Plně kompatibilní s Android 5

Ľuboslav Lacko

Vývoj aplikací pro Android

Rychlý přechod na vývoj pro Android

Od základů po pokročilé techniky

Tipy pro Android Wear a Google Glass



computer press

Ľuboslav Lacko

Vývoj aplikací pro Android

Computer Press
Brno
2015

Kniha byla zakoupena na serveru alza.cz.

Kupující: Martin Kadlas

Adresa: Konevova 910, 13000 Praha 3, cz

ID 20110-210027144460408221 12-169947-210

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.

Vývoj aplikací pro Android

Ľuboslav Lacko

Překlad: Martin Herodek
Obálka: Martin Sodomka
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Translation © Martin Herodek, 2015

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-4347-6

Vydalo nakladatelství Computer Press v Brně roku 2015 ve společnosti Albatros Media a. s.
se sídlem Na Pankráci 30, Praha 4. Číslo publikace 19014.
© Albatros Media a. s. 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.

Obsah

Úvod **11**

KAPITOLA 1

Nástroje pro vývoj **15**

Co budete potřebovat **15**

Instalace programovacího jazyka Java 15

Java 8 16

Vývojové prostředí Eclipse 17

Instalace a konfigurace Android SDK a doplňků ADT 17

Vytvoření emulátoru **22**

Vytvoření emulátoru platformy Android 4.4 Wearable 24

Spouštění aplikací na reálném zařízení 25

Ověření konfigurace na cvičném projektu **26**

Expresní seznámení se s vývojovým prostředím 31

Dalvik Debug Monitor Server (DDMS) 32

Spuštění aplikace v emulátoru 32

Spuštění aplikace na reálném zařízení 34

Android Studio **35**

Import projektů z Eclipse 39

Embarcadero RAD Studio XE6 **40**

Vizuální návrh aplikací v C++ pro Android 41

FireMonkey 41

App Tethering 43

Xamarin MonoTouch a Mono for Android **43**

Game Maker Studio na vývoj her **43**

Příklad vytvoření nejjednodušší hry 44

KAPITOLA 2

Architektura **49**

Operační systém Android **49**

Historie verzí 50

Android 5.0 Lollipop (API 21) **52**

Jakou nejstarší verzi Androidu podporovat v aplikaci **56**

Knihovny pro dopřednou kompatibilitu 57

Stručně o architektuře Androidu **58**

Linux Kernel 59

Libraries 60

Android Runtime 60
Application Framework 60
Aplikace 61

Bezpečnost na platformě Android 61
Oprávnění pro aplikaci 61

Základní součásti aplikace pro Android 64
Aktivity (Activity) 64
Služby (Services) 64
Broadcast Receivers 65
Poskytovatelé obsahu (Content providers) 65

Aktivita a její životní cyklus 65
Životní cyklus aktivity 66
Intent (záměr) 69
Předávání údajů a výsledků 70
Intent Filter 71

Úvod do asynchronního programování 74
AsyncTask 75
AsyncTaskLoader 76
IntentService 76
Handler 76

Možnosti ukládání údajů 76

KAPITOLA 3

Základní principy aplikace pro Android 77

Příklad – vytvoření projektu 77
Vytvoření projektu 77
Specifikace verzí 79
Ikony pro aplikaci 79
Výběr typu hlavní aktivity 80
Anatomie projektu 83
Definice objektů ve zdrojích (resources) 86
Aplikační manifest 90
Návrh uživatelského rozhraní 92
Přizpůsobení rozlišení obrazovky 94
Obrázek 9-patch 95

Kontejnery na rozmístění prvků 95
LinearLayout 96
RelativeLayout 98
FrameLayout 100
TableLayout 101
GridLayout 103
Vnoření kontejnerů 105
ScrolView 106

Příklad – definice rozložení prvků **107**
Obsluha události 113
Příklad – spuštění jiné aktivity **114**
Aplikační logika 115
Vytvoření nové aktivity 118
Ladění aplikace **121**
Příklad – Ohodnocení **123**
Příklad – zjištění informací o vašem zařízení **126**

KAPITOLA 4
Vizuální návrh uživatelského rozhraní **131**
Námět příkladu **131**
Rozbor řešení **131**
Návrh uživatelského rozhraní **132**
Programování aplikační logiky **137**
Domácí úkol **141**
Shrnutí **141**

KAPITOLA 5
Interakce s uživatelem **143**
Nabídka funkcionality – menu **143**
Navigace pomocí lišty Action Bar **143**
Přizpůsobení a rozdělení lišty Action Bar 147
Navigace na jinou aktivitu (a zpět) 148
Přidání widgetu na lištu Action Bar 150
Dotyky a gesta **154**
Multitouch 154
Příklad – zobrazení vícenásobných dotyků **156**
Detekce standardně používaných gest **160**
Příklad – univerzální počítadlo **160**
Příklad – využití vlastních gest **164**

KAPITOLA 6
Notifikace, alarmy **169**
Notifikace **169**
Příklad – zobrazení toastu s vlastním designem 169
Příklad – poslání notifikace 171
Alarmy **174**
Vytvoření alarmu 174
Příklad – ověření funkcionality alarmů 175

KAPITOLA 7
Seznamy objektů **179**
Námět příkladů **179**
Řešení s využitím ListActivity **179**
ListActivity s formátovaným zobrazováním položek **182**
Návrh uživatelského rozhraní 182
Aplikační kód 184
ListActivity s kontextovým menu položky **186**
Řešení s využitím ListView a statickými údaji v poli řetězců **188**
Návrh uživatelského rozhraní 188
Aplikační kód 189
Řešení s využitím ListView a údaji v kódu **190**
Návrh uživatelského rozhraní 190
Aplikační kód 191
Výběr více položek ze seznamu **192**
Návrh uživatelského rozhraní 192
Aplikační kód 193
Zobrazení hierarchické struktury **195**
Návrh uživatelského rozhraní 196
Aplikační kód 197
Seznam objektů s obrázky **202**
Návrh uživatelského rozhraní 202
Aplikační kód 203

KAPITOLA 8
Fragmenty **209**
Námět příkladu **209**
Rozbor řešení 210
Fragmenty **211**
Životní cyklus fragmentu 212
Vytvořte kompatibilní projekt 214
Zobrazení detailních informací 216
Zobrazení seznamu objektů 217
Aktivity na zapouzdření fragmentů 218

KAPITOLA 9
Broadcasty **223**
Broadcast Receiver **223**
Příklad – Broadcast Receiver registrovaný v aplikačním manifestu **225**
Příklad – dynamicky registrovaný Broadcast Beceiver **227**
Příklad – indikace příchozího hovoru **229**

KAPITOLA 10

Ukládání údajů **231**
Ukládání údajů **231**
Třída SharedPreferences 231
Příklad – uložení nejvyššího dosaženého skóre hry 232
Příklad – PreferenceFragment 234
Ukládání údajů do souboru **237**
Příklad – ukládání do souboru v interní paměti 238
Příklad – ukládání do souboru v externí paměti 240
Ukládání dočasných souborů **243**
Databáze SQLite **244**
Interakce aplikace s databází **244**
Příklad – čtenářský deník **245**
Rozbor řešení 245
Datový model **248**
Vytvoření databázové tabulky 250
Vkládání záznamů do databáze 251
Aktualizace záznamů 252
Vymazání záznamů 252
Výběr údajů z databázové tabulky 253
Návrh uživatelského rozhraní **254**
Hlavní aktivita 255
Aktivita na přidání záznamu 258
Aktivita na editování existujícího záznamu 259
Programování aplikační logiky **262**
Hlavní aktivita 262
Aktivita na přidání záznamu 263
Aktivita na editování záznamu 264

KAPITOLA 11

Aplikace pracující s údaji JSON, XML **269**
Námět **269**
Rozbor zadání 269
Zdroj údajů pro aplikaci **270**
Návrh uživatelského rozhraní **272**
Aplikace na načítání údajů ve formátu JSON **274**
Aplikace na načítání údajů ve formátu XML **278**
Varianta využívající Document Object Model 278
Varianta využívající XmlPullParser 281
Statická data ve formátu XML **286**
Aplikační kód 288

KAPITOLA 12
Grafika a animace **293**
Zobrazení obrázku přes XML návrh **294**
Zobrazení obrázku programově **295**
Zobrazení obrázku z Internetu **296**
Rozbor řešení 296
Povolení přístupu k Internetu 297
Návrh uživatelského rozhraní 297
Aplikační kód 298
Vykreslování základních grafických tvarů **300**
Vykreslování na Canvas **304**
Kreslení dotykem na Canvas **306**
Dynamické vykreslování na Canvas s využitím View **307**
Dynamické vykreslování na Canvas s využitím SurfaceView **310**
Animace **314**
Metody klasické animace 314
Příklad animace TransitionDrawable 314
Příklad animace AnimationDrawable 315
Příklad animace Tween 317
Property Animation 320
Příklad na ilustraci principu fungování Property Animation 321
Příklad komplexnějšího využití Property Animation 323

KAPITOLA 13
Multimédia **327**
Pasivní a aktivní konzumace multimédií **327**
Příklad – přehrání zvukového efektu a ovládání hlasitosti **328**
Příklad – přehrání videa **332**
Příklad – nahrávání zvuku **334**
Příklad – Snímání fotografie I **338**
Příklad – Snímání fotografie II **343**
Příklad – Snímání fotografie s využitím externí aplikace **346**

KAPITOLA 14
Senzory, mapové služby **349**
Integrované senzory chytrých telefonů a tabletů **349**
Získávání údajů ze senzorů 350
Příklad – zobrazení údajů z akcelerometru 351
Příklad – zobrazení filtrovaných údajů z akcelerometru 354
Příklad – magnetický kompas 357

Senzor osvětlení 361
Náklonoměr 361
Lokalizační a mapové služby 362
Příklad – určení polohy zařízení 363
Příklad – určení polohy zařízení 366
Zobrazení polohy na mapě 370
Přípravné kroky 371
Příklad zobrazení polohy na mapě 375
Příklad – zobrazení polohy v jiné aplikaci schopné zobrazit mapy 378

KAPITOLA 15
Služby a broadcasty 383
Služba a její životní cyklus 384
Příklad – přehrávání hudby na pozadí 386

KAPITOLA 16
Poskytování obsahu 391
Třída ContentProvider 391
Příklad – přístup ke kontaktům v zařízení 392
Příklad – přístup ke kontaktům, načítání údajů na pozadí 395
Příklad – aktualizace údajů 397
Příklad – vytvoření aplikace, která bude poskytovat údaje 400

KAPITOLA 17
Připojení ke cloudovým službám a sociálním sítím 405
Trendy a doporučení 406
Google Cloud Messaging for Android 407
Získání klíče Simple API Access 407
Vytvoření projektu na Google Cloud Console 408
Implementace na straně serveru 410
Vytvoření Azure Notification Hub 411
Vytvoření aplikace pro Android vysílající a přijímající notifikace 413
Posílání notifikací 421
Připojení aplikace pro Android k mobilní službě 423
Microsoft Azure Mobile Services 423
Vytvoření nové mobilní služby 424
Vytvoření aplikace pro Android využívající mobilní službu 426
Úprava existující aplikace pro Android, aby mohla využívat mobilní službu 432
Připojení aplikace k sociálním sítím 432
Vytvoření aplikace pracující s Facebookem 434
Vytvoření aplikace pro Android přihlašující se k Facebooku 436

KAPITOLA 18
Publikování aplikací do služby Google Play **441**
Registrace vývojářského účtu **441**
Vytvoření balíčku aplikace na publikování **442**
Publikování aplikace **443**
Příprava záznamu pro obchod 444

KAPITOLA 19
Nové platformy – Android Wear a Google Glass **449**
Android Wear **449**
Nabídka: Kontextový stream 450
Dotaz: Cue cards 451
Vytvoření emulátoru Android Wear 453
Projekt aplikace pro Android Wear 455
Google Glass **457**
Technické údaje 458
Návrh designu aplikací 458
Vývoj aplikací 459

Rejstřík **463**

Úvod

Vzhledem k aktuálním trendům v IT a životnímu stylu hlavně mladých lidí, kteří si bez mobilních zařízení – chytrých telefonů a tabletů – už nedokáží představit svou existenci, získávají na významu mobilní aplikace. Většina uživatelů se zařadila do hlavního proudu, tedy mezi uživatele aplikací. Někteří však mají vyšší ambice a chtějí například vyřešit svoje individuální požadavky, případně mají nápad a chtějí ho realizovat formou mobilní aplikace. Tato publikace vám formou praktických postupů odhalí know-how vývoje aplikací pro mobilní platformu Android, ať už se jedná o chytré telefony nebo tablety.



Poznámka: Podle údajů od agentur IDC a Gartner ovládá v současnosti Android přibližně 80 % trhu s mobilními přístroji a je provozován na více než miliardě telefonů a tabletů. To platí ve všeobecnosti, nikoliv však v podnikové sféře.

V publikaci se pokusíme zbořit mýtus o složitosti vývoje mobilních aplikací a nároků na vybavení pro tuto činnost. První pokusy s vývojem mobilních aplikací nevyžadují žádné investice, jelikož díky emulátorům dokážete vyvíjet i bez toho, abyste měli příslušné zařízení fyzicky k dispozici.

Publikace je určena i migrujícím vývojářům, kteří chtějí svou úspěšnou aplikaci, vytvořenou původně pro počítač nebo pro některou mobilní platformu, zpřístupnit i milionům uživatelů zařízení s Androidem.

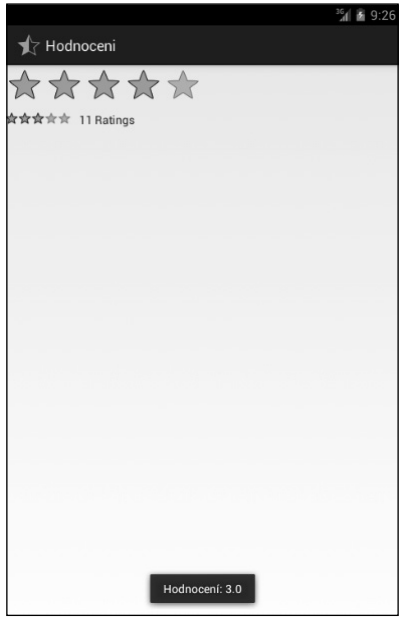
Publikace předpokládá určité předběžné znalosti:

- 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.

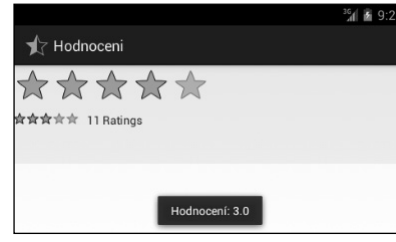
Všechny příklady v této publikaci byly odladěny na telefonu Sony Xperia L s operačním systémem Android 4.2.2 a tabletu Lenovo Yoga Y10 s Androidem 4.4.



Poznámka: Některé screenshoty cvičných aplikací zobrazují relevantní obsah pouze v horní a dolní části. Většina plochy uprostřed snímku obrazovky je prázdná. Proto jsme pro úsporu místa v publikaci tyto obrázky upravili tak, že jsme prázdnou oblast odstranili.



Obrázek Ú.1: Původní obrázek



Obrázek Ú.2: Upravený obrázek

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 najdete chybu, ať už chybu v textu nebo

v kódu, budeme rádi, pokud nám ji oznámíte. Ostatní uživatele tak můžete ušetřit frustrace a pomoci nám zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K2170* po klepnutí na odkaz Soubory ke stažení.

KAPITOLA 1

Nástroje pro vývoj

V této kapitole:
- Co budete potřebovat
- Vytvoření emulátoru
- Ověření konfigurace na cvičném projektu
- Android Studio
- Embarcadero RAD Studio XE6
- Xamarin MonoTouch a Mono for Android
- Game Maker Studio na vývoj her

Co budete potřebovat

K vývoji aplikací budete potřebovat čtyři základní nástroje a komponenty:

- Java Development Kit (JDK)
- Vývojové prostředí Eclipse
- Android Development Tools (ADT)
- Android Software Development Kit (SDK)

Alternativu představuje Android Studio, které však je v současnosti k dispozici jen ve verzi Early Access Preview. Aplikace pro Android je možné vyvíjet na platformě Windows, Linux i Mac. Vstupním bodem k získání vývojářských nástrojů, návodů a příkladů je stránka *http://developer.android.com*.

Instalace programovacího jazyka Java

Jazyk Java byl vytvořen vývojářským týmem firmy Sun Mircosystems pod vedením Jamese Goslinga. Původně měl být určen pro spotřební elektroniku. Pro zajímavost – původní název projektu byl Oak (dub). Po zjištění, že už existuje programovací jazyk s tímto jménem, byl přejmenován na Javu. Vývoj první verze byl ukončen v roce 1995.

Java je objektově orientovaný programovací jazyk, jehož syntaxe je podobná C nebo C++. Java je interpretovaný jazyk – to znamená, že program se nepřekládá přímo do strojového kódu, ale do takzvaného Java bajtkódu, který je následně interpretován virtuálním strojem Java. To zabezpečuje, že Java je na platformě nezávislý jazyk. O správu paměti se stará automatický

„garbage collector", jenž zabezpečuje, aby objekty, které se už nepoužívají, byly odstraněny. Problém však představuje skutečnost, že dopředu nevíme, kdy bude spuštěn.

Existují čtyři edice jazyka Java zaměřené na různá prostředí s různě rozsáhlým obsahem.

- **JavaCard** určená hlavně pro čipové karty
- **Java ME (Micro Edition)** zaměřená na mobilní telefony
- **Java SE (Standard Edition)** čili standardní verze pro klasické počítače
- **Java EE (Enterprise Edition)** určená pro rozsáhlé podnikové informační systémy

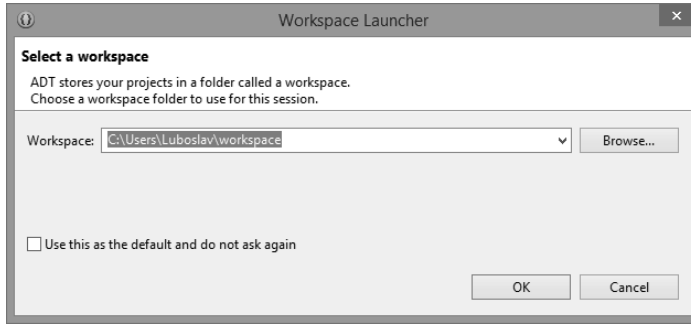


Poznámka: Pod taktovkou Javy běží více než 3 miliardy zařízení, aplikace celosvětově vytváří 9 milionů vývojářů.



Tip: Programovací jazyk Java, konkrétně Java Development Kit, stáhnete pro platformy Windows a Linux z webu Oracle. Pro platformu Mac OS stáhnete Javu ze stránek společnosti Apple. Platforma Java byla produktem společnosti Sun, tuto společnost však se všemi produkty a řešeními získala společnost Oracle, která Javu momentálně spravuje. Z jejich stránek je možné stáhnout a nainstalovat Java SDK.

Pokud se nejste jistí, zda máte JDK nainstalované, nainstalujte balík Android SDK. Ve složce *eclipse* spusťte aplikaci *eclipse*. Pokud JDK máte, spustí se vývojové prostředí normálně. Pokud tento doplněk nainstalovaný nemáte, aplikace se nespustí a vypíše chybové hlášení. Java Development Kit najdete snadno pomocí vyhledávače, do kterého zadáte frázi „Java JDK". V době psaní publikace byla k dispozici verze Java 8. Android SDK však v aktuální verzi dokázal pracovat jen se starší verzí JDK 7. Podle verze operačního systému, který máte na vývojářském počítači, si nainstalujte 32- nebo 64bitovou verzi Java Development Kitu.



Obrázek 1.1: Upozornění na nutnost instalace Java Development Kitu

JDK se na platformě Windows instaluje implicitně do složky *C:\Program Files\Java*.

Java 8

Přes konstatování, že Android SDK v aktuální verzi dokázal v době psaní publikace pracovat jen se starší verzí JDK 7, je jen otázkou času, kdy bude podporovat i verzi Java 8. Proto si v hrubých rysech představíme nejvýznamnější novinky této verze.

Podpora pro platformu Java SE 8 je tak jako u všech předchozích verzí automaticky dostupná ve vývojovém prostředí NetBeans v den uvedení na trh. Ostatní vývojová prostředí si také uvědomila důležitost a výjimečnost této verze Javy a začlenila její podporu také od prvních dní (například Eclipse IDE podporuje Java 8 od verze 4.3 Kepler).

Vývojáři budou schopní psát v nové Javě kód, který je kompaktnější a snáze udržovatelný. Klíčovými novinkami v JDK 8 významně redukujícími objem kódu jsou lambda výrazy nad velkými objemy dat, nový interpret jazyka JavaScript známý pod kódovým označením Nashorn a eliminované permanentní generování kódu z virtuálního stroje. Nashorn běží jako součást Java Virtual Machine a umožňuje javovým aplikacím využívat komponenty napsané v JavaScriptu, případně spouštět v JVM celé javascriptové aplikace. K snazší lokalizaci aplikací přispějí i nové možnosti práce s datem a časem.

Java 8 by měla podstatně urychlit vývojové cykly všech typů aplikací včetně mobilních a aplikací pro různé spotřebiče a výrobky v rámci „Internetu věcí". K dosažení avizovaných cílů přispívá i spolupráce ARM a Oraclu na definici a integraci technologií. Jedním z výsledků této spolupráce je i platforma Oracle JDK 8. V blízké budoucnosti se očekává expanze různých „wearable" zařízení, tedy inteligentních náramků, brýlí a podobně. Pro tato zařízení umožňuje Java 8 zjednodušení komunikace, lehkou škálovatelnost a zvýšenou robustnost vyvíjených aplikací. K efektivitě vývoje přispěje i vývojové prostředí pro vestavná (embedded) zařízení.

Vývojové prostředí Eclipse

Eclipse je open- source vývojové prostředí (IDE – Integrated Development Environment) určené primárně k programování v jazyce Java. Jeho flexibilita vám však umožňuje nainstalovat doplňky pro další programovací jazyky, například PHP, Python, C++, Ruby a další. Jelikož je Eclipse samotné napsáno v Jazyce Java, potřebujete mít nainstalované prostředí JRE (Java Runtime Environment).

Instalace a konfigurace Android SDK a doplňků ADT

Vývojářský balík Android SDK, pro který se často používá i zkratka ADK, je k dispozici zdarma na *http://developer.android.com/sdk*. Stáhněte soubor s názvem *adt-bundle-<os_platforma-datum>.zip*. V našem případě měl soubor pro platformu Windows název *adt-bundle-windows-x86_64-20140321.zip*, kde poslední posloupnost čísel udává datum vydání edice. Balík obsahuje tyto součásti:

- Vývojové prostředí Eclipse + ADT plugin
- Android SDK Tools
- Android Platform-tools
- Obrazy verzí operačního systému pro emulátor



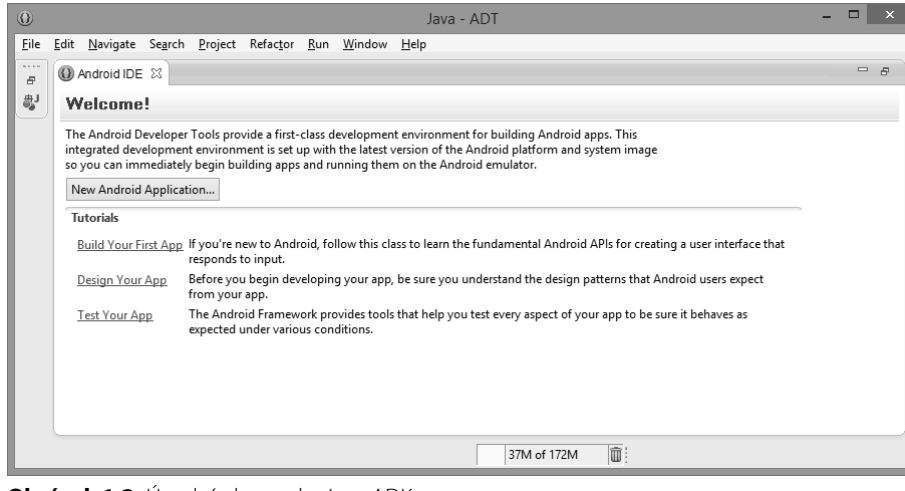
Poznámka: Android SDK není potřeba instalovat, abyste ale mohli začít vyvíjet aplikace, musíte ho rozbalit a hlavně nakonfigurovat. Konfigurace je jednodušší, než se na první pohled zdá, je však zapotřebí dodržet postup.

Námi popisovaný postup je pro operační systém Windows 8, ale úplně stejně můžete postupovat i ve starší verzi Windows 7.

1. V archivu staženém z webu je složka se stejným názvem jako název souboru archivu. Tuto složku rozbalte na vhodné místo. Doporučujeme vytvořit složku s výstižným názvem, například *Android*, *Vývoj* apod. Ve složce *adt-bundle-<os_platforma-datum>* jsou vnořené složky *eclipse*, *sdk* a spustitelná aplikace *SDK Manager*. Vývojové prostředí vyžaduje, aby bylo nainstalované prostředí Java Runtime Enviroment (JRE) nebo Java Development Kit (JDK).
2. Ve složce *eclipse* spusťte aplikaci *eclipse.exe*. Zobrazí se uvítací obrazovka s aktuálními informacemi, návody, příklady a podobně. V této fázi úvodní obrazovku zavřete.

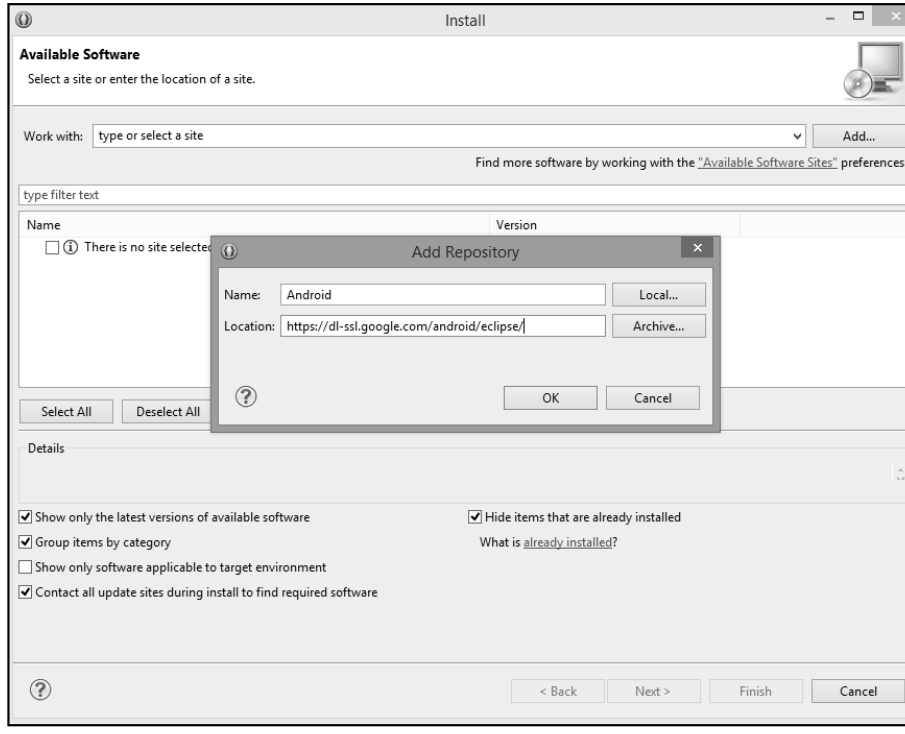


Tip: Eclipse můžete spustit z jakékoliv složky, do níž jste jej rozbalili – nevyžaduje žádnou instalaci. Jelikož je Eclipse vstupním bodem pro vývoj aplikací, doporučujeme vytvořit pro něj na ploše operačního systému zástupce.



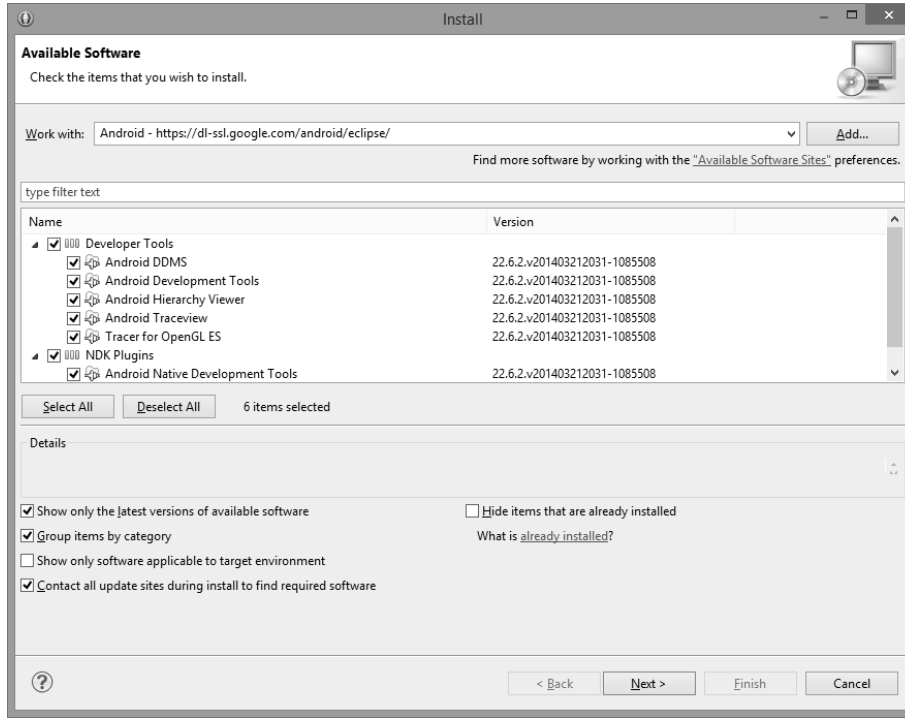
Obrázek 1.2: Úvodní obrazovka Java ADK

3. Aktivujte v nabídce položku **Help → Install New Software**. V instalačním dialogu klepněte na tlačítko **Add** a přidejte nový zdroj modulů *https://dl-ssl.google.com/android/eclipse/*. Zdroj modulů nějak nazvěte, například **Android**. Přidají se moduly Android Developers Tools (ADT).

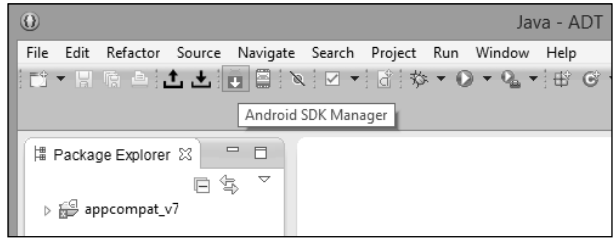


Obrázek 1.3: Instalace zásuvných modulů Android Developers Tools (ADT)

4. V seznamu modulů přibydou dvě položky: **Developer Tools** a případně i **NDK Plugins**. Obě položky zaškrtněte a klepněte na tlačítko **Next**. Dokončete instalaci modulů. Průběh jejich stahování a instalace můžete sledovat v dialogu.
5. Po ukončení instalace modulů je potřeba ukončit aplikaci Eclipse a znovu ji spustit. V námi instalované verzi stačilo souhlasit s nabídkou na restartování a vývojové prostředí se automaticky ukončilo a znovu spustilo.
6. Po nainstalování Android Developers Tools (ADT) je potřeba do vývojového prostředí Eclipse nakonfigurovat propojení na Android SDK. Tím se propojí moduly ADT nainstalované v předchozích krocích s SDK. Spusťte Android SDK Manager. Můžete ho spustit přímo z Eclipse pomocí ikony panelu nástrojů (ikona se symbolem zelené šipky směřující dolů).



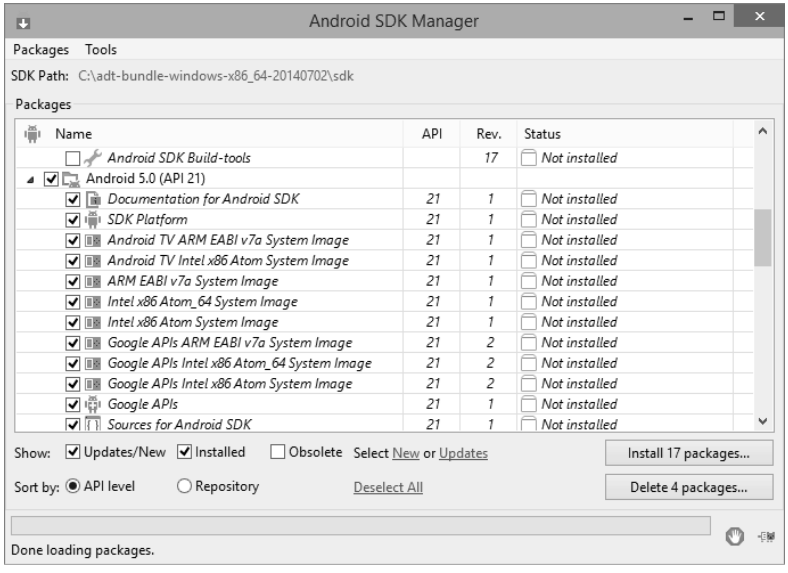
Obrázek 1.4: Konfigurace zásuvných modulů Android Developers Tools (ADT)



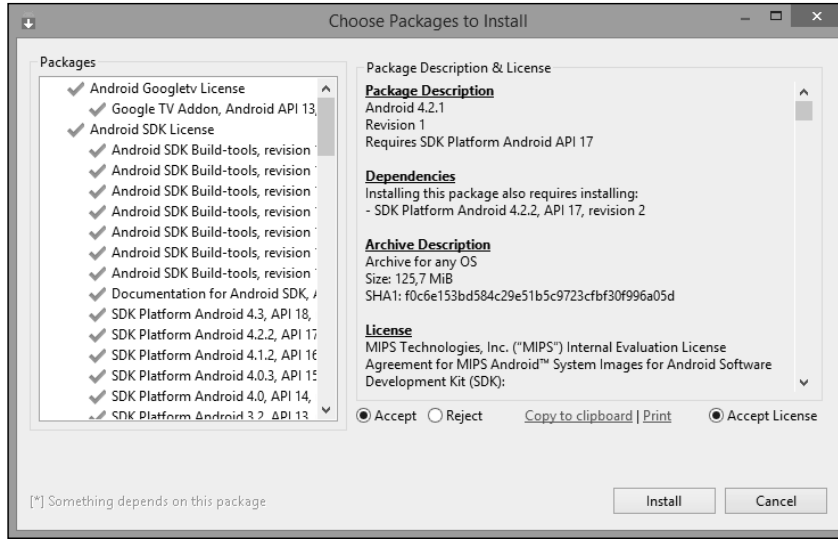
Obrázek 1.5: Spuštění nástroje Android SDK Manager přímo z prostředí Eclipse

7. V aplikaci Android SDK Manager označte položky, které chcete instalovat. Označte složky **Tools**, **Extras** a všechny verze Androidu, pro které chcete vyvíjet a testovat aplikace. Pokud chcete pokrýt i starší zařízení, nainstalujte vývojářskou podporu počínaje verzí 2.2. Tak pokryjete nejširší spektrum momentálně dostupných zařízení. Chytré telefony nemají dlouhou životnost, přístrojů s kdysi velmi rozšířenou verzí 2.2 je stále méně a méně. V současnosti převládají telefony a tablety s verzí 4.0 a vyšší. Verze 3.0 HoneyComb byla určena jen pro tablety a přístrojů s touto verzí v našich končinách není mnoho. Doporučujeme vydat se spíše cestou inovací. V době psaní publikace byla k dispozici verze ADT, která už podporovala novou platformu Android 5.0 Lollipop a Android 4.4W, kde W znamená „wearable“, čili variantu Androidu pro zařízení, která můžeme nosit na so-

bě. Typickým příkladem jsou hodinky od různých výrobců či populární brýle Google Glass, které však zatím nejsou v České republice ani na Slovensku oficiálně dostupné. Povzbuzujeme vás, abyste si SDK pro Android 5.0 Lollipop a Android 4.4 Wearable nainstalovali, vytvořili si pro tyto platformy emulátory a jako vývojáři se s nimi v předstihu seznámili.



Obrázek 1.6: Instalace balíčků v aplikaci Android SDK Manager



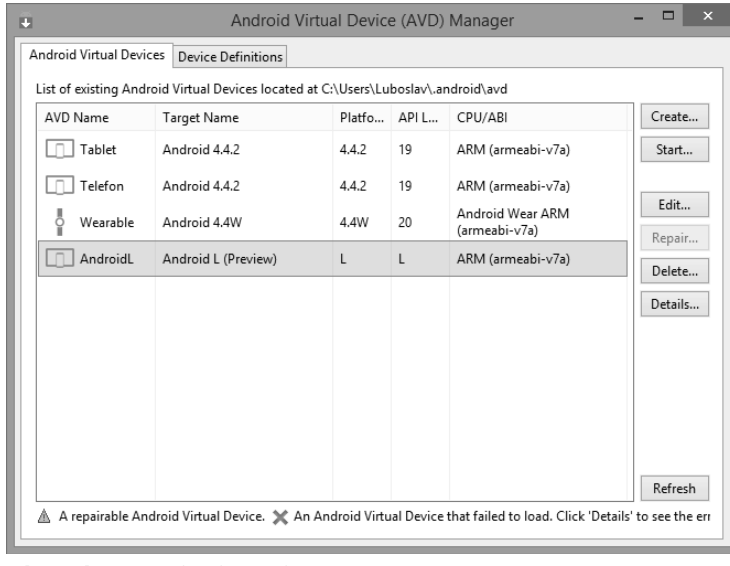
Obrázek 1.7: Souhlas s licenčními podmínkami při instalaci balíčků v aplikaci Android SDK Manager

8. V následujícím dialogu musíte postupně akceptovat licence všech balíčků, které hodláte instalovat. Prohlédněte si celý seznam, aby všude byly jen zelené ikonky bez červených křížků.

Vytvoření emulátoru

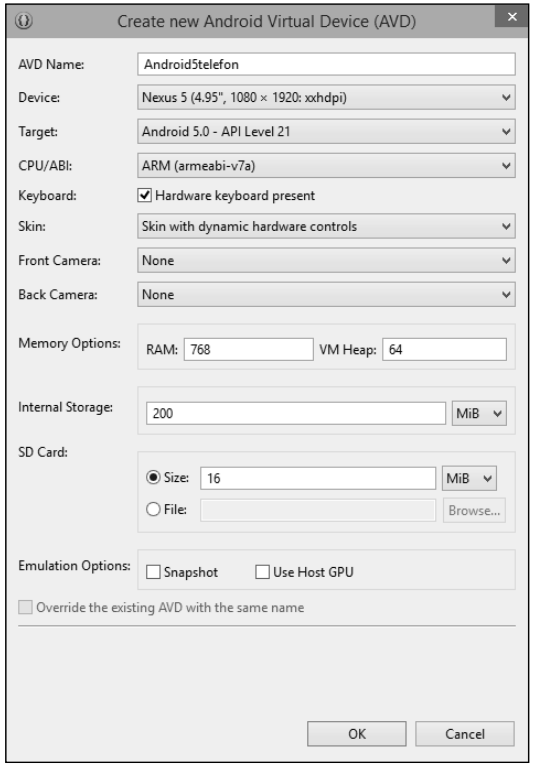
Zdálo by se, že pokud máte k dispozici jeden moderní chytrý telefon a jeden tablet s Androidem, dokážete pokrýt návrh, testování a ladění aplikací. Opak je pravdou. Výhodou a zároveň nevýhodou Androidu je variabilita různých zařízení s různými verzemi systému a různým rozlišením displeje. Neodmyslitelnou 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 SDK Manager v nabídce **Tools** zvolte položku **Manage AVDs**. Zkratka AVD znamená Android Virtual Devices. Zobrazí se okno aplikace **Android Virtual Device Manager**. Přepněte se na **Android Virtual Devices** a nainstalujte emulátor pro příslušnou verzi Androidu. Vyberte si verzi, kterou disponuje vaše nebo zamýšlené mobilní zařízení, pro nějž je aplikace určena.

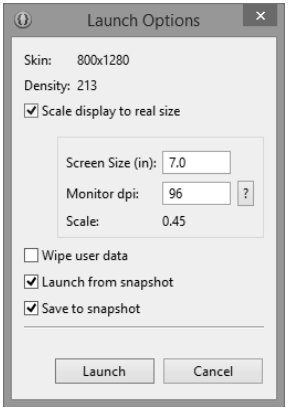


Obrázek 1.8: Android Virtual Device Manager

Pro účely této publikace doporučujeme vytvořit minimálně dva emulátory, oba pro verzi Android 4.2.2 (API Level 17) nebo 4.4: jeden emulátor telefonu s úhlopříčkou 4 palce a rozlišením 480 × 800, druhý emulátor tabletu s úhlopříčkou 7 až 10 palců. Pokud označíte možnost **Snapshot**, druhé a další spuštění emulátoru proběhne velmi rychle, protože AVD po zavření ukládá svůj aktuální stav.



Obrázek 1.9: Vytvoření emulátoru pro platformu Android 4.2.2



Obrázek 1.10: Dialog nastavení škálování zařízení

Abyste se seznámili s množnostmi nové verze Android 5.0 Lollipop, doporučujeme vytvořit i emulátor této platformy, jelikož nejnovější přístroje se budou postupně na tuto verzi upgradovat.

Při vytváření emulátoru nezapomeňte nakonfigurovat dostatečnou kapacitu paměti **SD Card**. Emulátor můžete spustit přímo z dialogu **Android Virtual Device Manager** tlačítkem **Start**. První spuštění emulátoru trvá trochu déle, u dalších spuštění je už doba náběhu přiměřená.

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 800 × 1 200, případně vyšším, na monitoru s vertikálním rozlišením 768 pixelů? Po spuštění emulátoru tlačítkem **Start** se zobrazí dialogové okno umožňující nastavení měřítka zobrazení. Doporučujeme označit volbu **Scale display to real size**.



Poznámka: Kvůli kompatibilitě vyberte nejnižší předpokládanou verzi systému. Takovéto aplikace budou na vyšších verzích fungovat bez problémů, opačně to ale neplatí.



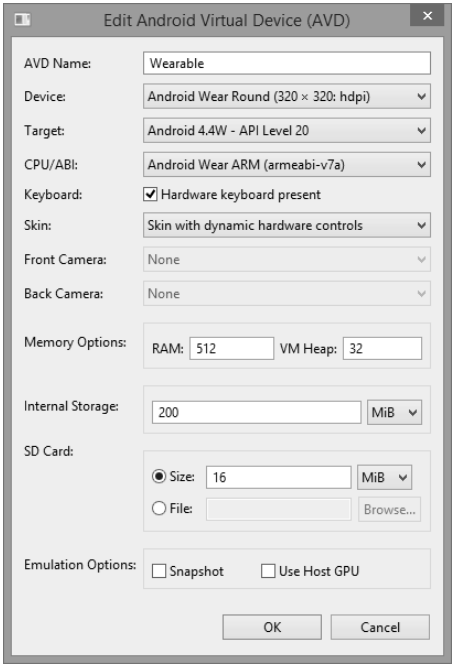
Obrázek 1.11: Spuštění emulátoru pro platformu Android 4.2.2

Pro hloubavější čtenáře je diskový obraz emulátoru v operačním systému Windows 8 uložen v adresáři *C:\Users\<uživatel>\.android*\. V souboru *config.ini* jsou základní parametry emulátoru.

Vytvoření emulátoru platformy Android 4.4 Wearable

Pro platformu Android 4.4 Wearable jsou k dispozici šablony **Android Wear Round** (320 × 320; hdpi) a **Android Wear Square** (280 × 280; hdpi).

Při prvním spuštění emulátoru probíhá inicializace operačního systému. Na emulátoru to trvá několik minut a během této doby doporučujeme, aby byl váš počítač připojen k Internetu. Po spuštění emulátoru pokračujte v konfiguraci zařízení podle pokynů.



Obrázek 1.12: Vytvoření emulátoru platformy Android 4.4 Wearable



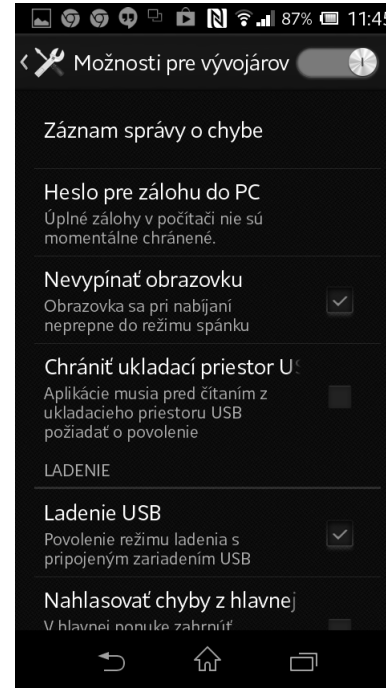
Obrázek 1.13: Emulátor Android 4.4 Wearable

Spouštění aplikací na reálném zařízení

Ke spuštění a ladění aplikací na reálném zařízení je potřeba přepnout jej do vývojářského módu. Klepněte v **Nastavení** na položku **Možnosti pro vývojáře**. Zaškrtněte položku **Ladění USB**. Doporučujeme zaškrtnout i položku **Nevypínat obrazovku**. Po označení této položky se v režimu, kdy je přístroj připojený přes USB kabel k počítači, nebude vypínat obrazovka.

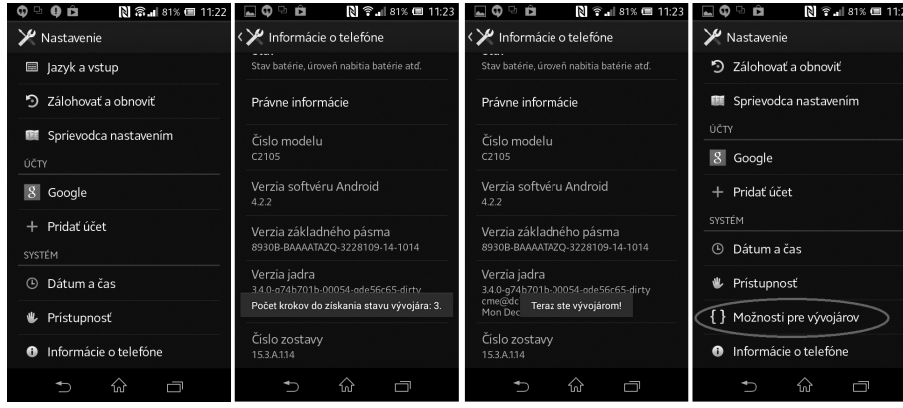
Na zařízeních s operačním systémem Android 4.2 a vyšším jsou možnosti pro vývojáře implicitně skryty a musíte je nejprve zobrazit:

1. V **Nastavení** klepněte na položku **Informace o telefonu**.
2. Následně najděte položku **Číslo sestavy**. Zpravidla je poslední v seznamu.



Obrázek 1.14: Povolení ladění přes USB

3. Opakovaně na tuto položku klepejte. Klepnutí musíte provést sedmkrát. Nejprve se zobrazí oznámení ve tvaru **Počet kroků do získání stavu vývojáře: 2**, které avizuje, kolikrát je potřeba ještě klepnout. Když se zobrazí oznámení **Nyní jste vývojářem**, přibude v **Nastavení** položka **Možnosti pro vývojáře**.



Obrázek 1.15: Postup odemknutí nabídky Možnosti pro vývojáře

Na straně počítače potřebujete ke 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.

Ověření konfigurace na cvičném projektu

Možná se ptáte, proč se pouštět do vytváření projektu mobilní aplikace dříve, než se seznámíte aspoň v hrubých rysech s architekturou 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.



Poznámka: 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.

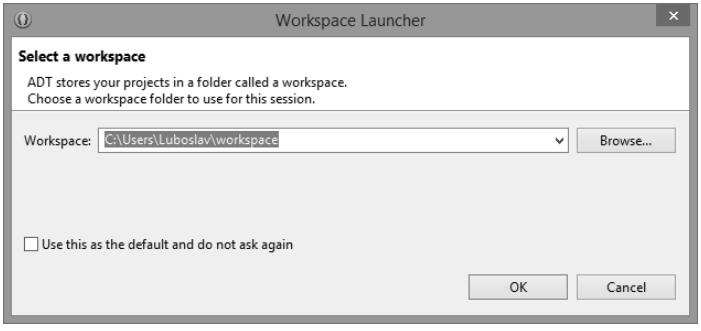
Začátečníci se v tomto příkladu nemusí snažit pochopit souvislosti. Návod na vytvoření cvičného projektu je uveden jako obrázkový postup krok za krokem.

Spusťte vývojové prostředí Eclipse. Zobrazí se dialogové okno s výběrem pracovního prostoru, tedy složky na disku, kam se budou ukládat projekty vývojového prostředí Eclipse a konfigurační soubory, které obsahují informace o rozmístění oken na pracovní ploše vývojového prostředí, informace o konfiguraci zásuvných modulů a podobně.



Poznámka: Praktickým důsledkem ukládání konfiguračních údajů je, že po opětovném spuštění se zobrazí pracovní plocha vývojového prostředí v takovém stavu jako při posledním ukončení práce.

Můžete mít společný pracovní prostor pro všechny projekty nebo můžete kvůli vyšší přehlednosti zvolit pro každý projekt samostatný pracovní prostor. Mezi pracovními prostory se můžete přepínat pomocí volby **File → Switch Workspace**. Po volbě se vývojové prostředí restartuje. Implicitně je nastavena složka *C:\Users\<uživatel>\workspace*. V prvním projektu můžete nechat nastavenou tuto složku, případně můžete postupovat systematicky a vytvořit i složku, do které budete umísťovat svoje projekty.



