hasSystemFeature(
    PackageManager.FEATURE_FREEFORM_WINDOW_MANAGEMENT);
}
```

Životní cyklus aplikace podporující Multi-Window

Režim Multi-Window se uplatní nejen pro dvě vedle sebe nebo pod sebou běžící aktivity aplikací nebo na velkých zařízeních aktivity aplikací v oknech, ale i pro více aktivit jedné aplikace. Pro nás je podstatné, že režim Multi-Window nezmění životní cyklus aktivity.

V režimu více oken může uživatel současně interagovat pouze s jednou aktivitou. Je to logické a funguje to intuitivně, uživatel interaguje s tou aktivitou, na jejíž plochu klepnul. Všechny ostatní aktivity jsou v pozastaveném stavu, i když jsou stále viditelné. Přece jen je tu jeden rozdíl, a to z hlediska priorit pozastavených aplikací. Systém dává viditelným pozastaveným aktivitám vyšší prioritu než aktivitám, které nejsou viditelné. V případě, že uživatel klepne na plochu jedné ze zobrazených, avšak pozastavených aktivit, tato aktivita se obnoví a předtím používaná aktivita se pozastaví.

Správně vytvořené aplikace mohou pracovat i v pozastaveném stavu. Například aplikace na přehrávání videa by měla i nadále pokračovat v jeho přehrávání, aplikace sociální sítě by i v pozastaveném, ale viditelném stavu měla zobrazovat nové statusy a podobně. Tomu je třeba přizpůsobit filozofii aplikace. Například aktivita přehrávající video, pokud zůstane viditelná, by po přechodu do stavu ošetřeného metodou `onPause()` měla přehrávat video. Přehrávání videa by se mělo zastavit až v metodě `OnStop()` a pokračovat v přehrávání v metodě `OnStart()`.

Když uživatel změní režim na Multi-Window, systém upozorní aktivitu na změnu konfigurace. Aktivita se také doví, kdy uživatel změní velikost okna aktivity v režimu Freeform, případně pokud se aktivita přepne zpět do režimu celé obrazovky. Představte si situaci, kdy je aktivita zobrazená na ploše displeje telefonu nebo tabletu na celé obrazovce v režimu na šířku. Po přepnutí do režimu Multi-Window se na obrazovce na šířku zobrazí vedle sebe dvě aktivity. Každá z nich bude mít k dispozici polovičku obrazovky a zároveň bude v režimu na výšku. Funguje to i opačně. Když je tablet v režimu na výšku, po přepnutí do režimu Multi-Window se na obrazovce zobrazí dvě aktivity pod sebou a každá bude v obdélníkovém okně na šířku.

V podstatě má přechod do režimu Multi-Window a zpět do režimu celé obrazovky pro aktivitu stejné důsledky, jako když ji systém upozorní, že se přístroj přepnul z režimu na výšku do režimu na šířku, až na to, že se změní i rozměry okna. Aktivitu můžete navrhnout tak, že bude reagovat na tyto změny, nebo původní aktivitu zrušíte a vytvoříte novou s novými rozměry.

Podporu Multi-Window můžete povolit nebo zakázat v aplikačním manifestu pomocí nastavení parametru:

```
android:resizeableActivity=["true" | "false"]
```

Například:

```
<activity
  android:name=".MainActivity"
  android:resizeableActivity="true"
  android:label="@string/app_name"
  android:theme="@style/AppTheme.NoActionBar">

  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```


Také můžete povolit nebo zakázat režim obraz v obraze (PIP) pomocí nastavení parametru:

```
android:supportsPictureInPicture=["true" | "false"]
```

Pokud je parametr `resizeableActivity` nastavený na hodnotu `false`, povolení režimu obraz v obraze bude ignorováno.

Fungování aktivit aplikace v režimu Multi-Window a módu Freeform můžete definovat pomocí několika parametrů.

```
<activity android:name=".MyActivity">
  <layout android:defaultHeight="500dp"
    android:defaultWidth="600dp"
    android:gravity="top|end"
    android:minHeight="450dp"
    android:minWidth="300dp" />
</activity>
```

Aktivita může zjistit, zda je v módu Multi-Window, pomocí programové konstrukce:

```
if (this.isInMultiWindowMode()) {
  // je v Multi-Window mode
}
else {
  //  nie je v Multi-Window mode
}
```

Příklad aplikace podporující Multi-Window

Námětem příkladu bude aplikace se dvěma aktivitami, které mohou v módu Multi-Window běžet společně na obrazovce zařízení. Připomínáme, že projekt aplikace je nutné vytvořit pro platformu Android 7.0 Nougat, případně vyšší.

Návrhový kód hlavní aktivity obsahuje jen text popisující, o kterou aktivitu se jedná, a tlačítko na spuštění druhé aktivity.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/activity_main"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context="sk.pcrevue.llacko.multiwindow1.MainActivity">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="PRVÁ AKTIVITA" />
  <Button
    android:text="START DRUHEJ AKTIVITY"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/button2"
    android:onClick="posliIntent" />
</LinearLayout>
```

V obsluze události stisknutí tlačítka pomocí záměru spustíme druhou aktivitu. Podtržený kód zabezpečí, aby se druhá aktivita spustila v jiném okně. Pod ním je kód, který v módu Freeform nastaví velikost okna aplikace na 150 × 150.

```
public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void posliIntent(View view) {
    Intent i = new Intent(this, DruhaAktivita.class);

    i.addFlags(Intent.FLAG_ACTIVITY_LAUNCH_ADJACENT|
      Intent.FLAG_ACTIVITY_MULTIPLE_TASK|
      Intent.FLAG_ACTIVITY_NEW_TASK);

    Rect rect = new Rect(0, 0, 150,150);
    ActivityOptions options = ActivityOptions.makeBasic();
    ActivityOptions bounds = options.setLaunchBounds(rect);
    startActivity(i, bounds.toBundle());
  }
}
```

Pomocí místní nabídky **New** → **Activity** → **Empty Activity** složky **App** vytvořte druhou aktivitu, ta bude obsahovat jen prvek `TextView` také s textem popisujícím, o kterou aktivitu se jedná.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/druha_aktivita"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="sk.pcrevue.llacko.multiwindow1.DruhaAktivita">

  <TextView
    android:text="DRUHÁ AKTIVITA"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView2"
    android:textAlignment="center" />
</RelativeLayout>
```


Nezapomeňte v aplikačním manifestu povolit fungování aktivit v módu Multi-Window.

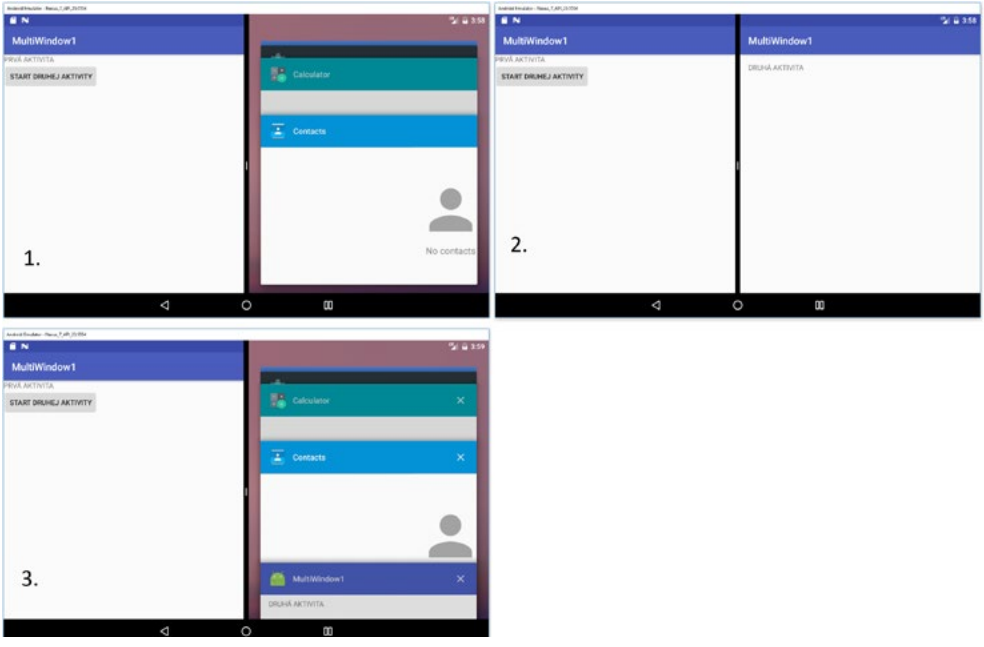
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.multiwindow1">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
      android:name=".MainActivity"
      android:resizeableActivity="true">

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity
      android:name=".DruhaAktivita">
      <layout android:defaultHeight="350dp"
        android:defaultWidth="450dp"
        android:gravity="start|end" />
    </activity>
  </application>
</manifest>
```

Nyní můžete aplikaci poprvé otestovat. Na obrázku jsou tři snímky označené číslem v levém dolním rohu.

Po spuštění aplikace podržte tlačítko ve tvaru sloupce, aby se zařízení přepnulo do módu dvou rozdělených oken. Aktivita aplikace bude v levém okně rozdělené obrazovky. To je stav na snímku 1. Po stisknutí tlačítka spouštějícího druhou aktivitu se tato spustí na pravé polovině obrazovky. To je stav na snímku 2. Po stisknutí tlačítka Multi-Window (nahradilo dočasně tlačítko na přepínání aplikací a má ikonu dvou obdélníků nad sebou) se druhá aktivita ocitne v seznamu aktivit aplikace na pozadí. To je stav na snímku 3.



Obrázek 20.8: Test aplikace v režimu Multi-Window

Pokud máte zařízení podporující Freeform, můžete vyzkoušet i tento mód.

Režim obraz v obraze (Picture-In-Picture)

Mód Picture-In-Picture (PIP) je primárně určený pro televizory s operačním systémem Android a ve verzi Android 7.0 umožňuje sledovat videa v rohu obrazovky při používání jiné aplikace nebo při navigaci mezi programy či mezi aplikacemi. Vaše aplikace může rozhodnout, kdy se spustí režim PIP: například když uživatel přejde zpět z prohlížení videa a začne procházet či vyhledávat jiný obsah; nebo když uživatel sleduje konec videa, tak hlavní obrazovka zobrazuje reklamní či souhrnné informace o dalším videu v pořadí, například o dalším dílu seriálu.

Okno PIP má rozměry 240 × 135 dp a zobrazí se v nejvyšší vrstvě v jednom ze čtyř rohů obrazovky. Uživatel může vyvolat menu PIP, které mu umožňuje přepínat okna PIP na celou obrazovku, nebo zavřít okno PIP podržením tlačítka Domů na dálkovém ovladači. Když se na hlavní obrazovce začne přehrávat jiné video, okno PIP se automaticky uzavře.

Aktivitu, která podporuje obraz v obraze, je nutné deklarovat v manifestu.