Obrázek 1.16: Výběr pracovního prostoru

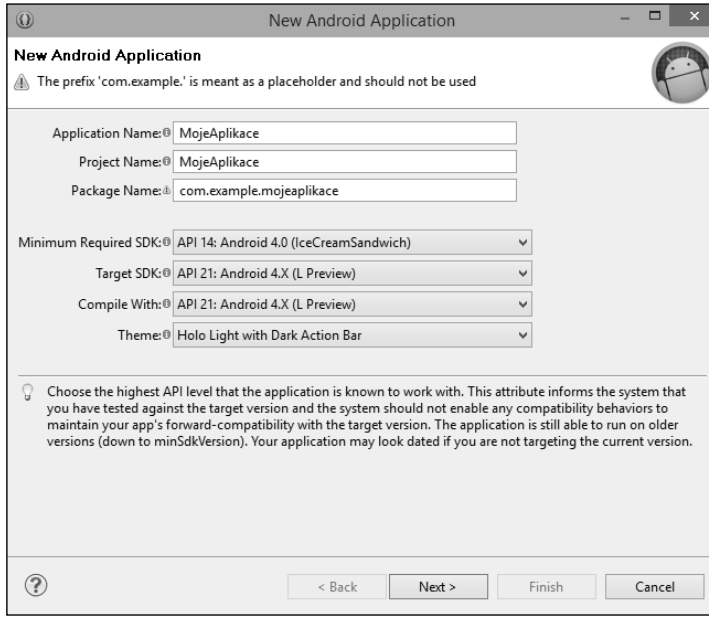
Pomocí nabídky **File → New → Android Application Project** vytvořte nový projekt. Můžete jej nazvat například MojeAplikace. Všimněte si tří polí pro zadání názvu. Název zadaný do pole **Application Name** se bude zobrazovat při spouštění projektu. Do pole **Package Name** se zadává název javového balíčku, do kterého bude projekt aplikace přibalen, například *com.example.mojeaplikace*. Stačí zadat název „MojeAplikace“ do pole **Application Name**, ostatní pole se automaticky vyplní implicitními názvy. Ponechejte i implicitně zadaný výběr maximální a minimální verze SDK. Implicitně je jako aktuální verze nastavená nejvyšší dostupná verze.

V době psaní publikace to byl Android 4.4 (KitKat). Jako nejnižší verze je nastaven Android 2.2 (Froyo). V poli **Theme** vybíráte barevné schéma aplikace. Implicitně je nastaveno světlé pozadí pro aplikaci a tmavé pozadí pro aplikační lištu (**Holo Light with Dark Action Bar**).

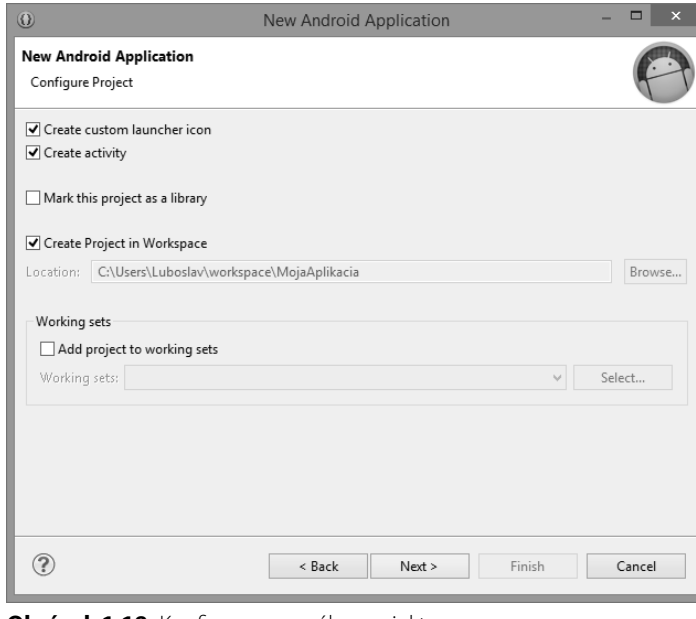
Vyberte verzi Androidu a vyplňte povinné položky **Project name**, **Application name** a **Package name** (ve tvaru *com.example.mojeaplikace*).

V dalším dialogovém okně ponechejte zaškrtnuté položky **Create custom launcher icon**, **Create Activity** a **Create Project in Workspace**.

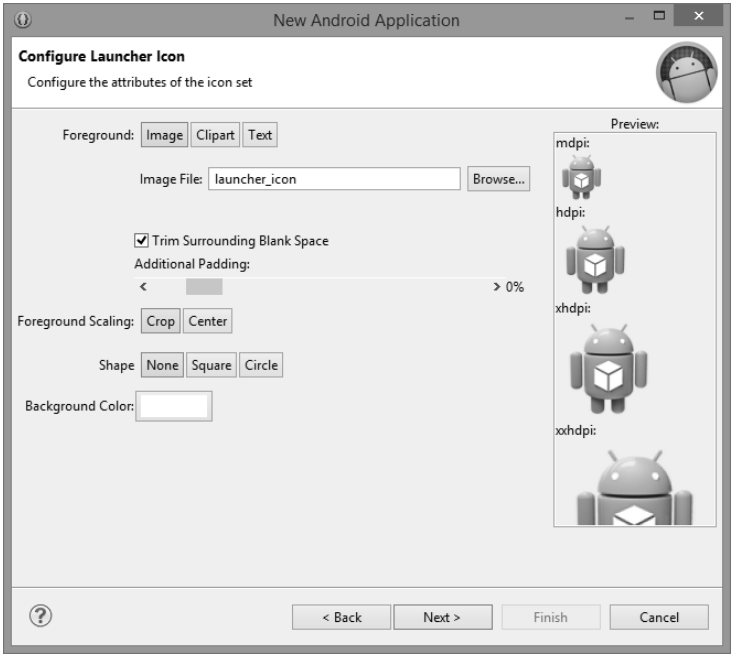
V následujícím dialogovém okně můžete změnit ikony aplikace. Jelikož je tato první aplikace cvičná a slouží k ověření správnosti instalace a konfigurace vývojového prostředí a SDK, není potřeba se v této fázi zdržovat návrhem ikon. Ponechejte implicitně nastavenou ikonu postavy Androida.



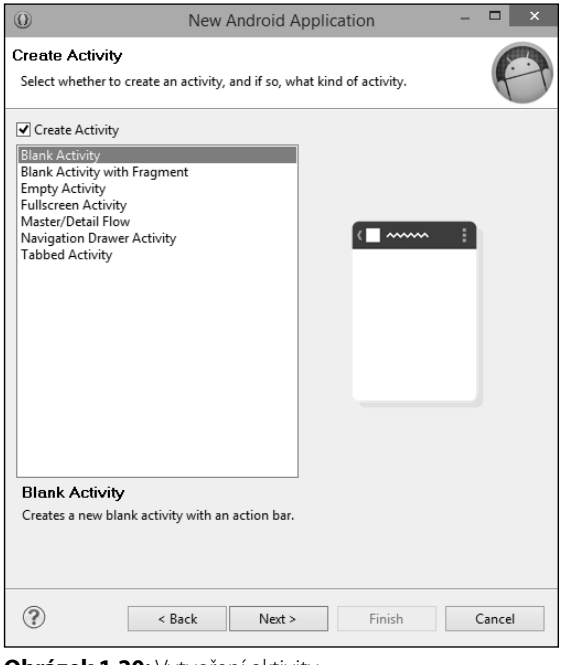
Obrázek 1.17: Vytvoření nového projektu



Obrázek 1.18: Konfigurace nového projektu



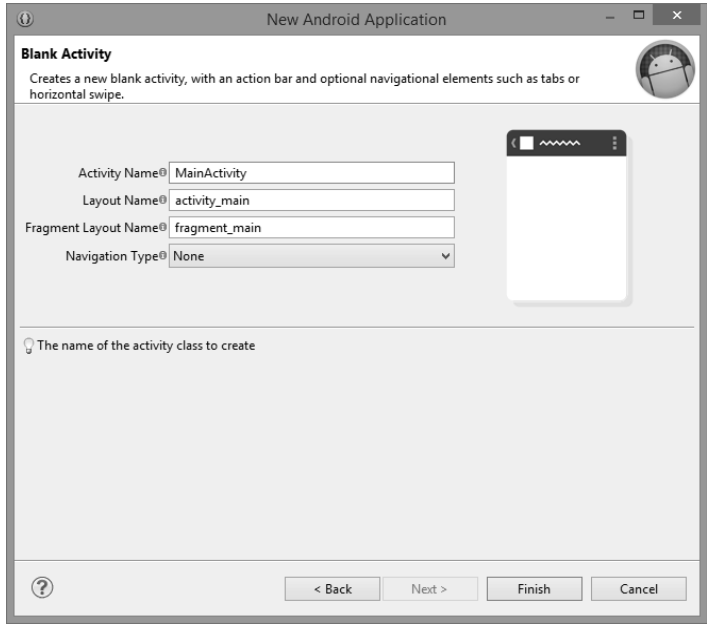
Obrázek 1.19: Ikony pro nově vytvořený projekt



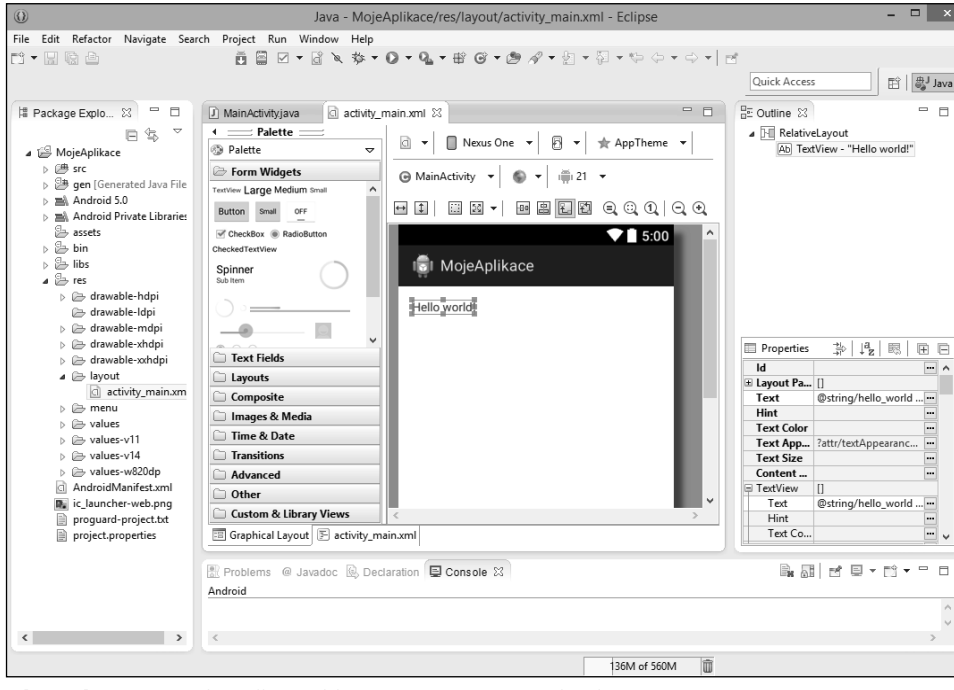
Obrázek 1.20: Vytvoření aktivity

V dalším dialogovém okně vytvoříte objekt typu **Activity**. V tomto projektu ponechejte implicitně vybranou volbu **Blank Activity**.

V následujícím dialogovém okně můžete nakonfigurovat detaily pro vybraný typ aktivity. Ponechejte implicitně nastavené hodnoty.



Obrázek 1.21: Konfigurace vybraného typu aktivity

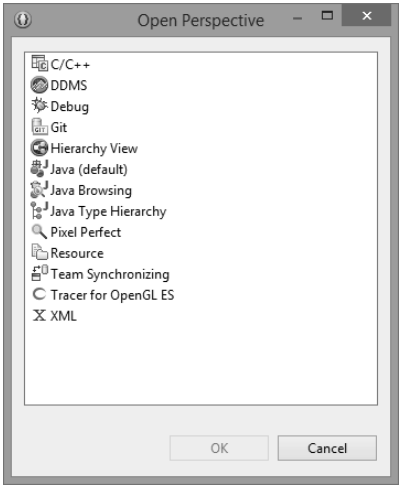


Obrázek 1.22: Projekt Hello World ve vývojovém prostředí Eclipse

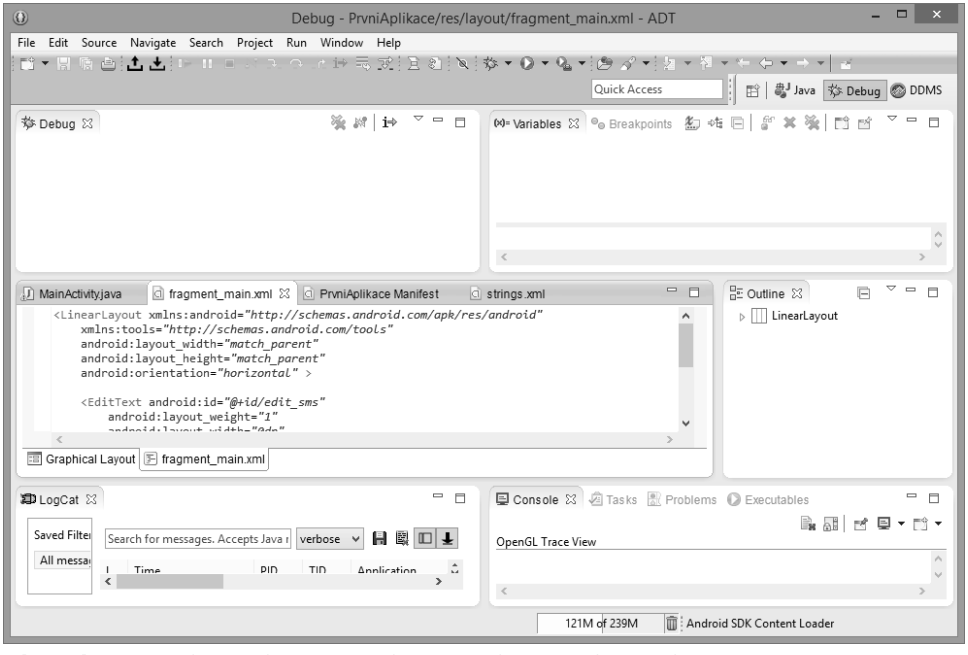
Tlačítkem **Finish** vytvořte projekt. S anatomií projektu se seznámíte v dalších kapitolách. V této fázi se pokusíte projekt spustit, nejprve na emulátoru a následně na reálném zařízení.

Expresní seznámení se s vývojovým prostředím

Po vytvoření projektu a jeho zobrazení ve vývojovém prostředí nastal vhodný okamžik na seznámení se s uživatelským rozhraním vývojového prostředí Eclipse. V levé části je úzké okno **Package Explorer**. Package (balík) slouží k vytváření nových jmenných prostorů – namespaces. Fyzická reprezentace balíku je adresář, který v souborech s příponou *.class* obsahuje přeložené třídy jazyka Java. Ve střední části je okno pro zdrojový kód, případně pro návrhové zobrazení. Vpravo a v dolní části jsou okna pro zobrazení hodnot parametrů, výpisy a podobně. Rozmístění pracovních oken vývojového prostředí se mění v závislosti na činnosti. Jiné je při návrhu prvků uživatelského rozhraní aplikace na platformě Android nazývaných widgety, jiné při psaní kódu v Javě, jiné při ladění.



Obrázek 1.23: Dialogové okno na přepínání perspektiv ve vývojovém prostředí Eclipse

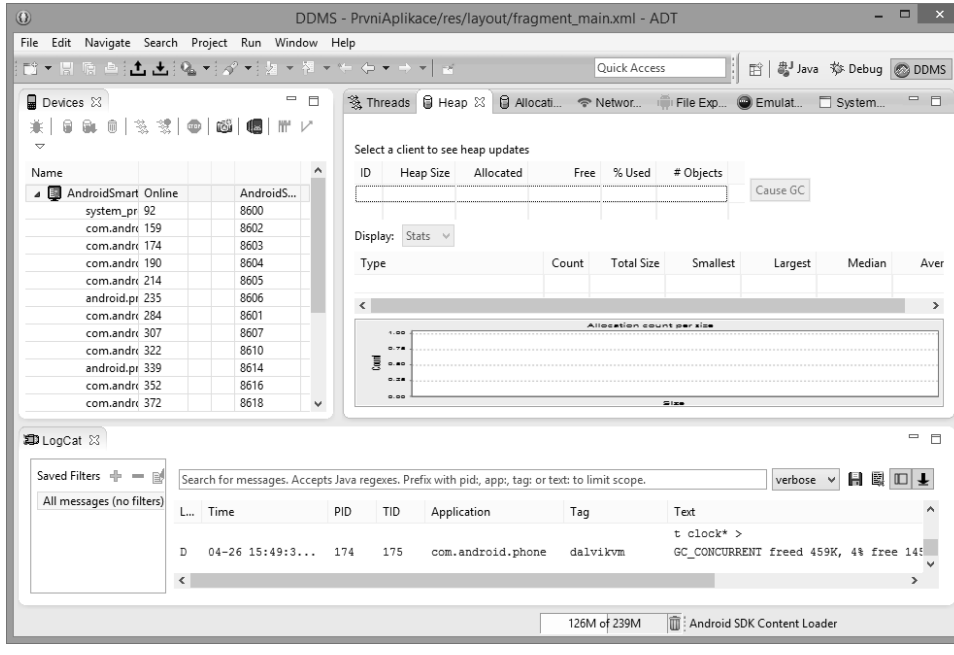


Obrázek 1.24: Rozložení oken vývojového prostředí v perspektivě Debug

Všimněte si v pravé horní části panelu nástrojů sekce implicitně se dvěma tlačítky: čtvercovým tlačítkem se symbolem plus v okně a obdélníkovým tlačítkem s nápisem **Java**. Tato sekce slouží k přepínání perspektiv, tedy rozmístění oken vývojového prostředí. Po stisknutí tlačítka se symbolem plus se zobrazí dialogové okno, pomocí něhož můžete na panel přidat tlačítka pro další perspektivy. Doporučujeme přidat perspektivy **Debug** a **DDMS** (Dalvik Debug Monitor Server).

Dalvik Debug Monitor Server (DDMS)

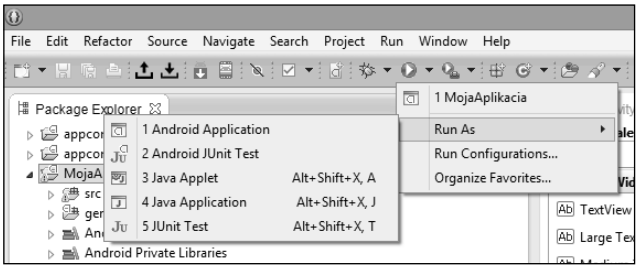
DDMS umožňuje sledovat fungování vaší aplikace na reálném zařízení se systémem Android. Vestavěný modul LogCat umožňuje v reálném čase sledovat všechny procesy, které probíhají na připojeném zařízení. Pro lepší přehlednost je možné události filtrovat a sledovat jen ty, které vás v daném okamžiku zajímají. DDMS umožňuje prohlížení alokované paměti připojeného zařízení a vytvořených objektů, kontrolu paralelně běžících vláken či sledování síťové komunikace.



Obrázek 1.25: Rozložení oken vývojového prostředí v perspektivě DDMS (Dalvik Debug Monitor Server)

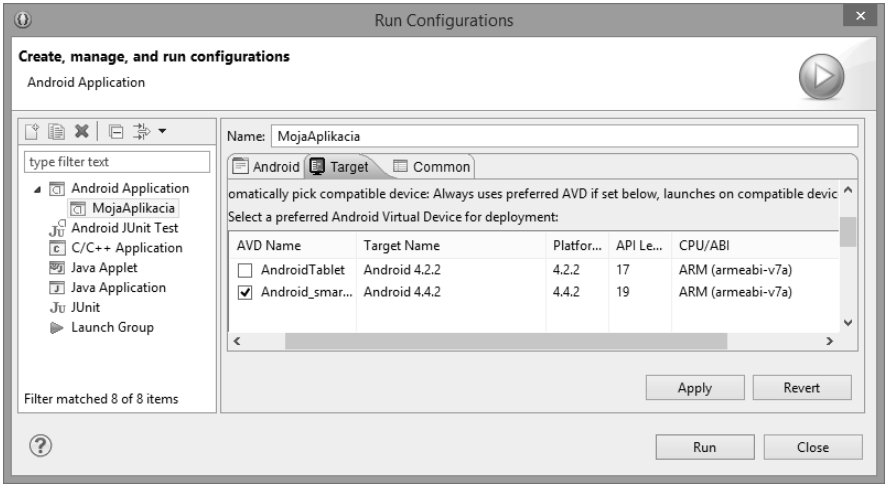
Spuštění aplikace v emulátoru

Klepnutím na zelenou šipku můžete aplikaci spustit v emulátoru mobilního zařízení. V dialogovém okně **Run As** vyberte možnost **Android Application**.



Obrázek 1.26: Spuštění aplikace

Pokud jste vytvořili více emulátorů, například v tomto případě emulátor zařízení typu chytrý telefon a emulátor zařízení typu tablet, pomocí položky **Run Configurations** zobrazíte dialogové okno s výběrem emulátoru, na kterém chcete aplikaci spustit.



Obrázek 1.27: Konfigurace spuštění aplikace

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.

```
[2014-04-06 20:37:25 - MojeAplikace] ------------------------------
[2014-04-06 20:37:25 - MojeAplikace] Android Launch!
[2014-04-06 20:37:25 - MojeAplikace] adb is running normally.
[2014-04-06 20:37:25 - MojeAplikace] Performing
  com.example.mojeaplikace.MainActivity activity launch
[2014-04-06 20:37:25 - MojeAplikace] Automatic Target Mode:
  Preferred AVD 'Android_smartfon' is not available. Launching new emulator.
[2014-04-06 20:37:25 - MojeAplikace] Launching a new emulator
  with Virtual Device 'Android_smartfon'
[2014-04-06 20:37:31 - MojeAplikace] New emulator found: emulator-5554
```

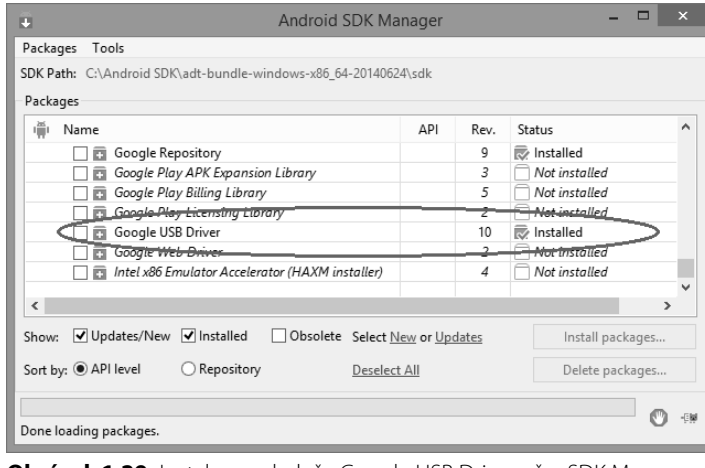
```
[2014-04-06 20:37:31 - MojeAplikace] Waiting for HOME
  ('android.process.acore') to be launched...
```



Obrázek 1.28: Spuštění aplikace v emulátoru

Spuštění aplikace na reálném zařízení

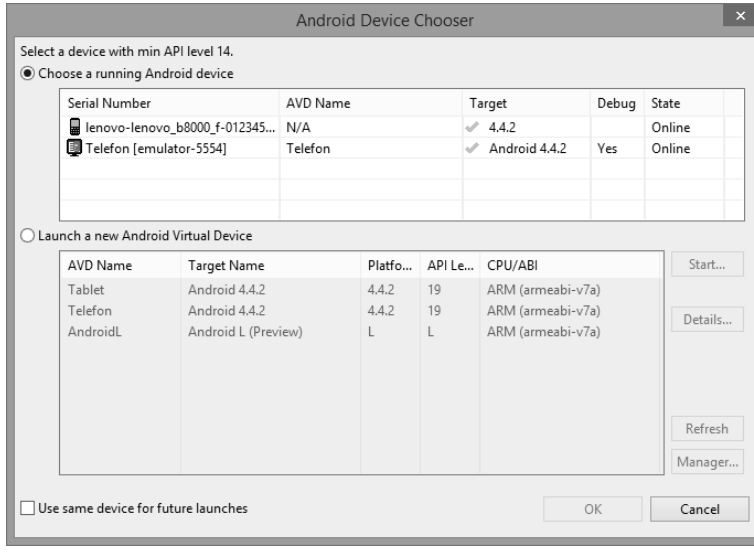
Aby bylo možné aplikaci spustit na reálném zařízení připojeném přes USB, je potřeba na vývojářském počítači nainstalovat USB ovladače pro ADB (Android Debug Bridge). Potom stačí



Obrázek 1.29: Instalace ovladače Google USB Driver přes SDK Manager

připojit zařízení, které má povolené ladění přes USB. Pro zařízení Nexus, případně některá další, postačí Google USB Driver, který doinstalujete přes SDK Manager. Pro ostatní zařízení je potřeba ovladač doinstalovat ze stránky výrobce. Některá zařízení, například Lenovo Yoga, mají možnost po připojení zařízení přes USB nastavit, aby se zařízení chovalo jako virtuální CD ROM, na kterém jsou ovladače.

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



Obrázek 1.30: Nabídka možností spuštění aplikace

Android Studio

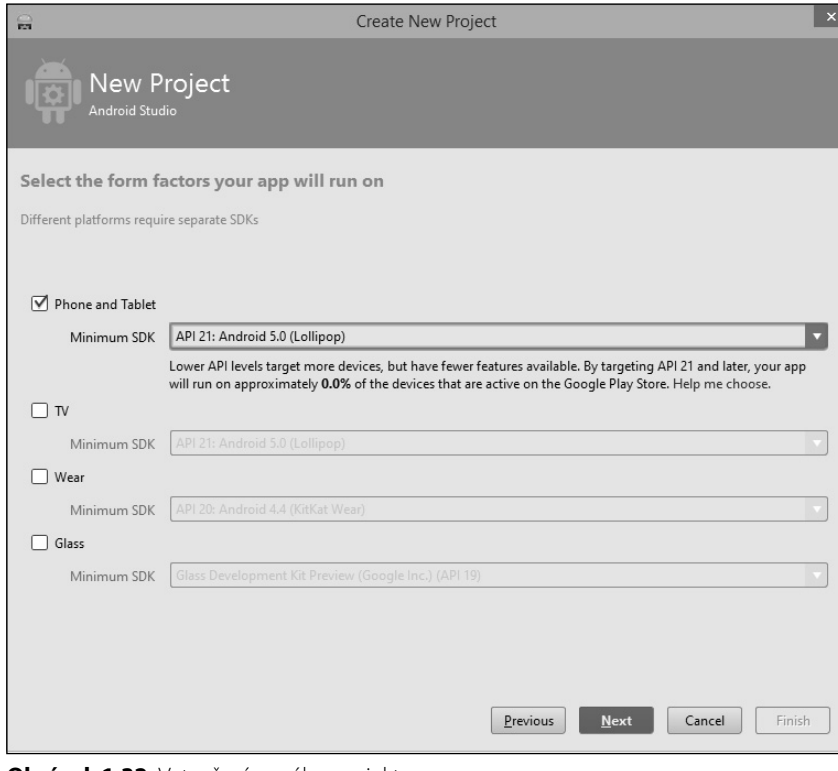
Z předchozí části o instalaci a konfiguraci vývojového prostředí Eclipse je zřejmé, že se nejedná o nástroj pro začátečníky. Také emulátory používané v Eclipse jsou velmi pomalé. Pro ilustraci: ani na vývojářském počítači se čtyřjádrovým procesorem Intel i7 nedokáže emulátor běžet tak rychle jako nejpomalejší reálné zařízení.

Na druhé straně je tu velký zájem o vývoj aplikací pro tuto nejpoužívanější platformu. Východiskem by mohlo být vývojové prostředí Android Studio vyvíjené Googlem na bázi komunitní platformy IntelliJ. Jednodušší návrh uživatelského rozhraní pro různá rozlišení obrazovky přispívá k výraznému zvýšení produktivity práce.

V době psaní publikace byl tento nástroj ve verzi Early Access Preview, tedy před finální verzí. Námi použitá verze 0.8.0 byla stabilní a plně funkční.



Obrázek 1.31: Úvodní obrazovka nástroje Android Studio

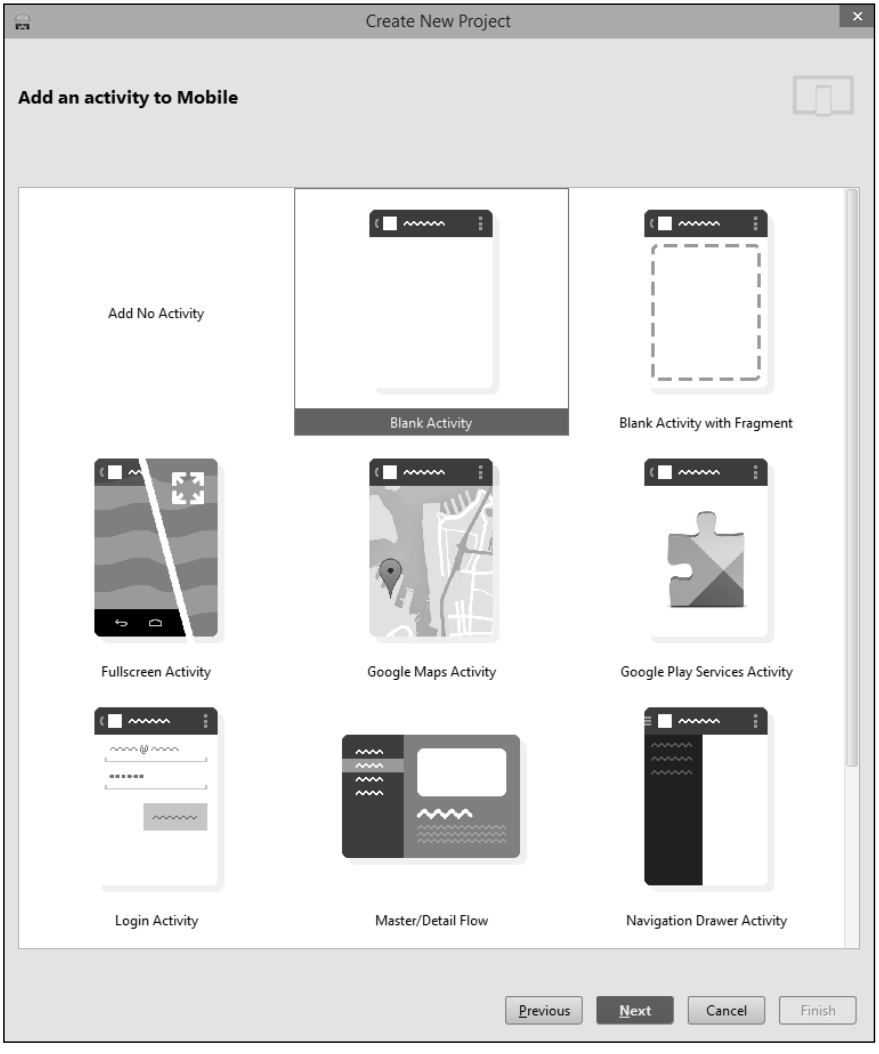


Obrázek 1.32: Vytvoření nového projektu

Instalace Android Studia je jednoduchá a proběhne doslova na jedno klepnutí. Stačí stáhnout instalační soubor a spustit ho. S výjimkou Javy (JDK od Oracle) není potřeba nic doinstalovávat ani konfigurovat. Součástí instalace jsou i kvalitní emulátory zařízení s operačním systémem Android. Po nainstalování se zobrazí okno se základní nabídkou, jejíž součástí je i vytvoření nového projektu.

Postup vytvoření nové aplikace je analogický s postupem v Eclipse. Android Studio využívá méně dialogových oken, která jsou však přehlednější a komplexnější.

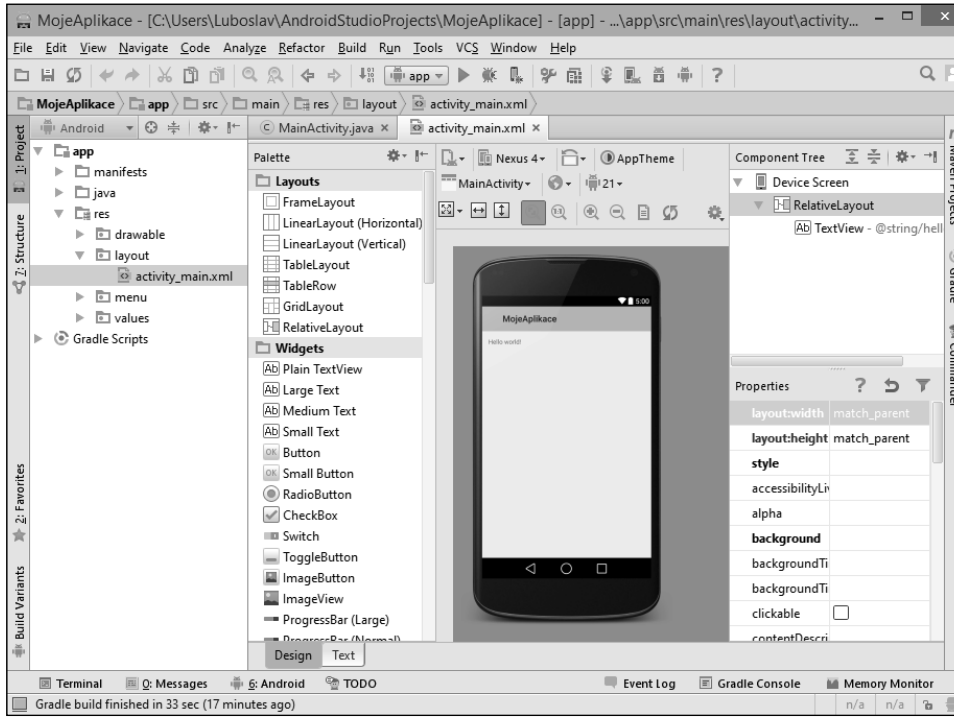
Můžete zvolit několik typů pro hlavní aktivitu aplikace včetně mapové či aktivity typu master-detail.



Obrázek 1.33: Výběr typu hlavní aktivity

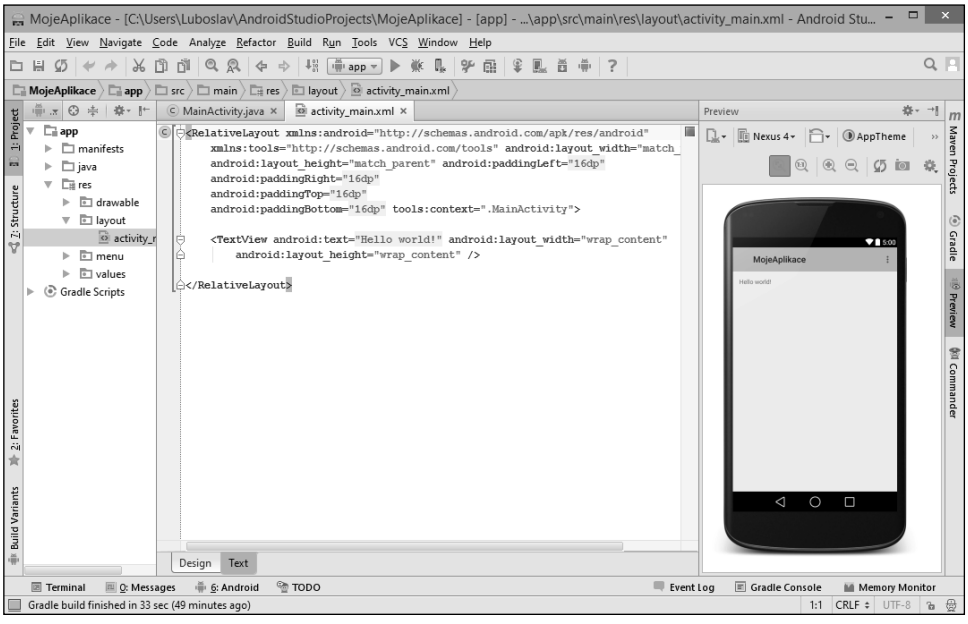
Další postup se odvíjí od nastaveného typu hlavní aktivity. Například pokud jste zvolili typ master-detail, systém vás vyzve k definování názvu položky a skupiny položek, přičemž skupina položek je plurálem od názvu položky; například objednávka/objednávky a podobně.

Podobně jako v Eclipse je i v Android Studiu možné navrhovat design buď v návrhovém módu, nebo přímo v XML souboru. Pokud zvolíte druhou možnost, Android Studio vám automaticky zobrazuje náhled změn. Na rozdíl od Eclipse se nemusíte stále přepínat mezi okny Design a XML.

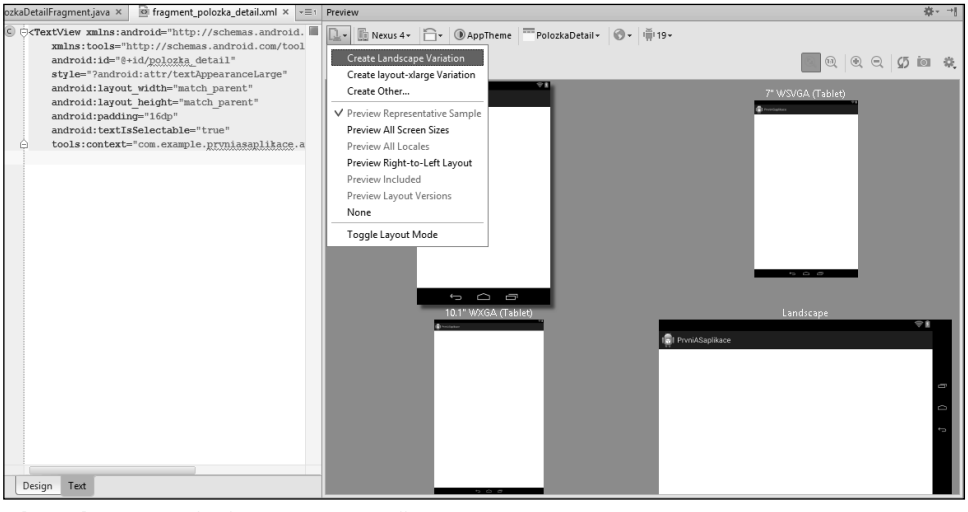


Obrázek 1.34: Uživatelské rozhraní Android Studia

Jelikož je Android implementovaný 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.



Obrázek 1.35: Při tvorbě uživatelského rozhraní přímo v XML kódu se automaticky zobrazuje jeho náhled



Obrázek 1.36: Mód zobrazení Preview All Screen Sizes

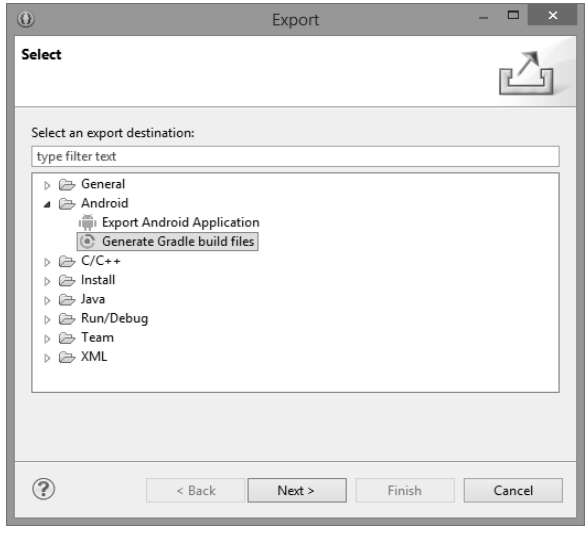
Import projektů z Eclipse

Projekty vytvořené v Eclipse můžete jednoduše importovat do Android Studia. V Eclipse klepněte na **File** → **Export**. Zobrazí se dialogové okno **Export**. Ve složce **Android** zvolte **Generate**

Gradle build files. Vyberte projekt, který chcete exportovat z Eclipse za účelem následného importu do Android Studia.



Poznámka: Nejnovější verze Android Studia umožňuje už přímý import ADT projektů z Eclipse bez nutnosti předchozího exportu.



Obrázek 1.37: Export projektu z Eclipse. Dialogové okno je z vývojového prostředí Eclipse a aktivuje se volbou File → Export.

Embarcadero RAD Studio XE6

Komu by se zdál produkt Embarcadero RAD Studio neznámý, stačí připomenout dva pojmy: Delphi a Borland C++. Vývojové prostředí Delphi, využívající populární programovací jazyk Pascal, vzniklo ve společnosti Borland. Ta se později přejmenovala na Inprise a po pár měsících se znovu vrátila k osvědčenému názvu Borland. Firma se však začala více věnovat ALM (Application Lifecycle Management) řešením a dosavadní vlajkové produkty, tedy vývojářské nástroje a populární databázi InterBase, odstavila na vedlejší kolej, přesněji je předala dceřiné společnosti CodeGear. Tuto nakonec koupila společnost Embarcadero, která se specializovala na vývoj databázových nástrojů pro velké firmy.

Tato část by také mohla mít název „jeden kód pro všechny platformy". Vývojáři v současnosti bojují se dvěma hlavními problémy – efektivitou a různorodostí platforem, ať už serverových, klientských nebo mobilních. Vývojové prostředí RAD Studio od společnosti Embarcadero, které je momentálně k dispozici ve verzi XE6, se snaží řešit oba naznačené problémy, jelikož do značné míry vzájemně souvisí. Vývoj a správa aplikace pomocí specifických nástrojů pro každou platformu jsou nákladné a časově náročné. Zkratka RAD znamená Rapid Application Development a nová verze umožňuje efektivně ve vizuálním prostředí vytvářet společný kód aplikací určených pro platformy Windows, Android, iOS a Mac OS X, všechno v rámci jed-

noho časového plánu bez nutnosti obětovat cokoliv z výkonu aplikací, jelikož tyto jsou kompilovány do nativního kódu.

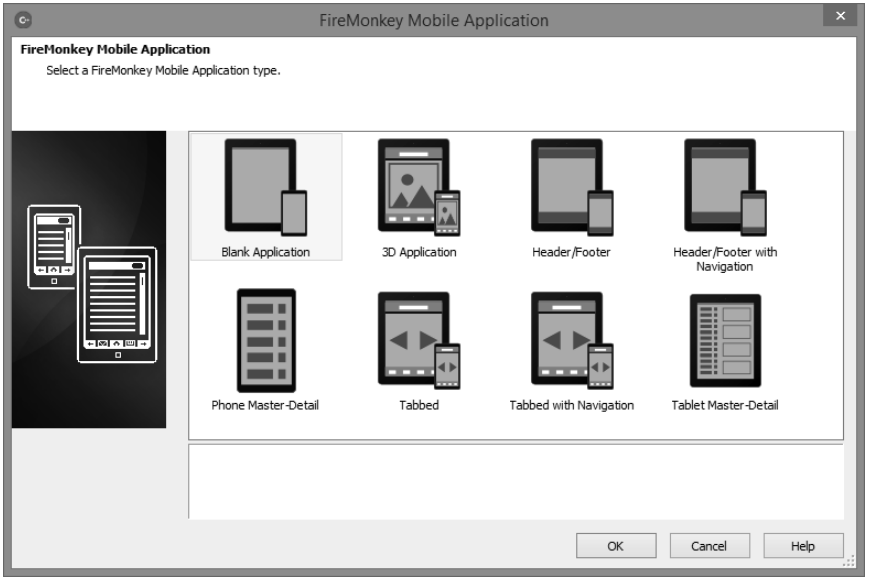
Firmy zabývající se vývojem aplikací si snadno dokáží spočítat, jakým koeficientem by se u nich násobila produktivita vývojářských týmů, pokud by mohly pro nejrůznější platformy použít jeden nástroj, jednotný programovací jazyk a jednotný framework. Multiplatformovost oceníte i po ukončení vývoje v dalších fázích životního cyklu aplikace.

Vizuální návrh aplikací v C++ pro Android

Verze XE6 přináší vizuální aplikační vývojové prostředí jazyka C++ pro platformu Android, což je důležité pro migrující vývojáře, jelikož nativně se aplikace pro Android vytvářejí ve vývojovém prostředí Eclipse v programovacím jazyku Java. RAD Studio XE6 však umožňuje aplikaci jednoduše portovat na všechny nejpoužívanější platformy. Odhadujeme, že bude možné využít přibližně 90 procent společného aplikačního kódu a 60–70 procent návrhu uživatelského rozhraní. Je logické, že bude potřeba změnit uspořádání vizuálních prvků při migraci aplikace z Windows či Mac OS na tablety s iOS či Androidem. Mění se i filozofie ovládání směrem k dotykům. Migrace aplikace na chytré telefony bude mnohem složitější, v mnoha případech bude potřeba kompletně změnit filozofii ovládání. Samozřejmostí je podpora v současnosti nejrozšířenějších verzí Androidu včetně 4.4 KitKat.

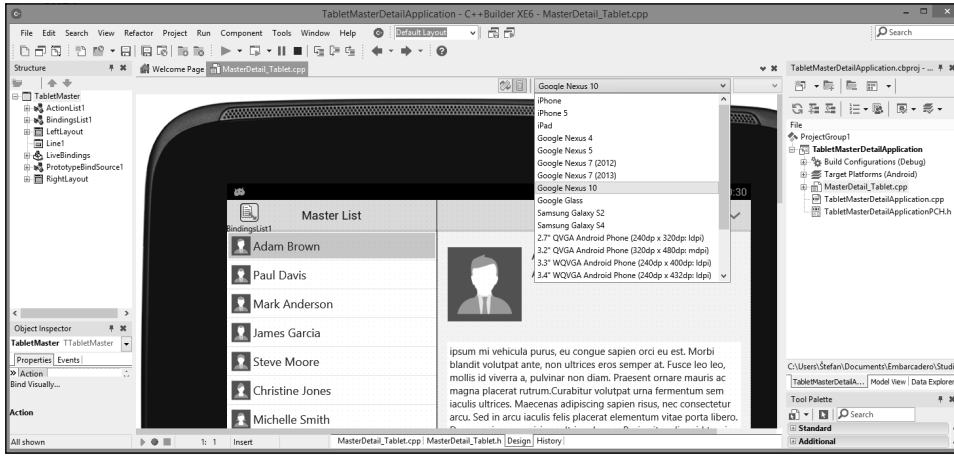
FireMonkey

Cílem tvůrců knihovny Fire Monkey je vývoj rychlých a vizuálně působivých obchodních aplikací na platformách Windows, Mac a iOS. Výsledkem překladu je nativní kód využívající

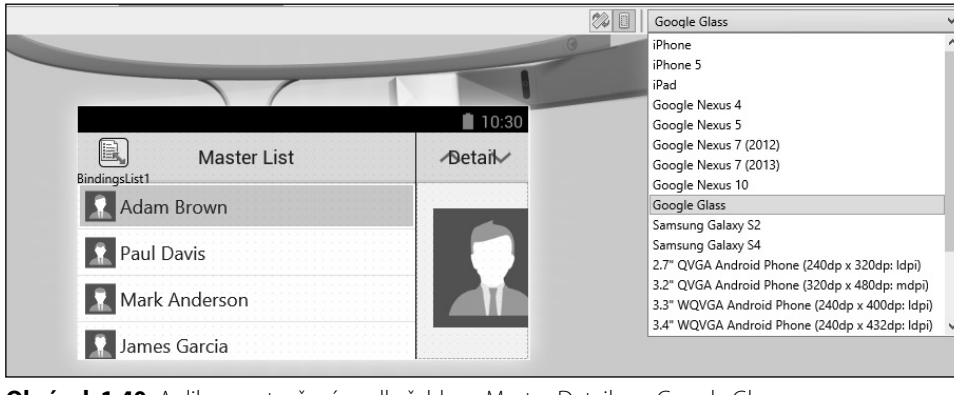


Obrázek 1.38: Šablony pro FireMonkey mobilní aplikace

nejen procesor, ale i grafický akcelerátor, což se výrazně projeví na rychlosti graficky bohatých aplikací. Hardwarovou akceleraci je možné využít i v mobilních zařízeních. FireMonkey umožňuje vývoj komplexních aplikací nejen s vysokým rozlišením grafické prezentační vrstvy na bázi vektorové grafiky, ale pro lepší znázornění „byznys grafiky" je možné využít i trojrozměrné (3D) zobrazení, animaci a širokou škálu obrazových efektů.



Obrázek 1.39: Aplikace vytvořená podle šablony Master-Detail pro tablet



Obrázek 1.40: Aplikace vytvořená podle šablony Master-Detail pro Google Glass

Bez ohledu na to, pro kterou platformu aplikaci vytváříte a který z programovacích jazyků C++ nebo Delphi (populární programovací jazyk vycházející ze syntaxe Pascalu) použijete, kompilátory v RAD XE6 vždy generují nativní, binárně spustitelné soubory pro procesory Intel nebo ARM. To je zárukou vysokého výkonu aplikace a díky rychlé odezvě i pozitivní uživatelské zkušenosti.

App Tethering

Aby se zjednodušila migrace na mobilní platformy a adaptace vývojářů využívajících knihovny vizuálních komponent VCL pro Windows bez vynaložení velkého úsilí, XE6 přináší nové komponenty označené „App Tethering". Tento pojem se dá nejvýstižněji přeložit jako provázání aplikací. V praxi to znamená možnost rozšířit existující aplikace využívající VCL i na mobilní platformy včetně nositelných zařízení, aniž by bylo nutné migrovat celou aplikaci pro Windows. Bude samozřejmě potřeba přizpůsobit uživatelské rozhraní a pro platformy s menším displejem adaptovat jen takové vlastnosti, které dávají smysl na mobilních zařízeních.