```
<activity android:name="VideoActivity"
  android:resizeableActivity="true"
  android:supportsPictureInPicture="true"
  android:configChanges=
    "screenSize|smallestScreenSize|screenLayout|orientation"
  ...
```

Vývoj aplikace pro Android TV

Z hlediska vytvoření projektu je postup vývoje aplikace pro Android TV stejný jako u klasických aplikací pro telefony a tablety. Samozřejmě je tu odlišnost v hardwaru, různí výrobci televizorů mohou mít různé hardwarové komponenty, avšak na úrovni hardware / operační systém musí být aplikováno standardní rozhraní, aby aplikace pro Android TV fungovaly na televizorech všech značek.

Specifika televizoru z hlediska běhu aplikací

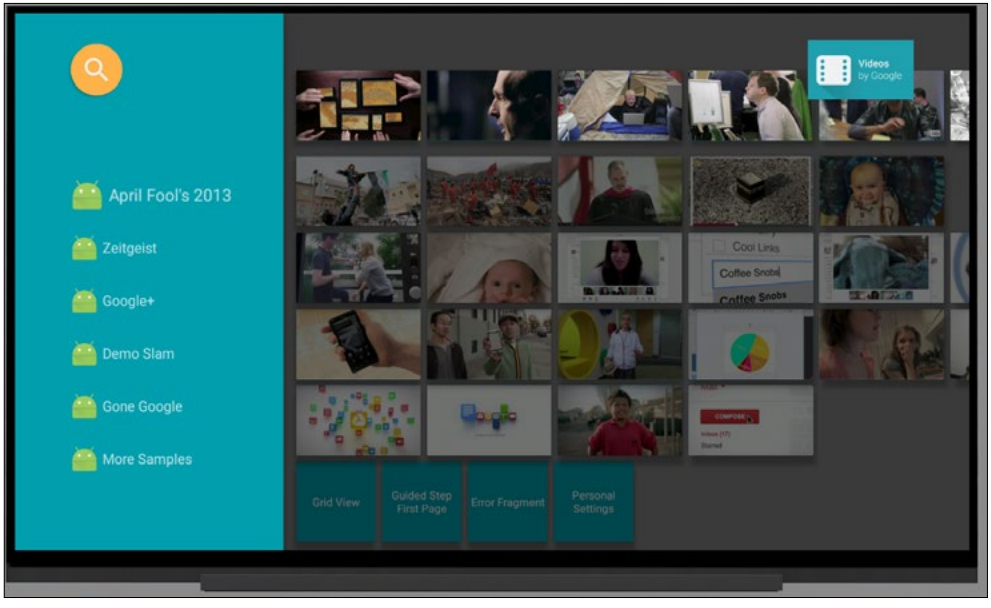
Aplikace musí vždy zohledňovat specifické rysy zařízení, na kterém bude fungovat. Není technicky problém vytvořit pro Android TV například textový editor, ale můžeme vám garantovat, že nebude mít úspěch, protože televizor se využívá k jinému účelu než počítač a nemá v standardní výbavě klávesnici. Zapomeňte na dotykovou obrazovku. Při návrhu uživatelského rozhraní aplikace se musíte soustředit na to, že uživatel má standardně k dispozici na ovládání televizoru dálkové ovládání a některé dražší televizory dokáží snímat gesta uživatele v prostoru pomocí zabudované kamery.



Obrázek 20.9: Typické dálkové ovládání pro Android TV

Při ovládání dálkovým ovladačem může uživatel pohybovat pomyslným kurzorem, případně přenášet fokus z objektu na objekt v osách X a Y pomocí tlačítek v takzvaném kurzorovém kruhu, který bývá kombinovaný s minijoystickem. Samozřejmě se dá pohyb v osách X a Y kombinovat

a pohybovat se kurzorem šikmo, už to ale není tak pohodlné. Ovládací prvky po obvodu obrazovky jsou snáze dosažitelné, protože nejdřív se provede pohyb v jedné a potom v druhé ose. Prvky uprostřed obrazovky jsou obtížněji dosažitelné a umístit na ně kurzor vyžaduje více úsilí. I při navigaci mezi prvky preferujte horizontální a vertikální přesuny a vyhněte se nutnosti šikmých přesunů kurzoru nebo fokusu označování prvků.



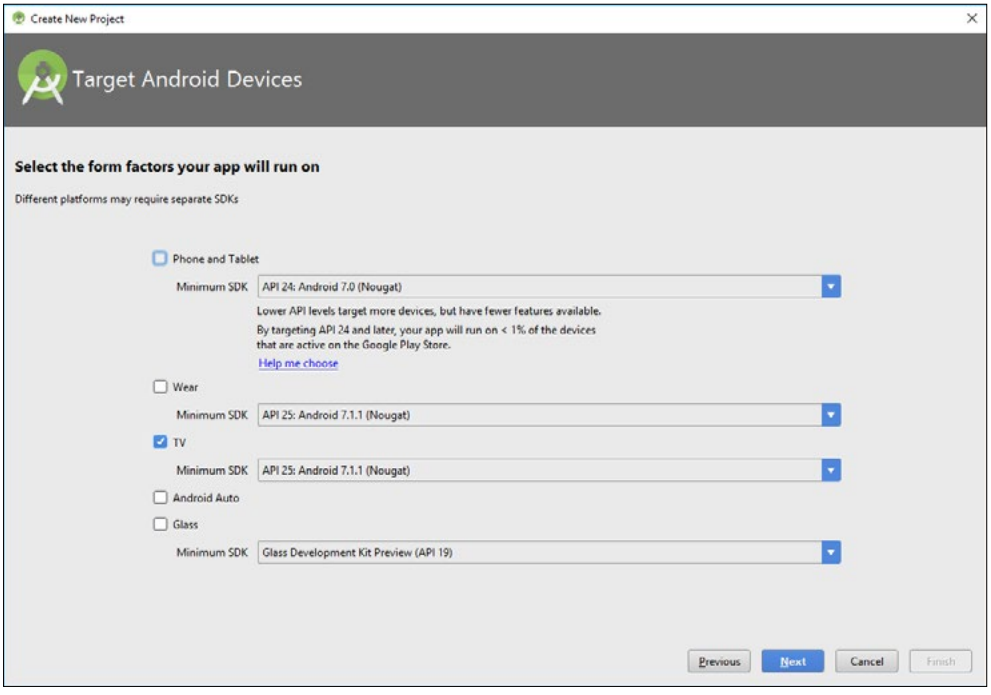
Obrázek 20.10: Předinstalovaná aplikace Videos by Google na emulátoru platformy Android TV

Televizní obrazovka má i další specifika. Většina lidí sedí ve vzdálenosti 2–3 m od televizoru, takže všechny ovládací prvky, obrázky, grafika a texty musí být dobře pozorovatelné, rozlišitelné a čitelné i z této vzdálenosti. Běžné televizory, na rozdíl od monitorů, nedokáží zpracovat extrémně barevné intenzity, proto se vyhněte použití barev vysoké či nízké intenzity, které by mohly splynout a uživatel by je na některém typu televizoru neuměl rozlišit. Také doporučujeme nechat u každého vertikálního okraje 48 a u horizontálního okraje 27 pixelů nevyužitých. Tak máte jistotu, že aplikace bude zobrazovat pouze na takzvané TV safe oblasti. Velké prvky a texty zároveň znamenají, že se jich na obrazovku vejde méně a posuvné lišty se těžko ovládají, proto doporučujeme pro delší seznamy použít dobře dostupné ovládací prvky na stránkování.

Vytvoření projektu aplikace

V našem příkladu jsme pro jednoduchost vytvořili projekt aplikace jen pro Android TV. V mnoha příkladech budete vytvářet projekt univerzální aplikace, která bude mít aktivity navržené pro klasická zařízení, tedy telefon a tablet, i aktivitu pro TV.

Pro tento projekt je k dispozici jen jeden typ hlavní aktivity s názvem **Android TV Activity**.



Obrázek 20.11: Vytvoření projektu pro Android TV

Projekt obsahuje tři aktivity:

- *activity_main.xml*
- *activity_details.xml*
- *playback_controls.xml*

Prohlédněte si pozorně i deklarace v aplikačním manifestu projektu. Aplikace má samozřejmě orientaci obrazovky na šířku (landscape). I toto pravidlo ovšem může mít výjimku, která ho potvrzuje – v některých případech, například pro účely reklamy, případně na dokreslení prezentační scény, může být televizor umístěný i na výšku. Typickým scénářem pro tuto orientaci obrazovky je i informační panel.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="sk.pcrevue.llacko.televizia1">

  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.RECORD_AUDIO" />

  <uses-feature
    android:name="android.hardware.touchscreen"
    android:required="false" />
  <uses-feature
    android:name="android.software.leanback"
    android:required="true" />
```

```
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/Theme.Leanback">
    <activity
      android:name=".MainActivity"
      android:banner="@drawable/app_icon_your_company"
      android:icon="@drawable/app_icon_your_company"
      android:label="@string/app_name"
      android:logo="@drawable/app_icon_your_company"
      android:screenOrientation="landscape">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LEANBACK_LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".DetailsActivity" />
    <activity android:name=".PlaybackOverlayActivity" />
    <activity android:name=".BrowseErrorActivity" />
  </application>

</manifest>
```

Aplikace využívá filter intent typu `LEANBACK_LAUNCHER`. Tento filter identifikuje aplikaci jako povolenou pro televizi, takže po publikování je v aplikačním obchodě Google Play identifikovaná jako televizní aplikace. Pokud filter intent `CATEGORY_LEANBACK_LAUNCHER` není v aplikaci deklarovaný, aplikace nebude pro uživatele přístupující ke Google Play z televizorů viditelná. Jak již bylo zmíněno v části o specifikách Android TV, aktivity vaší aplikace by neměly používat stejné rozložení prvků pro televizor jako v případě telefonů a tabletů. Uživatelské rozhraní televizní aplikace by mělo být jednodušší kvůli snadné navigaci pomocí dálkového ovládání z pohovky.

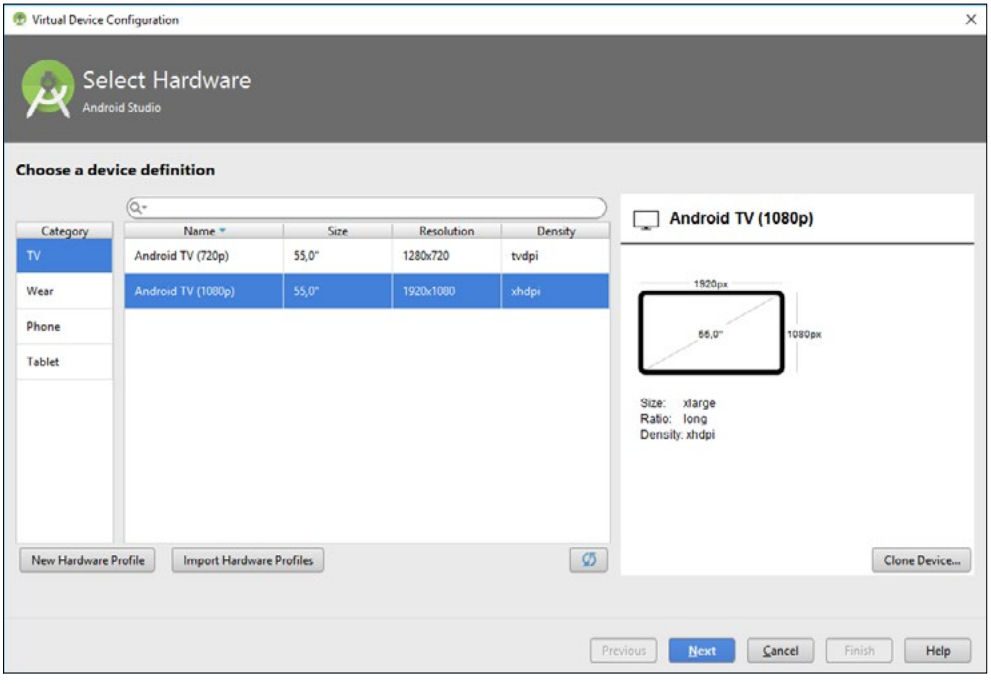
V neposlední řadě je potřeba si uvědomit, že Android TV nemá dotykovou obrazovku, modul telefonu, fotoaparát, mikrofon, Near Field Communications (NFC) ani senzory. Aplikace může deklarovat požadavky na hardware v manifestu, aby se zabezpečilo, že nebude nainstalována na zařízení, které tyto funkce neposkytuje. Když rozšiřujete existující aplikace pro použití na televizoru, pečlivě prostudujte manifest vzhledem k požadavkům na hardware, které by jí mohly zabránit v instalaci na televizor. Pokud vaše aplikace používá hardwarové funkce, například dotykový displej nebo fotoaparát, které nejsou v televizi dostupné, ale aplikace může fungovat bez jejich použití, modifikujte manifest své aplikace, aby deklaroval, že tyto funkce nejsou aplikací požadovány:

```
<uses-feature android:name="android.hardware.touchscreen"
  android:required="false"/>
<uses-feature android:name="android.hardware.faketouch"
  android:required="false"/>
<uses-feature android:name="android.hardware.telephony"
  android:required="false"/>
```

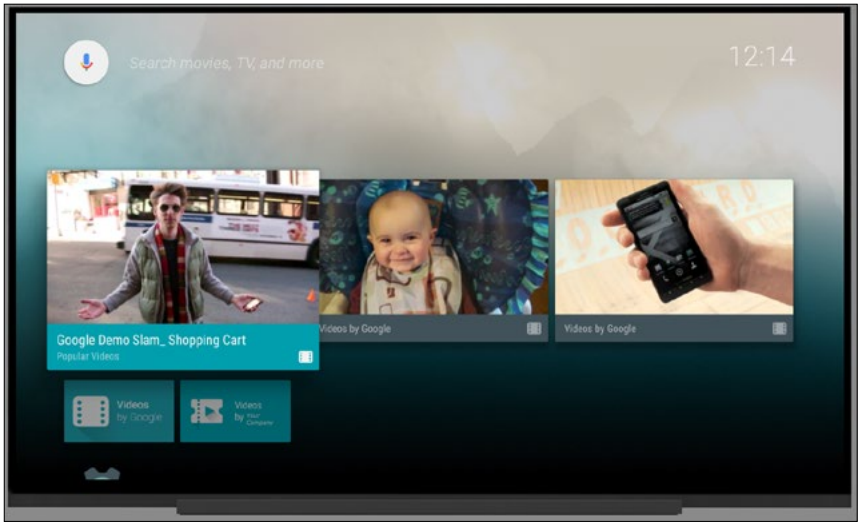
```
<uses-feature android:name="android.hardware.camera"
  android:required="false"/>
<uses-feature android:name="android.hardware.nfc"
  android:required="false"/>
<uses-feature android:name="android.hardware.location.gps"
  android:required="false"/>
<uses-feature android:name="android.hardware.microphone"
  android:required="false"/>
<uses-feature android:name="android.hardware.sensor"
  android:required="false"/>
```


Spuštění aplikace pro Android TV na emulátoru

Na testování a ladění je potřeba vytvořit na vývojářském počítači emulátor platformy Android TV s vhodnou verzí operačního systému. Jelikož jsme chtěli vyzkoušet režim PIP, tedy obraz v obraze, vytvořili jsme emulátor s operačním systémem Android 7.0 Nougat.

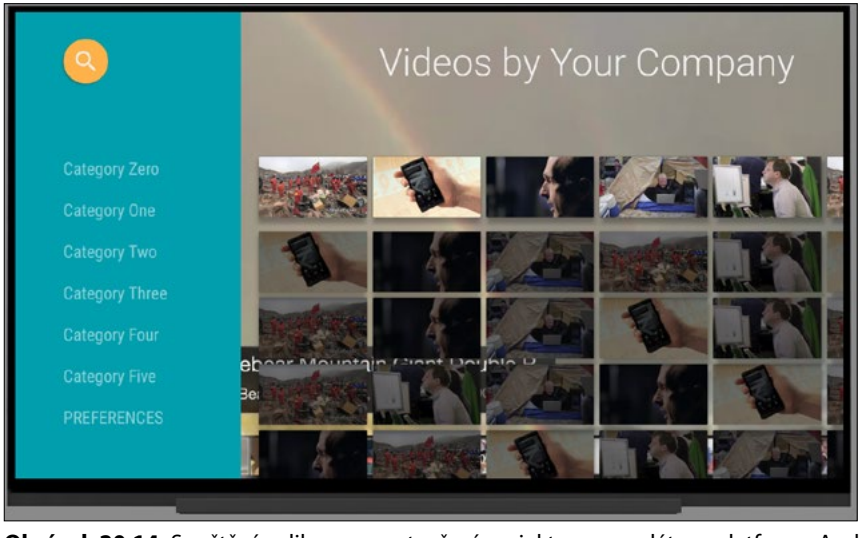


Obrázek 20.12: Vytvoření emulátoru platformy Android TV



Obrázek 20.13: Úvodní obrazovka emulátoru platformy Android TV

Na emulátoru je předinstalovaná jediná aplikace Videos by Google. Doporučujeme prohlédnout si i možnosti jejího nastavení. Při používání emulátoru je potřeba si uvědomit, že televizor nemá dotykovou obrazovku a primární metoda ovládání televizoru, bez ohledu na platformu, je dálkové ovládání. Menšinovým systémem, o kterém se v současnosti nedá říct, zda se časem rozšíří, je interaktivní mód pomocí gest snímaných kamerou na televizoru. U experimentů s emulátorem budete nejčastěji zřejmě používat systémové tlačítko „Zpět" ve tvaru šipky směřující vlevo na panelu emulátoru.