Xamarin MonoTouch a Mono for Android

Společnost Xamarin byla založena roku 2011 a zaměřuje se na vývoj multiplatformových nástrojů založených na Mono open-source – Mono Touch (iOS) a Mono for Android. Open-source projekt Mono byl uveden roku 2001 jako open-source verze výkonného prostředí Microsoft .NET. Knihovna Mono Touch byla uvedena na trh roku 2009, Mono for Android následně v březnu 2011. Nástroj Mono umožňuje vývojářům v .NET zaměřeným na jazyk C# vytvářet aplikace pro iOS a Android.

Aplikace vytvořené v nástroji Mono Touch jsou předkompilované do nativní JIT (Just in Time) kompilace a vytváří tím vnořené výkonné prostředí v rámci nativní aplikace. Xamarin také plánuje vydat nový designérský UI nástroj pro Android a poskytnout tak nativní vzhled a dojem z androidových aplikací.

Mnohé z existujících komponent knihoven pro .NET jsou dostupné i pro Mono a Xamarin. Tyto umožňují řešení včetně grafiky, analýzy a vrstvy datové abstrakce. Nástroje Mono Touch a Mono for Android jsou dostupné ve formě bezplatné permanentní trial verze, která nepovoluje publikování aplikací. Cena licence za profesionální verzi začíná na 400 USD za produkt a rok.

Jazyky C# a .NET mají významné zastoupení v podnikové sféře vývoje aplikací. Xamarin se zaměřuje právě na tyto vývojáře, kteří potřebují vytvářet aplikace pro iPhone, iPad a zařízení s Androidem.

Game Maker Studio na vývoj her

Specializovaný nástroj Game Maker Studio na vývoj her získáte na adrese *www.yoyogames.com/studio*. Jedná se o multiplatformní prostředí na jednoduchý vývoj her a aplikací. Vytvoření hry je možné publikovat pro všechny populární platformy: Android, iOS, Windows 8 a Windows Phone 8, HTML5, Facebook i klasický Windows desktop.



Poznámka: Game Maker Studio Edice Standard je bezplatná, verze Pro, která disponuje pokročilejšími vlastnostmi, například správou textur či možností ladění na mobilním zařízení s Androidem, je k dispozici za 99 dolarů.

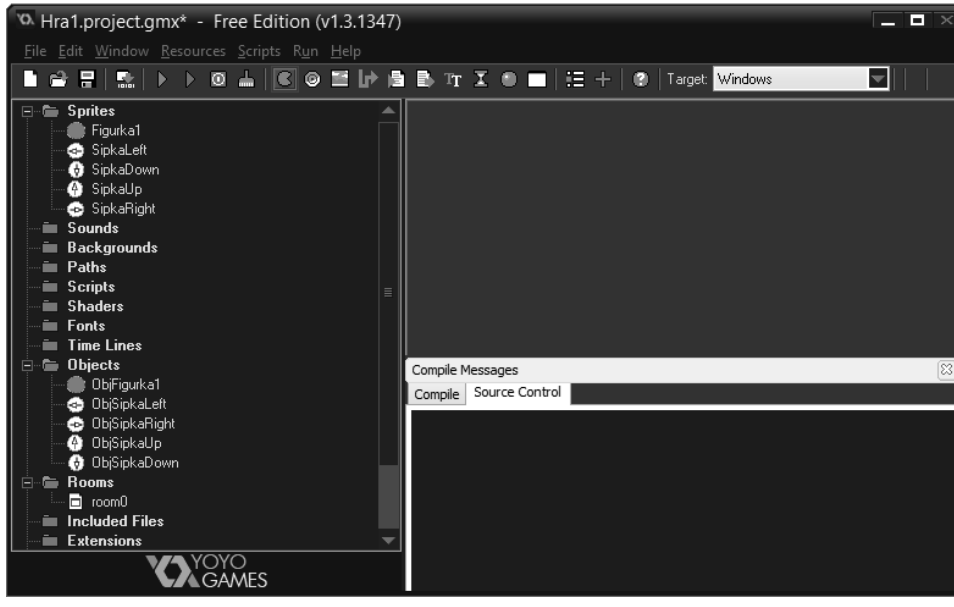


Tip: V operačním systému Windows doporučujeme spustit aplikaci Game Maker Studio jako správce.

Příklad vytvoření nejjednodušší hry

Cílem příkladu je postup vytvoření projektu hry, proto jako námět použijeme nejjednodušší „atrapu" hry, jaká se vůbec dá vymyslet – hra umožňuje řídit pohyb figurky pomocí kurzorových kláves bez jakéhokoliv herního algoritmu.

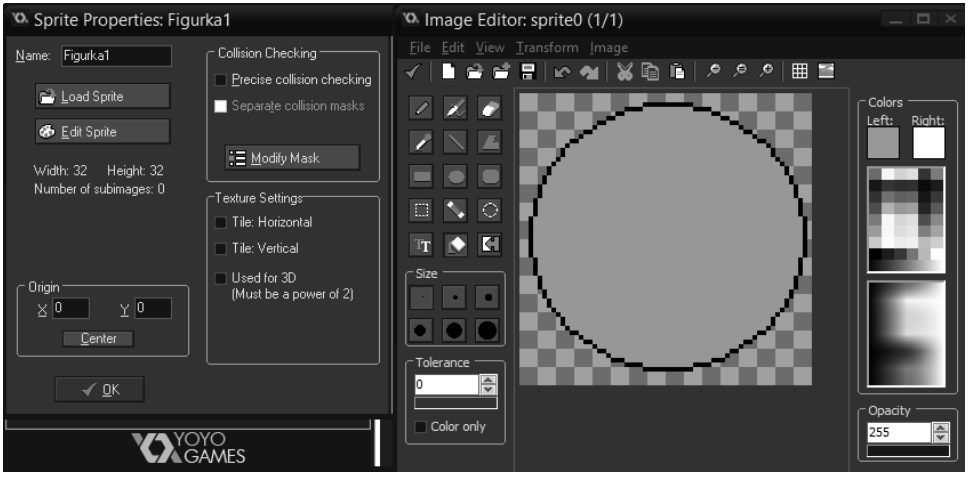
Vytvořte nový projekt s vhodným názvem. Po spuštění si všimněte vlevo složek pro jednotlivé typy objektů, které je možné v nástroji Game Maker Studio vytvořit. Nejprve vytvořte hráče. Klepněte na panelu nástrojů na ikonku se symbolem zelené figurky PacMan.



Obrázek 1.41: Game Maker Studio – složky pro jednotlivé typy objektů

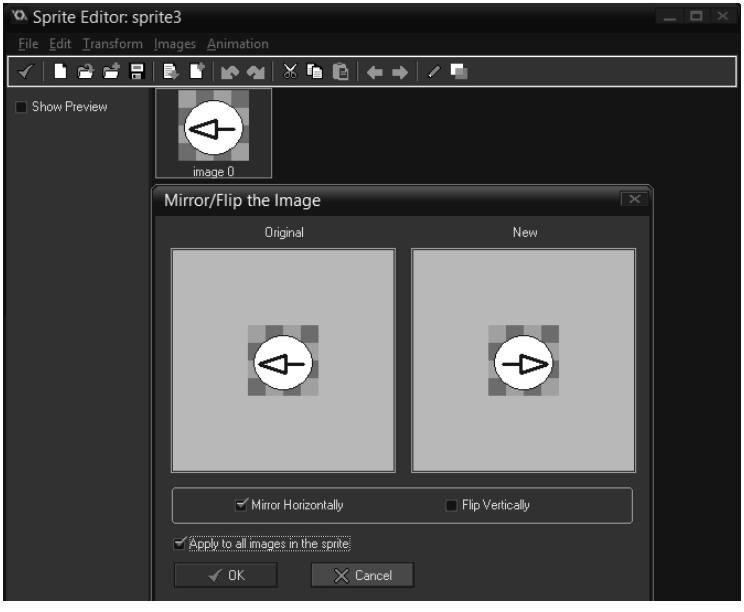
Zobrazí se dialogové okno pro vytvořený sprite (v doslovném překladu skřítek, v terminologii Game Maker Studia grafický návrh herních figurek). Dialogové okno obsahuje i informace o parametrech. Objekt hry, který právě vytváříte, nejprve vhodně pojmenujte. Pomocí tlačítka **Edit Sprite** a položky nabídky **New** vytvořte nový objekt, kde zadáte jeho rozměry. Následně můžete klepnutím na prázdný obrázek nakreslit objekt hry, například postavičku, překážku, stěnu, podlahu a podobně. Doporučujeme pomocí tlačítka s lupou zvětšit měřítko zobrazení. V našem příkladu jsme vytvořili jako symbol figurky zelený kroužek.

Stejným postupem je potřeba vytvořit i ovládací prvky, jelikož zařízení s Androidem nedisponují klávesnicí, pouze dotykovou obrazovkou, na které je potřeba ovládací prvky, například šipky, zobrazit.



Obrázek 1.42: Vytvoření objektu typu sprite

Při vytváření ovládacích prvků se šipkami můžete s výhodou využít funkce **Duplicate** v místní nabídce objektu ve složce **Sprites** v levém úzkém svislém okně. Stačí vám tedy vytvořit jednu šipku na ovládání směru pohybu herní figurky a zbylé tři vytvoříte jejím klonováním pomocí funkce **Duplicate** a následnou vhodnou transformací, například pootočením nebo vytvořením zrcadlového obrazu.



Obrázek 1.43: Vytváření šipek pro ovládání směru pohybu figurky klonováním



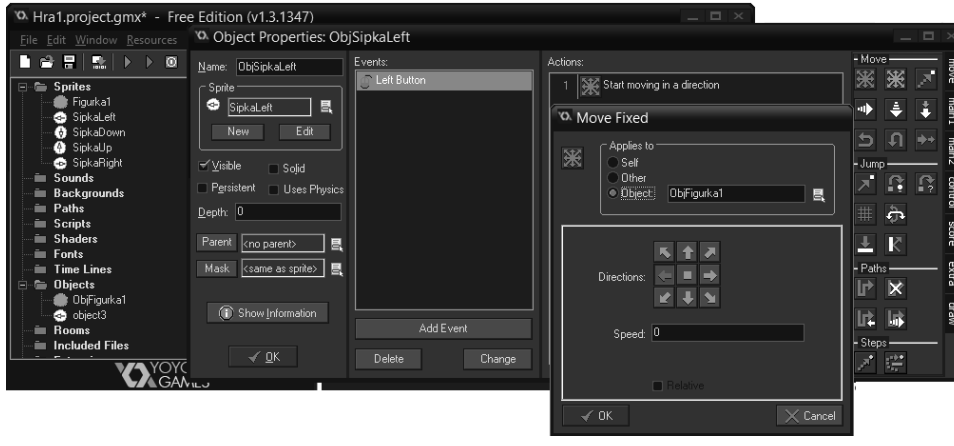
Dalším krokem je na základě figur (sprite), tedy grafických návrhů prvků hry, vytvořit objekty. Objekty je možné vytvářet pomocí ikony se symbolem zeleného kruhu. Do objektu přidejte předtím vytvořený sprite. Parametr **Depth** udává polohu hráče v ose Z.

Pro objekty je potřeba definovat obsluhu události klepnutí na objekt hry, například na šipku. Pomocí tlačítka **Add Event** vytvořte událost. Jako ekvivalent klepnutí se hodí událost **Mouse** → **Left button**.

Obrázek 1.44: Nabídka typů událostí pro objekt

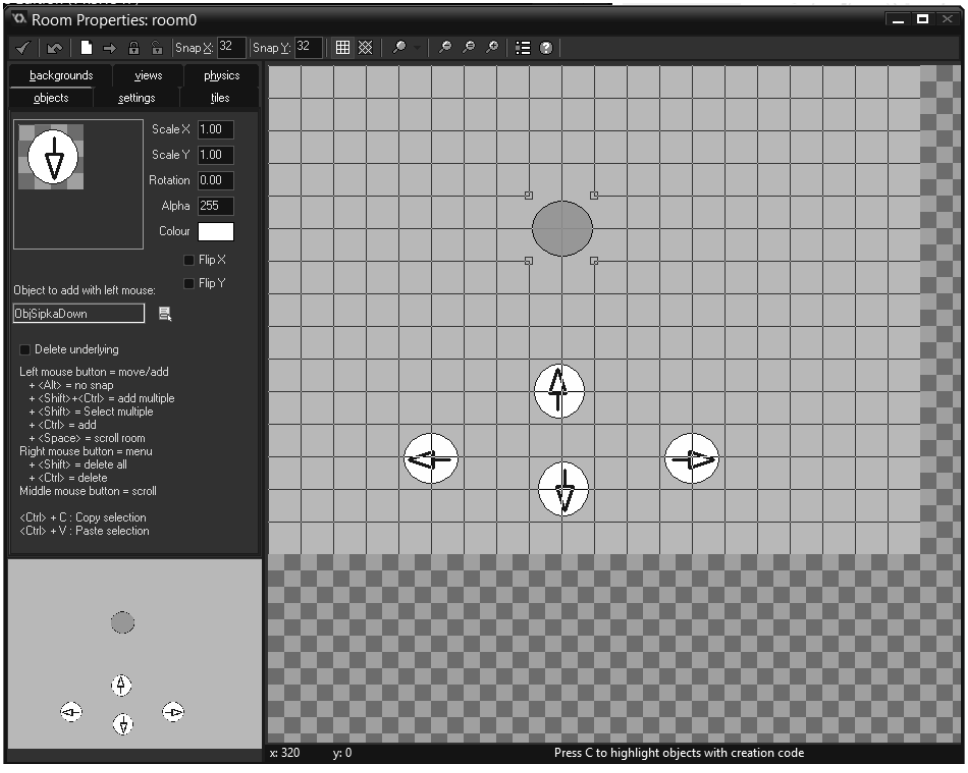
Akci definujete vizuálně v okně **Actions**. Přesuňte na plochu objekt **Move** (ikona se zelenými šipkami, první v sekci **Move**). Následně specifikujte požadovaný směr. Definujte rychlost pohybu, například 5, parametr **Applies to** nastavte na volbu **Object** a zadejte název objektu – figurky, která se má pohybovat.

Herní figurce definujte událost **Mouse** → **GlobalLeftRelease**. Jako akci přidejte ikonu stop, která je mezi směrovými šipkami, a rychlost 0.



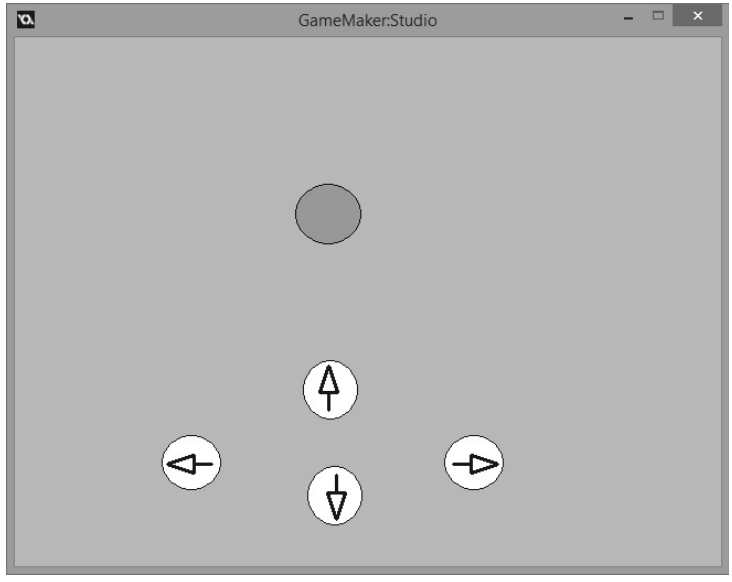
Obrázek 1.45: Vytváření objektů a definování akcí

Ve složce **Rooms** vytvořte pomocí volby **Create Room** novou herní místnost – herní plán. Místnost vhodně pojmenujte, například Level 1. Objekt herní figurky umístěte na vhodné místo hracího plánu. Na vhodné místo umístěte také šipky na ovládání pohybu herní figurky. Dbejte na ergonomii, aby bylo možné hru dobře ovládat.



Obrázek 1.46: Vytváření herního plánu

Hru spustíte pomocí tlačítka **Run the Game**. Hra se spustí v okně na vývojářském počítači, kde ji můžete odladit a následně ji můžete portovat na požadovanou platformu, v tomto případě na Android. Abyste mohli hru portovat, musíte mít nainstalované Android SDK.



Obrázek 1.47: Ladění hry

Už z této jednoduché ukázky vidíte, kolik námahy vám nástroj Game Maker Studio ušetřil v porovnání s tím, kdybyste museli všechno vytvářet a programovat klasicky v Javě.

KAPITOLA 2

Architektura

V této kapitole:
- Operační systém Android
- Android 5.0 Lollipop (API 21)
- Jakou nejstarší verzi Androidu podporovat v aplikaci
- Stručně o architektuře Androidu
- Bezpečnost na platformě Android
- Základní součásti aplikace pro Android
- Aktivita a její životní cyklus
- Úvod do asynchronního programování
- Možnosti ukládání údajů

Operační systém Android

Android je open-source platforma na bázi Linuxu určená hlavně pro mobilní zařízení, tedy chytré telefony, tablety, fotoaparáty a navigace. V současnosti se stále více přidávají i konvertibilní zařízení, tj. tablety s odpojitelnou klávesnicí, které se hodí i k méně náročnému podnikovému nasazení. Platforma Android se objevuje i v televizních přijímačích a různých dalších zařízeních.

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, které podporují 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). Na druhé straně, když Google vydá aktualizaci systému, ta není hned dostupná pro všechna zařízení, 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 Applem, kde je celý vývoj hardwaru v režii jedné firmy, a zároveň to představuje problém, protože aplikace běží na přístrojích s různým rozlišením displeje a různým výkonem procesoru a grafiky. V praxi to znamená různý komfort uživatelského rozhraní. 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 a založili ji čtyři zakladatelé 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 inteligentních telefonů podíl zanedbatelného 0,5 %. Únor 2009 přinesl Android 1.1 jako aktualizaci pro G1. V dubnu 2009 spatřil světlo světa první masový Android ve verzi 1.5 (Cupcake).

Jednotlivé verze operačního systému Android mají kromě číselného označení i kódové – tím je název zákusku.



Obrázek 2.1: Loga verzí Androidu

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.



Poznámka: Widgety jsou miniaplikace, které se dají umístit na domovskou obrazovku, s možností periodické aktualizace.

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, živé tapety, 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, USB modem, 3D galerie a sdílení kontaktů přes Bluetooth, možnost vytvořit Wi-Fi hotspot, 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 USB příslušenství.

Android 4.0 IceCream Sandwich (*„ruská" zmrzlina – 19. 10. 2011*) – tato verze odstraňuje rozdíly mezi verzemi pro mobilní zařízení a verzemi pro tablety. Novinky: 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 účtů, 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, fullscreen režim pro několik aplikací, podpora krokoměru a infračerveného ovládání, inteligentní vypínání nepotřebných procesů na pozadí.

O dynamickém vývoji Androidu svědčí i to, že odstup mezi verzemi je jen několik měsíců. 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
Android 3.2 Honeycomb	13
Android 4.0–4.0.2 IceCream Sandwich	14
Android 4.0.3–4.0.4 IceCream 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 4.4 KitKat Wear	20
Android 5.0 Lollipop	21

Pozornější čtenáři určitě postřehli, že názvy jednotlivých verzí Androidu jdou za sebou abecedně (Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, IceCream Sandwich, Jelly Bean, KitKat, Lollipop).

Android 5.0 Lollipop (API 21)

V době psaní publikace uvolnil Google SDK pro API 21, které mělo kódové označení Android L. S příchodem verze se označení změnilo na Android 5 Lollipop. Je důležité se seznámit s designovými i funkčními novinkami, abyste je mohli využívat v aplikacích. Spolu s SDK byly uvolněny i instalační obrazy, pomocí kterých bylo možné nový Android L nainstalovat na Nexus 5 a Nexus 7 model 2013.



Obrázek 2.2: Uživatelské rozhraní Android 5.0 Lollipop

Poznámka: Uživatelé si velmi rychle zvyknou na novinky v nové verzi operačního systému a začnou je očekávat i v aplikacích.

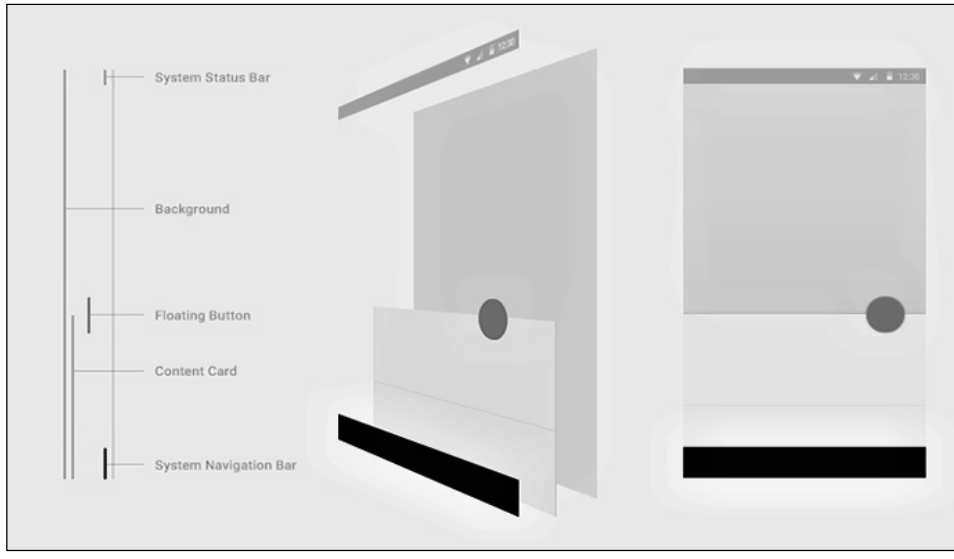
Jednou z velkých změn v nových verzích Androidu je nahrazení dosud používaného virtuálního stroje Dalvik, pod kterým běžely aplikace, novým enginem ART (Android Runtime).

Poznámka: Na některých zařízeních s Android 4.4 KitKat bylo možné nastavit ART jako zkušební funkcionalitu pro experimenty. V možnostech Pro vývojáře aktivujte funkci Výběr modulu runtime a následně Použít program ART.

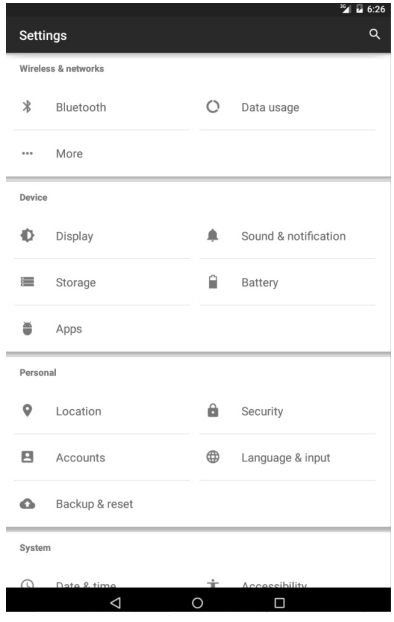
Hlavním přínosem ART je možnost kompilovat část kódu aplikací už při jejich instalaci. To významně urychlí spouštění aplikací a sekundárně prodlouží výdrž baterie, jelikož Dalvik proces kompilace vykonával při každém spuštění.

Například design aplikace Nastavení je praktickou ukázkou designových doporučení aplikací pro Android 5.0. Z designových novinek je potřeba zmínit inovovaný panel oznámení, který 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, kdy zařízení pomocí senzoru přizpůsobilo jas aktuálním světelným podmínkám. Android L přichází s novinkou s 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.

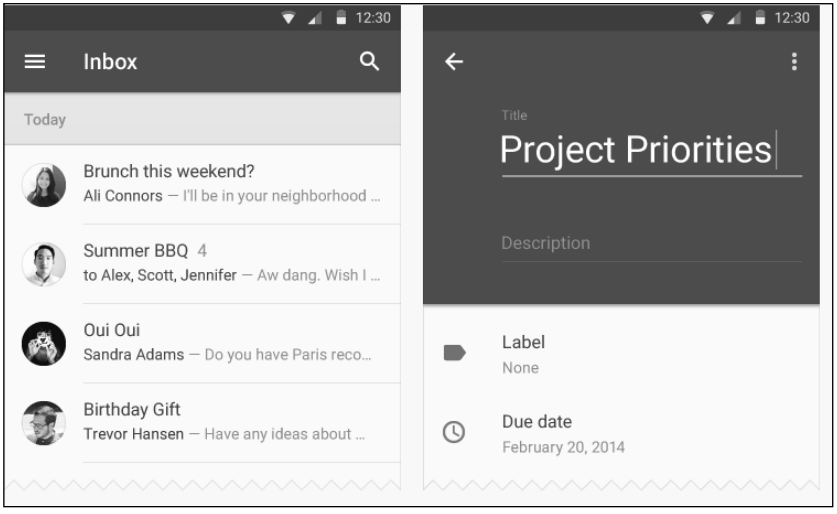


Obrázek 2.3: Design nové verze označovaný jako Material Design je přehlednější a pomocí různých vizuálních efektů, hlavně stínu, se snaží vzbudit zdání třetího rozměru

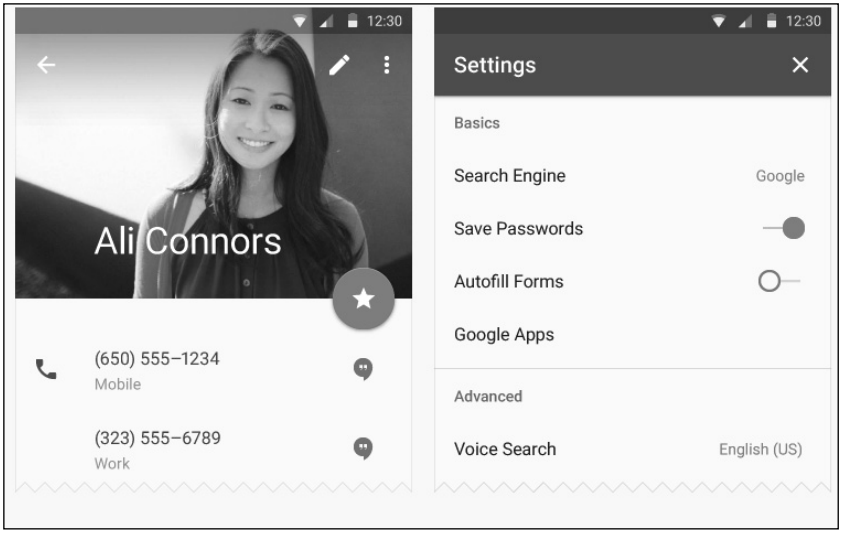


Obrázek 2.4: Design aplikace Nastavení je praktickou ukázkou designových doporučení pro aplikace pro Android 5.0

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.

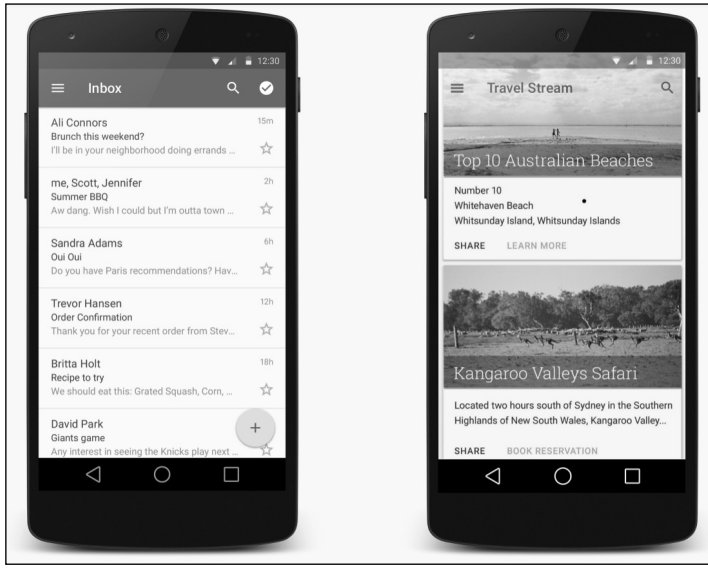


Obrázek 2.5: Příklad designu aplikace



Obrázek 2.6: Příklad designu aplikace

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

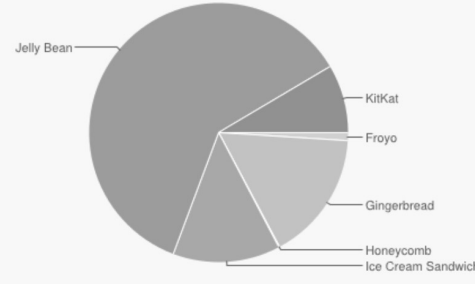


Obrázek 2.7: Nové prvky uživatelského rozhraní RecyclerView vlevo a CardView vpravo

Jakou nejstarší verzi Androidu podporovat v aplikaci

Doba životnosti mobilního telefonu je maximálně dva až tři roky, podobně je tomu i u tabletů, takže přístrojů se staršími verzemi rapidně ubývá. V době psaní publikace byla k dispozici nejnovější verze Android 5.0 (Lollipop), které odpovídá API 21.

Version	Codename	API	Distribution
2.2	Froyo	8	1.0%
2.3.3 - 2.3.7	Gingerbread	10	16.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	13.4%
4.1.x	Jelly Bean	16	33.5%
4.2.x		17	18.8%
4.3		18	8.5%
4.4	KitKat	19	8.5%



Obrázek 2.8: Podíl zařízení s jednotlivými verzemi Androidu. Stav ke květnu 2014.

Po vytvoření projektu je v aktuální verzi vývojového prostředí Eclipse implicitně nastavená jako minimální verze Android 2.2 (Froyo) – verze API 8. V současnosti podíl zařízení s verzí

2.2 klesl pod jedno procento, pro nové aplikace už je nemá smysl podporovat. Stále zajímavý je 16,2% podíl verze 2.3. Verze 3.2 Honeycomb určená pro tablety se příliš nerozšířila.



Poznámka: Nové aplikace proto až na výjimky, které chcete směrovat uživatelům starších přístrojů, stačí vyvíjet pro verzi 4.0 IceCream Sandwich a vyšší. Můžete tak bez omezení používat nové rysy uživatelského rozhraní.

Knihovny pro dopřednou kompatibilitu

Google se neustále snaží zdokonalovat uživatelské rozhraní Androidu. Úspěšní vývojáři pozorně sledují všechna vylepšení operačního systému, aby je mohli zapracovat do svých aplikací. Aplikace vytvářené původně pro verzi 2.2 vypadají vedle moderních aplikací pro verze 3.0 a 4.0 trochu jako chudí příbuzní, takže málokterého uživatele upoutají natolik, aby si je z aplikačního obchodu Google Play stáhl, nebo dokonce zakoupil. Naopak aplikace využívající prvky nových verzí navržené tak, aby je bylo možné spouštět i na zařízeních se staršími verzemi Androidu, budou pro uživatele těchto zařízení velmi atraktivní.

Aby bylo možné aplikace využívající nové rysy spouštět i na starších zařízeních, musí vývojáři do svých projektů přidat knihovny Android Support Library k dosažení tzv. dopředné kompatibility.



Poznámka: Vývojář se musí rozhodnout, zda příslušnou moderní funkcionalitu implementuje do aplikace přímo nebo přes knihovnu Android Support Library.

Typickým příkladem je Action Bar. Pokud bude do projektu implementovaný bez součinnosti s v7 appcompat library, bude fungovat jen na zařízeních s verzí Android 3.0 Honeycomb a vyšší. Pokud bude Action bar implementovaný přes knihovnu pro zachování kompatibility, bude možné aplikaci spouštět na zařízeních s API 7 a vyšším, to jest od verze Android 2.1 Eclair.

Číslo v označení verze knihovny znamená API, od kterého je možné aplikace využívající příslušnou verzi knihovny spouštět. Například aplikace využívající v4 support library je možné spouštět od verze API 4, tj. Android 1.6 Donut.

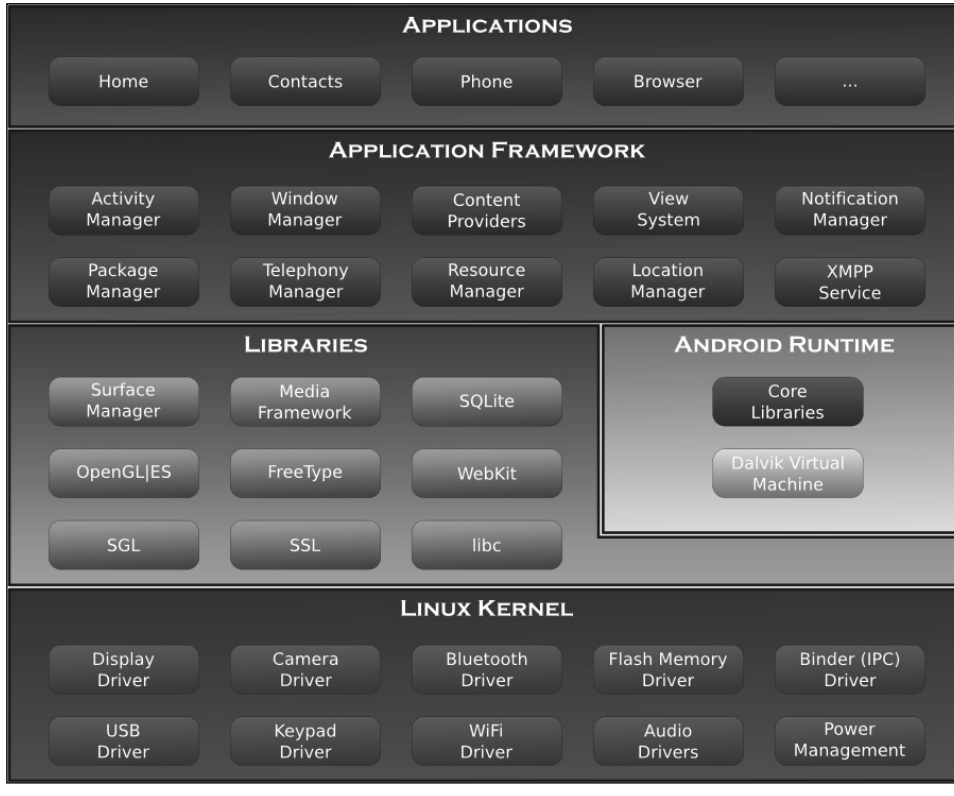
K dispozici jsou knihovny:

- **v4 support library** – umožňuje na zařízeních od verze Android 1.6 Donut spouštět aplikace využívající fragmenty, „rich" notifikaci, `LocalBroadcastManager`, `ViewPager`, `PagerTitleStrip`, `PagerTabStrip`, `DrawerLayout`, `SlidingPaneLayout`, `FileProvider` a mnohé jiné funkce, které přinesly novější verze Androidu. Knihovnu po instalaci Android Support Libraries najdete ve složce *<sdk>/extras/android/support/v4/*.
- **v7 appcompat library** – umožňuje na zařízeních od verze Android 2.1 Eclair spouštět aplikace využívající `ActionBar`, `ActionBarActivity` a `ShareActionProvider`. Knihovnu najdete ve složce *<sdk>/extras/android/support/v7/appcompat/*.
- **v7 gridlayout library** – přináší podporu kontejneru `GridLayout`. Knihovnu najdete ve složce *<sdk>/extras/android/support/v7/gridlayout/*.

- **v7 mediarouter library** – přináší podporu tříd `MediaRouter` a `MediaRouteProvider`. Najdete ji v složce *<sdk>/extras/android/support/v7/mediarouter/*.
- **v8 support library** – umožňuje na zařízeních od verze Android 2.2 Froyo spouštět aplikace využívající framework RenderScript. Knihovna je v balíčku *android.support.v8.renderscript*.
- **v13 support library** – umožňuje na zařízeních od verze Android 3.2 Honeycomb spouštět aplikace využívající Fragment user interface. Knihovna je ve složce *<sdk>/extras/android/support/v13/*.

Stručně o architektuře Androidu

K vývoji 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 2.9: 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.



Upozornění: Z důvodu variability hardwarových zařízení, na nichž může Android běžet, byla na úrovni jádra operačního systému vytvořena vrstva HAL (vrstva abstrahující hardware). HAL vytváří rozhraní pro komunikaci vyšších vrstev systému s hardwarem. Vývojáři aplikací nemusí díky HAL znát přesně hardwarové specifikace všech zařízení.

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, modul rozhlasového přijímače atd. 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 představuje *možnost přímé komunikace mezi procesy jedno z největších bezpečnostních rizik*.

Proto je na platformě Android implementován meziprocesový ovladač komunikace a volání metod zvaný Binder, které 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, pokud běží požadovaná aplikace. Na probuzení zařízení z režimu spánku v případě potřeby slouží objekty Alarm Timers.

Libraries

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 implementované 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

Android Runtime obsahuje sadu základných knihoven. Každá aplikace pro Android je samostatný proces využívající vlastní instanci virtuálního stroje DVM (Dalvik Virtual Machine). Tento zabezpečuje běh spustitelných souborů s příponou DEX. Soubory DEX vznikly kompilací z klasických souborů CLASS a JAR. Jsou kompaktnější než klasické soubory CLASS. Dalvik je optimalizovaný pro mobilní zařízení, to znamená, že bere v úvahu omezení možnosti napájení, menší paměť a podobně. Současně může běžet více instancí virtuálního stroje.



Poznámka: Dalvik Virtual Machine na mobilních zařízeních je analogií Java Virtual Machine na klasických počítačích, na rozdíl od JVM jsou aplikace spouštěny ve svých procesech.

Co se stane, když spustíte aplikaci napsanou v jazyku Java?

- Java kompilátor přeloží soubory zdrojového kódu vaší aplikace do více binárních souborů Javy.
- Nástroj s názvem DX transformuje tyto binární javové soubory do jednoho souboru ve formátu DEX. Sem se přidávají i jiné zdroje, které aplikace využívá, například obrázky.
- Virtuální stroj Dalvik potom tento soubor začne vykonávat.

Application 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 plocha 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 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 oznamovací lišta 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.

Bezpečnost na platformě Android

Android je otevřená platforma, která má robustní bezpečnost a přísnou bezpečnostní politiku. Operační systém byl navržen s vícevrstvou bezpečností, která poskytuje flexibilitu vyžadovanou pro otevřené platformy. Android převzal některé základní mechanizmy zabezpečení z Linuxu.

Důležitým bezpečnostním mechanizmem je takzvaný Sandbox pro izolovaný běh aplikací. Každá aplikace (APK) je spouštěna s vlastním přiděleným UID v separátním procesu. Aplikace dostanou přidělenou část ze souborového systému, kam mohou zapisovat soukromá data.

Od verze 3.0 podporuje Android šifrování souborového systému, takže všechna uživatelská data mohou být šifrována v jádře použitím dmcrypt implementace AES-128 s CBS a ESSIV:SHA256. Šifrovací klíč je chráněn AES-128 použitím klíče odvozeného od uživatelského hesla, což zaručuje ochranu před neautorizovaným přístupem.

Oprávnění pro aplikaci

Androidí aplikace nemohou přistupovat k některým komponentám a údajům, když nemají k takovému přístupu oprávnění. Důvody jsou pochopitelné, aplikace by bez vědomí uživate-

le neměla zhotovovat snímky, posílat SMS, vytáčet telefonní čísla a podobně. Hovory a SMS hlavně na audiotextová čísla mohou uživatele aplikace připravit o značné peníze, snímání fotografií, videa, zapnutí mikrofonu či odeslání informace o poloze zařízení je zase značný a nepříjemný zásah do soukromí.



Upozornění: Seznam oprávnění aplikace se zobrazí před její instalací a uživatel s nimi musí souhlasit, takže by se dalo konstatovat, že aplikace bude provádět činnosti, na které potřebuje oprávnění, s informovaným souhlasem uživatele.

V praxi to funguje jinak: uživatel potvrdí seznam oprávnění, aplikaci nainstaluje a... zapomene. Ruku na srdce, kdo by si pamatoval seznam oprávnění aplikace, kterou nainstaloval před několika měsíci. Konkurenční platformy iOS a Windows Phone jsou v tomto ohledu mnohem přísnější.

Oprávnění pro aplikaci na přístup k různým službám zařízení, které spadají do „Accessing Protected APIs", musí být deklarovány v manifestu. Manifest je v souboru *AndroidManifest.xml* v elementu `<uses-permission>` a je to nutná součást aplikace pro Android, kde vývojář musí definovat prostředky, které jeho aplikace bude používat. Tato práva jsou potom vynucována běhovým prostředím systému. Mezi „Accessing Protected APIs" patří například použití fotoaparátu, GPS, komunikačního rozhraní Bluetooth, funkcionality telefonování a posílání SMS a MMS zpráv, síťová konektivita a podobně.

Každá aplikace má vlastní seznam povolení, díky kterým může přistupovat k systémovým zdrojům nutným k jejímu fungování. Uživatel tato oprávnění sám povolí při instalaci. Takto mohou aplikace přistupovat ke kontaktům nebo SMS zprávám, získávat údaje o GPS poloze zařízení, přistupovat k Internetu apod.



Poznámka: Mnoho uživatelů věnuje málo pozornosti tomu, jaká oprávnění daná aplikace ke svému chodu požaduje a zda jsou opodstatněná, případně zda nejsou podezřelá. Důležité je proto porozumět, na co všechno mají konkrétní oprávnění dosah a jaké jsou možnosti jejich zneužití.

Vývojáři by měli mít přehled o nejběžnějších a také nejrizikovějších oprávněních, proto uvádíme jejich přehled:

- `CALL_PHONE` – umožňuje aplikaci volat telefonní čísla bez vědomí uživatele. Rizikem pro uživatele jsou poplatky, hlavně u audiotextových služeb. Zajímavé je, že oprávnění neumožňuje aplikaci volat na čísla tísňového volání. Mnohem transparentnější je volání s informovaným souhlasem uživatele, které funguje i bez tohoto povolení. Aplikace zobrazí obrazovku na zadávání telefonního čísla, přičemž číslo je už zadané. Uživatel jen stiskne tlačítko Volat nebo Odeslat.
- `SEND_SMS` – umožňuje aplikaci odesílat zprávy SMS. I v tomto případě jsou rizikem poplatky za zpoplatněné SMS služby.
- `WRITE_SMS` – umožňuje aplikaci přepisovat zprávy SMS uložené v zařízení nebo na kartě SIM. Škodlivé aplikace mohou vaše zprávy odstranit.

- `WRITE_EXTERNAL_STORAGE` – oprávnění umožňuje aplikaci zapisovat na kartu SD. Toto oprávnění často používají aplikace, které si zapisují vlastní údaje na kartu SD, případně pracují se soubory na kartě SD. Nejčastěji se jedná o dokumenty a multimédia.
- `READ_CONTACTS`, `WRITE_CONTACTS` – umožňuje aplikaci číst údaje o vašich kontaktech uložených v telefonu včetně informací o historii komunikace. Rizikem je získání kontaktů a jejich sdílení bez vědomí uživatele.
- `READ_CALENDAR` – umožňuje aplikaci číst události kalendáře uložené v telefonu včetně událostí přátel nebo spolupracovníků. Toto nastavení umožní aplikaci sdílet nebo ukládat vaše údaje kalendáře bez ohledu na důvěrnost nebo citlivost.
- `WRITE_CALENDAR` – umožňuje aplikaci přidat, odstranit nebo změnit události, které můžete upravit ve svém telefonu, včetně událostí přátel a spolupracovníků. Toto nastavení umožní aplikaci posílat zprávy pod jménem vlastníka kalendáře nebo upravit událost bez jeho vědomí.
- `READ_LOGS` – umožňuje aplikacím čtení z různých souborů deníku systému. Aplikace díky tomu může objevit všeobecné informace o činnostech uživatele souvisejících s přístrojem, které mohou obsahovat osobní nebo soukromé informace.
- `RECEIVE_BOOT_COMPLETED` – umožňuje aplikaci spustit se hned po startu systému. Toto nastavení může zpomalit náběh operačního systému a jeho výkon, protože aplikace bude neustále spuštěná.
- `GET_TASKS` – oprávnění umožňuje aplikaci načítat informace o aktuálních či nedávno spuštěných úlohách. Může to být bezpečnostní díra pro škodlivou aplikaci, která chce ukrást uživatelská data.
- `SYSTEM_ALERT_WINDOW` – umožňuje aplikaci vykreslování nad jinými aplikacemi nebo součástmi uživatelského rozhraní. Toto oprávnění aplikace zneužívají hlavně na zobrazování reklam.
- `KILL_BACKGROUND_PROCESSES` – umožňuje aplikaci ukončit procesy na pozadí ostatních aplikací. Může to zapříčinit zastavení ostatních aplikací. Toto povolení může zneužít škodlivá aplikace například na vypnutí procesu antivirového programu, který by detekoval její činnost.



Poznámka: Aplikace deklarují oprávnění, které samy používají, a také oprávnění vyžadované od dalších složek, které je chtějí využít.

Ilustrační příklad: Mějme aplikaci, která umožňuje uživateli vybrat kontakt z aplikace *Kontakty* a následně zobrazí polohu poštovní adresy vybraného kontaktu na mapě. K čemu vlastně v aplikaci dochází? Aplikace přistupuje k soukromým údajům v kontaktech a následně adresu přes internetové připojení odešle do služby Google Maps. Pokud by měla být aplikace striktní, musela by pokaždé vyžadovat souhlas uživatele s danou operací.

Aplikace tohoto typu využívá oprávnění `READ_CONTACTS`, `INTERNET_ACCESS`.

Androidí aplikace mohou definovat vlastní oprávnění.

Základní součásti aplikace pro Android

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

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.



Poznámka: 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 telefonu, případně je možné 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 apod.) 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 a 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 instance aktivit vytvořeny, kdy budou odsunuty na pozadí či zničeny.



Poznámka: 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 `Activity`. 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, ze 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.

Většinou aktivita 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 úloha Back Stack se na platformě Android rozumí sada souvisejících aktivit, které mohou, ale nemusí být součástí té samé aplikace.



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

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



Poznámka: Aktivita může spouštět jinou aktivitu včetně aktivit jiné aplikace.

Životní cyklus aktivity

Životní cyklus aktivity je definovaný 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 nebo 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 dia-

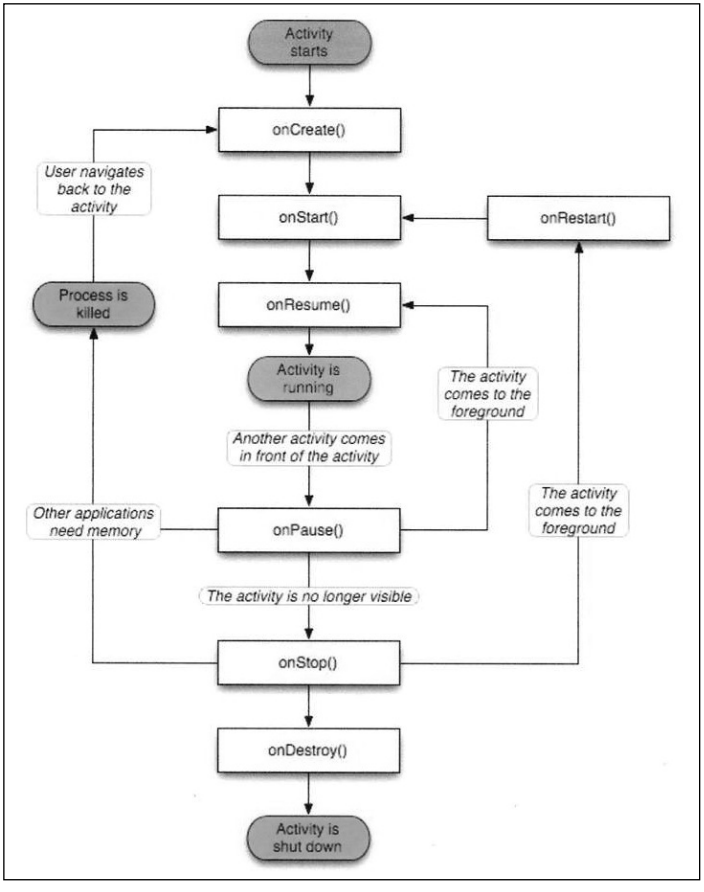
logovým oknem. V případě akutního nedostatku paměti může systém aplikaci v tomto stavu odstranit.

- **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 ve stavu pozastavené.



Poznámka: Pokud uživatel potřebuje obnovit aktivitu ve stavu pozastavená nebo zastavená a používat ji na popředí, dochází k restartu 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. Je potřeba mít 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.



Obrázek 2.10: Schéma životního cyklu aktivity

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 záměru. 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 následuje spuštění metody `onResume()`. Metoda `onResume()` se volá tehdy, když aktivita přechází z pozadí do popředí.

onPause()

V případě, že je spuštěná jiná aktivita, přechází ta, 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 **Plocha** (**Home**).



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

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



Upozornění: 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.

Intent (záměr)

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



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

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



Poznámka: `Intent` může spouštět i aktivity jiných aplikací.

Záměr 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 záměr splnit, a pokud jich najde více, nabídne uživateli možnost výběru. Můžete si změnit aplikaci, která vám zobrazuje seznam kontaktů či vykonává jinou činnost. Následně je vybraná aktivita spuštěna a příslušný záměr je jí odevzdán jako vstup.