Obrázek 20.14: Spuštění aplikace po vytvoření projektu na emulátoru platformy Android TV

KAPITOLA

21

Publikování aplikací do aplikačního obchodu Google Play

Témata kapitoly:

- **Reklama v mobilních aplikacích**
- **Možnosti a povinnosti vývojáře vůči aplikačnímu obchodu a publikované aplikaci**
- **Registrace vývojářského účtu**
- **Vytvoření stránky vývojáře**
- **Vytvoření balíčku aplikace na publikování**
- **Pravidla publikování aplikace**
- **Publikování aplikace**

Google Play je nejnavštěvovanější obchod s aplikacemi pro Android. Umožňuje uživatelům vyhledat a stahovat aplikace pro tuto platformu. Distribuce aplikací prostřednictvím služby Google Play není výhradní, takže svoje aplikace můžete distribuovat i prostřednictvím dalších kanálů. Nezapomínejte však, že verze zveřejněná v obchodě musí splňovat všechna pravidla služby Google Play. Aplikační obchod Google Play nelze používat k distribuci aplikací, jejichž cílem je zjednodušení distribuce softwarových aplikací a her určených pro zařízení se systémem Android mimo Google Play.

V této kapitole ukážeme nejen postup přípravy aplikace k publikování, vytvoření vývojářského účtu a publikování aplikace, ale budeme se věnovat i možnostem propagace aplikace a různým formám, jak za aplikaci přímo či nepřímo získat peníze.

Reklama v mobilních aplikacích

V aplikačních obchodech pro dominantní mobilní platformy můžete některé aplikace zakoupit, jiné sáhnout zdarma. I vývojáři zdarma distribuovaných aplikací však chtějí za svou práci odměnu, a tak mnohé z těchto aplikací zobrazují reklamu.

Pro aplikace, které jsou na první pohled zdarma, se v praxi kromě profitu ze zobrazovaných reklam využívá více marketingových distribučních modelů. Často používaný je tandem aplikace zdarma + placená aplikace. Účelem aplikace zdarma je formou demo verze propagovat placenou aplikaci.

Podle výsledků miniankety mezi uživateli mobilních zařízení platformy Android různých věkových kategorií chodí do sekce placených aplikací v aplikačních obchodech pravidelně méně než 5 procent oslovených, avšak většina z nich si přibližně jednou za tři měsíce prohlédne i žebříček top v kategorii placených aplikací. Podle praktických zkušeností autorů s tandemem

základní aplikace + aplikace s rozšířenou funkcionalitou si aplikaci s rozšířenou funkcionalitou koupí přibližně jeden ze 150 uživatelů základní verze.

I demo (či „light“) aplikace však v poslední době začínají zobrazovat reklamu, čímž odrazují potenciální zájemce. Velmi populární je model „freemium“, který využívá nákupy v aplikaci. Základní funkce jsou zdarma, pokročilejší si uživatel musí dokoupit. Další z možností je apelovat na snahu spokojených uživatelů nějak se zavděčit. Proto někteří vývojáři ve vhodný okamžik, nejčastěji společně s výzvou na ohodnocení aplikace, zobrazí i číslo svého účtu. Jiní vývojáři publikují placenou aplikaci za nejnižší možnou cenu, která má jen symbolickou funkčnost a jejímž účelem je, aby si ji spokojení uživatelé jiných aplikací od stejného vývojáře koupili a takto ho podpořili.

Formáty reklamy v mobilních aplikacích

V mobilních aplikacích se můžete setkat hned s několika formáty reklamy:

- **Multimediální reklamy** (rich media ads) naplno využívají vlastnosti moderních zařízení, co se týká prezentace multimédií a interakce pomocí senzorů. Nejčastěji se tento typ reklamy zobrazuje při spuštění nebo ukončení aplikace.
- **Vsunutá reklama** – nejčastěji se využívá fullscreen reklama formátu pop-up, která se zobrazí přes celou obrazovku. V praxi nabízí vyšší míry prokliknutí než banner. I zde platí, že méně je často více. Je třeba být opatrný, pokud jde o frekvenci zobrazování tohoto formátu reklamy. Vsunutá reklama je populární především v mobilních hrách, kde by měla být uvedena na strategických místech, která nenarušují uživatelskou zkušenost. Často se jako vsunutá reklama využívá video. Standardně trvá 15 nebo 30 sekund s možností přeskočit po 5 sekundách, pokud přesahuje 15 sekund v závislosti na SDK reklamní sítě.
- **Banner** – reklamní proužek – se do aplikace zakomponuje nejsnáze a zobrazí se v horní nebo dolní části obrazovky. V tomto případě však naplno platí, že za málo námahy bude málo profitu. Bannery jsou všeobecně vnímané uživateli jako obtíž a míra prokliků je nižší.
- **Nativní reklama** – reklama tohoto typu je zakomponovaná do aplikace tak, aby nerušila uživatelskou zkušenost a působila jako integrální součást aplikace (například se do aplikace přidá tlačítko na zobrazení reklamy).
- **Uvítací plocha** – přes celou obrazovku vítá uživatele v aplikaci a propaguje nebo doporučuje aplikaci podobné té, kterou uživatel právě spustil.

Odhad profitu z reklamy v aplikacích

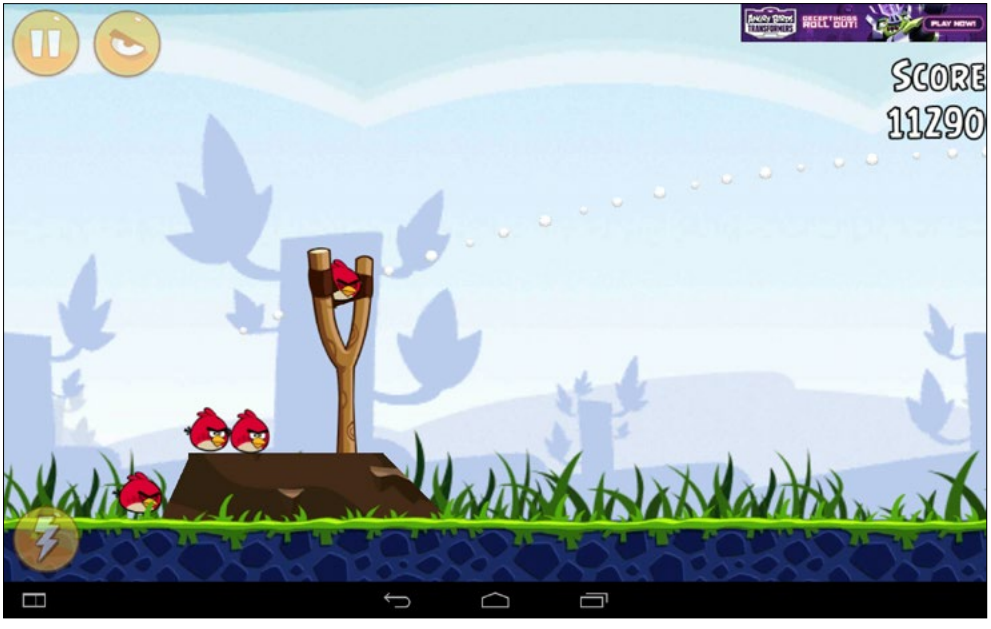
Pro aplikace zobrazující reklamy je charakteristický vysoký počet stažení. Tuto metriku na odhad příjmu z reklamy nelze použít jako klíčovou. Pokud je zobrazování reklam otravné, uživatelé aplikaci přestanou používat, případně ji hned po prvním spuštění odinstalují. Proto je potřeba najít vyhovující kompromis, jelikož pro dodavatele aplikace profitujícího ze zobrazování reklam generují příjmy jen aktivní uživatelé. Ve všeobecnosti platí, že reklamy mají smysl, pokud aplikace zobrazí aspoň milion reklam měsíčně. To znamená předpokládaný příjem 200–300 USD.

Jednoduchou matematikou se dopracujete ke zjištění, že pokud každý aktivní uživatel spustí aplikaci jednou denně a tato mu zobrazí pět reklam, musí si aplikaci stáhnout aspoň 7000 lidí. Pokud aplikace zobrazí jen jednu reklamu, potřebuje minimálně 33 tisíc stažení. Relevantnost

našich výpočtů potvrzují i názory vývojářů prezentované na různých fórech, podle kterých 10 000 aktivních uživatelů měsíčně vygeneruje příjem z reklamy přibližně 700 USD. Pokud se aplikace prodává za 1,5 USD, stačí k zisku stejné sumy 300 stažení, jelikož nějakou část, zpravidla 30 procent, si stáhne aplikační obchod.

POZNÁMKA

Pro zajímavost: Aplikace Angry Birds, která má jen na Androidu více než 100 milionů stažení, generovala na začátku roku 2013 denně přibližně 15 000 USD za zobrazování reklamy.



Obrázek 21.1: Příklad decentního zobrazování reklam v jedné z nejúspěšnějších her pro mobilní platformy

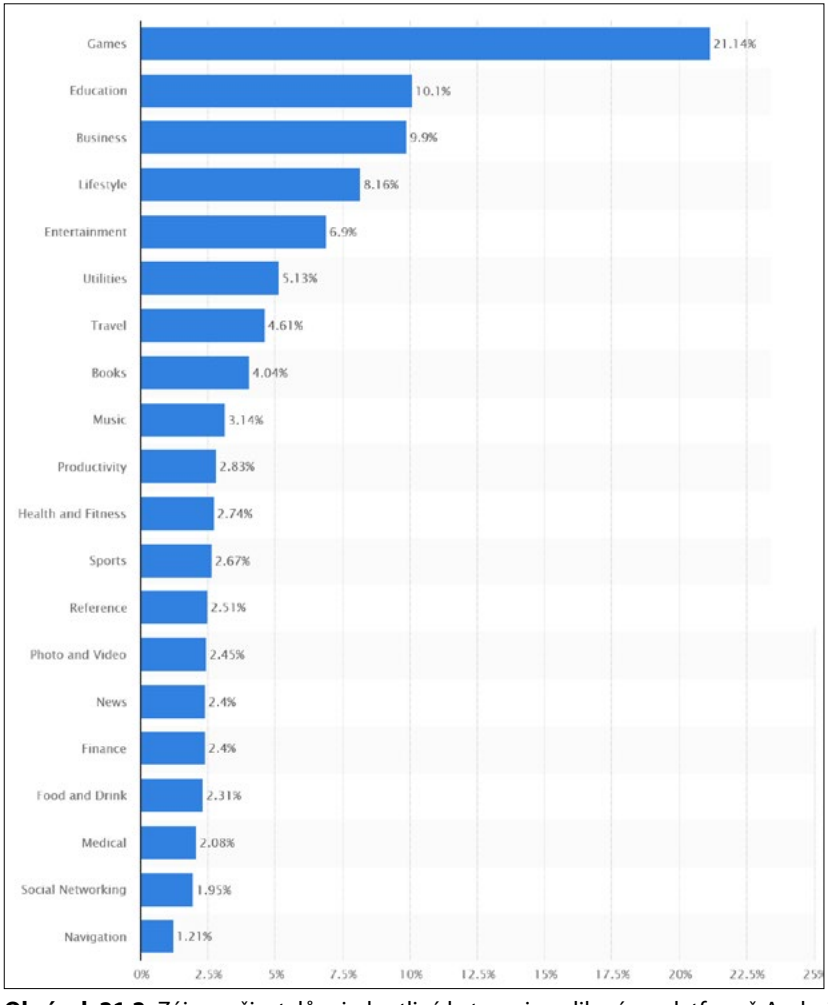
Specifika a cílení reklamy

Aby mohl vývojář profitovat z reklam v mobilních aplikacích, musí mít na paměti specifika tohoto inzertního kanálu, především z hlediska cílení reklamy a možností zobrazování na malých obrazovkách telefonů. Reklamu je možné zobrazit jen na určitý čas, aby uživatele zaujala, ale nerušila a nezpůsobila takzvanou „bannerovou slepotu". Díky masovému rozšíření využívá telefony a tablety prakticky každý, od malých dětí přes teenagery, studenty a manažery až po seniory. Proto mají cílené reklamy význam jen u aplikací se specifickým námětem určených pro užší okruh uživatelů.

Při cílení reklamy nejvíce pomáhá přesné specifikování klíčových slov, podle kterých může uživatel aplikaci v obchodě vyhledat. Okruh uživatelů vymezuje i platforma či formát zařízení. iPhony a iPady využívají více manažeři než teenageři. Na tablety a telefony by se měla aplikovat jiná filozofie zobrazování reklamy. Aplikaci v telefonu s výjimkou her má uživatel spuštěnou zpravidla několik desítek sekund, dokud nezjistí relevantní informaci. Aplikace na tabletech bývají obvykle spuštěny delší čas.

Velkou výhodou mobilních zařízení z hlediska cílení reklamy je možnost určení polohy, kde se majitel zařízení momentálně nachází. Kromě zacílení podle věku, pohlaví a konkrétních aplikací, ve kterých se mohou zobrazovat reklamy, má inzerent k dispozici několik dalších způsobů zacílení:

- Kategorie mobilních aplikací: zobrazují se reklamy v aplikacích spadajících do určitých kategorií, například hry, nástroje, zprávy a počasí, sport, zdraví, životní styl...
- Kategorie nainstalovaných aplikací: v případě tohoto typu zacílení může inzerent aktivovat zobrazování reklam pouze těm, kteří si nainstalovali aspoň jednu placenou aplikaci ze zacílené kategorie.
- Nové mobilní zařízení: zobrazování reklamy lidem, kteří si nedávno aktivovali telefon a mohli by se zajímat o přidání nových aplikací.



Obrázek 21.2: Zájem uživatelů o jednotlivé kategorie aplikací na platformě Android. Zdroj statista.com

Modely platby od inzerentů

Modely a ceny za reklamu jsou v různých inzertních systémech různé. Většina z nich využívá více různých modelů:

- **Cena za zobrazení:** model, u kterého inzerent platí za zobrazení reklamy, je ve službě Google AdWords označován zkratkou CTZ (cena za tisíc zobrazení). Microsoft Advertising tento model označuje CPM (Cost Per Mille). Tato metoda se inzerentovi vyplatí, hlavně pokud chce zvýšit povědomí o své aplikaci.
- **Cena za klepnutí**: pokud inzerent platí za klepnutí na reklamu, Google AdWords tento model označuje CZK a Microsoft Advertising CPC (Cost Per Click). Pro inzerenta je tento model výhodný například tehdy, pokud chce zvýšit návštěvnost svých webových stránek.
- **Cena za realizovanou transakci:** u modelu označovaného Google CZA, tj. cena za akvizici, případně u Microsoftu CPA (Cost Per Action) zaplatí inzerent až v případě, že uživatel aplikace dokončí specifickou akci, například se zaregistruje k inzerované službě nebo si objedná inzerované zboží. Metoda CZA je výhodná pro inzerenty se sezónními kampaněmi v příslušné službě.

Většina inzerentů začíná nejdříve se základní strategií nabízení CZK, což znamená, že náklady se odvíjejí na základě počtu klepnutí, která získají jejich reklamy. Suma účtovaná za klepnutí závisí částečně na maximální nabídce ceny za klepnutí, kterou nastaví při zadávání kampaně. V konečném důsledku zaplatí méně, maximálně tolik, kolik je minimálně potřeba na udržení požadované pozice jejich reklamy. Z hlediska cílení reklamy bude model platby za zobrazení reklamy CPM výhodný pro masově využívané aplikace. Naproti tomu platba za akci u modelu CPA, která je neporovnatelně vyšší než u modelů CPM a CPC, bude vhodnější pro specializované aplikace.

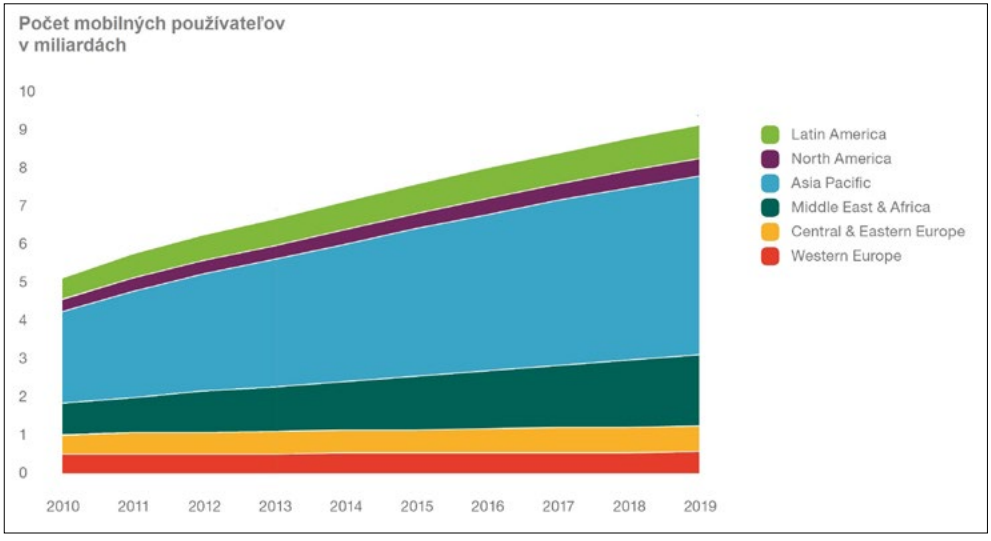
Kritéria úspěšnosti aplikace

Aplikační obchody umožňují vývojáři sledovat počty stažení a nákupů jeho aplikace včetně demografických charakteristik. Může si nastavit filtr podle země, věku a podobně. Na základě těchto údajů dokáže vývojář přesněji identifikovat potřeby lokalizace aplikace do dalších jazyků, případně zaměřit propagační aktivity na konkrétní zemi, věkovou skupinu a podobně.

Aby vývojář dosáhl signifikantních příjmů z reklamy v aplikaci, musí být aplikace koncipovaná jako mezinárodní, to znamená, že by měla mít uživatelské rozhraní minimálně v angličtině. Lepší příjmy dosáhne, pokud bude mít aplikace i španělskou, německou a francouzskou verzi. A jak dosáhne lepších příjmů? Porovnejte si graf statistiky stahování aplikace s anglickým uživatelským rozhraním s grafem počtu uživatelů mobilních platforem podle geografického hlediska. Kdyby měla aplikace i čínské a japonské uživatelské rozhraní, počet stažení by byl mnohem větší.

Důležitým kritériem jsou i tržní podíly mobilních platforem. U Androidu je dobrou zprávou pro vývojáře, že tato platforma jednoznačně vede tržním podílem 84,7 %. Horší zprávou je, že aplikační obchod obsahuje více než 1,3 milionu aplikací, ze kterých je přibližně milion bezplatných. Aplikace, ve které chce vývojář profitovat z reklamy, se musí v této konkurenci prosadit a zaujmout uživatele natolik, aby si ji stáhli a používali. Tržní podíl iOS je 11,7 % uživatelů, avšak vzhledem k ceně iPhonů a iPadů využívá tyto přístroje specifická skupina lidí. Prosadit se s aplikací je ještě těžší než u Androidu, protože i přes podstatně nižší tržní podíl platformy je v aplikačním obchodě 1,2 milionu aplikací. Windows jako nejmladší platforma má jen 3%

tržní podíl a v aplikačním obchodě je přibližně 300 000 aplikací. Osud této platformy je navíc v době dokončování rukopisu nejasný.



Obrázek 21.3: Počet uživatelů mobilních platforem podle geografického hlediska

Google AdWords

Reklamní systém Google AdWords umožňuje vedle reklam na webových stránkách na základě vyhledávání zobrazovat i reklamu v mobilních aplikacích. Služba AdWords dokáže dynamicky vyladit reklamy, aby vzhledem k možnostem zařízení zobrazovaly relevantnější informace. Inzerent si může nakonfigurovat, aby služba přiřazovala jeho reklamu k aplikacím prostřednictvím obsahové sítě, nebo zacílit reklamu na konkrétní kategorie aplikací. Z hlediska inzerentů funguje služba na principu dobíjení kreditu. Každý zadavatel reklamy si může nastavit svůj denní rozpočet – když se vyčerpá, inzerát se v daný den přestane zobrazovat. Vývojáři přidávají zobrazování reklamy přes Google Mobile Ads SDK

Mobilní aplikace šířené prostřednictvím aplikačních obchodů mohou zobrazovat reklamy, ale musí poskytovat více (funkcionality, přínosu…) než jen zobrazení reklamy či zobrazení obsahu webové stránky. Aby reklamy uživatele příliš nevyrušovaly, zavedli provozovatelé aplikačních obchodů Apple, Microsoft i Google více nebo méně přísná pravidla, jak se mohou reklamy zobrazovat, aby aplikace prošla schvalovacím procesem. AirPush reklamy na notifikační liště Google povolil jen na propagaci nákupů přímo v aplikaci. Aplikace nesmí měnit okolí, tedy domovskou obrazovku, widgety či záložky v prohlížeči. Aplikace s výjimkou antimalwarových nesmí uživatele informovat o možnosti nahrazení aplikací třetích stran za jiné, kromě bezpečnostních aplikací, na které se toto pravidlo nevztahuje.

Obzvlášť kontroverzním tématem jsou aplikace určené dětem. Mnozí psychologové upozorňují, že reklamy příliš ovlivňují děti, a iniciují aktivity vedoucí k zákazu takovýchto reklam. Proti zrušení zobrazování reklam v aplikacích určených dětem jsou hlavně vývojáři, jelikož jde o důležitý zdroj jejich příjmů. Proto prosazují názor, že za svoje děti jsou zodpovědní zejména rodiče, a proto by žádné negativní ovlivňování reklamami nemělo přicházet v úvahu.

Zviditelněte se ve vyhledávání Google Play

Vyhledávání ve službě Google Play je klíčovým nástrojem, pomocí kterého mohou uživatelé hledat relevantní a oblíbené aplikace pro zařízení s Androidem. Aby uživatelé ve službě Google Play mohli najít vaše aplikace, je důležité vyplnit všechny informace o aplikaci a optimalizovat záznam v obchodě.

Text záznamu v obchodě je pro viditelnost vaší aplikace rozhodující. Při optimalizaci záznamu v obchodě Google Play se můžete řídit následujícími tipy:

- **Název** musí být jedinečný a přístupný. Vyhněte se běžným výrazům a pokuste se vystihnout, o co se ve vaší aplikaci jedná. Název by měl být stručný a jasný. V některých zařízeních, v nichž uživatelé prohlížejí internet, se mohou delší názvy zkrátit. V názvech nepoužívejte drobné zkomoleniny běžných slov – uživatelé by mohli hledané výrazy zadat ve správném tvaru a vaši aplikaci by nenašli.
- **Popis** – zaměřte se na uživatele a na to, co jim vaše aplikace může nabídnout. Popis aplikace si prohlédněte v aplikaci Google Play a v internetovém obchodě a zkontrolujte, zda se zásadní části textu zobrazují nad záhybem stránky. V popisu využívejte doporučené postupy SEO, ale nezapomeňte dodržovat Pravidla pro obsah ve službě Google Play týkající se nevyžádané pošty a zneužití IP adres (například spamování klíčových slov, odcizení identity atd.).
- **Grafická a obrazové díla** – díky ikonám, obrázkům a snímkům obrazovky může vaše aplikace ve výsledcích vyhledávání, kategoriích a vybraných aplikacích lépe vyniknout. Do záznamu v obchodě sice není nutné nahrávat všechny grafické položky, avšak doporučujeme přidat kvalitní snímky obrazovky, které aplikaci předvedou na podporovaných platformách (telefon, sedmipalcový tablet, desetipalcový tablet).
- **Lokalizace záznamu v obchodě.** V případě záznamů v obchodě, které pro aplikaci přímo nedefinujete, nabízí Google automatický strojový překlad. Profesionální překlad vašeho popisu může vést k lepším výsledkům vyhledávání a lepší viditelnosti vaší aplikace pro uživatele z celého světa. V Konzoli pro vývojáře můžete zakoupit překlady hotových textů překladatelem od dodavatele třetí strany.

Vyhledávání ve službě Google Play zohledňuje celkový dojem z vaší aplikace na základě chování uživatelů a zpětné vazby. Celkové hodnocení aplikace je založeno na kombinaci hodnocení, recenzí, počtu stažení a dalších faktorech. Podrobnosti o konkrétních vahách a hodnotách jsou součástí vyhledávacího algoritmu ve vlastnictví společnosti Google, viditelnost své aplikace ale můžete zlepšit následovně:

- Vytvořte trvalé a smysluplné uživatelské prostředí.
- Aplikaci udržujte a zlepšujte prostřednictvím pravidelných aktualizací.
- Nabádejte uživatele k poskytnutí zpětné vazby v podobě hodnocení a komentářů.
- Poskytujte co nejlepší podporu zákazníka – odpovídejte uživatelům a pomáhejte jim řešit problémy.

Možnosti a povinnosti vývojáře vůči aplikačnímu obchodu a publikované aplikaci

Aplikace můžete publikovat zdarma nebo za poplatek. Uživateli nesmíte začít účtovat poplatek za aplikaci, která byla původně zadarmo. Je to možné pouze v případě, že poplatek souvisí s alternativní verzí aplikace. Jako vývojář odpovídáte za podporu své aplikace. Kupující mají pokyny kontaktovat vývojáře v souvislosti s jakýmikoliv chybami nebo problémy s výkonností aplikací, které byly staženy a nainstalovány ze služby Google Play. Vy budete nést výhradní zodpovědnost, a společnost Google nebude mít žádnou zodpovědnost za poskytování nebo řešení podpory a údržby vaší aplikace.

Musíte uvést a udržovat platné a přesné kontaktní informace, které budou zobrazeny na každé stránce s podrobnostmi o aplikaci v Obchodě a zpřístupněny všem uživatelům pro účely podpory zákazníka a právní účely. V případě placených aplikací nebo nákupů v aplikacích musíte reagovat na dotazy podpory zákazníka do tří pracovních dní. Na záležitosti týkající se podpory aplikace, které byly označeny společností Google jako urgentní, musíte reagovat do 24 hodin.

UPOZORNĚNÍ

Nedostatečné informace nebo nedostatečná podpora vašich aplikací může mít za následek nízké hodnocení produktů, jejich méně výhodné umístění, nízký objem prodeje, fakturační spory nebo odstranění z aplikačního obchodu.

V případě placených aplikací opravňujete společnost Google vrátit kupujícímu celou platbu ve výši ceny produktu nebo transakce v aplikaci vaším jménem, pokud kupující požádá o vrácení platby do 48 hodin od nákupu. Ve všech dalších ohledech budou platit standardní smluvní podmínky.

Registrace vývojářského účtu

Nejprve se musíte zaregistrovat jako vývojář na adrese *https://play.google.com/apps/publish/signup/*. Poplatek za registraci je 25 dolarů. Aplikaci můžete šířit zdarma nebo za vámi stanovenou cenu. V druhém případě si Google vezme 30 procent, takže pokud budete aplikaci prodávat za dolar, dostanete 70 centů. Abychom to sumarizovali, je to startup jako z učebnice. Relativně nízký registrační poplatek může být v případě úspěšné aplikace základem dobrého byznysu.

Přihlaste se k účtu Google, který bude vystupovat v úloze vlastníka účtu vývojáře. Následně musíte vyjádřit souhlas s Distribuční smlouvou pro vývojáře služby Google Play.