V souvislosti se záměry se nejčastěji používají metody:

- `intent.setClass(MainActivity.this, NewActivity.class);`
  Metoda slouží k upřesnění, jaká konkrétní třída se má po odeslání záměru spustit. První parametr `Context` udává odkaz na aktuální komponentu. Druhý parametr `Class` definuje třídu, která se má po odeslání záměru spustit.

- `intent.putExtra("pocet", pocet);`
  Metoda slouží k předávání 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 bude vytvořena nová instance.
- `singleTop` – vždy bude vytvořena nová instance, pokud už není instance aktivity na vrcholu zásobníku.
- `singleTask` – bude vytvořena nová úloha, 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 úlohy je vždy jen jedna aktivita.



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

Předávání údajů a výsledků

Třída `Bundle` umožňuje přenos údajů mezi komponentami. Údaje jsou ukládány 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 `getExtras()` třídy `Intent`.

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`.

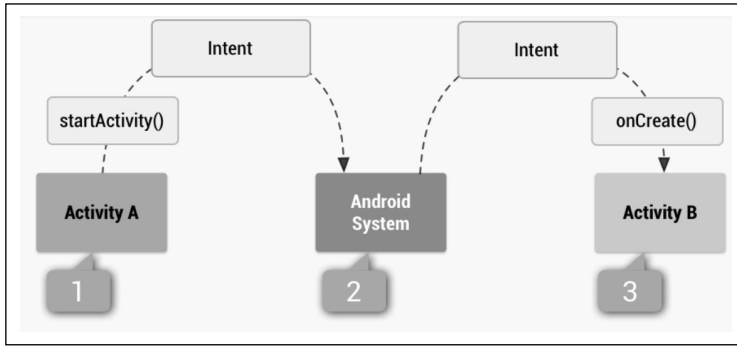
Intent Filter

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`, definující typy záměrů, které 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 2.11: 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 vykonána, 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, zobrazení dokumentu či webové stránky.

Třída `Intent` je v podstatě datová struktura, která se používá na dva účely. Umožňuje zadat operaci, kterou je potřeba provést, nebo může záměr 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`.

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.

Příklad zobrazení objektu na mapě:

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

Příklad vytočení telefonního čísla:

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

Všimněte si, že řetězce, které obsahují údaje, jsou nejprve odevzdány 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 `Intent`. 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 může aktivita vytočit telefonní číslo, potom by to měla 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 to není možné, nechá konečné rozhodnutí na uživateli aplikace.

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

Úvod do asynchronního programování

Každé aplikaci je přiděleno takzvané hlavní vlákno sloužící k interakci s uživatelem. Všechny komponenty aplikace jsou vytvořeny v tomto vlákně UI, proto každá interakce s komponentami musí probíhat z tohoto vlákna.

Všechny dlouhotrvající operace je potřeba přesunout mimo vlákno UI do samostatných vláken. Určujícím faktorem, co je dlouhotrvající operace, je předpokládaný čas trvání 50 až 100 milisekund i na nejpomalejším zařízení.

Pokud byste realizovali déletrvající operace v hlavním vlákně, zhorší se uživatelská zkušenost a uživatel si vaši aplikaci odinstaluje.

V mnoha případech je úlohou aplikační logiky v aktivitě aplikace načítat údaje z webové služby, ze souboru, z databáze. Samotný proces načítání údajů trvá nějaký čas. Někdy to při pomalém připojení nebo zaneprázdněné webové službě či databázi obsahující velké množství údajů může trvat déle, například několik sekund. Pokud byste údaje načítali v hlavním vlákně kódu aplikace, ta by po dobu čekání na údaje neodpovídala na podněty uživatele. Pokud by tento proces trval déle, operační systém Android by vaši aplikaci po pěti sekundách násilně ukončil s chybovým hlášením **Application Not Responding**. Pokud by i zmíněná operace trvala kratčeji, řekněme 3 sekundy, operační systém by aplikaci sice ještě neukončil, ale s vysokou pravděpodobností by tak učinil nervózní uživatel.

Proto operace, u kterých předpokládáme, že budou trvat déle, realizujte jako asynchronní, takže nebudou blokovat aplikaci, která je spustila, ale poběží paralelně s ní.



Poznámka: Při tvorbě aplikací, které provádějí úlohy, jež mohou trvat déle, například načítání údajů z webu, práce se soubory a podobně, je potřeba si uvědomit, že tyto dlouhotrvající operace nesmí probíhat v hlavním vlákně.

Zároveň platí jedno omezení: *prvky uživatelského rozhraní je možné modifikovat jen z hlavního vlákna aplikace*. Proto asynchronní třídy poskytují metody na jejich modifikaci v hlavním vlákně.

Podobně jako procesory v počítačích disponují i procesory zařízení s Androidem více procesorovými jádry a možností multitaskingu, takže více aplikací a vláken jejich procesů může běžet současně.



Poznámka: `Thread` (vlákno) je jedním z mnoha výpočetních procesů, které běží současně v rámci procesů operačního systému. Z hlediska implementace má každé vlákno svůj vlastní čítač instrukcí a zásobník. S ostatními vlákny sdílí paměťové oblasti a statické proměnné.

Na úrovni programovacího jazyka Java se jedná o implementaci objektu `Java.lang.Thread`. Vlákna jsou implementována jako runnable interface, které musí mít metodu `void run()`, jež nemá žádné argumenty a nevrací žádné hodnoty. Pomocí metody `start()` se vlákno spustí a použitím metody `sleep(long time)` se vlákno „uspí“ na definovanou dobu.

Pomocí metody `wait()` se vlákno uvede do stavu čekání na jiné vlákno, které například stahuje údaje z Internetu. Když toto vlákno dokončí svou činnost, pomocí metody `notify()` probudí vlákno, které na něj čekalo.

Asynchronní programování je vzhledem k typům scénářů, ve kterých se používají aplikace pro Android, například načítání údajů z webu, velmi důležité, proto mu budeme věnovat více pozornosti. Android nabízí několik možností, jak realizovat dlouhotrvající úlohy.

AsyncTask

Třída `AsyncTask` je určena pro jednodušší asynchronní úlohy. Úloha běží v samostatném vlákně. Zároveň tato třída poskytuje metody na modifikaci prvků uživatelského rozhraní v hlavním vlákně.

```
private class MojaAsynchronnaTrieda extends AsyncTask<URL, Integer, Long>
{
  protected Long doInBackground(URL... urls)
  {
    //déle, trvající operace
    ...
    // předčasné ukončení, pokud je volána metoda cancel()
    if (isCancelled()) break;

    return vysledok;
  }

  protected void onProgressUpdate(Integer... progress)
  {
    //případná indikace průběhu úlohy
    ...
  }

  protected void onPostExecute(Long result)
  {
    //činnost po ukončení úlohy
    ...

  }
}
```

Třída může obsahovat i metodu `onPreExecute()`, obsahující kód, který se má provést před spuštěním procesu na pozadí.

Při realizaci úloh v samostatných vláknech je potřeba zohlednit životní cyklus aktivity. Může se stát, že aktivita, ze které bylo asynchronní vlákno spuštěno, bude z nějakého důvodu ukončena a kód v tomto vlákně může běžet déle než aktivita. Například při změně orientace zařízení se aktivita ukončí a znovu vytvoří, ale už se neprojeví modifikace prvků uživatelského rozhraní v `onPostExecute()`. Proto se `AsyncTask` používá hlavně na krátké asynchronní operace, kterým nevadí případný restart aktivity. Všimněte si v našem fragmentu kódu, že je

důležité kontrolovat, zda úloha nebyla přerušena. Kontroluje se to v metodě `isCancelled()`. Tuto třídu použijeme k načítání JSON údajů z webové služby pro náš příklad.

AsyncTaskLoader

Jak vyplývá z názvu, tato asynchronní třída je primárně určena k načítání údajů na pozadí z různých zdrojů, přičemž načítání může trvat déle.

IntentService

Třída je koncipovaní ve smyslu filozofie fire-and-forget, tedy pošli a zapomeň.

Handler

Třída na manuální zpracování zpráv ve frontě.

Možnosti ukládání údajů

Údaje lze ukládat několik způsoby:

- **Shared Preferences** – jednoduché údaje párový klíč-hodnota. Tyto údaje se ukládají do vnitřní paměti zařízení.
- **Internal Storage** – ukládání údajů do vnitřní paměti zařízení. Vnitřní paměť u mnoha zařízení nižší cenové třídy má poměrně malou kapacitu.
- **External Storage** – ukládání údajů na paměťovou kartu. Má větší kapacitu než vnitřní paměť, je však potřeba vzít v úvahu, že zařízení paměťovou kartu mít může, ale nemusí.
- **SQLite databáze** – privátní souborová databáze, která je součástí operačního systému.
- **Síťové úložné prostory** – vlastní nebo firemní servery, služby typu Disk Google, OneDrive, Box, Dropbox…

KAPITOLA 3

Základní principy aplikace pro Android

V této kapitole:
- Příklad – vytvoření projektu
- Kontejnery na rozmístění prvků
- Příklad – definice rozložení prvků
- Příklad – spuštění jiné aktivity
- Ladění aplikace
- Příklad – Ohodnocení
- Příklad – zjištění informací o vašem zařízení

V dalších kapitolách budou mít začátečníci i migrující vývojáři dostatek zkušeností na komplexnější příklady. Avšak tyto zkušenosti je potřeba někde získat – a to je účelem této kapitoly. Na jednoduchých příkladech budeme demonstrovat základní principy vývoje aplikace pro Android a tvorby interaktivního uživatelského rozhraní. Představíme základní pilíře – „stavebnicové díly" pro komplexnější aplikace v dalších kapitolách.

Příklad – vytvoření projektu

V závěru první kapitoly byl stručně uveden postup vytvoření, přeložení a spuštění a projektu aplikace typu „Hello World". Cílem příkladu, který byl prezentován jako obrázkový návod bez hlubšího popisu, bylo ověřit správnost konfigurace vývojového prostředí, emulátoru a propojení vývojářského počítače s reálným zařízením s operačním systémem Android.

V projektu vytvořeném stejným postupem se zaměříme na jeho anatomii, to znamená, že popíšeme význam jednotlivých souborů, které projekt tvoří. Projekt ve vývojovém prostředí Eclipse s nainstalovaným a správně nakonfigurovaným Android SDK vytvoříte pomocí volby nabídky **File → New → Android Application Project**.

Vytvoření projektu

V prvním dialogovém okně na vytvoření projektu jsou tři editační pole na zadání názvu:

- **Application Name** – název zadaný do tohoto pole se bude zobrazovat při spouštění aplikace. Napište do pole vhodný název, v tomto případě „Výpočet BMI". Jelikož se jedná

o název, který uživatel uvidí, úmyslně jsme použili název s diakritikou. Diakritika je v textech uživatelského rozhraní samozřejmostí.

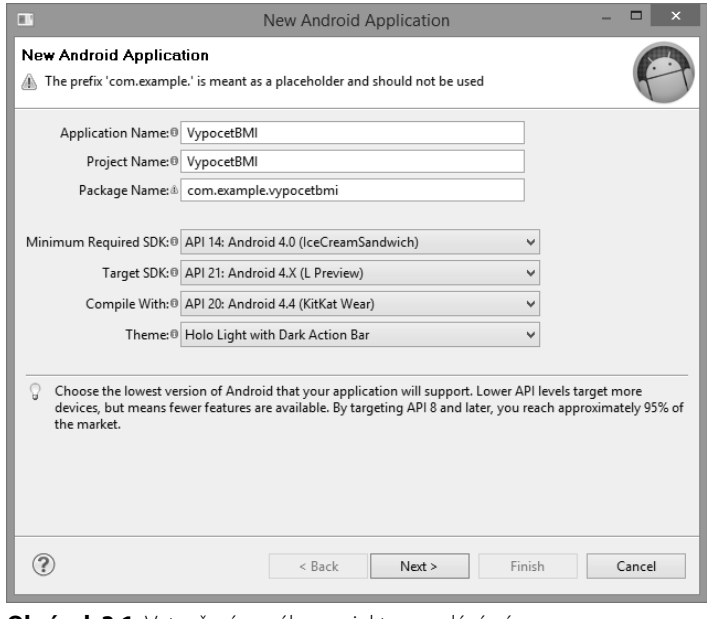


Tip: Při zadávání názvu aplikace se automaticky doplní název i do pole **Project Name** a **Package Name**.

- **Project Name** – název projektu je název adresáře projektu, který se zobrazí ve vývojovém prostředí Eclipse. Po zadání názvu aplikace do pole **Application Name** se v poli názvu projektu zobrazí stejný název s vynechanými mezerami, přičemž slovo následující za mezerou bude začínat velkým písmenem, tedy „VýpočetBMI". Zatímco v názvech viditelných v uživatelském prostředí jsme vás povzbuzovali k používání diakritiky, v názvech projektů, proměnných a podobně diakritiku používat nedoporučujeme. Dosáhnete tím určité integrity, jelikož programovací jazyk Java i XML kód pro návrh uživatelského rozhraní využívají anglické názvy klíčových slov, elementů a podobně.
- **Package Name** – do tohoto pole se zadává název javového balíčku, do kterého bude projekt aplikace přibalen. Například `com.example.prvniaplikace`. Název balíčku je zároveň jmenný prostor balíčku pro vaši aplikaci. Název balíčku musí být jedinečný v rámci všech balíčků nainstalovaných v systému Android. Možná si položíte otázku, jak byl odvozen název `com.example.vypocetbmi`? Připomíná vám něco? Ano, je to název, který začíná s opačným názvem domény, například domény vaší firmy a podobně.



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



Obrázek 3.1: Vytvoření nového projektu – zadání názvu

Specifikace verzí

Důležitý je výběr maximální a minimální verze SDK, tj. na jaké škále verzí operačního systému Android bude možné vaši aplikaci spustit. Implicitně je jako aktuální verze nastavena nejvyšší dostupná. Aktuálně nastavená verze se dá kdykoliv změnit v místní nabídce projektu – po klepnutí pravým tlačítkem myši na název projektu zvolte **Properties → Android**.



Poznámka: 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 Android manifestu (soubor *AndroidManifest.xml*) snadno změnit, pokud však 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.

V době psaní publikace byla nejvyšší dostupnou verzí Android 5.0 (Lollipop) – verze API 21 a Android 4.4 (KitKat Wear) – verze API 20. Jako nejnižší verze je v aktuální verzi vývojového prostředí Eclipse implicitně nastavena verze Android 2.2 (Froyo) – verze API 8. Je potřeba si uvědomit, že doba životnosti mobilního telefonu je dva až tři roky, takže přístrojů se staršími verzemi rapidně ubývá. V současnosti už verzi 2.2 nemá pro nové aplikace smysl podporovat.



Tip: Nové aplikace proto až na výjimky, které chcete směřovat i pro uživatele starších přístrojů, stačí psát pro verzi API 14: Android 4.0 - IceCream Sandwich a vyšší. Můžete tak bez omezení používat nové prvky uživatelského rozraní.



Poznámka: Pozorní čtenáři si všimli, že jsme překročili verzi 3.0 – Honeycomb. Ta byla určena hlavně pro první tablety. Na současných tabletech se tato verze už prakticky nevyskytuje.

Ikony pro aplikaci

V poli **Theme** se vybírá barevné schéma aplikace. K dispozici jsou barevná schémata:

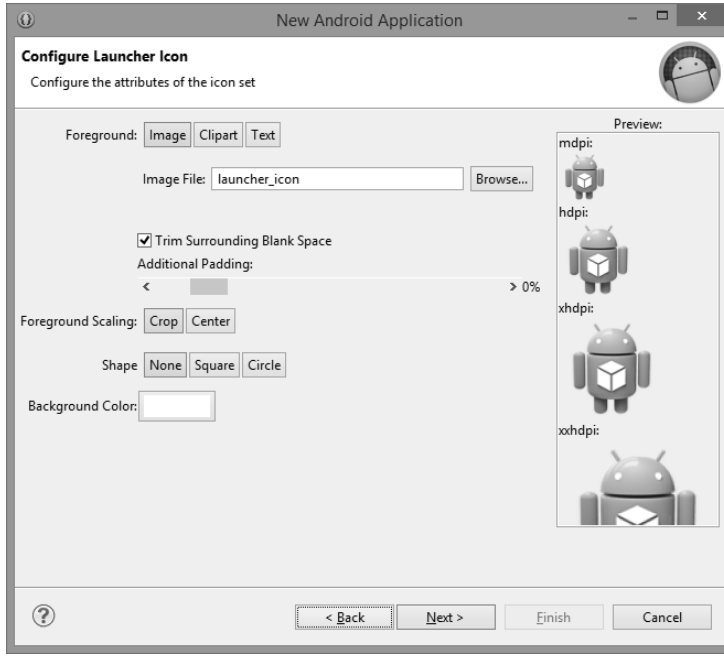
- None
- Holo Dark
- Holo Light
- Holo Light with Dark Action Bar

Implicitně je nastavena voba **Holo Light with Dark Action Bar**, tedy světlé pozadí pro aplikaci a tmavé pozadí pro aplikační lištu.

V dalším dialogovém okně ponechejte zaškrtnuté položky **Create custom launcher icon**, **Create Activity** a **Create Project in Workspace**. Význam těchto položek se postupně dozvíte.

Vytváření projektu pokračuje dialogovým oknem, ve kterém můžete změnit ikony aplikace. Pokud vytváříte cvičnou aplikaci za účelem výuky či na ověření nějakého technologického

principu, můžete ponechat implicitně nastavenou ikonu figurky Androida. Pro reálnou aplikaci samozřejmě musíte navrhnout vhodné a atraktivní ikony. Na atraktivnosti ikony totiž často závisí, zda si uživatel vaši aplikaci z Google Play stáhne, případně koupí. A po nainstalování aplikace může atraktivní ikona asociovat dobrou aplikaci a motivovat uživatele, aby ji spustil.



Obrázek 3.2: Ikony pro nově vytvořený projekt

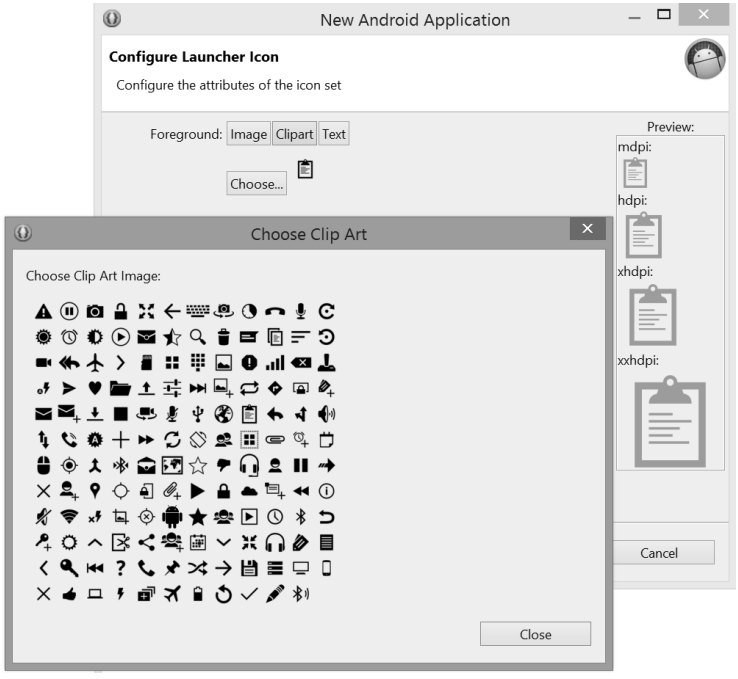
Pokud chcete pro svou aplikaci jednoduché ikony, které by odpovídaly charakteru a námětu aplikace a odlišily ji od jiných ve vašem telefonu nebo tabletu, můžete vyzkoušet ikony z galerie clipartů. Galerii zobrazíte pomocí tlačítka **Clipart**. Pokud chcete mít pro aplikaci unikátní grafický vizuál, musíte si ho navrhnout sami nebo – a to je ještě lepší – nechat návrh na zkušeném designérovi.

Výběr typu hlavní aktivity

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



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



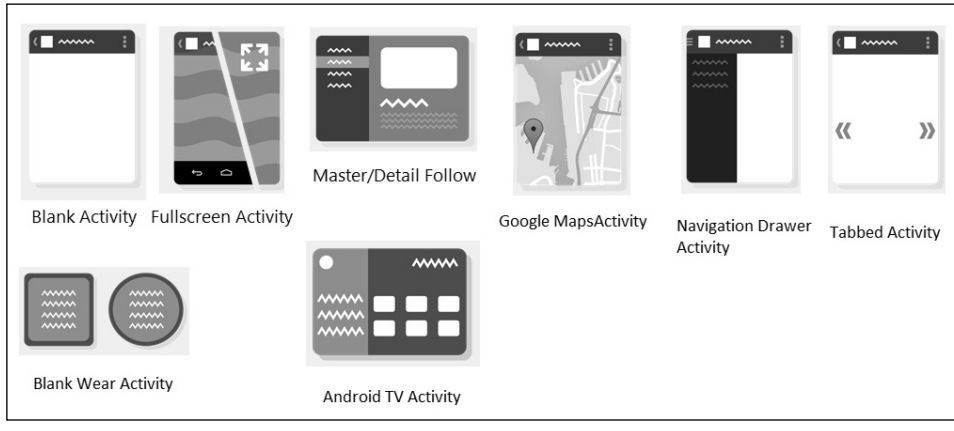
Obrázek 3.3: Nabídka ikon z galerie clipartů

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

- **Android TV Activity** – pro Android TV na bázi Leanback Support Library
- **Blank Activity** – jako volitelný prvek navigace se používá navigační lišta ActionBar s ovládacími prvky
- **Blank Activity with Fragment** – obsahuje ActionBar s ovládacími prvky a fragment
- **Blank Wear Activity** – šablona aktivity pro platformy Android Wear
- **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 Google Play Services
- **Master/Detail Follow** – 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
- **Tabbed Activity** – aktivita typu Blank Activity s možností přepínání složek gestem vodorovného posunu



Obrázek 3.4: Výběr typu aktivity



Obrázek 3.5: Typ uživatelského prostředí názorněji než název vysvětluje obrázek v dialogovém okně

Pro tuto jednoduchou aplikaci označte volbu **Empty Activity**.

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

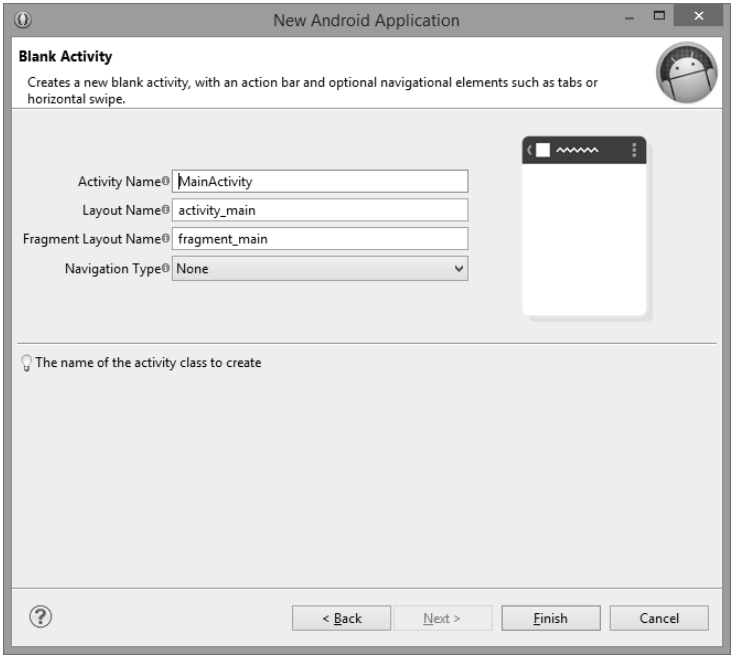
- Activity Name
- Layout Name
- Fragment Layout Name
- Navigation type

Poznámka: Pro typ aktivity **Empty Activity** se nastavují jen první dva parametry.

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 XML souboru 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



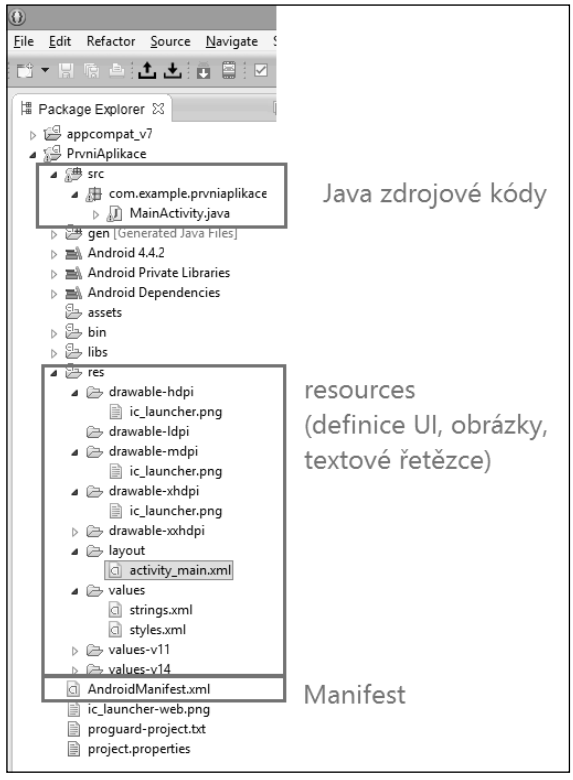
Obrázek 3.6: Konfigurace vybraného typu aktivity

Tlačítkem **Finish** dokončete vytvoření projektu.

Anatomie projektu

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

- *src* – ve složce se nacházejí soubory s kódem tříd v programovacím jazyku Java.
- *res* – ve složce se nacházejí XML soubory s definicí uživatelského rozhraní.
- *assets* – složka obsahuje soubory přibalené k aplikaci, například textové či databázové soubory.
- *android-support-v4* – složka obsahuje knihovny umožňující využívat funkcionality, které přinesly nové verze Androidu (typicky od verze Android 4) i ve starších verzích.



Obrázek 3.7: Složky s nejdůležitějšími součástmi projektu

Definice uživatelského rozhraní

Po vytvoření projektu se v hlavním okně uživatelského rozhraní vývojového prostředí Eclipse zobrazí definice návrhu uživatelského rozhraní v souboru *activity_main.xml*. Soubor je ve složce */res/layout/*.

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.

Uživatelskému rozhraní a jeho interakci s aplikační logikou se budeme věnovat celý zbytek kapitoly, takže na tomto místě budeme struční.

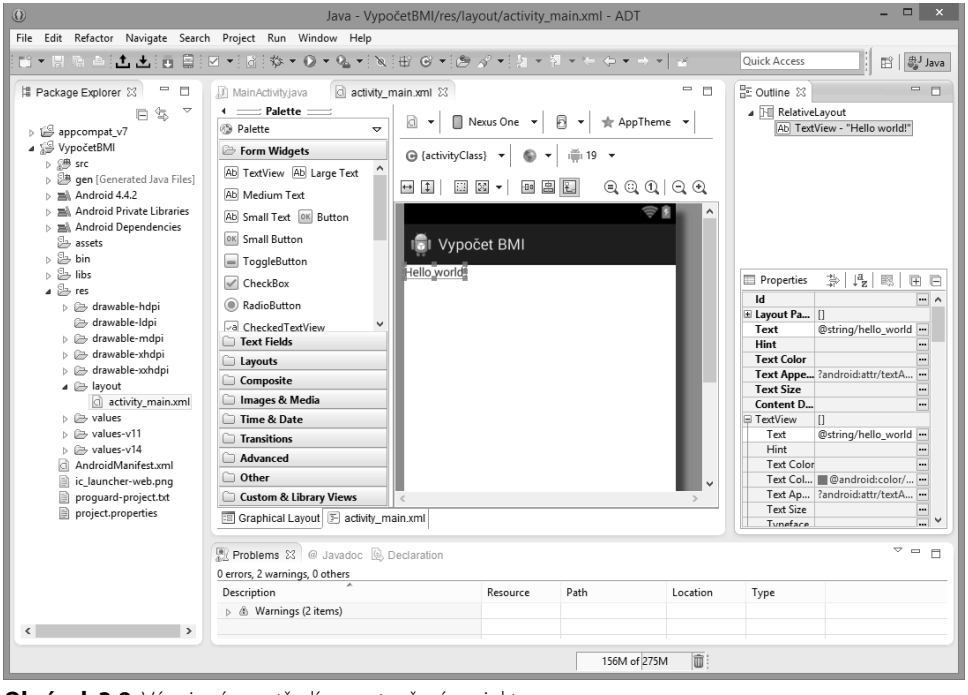
XML dokument má jeden kořenový element `RelativeLayout`, do kterého se postupně vnořují prvky (v originální terminologii `Views`), které tvoří uživatelské rozhraní aplikace. V následující části se o tomto elementu, který reprezentuje skupinu pohledů (`View Group`), dozvíte více podrobností.

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

</RelativeLayout>
```

Význam tohoto kódu nejlépe pochopíte, když přepnete vývojové prostředí do režimu zobrazení grafického návrhu. Přepněte se klepnutím na kartu **Graphical Layout** v levé dolní části hlavního okna vývojového prostředí. Zobrazí se grafická podoba uživatelského rozhraní nové aplikace. Jediným prvkem na ploše aplikace je textové oznámení.



Obrázek 3.8: Vývojové prostředí po vytvoření projektu

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.

Z dosud uvedeného vyplývá, že při návrhu uživatelského rozhraní můžete přistupovat buď deklarativně pomocí vizuálního návrhu, nebo imperativně – psaním XML kódu 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***.** Po vytvoření projektu najdete kartu tohoto souboru v horní části hlavního okna – vlevo vedle karty souboru *activity_main.xml*. Pokud byste ho zavřeli a chtěli znovu otevřít, najdete ho přes odkaz v levém okně **Package Explorer** ve složce **PrvniAplikace** → **src** → **com.example.prvniaplikace**. V souboru najdete kód hlavní aktivity.

```
package com.example.prvniaplikace;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends Activity
{

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



Upozornění: 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. Pro doplnění chybějících importů má totiž vývojové prostředí Eclipse „kouzelnou" klávesovou zkratku **Ctrl+Shift+O**, která importy automaticky doplní.

Aplikace může zapouzdřovat více samostatných aktivit, jsou však vzhledem k možnosti platformy řízeny sériově, to znamená, že uživatel s nimi pracuje postupně. Metoda `OnCreate()` je volaná operačním systémem 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`.

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*.



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 XML kódu, kódu v Javě a manifestu na tyto řetězce budete jen odkazovat.



Poznámka: V publikaci toto pravidlo občas ve cvičných aplikacích pro názornost a zkrácení výpisů porušíme.

Vraťme se k návrhu uživatelského rozhraní v souboru */res/layout/activity_main.xml*. V prvním pokusném zásahu do aplikace Hello World bude změna textu oznámení v elementu `TextView`.

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

Přesněji v definici tohoto elementu je jen odkaz na definici textového řetězce. Definice řetězců jsou uloženy v soboru *res/values/string.xml*.

```
<resources>
  <string name="app_name">MojePrvniAplikace</string>
  <string name="hello_world">Hello world!</string>
</resources>
```

Kromě textů je možné definovat i formátování jejich výpisů pomocí HTML tagů a escape sekvencí. V některých případech je užitečné definovat množná čísla podstatných jmen pro různé jazyky.



Upozornění: V reálných aplikacích vždy při definování textových řetězců definujte odkazy na soubor s řetězci v *res/strings*.

Pokud byste vkládali řetězce natvrdo do XML kódu 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).

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

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.nazev_souboru_bez_pripony`. Proto musí být názvy souborů ve zdrojích 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` atd. Můžete definovat samostatné textové řetězce, případně pole řetězců.

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

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

Například:

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

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

Pole řetězců

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

```
<resources>
  <string-array name="planety_array">
    <item>Merkur</item>
    <item>Venuše</item>
    <item>Země</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 plurálu. Například:

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

Některé plurály 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 místnosti je jedna osoba.</item>
      <item quantity="few"> V místnosti jsou %d osoby.</item>
      <item quantity="other"> V místnosti je %d osob.</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 originální terminologii qualifiers, které umožňují aplikaci přizpůsobit různému rozlišení obrazovky, lokalizovat 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 originální 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 – *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 ve zdrojích nejprve ve složce *values-en-rUS*, a pokud neuspěje, hledá o jednu hierarchickou úroveň výše ve složce *values-en*. Pokud neuspěje ani tam, hledá ve složce *values*. A 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 aplikace. Definuje jednotlivé komponenty, nastavení konfigurace a oprávnění aplikace.

Aplikační manifest najdete v souboru *AndroidManifest.xml* v hlavní složce projektu. Nejprve prozkoumáme XML soubor a potom si prohlédneme manifest v návrhovém zobrazení. V Eclipse otevřete manifest a přepněte se na **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>
```

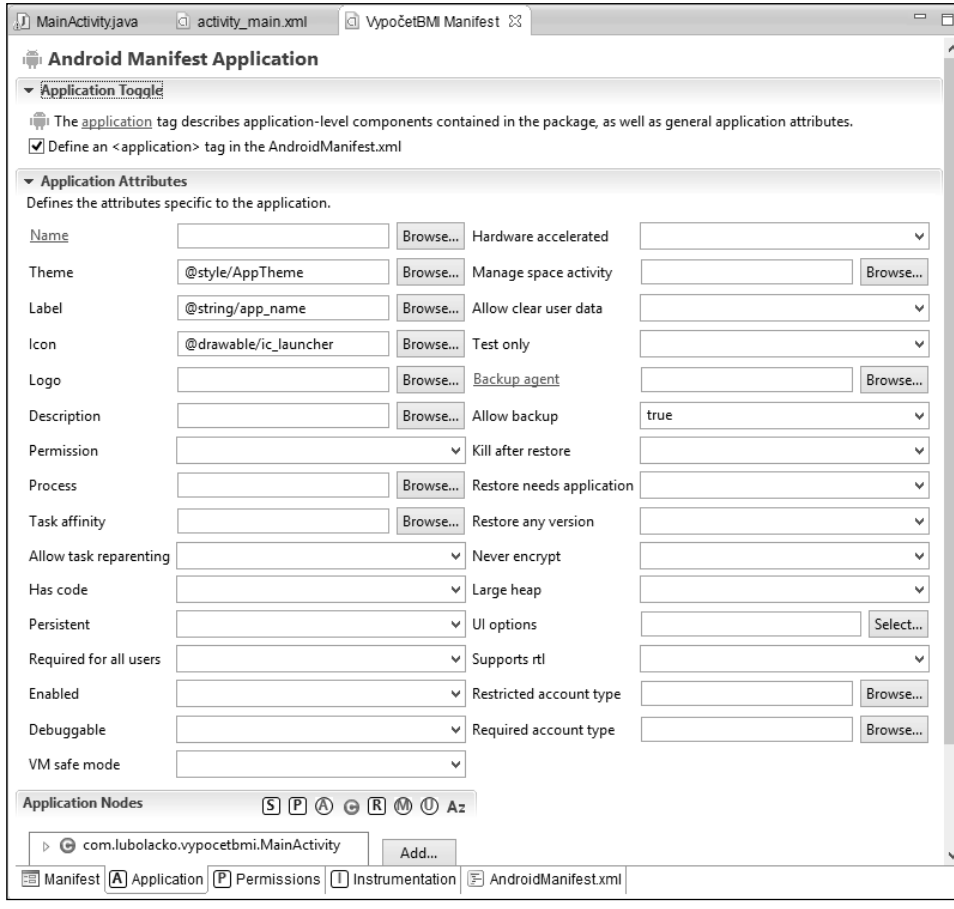
V elementu `<activity>` je seznam všech aktivit aplikace.

Pozornost věnujte i sekci `<intent-filter>`. 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 aktivitu, 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.

Obsah manifestu můžete pohodlně měnit v interaktivním návrhovém prostředí na kartách **Manifest**, **Application**, **Permissions** a **Instrumentation**. Prozkoumejte kartu **Application**, kde jsou informace týkající se aplikace.



Obrázek 3.9: Manifest v návrhovém zobrazení

Návrh uživatelského rozhraní

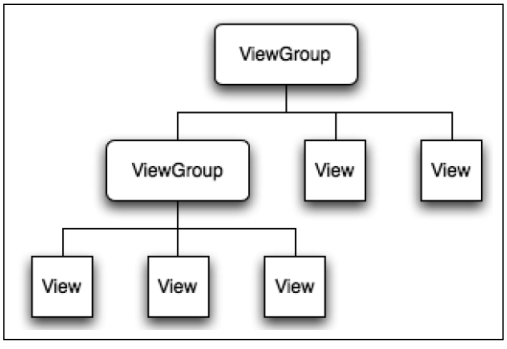
Zařízení, na kterých budou aplikace spouštěny, mají různou velikost obrazovky a různé rozlišení. Proto by programátoři migrující z klasických aplikací pro Windows měli zapomenout na praxi definování přesných souřadnic polohy a konkrétních rozměrů ovládacích prvků. Na Androidu definujete vzájemné uspořádání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

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 pohledů.

- **ViewGroups** (skupiny pohledů) jsou rozšířením pohledů pro tvorbu složených prvků, které, jak vyplývá z názvu, obsahují více pohledů.
- **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 *activity_main.xml* všimněte, že prvek `<TextView>` je umístěn uvnitř skupiny pohledů `<RelativeLayout>`.



Obrázek 3.10: Hierarchie objektů ViewGroup

Nejčastěji se k návrhu uživatelského rozhraní využívají 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 k 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 Radio Button – 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.

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 parametrům displeje. Rozlišení se uvádí v dpi (dots per inch), tj. počtu bodů na palec.

Některé parametry, například rozměry okrajů a rozestupů mezi prvky, jsou v jednotkách dp (density independent pixel) definovaných vztahem:

1dp = 160px/dpi

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

Pro rozměry ikon pro různá rozlišení platí tabulka:

mdpi	160 dpi	48 × 48 px	1x
hdpi	240 dpi	42 × 72 px	1.5x
xhdpi	320 dpi	96 × 96 px	2x
xxhdpi	480 dpi	144 × 144 px	3x
xxxhdpi	640 dpi	192 × 192 px	4x

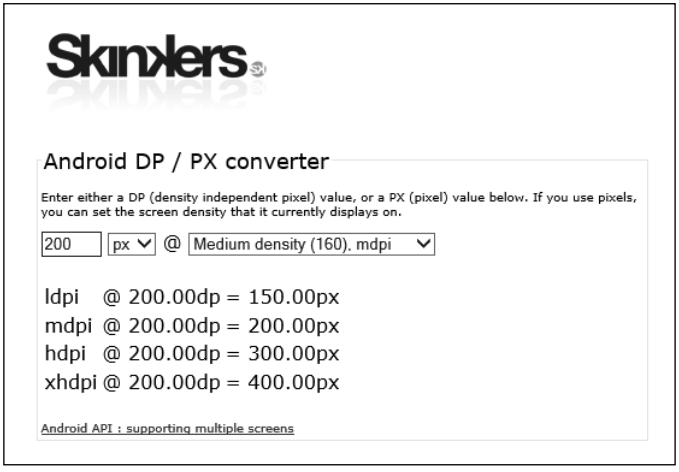
U obrázků pro každou hustotu ldpi, mdpi, hdpi a xhdpi je potřeba vytvořit obrázek v rozdílném rozlišení, a to v poměru 3:4:6:8. Například pro obrázek, který má v ldpi rozměry 180 × 180 pixelů, musíte vytvořit varianty:

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

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



Tip: Konvertor mezi dp a pixely najdete například na *http://labs.rampinteractive.co.uk/android_dp_px_calculator/*.



Obrázek 3.11: Konvertor dp na webu

Po přečtení této knihy pro vás nebude problém takovouto aplikaci napsat a publikovat na Google Play.

Obrázek 9-patch

Jedním z prostředků, jak se vyrovnat s různými rozměry a rozlišením obrazovek zařízení s Androidem, jsou takzvané 9-patch obrázky. Jedná se o speciální formát souboru PNG, díky kterému můžete v aplikaci použít například obrázek pro tlačítko s různými velikostmi a délkami stran bez deformace. V praxi to funguje tak, že v transparentním obrázku PNG nakreslíte černou čarou tloušťky jeden pixel na každé straně tlačítka a tím vyznačíte plochy, které se mají roztahovat. Zbytek obrázku zůstane netknutý. Díky tomu se nebudou deformovat rohy u zaoblených tlačítek, i když budou zvětšená, v tomto případě natažená pouze v jednom směru. K vytváření 9-patch obrázků slouží nástroj *draw9patch*, který je součástí SDK. Obrázky se ukládají s názvy podle konvence *jmeno_souboru.9.png* a Android automaticky rozpozná, že se jedná o obrázek typu 9-patch.

Kontejnery na rozmístění prvků

K definici rozmístění prvků uživatelského rozhraní slouží takzvané vizuální kontejnery, v originální terminologii layouts. Jsou odvozeny od třídy `ViewGroup`.



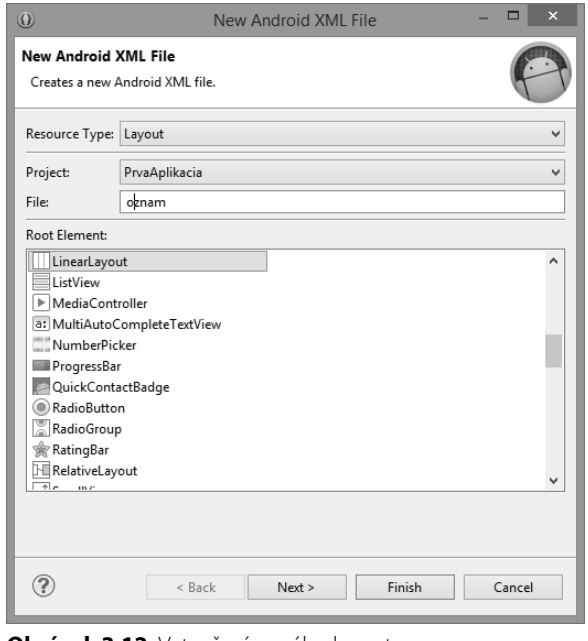
Poznámka: Rozmístění prvků musí být flexibilní; za všechny uvedeme jen dva důvody – různé rozměry obrazovek a změnu orientace zařízení.

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.

XML soubory s definicemi layoutů jsou ve vývojovém prostředí Eclipse v rámci projektu umístěny ve složce **Layouts**. Nový soubor vytvoříte pomocí místní nabídky této složky a volby **New → Android XML File**. Následně se zobrazí dialogové okno s nabídkou na vytvoření nového definičního XML souboru. Soubor může obsahovat nejen layout, ale libovolný prvek uživatelského rozhraní.



Upozornění: Jméno XML souboru s návrhem layoutu musí začínat malým písmenem.



Obrázek 3.12: Vytvoření nového layoutu

LinearLayout

Umožňuje uspořádat vnořené prvky (`view`) 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 pa-

rametr `weight` na 0 a třetí na hodnotu 1, tak 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 nemají 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`.

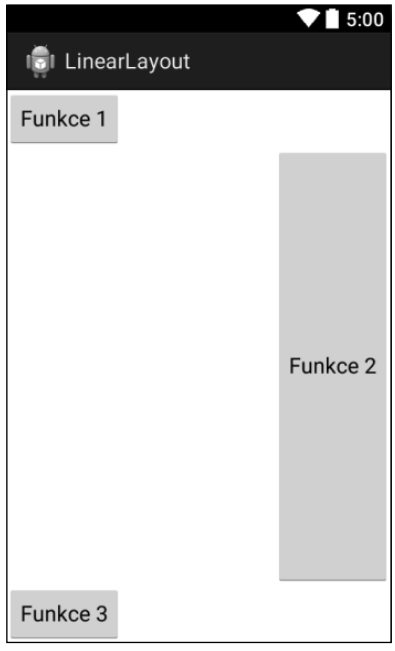
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context=
    "com.example.prvniaplikace.MainActivity$PlaceholderFragment">

  <Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="left"
    android:text="@string/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="@string/button2" />

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

</LinearLayout>
```



Obrázek 3.13: Ilustrace fungování parametrů gravity a weight

RelativeLayout

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

Příklady 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`



Poznámka: Prvky jsou implicitně umísťovány do levého horního rohu.

V ilustračním příkladu ukážeme nejprve určení polohy `view` 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"
  tools:context="${packageName}.${activityClass}" >
```

```
  <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" />

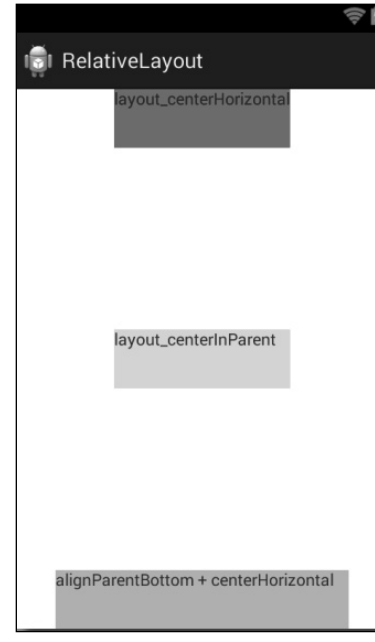
  <TextView
    android:layout_width="250dp"
    android:layout_height="50dp"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:background="@color/orange"
    android:text="alignParentBottom + centerHorizontal" />

</RelativeLayout>
```

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.



Obrázek 3.14: Relativní umisťování prvků vůči rodičovskému view a relativnímu 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"
  tools:context="${packageName}.${activityClass}" >

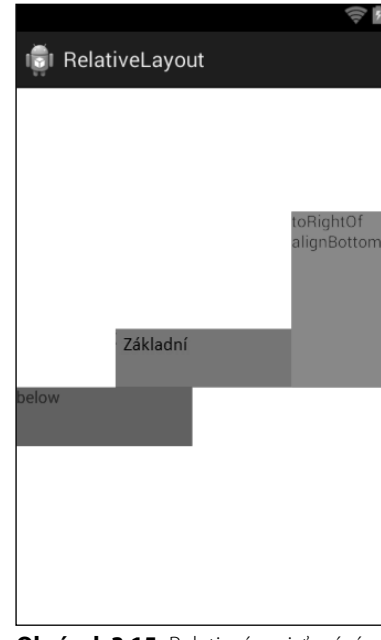
  <TextView
    android:id="@+id/zakladni"
    android:layout_width="150dp"
    android:layout_height="50dp"
    android:layout_centerInParent="true"
    android:background="@color/gray"
    android:text="Základní"
    android:textColor="@color/black" />

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

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

</RelativeLayout>
```

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.



Obrázek 3.15: Relativní umisťování prvků vůči jiným prvkům

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 layoutu `FrameLayout` 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`.

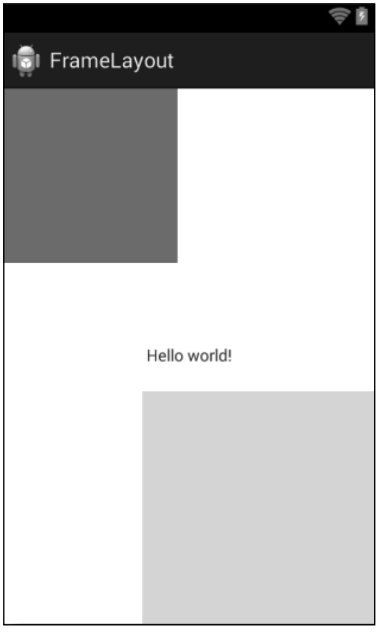
```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${packageName}.${activityClass}" >

  <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>
```



Obrázek 3.16: Příklad rozmístění prvků s využitím kontejneru 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. Z důvodu 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="${packageName}.${activityClass}" >
```

```
  <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="Denní maxima a minima teplot"
      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>

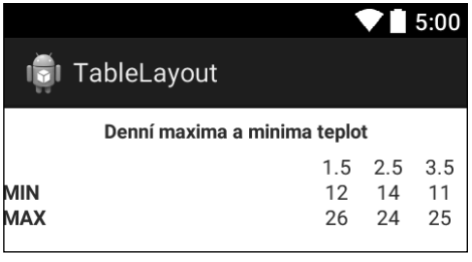
  <TableRow
    android:id="@+id/tableRowHttps"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >

    <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 3.17: Příklad rozmístění prvků s využitím kontejneru TableLayout

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:useDefaultMargins="true"
  android:alignmentMode="alignBounds"
  android:columnOrderPreserved="false"
  android:columnCount="4"
  tools:context="${packageName}.${activityClass}" >

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

  <TextView
    android:text="Přihlaste se k odběru našich novinek:"
    android:textSize="16dip"
    android:layout_columnSpan="4"
    android:layout_gravity="left"/>

  <TextView
    android:text="E-mail:"
    android:layout_gravity="right"/>

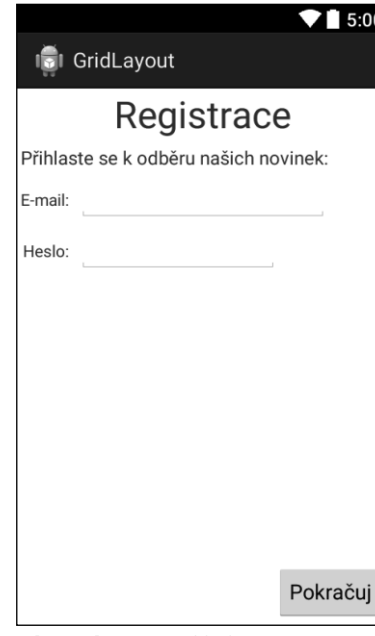
  <EditText
    android:ems="10"/>

  <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 3.18: Příklad rozmístění prvků s využitím kontejneru GridLayout

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.