UPOZORNĚNÍ

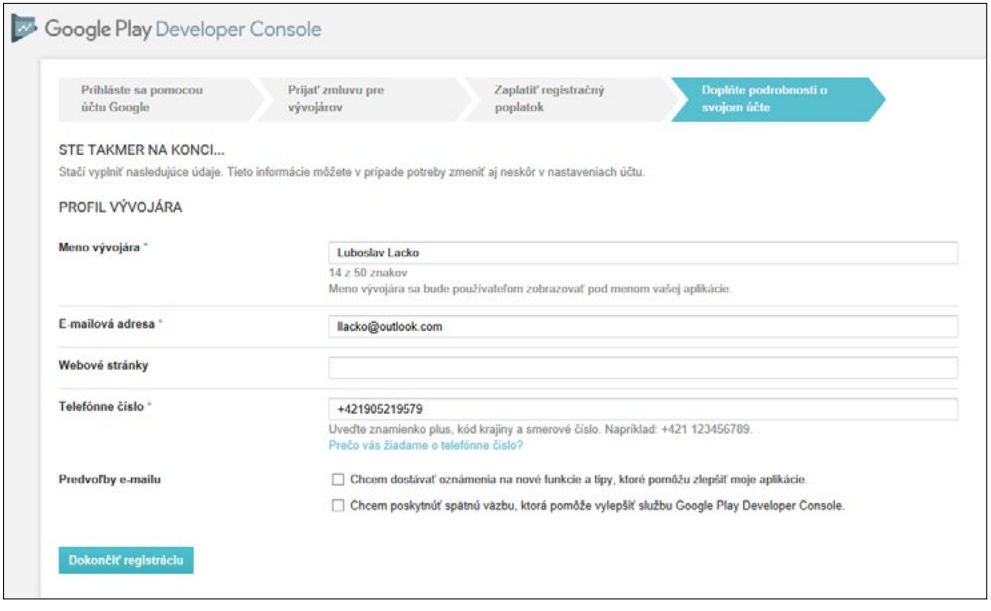
Pokud váš účet v minulosti tuto smlouvu porušil, registrace vývojáře do služby Google Play nebude možná.

Pokud chcete stanovit ceny svých aplikací, musíte mít účet služby Google Wallet Merchant Center. Služba Google Play nabízí několik způsobů zpeněžení aplikace:

- **Reklamy v aplikaci** – stažení aplikace je bezplatné, aplikace obsahuje reklamy v aplikaci
- **Prodej aplikace** – cena aplikace je známá předem, účtuje se před tím, než ji uživatel stáhne
- **Nákupy v aplikaci** – v aplikaci lze zakoupit obsah, například další úrovně hry a podobně
- **Odběry** – v aplikaci se bude nacházet obsah, který bude dostupný po uhrazení pravidelného předplatného

UPOZORNĚNÍ

Cenu aplikace můžete kdykoliv upravit. Upozorňujeme však, že k aplikaci, kterou zveřejníte jako bezplatnou, nebudete moct později přidat cenu. Budete muset vytvořit novou aplikaci a cenu přidat k ní.



Obrázek 21.4: Registrace vývojářského účtu

Vytvoření stránky vývojáře

V zájmu propagace své značky a aplikací v Google Play si můžete vytvořit stránku vývojáře. Pokud chcete vytvořit stránku vývojáře, musíte mít ve službě Google Play aspoň jednu zveřejněnou aplikaci. Když si vytvoříte stránku vývojáře, uživatelé ji budou moct navštívit v Obchodě Play. Google vám přiřadí webovou adresu, pomocí které budete moct svou stránku sdílet. Uživatelé mohou na stránce vývojáře získat další informace o vaší značce, zjistit informace o aplikaci, kterou přednostně vyberete, a najít další aplikace, které jste zveřejnili v Google Play.

Stránky vývojáře nejsou povinné. Pokud si stránku vývojáře nevytvoříte, další aplikace, které jste zveřejnili, se budou nadále zobrazovat ve vašich sbírkách aplikací konkrétního vývojáře ve službě Google Play.

Vytvoření balíčku aplikace na publikování

UPOZORNĚNÍ

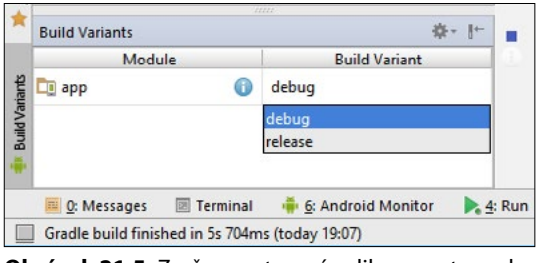
Aplikaci před publikováním důkladně otestujte.

Aby bylo možné aplikaci šířit přes aplikační obchod, je potřeba splnit několik základních podmínek. Aplikace musí mít definované atributy `android:versionCode` a `android:versionName` v elementu `<manifest>` souboru *manifest*. První z atributů se používá na interní identifikaci aplikace a druhý na identifikaci aktuální verze. Zároveň musí být v manifestu definované i atributy `android:icon` a `android:label`. Také je potřeba vytvořit ikony a ostatní grafické prvky v rozlišeních, které bude aplikace podporovat. Až po ukončení těchto přípravných fází můžete zkompilovanou aplikaci zabalit, podepsat a zveřejnit.

UPOZORNĚNÍ

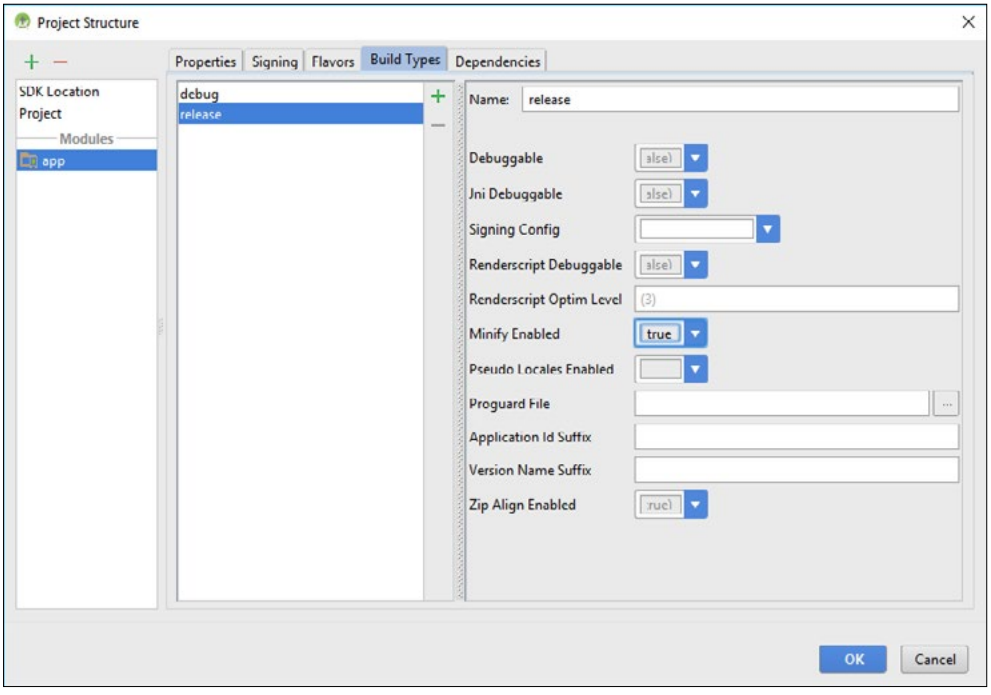
Aplikaci je před finální kompilací a sestavením potřeba podepsat vývojářským privátním klíčem.

Ve vývojovém prostředí Android Studio je potřeba nejprve změnit sestavování aplikace, aby se do finální verze určené k publikování nepřidávaly informace potřebné k ladění. V levém dolním rohu okna Android Studio klepněte na položku **BuildVariants** a ve sloupci **Build Variant** změňte **debug** na **release**.



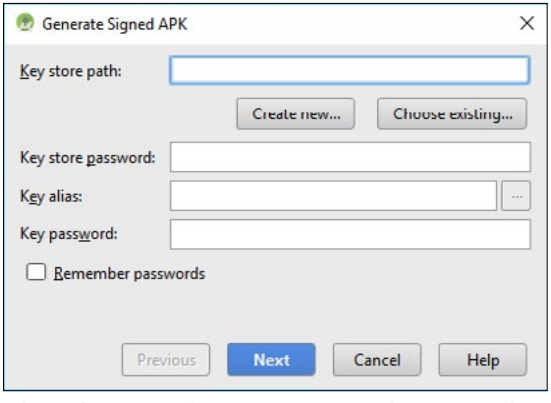
Obrázek 21.5: Změna sestavení aplikace na typ release vhodný k publikování

Dalším krokem je vytvoření klíče k podepsání aplikace. Pomocí hlavní nabídky Android Studia **File → Project Structure** zobrazte dialog **Project Structure**. Na levém panelu zobrazte položku **app**. Na kartě **Build Types** přepněte na položku **release**. Volbu **Minify Enabled** nastavte na hodnotu **true**.



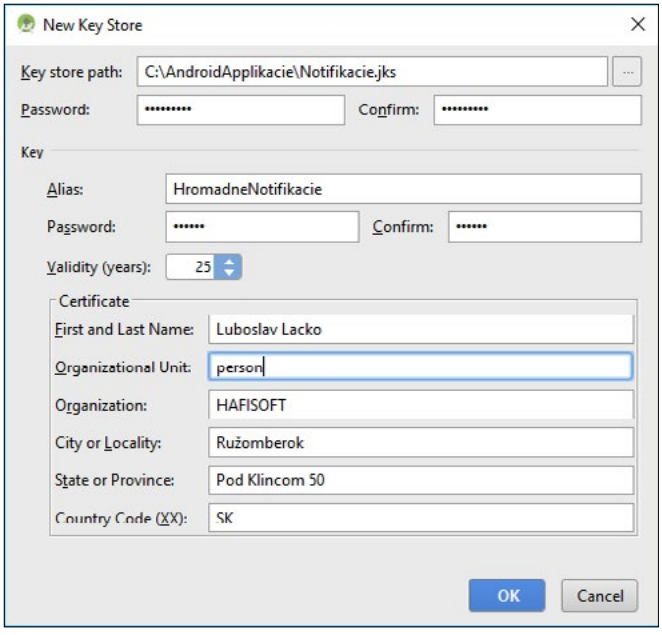
Obrázek 21.6: Dialog Project Structure

Pokračujeme vytvořením souboru *keystroke*. V nabídce Android Studia **Build → Generate Signed APK** zobrazte dialog **Generate Signed APK Wizard**.



Obrázek 21.7: Dialog Generate Signed APK Wizard

Pokud nemáte vytvořený klíč, označte v dialogu položku **Create new** a vyberte cílové umístění souboru s klíči. Dialogové okno **New Key Store** je rozdělené do dvou částí. Horní část se vztahuje k souboru úložiště klíčů. V této sekci zadejte silné heslo, se kterým chrání soubor paměť. Spodní část dialogu se vztahuje k uvolnění klíče uloženého v klíčovém souboru úložiště.



Obrázek 21.8: Dialog New Key Store

Soubor si dobře uložte. Parametr **Validity** nastavte aspoň na 25 let a do položky **First and Last Name** napište svoje jméno a příjmení. Po potvrzení údajů se vytvoří soubor s klíčem s příponou JKS a soubor aplikace s příponou APK.

Pravidla publikování aplikace

Aplikace i její záznam v obchodě projdou po nahrání několika kontrolami a ověřeními, které mají odhalit, zda jste splnili všechny požadavky na zveřejnění. Soubor APK bude po nahrání ve stavu Koncept, a to až do té doby, než budete připraveni na jeho zveřejnění. Pokud jste nesplnili požadavky na zveřejnění, klepněte na tlačítko **Proč nemohu zveřejnit aplikaci?** Takto můžete zobrazit seznam požadavků, které vaše aplikace nesplňuje.

Google nepovoluje aplikace se zavádějícími, nerelevantními, nadměrnými nebo nevhodnými metadaty, která zahrnují popis aplikace, název, ikonu, snímky obrazovky a propagační obrázky. V popisech aplikace také nepovoluje doporučení uživatelů, problémem je i příliš mnoho podrobností, zavádějící odkazy na jiné aplikace nebo produkty či opakující se, nadměrná nebo nerelevantní klíčová slova. Pamatujte, že váš záznam v obchodě by měl být vhodný pro širokou veřejnost. Nepoužívejte v něm nevhodný text, obrázky ani videa.

- Nepoužívejte obrázky se sexuálním podtextem, na kterých jsou prsa, pozadí, genitálie, případně další fetišizované části těla nebo obsah, ať už jsou ilustrované, nebo skutečné.
- Nepoužívejte výrazy nevhodné pro širokou veřejnost. V záznamu aplikace nepoužívejte hrubé ani vulgární výrazy. Pokud jde o důležitý prvek vaší aplikace, musíte jeho prezentaci v záznamu v obchodě cenzurovat.

- Nepoužívejte explicitní násilí nápadně vyobrazené v ikonách aplikace, propagačních obrázcích nebo videích.
- Nepoužívejte vyobrazení nelegálního užívání drog. V záznamu v obchodě musí být dokonce i vzdělávací, dokumentární, vědecký nebo umělecký obsah týkající se užívání drog vhodný pro všechny skupiny publika.

Osvědčilo se vyzdvihnutí předností vaší aplikace. Ujistěte se, že název a popis vaší aplikace přesně popisují její funkčnost. Uveďte stručný a výstižný popis vaší aplikace. Kratší popisy zanechají lepší uživatelský dojem hlavně na zařízeních s menšími obrazovkami. Přehnaná délka, podrobnosti nebo opakování mohou vést k porušení těchto pravidel.

Google pro aplikační obchod nepovoluje aplikace, které přímo nebo nepřímo využívají propagační postupy považované za klamavé nebo poškozující uživatele či ekosystém vývojářů, nebo aplikace, které z takovýchto postupů těží. Zahrnuje to aplikace, které využívají klamavé reklamy na webech, v aplikacích či jiných prostředcích včetně upozornění, která jsou podobná systémovým upozorněním a výstrahám, propagační nebo instalační taktiky, které přesměrovávají uživatele do služby Google Play nebo stahují aplikace bez informovaného přičinění uživatele.

Hodnocení a recenze slouží jako ukazatele kvality aplikace. Uživatelé důvěřují tomu, že jsou spolehlivé a relevantní. Vývojáři se nesmí pokoušet ovlivnit umístění aplikací v Obchodě. Zahrnuje to zlepšování hodnocení nebo recenzí produktů protizákonnými prostředky, jako jsou podvodné instalace a placené nebo falešné recenze a hodnocení. Aplikace nesmí požádat uživatele o hodnocení slíbením odměny. Problémem je opakované odesílání hodnocení s cílem ovlivnit umístění aplikace na hlavní stránce Obchodu Google Play či odesílání recenzí či nabádání uživatelů aby odesílali recenze obsahující nevhodný obsah, včetně partnerských kuponů, herních kódů, e-mailových adres nebo odkazů na weby či jiné aplikace.