```
<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="${packageName}.${activityClass}" >

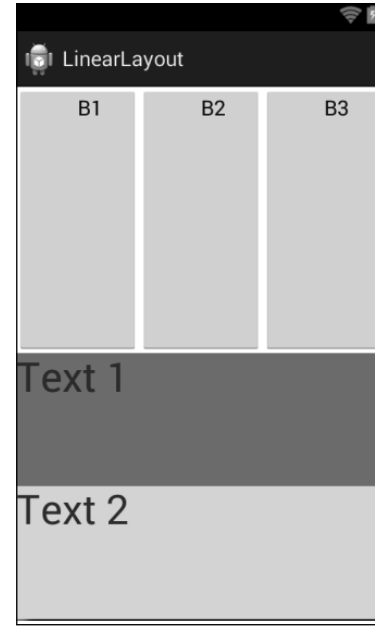
  <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">

    <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>
```



Obrázek 3.19: Kombinace vnořených kontejnerů

ScrolView

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 `ScrolView` v konkrétní poloze, nastavíte ji pomocí parametrů `android:scrollY`, respektive `android:scrollX`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <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="@drawable/ic_launcher" />

  <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 3.20: Příklad použití dvou prvků typu ScrollView

Příklad – definice rozložení prvků

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 v vizuálním návrhovém prostředí Eclipse pomocí pár klepnutí přidat na plochu aplikace prvky `EditText` a `Button`. Toto je správný a nejrychlejší postup, ale z hlediska metodiky ho doporučujeme až v dalších příkladech. V prvním příkladu je mnohem lepší návrh editováním XML kódu.

V projektu vytvořeném s využitím šablony **Blank 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 příklad 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.

Vizuální návrh uživatelského rozhraní se realizuje nad souborem */res/layout/main.xml*.

V návrhovém kódu vymažte prvek `<TextView>` a skupinu pohledů `<RelativeLayout>` nahraďte skupinou pohledů `<LinearLayout>`. Atribut `android:orientation` nastavte na hodnotu `"horizontal"`. Je potřeba nastavit i parametry `android: layout_width` a `android: layout_height`, aby bylo možné určit velikost zobrazení. V tomto případě se nastaví oba parametry na hodnotu `"match_parent"`, to znamená, že dceřiný prvek přebere rozměry rodičovského prvku.

```
<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}" >
  ...
</LinearLayout >
```



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.

```
<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:layout_marginTop="35dp"
  android:onClick="onClick"
  android:text="@string/text_TlacidlaOK" />
```

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` už znaménko plus nepotřebují.

Pro zajímavost, seznam přidaných identifikátorů a jejich číselné hodnoty najdete v souboru *R.java*. Soubor je ve složce *gen*.

```
public final class R {
  public static final class attr {
  }
  public static final class drawable {
    public static final int ic_launcher=0x7f020000;
  }
  public static final class id {
    public static final int etVaha=0x7f060001;
    public static final int etVyska=0x7f060000;
  }
  public static final class layout {
    public static final int activity_main=0x7f030000;
    public static final int activity_vypis=0x7f030001;
  }
  public static final class string {
    public static final int app_name=0x7f040000;
    public static final int button_title=0x7f040001;
    public static final int et_labelVaha=0x7f040003;
    public static final int et_labelVyska=0x7f040002;
    public static final int hello_world=0x7f040007;
    public static final int nespravne_cislo=0x7f040004;
    public static final int title_activity_vypis=0x7f040006;
    public static final int zobraz_bmi=0x7f040005;
  }
  public static final class style {

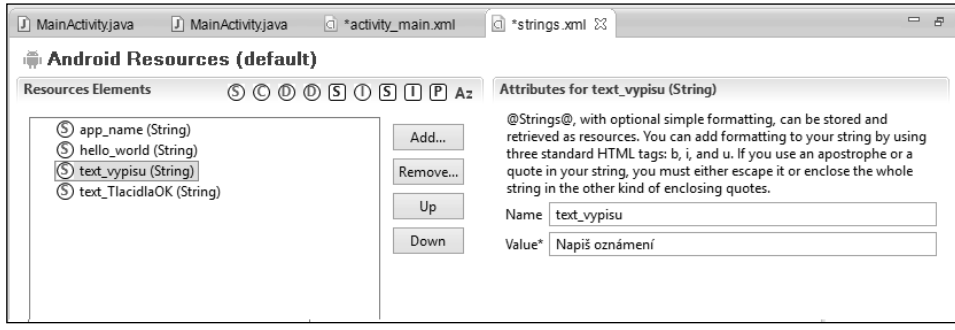
  }
}
```

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.

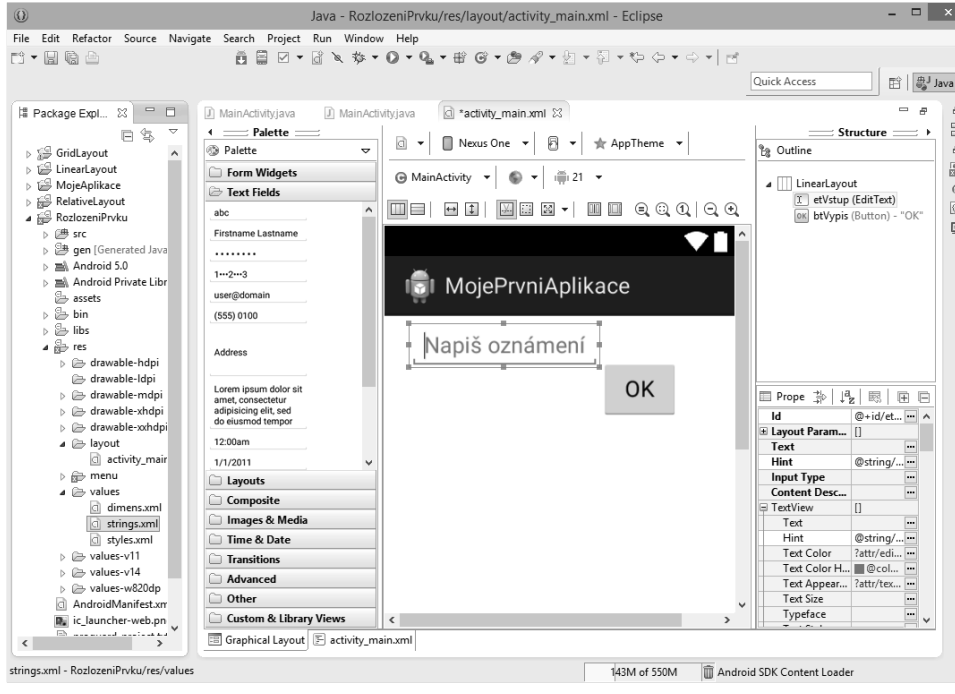
Parametr `android:hint` obsahuje textový řetězec, který se zobrazí, když je textové pole prázdné. Můžete sice použít přímo textový řetězec, takovéto zjednodušení by vám ale později pravděpodobně přineslo více problémů než užitku. Mnohem systematičtější je použití reference na hodnotu řetězce ve zdrojích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

```
<resources>
  <string name="app_name">MojePrvniAplikace</string>
  <string name="hello_world">Hello world!</string>
  <string name="text_vypisu">Napiš oznámení</string>
  <string name="text_TlacidlaOK">OK</string>
</resources>
```



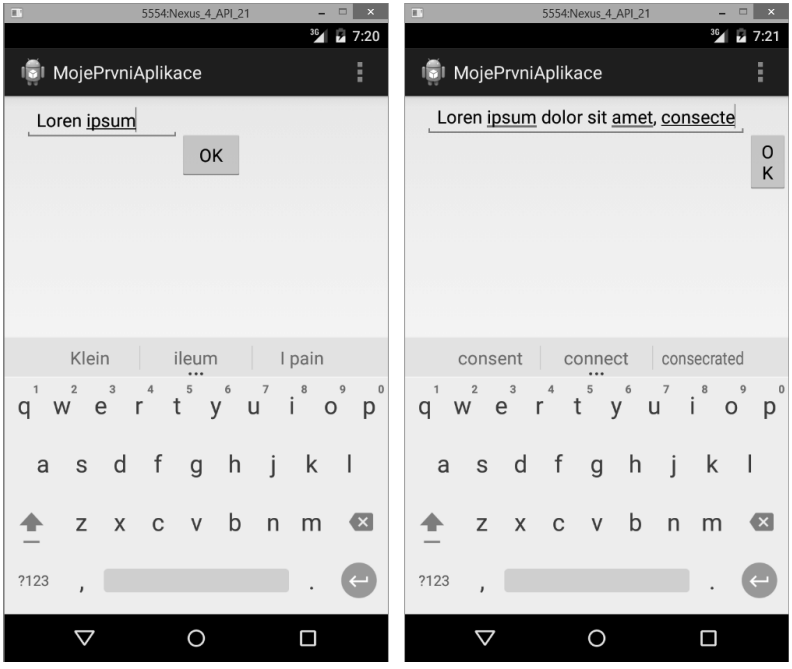
Obrázek 3.21: Soubor res/values/strings.xml v zobrazení vizuálního editoru prostředků

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



Obrázek 3.22: Vizuální reprezentace prvků uživatelského rozhraní

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 3.23: 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, tj. 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 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 (0 dp). Systém potom vypočítá relevantní šířku, aby prvek definovaným způsobem vyplnil zbývající prostor.

Upravte definici prvku `EditText`.

```
<EditText android:id="@+id/etVstup "
  android:layout_weight="1"
  android:layout_width="0dp"
  android:layout_height="wrap_content" />
```



Obrázek 3.24: Test uživatelského rozhraní, psaní delší textové zprávy po úpravě

Kompletní XML kód 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 XML kódu definice tlačítka jsme přidali řádek

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

definující, co se stane po klepnutí na tlačítko. Jelikož je Android neodmyslitelně spojený se zařízeními s dotykovým displejem, aktivace se fyzicky provede klepnutím na tlačítko.

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

Tělo metody bude v souboru *MainActivity.java*. Soubor najdete ve složce *src/adresar_aplikace*. Budeme postupovat metodicky. Nejprve v obsluze události vypíšeme jednoduché oznámení zadané přímo v kódu.

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`.



Upozornění: Nezapomeňte pomocí klávesové zkratky **Ctrl+Shift+O** doplnit chybějící údaje jmenných prostorů tříd.

```
public void sendMessage(View view)
{
  String sTextOznamu = "Tlačítko stisknuto";
  Toast.makeText(getApplicationContext(), sTextOznamu,
    Toast.LENGTH_LONG).show();
}
```

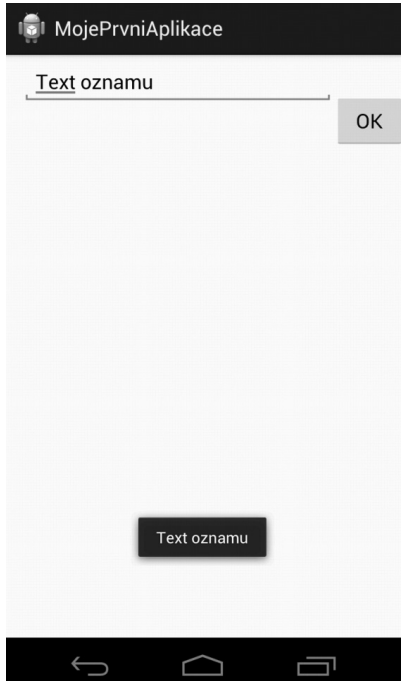


Poznámka: Metody na obsluhu událostí musí být typu `public void`.

Do splnění zadání příkladu chybí už jen malý krůček: 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 XML soubory. Například `R.layout.vystup_dialog` odkazuje na soubor *vystup_dialog.xml* ve složce */res/layout*.

```
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();
}
```



Obrázek 3.25: Test metody na obsluhu události

Příklad – spuštění jiné aktivity

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

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

V hlavní aktivitě bude několik prvků `view`, 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` s vertikálním rozmístěním prvků. Jelikož uživatel bude zadávat jen čísla, nakonfigurujeme prvek `EditText` tak, aby bylo prostřednictvím něj možné zadat jen textový řetězec sestávající z číslic.

```
<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="${packageName}.${activityClass}" >

  <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" />
```



Obrázek 3.26: Uživatelské rozhraní aktivity na zadání vstupních parametrů

```
</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">Musíte zadat číselnou hodnotu</string>
  <string name="zobraz_bmi">Vaše BMI je %1$.2f.</string>
</resources>
```

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; to znamená, že 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` je použita metoda `findViewById(int id)`. Parametr je identifikátor požadovaného prvku. Text následně konvertujeme na celé číslo.

```
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 runtime vazby mezi jednotlivými složkami, například aktivitami a argumenty na odevzdání parametrů. Záměr se odevzdá operačnímu systému a ten se snaží zařídit, aby byl 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:

```
public class MainActivity extends Activity
{

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

  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);
  }

  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 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 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);
  }
}
```

Vytvoření nové aktivity

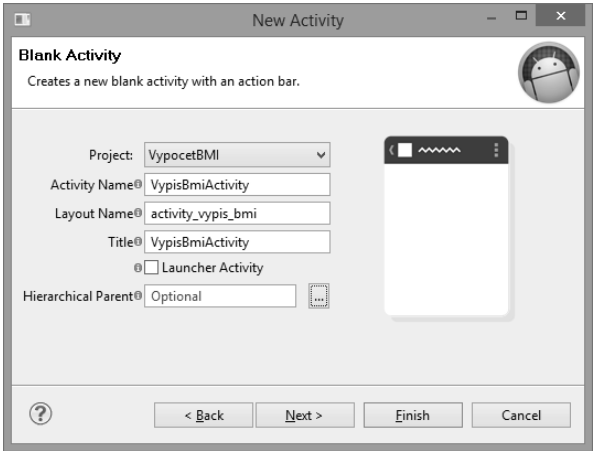
Následně je potřeba vytvořit novou aktivitu na výpis vypočítaného BMI. K vytvoření aktivity, která obsahuje i XML definiční soubor uživatelského rozhraní, můžete s výhodou použít průvodce vytvořením aktivity. Spustíte ho přes tlačítko **New**, které najdete na panelu nástrojů vývojového prostředí Eclipse úplně vlevo. Následně v nabídce **Android** vyberte položku **Android Activity**. K vytvoření nové aktivity je k dispozici široká nabídka šablon.



Obrázek 3.27: Nabídka šablon na vytvoření nové aktivity

Zdrojový kód třídy nové aktivity *VypocetBmiActivity.java* najdete ve složce *src* ve vnořené složce *com.example.nazevprojektu*. Soubor na návrh uživatelského rozhraní *activity_vypis_bmi.xml* vytvořený průvodcem v této aplikaci nevyužijete.

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



Obrázek 3.28: Průvodce vytvořením nové aktivity

Začněte tvořit kód třídy nové aktivity. V hlavní aktivitě jsme odkazovali na textový řetězec s unikátním identifikátorem objektu na odevzdání parametru BMI. Proto musíme 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>
```

Kompletní zdrojový kód třídy nové aktivity je:

```
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);
  }
}
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.bmivypocet"
  android:versionCode="1"
  android:versionName="1.0" >

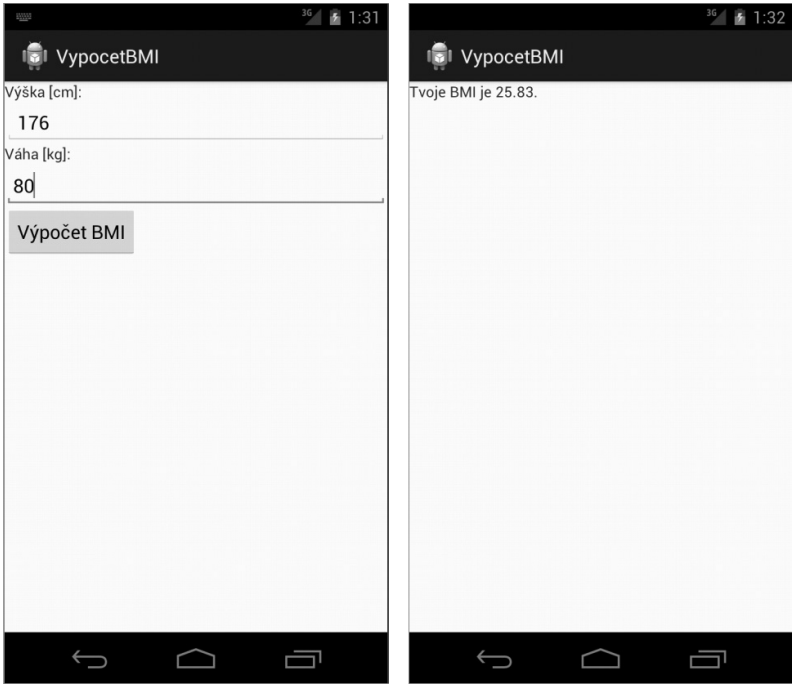
  <uses-sdk
    android:minSdkVersion="14"
    android:targetSdkVersion="19" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.bmivypocet.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>
    <activity
      android:name=".VypisBmiActivity"
      android:label="@string/title_activity_vypis" >
    </activity>
  </application>

</manifest>
```

Nyní můžete aplikaci vyzkoušet.



Obrázek 3.29: Test aplikace – vlevo je hlavní aktivita zobrazená po spuštění aplikace určená k zadání parametrů na výpočet BMI a vpravo je aktivita, která vypíše vypočítané BMI

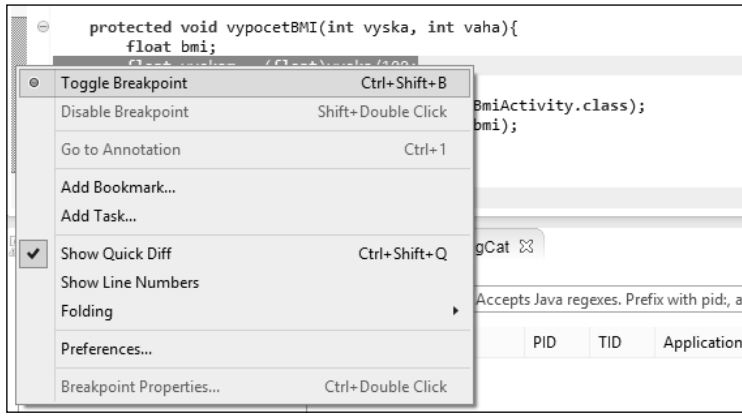
Ladění aplikace

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, matematické operace v pohyblivé řádové čárce však 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, případně odstranění předtím umístěné zarážky, se používá položka **Toggle Breakpoint** z místní nabídky editoru kódu nebo klávesová zkratka **Ctrl+Shift+B**.



Obrázek 3.30: Místní nabídka při umístění zarážky

Následně tlačítkem **Debug** (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 na ovládání krokování. Při ladění nejčastěji využijete:

- Step Into (F5)
- Step Over (F6)
- Step Return (F7)
- Skip All Breakpoints

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 je

```
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);
}
```

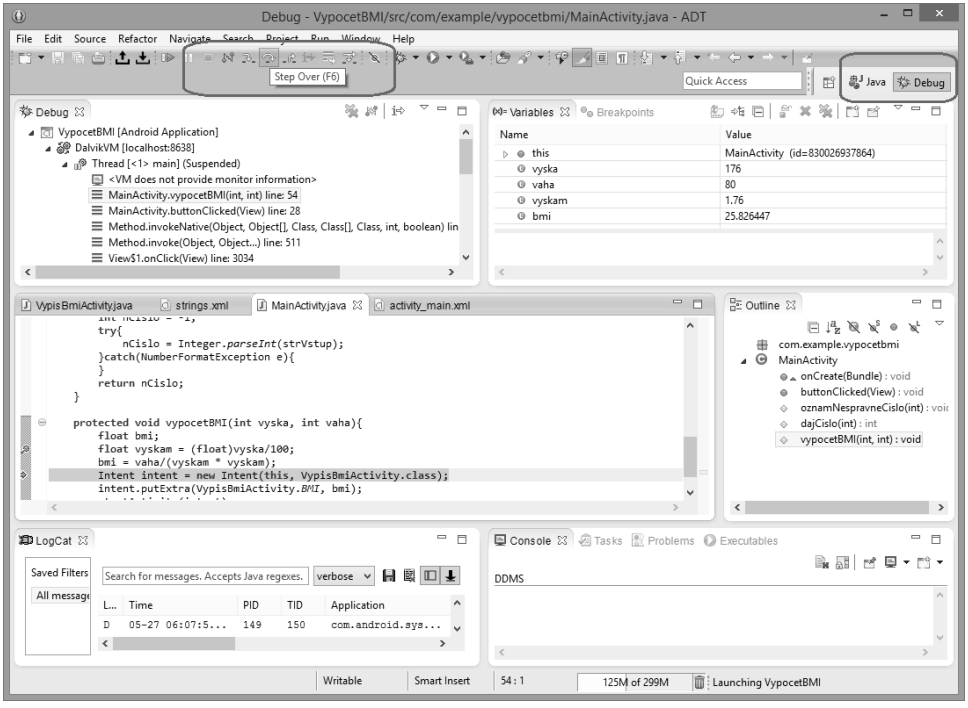
Změňte řádek s konverzí centimetrů na metry

```
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 `vyska` je typu `integer`.



Obrázek 3.31: Vývojové prostředí v režimu ladění

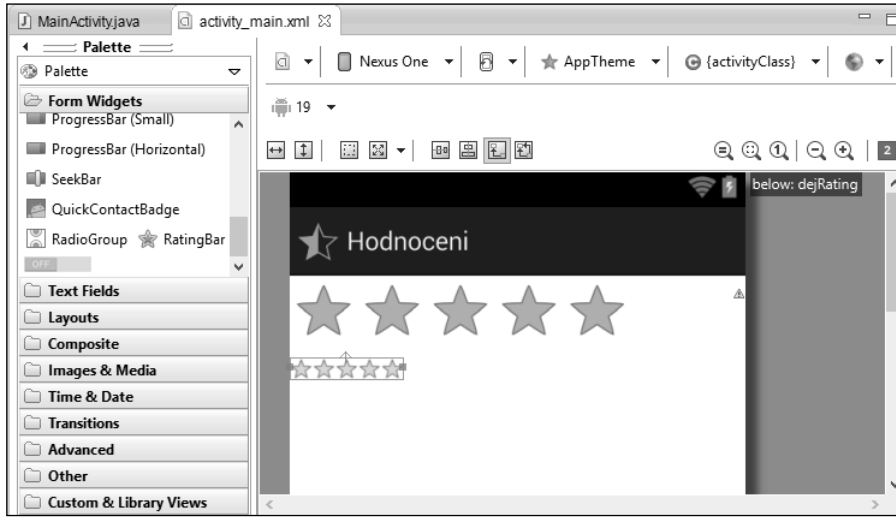
Příklad – Ohodnocení

Z mobilních, ale i webových aplikací je notoricky známá možnost hodnocení, ať už je to hudební skladba, příspěvek na blogu či sociální síti a podobně. K tomuto účelu se nejvíce využívá prvek `RatingBar`, který se vizuálně jeví jako seskupení pěti hvězdiček, přičemž uživatel hodnotí příslušný objekt klepnutím na hvězdičku.

Vytvořte aplikaci umožňující hodnocení položky. Uživateli se po spuštění aplikace zobrazí dosavadní hodnocení, tj. průměr počtu hvězdiček, které mu přiřadili lidé, kteří položku hodnotili před vámi. Zároveň se zobrazí i jejich počet.

Návrh uživatelského rozhraní

Abychom dosáhli nejčastěji používaného designu uživatelského rozhraní pro hodnocení položky, využijeme dva prvky typu `RatingBar` různé velikosti. Pomocí většího z nich uživatel hodnotí a menší prvek slouží k zobrazení průměrného hodnocení. Náš záměr při návrhu uživatelského rozhraní nejlépe objasní obrázek.



Obrázek 3.32: Návrh uživatelského rozhraní hlavní aktivity

Vedle menšího prvku `RatingBar` bude prvek `TextView` na zobrazení počtu lidí, kteří položku dosud hodnotili. Pro větší prvek nastavíme parametr `stepSize` na hodnotu 1, takže bude zobrazovat hodnotu jen na „celé“ hvězdičky. Pro menší prvek nastavíme parametr `stepSize` na hodnotu 0,1, takže plocha hvězdičky dokáže zobrazit 10 stavů od prázdné až po plnou.

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

  <RatingBar
    android:id="@+id/dejRating"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:numStars="5"
    android:stepSize="1.0" />

  <RatingBar
    android:id="@+id/nastavRating"
    style="?android:attr/ratingBarStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/dejRating"
    android:isIndicator="true"
    android:numStars="5"
    android:stepSize="0.1" />

  <TextView
```

```
    android:id="@+id/pocetText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/dejRating"
    android:layout_toRightOf="@+id/nastavRating"
    android:paddingLeft="10dp" />
</RelativeLayout>
```

Aplikační kód

V kódu hlavní aktivity je implementovaný listener `OnRatingBarChangeListener` reagující na změnu hodnoty zadané uživatelem přes `RatingBar`. V metodě `onRatingChanged` se vypočítá nové průměrné hodnocení a zobrazí se na menším prvku `RatingBar`.

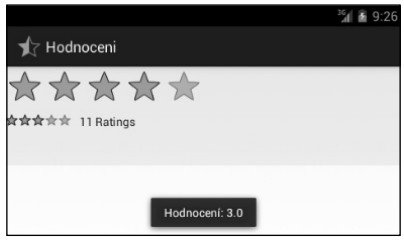
```
public class MainActivity extends Activity implements
  OnRatingBarChangeListener
{
  RatingBar dejRatingBar, nastavRatingBar;
  TextView pocetText;
  int pocet;
  float prumHodnoceni;

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

      dejRatingBar = (RatingBar) findViewById(R.id.dejRating);
      nastavRatingBar = (RatingBar) findViewById(R.id.nastavRating);
      pocetText = (TextView) findViewById(R.id.pocetText);

      nastavRatingBar.setRating(prumHodnoceni);
      dejRatingBar.setOnRatingBarChangeListener(this);
    }

    public void onRatingChanged(RatingBar rateBar, float rating,
      boolean fromUser)
    {
      DecimalFormat decimalFormat = new DecimalFormat("#.#");
      prumHodnoceni = (prumHodnoceni * pocet + rating) / ++pocet;
      prumHodnoceni = Float.valueOf(decimalFormat.format(prumHodnoceni));
      Toast.makeText(MainActivity.this,
        "Hodnocení: " + prumHodnoceni, Toast.LENGTH_SHORT).show();
      nastavRatingBar.setRating(prumHodnoceni);
      pocetText.setText(pocet + " Ratings");
    }
}
```



Obrázek 3.33: Test aplikace (redukovaný obrázek)

Příklad – zjištění informací o vašem zařízení

Spustili jste si už některou z aplikací na reálném zařízení? Vlastně ani nebylo potřeba, emulátory telefonů a tabletů jsou sice pomalé, ale natolik realistické, že na vyzkoušení jednoduchých příkladů postačí. Abychom vás motivovali připojit reálné zařízení, ukážeme příklad, pomocí kterého se o svém zařízení dozvíte některé zajímavé informace.

Vytvořte projekt aplikace pro Android s názvem například `MojeZarizeni`. V aplikaci budou použity tyto textové řetězce:

```
<resources>
  <string name="app_name">MojeZařízení</string>
  <string name="bat_kapacita">Baterie:</string>
  <string name="disp_horizontal">Rozlišení vodorovně:</string>
  <string name="disp_vertikal">Rozlišení svisle:</string>
  <string name="cpu_frekvence">Frekvence CPU:</string>
  <string name="cpu_jadra">Počet jader CPU:</string>
  <string name="ram_celkem">Kapacita RAM:</string>
  <string name="ram_volna">Volná RAM:</string>
</resources>
```

Abychom návrh co nejvíc zjednodušili, budeme zobrazovat jen dvousloupcovou tabulku popis-hodnota. Na hlavní aktivitu můžeme využít vizuální kontejner `GridLayout`. Pro jednoduchost zobrazíme jen tři údaje: změnu hodnoty akcí 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="${packageName}.${activityClass}" >

  <TextView
    android:layout_column="0"
    android:layout_gravity="right"
```

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

  <TextView
    android:layout_column="1"
    android:id="@+id/dispHorizontal"/>

  <TextView
    android:layout_column="0"
    android:layout_gravity="right"
    android:text="@string/disp_vertikal" />

  <TextView
    android:layout_column="1"
    android:id="@+id/dispVertikal"/>

  <TextView
    android:layout_column="0"
    android:layout_gravity="right"
    android:text="@string/bat_kapacita" />

  <TextView
    android:layout_column="1"
    android:id="@+id/batKapacita"/>

  <TextView
    android:layout_column="0"
    android:layout_gravity="right"
    android:text="@string/cpu_jadra" />

  <TextView
    android:layout_column="1"
    android:id="@+id/cpuJadra"/>

  <TextView
    android:layout_column="0"
    android:layout_gravity="right"
    android:text="@string/ram_celkem" />

  <TextView
    android:layout_column="1"
    android:id="@+id/ramCelkem"/>

  <TextView
    android:layout_column="0"
    android:layout_gravity="right"
    android:text="@string/ram_volna" />

  <TextView
    android:layout_column="1"
    android:id="@+id/ramVolna"/>

</GridLayout>
```

V kódu hlavní aktivity postupně zjistíme všechny parametry, které nás zajímají. Rozlišení obrazovky v pixelech zjistíme prostřednictvím objektu `DisplayMetrics`. Počet jader získáte z objektu `Runtime` pomocí funkce

```
int jadra = Runtime.getRuntime().availableProcessors();
```

Složitější je získání údajů o frekvenci procesoru, které nejsou dostupné přes žádnou knihovnu. Pokud byste chtěli v aplikaci zobrazit tento údaj, mohli byste ho pomocí metody `RandomAccessFile()` načítat například ze souboru */sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state*.

V tomto souboru je více hodnot, jelikož aktuální frekvence procesorů ARM kolísá – je řízena podle aktuální potřeby. Proto je z tohoto souboru nutné načíst nejvyšší hodnotu. Z uvedeného vyplývá, že soubor je vhodnější na načítání hodnot pro aplikaci, která by chtěla monitorovat průběžný výkon procesoru. Pro naši aplikaci je snazší načítat hodnotu maximální frekvence procesoru ze souboru */sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq*.

```
String cpuFQ = "--";
RandomAccessFile reader;
try
{
  reader = new RandomAccessFile(
    "/sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_max_freq", "r");
  cpuFQ = reader.readLine();
  reader.close();
}
catch (FileNotFoundException e)
{
  e.printStackTrace();
}
catch (IOException e)
{
  e.printStackTrace();
}
```

Údaje o aktuální frekvenci procesoru jsou v souboru */sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq*.

Z objektu `Runtime` získáte nejen údaje o počtu jader procesoru, ale i údaje o celkové kapacitě paměti a její momentálně volné kapacitě.

Údaje o momentální kapacitě baterie získáte podobně jako frekvenci procesoru ze souboru ve složce *sys*, konkrétně */sys/class/power_supply/battery/capacity*.

Kompletní kód hlavní aktivity je:

```
public class MainActivity extends Activity
{

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
```

```
    setContentView(R.layout.activity_main);

    TextView dispHorizontal = (TextView)findViewById(R.id.dispHorizontal);
    TextView dispVertikal = (TextView)findViewById(R.id.dispVertikal);
    TextView cpuJadra = (TextView)findViewById(R.id.cpuJadra);
    TextView ramCelkem = (TextView)findViewById(R.id.ramCelkem);
    TextView ramVolna = (TextView)findViewById(R.id.ramVolna);
    TextView batKapacita = (TextView)findViewById(R.id.batKapacita);

    //displej
    WindowManager wm = (WindowManager)
      getSystemService(Context.WINDOW_SERVICE);
    Display disp = wm.getDefaultDisplay();
    DisplayMetrics metrics = new DisplayMetrics();
    disp.getMetrics(metrics);
    int sirka = metrics.widthPixels;
    int vyska = metrics.heightPixels;

    //cpu
    int jadra = Runtime.getRuntime().availableProcessors();

    //RAM
    long total = Runtime.getRuntime().totalMemory();
    long free = Runtime.getRuntime().freeMemory();

    //baterie
    String bat = null;
    try
    {
      RandomAccessFile rf = new RandomAccessFile(
        "/sys/class/power_supply/battery/capacity", "r" );
      bat = rf.readLine();
      rf.close();
    }
    catch ( IOException ex )
    { ex.printStackTrace(); }

    dispHorizontal.setText(Integer.toString(sirka));
    dispVertikal.setText(Integer.toString(vyska));
    cpuJadra.setText(Integer.toString(jadra));
    ramCelkem.setText(Long.toString(total));
    ramVolna.setText(Long.toString(free));
    batKapacita.setText(bat);
  }
}
```



Obrázek 3.34: Test aplikace zobrazující informace o zařízení

KAPITOLA 4

Vizuální návrh uživatelského rozhraní

V této kapitole:
- Námět příkladu
- Rozbor řešení
- Návrh uživatelského rozhraní
- Programování aplikační logiky
- Domácí úkol
- Shrnutí

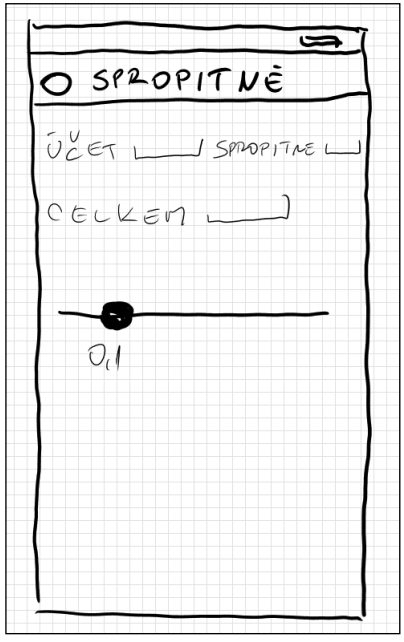
Možná jste si při procházení příkladu v předchozí kapitole řekli, že vývojové prostředí Eclipse vám práci příliš neulehčilo. S výjimkou průvodce na vytvoření projektu a průvodce na vytvoření nové aktivity sloužilo vývojové prostředí víceméně jako průzkumník a editor. Možnosti vizuálního návrhu uživatelského rozhraní jsme v prvním příkladu úmyslně nevyužili. Důvod byl jednoduchý. Při „manuální" tvorbě XML dokumentu s definicí uživatelského rozhraní jste si uvědomovali všechny souvislosti, které potřebujete znát.

Námět příkladu

Abychom ukázali možnosti tvorby uživatelského rozhraní v interaktivním návrhovém prostředí, vytvoříme podobný příklad. Tentokrát bude jeho námětem jednoduchá pomůcka na výpočet spropitného v restauraci. Aplikace bude mnohem interaktivnější, co se týká zadávání údajů. Zároveň ukážeme postup, jakým si aplikace může uchovat údaje, se kterými naposledy pracovala. Nepůjde o databázi či soubor se stovkami položek – tuto problematiku budeme řešit v dalších kapitolách. Této aplikaci bude stačit pamatovat si tři údaje: výšku účtu před připočítáním spropitného, procento, které míníte věnovat obsluze jako spropitné, případně i celkovou výšku účtu po připočítání spropitného, i když tuto hodnotu dokáže aplikace hravě vypočítat z prvních dvou údajů.

Rozbor řešení

Příklad bude odlišný i koncepcí. Jelikož máme předem jasné, jak bude aplikace fungovat a jaké texty bude vypisovat, připravíme si všechny potřebné podklady, v tomto případě návrh designu obrazovky a texty, v souboru *res/strings.xml*, předem. Aplikace bude fungovat interaktivně a bude využívat jedinou aktivitu na zadání údajů i pro zobrazení výsledků.



Obrázek 4.1: Skica uživatelského rozhraní aplikace Spropitné

Výška účtu před rozšířením o spropitné bude jediným údajem, který se bude zadávat z klávesnice jako desetinné číslo. K zadávání procentuální hodnoty spropitného se bude více hodit prvek simulující posuvný potenciometr. Předběžně si načrtneme rozmístění ovládacích prvků aplikace. Moderní tablety s dotykovou obrazovkou umožňují kreslit, proč toho tedy nevyužít?

Vytvořte v Eclipse nový projekt s názvem Spropitné. V tomto příkladu jsme jako nejnižší podporovanou verzi nastavili Api 14: Android 4.0 (IceCream Sandwich). Aplikace nebude využívat žádné speciality verze 4.0, můžete proto nastavit i API 8: Android 2.2 (Froyo) bez nutnosti využití knihoven kompatibility. Jako název aplikace jsme zadali Spropitné, název projektu `Spropitne` (bez diakritiky) a název balíčku `com.lubolacko.spropitne`. Vyhneme se cvičné doméně `example`, kdybychom náhodou vytvořili aplikaci, kterou bychom chtěli publikovat do aplikačního obchodu Google Play.

Vyberte typ hlavní aktivity **Empty Activity**. Jako název aktivity zadejte **Spropitne**. Ponechejte odvozený název layoutu `activity_spropitne`.

Po vytvoření projektu vývojové prostředí zobrazí návrhové zobrazení hlavní a jediné aktivity. Na ploše obrazovky je prvek `TextView` se statickým nápisem `Hello World`. Po letmém nahlédnutí do XML kódu návrhu layoutu hlavní aktivity zjistíte, že ovládací prvky budou zapouzdřeny v kontejneru typu `Relative Layout`. V úvodu příkladu jsme deklarovali, že uživatelské rozhraní vytvoříme v návrhovém zobrazení, takže se přepněte zpět na kartu **Graphical Layout**. Jako první krok vymažte prvek `TextView` s nápisem `Hello World`.

Návrh uživatelského rozhraní

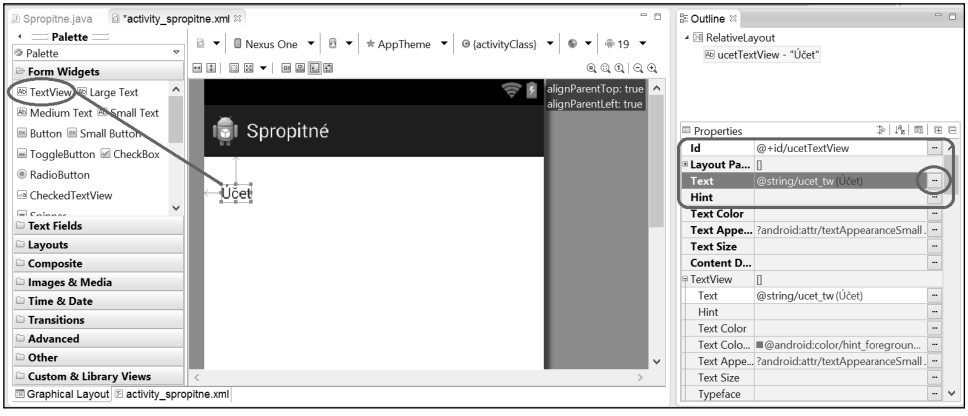
Při návrhu budeme postupovat koncepčně, proto dříve než začneme rozmísťovat prvky uživatelského rozhraní, zadáme všechny statické texty, které budou tyto prvky využívat, do zdrojů – do souboru *res/strings.xml*.

```
<resources>
  <string name="app_name">Spropitné</string>
  <string name="action_nastaveni">Nastav</string>
  <string name="ucet_tw">Účet</string>
  <string name="ucet_et">0.0</string>
  <string name="spropitne_tw">Spropitné</string>
  <string name="spropitne_et">.1</string>
  <string name="celkem_tw">Celkem</string>
```

```
  <string name="celkem_et">0.0</string>
  <string name="zmena_spropitne_tw">Uprav spropitné</string>
</resources>
```

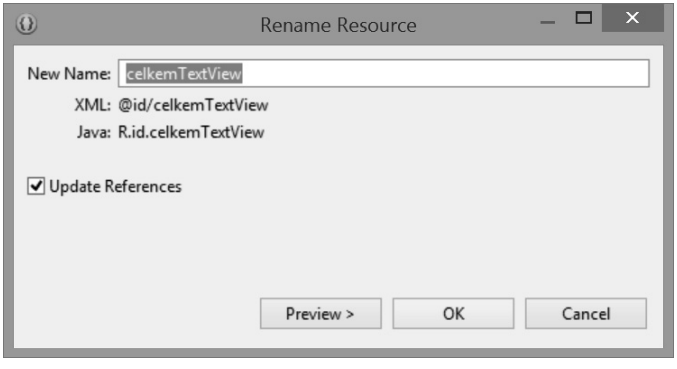
Následně budeme při zadávání prvků přiřazovat texty existujícím řetězcům ve zdrojích v návrhovém zobrazení.

Umístěte na plochu prvek `TextView`. Podívejte se na náčrt uživatelského rozhraní, jedná se o prvek s textem **Účet** vlevo nahoře. Zelené šipky a okno s relativními vzdálenostmi v dp určují polohu prvku.



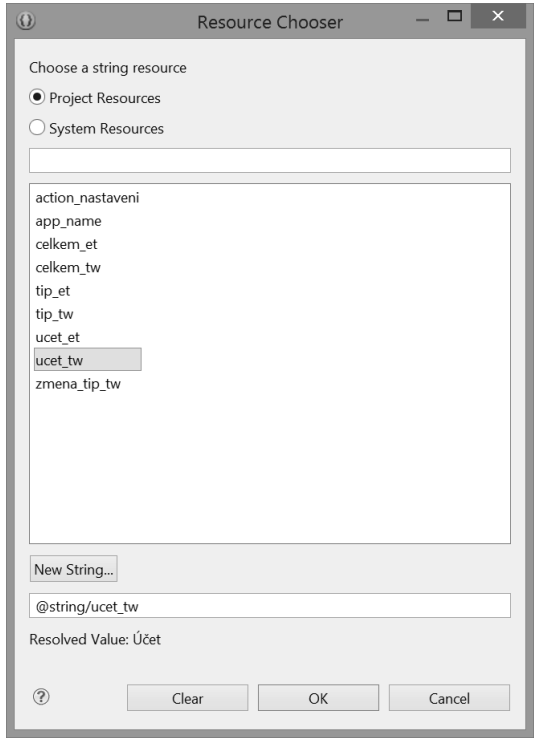
Obrázek 4.2: Návrh uživatelského rozhraní

Po umístění prvku změňte v okně **Properties** jeho **id** na `@+id/ucetTextView`. Po změně **id** budete dotázáni, zda chcete změnit reference na prvek v celém projektu. Tuto změnu potvrďte.



Obrázek 4.3: Souhlas se změnou referencí při změně názvu prvku v návrhovém zobrazení

Ještě je potřeba propojit parametr `text` s textovým řetězcem ve zdrojích. Všimněte si v okně **Properties** vedle parametru **Text** úplně vpravo tlačítka se třemi tečkami. Pomocí tohoto tlačítka zobrazíte dialogové okno **Resource Chooser** se seznamem textových řetězců. Přiřaďte prvku `TextView` řetězec s identifikátorem `ucet_tw`.

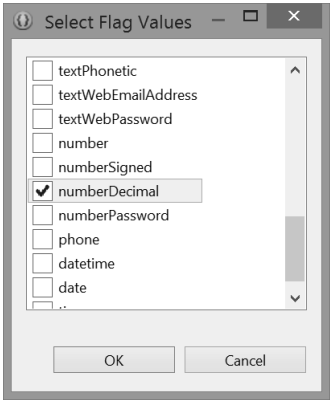


Obrázek 4.4: Přiřazení textu ze zdrojů

Pro zajímavost se můžete podívat do XML souboru s definicí layoutu, jak se vámi provedené změny projevily v návrhovém kódu. Poloha prvku `TextView` je definovaná parametry `layout_marginLeft` a `layout_marginTop` vůči rodičovskému kontejneru.

```
<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/ucetTextView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentLeft="true"
  android:layout_alignParentTop="true"
  android:layout_marginLeft="15dp"
  android:layout_marginTop="15dp"
  android:text="@string/ucet_tw" />
</RelativeLayout>
```



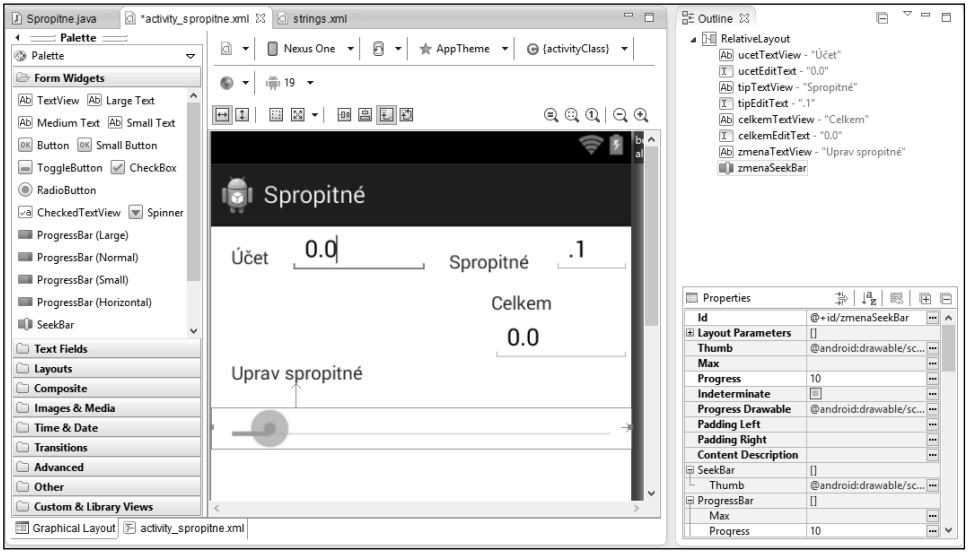
Obrázek 4.5: Výběr typu zadávaných hodnot pro prvek EditText

Pokračujte v návrhu přidáním prvku typu `EditText` napravo od prvku `TextView`. Ve složce **Text Fields** je několik variant tohoto prvku podle toho, jaký údaj budete zadávat. Použijte prvek **Plain Text**.

Podobně jako u prvku `TextView` změňte v okně **Properties** `ID` a parametr `Text`, tedy text, který se zobrazí uvnitř prvku `EditView` jako implicitní hodnota. Jelikož do tohoto prvku budete zadávat pouze číselné hodnoty s desetinnou čárkou, klepněte na tlačítko se třemi tečkami vpravo od parametru **Input Type** a v dialogovém okně **Select Flag Values** zaškrtněte volbu **numberDecimal**.

Šířka prvku je implicitně definovaná 10 ems, což odpovídá 10 znakům. Sedmiciferné sumy (+ desetinná čárka a dvě místa za desetinnou čárkou) platit v restauracích nebudete, proto můžete parametr **ems** změnit na hodnotu 5.

Pokračujte v návrhu uživatelského rozhraní prvky na zobrazení procenta spropitného a finální částky na účtu po přičtení spropitného. Všímejte si zelené šipky, která zobrazuje relativní polohu právě přidávaného prvku vůči ostatním prvkům. Na změnu procenta spropitného je použit prvek `SeekBar`.



Obrázek 4.6: Finální fáze návrhu uživatelského rozhraní – přidání prvku SeekBar

Výsledkem vizuálního návrhu byl v našem případě XML kód v souboru *activity_spropitne.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/ucetTextView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignParentLeft="true"
  android:layout_alignParentTop="true"
  android:layout_marginLeft="15dp"
  android:layout_marginTop="15dp"
  android:text="@string/ucet_tw" />

<EditText
  android:id="@+id/ucetEditText"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginLeft="15dp"
  android:layout_toRightOf="@+id/ucetTextView"
  android:ems="5"
  android:inputType="numberDecimal"
  android:text="@string/ucet_et" >
  <requestFocus />
</EditText>

<TextView
  android:id="@+id/spropitneTextView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignBottom="@+id/ucetEditText"
  android:layout_marginLeft="15dp"
  android:layout_toRightOf="@+id/ucetEditText"
  android:text="@string/spropitne_tw" />

<EditText
  android:id="@+id/spropitneEditText"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_above="@+id/celkemTextView"
  android:layout_marginLeft="18dp"
  android:layout_toRightOf="@+id/spropitneTextView"
  android:ems="4"
  android:inputType="numberDecimal"
  android:text="@string/spropitne_et" />

<TextView
  android:id="@+id/celkemTextView"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_alignLeft="@+id/celkemEditText"
  android:layout_below="@+id/ucetEditText"
```

```
    android:layout_marginTop="14dp"
    android:text="@string/celkem_tw" />

  <EditText
    android:id="@+id/celkemEditText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/spropitneEditText"
    android:layout_below="@+id/celkemTextView"
    android:ems="5"
    android:inputType="numberDecimal"
    android:text="@string/celkem_et" />

  <TextView
    android:id="@+id/zmenaTextView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/ucetTextView"
    android:layout_below="@+id/celkemEditText"
    android:text="@string/zmena_spropitne_tw" />

  <SeekBar
    android:id="@+id/zmenaSeekBar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/zmenaTextView"
    android:layout_marginTop="17dp"
    android:progress="10" />

</RelativeLayout>
```

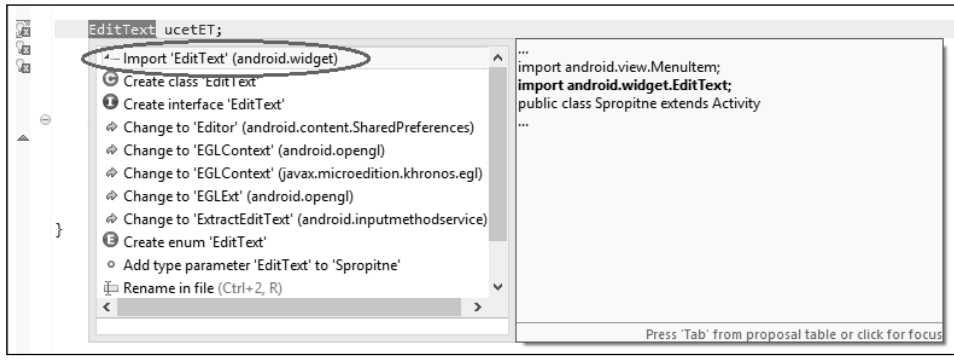
Programování aplikační logiky

Po návrhu prvků tvořících uživatelské rozhraní přenesme pozornost do souboru *Spropitne.java*, ve kterém budeme programovat funkcionalitu aplikace. Nebude to však jen triviální výpočet přičtení procentuální hodnoty, zaměříme se na ukládání klíčových údajů aplikace.



Poznámka: Naším cílem bude, aby uživatel pokaždé našel aplikaci v takovém stavu, v jakém ji zanechal.

V předchozí kapitole jsme vás upozorňovali na klávesovou zkratku **Ctrl+Shift+O**, pomocí které je možné automaticky doplnit do kódu všechny chybějící reference jmenných prostorů. K tomuto účelu však můžete použít i místní nabídku chybové ikony. V řádku javového kódu, ve kterém vývojové prostředí při průběžné kontrole syntaxe zjistí chybu, přidá ikonku s červeným křížkem. V místní nabídce této ikonky se dozvíte více podrobností o chybě, a pokud je to chybějící reference na jmenný prostor, je v nabídce položka k přidání.



Obrázek 4.7: Přidání reference přes místní nabídku

Aby bylo možné v metodě `onSaveInstanceState()` ukládat stavy proměnných, je potřeba definovat jednak samotné proměnné a také textové řetězce s názvy.

```
private static final String UCET = "UCET";
private static final String CELKEM = "CELKEM";
private static final String SPROPITNE = "SPROPITNE";

private double dUcet;
private double dCelkem;
private double dSpropitne;
```

Kód metody `onSaveInstanceState()`, ve které se při přechodu aplikace do neaktivního stavu uloží hodnoty proměnných, volá nejprve metodu rodičovské třídy.

```
protected void onSaveInstanceState(Bundle outState)
{
  super.onSaveInstanceState(outState);
  outState.putDouble(CELKEM, dCelkem);
  outState.putDouble(SPROPITNE, dSpropitne);
  outState.putDouble(UCET, dUcet);
}
```

Údaje se načítají v metodě `onCreate()`, samozřejmě pouze v případě, že už byly předtím uloženy. Pokud uložené nebyly, například při prvním spuštění aplikace, proměnné se nastaví na inicializační hodnoty. V metodě `onCreate` se přiřazují i vizuální prvky podle identifikátoru a definují listenery.

```
EditText ucetET;
EditText celkemET;
EditText spropitneET;
SeekBar  zmenaSB;

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

```
  if(savedInstanceState == null)
  {
    dUcet = 0.0;
    dCelkem = 0.0;
    dSpropitne = .15;
  }
  else
  {
    dUcet = savedInstanceState.getDouble(UCET);
    dCelkem = savedInstanceState.getDouble(CELKEM);
    dSpropitne = savedInstanceState.getDouble(SPROPITNE);
  }

  // Výše účtu
  ucetET = (EditText) findViewById(R.id.ucetEditText);
  // Výše spropitného
  spropitneET = (EditText) findViewById(R.id.spropitneEditText);
  // Účet + spropitné
  celkemET = (EditText) findViewById(R.id.celkemEditText);
  zmenaSB = (SeekBar) findViewById(R.id.zmenaSeekBar);

  zmenaSB.setOnSeekBarChangeListener(spropitneSBListener);
  ucetET.addTextChangedListener(ucetListener);
}
```

V aplikaci jsou použity dva listenery. Jeden slouží ke sledování změny hodnoty zadané v prvku typu `EditText`, pomocí kterého uživatel zadává výšku účtu. Druhý listener slouží ke sledování změny polohy prvku `SeekBar`, pomocí něhož uživatel zadává procento spropitného.

K doplnění kostry kódu metod listeneru vám poradíme další z mnoha funkcí na zjednodušení vývoje, kterými disponuje vývojové prostředí Eclipse. Napište prázdnou kostru kódu listeneru a umístěte kurzor na jeho název. Ve žlutém obdélníku se zobrazí místní nabídka s volbou **Add unimplemented methods**. Po klepnutí na volbu budou doplněny metody `beforeTextChanged()`, `onTextChanged()` a `afterTextChanged()`.

```
private TextWatcher ucetListener = new TextWatcher()
{ };
```



Obrázek 4.8: Přidání neimplementovaných metod listeneru

V aplikaci využijete jen metodu `onTextChanged()`, ve které se do proměnné `dUcet` načítá nová, uživatelem zadaná hodnota. I když je uživatel při zadávání omezen tím, že může v prvku `EditText` zadávat jen číselné hodnoty, v reálné aplikaci doporučujeme převod z textového řetězce na `double` umístit do sekce `try – catch` na ošetření případných výjimek.

```
private TextWatcher ucetListener = new TextWatcher()
{
  @Override
  public void beforeTextChanged(CharSequence s, int start, int count,
    int after) {// TODO Auto-generated method stub
  }

  @Override
  public void onTextChanged(CharSequence s, int start, int before,
    int count)
  {// TODO Auto-generated method stub
    try
    {
      dUcet = Double.parseDouble(s.toString());
    }
    catch(NumberFormatException e){dUcet = 0.0;}
    aktualizujHodnoty();
  }

  @Override
  public void afterTextChanged(Editable s)
  {
    // TODO Auto-generated method stub
  }
};
```

Následně se zavolá metoda na aktualizaci hodnot.

```
private void aktualizujHodnoty()
{
  double dSpropitne = Double.parseDouble(spropitneET.getText().toString());
  double dCelkem = dUcet + (dUcet * dSpropitne);
  celkemET.setText(String.format("%.02f", dCelkem));
}
```

Zbývá implementovat poslední funkcionalitu, propojení prvku `SeekBar` (vizuální analogie posuvného potenciometru) s prvkem typu `EditText`. Podobně jako u listeneru pro `TextWatcher` poradíme vám i v tomto případě další z mnoha funkcí na zjednodušení vývoje, kterými disponuje vývojové prostředí Eclipse. Napište prázdnou kostru kódu listeneru a umístěte kurzor na jeho název. Ve žlutém obdélníku se zobrazí místní nabídka s volbou **Add unimplemented methods**. Po klepnutí na volbu budou doplněny metody `beforeTextChanged()`, `onTextChanged()` a `afterTextChanged()`.

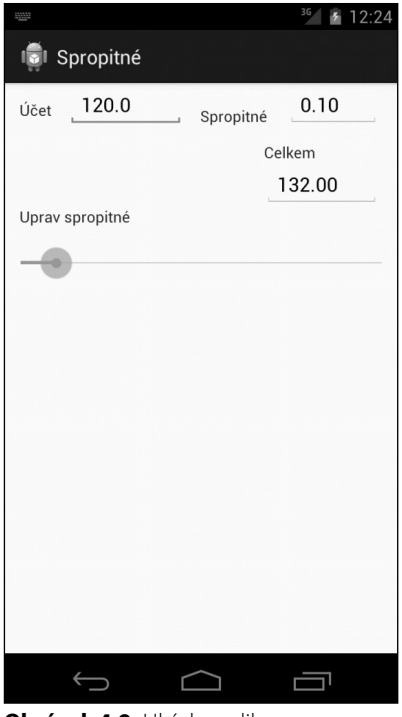
```
private OnSeekBarChangeListener spropitneSBListener = new
OnSeekBarChangeListener()
{
  Override
  public void onProgressChanged(SeekBar seekBar, int progress,
    boolean fromUser)
  {
    // TODO Auto-generated method stub
    dSpropitne = (zmenaSB.getProgress()) * .01;
    spropitneET.setText(String.format("%.02f", dSpropitne));
```

```
    aktualizujHodnoty();
  }

  @Override
  public void onStartTrackingTouch(SeekBar seekBar)
  {
    // TODO Auto-generated method stub
  }

  @Override
  public void onStopTrackingTouch(SeekBar seekBar)
  {
    // TODO Auto-generated method stub
  }
};
```

V této fázi můžete aplikaci otestovat, hlavně co se týká automatického výpočtu celkové výšky účtu po změně procenta spropitného.



Obrázek 4.9: Ukázka aplikace

Domácí úkol

Pokud jste aplikaci otestovali důkladně, zjistili jste, že sice můžete zadat hodnotu spropitného i do prvku `EditText`, změna této hodnoty se ale nesleduje, takže po změně procenta spropitného se nevypočítá nová celková hodnota účtu. Doplňte do aplikace listener, který bude sledovat i změnu hodnoty v tomto prvku a vypočítá hodnotu, kterou je potřeba zaplatit. Také je potřeba změnit polohu prvku `SeekBar`, aby korespondovala s novou hodnotou.

Shrnutí

Po absolvování příkladu popsaného v této kapitole byste měli rutinně ovládat návrh uživatelského rozhraní ve vývojovém prostředí Eclipse.

KAPITOLA 5

Interakce s uživatelem

V této kapitole:
- Nabídka funkcionality – menu
- Navigace pomocí lišty Action Bar
- Dotyky a gesta
- Příklad – zobrazení vícenásobných dotyků
- Detekce standardně používaných gest
- Příklad – univerzální počítadlo
- Příklad – využití vlastních gest

Nabídka funkcionality – menu

Nabídka funkcionality může být v aplikaci pro Android uživateli prezentovaná několika způsoby. K dispozici jsou tři typy menu (nabídek):

- **Options Menu** – zobrazí se, pokud uživatel stiskne tlačítko menu na liště Action Bar (nebo hardwarové tlačítko na starších zařízeních). Toto menu slouží k navigaci uvnitř aplikace.
- **Kontextové menu** – váže se k prvku uživatelského rozhraní. Zobrazí se zpravidla podržením prstu na příslušném prvku.
- **Popup Menu** – na rozdíl od kontextového menu, které aktivuje uživatel, Popup Menu se zobrazí samo v návaznosti na aplikační logice, pokud je vyžadována součinnost uživatele. Menu je vizuálně ukotveno k prvku, ke kterému se váže. Naviguje 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 aktivity hlavní.

Navigace pomocí lišty Action Bar

Verze 3.0 Honeycomb přinesla takzvaný Action Bar – lištu 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. Action Bar je pro uživatele jakýmsi záchytným bodem, standardizovaným prvkem umožňujícím rychle si osvojit ovládání aplikace. Operační systém navíc Action Bar elegantně přizpůsobí různým konfiguracím obrazovky.

Vytvořte projekt aplikace, přičemž parametr **Minimum Required SDK** zadejte na hodnotu API 14: Android 4.0 (IceCream Sandwich) a nastavte jako typ hlavní aktivity **Blank Activity**. Action Bar je totiž už součástí této šablony aktivity. Parametr **Theme** ponechejte na implicitně nastavené hodnotě **Holo Light with Dark Action Bar**.



Poznámka: Pokud potřebujete vytvořit aplikaci kompatibilní s nižšími verzemi Androidu, nastavte nejnižší verzi API 8: Android 2.2 Froyo. Všimněte si ve složce *Android Private Libraries*, že součástí projektu je i knihovna v7 appcompat.

Eclipse v projektu typu **Blank Activity** vytvoří automaticky u nového projektu i metodu pro Options Menu:

```
public boolean onCreateOptionsMenu(Menu menu)
{
  getMenuInflater().inflate(R.menu.activity_main, menu);
  return true;
}
```

Pokud uživatel po spuštění aplikace aktivuje menu, Android volá v závislosti na jeho typu metodu `onCreateOptionsMenu()`, případně `onCreateContextMenu()`, v které pomocí objektu `MenuInflater` menu naplní 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 *activity_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:tools="http://schemas.android.com/tools"
  tools:context="com.example.navigace.MainActivity" >

  <item android:id="@+id/action_vyhledavani"
    android:icon="@drawable/ic_action_search"
    android:title="@string/action_vyhledavani"
    android:showAsAction="ifRoom"/>

  <item android:id="@+id/action_foto"
    android:icon="@drawable/ic_action_camera"
    android:title="@string/action_foto"
    android:showAsAction="ifRoom" />

  <item android:id="@+id/action_nastaveni"
    android:icon="@drawable/ic_action_settings"
```

```
    android:title="@string/action_nastaveni"
    android:showAsAction="ifRoom" />

  <item android:id="@+id/action_napoveda"
    android:icon="@drawable/ic_action_help"
    android:title="@string/action_napoveda"
    android:showAsAction="never"/>

  <item android:id="@+id/action_info"
    android:icon="@drawable/ic_action_about"
    android:title="@string/action_info"
    android:showAsAction="never" />

</menu>
```

Všimněte si atributu `showAsAction`. Hodnota `never` definuje, že položka se zobrazí jen v menu, a ne na liště Action Bar. 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, protože aplikace s více položkami, které by se normálně na displej nevešly, pravděpodobně nebude vypadat dobře.



Poznámka: Pokud vaše aplikace využívá knihovny Support Library pro kompatibilitu se staršími verzemi Androidu, nemůžete použít atribut **`showAsAction`** z jmenného prostoru **`android:`**, ale musíte jej vzít z jmenného prostoru vaší aplikace.

Do složky *res/drawable/* je potřeba přidat ikony pro položky menu na Action Bar. 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.

Nezapomeňte do zdrojů doplnit texty řetězců.

```
<resources>
  ...
  <string name="action_vyhledavani">Vyhledávání</string>
  <string name="action_foto">Foto</string>
  <string name="action_nastaveni">Nastavení</string>
  <string name="action_napoveda">Nápověda</string>
  <string name="action_info">O aplikaci</string>
</resources>
```

Po vytvoření projektu se předpokládá, že Action Bar bude generovaný přes knihovnu pro zachování kompatibility s nižšími verzemi Androidu. Hlavní aktivita je odvozená od třídy `ActionBarActivity`:

```
public class MainActivity extends ActionBarActivity
```

Třída využívá import:

```
import android.support.v7.app.ActionBar
```

Pokud je vaše aplikace určena pouze pro API 11: Android 3.0 (Honeycomb) a vyšší, použijte import:

```
import android.app.ActionBar
```

V kódu aktivity upravte metodu `onCreateOptionsMenu()` na vytvoření menu:

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.main, menu);
  return super.onCreateOptionsMenu(menu);
}
```

Dále upravte metodu `onOptionsItemSelected()`, v které se zpracovává případné klepnutí na některý z prvků na liště Action Bar. Implicitní kód šablony vložený při vytváření projektu přepište kódem obsahujícím definování akcí, které se mají realizovat při výběru položky. Aktivní bude zatím pouze položka `Nastavení`.

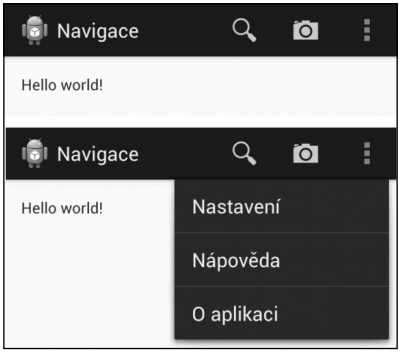
```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
  switch (item.getItemId())
  {
    case R.id.action_vyhledavani:
      return true;
    case R.id.action_foto:
      return true;
    case R.id.action_nastaveni:
      aktivujNastaveni();
      return true;
    case R.id.action_napoveda:
      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 `aktivujNastaveni()` jsme vypsali jen text přes Toast.

```
public void aktivujNastaveni()
{
  Toast.makeText(this, "Nastavení.",Toast.LENGTH_SHORT).show();
}
```

Po spuštění aplikace stačí aktivovat nabídku pomocí tlačítka v pravé horní části lišty.



Obrázek 5.1: Zobrazení nabídky na liště Action Bar. V dolní části je rozbalená nabídka.

Přizpůsobení a rozdělení lišty Action Bar

Action Bar 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);
```

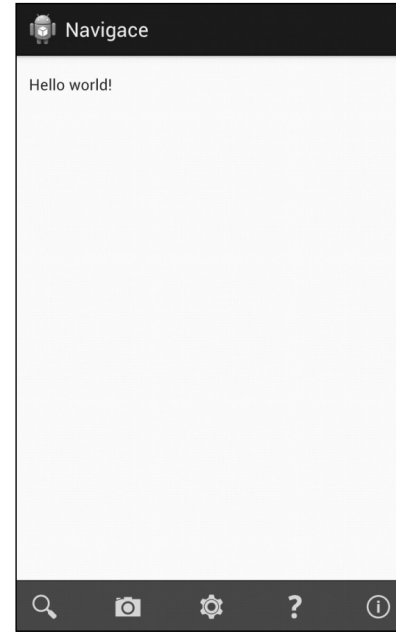
Pokud lišta Action Bar obsahuje více tlačítek nebo jiných ovládacích prvků, na zařízeních s úzkým displejem v režimu „na výšku“ může nastat problém, aby se prvky na lištu vešly. V takovém případě je možné rozdělit Action Bar na dvě lišty – jedna zůstane v horní části displeje a druhá, rozšířená lišta, se zobrazí u spodního okraje. Možnost rozdělit Action Bar 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 5.2: Po rozdělení lišty Action Bar se zobrazí tlačítka nabídek na přídavné liště v dolní části obrazovky

Navigace na jinou aktivitu (a zpět)

Lišta Action Bar 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. Novou aktivitu vytvoříte ve vývojovém prostředí Eclipse nejsnáze pomocí tlačítka **New** (na panelu nástrojů úplně vlevo) a následně ve složce **Android** zvolte položku **Android 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>
```

V souboru *NapovedaActivity.java* je potřeba v metodě `onCreate()` volat metodu `setDisplay HomeAsUpEnabled()`, která zobrazí tlačítko pro návrat do rodičovské aktivity.

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_napoveda);
  ActionBar actionBar = getActionBar();
  actionBar.setDisplayHomeAsUpEnabled(true);
}
```

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;
...
```

Po spuštění aplikace vyzkoušejte návrat na rodičovskou aktivitu.



Obrázek 5.3: Spuštění dceřiné aktivity. Všimněte si na liště vlevo ikony se šipkou umožňující návrat do rodičovské aktivity.

Přidání widgetu na lištu Action Bar

Přidání tlačítka s ikonou vyhledávání nabylo náhodné, vyhledávání je dobrým reálným příkladem z praxe na přidání prvku přímo na plochu lišty Action Bar. Uživatel po aktivaci vyhledávání logicky očekává, že se mu zobrazí prvek na zadání vyhledávací fráze.



Poznámka: V této aplikaci nám půjde o přidání widgetu na lištu Action Bar, ne o implementaci mechanizmu vyhledávání.

V definičním XML souboru *Main.xml* ve složce *menu* upravte položku pro vyhledávání – atribut `android:actionViewClass` nastavte na hodnotu `android.widget.SearchView`.

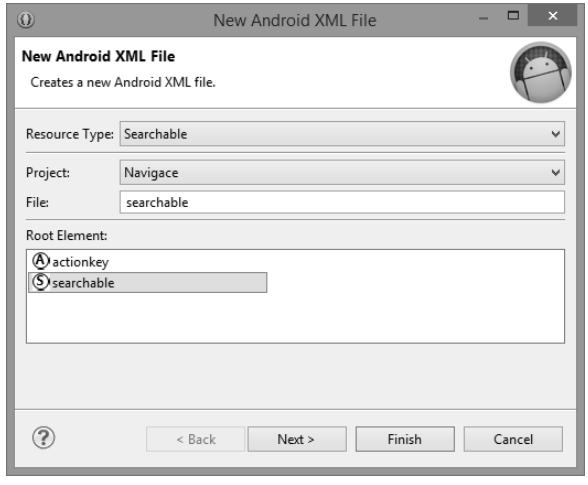
```
<item android:id="@+id/action_vyhledavani"
  android:icon="@drawable/ic_action_search"
  android:title="@string/action_vyhledavani"
  android:actionViewClass="android.widget.SearchView"
  android:showAsAction="always"/>
```



Poznámka: Pokud potřebujete dosáhnout kompatibility aplikace se staršími verzemi Androidu, použijte atribut `app:actionViewClass="android.support.v7.widget.SearchView"`.

Implementace mechanizmu vyhledávání vyžaduje vytvoření definičního souboru například s názvem *searchable.xml* ve složce *res/xml*. Pokud ve vašem projektu složka *res/xml* nebyla vytvořena, musíte ji nejprve vytvořit.

Definiční soubor pro vyhledávání vytvoříte nejjednodušeji pomocí nabídky **New → Android XML File**. V dialogovém okně vytvoření nového XML souboru nastavte položku **Resource Type** na hodnotu **searchable** a **Root Element** vyberte **searchable**.



Obrázek 5.4: Vytvoření definičního souboru pro vyhledávání

Jelikož v příkladu nebudete implementovat skutečný mechanizmus vyhledávání, postačí jednoduchý definiční soubor.

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
  android:label="@string/app_name"
  android:hint="@string/search_hint">
</searchable>
```

V kódu hlavní aktivity doplňte do metody `onCreateOptionsMenu()` inicializaci `SearchView`.

```
public boolean onCreateOptionsMenu(Menu menu)
{
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.main, menu);

  SearchManager searchManager = (SearchManager)
    getSystemService(Context.SEARCH_SERVICE);
  SearchView searchView = (SearchView)
    menu.findItem(R.id.action_vyhledavani).getActionView();
  searchView.setSearchableInfo(
    searchManager.getSearchableInfo(getComponentName()));

  return super.onCreateOptionsMenu(menu);
}
```

Vytvořte novou aktivitu na zobrazení nalezených výsledků vyhledávání, například s názvem `VysledkyActivity`. V této aplikaci jsme neimplementovali mechanizmus vyhledávání, proto vypíšeme jen uživatelem zadanou frázi. Definice uživatelského rozhraní bude v souboru *Activity_vysledky.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">

  <TextView
    android:id="@+id/textVysledky"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

</RelativeLayout>
```

Kód v souboru *VysledkyActivity.java*:

```
public class VysledkyActivity extends Activity
{

  private TextView txtVysledky;

  @Override
  protected void onCreate(Bundle savedInstanceState)
```

```
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_vysledky);
    ActionBar actionBar = getActionBar();
    actionBar.setDisplayHomeAsUpEnabled(true);
    txtVysledky = (TextView) findViewById(R.id.textVysledky);
    handleIntent(getIntent());
  }

  @Override
  protected void onNewIntent(Intent intent)
  {
    setIntent(intent);
    handleIntent(intent);
  }


  private void handleIntent(Intent intent)
  {
    if (Intent.ACTION_SEARCH.equals(intent.getAction()))
    {
      String query = intent.getStringExtra(SearchManager.QUERY);
      txtVysledky.setText("Dotaz: " + query);
    }
  }
}
```

Nakonec je potřeba doplnit manifest v souboru *AndroidManifest.xml*. Do elementu hlavní aktivity doplňte element definující propojení na vyhledávání.

```
<meta-data
  android:name="android.app.default_searchable"
  android:value=".VysledkyActivity" />
```

- Parametr `android.app.default_searchable` můžete definovat buď uvnitř tagu `<application>`, nebo v tagu `<activity>`.
- Parametr `android.app.searchable` v aktivitě na zobrazování výsledků vyhledávání odkazuje na soubor *searchable.xml*.
- Parametr `android.intent.action.SEARCH` definuje `intent-filter`.

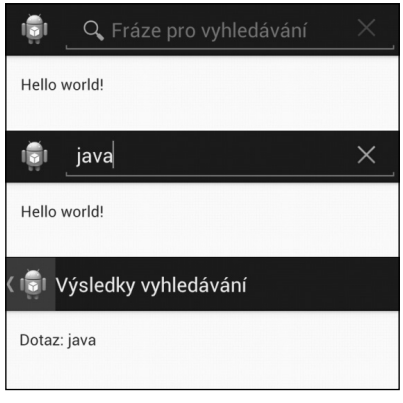
Kompletní kód elementu `application` bude:

```
<application
  android:allowBackup="true"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
  <activity
    android:name="com.example.navigace.MainActivity"
    android:label="@string/app_name" >
    <meta-data
      android:name="android.app.default_searchable"
      android:value=".VysledkyActivity" />
```

```
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
  <activity
    android:name="com.example.navigace.NapovedaActivity"
    android:label="@string/title_activity_napoveda"
    android:parentActivityName="com.example.navigace.MainActivity" >
  </activity>
  <activity
    android:name="com.example.navigace.VysledkyActivity"
    android:label="@string/title_activity_vysledky"
    android:parentActivityName="com.example.actionbar.MainActivity" >
    <intent-filter>
      <action android:name="android.intent.action.SEARCH" />
    </intent-filter>
    <meta-data
      android:name="android.app.searchable"
      android:resource="@xml/searchable" />
  </activity>
</application>
```

Pro úplnost uvádíme i definice textových řetězců v souboru *strings.xml*.

```
<resources>
  <string name="app_name">Navigace</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_vyhledavani">Vyhledávání</string>
  <string name="action_foto">Foto</string>
  <string name="action_nastaveni">Nastavení</string>
  <string name="action_napoveda">Nápověda</string>
  <string name="action_info">O aplikaci</string>
  <string name="title_activity_napoveda">Nápověda</string>
  <string name="text_napovedy">Tady bude text nápovědy</string>
  <string name="search_hint">Fráze pro vyhledávání</string>
  <string name="title_activity_vysledky">Výsledky vyhledávání</string>
  <string name="action_settings">Settings</string>
</resources>
```



Obrázek 5.5: Vyhledávání

Dotyky a gesta

Na rozdíl od klasických aplikací, které jsou koncipované na ovládání klávesnicí a myší, telefony a tablety s operačním systémem Android se ovládají prostřednictvím dotykového displeje.



Poznámka: Zařízení, která mají na externí nebo zabudované klávesnici touchpad, mají implementovaný i kurzor myši. Některá zařízení disponují trackballem.

Typickým příkladem je tablet Lenovo Yoga 10, 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.



Poznámka: Implicitní ovládání platformy Android je 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).



Poznámka: Jako nejjednodušší příklad na demonstraci dotyků se nabízí kreslení prstem na displeji. Tento příklad najdete v kapitole Grafika a animace.

Multitouch

Multitouch 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.

Ke zpracování pohybových gest z různých vstupních zařízení se využívá třída `MotionEvents`. Obsahuje kód akce, tedy gesta a jeho parametry, respektive 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 Xovou souřadnici ukazatele se zadaným indexem.
- `getY(int pointerIndex)` – vrátí Yovou 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, je volána 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 zpětná volání, která budou volaná při událostech, jako je stisknutí, uvolnění nebo přetažení prstu po obrazovce. Listener se registruje pomocí metody `View.setOnTouchListener()`.



Poznámka: Metoda **`onTouch`** je volaná 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 akcí `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 vícenásobných dotyků

V prvním příkladu budeme indikovat vícenásobné dotyky na obrazovce. Podobně jako se zaběhla aplikace s námětem „Hello World" pro první aplikaci začátečníka, při vývoji na dané platformě se pro první kroky 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"
  android:id="@+id/plocha"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
</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. Pro jednoduchost a přehlednost můžete kód třídy `DotekView` umístit v souboru s kódem hlavní aktivity.

```
private class DotekView extends View
{
  private float xx, yy;
  final static private int MAX_SIZE = 400;
  private int nDoteku = 0;
  final private Paint mPaint = new Paint();

  public DotekView(Context context, float x, float y)
  {
    super(context);
    xx = x;
    yy = y;
    mPaint.setStyle(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 pocetDoteku (int doteku) {nDoteku = doteku;}

  @Override
  protected void onDraw(Canvas canvas)
  {
    canvas.drawCircle(xx, yy, MAX_SIZE / nDoteku, 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`, potom se překreslí kruhy odpovídající dotykům prstů, které se pohnuly. V aplikaci inicializujeme překreslení, jen pokud je pohyb větší než 3 dp.

```
public class MainActivity extends Activity
{
  private static final int MIN_DELTA = 2;

  final private static LinkedList<DotekView> neaktivniDoteky =
    new LinkedList<DotekView>();
  @SuppressLint("UseSparseArrays")
  final private static Map<Integer, DotekView> aktivniDoteky =
    new HashMap<Integer, DotekView>();
```

```
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 OnTouchListener()
  {
    @Override
    public boolean onTouch(View v, MotionEvent motion)
    {
      switch (motion.getActionMasked())
      {
        case MotionEvent.ACTION_DOWN:
        case MotionEvent.ACTION_POINTER_DOWN:
        {
          int index = motion.getActionIndex();
          int id = motion.getPointerId(index);

          DotekView dotek = neaktivniDoteky.remove();
          if (dotek != null)
          {
            aktivniDoteky.put(id, dotek);
            dotek.setXLoc(motion.getX(index));
            dotek.setYLoc(motion.getY(index));
            aktualizujDoteky(aktivniDoteky.size());
            frame.addView(dotek);
          }
          break;
        }

        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_POINTER_UP:
        {
          int index = motion.getActionIndex();
          int id = motion.getPointerId(index);

          DotekView dotek = aktivniDoteky.remove(id);
          if (dotek != null)
          {
            neaktivniDoteky.add(dotek);
            aktualizujDoteky(aktivniDoteky.size());
            frame.removeView(dotek);
          }
          break;
        }
```

```
          case MotionEvent.ACTION_MOVE:
          {
            for (int idx = 0; idx < motion.getPointerCount(); idx++)
            {
              int ID = motion.getPointerId(idx);

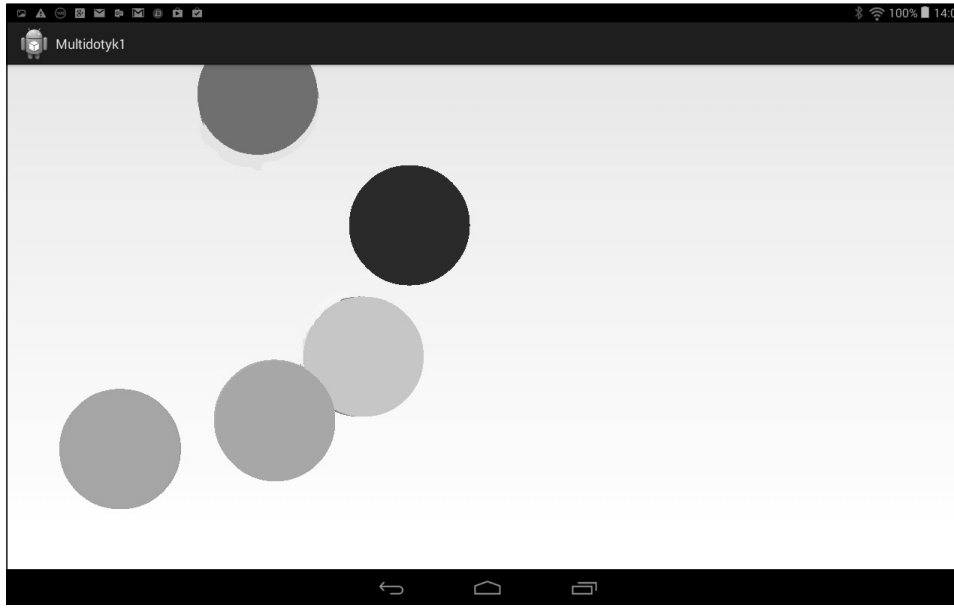
              DotekView dotek = aktivniDoteky.get(ID);
              if (null != dotek)
              {

                // pouze při pohybu nad minimální vzdálenost
                if (Math.abs(dotek.getXLoc() -
                  motion.getX(idx)) > MIN_DELTA
                  || Math.abs(dotek.getYLoc()
                  - motion.getY(idx)) > MIN_DELTA)
                {
                  dotek.setXLoc(motion.getX(idx));
                  dotek.setYLoc(motion.getY(idx));
                  dotek.invalidate();
                }
              }
            }
            break;
          }
          default:
        }
        return true;
      }

      private void aktualizujDoteky(int numActive)
      {
        for (DotekView dotek : aktivniDoteky.values())
          dotek. pocetDoteku (numActive);
      }
    });
  }

  private void initViews()
  {
    for (int idx = 0; idx < 10; idx++)
    {
      neaktivniDoteky.add(new DotekView(this, -1, -1));
    }
  }
}
```

Aplikaci otestujte tak, že nejprve přiložíte jeden prst na obrazovku a pohybujete jím. Sledujte překreslování kruhu. Potom přiložte další prsty a pohybujte jimi po obrazovce.



Obrázek 5.6: Test aplikace znázorňující vícenásobné dotyky

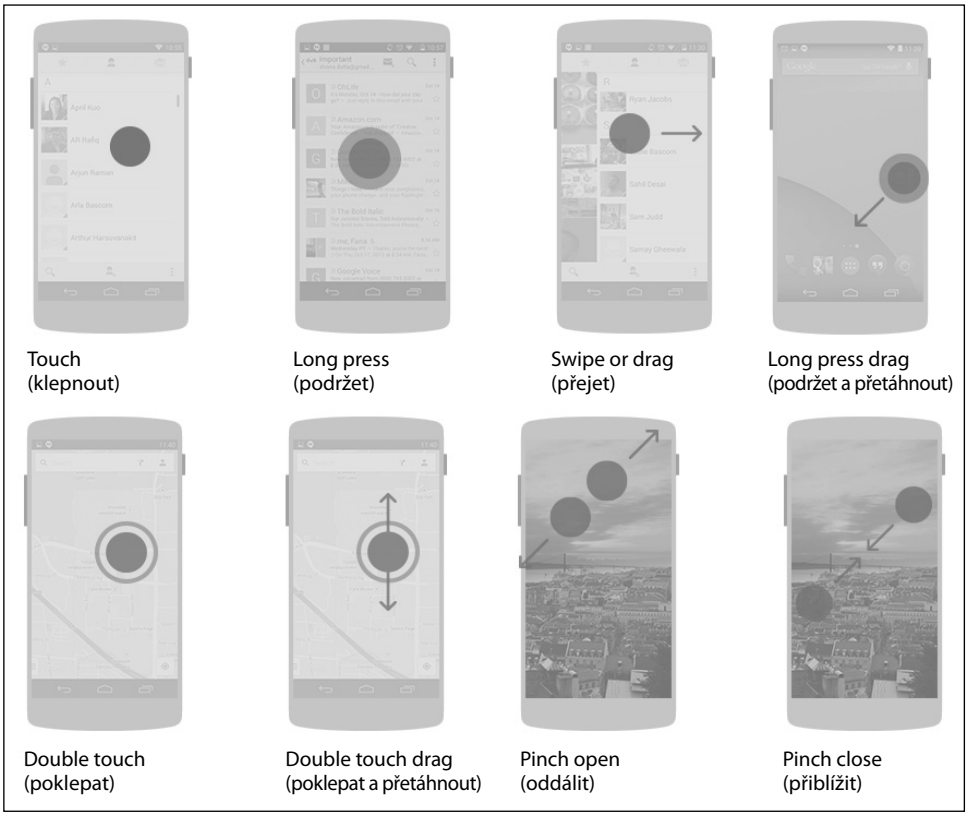
Detekce standardně používaných gest

V předchozím příkladu jsme ukázali detekci vícenásobných dotyků včetně pohybu jednotlivých prstů po obrazovce. Android využívá pro některé úkony standardní gesta, na která si uživatelé zvykli a očekávají, že je budou využívat i v aplikacích.

Na detekci gest je potřeba vytvořit instanci třídy `GestureDetector` a implementovat rozhraní `GestureDetector.OnGestureListener`. V aktivitě přepíšete metodu `onTouchEvent()`.

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, kterou 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.



Obrázek 5.7: Zaběhlá gesta a jejich významy na platformě Android

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 `View`.

```
<ViewFlipper xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/view_flipper"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <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 `TextViev`.

```
<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, tj. takzvané gesto fling, potom se aktuální `TextView` bude 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 fling, 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.



Obrázek 5.8: Test aplikace Počítadlo

Animace pohybu se realizuje přepínáním layoutů v metodě `prepniLayout()`.

```
public class MainActivity extends Activity
{

  private ViewFlipper flipper;
  private TextView textView1, textView2;
  private int stav, pocitadlo;
  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);

    stav = 0;
    flipper = (ViewFlipper) findViewById(R.id.view_flipper);
    textView1.setText(String.valueOf(pocitadlo));

    gestDetector = new GestureDetector(this,
      new GestureDetector.SimpleOnGestureListener()
    {
      @Override
      public boolean onFling(MotionEvent e1,
        MotionEvent e2,float rychlostX, float rychlostY)
      {
        if (rychlostX <-10.0f)
        {
          stav = stav == 0 ? 1: 0;
          prepniLayout(stav);
        }
        return true;
      }
    });
  }

  @Override
```

```
  public boolean onTouchEvent(MotionEvent event)
  {
    return gestDetector.onTouchEvent(event);
  }

  public void prepniLayout(int switchTo)
  {
    stav = switchTo;

    flipper.setInAnimation(animaceZprava());
    flipper.setOutAnimation(animaceZleva());

    pocitadlo++;
    if (switchTo == 0)
      textView1.setText(String.valueOf(pocitadlo));
    else
      textView2.setText(String.valueOf(pocitadlo));
    flipper.showPrevious();
  }

  private Animation animaceZprava()
  {
    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 animaceZleva()
  {
    Animation zleva = 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);
    zleva.setDuration(500);
    zleva.setInterpolator(new LinearInterpolator());
    return zleva;
  }
}
```

Příklad – využití vlastních gest

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 v „papírové“ informatice. Například zkontrolování něčeho se označí fajfkou pomo-

cí tužky nebo fixy 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 akce by měla velkou nevýhodu, že by bylo 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.

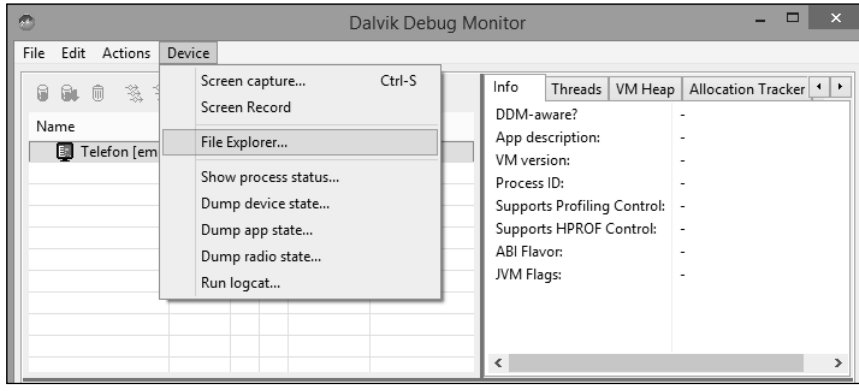
Nejprve je potřeba definovat vlastní gesta. Součástí SDK je aplikace **Gestures Builder**, která je standardní aplikační výbavou emulátoru. Spusťte emulátor a najděte v seznamu aplikací ikonu **Gestures Builder**. Je ve tvaru poznámkového bloku s řeckým písmenem alfa. Pomocí tlačítka **Add gesture** přidejte gesta, která budete v aplikaci potřebovat. V našem případě jsme definovali čtyři gesta s významem „ano“, „ne“, „další“ a „předchozí“.



Obrázek 5.9: Definice vlastních gest v aplikaci Gesture Builder

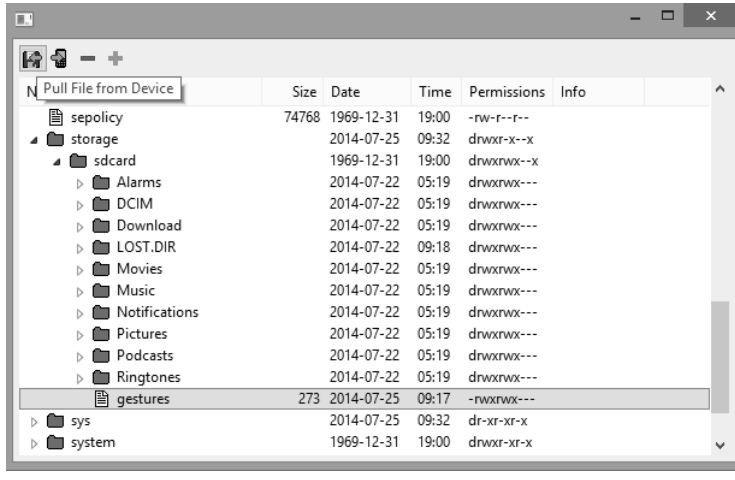
Definice gest budou v emulátoru uloženy v souboru */mnt/sdcard/gestures*. Tento soubor je potřeba zkopírovat do projektu vaší aplikace například do souboru */res/raw directory*.

Jak však zkopírovat soubor z emulátoru do hostitelského počítače? K tomuto účelu vám poslouží nástroj **Dalvik Debug Monitor**. Spustíte ho 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**.



Obrázek 5.10: Spuštění nástroje File Explorer v Dalvik Debug Monitoru

V nástroji **File Explorer** najděte soubor *gestures* a pomocí tlačítka **Pull File from Device** (první tlačítko zleva na liště s tlačítky) ho zkopírujte do vybrané složky hostitelského vývojářského počítače.



Obrázek 5.11: Kopírování souboru gestures z emulátoru do hostitelského počítače

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"
  android:id="@+id/main"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <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>
```

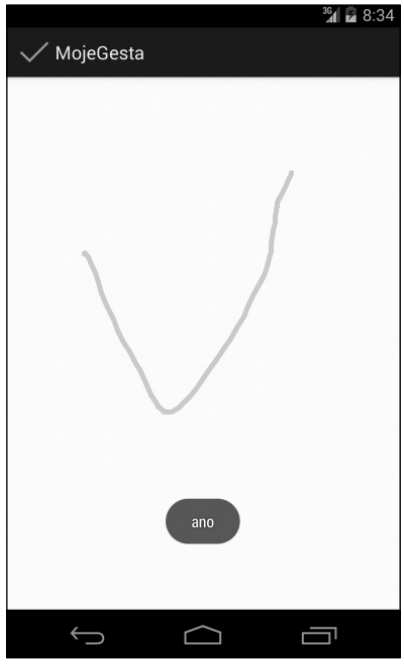
V kódu hlavní aktivity se pomocí objektu `Toast` vypíše název rozpoznaného gesta.

```
public class MainActivity extends Activity implements
  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();
    }
  }
}
```



Obrázek 5.12: Rozpoznání uživatelem zadaného gesta

KAPITOLA 6

Notifikace, alarmy

V této kapitole:
- Notifikace
- Alarmy

Notifikace

Notifikace jsou zprávy, někdy i rozsáhlejší, které potřebuje aplikace ve vhodné situaci zobrazit uživateli mimo běžné uživatelské rozhraní aplikace.

Android podporuje dva různé druhy upozornění. Zprávy Toast, jejichž název je odvozen od toho, že zpráva „vyskočí" na obrazovku aplikace doslova jako topinka z topinkovače. Zprávy Toast jako nejjednodušší prostředek na výpis textu jste už mnohokrát použili v aplikacích v předchozích kapitolách. Druhým typem je oznámení v oznamovací oblasti nebo panelu oznámení.

Tato oznámení jsou viditelná a mohou přistupovat k systémům řízení oznamovací oblasti v horní části zařízení.



Poznámka: Notifikace se často používají k upozornění uživatele, že nějaká činnost, která trvá déle, byla dokončena. Typickým příkladem je stahování souborů z Internetu.

Příklad – zobrazení toastu s vlastním designem

Klasický toast je text na pozadí, který se zobrazí v okénku. V tomto příkladu vytvoříte vlastní návrh okna toastu a kromě textu zobrazíte i obrázek. S návrhem layoutu si nemusíte dělat žádné starosti, stačí na plochu hlavní aktivity přidat tlačítko.

```
<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/btToast"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"
```

```
    android:text="Toast"/>

</LinearLayout>
```

Více pozornosti budeme věnovat návrhu vlastního layoutu toastu v souboru *muj_toast.xml*. Layout obsahuje prvek `ImageView` na zobrazení ikony a `TextView` na zobrazení textu oznámení.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/toast_layout_root"
  android:orientation="horizontal"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:padding="10dp"
  android:background="#7777" >
  <ImageView android:id="@+id/image"
    android:layout_width="64dp"
    android:layout_height="64dp"
    android:layout_marginRight="10dp"
    android:src="@drawable/ic_launcher"
    android:layout_centerVertical="true"
    android:contentDescription="popis" />
  <TextView android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FFF"
    android:text="Text oznámení!"
    android:layout_centerVertical="true"
    android:layout_toRightOf="@id/image"
    android:textSize="40sp" />
</RelativeLayout>
```



Obrázek 6.1: Zobrazení toastu s vlastním layoutem

Kód hlavní aktivity po stisknutí tlačítka zobrazí toast. V metodě `setView` se naplní layout toastu z XML souboru.

```
public class MainActivity extends Activity
{

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  public void onClick(View View)
  {
    Toast toast = new Toast(getApplicationContext());
    toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
    toast.setDuration(Toast.LENGTH_LONG);
    toast.setView(getLayoutInflater().inflate(R.layout.muj_toast,null));
    toast.show();
  }
}
```

Příklad – poslání notifikace

Nevýhodou toastu je, že se na definovanou dobu zobrazí a potom zmizí. Pokud se uživatel právě na displej nedívá, notifikaci neuvidí. V následujícím příkladu pošle aplikace notifikaci, která se zobrazí v oznamovací oblasti, kde si ji uživatel může kdykoliv zobrazit.

Pomocí panelu oznámení mohou aplikace i Android samotný informovat uživatele o různých událostech. Panel oznámení je možné gestem tažení prstu směrem dolů vysunout a zobrazit další informace, které byly umístěny v oznamovací oblasti.



Upozornění: Z procesu běžícího na pozadí, bez ohledu na to, zda se jedná o službu nebo Broadcast Receiver, nespouštějte aktivity přímo. Určitě byste se uživateli nezavděčili, spíš byste jej rozzlobili, jelikož nevíte, co právě dělá nebo co má v úmyslu. Namísto toho vytvořte notifikaci, která mu poskytne požadovanou informaci, a uživatel se sám rozhodne, zda spustí nějakou aplikaci.

Aby bylo možné přehrání zvuku, vytvořte ve složce *res* složku *raw* a do ní nakopírujte soubor ve formátu MP3 se zvukem, který má zaznít při upozornění. Soubor se zvukem snadno získáte z Internetu.

Layout hlavní aktivity obsahuje jedno tlačítko, podobně jako v předchozí aplikaci. Liší se jen textem, jelikož neposíláme toast, ale notifikaci.

```
<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/btNotifikace"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="Notifikace"/>
</LinearLayout>
```

Abychom demonstrovali co nejvíce funkcionality týkající se oznámení, vytvoříme další aktivitu. Tato se spustí, když uživatel klepne na oznámení v panelu oznámení. Aktivita v tomto příkladu vypíše konstantní textové oznámení. V reálné aplikaci by vypsala podrobnosti o události, která notifikaci vyvolala.

Layout v souboru *activity_notif.xml*:

```
<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="wrap_content"
```

```
    android:text="Aktivita spuštěna z oznámení"
    android:textSize="24sp">
  </TextView>
</LinearLayout>
```

Soubor *NotifActivity.java*:

```
public class NotifActivity extends Activity
{
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_notif);
  }
}
```

Nezapomeňte tuto aktivitu přidat do seznamu aktivit v souboru *AndroidManifest.xml*.

```
<activity android:name=".NotifActivity" />
```

V kódu hlavní aktivity definujeme texty nadpisu a textu oznámení a také zvuk a vzor vibrací, které uživatele na oznámení upozorní. V metodě `onCreate` vytvoříme záměr, který se odešle po klepnutí na oznámení v panelu oznámení. Záměr spustí aktivitu `NotifActivity`.

Ke správě notifikací je určena systémová služba `NotificationManager`. Notifikace bude definovaná prostřednictvím objektu `Notification.Builder`.