Google nepovoluje aplikace, které obsahují či propagují sexuálně explicitní obsah, jako je pornografie, nebo ve všeobecnosti obsah ani služby, které mají sloužit k sexuálnímu uspokojení. Zahrnuje to znázornění sexuálních aktů či naznačujících póz, propagační obrázky sexuálních pomůcek, propagaci sexuálně explicitního obsahu generovaného uživateli, obsah, který znázorňuje, opisuje nebo podporuje zoofilii, a aplikace, které propagují eskortní služby nebo služby, jež poskytují sexuální praktiky za kompenzaci. Google netoleruje absolutně žádný obsah zobrazující sexuální zneužívání dětí. Pokud se dozví o obsahu, který zobrazuje sexuální zneužívání dětí, nahlásí to příslušným orgánům a odstraní příslušné účty Google uživatelů zapojených do distribuce.

Nejsou povoleny aplikace, které znázorňují nebo propagují bezdůvodné šíření násilí či jiné nebezpečné činnosti (například grafické znázornění nebo opis reálného násilí či násilných hrozeb jakékoliv osobě či zvířeti, pokyny k násilné činnosti včetně pokynů k výrobě bomb či zbraní, pokyny ke spáchání sebevraždy). Zakázané jsou aplikace, které propagují sebepoškozování. Za násilí se považují i hrozby, obtěžování nebo šikana, takže problematické jsou aplikace s obsahem generovaným uživateli, ve kterých chybí dostatečné zabezpečení proti hrozbám, obtěžování nebo šikaně, hlavně neplnoletých, a aplikace obsahující příspěvky, komentáře nebo fotografie v rámci aplikace, jejichž hlavním účelem je obtěžovat nebo vybrat jinou osobu s cílem zneužívání, škodlivého útoku nebo zesměšňování.

Nejsou povoleny aplikace podněcující k nenávisti vůči určitým skupinám osob na základě jejich rasového nebo etnického původu, náboženství, postižení, pohlaví, věku, statusu veterána nebo

sexuální orientace či rodové identity. Google nepovoluje aplikace, které zneužívají přírodní katastrofy, krutý čin, konflikt, úmrtí nebo jinou tragickou událost ke zvýšení publicity nebo které v souvislosti s takovouto událostí neprojevují přiměřenou ohleduplnost.

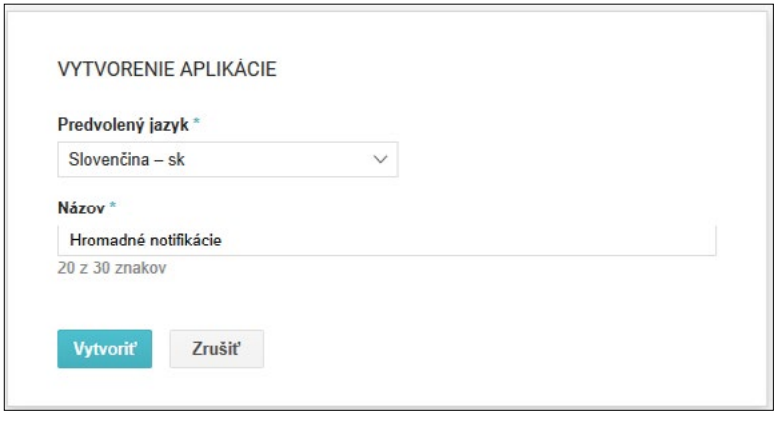
Google nepovoluje obsah ani služby zabezpečující internetové hazardní hry, jako jsou například internetová kasina, sportovní sázkové hry, loterie nebo hry založené na šikovnosti, které nabízí peníze nebo jinou hodnotnou výhru. Zakázané jsou aplikace, které nabízejí uživatelům možnost vydělání skutečných peněz.

Reklamy na hazard jsou povoleny pouze tehdy, když splňují následující požadavky:

- Aplikace a reklamy včetně inzerentů obsahu souvisejícího s hazardem musí být v souladu se všemi příslušnými zákony a standardy odvětví v případě všech míst, na kterých se zobrazuje reklama na hazard
- Reklama musí splňovat místní požadavky týkající se licence v případě všech propagovaných produktů a služeb souvisejících s hazardem
- Aplikace nezobrazuje reklamy na hazard osobám, které jsou mladší než osmnáct let
- Aplikace není součástí programu Designed for Families
- Aplikace není zacílena na primární cílové publikum, kterým jsou osoby mladší než 18 let
- Vstupní stránka reklamy zřetelně zobrazuje informace o zodpovědném hazardu

Publikování aplikace

Pomocí Konzole pro vývojáře Google Play můžete nahrát svoje soubory APK, spravovat záznam v obchodě a nakonfigurovat integrované služby a funkce. Na stránce **Všechny aplikace** klepněte na tlačítko **+ Přidat novou aplikaci** a nahrajte soubor APK. Také je potřeba vyplnit údaje o aplikaci, takzvaný záznam aplikace pro obchod. V prvním kroku publikování vyberte jazyk a zadejte název aplikace



Obrázek 21.9: Publikování aplikace, první dialog

Pokračujete zadáním krátkého a úplného popisu aplikace. Tyto popisy musí být pravdivé. Záznam obsahuje tyto údaje:

- **Jazyk:** Tento údaj označuje jazyk vaší aplikace. Můžete přidat překlady názvu a popisu aplikace a propagovat tak svou aplikaci i mezi uživateli, kteří hovoří jinými jazyky. Překlady se zobrazí ve službě Google Play.
- **Název:** Název aplikace, který se zobrazí ve službě Google Play. Pro každý jazyk můžete zadat jiný název.
- **Krátký popis:** Toto je první text, který uživatelé uvidí, když se na záznam vaší aplikace v obchodě podívají z aplikace Obchod Play. Po klepnutí na tento text v aplikaci Obchod Play zobrazí úplný popis. V tomto poli je povolených maximálně 80 znaků.
- **Úplný popis:** Popis vaší aplikace ve službě Google Play. Na překlad popisu do dalších jazyků můžete využít funkci překládání. V tomto poli je povolených maximálně 4 000 znaků.
- **Co je nové v této verzi?:** Přidejte poznámky o změnách v konkrétní verzi zveřejněné aplikace. Informace o této funkci a návod na její použití najdete v článku Popis novinek v aplikaci v centru pomoci.
- **Typ aplikace:** Služba Google Play je rozdělena na dva hlavní typy aplikací: aplikace a hry. Vyberte některou z nich.
- **Kategorie:** Vyberte kategorii aplikace. Přečtěte si více informací o kategoriích.

Aby vaše aplikace zaujala uživatele natolik, že si ji koupí, je potřeba věnovat pozornost vytvoření záznamu pro obchod, který obsahuje nejen název a popis aplikace, ale i různé grafické prvky, snímky obrazovky a video.

POZNÁMKA

Při oslovení nových uživatelů je nesmírně důležité předvést funkce aplikace pomocí multimédií. Grafické prvky, snímky obrazovky a videa můžete použít na zvýraznění a propagaci aplikace v službě Google Play a v jiných propagačních kanálech.

Snímky obrazovky

Pro každou podporovanou platformu zařízení (telefon, 7palcový tablet a 10palcový tablet) můžete nahrát až 8 snímků obrazovky. Pokud chcete zveřejnit záznam v obchodě, musíte nahrát aspoň 2 snímky obrazovky ve formátu JPEG nebo 24bitový soubor PNG (bez kanálu alfa). Minimální rozměr je 320 a maximální 3 840 pixelů. Větší rozměr snímku obrazovky může být maximálně dvakrát větší než menší rozměr.

TIP

Aby se vaše aplikace mohla ve službě Google Play zobrazovat v sekci **Vytvořené pro tablety**, musíte nahrát snímky obrazovky tabletu.

Pokud je k dispozici video, bude ve službě Google Play vždy zobrazeno jako první. Za ním budou následovat záběry obrazovky pro zařízení, ve kterém si uživatel aplikaci prohlíží. V Google Play se upřednostňují snímky obrazovky pro tablety a zobrazují se před snímky obrazovky pro telefony.

Ikona ve vysokém rozlišení

Nutnou podmínkou je upload ikony ve vysokém rozlišení, který se využívá na různých místech ve službě Google Play. Je to 32bitový soubor PNG (s kanálem alfa) o rozměrech 512×512 pixelů. Maximální velikost souboru činí 1 024 kB.

POZNÁMKA

Ikona ve vysokém rozlišení nenahrazuje ikonu aplikace ve spouštěči, mělo by jít o její kvalitnější verzi ve vysokém rozlišení.

Hlavní grafika

Hlavní grafika se zobrazuje v horní části stránky Záznam v obchodě aplikace Obchod Play. Jde o účinný nástroj na kreativní představení funkcionality aplikace a na přilákání uživatelů. Hlavní grafika je JPEG nebo 24bitový soubor PNG (bez kanálu alfa) s rozměry 1 024 × 500 pixelů.

POZNÁMKA

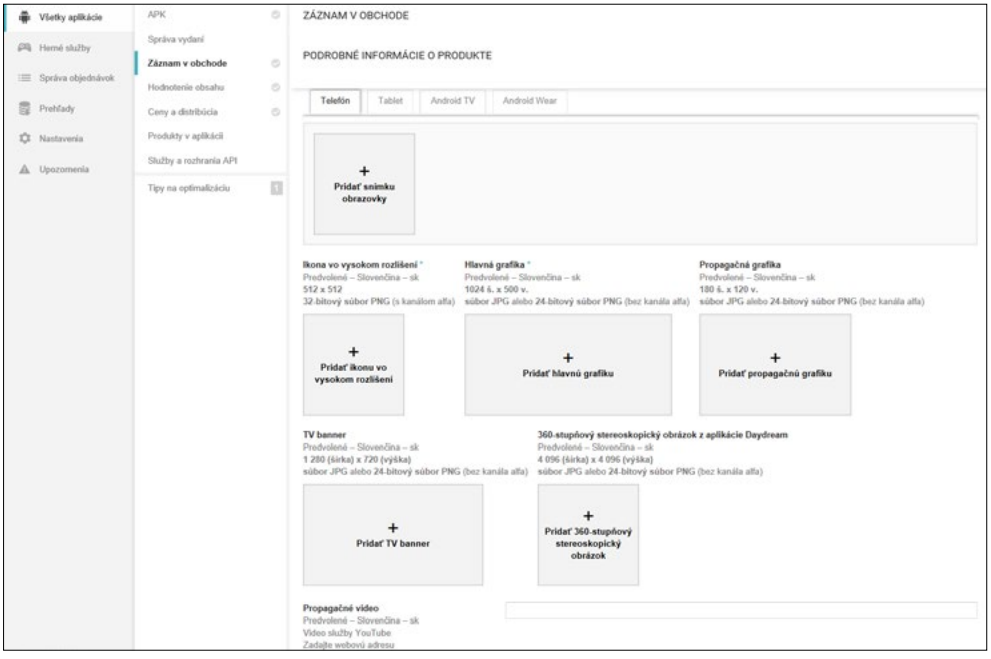
Pokud jste poskytli propagační video, hlavní grafiku překryje tlačítko na přehrání tohoto videa.

Tipy na hlavní grafiku:

- Žádný text ani důležité vizuální informace neumísťujte blízko okrajů prvku, hlavně ne v dolní třetině rámce. Loga a text se snažte umístit vodorovně i svisle do středu rámce.
- Pokud zahrnujete text, použijte velké písmo.
- Tato grafika se může zobrazovat samostatně bez ikony aplikace.

Propagační grafika

Propagační grafika se používá na propagaci ve verzích operačního systému Android starších než 4.0. Tento obrázek o rozměrech 180×120 px není zapotřebí na uložení a zveřejnění záznamu v obchodě.



Obrázek 21.10: Přidání grafických prvků do záznamu aplikace pro obchod

Propagační video

Zadáním webové adresy do tohoto pole můžete do služby Google Play přidat propagační video ze služby YouTube. Propagační video se mezi grafickými prvky v internetovém obchodě Google Play zobrazuje na prvním místě. Pokud používáte aplikaci Obchod Play, hlavní grafika bude překryta tlačítkem Přehrát. Propagační video není povinné, ale doporučujeme ho. Díky němu můžete potenciálním uživatelům předvést funkce vaší aplikace zábavným a poutavým způsobem. Videa by měla být krátká (30 sekund až 2 minuty) a měla by být zaměřena hlavně na nejzajímavější funkce.

TIP

Pokud chcete omezit propagaci aplikace v internetových nebo mobilních službách Google, musíte na stránce **Cena a distribuce** zaškrtnout políčko **Deaktivace marketingu**.

Maximální podporovaná velikost pro jeden soubor APK je 50 MB. Pomocí souborů rozšíření můžete nahrát další prvky, například grafiku. Pokud v jednom souboru APK nemůžete podporovat všechna požadovaná zařízení, můžete do daného záznamu aplikace nahrát více souborů APK, přičemž každý z nich bude zacílen na jinou konfiguraci zařízení.

Upozorňujeme, že názvy balíčků pro soubory aplikací jsou jedinečné a trvalé, vybírejte je proto s rozvahou. Názvy balíčků se nedají odstranit ani opětovně použít v budoucnosti. Nezapomínejte, že pokud jste ztratili svoje úložiště klíčů, budete muset aplikaci zveřejnit s novým názvem balíčku a novým klíčem. Měli byste také aktualizovat popis původní aplikace a zrušit její zveřejnění.