```
public class MainActivity extends Activity
{
  private static final int NOTIF_ID = 1;
  private int pocetNotifikaci;

  private Intent notifIntent;
  private PendingIntent pendingIntent;

  private final CharSequence ticker = "Lorem ipsum dolor sit amet...!";
  private final CharSequence nadpis = "Notifikace";
  private final CharSequence oznam = "Máš notifikaci!";

  private Uri soundURI =
    Uri.parse("android.resource://com.example.notifikace1/" + R.raw.alarm);
  private long[] vibrace = { 0, 300, 200, 300 };

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

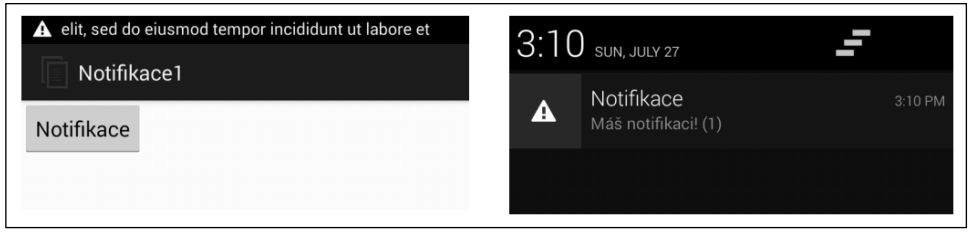
    notifIntent = new Intent(getApplicationContext(), NotifActivity.class);
    pendingIntent = PendingIntent.getActivity(getApplicationContext(), 0,
      notifIntent, Intent.FLAG_ACTIVITY_NEW_TASK);
  }
```

```
  public void onClick(View View)
  {
    Notification.Builder notificationBuilder = new Notification.Builder(
      getApplicationContext())
      .setTicker(ticker)
      .setSmallIcon(android.R.drawable.stat_sys_warning)
      .setAutoCancel(true)
      .setContentTitle(nadpis)
      .setContentText(oznam + " (" + ++pocetNotifikaci + ")")
      .setContentIntent(pendingIntent).setSound(soundURI)
      .setVibrate(vibrace);

    NotificationManager mNotificationManager = (NotificationManager)
      getSystemService(Context.NOTIFICATION_SERVICE);
    mNotificationManager.notify(NOTIF_ID,
      notificationBuilder.build());
  }
}
```

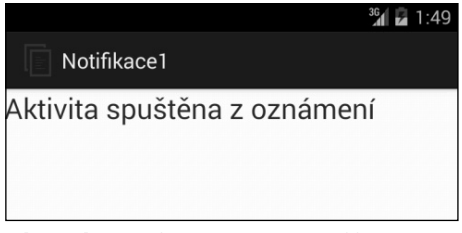
Po stisknutí tlačítka aplikace vyšle systému notifikaci. Ta se projeví zvukem, vibrací, zobrazením ikony v oznamovací oblasti a postupným rolováním textu. Úmyslně jsme zadali dlouhý text, aby se rolování projevilo. Po odrolování textu zůstane zobrazená pouze ikona notifikace.

Po zobrazení panelu oznámení se zobrazí oznámení vaší aplikace. To sestává z ikony, názvu, textu oznámení a data. Oznámení také obsahuje číslo v závorkách, které udává, kolikrát bylo oznámení přijato. Zkuste poslat oznámení z aplikace znovu a toto číslo v závorkách se bude inkrementovat.



Obrázek 6.2: Test aplikace posílající notifikaci

Pokud uživatel klepne na oznámení v okně notifikačního centra, spustí se druhá aktivita aplikace, která vypíše textové oznámení.



Obrázek 6.3: Aktivita spuštěná po klepnutí na notifikaci

Alarmy

V dosud probíraných příkladech aplikace realizovala činnost, případně sekvenci činností, ihned po spuštění. Často se však vyskytuje požadavek, aby aplikace dělala něco v přesně stanovený čas, například aby za 15 minut uživatele na něco upozornila, případně aby něco dělala periodicky. K tomuto účelu slouží alarmy.



Poznámka: Alarm na platformě Android je mechanizmus umožňující poslání objektu `intent` do budoucnosti, do přesně definovaného časového okamžiku nebo do více časových okamžiků.

Úlohou aplikace je vytvořit a nastavit alarmy. K tomuto účelu je k dispozici rozhraní `Alarm Manager`, umožňující nastavení a zrušení alarmů. Po vytvoření a nastavení alarmů a ukončení aplikace, která to realizovala, může zařízení přejít do režimu spánku a aktivuje se až v definovaný časový okamžik.



Upozornění: Alarmy nastavené aplikacemi budou zrušeny po úplném vypnutí, případně restartování zařízení.

Vytvoření alarmu

Pomocí metody

```
getSystemService(Context.ALARM_SERVICE)
```

musí aplikace nejprve získat referenci na `AlarmManager`.

Jednorázový alarm nastavíte pomocí metody

```
void set(int typ, long casAktivace, PendingIntent operace)
```

K nastavení alarmu opakujícího se v periodických časových intervalech použijete metodu

```
void setRepeating(int typ, long casAktivace, long interval,
  PendingIntent operace)
```

Nebo můžete použít metodu

```
void setInexactRepeating(int typ, long casAktivace, long interval,
  PendingIntent operace)
```



Poznámka: Počínaje verzí API 19 (KitKat) jsou všechny alarmy aktivované jako **`inexact`**, což ve verzích API <=18 zabrání opakovanému buzení zařízení při více těsně za sebou jsoucích alarmech. Jak vyplývá z názvu **`inexact`** (nepřesný), alarm se nemusí aktivovat v přesně nastaveném čase, ale například zařízení se probudí a alarm aktivuje současně s jiným alarmem nastaveným na vhodnou dobu. Třeba když je dvojice alarmů **`inexact`** naplánovaná s 15sekundovým zpožděním, systém je aktivuje současně.

Takovéto nastavení se dá použít tam, kde na přesném čase nezáleží, například u připomínky, že je potřeba nakrmit rybičky v akváriu a podobně.

K definování intervalu je v metodě `setInexactRepeating` potřeba použít předdefinované konstanty:

- INTERVAL_FIFTEEN_MINUTES
- INTERVAL_HALF_HOUR
- INTERVAL_HOUR
- INTERVAL_HALF_DAY
- INTERVAL_DAY

Parametr typ určuje, jakým způsobem se interpretuje parametr casAktivace: buď jako reálný čas, kdy hodnota parametru udává počet milisekund od půlnoci 1. ledna 1970 (RTC_...), nebo jako relativní čas od posledního spuštění systému (ELAPSED_...). Tento parametr zároveň určuje chování systému v případě, že nastal čas alarmu. Jednou z možností je probudit zařízení v okamžiku alarmu a doručit záměr (RTC_WAKEUP), druhou pokračovat ve stavu sleep a záměr doručit při nejbližším probuzení zařízení (RTC).

Příklad – ověření funkcionality alarmů

Na praktické ověření jednotlivých typů alarmů vytvoříme cvičný příklad. Jeho uživatelské rozhraní tvoří čtyři tlačítka. První tři aktivují alarmy a třetí slouží ke zrušení opakování alarmu.

```
<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:id="@+id/bt_single"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/nastav_single" >
  </Button>

  <Button
    android:id="@+id/bt_repeating"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/nastav_repeat" />

  <Button
    android:id="@+id/bt_inexact"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/nastav_inexact" />

  <Button
    android:id="@+id/bt_zrus"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/zrus_repeat" />
</LinearLayout>
```



Obrázek 6.4: Uživatelské rozhraní aplikace

Pro úplnost uvádíme i texty definované v souboru *strings.xml*.

```
<resources>
  <string name="app_name">Alarm Demo</string>
  <string name="text_upozorneni">Nakrm rybičky!</string>
  <string name="nastav_single">Nastav Single Alarm</string>
  <string name="nastav_repeat">Nastav Repeating Alarm</string>
  <string name="nastav_inexact">Nastav Inexact Repeating Alarm</string>
  <string name="zrus_repeat">Zruš Repeating Alarm</string>
</resources>
```

Dříve než přistoupíte k programování kódu hlavní aktivity, vytvořte třídu odvozenou od `BroadcastReceiver`, která bude přijímat záměry a zobrazovat notifikace. Abychom demonstrovali i přehrání zvuku, vytvořte ve složce *res* složku *raw* a do ní nakopírujte soubory ve formátu MP3 se zvukem, který má zaznít při alarmu. Soubor se zvukem alarmu snadno získáte z Internetu. Alarm se projeví i jako sekvence vibrací.

```
public class NotifReceiver extends BroadcastReceiver
{
  private static final int NOTIF_ID = 1;
  private static final String TAG = "NotifReceiver";

  private Intent notifIntent;
  private PendingIntent pendIntent;

  //zvuk a vibrace
  private Uri soundURI
    = Uri.parse("android.resource://com.example.alarmdemo/" + R.raw.alarm);
  private long[] mVibratePattern = { 0, 300, 100, 300 };

  @Override
  public void onReceive(Context context, Intent intent)
  {

    notifIntent = new Intent(context, MainActivity.class);
    pendIntent = PendingIntent.getActivity(context, 0,
      notifIntent, Intent.FLAG_ACTIVITY_NEW_TASK);

    Notification.Builder notificationBuilder = new
      Notification.Builder(context).setTicker("Upozornění!")
      .setSmallIcon(android.R.drawable.stat_sys_warning)
      .setAutoCancel(true).setContentTitle("Nezapomeň")
      .setContentText("nakrmit rybicky!!").setContentIntent(pendIntent)
      .setSound(soundURI).setVibrate(mVibratePattern);

    NotificationManager notifManag = (NotificationManager)
      context.getSystemService(Context.NOTIFICATION_SERVICE);
    notifManag.notify(NOTIF_ID, notificationBuilder.build());

    Log.i(TAG,"Notifikace:" +
      DateFormat.getDateTimeInstance().format(new Date()));

  }
}
```

V kódu hlavní aktivity je inicializace objektu `AlarmManager`, vytvoření záměru pro notifikační `BroadcastReceiver` a v obsluze událostí klepnutí na tlačítka nastavení jednotlivých typů alarmů. V reálné aplikaci byste nastavení alarmu oznámili uživateli, například přes toast.

```
public class MainActivity extends Activity
{
  private AlarmManager alarmManag;
  private Intent notifIntent;
  private PendingIntent notifPendingIntent;
  private static final long SPOZDENI = 30000L;

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

    alarmManag = (AlarmManager) getSystemService(ALARM_SERVICE);
    notifIntent = new Intent(MainActivity.this, NotifReceiver.class);
    notifPendingIntent = PendingIntent.getBroadcast(MainActivity.this, 0,
      notifIntent, 0);

    final Button btSingle = (Button) findViewById(R.id.bt_single);
    btSingle.setOnClickListener(new OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        alarmManag.set(AlarmManager.RTC_WAKEUP,
          System.currentTimeMillis() + SPOZDENI,
          notifPendingIntent);
      }
    });

    final Button btRepeating = (Button) findViewById(R.id.bt_repeating);
    btRepeating.setOnClickListener(new OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
        alarmManag.setRepeating(AlarmManager.ELAPSED_REALTIME,
          SystemClock.elapsedRealtime() + SPOZDENI,
          AlarmManager.INTERVAL_FIFTEEN_MINUTES,
          notifPendingIntent);
      }
    });

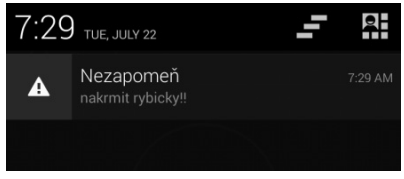
    final Button btInexact = (Button) findViewById(R.id.bt_inexact);
    btInexact.setOnClickListener(new OnClickListener()
    {
      @Override
      public void onClick(View v)
```

```
      {
        alarmManag.setInexactRepeating(
          AlarmManager.ELAPSED_REALTIME,
          SystemClock.elapsedRealtime() + SPOZDENI,
          AlarmManager.INTERVAL_FIFTEEN_MINUTES,
          notifPendingIntent);
      }
    });

    final Button cancelRepeatingAlarmButton = (Button)
      findViewById(R.id.bt_zrus);
    btSingle.setOnClickListener(new OnClickListener()
    {
      @Override
      public void onClick(View v)
      {
         alarmManag.cancel(notifPendingIntent);
      }
    });
  }
}
```

Nezapomeňte do souboru *AndroidManifest.xml* přidat `receiver`:

```
<receiver android:name=".NotifReceiver">
</receiver>
```



Obrázek 6.5: Notifikace

KAPITOLA 7

Seznamy objektů

V této kapitole:
- Námět příkladů
- Řešení s využitím ListActivity
- ListActivity s formátovaným zobrazováním položek
- ListActivity s kontextovým menu položky
- Řešení s využitím ListView a statickými údaji v poli řetězců
- Řešení s využitím ListView a údaji v kódu
- Výběr více položek ze seznamu
- Zobrazení hierarchické struktury
- Seznam objektů s obrázky

Zamysleli jste se někdy nad tím, jak využíváte aplikace na svém chytrém telefonu, a to bez ohledu na platformu? Vytáhnete telefon z kapsy, spustíte aplikaci a zjistíte informaci, kterou potřebujete. Druh informace, kterou uživatel potřebuje, si zvolí tím, že vybere aplikaci, která příslušný druh informace dokáže poskytnout. Dalším krokem je zpravidla výběr požadované informace nebo objektu ze zobrazeného seznamu. Proto věnujeme zobrazování seznamů celou kapitolu.

Námět příkladů

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.

Řešení s využitím 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.

Vytvořte aplikaci a v kódu hlavní aktivity 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);

  }
}
```

V metodě `onCreate()` z kódu vygenerovaného průvodcem musíte vymazat řádek, který propojuje kód aktivity s XML layoutem.

```
setContentView(R.layout.activity_main);
```

Aktivita odvozená od `ListActivity` totiž nepotřebuje soubor s layoutem, jelikož disponuje layoutem zabudovaným.



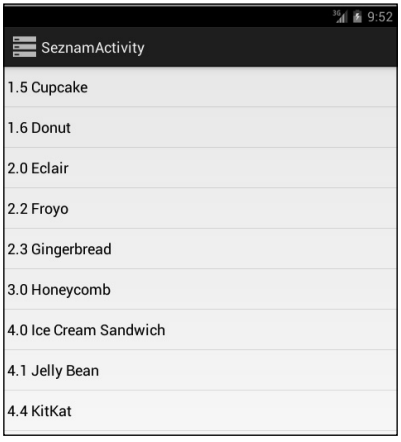
Upozornění: Pokud byste propojení na XML soubor layoutu neodstranili, došlo by k chybě.

Soubor *activity_main.xml* nepotřebujete, a můžete ho proto 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");
    ArrayAdapter<String> listAdapter =
      new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
      verze );
    setListAdapter(listAdapter);
  }
}
```



Obrázek 7.1: Zobrazení seznamu objektů pomocí ListActivity

Určitě budete chtít implementovat obsluhu události klepnutí na položku. Upravte kód metody `onCreate()` takto:

```
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.setOnItemClickListener(new OnItemClickListener()
  {
    public void onItemClick(AdapterView<?> parent, View view,
      int position, long id)
    {
      Toast.makeText(getApplicationContext(), ((TextView) view).getText(),
        Toast.LENGTH_SHORT).show();
    }
  });
}
```

Kromě klepnutí je možné implementovat i podržení prstu na dotykovém displeji v místě, kde je zobrazená některá z položek. Pokud metoda `boolean onItemLongClick()` vrátí hodnotu `true`, klepnutí a následné podržení položky je považováno za zpracované a systém už nevolá listener `nListItemClick()`.

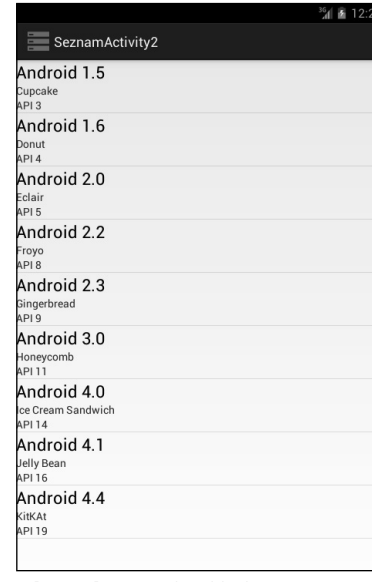
```
protected void onCreate(Bundle savedInstanceState)
{
  super.onCreate(savedInstanceState);
  List<String> verze = Arrays.asList("1.5 Cupcake", "1.6 Donut",...);
```

```
  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 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;
    }
  });
}
```

ListActivity s formátovaným zobrazováním položek

V předchozím řešení jsme deklarovali, že aktivita odvozená od `ListActivity` nepotřebuje návrh layoutu v XML souboru. To platí pro zapouzdření aktivity jako celku. Layout se však často používá 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 7.2: 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`. V metodě `onCreate()` musíte vymazat řádek s metodou `setContentView(R.layout.activity_main)`. Vymažte soubor *activity_main.xml* s návrhem layoutu hlavní aktivity vygenerovaný vývojovým prostředím při vytváření projektu aplikace.

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/verze"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    style="@android:style/TextAppearance.Large"/>

  <TextView android:id="@+id/nazev"
    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 VerzeAndroidu
{
  private String verze;
  private String verze;
  private String api;

  public VerzeAndroidu(String verze, String nazev, String api)
  {
    this.verze = verze;
    this.nazev = nazev;
    this.api = api;
  }

  public String getVerze()
  {
    return verze;
  }

  public String getNazev()
  {
    return nazev;
  }

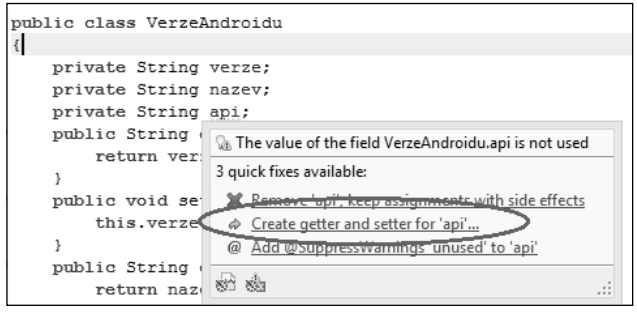
  public String getApi()
  {
```

```
    return api;
  }
}
```



Tip: K vytvoření metod na přístup k atributům třídy stačí podržet chvíli kurzor na názvu atributu a ze zobrazené nabídky vybrat položku **Create getter and setter for**.

Jelikož se hodnoty nastavují pouze v konstruktoru, v tomto příkladu potřebujete jen metody `getNazevAtributu`.



Obrázek 7.3: Vytvoření metod „getter" a „setter" na přístup k atributům třídy

Aplikační kód

Datový model (tedy údaje, které se zobrazí jako položky) je spravovaný v takzvaném adaptéru, což je třída odvozená od třídy `BaseAdapter`.



Poznámka: Na úrovni adaptéru se realizují i případné operace s údaji, například řazení 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 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("verze", ver.getVerze());
polozkyMap.put("nazev", ver.getNazev());
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 = {"verze", "nazev","api"};
int[] idAtributu = { R.id.verze, R.id.nazev, 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<VerzeAndroidu> verze = Arrays.asList(
      new VerzeAndroidu("Android 1.5", "Cupcake", "API 3"),
      new VerzeAndroidu("Android 1.6", "Donut", "API 4"),
      new VerzeAndroidu("Android 2.0", "Eclair", "API 5"),
      ...
      setListAdapter(getListAdapter(verze));
  }

  private ListAdapter getListAdapter(List<VerzeAndroidu> verze)
  {
    String[] nazvyAtributu = {"verze", "nazev","api"};
    int[] idAtributu = { R.id.verze, R.id.nazev, R.id.api};
    SimpleAdapter adapter = new SimpleAdapter
      (this, getListAdapterData(verze),
      R.layout.list_items, nazvyAtributu, idAtributu);
    return adapter;
  }

  private List<Map<String, ?>>
    getListAdapterData(List<VerzeAndroidu> verze)
  {
    List<Map<String, ?>>
      list = new ArrayList<Map<String, ?>>(verze.size());
    for (VerzeAndroidu ver : verze)
    {
      Map<String, String> polozkyMap = new HashMap<String, String>();
      polozkyMap.put("verze", ver.getVerze());
      polozkyMap.put("nazev", ver.getNazev());
      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 dalšími prvky.

ListActivity s kontextovým menu položky

V prvním příkladu jsme ukázali jednoduchou obsluhu události „klepnutí a podržení". 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.

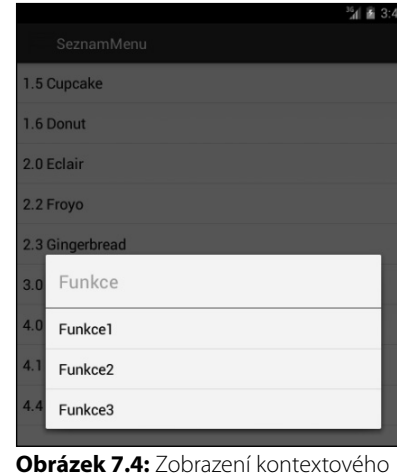
Metoda `onCreateContextMenu()` na vytvoření položek menu je společná pro všechny prvky, které budou zobrazovat menu, proto je v reálné aplikaci potřeba zkontrolovat, zda se jedná o menu pro předmětný prvek. Provedeme to pro ilustraci i ve cvičném příkladu, kde je takovýto prvek pouze jeden. Pomocí metody `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 7.4: Zobrazení kontextového menu položky

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", ...);
    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,
    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;
      ...
      default:
        return super.onContextItemSelected(item);
    }
  }
}
```

Řešení s využitím ListView a statickými údaji v poli řetězců

V první 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 name="app_name">Seznam1</string>
  <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 2.1 Eclair</item>
    <item>Android 2.2 Froyo</item>
    <item>Android 2.3 Gingerbread</item>
    <item>Android 1.5 Cupcake</item>
    <item>Android 3.1 Honeycomb</item>
    <item>Android 3.2 Honeycomb</item>
    <item>Android 4.0 Ice Cream Sandwich</item>
    <item>Android 4.1 Jelly Bean</item>
    <item>Android 4.2 Jelly Bean</item>
    <item>Android 4.3 Jelly Bean</item>
    <item>Android 4.4 KitKat</item>
    <item>Android 5.0 Lollipop</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="${packageName}.${activityClass}" >

  <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 7.5: Test fungování aplikace. V dolní části je toastový výpis, který se zobrazí po klepnutí na položku.

Řešení s využitím ListView a údaji v kódu

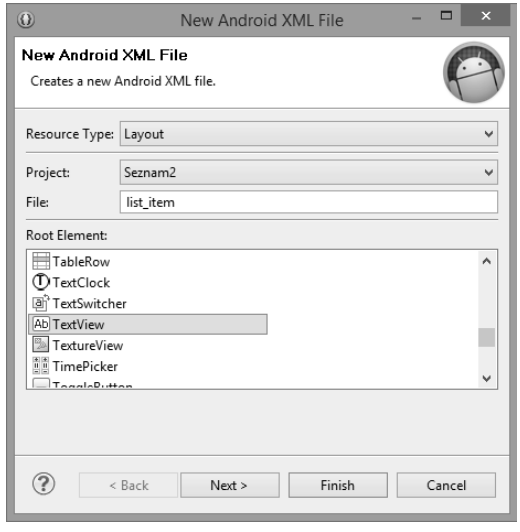
V předchozí aplikaci jsme napojili prvek `ListView` na zdroj údajů ve zdrojích přímo v XML kódu návrhu layoutu přes atribut `android:entries`. V tomto příkladu naplníme prvek `ListView` v kódu aktivity. Výhody jsou zřejmé – údaje je možné různým způsobem předzpracovat, prvek naplnit z různých zdrojů údajů, například z databázové tabulky SQLite, z JSON, XML... Abyste měli možnost porovnání, 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="${packageName}.${activityClass}" >

  <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`. Nový layout nejjednodušeji vytvoříte, pokud na složku layout klepnete pravým tlačítkem myši a z místní nabídky zvolíte **New → Android XML File**. V dialogovém okně vyberte **TextView** jako typ kořenového elementu.

Obrázek 7.6: Dialogové okno na vytvoření nového layoutu

V kódu změňte hodnoty atributů `layout_width` a `layout_height` na hodnotu `fill_parent` a pro lepší úpravu nastavte hodnotu atributu `padding`.

```
<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 je 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 je možno 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 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);

    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 hodnoty definovat přímo v kódu.

```
static final String[] verze =
{
  "Android 1.5 Cupcake",
  "Android 1.6 Donut",
  ...
};
```

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">Výsledek 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 `RelativeLayout`. 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="${packageName}.${activityClass}" >

  <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. Postup vytvoření layoutu je popsán v předchozím příkladu.

```
<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>
```

Aplikační kód

V kódu hlavní aktivity načítáme hodnoty z pole řetězců a odevzdáme je 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 Activity implements 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 adaptéru
      int position = checked.keyAt(i);
      // přidej označené položky
      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(),
      VysledekActivity.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 VysledekActivity extends Activity
{
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.vysledek);

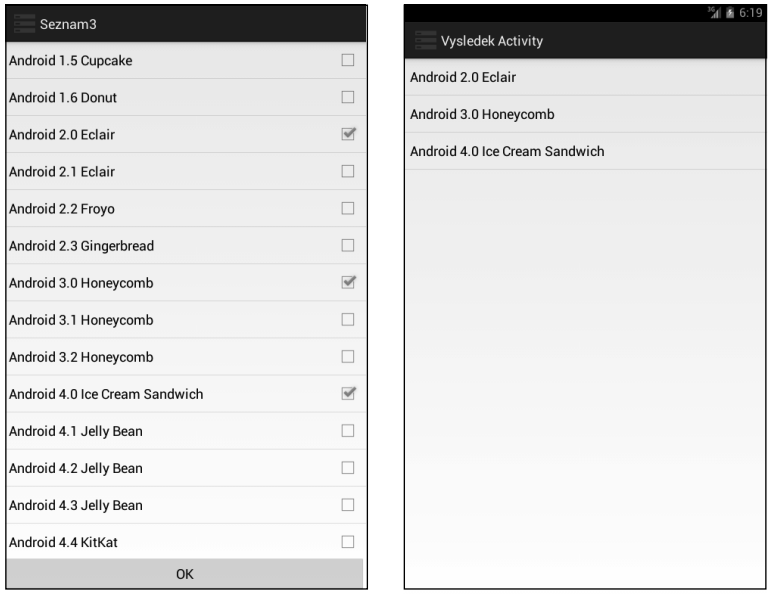
    Bundle b = getIntent().getExtras();
    String[] resultArr = b.getStringArray("selectedItems");
    ListView lv = (ListView) findViewById(R.id.vysledekList);

    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*.

```
<activity
  android:name=".VysledekActivity"
  android:label="@string/vysledek_activity" >
</activity>
```

Nyní můžete aplikaci spustit a otestovat její fungování.



Obrázek 7.7: Test aplikace se dvěma aktivitami

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, zobrazí se po rozvinutí kategorie 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 této úrovni hierarchie budou zobrazeny položky:

- Verze 1.x
- Verze 2.x

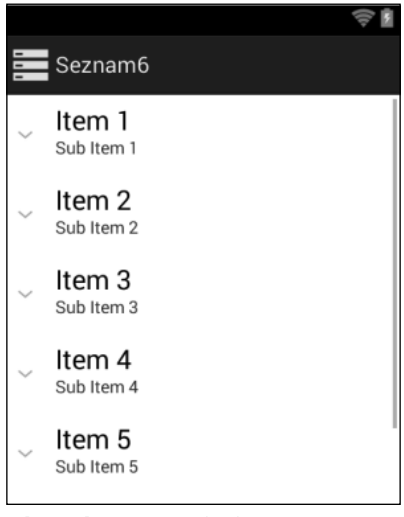
- Verze 3.x
- Verze 4.x

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="${packageName}.${activityClass}" >

  <ExpandableListView
    android:id="@+id/verze_list"
    android:layout_width="match_parent"
    android:layout_height="fill_parent" >
  </ExpandableListView>
</RelativeLayout>
```



Obrázek 7.8: V návrhu layoutu si všimněte levé části šipky směřující dolů na rozbalení skupin položek

Na rozdíl od `ListView` pro rozbalitelný seznam položek je potřeba navrhnout oba dva layouty: layout pro skupinu položek a layout pro vnořenou položku.

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 Eclipse, 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.

```
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 OnClickListener()
    {
      public void onClick(View v)
      {
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setMessage("Skutečně 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 Activity
{
  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 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.x");
    skupinyList.add("Verze 2.x");
    skupinyList.add("Verze 3.x");
    skupinyList.add("Verze 4.x");
  }

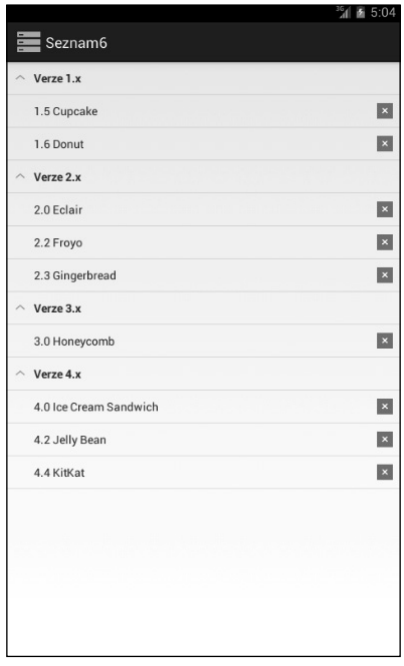
  private void vytvorPolozky()
  {
    String[] v1 = { "1.5 Cupcake","1.6 Donut"};
    String[] v2 = { "2.0 Eclair", "2.2 Froyo", "2.3 Gingerbread " };
    String[] v3 = { "3.0 Honeycomb"};
    String[] v4 = { "4.0 Ice Cream Sandwich", "4.2 Jelly Bean",
      "4.4 KitKat"};

    skupinyOS = new LinkedHashMap<String, List<String>>();

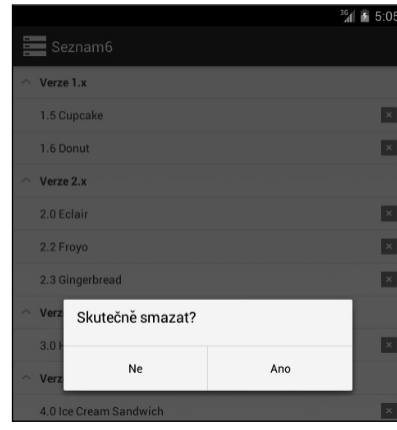
    for (String laptop : skupinyList)
    {
      if (laptop.equals("Verze 1.x"))
        naplnPolozky(v1);
      else if (laptop.equals("Verze 2.x"))
        naplnPolozky(v2);
      else if (laptop.equals("Verze 3.x"))
        naplnPolozky(v3);
      else if (laptop.equals("Verze 4.x"))
        naplnPolozky(v4);

      skupinyOS.put(laptop, polozkyList);
    }
  }

  private void naplnPolozky(String[] laptopModels)
  {
    polozkyList = new ArrayList<String>();
    for (String model : laptopModels)
      polozkyList.add(model);
  }
}
```



Obrázek 7.9: Zobrazení rozvinuté hierarchické struktury



Obrázek 7.10: Dialog na potvrzení výmazu položky

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 objevují i obrázky. Text které obsahuje více řádků, například nadpis a podnadpis, které jsou vypsány různým fontem. Také naše cvičné údaje o verzích Androidu si o takovýto způsob přímo říkají – ke každému kódovému označení s názvem zákusku přece lze 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="${packageName}.${activityClass}" >

  <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 i 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, které 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, například *drawable-hdpi*. V kódu se budeme odkazovat jen na název obrázku bez přípony. Název obrázku musí začínat písmenem, například *R.drawable.o15*.

Kompletní kód v souboru *MainActivity.java*. Pro úsporu místa je seznam položek zkrácen.

```
public class MainActivity extends Activity implements OnItemClickListener
{
  public static final String[] verze = new String[]
  {
    "Android 1.5",
    "Android 1.6",
    ...
  };

  public static final String[] api = new String[]
  {
    "Cupcake, API 3",
    "Donut, API 4",
    ...
  };

  public static final Integer[] images =
  {
    R.drawable.o15,
    R.drawable.o16,
    ...
  };

  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);
    }

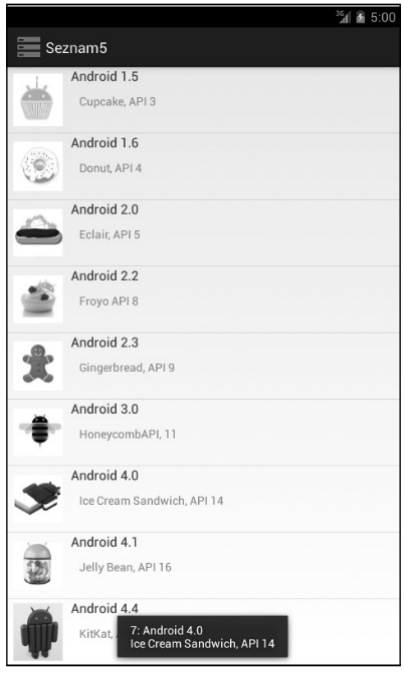
    listView = (ListView) findViewById(R.id.list);
```

```
    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 7.11: Test fungování aplikace na zobrazení seznamu s obrázky a strukturovaným textem

KAPITOLA 8

Fragmenty

V této kapitole:
- Námět příkladu
- Fragmenty

S příchodem tabletů začalo pro vývojáře dilema. Pokud byste aplikaci určenou pro telefony spustili na velkém displeji tabletu, většina obrazovky by nebyla využita. Opačně, pokud byste aplikaci navrhli primárně pro tablet, na menších displejích telefonů by uživatelské rozhraní aplikace nebylo čitelné nebo by se vůbec nedalo zobrazit. Jedním z řešení – vedle vektorové grafiky a relativních layoutů pro rozmístění ovládacích prvků – jsou fragmenty.



Poznámka: 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.



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

Existují dva hlavní způsoby, jako mohou být fragmenty přidány do aktivity:

- staticky – fragment je vložen do layoutu aktivity
- programově – pomocí objektu `FragmentManager`

Námět příkladu

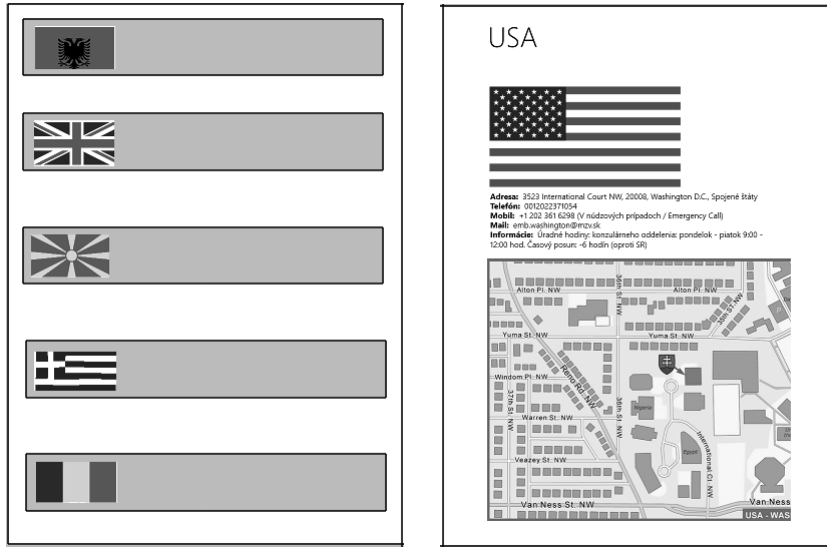
Potřebujeme vytvořit aplikaci se seznamem zastupitelství v zahraničí, tedy velvyslanectví a generálních konzulátů. Takováto aplikace by mohla být vynikající pomůckou pro turisty, kteří v zahraničí ztratí cestovní doklady nebo se ocitnou v nouzi a potřebují pomoc českého nebo slovenského zastupitelství. Údaje o zastupitelstvích jsou součástí aplikace (adresy velvyslanectví se často nemění), takže aplikace nepotřebuje připojení k Internetu, jelikož v některých zemích

jsou roamingové poplatky za přenos dat vysoké. Aplikace obsahuje podrobnosti o každém zastupitelském úřadu, včetně telefonického a e-mailového kontaktu, nouzové mobilní číslo, na které lze volat mimo úřední hodiny, adresy s mapou a otevírací hodiny.

Rozbor řešení

V tomto příkladu nebudeme prioritně řešit způsob přibalení údajů, ale zaměříme se na jejich vizuální reprezentaci, která bude rozdílná na telefonech a na tabletech.

Na telefonu s relativně malým displejem se na celé ploše displeje zobrazí jako hlavní aktivita aplikace seznam objektů, v tomto případě zemí. Po klepnutí na některý objekt se na ploše obrazovky zobrazí jiná aktivita s detailními informacemi o vybraném objektu.



Obrázek 8.1: Zobrazení seznamu objektů (vlevo) a po výběru objektu zobrazení podrobnosti na nové obrazovce (vpravo)

Aplikace by v takovéto podobě fungovala i na tabletu, plocha v pravé části displeje by však zůstala nevyužitá.

Od verze Android 3.0 Honeycomb je možné při tvorbě uživatelského rozhraní s výhodou využívat fragmenty umožňující zapouzdřit uživatelské rozhraní `Activity` i s metodami tvořícími aplikační logiku.

Poznámka: Fragment je možné považovat za jakousi miniaktivitu, která se může stát součástí jiné aktivity.

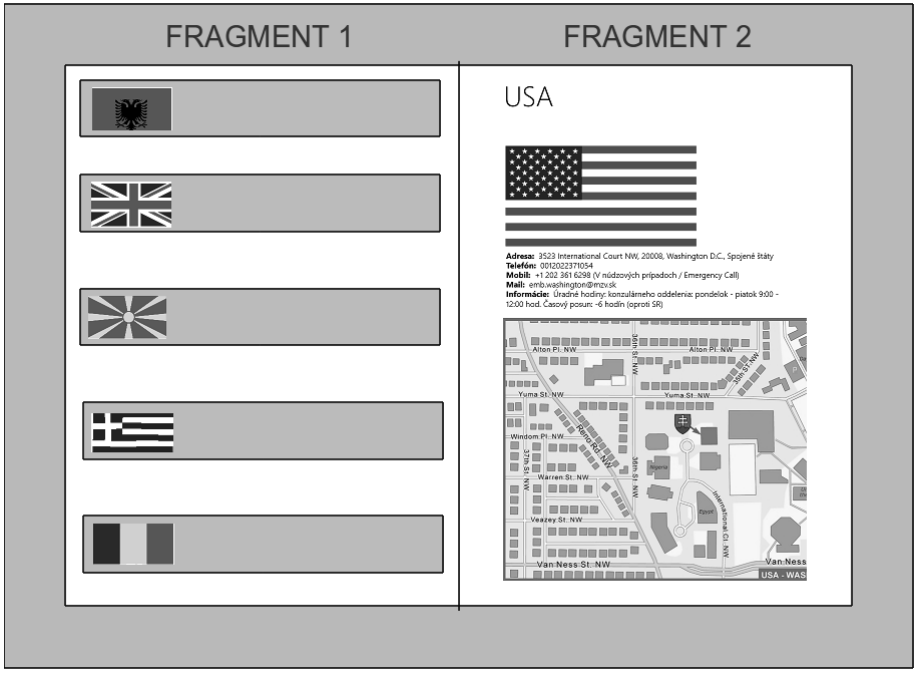
Jelikož se `Fragment` může stát součástí jiné aktivity nebo fragmentu, můžete škálovat uživatelské rozhraní pro různé velikosti displeje. V aplikaci zastupitelstva s využitím fragmentů se

na tabletu zobrazí jakoby dvě aktivity vedle sebe: vlevo seznam objektů a vpravo detailní informace o vybraném objektu.



Poznámka: Fragmenty lze díky podpůrné knihovně Android Compatibility Library (ACL) využívat i v aplikacích pro nižší verze Androidu.

Podpůrná knihovna, jejíž třídy se nacházejí v jmenném prostoru `android.support.v4.app`, neumožňuje vložení fragmentu do layoutu aktivity pomocí XML kódu.



Obrázek 8.2: Zobrazení seznamu objektů i detailů o vybraném objektu na velkém displeji tabletu

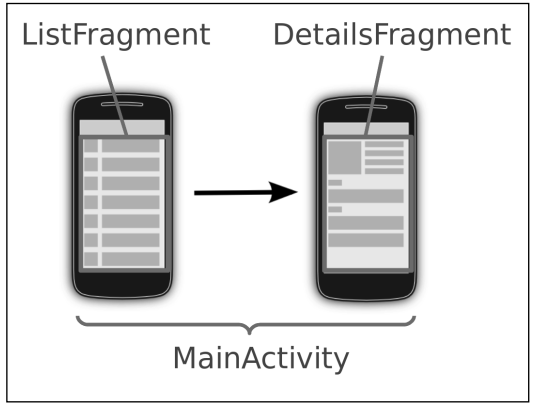
Fragmenty

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

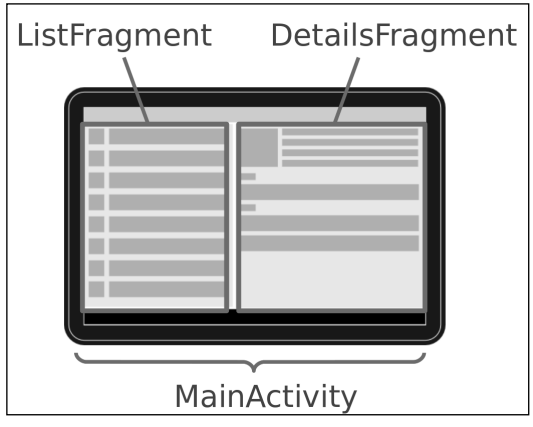
seznam objektů a po výběru některého z nich překryje obrazovku druhá aktivita s detailními informacemi o vybraném objektu.

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 8.3: Zobrazení fragmentů List a Details na zařízeních s malým displejem



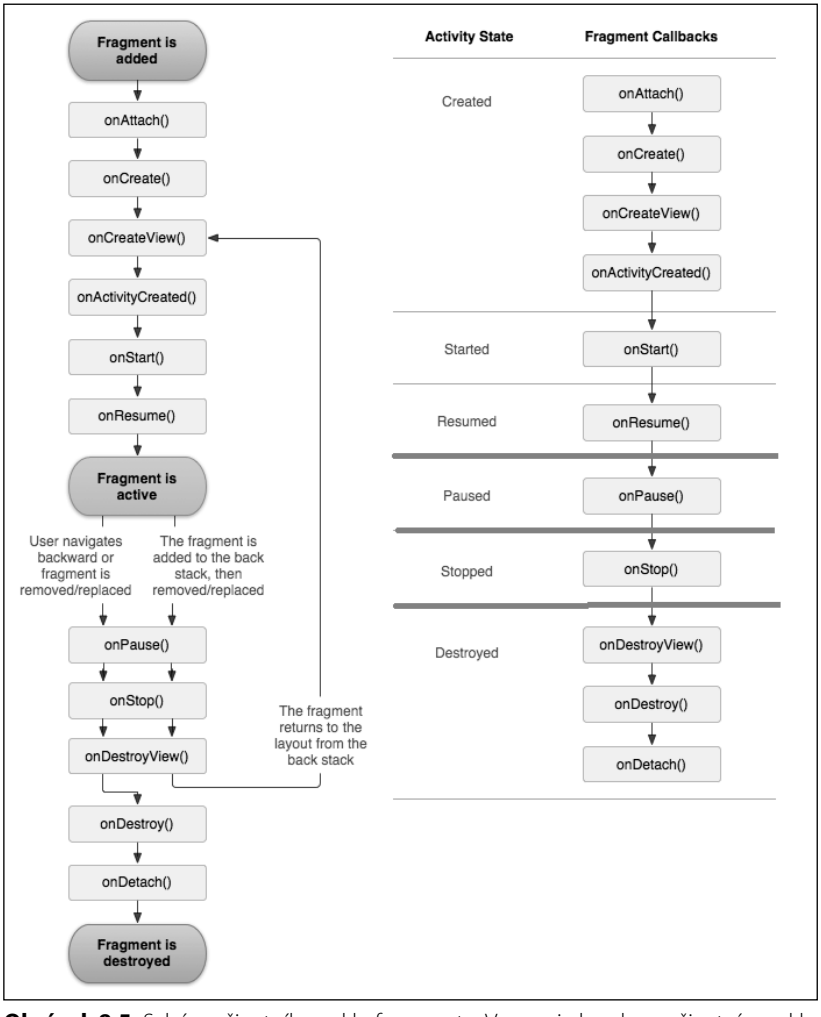
Obrázek 8.4: Zobrazení fragmentů List a Details na zařízeních s velkým displejem

Životní cyklus fragmentu

Z hlediska životního cyklu funguje fragment 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()`.

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 odevzdávaná 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.



Obrázek 8.5: Schéma životního cyklu fragmentu. Vpravo je korelace s životním cyklem aktivity.

Podobně jako aktivita může být i fragment ve třech základních stavech:

- **Resumed** (spuštěný) – fragment je zobrazen 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()`.

Vytvořte kompatibilní projekt

Vytvořte projekt aplikace pro Android. Abychom demonstrovali kompatibilitu fragmentů i na starších verzích, nastavte minimální verzi SDK na API 8 (Android 2.2 Froyo). Pokud byste jako minimální verzi SDK nastavili API 11, mohli byste vytvořit hlavní aktivitu typu master-detail. V tomto příkladu vytvořte hlavní aktivitu typu **Empty Activity**. Hlavní aktivitu vhodně pojmenujte, například `SeznamZastupitelstev`.

Abyste mohli využívat podpůrnou knihovnu `android.support.v4.app`, zpřístupňující fragmenty i pro přístroje s verzí Androidu nižší než 3.0, musíte mít nainstalovanou komponentu Android Support Library. Přes Android SDK Manager zkontrolujte, zda máte podpůrnou knihovnu nainstalovanou, a pokud ne, doinstalujte ji.

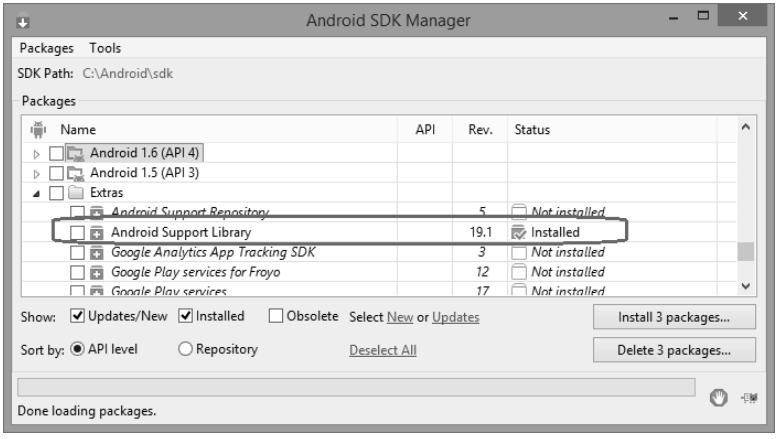
V projektu vytvořte nový adresář *libs*. Z adresáře, ve kterém máte nainstalované SDK (*<sdk>\extras\android\support\v4*), překopírujte soubor *android-support-v4.jar* do nově vytvořeného adresáře *libs*. V místní nabídce souboru *android-support-v4.jar* zvolte položku **Build Path → Add to Build Path**.



Poznámka: Pokud jste tyto kroky (**Build Path → Add to Build Path**) realizovali v některém z předchozích projektů, nově vytvořený projekt už bude mít vytvořený adresář *libs* se souborem *android-support-v4.jar*.

Uživatelské rozhraní aktivity na zobrazení detailů sestává kvůli jednoduchosti z jediného prvku typu `TextView` v kontejneru `LinearLayout`. Jelikož informace o objektu mohou přesáhnout rozměry obrazovky, návrh aktivity obsahuje i prvek `ScrollView`, umožňující svislé rolování.

V reálné aplikaci by se zobrazila kombinace textových informací (různým fontem a barvou) a obrázků.



Obrázek 8.6: Android Support Library v Android SDK Manager

Soubor *detail.xml*:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" >

  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

    <TextView
      android:id="@+id/detaily"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"/>
  </LinearLayout>

</ScrollView>
```

V souboru *nazov.xml* je definice zobrazení položek v seznamu.

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:padding="15dp"
  android:textAppearance="?android:attr/textAppearanceMedium"
  android:id="@+id/nazov">
</TextView>
```

Zobrazení detailních informací

Třída `DetailFragment` na zobrazení detailních informací o vybraném objektu je odvozena od třídy `Fragment`. Každá takto odvozená třída musí navíc přepisovat metodu `onCreateView()`.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
  Bundle savedInstanceState) {}
```

Parametr, v tomto případě index aktuálně vybrané položky, se na rozdíl od aktivit nepředává pomocí záměrů, ale přímo pomocí argumentů přes objekt typu `Bundle`.

```
public static DetailFragment newInstance(int index)
{
  DetailFragment f = new DetailFragment();
  Bundle argumenty = new Bundle();
  argumenty.putInt(VYBER, index);
  f.setArguments(argumenty);
  return f;
}
```

Kompletní zdrojový kód fragmentu zobrazujícího detailní informace o zastupitelství ve vybrané zemi je v souboru *DetailFragment.java*.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class DetailFragment extends Fragment
{
  public static final String VYBER = "vyber";

  protected static String[] informacie = new String[]
  {
    "Albánie.\n"
      + "Velvyslanectví České republiky v Tiraně\n"
      + "Rruga Skënderbej 10, Tirana, Albánie.\n "
      + "TEL: 0035542234004 ",
      "Info o Belgii",
      "Info o Bělorusku"
  };

  public static DetailFragment newInstance(int index)
  {
    DetailFragment f = new DetailFragment();
    Bundle argumenty = new Bundle();
    argumenty.putInt(VYBER, index);
    f.setArguments(argumenty);
    return f;
  }
```

```
  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState)
  {
    View vw = inflater.inflate(R.layout.detail, container, false);
    int index = getArguments().getInt(VYBER, 0);
    TextView tv = (TextView) vw.findViewById(R.id.detaily);
    tv.setText(informacie[index]);
    return vw;
  }
}
```

Zobrazení seznamu objektů

Kód fragmentu obsahuje přímo statické údaje, v tomto případě jen několik položek. Seznam se naplní v metodě `onActivityCreated` přes objekt typu `Context`. 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.