Rejstřík

#

<activity>, 118
<intent-filter>, 118
<uses-sdk>, 117
2D grafika, 371
3D grafika, 371

A

AbsoluteLayout, 147
ACCESS_FINE_LOCATION, 478
ACID (Atomicity, Consistency, Isolation, Durability), 323
Action Bar
– přizpůsobení, 239
– rozdělení, 239
ACTION_DOWN, 211
ACTION_UP, 211
ActionBar, 236
Activities, 136
Activity Manager, 77
Activity Name, 100
adaptér, kurzorový, 331
ADB (Android Debug Bridge), 36
ADT (Android Development Tools), 17
AIDL (Android Interface Definition Language), 550
akcelerometr, 432
– filtrované údaje, 438
– zobrazení údajů, 434
aktivita, 83, 136, 197
– hlavní, 99, 179, 186
– konfigurace typu, 101
– na editování záznamu, 342, 347
– na popředí, 86
– na přidání záznamu, 341, 346
– na zobrazení detailu, 194
– nastavení příznaku, 93
– navigace, 240
– obnovení stavu, 91
– pozastavená, 86
– přidání, 275–276
– spuštění, 92
– uložení stavu, 91
– vytvoření, 165
– využívající fragment, 182
– zastavená, 86
– životní cyklus, 84, 86, 88
Aktivita Master, 291
aktualizace
– průběžná, 26
– údajů, 559
– záznamu, 334
aktuální poloha, 483
AlertDialog, 281
anatomie
– Androidu, 65
– aplikace, 79
– projektu, 96
Android 5.0 Lollipop, 68
Android 6.0 Marshmallow, 70
Android 7.0 Nougat, 72
Android Monitor, 38
Android SDK, 20, 22
Android Studio, 17, 20
– instalace, 20
– konfigurace, 22
– schéma, 23
– spuštění, 22
Android TV, 609
– emulátor, 613
– vytvoření projektu, 610

Android TV Activity, 610
Android ViewGroup, 135
Android Virtual Device, 20
Android Wear, 29, 573
– emulátor, 576
– projekt, 579
AndroidManifest.xml, 117
Angry Birds, 82
animace, 390
– čas interpolace, 397
– doba trvání, 397
– fáze, 394
– klasická, 391
– metody, 391
– prvků uživatelského rozhraní, 401
– přesunu tlačítek, 402
– Tween, 394
AnimationDrawable, 391–392
APK, 628
aplikace, 77
– externí, 426
– identifikační karta, 119
– kritéria úspěšnosti, 619
– ladění, 169
– master-detail, 189
– multiplatformní, 536
– na popředí, 589
– nákupy, 623
– nativní, 536
– prodej, 623
– publikování, 615, 624, 628
– reklama, 623
– sdílení obsahu, 489
– se dvěma panely, 185
– test, 182
aplikační
– kód, 111
– logika, 162
– manifest, 117
App Bar, 233
application name, 97
architektura Androidu, 74
ArrayAdapter, 260, 276
ART (Android Runtime), 76
AudioManager, 407, 409
autentizace, 448
autorizace, 536
AVD manager, 33, 54
Azure Notification Hub, 523
– vytvoření, 525
Azure Portal, 525

B

Back Stack, 85
balíček, vytvoření, 624
banner, 616
barometr, 432
barva, doporučení, 125
Basic Activity, 100, 234, 248
beginDelayedTransition(), 402
bezpečnostní riziko, 75
bindService(), 544
Bluetooth, komunikace, 455
branch, 62
broadcast, 543
Broadcast receivers, 84
BufferedReader, 353
buttonClicked, 162

C

CallbackManager, 496
Camera, 408, 418, 473
Canvas, 371, 381
– dynamické vykreslování, 384, 387
– kreslení dotykem, 382
CardView, 70, 301–302
cílení reklamy, 617
cílová platforma, 98
cipherInit(), 450
CLASS, 76
close(), 327
cloud, 509
Command Line Shell For SQLite, 329
Company Domain, 97

ConstraintLayout, 404
Content provider, 84
contentDescription, 489
ContentProvider, 489, 551
ContentResolver, 551
contentTitle, 489
contentURL, 489
CoordinatorLayout, 147
CryptoObject, 450
Cue cards, 575
CursorLoader, 558

Č

čtenářský deník, 328

D

data, statická, 365
DatabaseHelper, 568
databáze, 309
– interakce, 326
– souborová, 323
– SQLite, 323
– vložení záznamu, 333
– zabudovaná, 324
databázová tabulka, 332
DataModel, 331
datová zpráva, 512
datový model, 331
Debugg, 170
definice
– objektů ve zdrojích, 113
– rozložení prvků, 109, 151
– uživatelského rozhraní, 107
DELETE, 330
delete(), 327
Design, 109
design
– Material, 121
– metrika, 127
– škálovatelný, 121
detail, zobrazení, 194
Detail Flow, 291
detekce gest, 218–219
disable(), 456
displej, multidotykový, 210
DocumentBuilder, 358
dočasný soubor, 322
doInBackground(), 376
dokončení kódu, 153
DOM (Document Object Model), 357
dotyk, 209
drawBitmap(), 381
drawColor(), 381
drawOval(), 381
drawPoints(), 381
drawText(), 381
DSL (Domain Specific Language), 49
dynamická
– elevace, 124
– grafika, 379
dynamické vykreslování, 384, 387
dynamický layout, 176

E

Eclipse, 17
editace záznamu, 347
editor kódu, 46
EditView, 328, 342
element
– <activity>, 118
– <string-array>, 272
– <uses-sdk>, 117
elevace
– dynamická, 124
– statická, 124
ELF, 76
Empty Activity, 161, 456
emulátor, 33
– Android Wear, 576
– parametry, 57
– snapshot, 56
– vytvoření, 54
enable(), 456
END_DOCUMENT, 361

END_TAG, 361
enterAlways, 248
enterAlwaysCollapsed, 248
environmentální senzor, 431
execSQL(), 327, 332–333
ExpandableListView, 279
explicitní
– intent, 197, 200
– záměr, 94
externí
– aplikace, 426
– úložiště, 309, 317

F

Facebook, 487
– přihlášení, 491
Facebook for developers, 487
Facebook SDK, 502
FacebookCallback(), 501
FCM (Firebase Cloud Messaging), 509–510
– konfigurace, 518
– vytvoření, 523
– zprávy, 512
File Explorer, 318
filozofie aplikace, 79
filtr
– HighPass, 439
– LowPass, 439
FingerPrintManager, 450
Firebase, 511
firemní systém, 536
firstChild, 358
Floating Action Button, 230
FloatingActionButton, 113, 234
font, doporučené rozměry, 130
Fork, 63
formát
– JSON, 349
– XML, 349
formátované zobrazení položek, 263
fotografie, snímání, 418, 423, 426
fragment, 173–174
– layout, 178
– programově přidaný, 173
– staticky přidaný, 173
– vytvoření, 244
– vytvoření layoutu, 185
– vytvoření třídy, 180
– životní cyklus, 175
FrameLayout, 142
Framework Transitions, 401
Freeform, 603
funkcionalita, 232

G

GCM (Google Cloud Messaging), 509
geomagnetický senzor, 432
gesto, 209
– detekce, 218–219
– pinch, 221
– vlastní, 226
Gesture Sensor, 432
GestureDetector, 218
Gestures Builder, 227
get<type>(), 327
getAll(), 310
getBoolean(), 310
getContentResolver(), 552
getCount(), 327
getDefaultSensor(), 433
getExternalStorageState(), 320
getFragmentManager(), 177
getInclination(), 442
getItemCount(), 302
getMenuInflater(), 235
getName(), 456
getOrientation(), 442
getRotationMatrix(), 442
getSharedPreferences(), 310
getState(), 456
getString(), 310
getSystemService(), 407
GitHub, 57
– vytvoření účtu, 58

GitShell, 64
Google AdWords, 620
Google Maps Android API, 472
Google Maps API, klíč, 474
Google Pixel, 53
Google Play
– povinnosti vývojáře, 622
– přidání do projektu, 527
– publikování aplikací, 615
– vyhledávání, 621
Google Play Services, 473
Google USB Driver, 36
GoogleMap, 472
GPS, 461, 469
GPS_PROVIDER, 462
Gradle, 17, 49
grafický tvar, 377
grafika, 371
– dynamická, 379
– hlavní, 630
– propagační, 630
– statická, 377
GridLayout, 143, 145, 351
GridLayoutManager, 302
GSM, 469
gyroskop, 432

H

HAL (Hardware Abstraction Layer), 75
Hall Sensor, 432
hardware, možnosti, 52
hasSystemFeature(), 428
hierarchická struktura, 279
HighPass, 439
historie verzí, 66
hlavní
– aktivita, 29, 99, 179, 186
– grafika, 630
hodinky, poslání notifikace, 584
HttpResponse, 354
hudba, přehrávání na pozadí, 546
Hyper-V, 30–32

I

identifikace senzoru, 433
identifikační karta, 119
ikona ve vysokém rozlišení, 630
imageURL, 489
ImageView, 373, 377
implicitní
– intent, 204
– záměr, 94
initLoader(), 558
InputStream, 354
INSERT, 329
insert(), 327
instalace
– Android Studio, 20
– Javy, 17
integrovaný senzor, 431
Intel Hardware Accellerated Execution Manager, 32
intent, 92, 197
Intent
– Action, 95
– Category, 95
– Component, 95
– Data, 95
intent
– explicitní, 197, 200
– implicitní, 204
Intent
– Extras, 96
– Flags, 96
– Type, 95
Intent Filter, 94
IntentService, 528
interakce
– s databází, 326
– s uživatelem, 209
internet, přístup, 375
interní úložiště, 309, 317
invalidate(), 381
iOS, 15

isDrawerOpen(), 256
isEnabled(), 456

J

JAR, 76
Java, 48
– instalace, 17
Java API Framework, 76
JAVA_HOME, 19
JDK (Java Development Kit), 17
– instalace, 19
– verze, 18
JSON, 349
– načtení údajů, 353
JsonReader, 353

K

karta, 242
– identifikační, 119
KeyGenerator, 449
klasická animace, 391
knihovna, nativní, 76
kombinace vnořených kontejnerů, 148
kompas, magnetický, 442
komunikace, 431
– Bluetooth, 455
konfigurace služby, 538
kontakt
– načítání, 558
– uložení, 553
kontejner
– pro umístění prvků, 137
– vnoření, 148
kontextové
– menu, 232
– menu položky, 267
kontextový stream, 575
kontrola oprávnění, 463
kreslení dotykem, 382
kritéria úspěšnosti reklamy, 619
kurzorový adaptér, 331

L

ladění, 37
– aplikace, 169
layout
– dynamický, 176
– fragmentů, 178
– statický, 176
Layout Name, 100
Light Sensor, 432
LinearLayout, 137
LinearLayoutManager, 301
Linux, 15
Linux Kernel, 75
ListActivity, 257, 259, 263, 267, 554
ListAdapter, 257, 260
listener, nListItemClick(), 262
ListView, 69, 191
– s údaji v kódu, 272
lišta, skrývání, 248
Location, 461
Location Manager, 77
LocationListener, 462
LocationManager, 461
LocationProvider, 461
lockCanvas(), 381
logika, aplikační, 162
lokalizační služba, 461
LowPass, 439

M

magnetický kompas, 442
MainActivity, 156
MainActivity.java, 111
manifest, aplikační, 117
MAP_TYPE_SATELLITE, 485
mapa, 461
– zobrazení polohy, 472, 478
MapFragment, 472
mapová služba, 461
Marker, 473
master-detail, 189

Material, 121
MediaController, 412
MediaPlayer, 407, 549
MediaRecorder, 408
menu, 232
– kontextové, 232
– options, 232
– pop-up, 232
Messenger, 550
metoda
– beginDelayedTransition(), 402
– bindService(), 544
– cipherInit(), 450
– close(), 327
– delete(), 327
– disable(), 456
– doInBackground(), 376
– drawBitmap(), 381
– drawColor(), 381
– drawOval(), 381
– drawPoints(), 381
– drawText(), 381
– enable(), 456
– execSQL(), 327, 332–333
– FacebookCallback(), 501
– get<type>(), 327
– getAll(), 310
– getBoolean(), 310
– getContentResolver(), 552
– getCount(), 327
– getDefaultSensor(), 433
– getExternalStorageState(), 320
– getFragmentManager(), 177
– getInclination(), 442
– getName(), 456
– getOrientation(), 442
– getRotationMatrix(), 442
– getSharedPreferences(), 310
– getState(), 456
– getString(), 310
– getSystemService(), 407
– hasSystemFeature(), 428
– initLoader(), 558
– insert(), 327
– invalidate(), 381
– isDrawerOpen(), 256
– isEnabled(), 456
– lockCanvas(), 381
– move(), 327
– moveToFirst(), 327
– moveToLast(), 327
– moveToNext(), 327
– newView(), 556
– next(), 361
– onAccuracyChanged(), 433
– onActivityResult(), 497
– onBackPressed(), 254
– onBind(), 545
– onCompletion(), 548
– onCreate(), 327, 545
– onCreateView(), 495
– onEnterAmbient(), 591
– onGesturePerformed(), 229
– onItemLongClick(), 262
– onLoadFinished(), 558
– onNavigationItemSelected(), 255
– onNewIntent(), 592
– onPostExecute(), 376
– OnReceive(), 534
– onSensorChanged(), 433–434, 444
– onStartCommand(), 544
– onUpdateAmbient(), 591
– onUpgrade(), 327
– onWindowFocusChanged(), 395
– openFileInput(), 317
– openFileOutput(), 317
– pause(), 407
– prepare(), 407
– putExtra(), 198
– putInt(), 310
– putString(), 310
– query(), 327
– rawQuery(), 327
– release(), 408

– remove(), 310
– requestLocationUpdates(), 462
– seekTo(), 408
– sendNotification(), 534
– setAntiAlias(), 382
– setAudioSource(), 414
– setClass(), 198
– setColor(), 382
– setContentView(), 373, 383
– setDataSource(), 407
– setName(), 456
– setOnLoadCompleteListener(), 409
– setOutputFormat(), 414
– setStrokeWidth(), 382
– setTextSize(), 382
– setVideoSource(), 414
– start(), 407
– startActivity(), 199
– startDiscovery(), 456
– startForeground(), 549
– startService(), 544
– stop(), 408
– stopSelf(), 544
– stopService(), 544
– surfaceDestroyed(), 388
– surfaceChanged(), 388
– takePicture(), 421, 424
– toString(), 265, 287
– unbindService(), 544
– unlockCanvasAndPost(), 381
– update, 334
– withEndAction(), 400
metrika designu, 127
Microsoft Azure Mobile Apps, 536
minimální verze SDK, 28
množné číslo, 115
mobilní služba, 535
model, datový, 331
modifikátor, 116
monitorování spuštěné aplikace, 38
Morphing, 106
move(), 327
moveToFirst(), 327
moveToLast(), 327
moveToNext(), 327
možnosti
– hardware, 52
– senzoru, 433
MP3, 546
mřížka uživatelského rozhraní, 127
Multi-Window, 73, 601
– životní cyklus aplikace, 604
multidotykový displej, 210
multimédia, 407
multimediální reklama, 616
multiplatformní
– aplikace, 536
– operační systém, 65

N

nahrávání, 415
– videa, 427
– zvuku, 414
náklonoměr, 446
nákupy v aplikaci, 623
násobky rozměrů, 131
nástroje pro vývoj, 17
nativní
– aplikace, 536
– knihovna, 76
– reklama, 616
navigace, 461
– na jinou aktivitu, 240
– pomocí menu, 234
Navigation Drawer, 250
navození prostoru, 122
návrh uživatelského rozhraní, 103, 136, 263
návrhové zobrazení, 253
newView(), 556
next(), 361
Nexus, 52
Node.js, 538
NodeList, 358
non-streaming sensor, 434

Notification Manager, 77
notifikace
– poslání do hodinek, 584
– přijímání, 520
– vylepšená, 595
notifikační zpráva, 512
novinky, 595

O

objekt
– přidání do seznamu, 298
– s obrázkem, 286
– seznam, 257
obraz v obraze, 601, 608
obrázek
– poslání, 497
– z internetu, 375
– zobrazení, 372
– zobrazení programově, 373
obrazovka
– rozdělení, 601
– rozlišení, 125
– variabilita, 131
obsah
– poskytování, 551
– poskytovatel, 84
– rozdělení na karty, 242
– sdílení, 489
obsluha události, 156, 159
odběr, 623
odemknutí pro vývoj, 35
offline režim, 536
okraje, doporučení, 129
onAccuracyChanged(), 433
onActivityCreated(), 176
onActivityResult(), 497
onAttach(), 175
onBackPressed(), 254
onBind(), 545
onBindViewHolder(), 302
onCompletion(), 548
onCreate(), 87, 112, 327, 333, 545
onCreateContextMenu(), 235
onCreateOptionsMenu(), 235
onCreateView(), 176, 495
onCreateViewHolder(), 302
onDestroy(), 88
onDestroyView(), 176
onDetach(), 176
onDown(), 219
onEnterAmbient(), 591
onFling(), 219, 224
onGesturePerformed(), 229
onLoadFinished(), 558
onLongPress(), 219
onMessageReceived(), 512, 522
onNavigationItemSelected(), 255
onNewIntent(), 592
onPause(), 87, 175
onPostExecute(), 376
OnReceive(), 534
onResume(), 87
onScroll(), 219
onSensorChanged(), 433–434, 444
onShowPress(), 219
onSingleTapUp(), 219
onStart(), 87
onStartCommand(), 544
onStop(), 88
onTouchEvent, 383
onUpdate(), 333
onUpdateAmbient(), 591
onUpgrade(), 327
onWindowFocusChanged(), 395, 400
openFileInput(), 317
openFileOutput(), 317
oprávnění, kontrola, 463
options menu, 232
orientace, změna, 178
OS X, 15
osvětlení, senzor, 446
otisk prstů, 447
ověřování, 536
oznámení, 537

P

Package Manager, 77
package name, 97
PackageManager, 428
panel, vysouvací, 250
paralelní větev, 62
pause(), 407
Picture-In-Picture, 608
pinch, 221
platba od inzerentů, 619
platforma, cílová, 98
plátno, vytvoření, 381
plocha
– displeje, 587
– poměr stran, 130
– uvítací, 616
PNG, 373
počítadlo, univerzální, 223
podpora více oken, 601
pohled, 136
pohybový senzor, 431
pole řetězců, 115, 270
poloha
– aktuální, 483
– zařízení, 462, 468
– zobrazení na mapě, 472, 478
polohový senzor, 431
poměr stran, 130
pop-up, menu, 232
poskytování
– obsahu, 551
– údajů, 562
– údajů z databáze, 565
poskytovatel obsahu, 84, 309
poslání
– obrázku, 497
– statusu, 497
– zprávy, 514, 523
pozastavená aktivita, 86
pozicování, relativní, 141
pravidla publikování aplikace, 626
pravidlo násobků rozměrů, 131
PreferenceFragment, 313
PreferenceScreen, 314
prepare(), 407
Preview All Screen Sizes, 109–110
Preview Representative Sample, 110
primární klíč, 331
prodej aplikace, 623
profit z reklamy, 616
projekt
– složka, 100
– soubor, 100
– struktura, 41, 293
– vytvoření, 27, 96
proměnné prostředí, 19
propagační
– grafika, 630
– video, 631
Property Animation, 396
– komplexní využití, 399
Proximity Sensor, 432
prst, otisk, 447
průběžná
– aktualizace, 26
– analýza kódu, 47
prvky, rozmístění, 133, 151
přehrávání
– hudby na pozadí, 546
– videa, 412
– zvuku, 408
přidání záznamu, 341, 346
přihlášení, Facebook, 491
přijmutí notifikace, 520
příkazový řádek, 325
připojení k mobilní službě, 535
přístup
– k internetu, 375
– ke kontaktům, 558
– údajům, 537
přizpůsobení rozlišení obrazovky, 125
publikování
– aplikace, 628

– aplikací, 615
Pull, 63
Push, 63
putExtra(), 198
putInt(), 310
putString(), 310

Q

query(), 327

R

rámeček, vypnutí, 41
rawQuery(), 327
RecyclerView, 69, 301–302
referenční zařízení, 52
registrace vývojářského účtu, 622
RegistrationIntentService, 530
reklama, 615
– cena za klepnutí, 619
– cena za realizovanou transakci, 619
– cena za zobrazení, 619
– cílení, 617
– multimediální, 616
– nativní, 616
– odhad profitu, 616
– specifika, 617
– v aplikaci, 623
– vsunutá, 616
Relative Layout, 274
RelativeLayout, 139
relativní pozicování, 141
release(), 408
remove(), 310
rendering pixelů, 384
repozitář
– veřejný, 59
– vytvoření, 60
requestLocationUpdates(), 462
resources, 113
režim, volný, 601
Ringtone, 407
RingtoneManager, 407
rozdělení obrazovky, 601
rozestup řádků, 130
rozlišení obrazovky, 125
rozložení prvků, 109, 151
rozmístění prvků, 133

Ř

řádek
– příkazový, 325
– rozestup, 130

S

scroll, 248
ScrollView, 142, 150
sdílené vlastnosti, 309
sdílení obsahu, 489
SDK, minimální verze, 28
SDK Manager, 23, 25
seekTo(), 408
sekce, <intent-filter>, 118
SELECT, 326, 329
sendNotification(), 534
Sensor, 432
SensorEvent, 432, 436
SensorEventListener, 433
SensorManager, 432
senzor, 431
– environmentální, 431
– identifikace, 433
– integrovaný, 431
– možnosti, 433
– osvětlení, 446
– pohybový, 431
– polohový, 431
– získávání údajů, 432
setAntiAlias(), 382
setAudioSource(), 414
setClass(), 198
setColor(), 382
setContentView(), 112, 373, 383
setDataSource(), 407
setName(), 456

setOnLoadCompleteListener(), 409
setOutputFormat(), 414
setStrokeWidth(), 382
setTextSize(), 382
setVideoSource(), 414
seznam
– detailů, 193
– objektů, 191, 257, 301
– objektů s obrázky, 286
– přidání objektu, 298
– s obrázky, 258
– výběr více položek, 273
SharedPreferences, 309
Shazam, 79
síťové úložiště, 309
skrývání aplikační lišty, 248
služba, 84, 543
– konfigurace, 538
– lokalizační, 461
– mapová, 461
– spuštěná, 544
– vázaná, 544
– životní cyklus, 544
snímač otisků prstů, 447
snímání fotografie, 418, 423, 426
snímek obrazovky, 629
sociální síť, 487
soubor
– dočasný, 322
– s definicí, 105
– uložení údajů, 316
– úprava, 63
souborová databáze, 323
SoundPool, 407, 409
specifika reklamy, 617
správa verzí zdrojového kódu, 57
spuštěná služba, 544
spuštění
– jiné aktivity, 161
– na reálném zařízení, 36
– změna způsobu, 208
SQL, 326
SQLite, 309, 323
SQLiteDatabase, 333
SQLiteOpenHelper, 331
StaggeredGridLayoutManager, 302
START_DOCUMENT, 360
START_TAG, 360
start(), 407
startActivity(), 118, 199
startDiscovery(), 456
startForeground(), 549
startService(), 544
statická
– data, 365
– elevace, 124
– grafika, 377
statický layout, 176
status, poslání, 497
stav
– Paused, 176
– pozastavený, 176
– Resumed, 176
– spuštěný, 176
– Stopped, 176
– zastavený, 176
stop(), 408
stopSelf(), 544
stopService(), 544
stránka vývojáře, 623
stream, kontextový, 575
streaming sensor, 434
strings.xml, 104
struktura, projektu, 41, 293
surfaceDestroyed(), 388
surfaceChanged(), 388
SurfaceView, 381, 387, 412
synchronizace offline dat, 537

Š

škálovatelný design, 121

T

TabLayout, 245

TableLayout, 143
TableRow, 340
tabulka, databázová, 332
takePicture(), 421, 424
televizor, 609
test, aplikace, 182
TEXT, 360
textový, řetězec, 115
tlačítko, FloatingActionButton, 113
toString(), 265, 287
TransitionDrawable, 391
třída
– Android ViewGroup, 135
– ArrayAdapter, 265
– AudioManager, 407
– BufferedReader, 353
– Camera, 418
– ContentProvider, 551
– ContentResolver, 551
– CryptoObject, 450
– DatabaseHelper, 568
– FingerPrintManager, 450
– FragmentManager, 177
– FragmentTransaction, 177
– fragmentu, 180, 187
– GestureDetector, 218
– hlavní aktivity, 181
– JsonReader, 353
– ListActivity, 259, 554
– LoaderManager, 558
– Location, 461
– RegistrationIntentService, 530
– SharedPreferences, 309
– SQLiteOpenHelper, 331
– SurfaceView, 381, 387
– View, 135
– XMLPullParseHandler, 368
tvar, grafický, 377
Tween, 394
TweenAnimation, 391
typ prvku, 106

U

údaje aktualizace, 559
událost, 135
– dotyku, 211
– obsluha, 156, 159
– onWindowFocusChanged(), 400
uložení
– do externí paměti, 320
– do interní paměti, 317
– dočasných souborů, 322
– kontaktu, 553
– skóre, 310
– údajů, 309
– údajů do souboru, 316
úložiště
– externí, 309, 317
– interní, 309, 317
– síťové, 309
unbindService(), 544
univerzální počítadlo, 223
unlockCanvasAndPost(), 381
UPDATE, 330
uses-permission, 375
uvítací plocha, 616
uživatel, interakce, 209
uživatelské rozhraní, 135

V

variabilita
– obrazovek, 131
– zařízení, 131
vázaná služba, 544
verze, historie, 66
veřejný repozitář, 59
větev
– paralelní, 62
– vytvoření, 62
vícenásobný dotyk, 214
video
– nahrávání, 427
– propagační, 631

– přehrávání, 412
VideoView, 412
View, 135
View System, 77
ViewFlipper, 224
ViewGroup, hierarchie objektů, 136
ViewGroups, 136
Views, 108, 136
vizuální návrh, 103
vlastní gesto, 226
vlastnosti prvku, 44
vnoření kontejnerů, 148
volný režim, 601
vsunutá reklama, 616
výběr
– více položek se seznamu, 273
– záznamů z tabulky, 335
vyhledávání, Google Play, 621
vykreslování
– dynamické, 384, 387
– na Canvas, 381
vymazání záznamu, 334
vypnutí rámečku, 41
vysouvací panel, 250
vytvoření
– aktivity, 165
– Azure Notification Hub, 525
– databázové tabulky, 332
– emulátoru, 54
– fragmentu, 244
– hlavní aktivity, 30
– layoutu fragmentu, 185
– plátna, 381
– projektu, 96
– projektu aplikace, 27
– repozitáře, 60
– větve, 62
vývojář, 35
– povinnosti, 622
– stránka, 623
vývojové
– nástroje, 17
– prostředí, 39

W

WAKE_LOCK, 589
Waze, 80
Wear Round, 581
Wear Square, 581
Wi-Fi, 469
Window Manager, 77
Windows, 15
withEndAction(), 400
WRITE_EXTERNAL_STORAGE, 322

X

XML, 349
– načtení údajů, 357
– statická data, 365
– zobrazení obrázku, 372
XMLPullParseHandler, 368
XmlPullParser, 360

Z

zabudovaná databáze, 324
základní grafický tvar, 377
záměr
– explicitní, 94
– implicitní, 94
zarážka, 169
zarovnání prvků, 127
zařízení
– odemknutí pro vývoj, 35
– poloha, 468
– variabilita, 131
zastavená aktivita, 86
záznam
– aktualizace, 334
– editace, 342, 347
– přidání, 341, 346
– vložení do databáze, 333
– výběr, 335
– vymazání, 334
změna orientace, 178

zobrazení
– detailu, 194
– hierarchické struktury, 279
– návrhové, 253
– polohy, 472, 478
– událostí dotyku, 211
– vícenásobných dotyků, 214
zpráva
– datová, 512
– non-collapsible, 513
– notifikační, 512
– poslání, 514, 523
způsob spuštění, 208
zvětšování objektu, 396
zvuk
– nahrávání, 414
– přehrávání, 408

Mistrovství – Android

Ľuboslav Lacko

Překlad: Martin Herodek
Obálka: IMIDEA
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Translation © Martin Herodek, 2017

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN tištěné verze 978-80-251-4875-4
ISBN e-knihy 978-80-251-4878-5 (1. zveřejnění, 2017)

Cena uvedená výrobcem představuje nezávaznou doporučenou spotřebitelskou cenu.

Vydalo nakladatelství Computer Press v Brně roku 2017 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 25 266.
© Albatros Media a. s., 2017. Všechna práva vyhrazena. Žádná část této publikace nesmí být kopírována a rozmnožována za účelem rozšiřování v jakékoli formě či jakýmkoli způsobem bez písemného souhlasu vydavatele.

1. vydání

ALBATROS MEDIA a.s.

Kompletní nabídku titulů naleznete na
www.albatrosmedia.cz