Kompletní zdrojový kód fragmentu se seznamem států je v souboru *ZoznamFragment.java*.

```
public class ZoznamFragment extends ListFragment
{
  protected static String[] staty = new String[] { "Albánie", "Belgie",
    "Bělorusko"};

  private SelectedListener selListener;

  @Override
  public void onActivityCreated(Bundle savedInstanceState)
  {
    super.onActivityCreated(savedInstanceState);
    Context ctx = getActivity();
    setListAdapter(new ArrayAdapter<String>(ctx, R.layout.nazov, staty));
  }

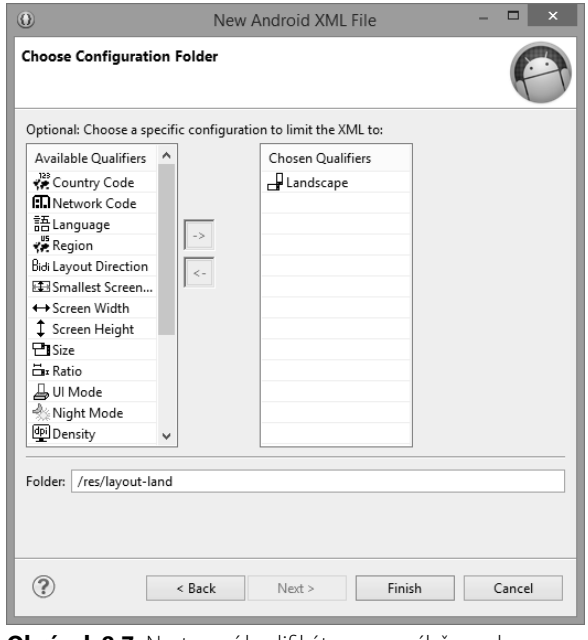
  @Override
  public void onAttach(Activity activity)
  {
    super.onAttach(activity);
    try
    {
      selListener = (SelectedListener) activity;
    } catch (ClassCastException e)
    {
      throw new ClassCastException(activity.toString()
        + " nie je implementovany listener");
    }
  }
```

```
  @Override
  public void onListItemClick(ListView l, View v, int pozice, long id)
  {
    selListener.onSelected(pozice);
  }

  public interface SelectedListener
  {
    public void onSelected(int vyber);
  }
}
```

Aktivity na zapouzdření fragmentů

Fragmenty musí být vloženy v aktivitách. Po vytvoření projektu se automaticky vytvoří hlavní aktivita. Fragment můžete do aktivity vložit buď pomocí layoutu, nebo dynamicky pomocí kódu. Tato metoda umožňuje odevzdat fragmentu údaje, například index vybrané položky, přímo při vytvoření v aktivitě.



Obrázek 8.7: Nastavení kvalifikátoru pro výběr souboru

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.

Hlavní aktivita aplikace bude obsahovat dva XML soubory s definicemi layoutu umístěné ve složkách */res/layout/* a */res/layout-land/*. Systém si podle orientace zařízení vybere vhodný soubor.

Vytvořte ve složce *res* vnořenou složku *layout-land* a v ní soubor typu Android XML s názvem *activity_zoznam_zastupitelstiev.xml*. Průvodce vytvořením souboru vám nabídne nastavení kvalifikátoru **Landscape** pro výběr složky. Nabídku akceptujte.

Kompletní definiční soubory layoutů hlavní aktivity budou v souboru */res/layout/activity_zoznam_zastupitelstiev.xml*:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent" android:layout_height="match_parent">

  <fragment class="com.example.slovenskezastupitelstva.ZoznamFragment"
    android:id="@+id/zoznam"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
</FrameLayout>
```

Soubor */res/layout-land/ activity_zoznam_zastupitelstiev.xml*:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal" >

  <fragment class="com.example.slovenskezastupitelstva.ZoznamFragment"
    android:id="@+id/zoznam"
    android:layout_weight="2"
    android:layout_width="0px"
    android:layout_height="match_parent" />

  <FrameLayout
    android:id="@+id/detail"
    android:layout_weight="3"
    android:layout_width="0px"
    android:layout_height="match_parent" />

</LinearLayout>
```

Nastavení atributů `layout_weight` definuje horizontální rozdělení plochy obrazovky mezi fragmenty v poměru 1 : 3, přičemž širší fragment je určen k zobrazení detailních informací.

Aktivita detail bude využívat nový `FrameLayout`. Vytvořte ve složce */res/layout/* nový soubor *detail_activity.xml*, přičemž v dialogovém okně při jeho vytvoření označte volbu **FrameLayout**. Aby bylo možné layout identifikovat, je potřeba mu doplnit identifikátor.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:id="@+id/detailAct">
</FrameLayout>
```

Při orientaci na šířku 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.

Soubor *DetailActivity.java*:

```
public class DetailActivity extends FragmentActivity {
  public static final String VYBER = "vyber";

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_detail);
    Intent i = getIntent();
    int id = i.getIntExtra(VYBER, 0);
    DetailFragment f = DetailFragment.newInstance(id);
    getSupportFragmentManager().beginTransaction().add(R.id.detailAct,
      f).commit();
  }
}
```

A nakonec uvedeme kompletní kód hlavní aktivity aplikace v souboru *ZoznamZastupitelstiev.java*. V této aktivitě se podle `id` layoutu hlavní aktivity zjišťuje, zda se použije jednosloupcový, nebo dvousloupcový layout.

```
import android.support.v4.app.FragmentActivity;
import android.support.v4.app.FragmentTransaction;

public class ZoznamZastupitelstiev extends FragmentActivity implements
  ZoznamFragment.SelectedListener
{

  private boolean dvaPanely;

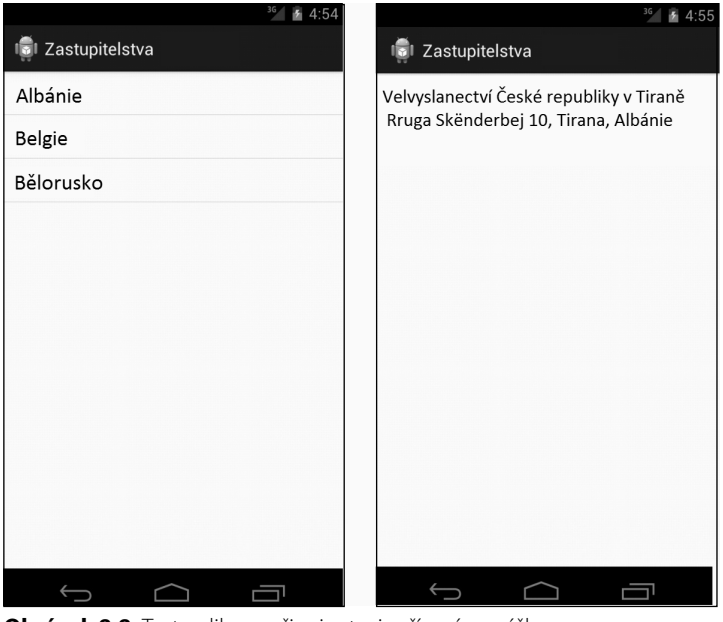
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_zoznam_zastupitelstiev);
    dvaPanely = findViewById(R.id.detail) != null;
  }

  @Override
  public void onSelected(int vyber)
  {
    //na šířku - dva fragmenty vedle sebe
    if (dvaPanely)
    {
      DetailFragment frg = DetailFragment.newInstance(vyber);

      FragmentTransaction frgt = getSupportFragmentManager()
        .beginTransaction();
      frgt.replace(R.id.detail, frg);
      frgt.addToBackStack(null);
      frgt.commit();
    } else
```

```
    { // na výšku - jeden fragment
      Intent i = new Intent(this, DetailActivity.class);
      i.putExtra(DetailActivity.VYBER, vyber);
      startActivity(i);
    }
  }
}
```

Pokud jste nezapomněli přidat `DetailActivity` do aplikačního manifestu v souboru *Android Manifest.xml*, můžete aplikaci otestovat. Otestujte zobrazení 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. Na změnu orientace emulátoru použijte klávesovou zkratku **Ctrl+F12**.



Obrázek 8.8: Test aplikace při orientaci zařízení na výšku



Obrázek 8.9: Test aplikace při orientaci zařízení na šířku

KAPITOLA 9

Broadcasty

V této kapitole:
- Broadcast Receiver
- Příklad – Broadcast Receiver registrovaný v aplikačním manifestu
- Příklad – dynamicky registrovaný Broadcast Beceiver
- Příklad – indikace příchozího hovoru

Broadcast Receiver

Záměry mohou sloužit i k posílání zpráv mezi komponentami. Většinou se jedná o reakce na určité změny v systému. Po odeslání broadcastu se pošle záměr všem Broadcast Receiverům zaregistrovaným na danou akci.

„Broadcast Receiver" je komponenta, díky které aplikace ví o jednotlivých systémových událostech. Mohli bychom ji nazvat i přijímač oznámení nebo možná výstižněji přijímač oběžníků. Oběžník může obsahovat informaci o přijetí SMS či MMS, o nízké hodnotě energie v baterii zařízení, o změně GPS souřadnic, o vložení SD karty a podobně.

Kromě systémových událostí je možné vytvářet události i samotnou aplikací a ty následně pomocí broadcastu odchytávat. Přestože broadcast nemá vlastní grafické uživatelské rozhraní, dokáže vytvářet notifikace, které uživatele přes Notification Manager upozorňují na různé události.



Poznámka: Podobně jako aktivity a služby se i Broadcast Receivery aktivují asynchronní zprávou „Intent".

Aplikace může mít libovolný počet přijímačů oznámení, které jsou pro danou aplikaci relevantní. Tyto přijímače se vytvářejí jako rozšíření základní třídy `BroadcastReceiver`.

Základní postup by se dal shrnout do čtyř bodů:

1. Registrace Broadcast Receiverů
2. Generování záměru a jeho odeslání do systému
3. Android doručí záměr zaregistrovaným přijímačům
4. Zpracování události v metodě `onReceive()`

Životní cyklus receiveru je vázaný na danou komponentu. Jinak skončí po přijetí zprávy a dokončení `onReceive()`.

Metoda `onReceive()` má dva parametry: `context` a `intent`. Tato metoda běží v hlavním vlákně, proto by operace v ní měly trvat jen velmi krátce, aby nebylo hlavní vlákno blokované. Proto pokud se předpokládá časově náročnější operace, jako je například zpracování MMS, případně ukládání údajů do databáze a podobně, v metodě `onReceive`, která se aktivuje po přijetí MMS, se jen spustí služba na pozadí, která zdlouhavější operaci zrealizuje.

Broadcast je považován za platný jen tehdy, když běží metoda `OnReceive`. Po jejím doběhnutí Android Broadcast Receiver ukončí. Z toho logicky vyplývá, že tímto mechanizmem není možné spustit asynchronní operace, které budou muset později realizovat callback do receiveru. Například není možné zobrazit dialog ani spustit aktivitu pomocí metody `startActivityForResult()`.



Upozornění: Metoda `onReceive()` musí skončit do 10 sekund. Pro dlouhotrvající operace je potřeba spustit službu.

Broadcast Receivery mohou být v aplikacích registrovány buď staticky v souboru *Android Manifest.xml*, nebo dynamicky voláním metody `registerReceiver()`. Broadcasty se registrují dynamicky, pokud jsou potřeba pouze tehdy, když je aktivita viditelná.

U statické registrace je potřeba do manifestu přidat tagy `<receiver>` a `<intent-filter>`.

```
<receiver
  android:enabled=["true" | "false"]
  android:exported=["true" | "false"]
  android:icon="drawable resource"
  android:label="string resource"
  android:name="string"
  android:permission="string"
  android:process="string" >
  ...
</receiver>
```

- `enabled` – umožňuje povolit nebo zakázat receiver.
- `exported` – pokud je hodnota atributu nastavena na hodnotu `true`, potom tento přijímač může přijímat vysílání i mimo aplikace. Pokud je hodnota nastavena na `false`, potom přijímač může přijímat pouze záměry, které jsou vysílány dalšími komponentami aplikace.
- `name` – název třídy, která implementuje přijímač.
- `permission` – řetězec s oprávněními, která musí mít přijímač pro příjem záměrů.

Broadcast je možné odeslat dvěma způsoby:

- **Normal Broadcast** odesílaný pomocí metody `Context.sendBroadcast()`. Broadcast je asynchronní, všechny zainteresované Broadcast Receivery ho tedy dostanou v nedefinovaném pořadí. Zpravidla jsou broadcasty zpracovány postupně, aby se zabránilo nárazovému zatížení systému



Poznámka: Normal Broadcast není možné zrušit.

- **Ordered Broadcast** – posílá se pomocí metody `Context.sendOrderedBroadcast()`. V daném okamžiku zpracovává zprávu maximálně jeden Broadcast Receiver. Pořadí se určí pomocí atributu `android:priority` v objektu `IntentFilter` daného receiveru.



Poznámka: Po zpracování zprávy může každý Brodcast Receiver určit, zda odevzdá výsledek dalšímu Broadcast Receiveru v pořadí nebo zruší daný broadcast a dále už se tento odevzdávat nebude.

Nejčastěji se používají broadcasty vázané na systémové události:

- `ACTION_BOOT_COMPLETED` – posílá se po náběhu systému. Tento broadcast vyžaduje oprávnění `RECEIVE_BOOT_COMPLETED`.
- `ACTION_MEDIA_BUTTON` – odešle se po stisknutí tlačítka ovládajícího přehrávač (např. na sluchátkách). Záměr v poli `EXTRA_KEY_EVENT` obsahuje dodatečné informace o daném tlačítku.
- `ACTION_CAMERA_BUTTON` – odešle se po stisknutí tlačítka fotoaparátu.
- `ACTION_MEDIA_EJECT` – odesílá se po vysunutí externího úložiště. Aplikace musí ukončit z tohoto úložiště čtení a zápisy.
- `ACTION_MEDIA_MOUNTED` – odešle se po vložení externího úložiště.
- `ACTION_MEDIA_UNMOUNTED` – odešle se po vyjmutí externího úložiště.
- `ACTION_BATTERY_LOW` – odešle se spolu s oznámením o nízkém stavu energie v baterii zařízení.
- `ACTION_SCREEN_ON` – odešle se po zapnutí obrazovky.
- `ACTION_SCREEN_OFF` – odešle se po vypnutí obrazovky.



Poznámka: Pokud je vyžadováno oprávnění, daný broadcast přijmou pouze receivery, které toto oprávnění mají definované v *AndroidManifest.xml* v sekci `<uses-permission>`.

Vyžádání oprávnění při přijímání broadcastu se definuje při zaregistrování Broadcast Receiveru.

Příklad – Broadcast Receiver registrovaný v aplikačním manifestu

V souboru *AndroidManifest.xml* deklarujte receiver. Deklaraci můžete vložit před závěrečný tag `</application>`.

```
<receiver
  android:name=".Receiver"
  android:exported="false" >
  <intent-filter>
    <action android:name="com.example.obeznik1.zobraz_toast" >
    </action>
  </intent-filter>
</receiver>
```

V manifestu je potřeba deklarovat i oprávnění, které bude potřebovat aplikace přijímající záměr. V tomto případě je to jedna a ta samá aplikace.

```
<uses-permission android:name="android.permission.VIBRATE" />
```

Návrh layoutu hlavní aktivity obsahuje jediné tlačítko na odeslání záměru.

```
<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/button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="Broadcast"
    android:textSize="30sp">
  </Button>

</LinearLayout>
```

V kódu hlavní aktivity po stisknutí tlačítka bude v metodě `sendBroadcast` vyslán záměr. Následně bude přijat a zobrazí toast.

```
public class MainActivity extends Activity
{
  private static final String MUJ_INTENT =
    "com.example.obeznik1.zobraz_toast";

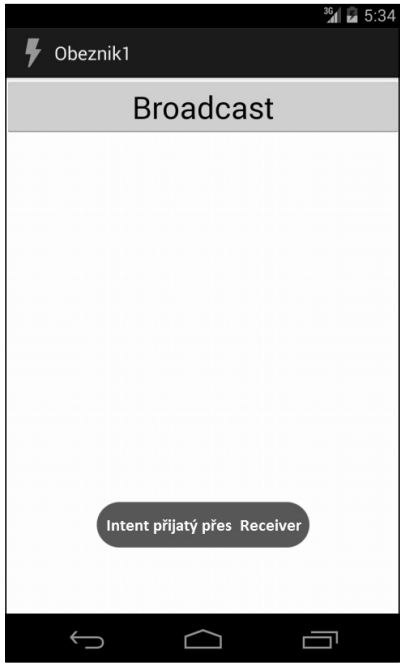
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }

  public void onClick(View View)
  {
    sendBroadcast(new Intent(MUJ_INTENT),
    android.Manifest.permission.VIBRATE);
  }
}
```

Třída `Receiver` rozšiřuje `BroadcastReceiver` a indikuje přijetí broadcastu zavibrováním a oznámením přes toast.

```
public class Receiver extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
    Vibrator v = (Vibrator) context
      .getSystemService(Context.VIBRATOR_SERVICE);
    v.vibrate(300);
    Toast.makeText(context, "Intent přijatý přes Receiver",
      Toast.LENGTH_LONG).show();
```

```
  }
}
```



Obrázek 9.1: Uživatelské rozhraní aplikace

Příklad – dynamicky registrovaný Broadcast Beceiver

Příklad bude fungovat stejně jako předchozí, jen Broadcast Receiver bude registrovaný dynamicky. Návrh layoutu hlavní aktivity je také stejný a obsahuje pouze jediné tlačítko na odeslání záměru. Stejný je i kód třídy `Receiver` rozšiřující `BroadcastReceiver`. Tato třída indikuje přijetí broadcastu zavibrováním a textovým oznámením přes toast.

Rozdíl je v kódu hlavní aktivity, kde se využívá metoda `registerReceiver` třídy `LocalBroadcastManager`. V případě potřeby můžete volat metodu `unRegisterReceiver` na zrušení registrace receiveru.

```
public class MainActivity extends Activity
{
  private static final String IFILTER =
    "com.example.obeznik2.zobraz_toast";
  private final IntentFilter intentFilter = new IntentFilter(IFILTER);
  private final Receiver receiver = new Receiver();
  private LocalBroadcastManager localBroadcastMgr;

  @Override
```

```
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    localBroadcastMgr = LocalBroadcastManager
      .getInstance(getApplicationContext());
    localBroadcastMgr.registerReceiver(receiver, intentFilter);
    setContentView(R.layout.activity_main);
  }

  public void onClick(View View)
  {
    localBroadcastMgr.sendBroadcast(new Intent(IFILTER));
  }

  @Override
  protected void onDestroy()
  {
    localBroadcastMgr.unregisterReceiver(receiver);
    super.onDestroy();
  }
}
```

Můžete vyzkoušet napsat více recieverů a podle toast notifikací sledovat, v jakém pořadí záměr jednotlivé receivery přijaly.

Soubor *Receiver1.java*:

```
public class Receiver1 extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
    Toast.makeText(context, "Intent přijatý přes Receiver1",
    Toast.LENGTH_LONG).show();
  }
}
```

Soubor *Receiver2.java*:

```
public class Receiver2 extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
    Toast.makeText(context, "Intent přijatý přes Receiver2",
      Toast.LENGTH_LONG).show();
  }
}
```

V dalším pokusu vyzkoušejte receivery s různě nastavenou prioritou.

Soubor *Receiver1.java*:

```
public class Receiver1 extends BroadcastReceiver
{
```

```
  @Override
  public void onReceive(Context context, Intent intent)
  {
    if (isOrderedBroadcast())abortBroadcast();
    Toast.makeText(context, "Intent přijatý přes Receiver1",
      Toast.LENGTH_LONG).show();
  }
}
```

Priority receiverů jsou definované v souboru *AndroidManifest.xml*.

```
<receiver
  android:name=".Receiver2"
  android:exported="false" >
  <intent-filter android:priority="10" >
    <action android:name="com.example.Obeznik3.zobraz_toast" >
    </action>
  </intent-filter>
</receiver>
<receiver
  android:name=".Receiver3"
  android:exported="false" >
  <intent-filter android:priority="1" >
    <action android:name=" com.example.Obeznik3.zobraz_toast " >
    </action>
  </intent-filter>
</receiver>
```

Prioritu pro receiver nastavíte i v kódu.

```
private final Receiver1 mReceiver = new Receiver1();
...
IntentFilter intentFilter = new IntentFilter(IFILTER);
intentFilter.setPriority(3);
registerReceiver(mReceiver, intentFilter);
```

Příklad – indikace příchozího hovoru

V tomto příkladu aplikace zareaguje na záměr indikující změnu stavu telefonu. Aplikace potřebuje v manifestu povolení ke čtení stavu telefonu:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

a registraci záměru

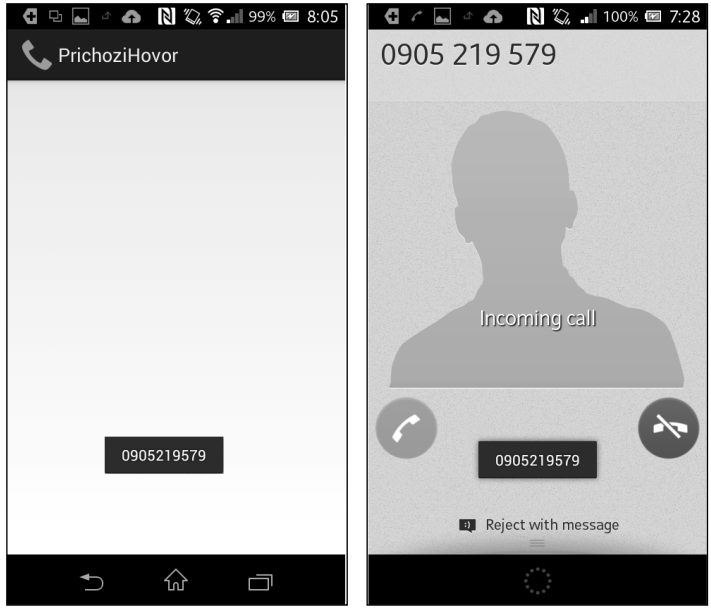
```
<receiver android:name="TelefonReceiver" >
  <intent-filter>
    <action android:name="android.intent.action.PHONE_STATE" >
    </action>
  </intent-filter>
</receiver>
```

Jelikož budeme příchozí hovor indikovat přes toast, neobsahuje layout hlavní aktivity žádné prvky. Ani do kódu hlavní aktivity není potřeba po vytvoření projektu nic doplňovat. Na příjem záměru vytvořte třídu `TelefonReceiver`, která rozšiřuje třídu `BroadcastReceiver`.

Kód v souboru *TelefonReceiver.java*.

```
public class TelefonReceiver extends BroadcastReceiver
{
  @Override
  public void onReceive(Context context, Intent intent)
  {
    Bundle extras = intent.getExtras();
    if (extras != null)
    {
      String stav = extras.getString(TelephonyManager.EXTRA_STATE);
      if (stav.equals(TelephonyManager.EXTRA_STATE_RINGING))
      {
        String cislo =
          extras.getString(TelephonyManager.EXTRA_INCOMING_NUMBER);
        Toast.makeText(context, cislo, Toast.LENGTH_LONG).show();
      }
    }
  }
}
```

Pokud na telefon někdo zavolá po spuštění aplikace zobrazí se toast s číslem volajícího. Aplikaci téměř okamžitě překryje aktivita indikující příchozí hovor, avšak toastová notifikace zůstane nějaký čas zobrazena.



Obrázek 9.2: Vlevo je toast v aplikaci Příchozí hovory. Při vyzvánění příchozího hovoru aplikaci překryje aktivita indikující příchozí hovor (vpravo), toastová notifikace ale zůstane chvíli zobrazená.

KAPITOLA 10

Ukládání údajů

V této kapitole:
- Ukládání údajů
- Ukládání údajů do souboru
- Ukládání dočasných souborů
- Databáze SQLite
- Interakce aplikace s databází
- Příklad – čtenářský deník
- Datový model
- Návrh uživatelského rozhraní
- Programování aplikační logiky

Ukládání údajů

I mobilní aplikace mohou pracovat s relativně velkými objemy dat. Android proto obsahuje několik tříd na ukládání údajů a má implementovanou databázi SQLite.

Třída SharedPreferences

Třída `SharedPreferences` 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`.

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 a v databázové terminologii provést commit, aby se natrvalo zapsaly pomocí metody `SharedPreferences.Editor.commit()`. Po potvrzení údajů je možné aplikaci ukončit a změny budou zapsány natrvalo.

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 1–1000. 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"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <TextView
    android:id="@+id/top_label"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/top_string"
    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="@string/nuluj_string" >
  </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="@string/hra_string" >
  </Button>

</RelativeLayout>
```

Definice řetězců v souboru *strings.xml*.

```
<resources>
  <string name="app_name">Top skóre</string>
  <string name="hra_string">HRA</string>
  <string name="top_string">Top skóre:</string>
  <string name="nuluj_string">Nuluj</string>
</resources>
```

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 tou 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 Activity
{
  private static String TOP_SKORE = "top_skore";

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    final SharedPreferences sharedPreferences =
      getPreferences(MODE_PRIVATE);
    setContentView(R.layout.activity_main);

    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 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 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"));
      }
    });
  }
}
```

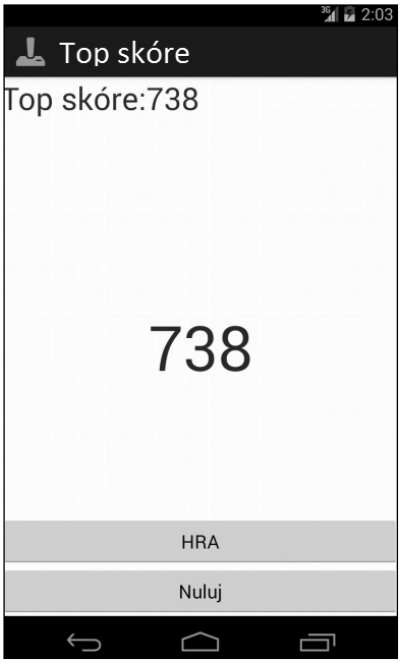
Příklad – PreferenceFragment

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 k tomuto účelu 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, v tomto případě je 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 k tomuto účelu uživatelské rozhraní.



Obrázek 10.1: Test aplikace. Zkuste po nastavení skóre aplikaci restartovat. Měla by se zobrazit naposledy uložená hodnota top skóre.

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="com.example.nickname.PrefActivity$MujPreferenceFragment"
  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 Nickname "
    android:key="nickname"
```

```
    android:negativeButtonText="Zruš"
    android:positiveButtonText="OK"
    android:title="Nickname">
  </EditTextPreference>
</PreferenceScreen>
```

V kódu hlavní aktivity vytvoříme obsluhu události 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));
  }
}
```

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 = "MujPrefsFragment";

    private OnSharedPreferenceChangeListener listener;
    private Preference preference;

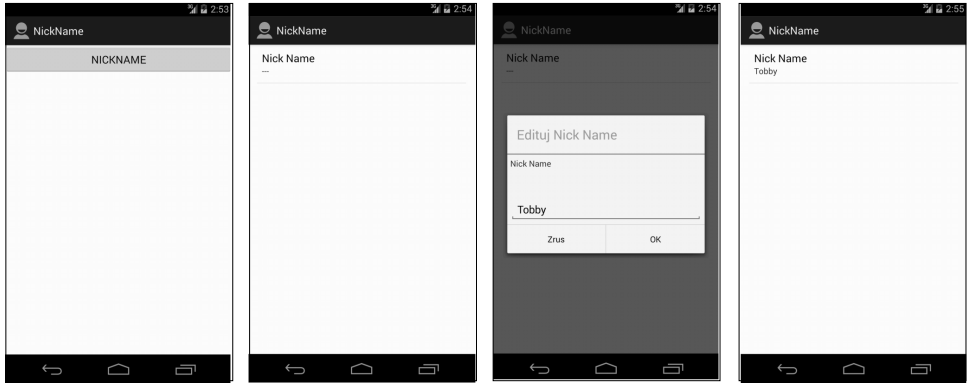
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.nick_prefs);
      preference = (Preference) getPreferenceManager()
        .findPreference(NICKNAME);
```

```
      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

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 příkla-

dem 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 MAC.

Příklad – ukládání do souboru v interní paměti

Aplikace po spuštění zkontroluje, zda existuje textový soubor s daným jménem. Pokud ne, potom 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:

```
public class PrefActivity extends Activity
{
  private static final String NICKNAME = "nickname";

  @Override
  protected void onCreate(Bundle savedInstanceState)
```

Kód hlavní aktivity obsahuje metody pro zápis údajů do textového souboru a jejich čtení.

```
public class MainActivity extends Activity
{
  private String TAG = "SouborInterni";
  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");}
  }

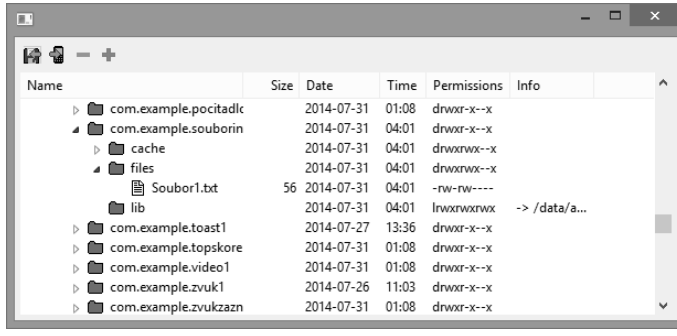
  private void souborZapis() throws FileNotFoundException
  {
    FileOutputStream fos = openFileOutput(soubor, MODE_PRIVATE);
    PrintWriter pw = new PrintWriter(new BufferedWriter(
      new OutputStreamWriter(fos)));
    pw.println("Lorem ipsum dolor sit amet");
    pw.println("consectetur adipisicing 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 průzkumníka souborů na reálném zařízení nebo pomocí nástroje Dalvik Debug Monitor na emulátoru můžete zkontrolovat, že soubor byl skutečně vytvořen. 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 soubor vytvořen

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.



Upozornění: 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`.

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"
  android:id="@+id/linearLayout"
  android:layout_width="match_parent"
```

```
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <ImageView
    android:id="@+id/image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:contentDescription="@string/hello_world"
    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 Activity
{
  private String TAG = "SouborExterni";

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

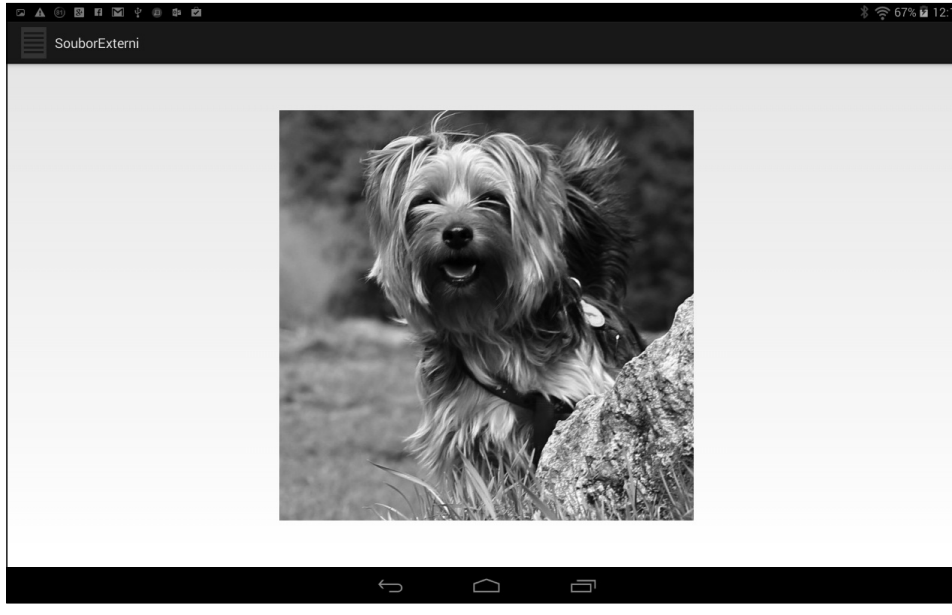
    if (Environment.MEDIA_MOUNTED.equals(Environment
      .getExternalStorageState()))
    {
      File soubor =
        new File(getExternalFilesDir(Environment.DIRECTORY_PICTURES),
        "tobby.jpg");

      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));
```

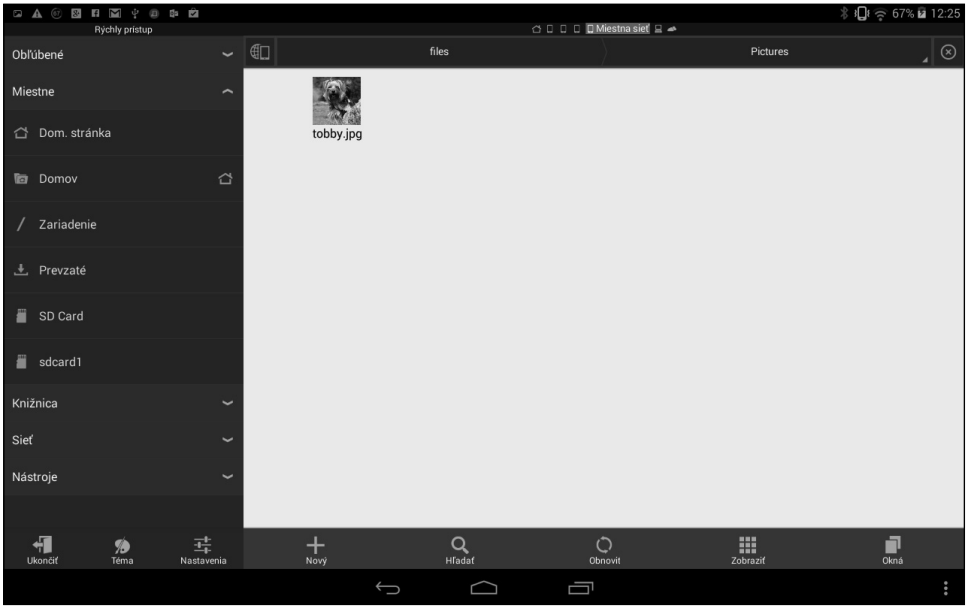
```
      kopiruj(bis, bos);
    }
    catch (FileNotFoundException e)
    {Log.e(TAG, "FileNotFoundException");}
  }

  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 10.4: Test aplikace

Pomocí vhodného souborového manažeru můžete zkontrolovat, zda se soubor skutečně zkopíroval do externího úložiště.



Obrázek 10.5: Soubor v aplikaci ES File Explorer

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(); // popř. 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, ať už je to Android, Windows 8, iOS, Mono nebo třeba Python.



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, Oraclu, 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. Proto se používá například pro CMS (content management system) weby či jako připojená databáze pro desktopové nebo mobilní aplikace.



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 SSD disky, 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 také vytvářet pohledy. SQLite podporuje transakce ACID (Atomicity, Consistency, Isolation, Durability).

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 SQL příkazy a databázový server je vykoná. Pokud se jedná o dotaz, typicky SQL příkaz `SELECT`, databáze vrátí množinu výsledků, které dotazu vyhovují.

Při ladění databázových aplikací na klasických platformách – Windows, Windows Server, Linux, Mac OS – je dobrým zvykem nejprve SQL příkazy, s kterými bude aplikace pracovat, samostatně otestovat. V praxi to probíhá tak, že se přes klientskou konzolovou aplikaci zadá-

vají 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 apod.
- Vkládání údajů do databázových tabulek
- Výběr údajů pomocí příkazů `SELECT`
- Změny a mazání údajů

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 ke které je 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, správné fungování aplikace vyžaduje i pátý atribut, unikátní identifikátor každé položky. Starosti s unikátním identifikátorem naštěstí mít nebudete, 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.

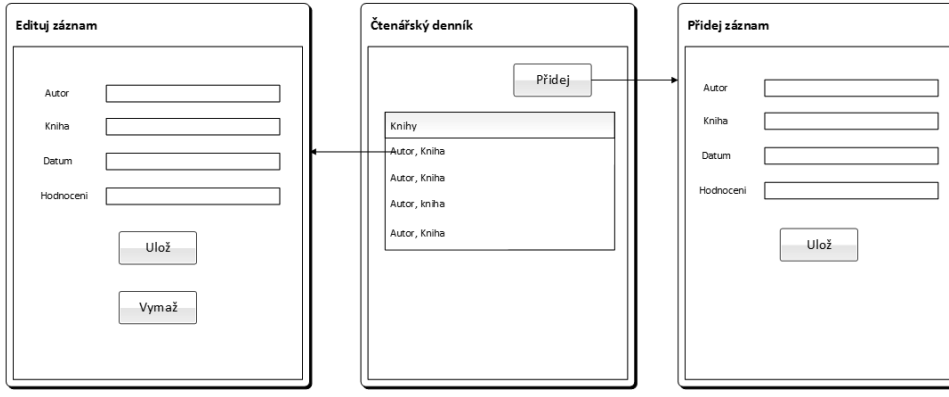
Z hlediska uživatelského rozhraní vytvoříme postupně tři aktivity.

- **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

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 SQLite databázi, která je součástí Androidu, nebudeme ale zadávat SQL příkazy na zařízeních s Androidem – hlavně na telefonech by to nebylo pohodlné. Využijeme skutečnosti, že SQLite je multiplatformní databáze, a nainstalujete si ji na vývojářský počítač s Windows, případně Mac OS. 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, spustit a můžete zadávat SQL příkazy.



Obrázek 10.6: Nákres obrazovek aktivit

Z archívu vybalte soubor *sqlite.exe* a spusťte ho. Na rozdíl od aplikace nemusíte vytvářet novou databázi, 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`.

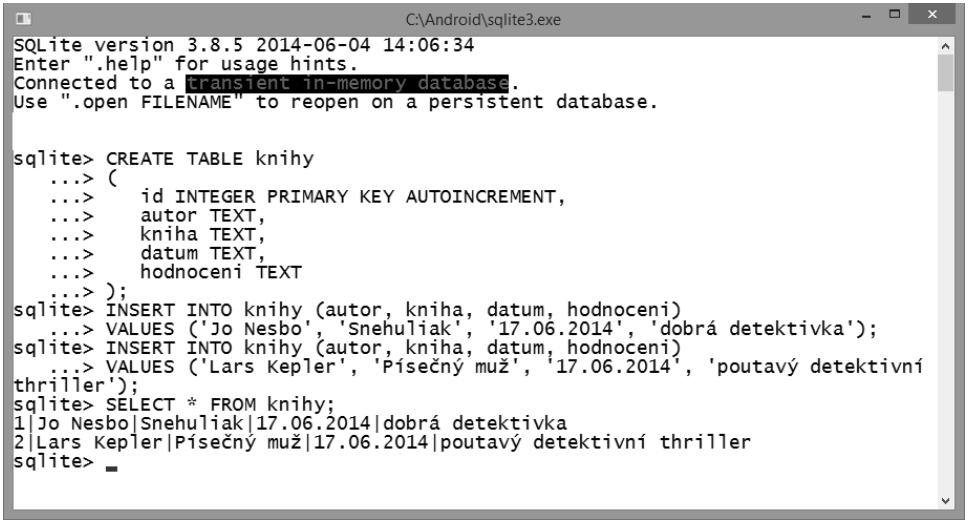
Následně otestujte příkaz `INSERT` na vložení nového záznamu do právě vytvořené tabulky. Vložíme dva záznamy:

```
INSERT INTO knihy (autor, kniha, datum, hodnoceni)
  VALUES ('Jo Nesbo', 'Sněhulák', '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 10.7: Testování SQL příkazů v konzolové aplikaci Sqlite3.exe

Ve výpisu si všimněte, že pro vámi vkládané záznamy byly vygenerované 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 SQL příkazech.

Datový model

Vytvořte projekt aplikace pro Android a pro hlavní aktivitu vyberte šablonu **Empty Activity**. V našem projektu jsme jako nejnižší verzi nastavili API 14: Android 4.0 (IceCream Sandwich).

V aplikaci potřebujete vytvořit vrstvu, která bude řešit ukládání 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.

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 nebo 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)`.

Ve složce zdrojových kódů Javy, v tomto případě `com.example.ctenarskydenik`, 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 SQL příkazu 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.



Poznámka: 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 in-memory databáze.
- `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 SQL příkazu 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 SQL příkaz.

```
@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 SQL příkazy, 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 tomto 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 SQL příkazu `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 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 `dejZaznamy()` vrátí všechny záznamy ve formě pole `ArrayList`.

```
public ArrayList<HashMap<String, String>> dejZaznamy()
{
  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);

  //Přesun na první záznam
  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 další záznam
  }
  return alVysledky;
}
```

Metodu můžete jednoduše modifikovat, aby byly vybrány pouze záznamy, které vyhovují specifické podmínce. Stačí do SQL příkazu `SELECT` doplnit klauzuli `WHERE`.

Metoda `dejZaznam()` vrátí konkrétní záznam podle zadaného identifikátoru `_id`. Záznam je vrácen jako objekt typu `HashMap`.

```
public HashMap<String, String> dejZaznam(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 Eclipse, nebo přímo v definičním XML souboru, 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.

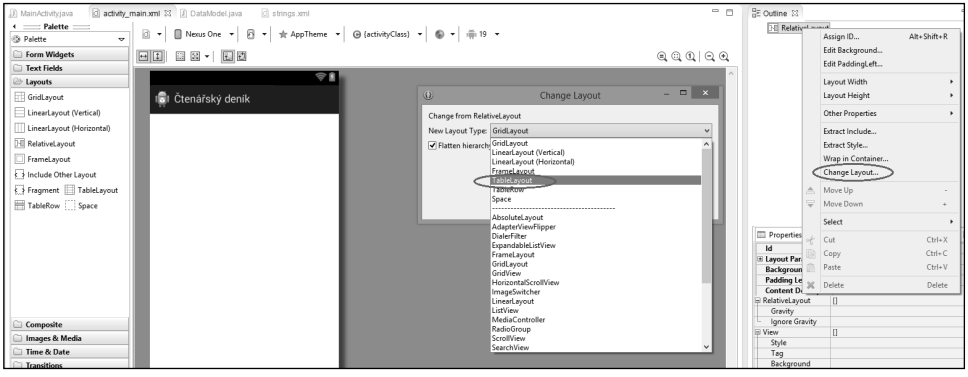
Abyste mohli po přidání každého prvku nastavit jeho parametry včetně textových řetězců v souboru *res/strings.xml*, definujte nejprve v tomto souboru všechny textové řetězce, které bude aplikace využívat.

```
<resources>
  <string name="app_name">Čtenářský deník</string>
  <string name="s_autor">Autor</string>
  <string name="s_kniha">Kniha</string>
  <string name="s_datum">Datum</string>
  <string name="s_hodnoceni">Hodnocení</string>
  <string name="nadpis_Knihy">Knihy</string>
  <string name="nadpis_pridej">Přidej záznam</string>
  <string name="nadpis_edituj">Edituj knihy</string>
  <string name="bt_pridej">Přidej</string>
  <string name="bt_ulož">Ulož</string>
  <string name="bt_edituj">Edituj</string>
  <string name="bt_vymaz">Vymaž</string>
</resources>
```

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`, který je podrobněji popsán ve třetí kapitole. 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 v návrhovém prostředí Eclipse. V okně **Outline** zobrazte místní nabídku položky **RelativeLayout**. Zvolte položku **ChangeLayout**. V zobrazeném dialogovém okně změňte typ na **TableLayout**.



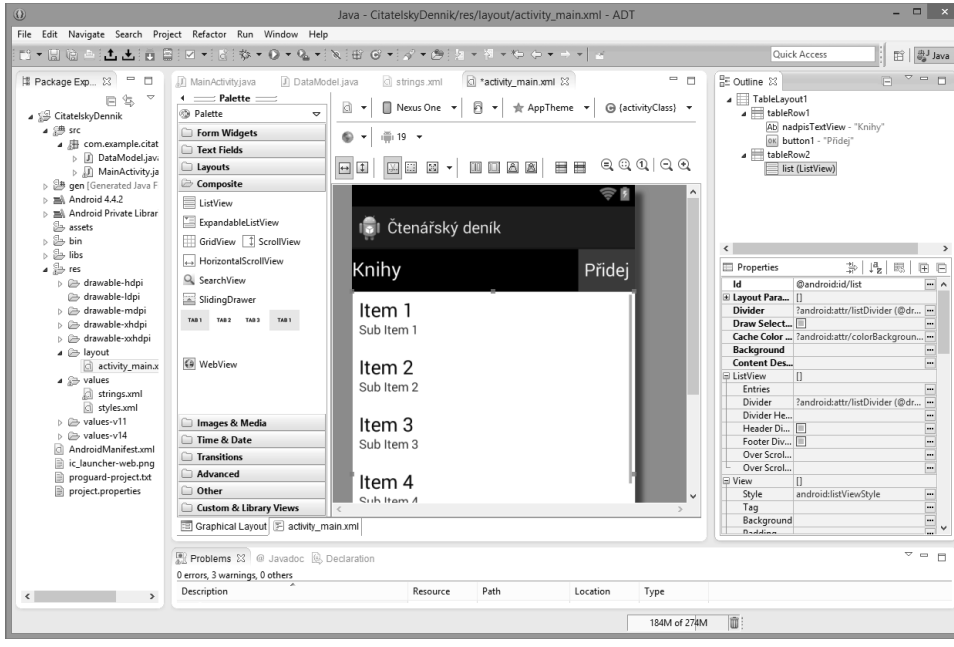
Obrázek 10.8: Změna typu layoutu ve vývojovém prostředí

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`, který je určen na výpis jednotlivých položek. Řádky, tj. prvky `TableRow`, přidáváte v Eclipse ze složky **Layouts**.

Výsledek vizuálního návrhu si můžete prohlédnout v souboru.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/TableLayout1"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${packageName}.${activityClass}" >

  <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="@string/nadpis_Knihy"
```



Obrázek 10.9: Návrh uživatelského rozhraní hlavní aktivity

```
      android:textAppearance="?android:attr/textAppearanceLarge"
      android:textColor="#FFFFFF"
      android:layout_weight="1"/>

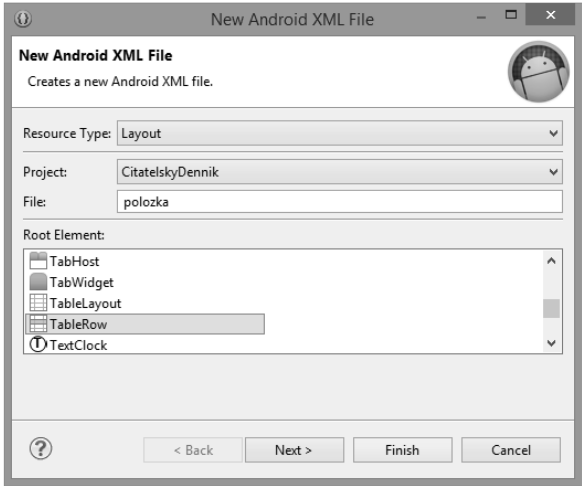
    <Button
      android:id="@+id/pridejKnihuButton"
      android:background="#444444"
      android:onClick="pridejKnihu"
      android:text="@string/bt_pridej"
      android:textColor="#FFFFFF"
      android:textSize="20sp" />
  </TableRow>

  <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`.



Obrázek 10.10: Vytvoření layoutu k zobrazení položky seznamu ListView

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 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/scroll_view_1"
      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="pridejKnihu"
      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 obsahuje i obrazovka této aktivity 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*
- *PridejZaznam.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 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.dejZaznamy();

    if(knihyHM.size()!=0)
    {
      ListView lw = getListView();
      lw.setOnItemClickListener(new OnItemClickListener()
      {
        //klepnutí na položku seznamu
```

```
        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.autorET, R.id.knihaET});
      setListAdapter(adapter);
    }
  }

  public void pridejKnihu(View view)
  {
    Intent theIntent = new Intent(getApplication(), NovyZaznam.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 dění přenese zpět na hlavní obrazovku.

Kompletní kód v souboru *PridejZaznam.java*.

```
public class PridejZaznam extends Activity
{
  EditText autorET;
  EditText knihaET;
  EditText datumET;
  EditText hodnoceniET;

  DataModel dm = new DataModel(this);

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.pridej_zaznam);

    autorET = (EditText) findViewById(R.id.autorET);
    knihaET = (EditText) findViewById(R.id.knihaET);
    datumET = (EditText) findViewById(R.id.datumET);
```

```
    hodnoceniET = (EditText) findViewById(R.id.hodnoceniET);
  }

  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", hodnoceniET.getText().toString());

    dm.vlozZaznam(dotazHM);
    // volá 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 **Edituj** 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.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.hodnoceniET);
```

```
    Intent ii = getIntent();
    String knihaId = ii.getStringExtra("knihaId");
    HashMap<String, String> knihaHM = dm.dejZaznam(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.hodnoceniET);

    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.

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

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

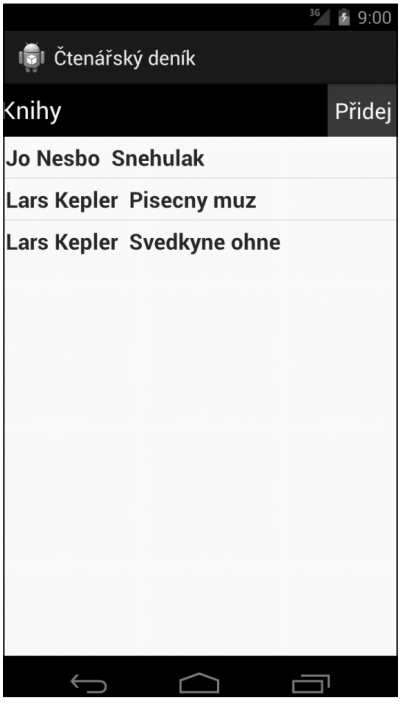
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.ctenarskydenik.MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
  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

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name="com.example.ctenarskydenik.PridejZaznam"
      android:label="" >
    </activity>
    <activity android:name="com.example.ctenarskydenik.EditujZaznam"
      android:label="" >


  </application>

</manifest>
```

Nyní můžete aplikaci otestovat. Přidejte několik záznamů a zkuste je editovat a mazat.



Obrázek 10.13: Spuštění aplikace – obrazovka hlavní aktivity se seznamem knih



Obrázek 10.14: Spuštění aplikace – obrazovka aktivity na přidání nového záznamu

KAPITOLA 11

Aplikace pracující s údaji JSON, XML

V této kapitole:
- Námět
- Zdroj údajů pro aplikaci
- Návrh uživatelského rozhraní
- Aplikace na načítání údajů ve formátu JSON
- Aplikace na načítání údajů ve formátu XML
- Statická data ve formátu XML

Námět

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. Možná to specifikováno nebude a potom budete muset webovou aplikaci poskytující údaje vytvořit sami.

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 nebo 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 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 konzolu *https://developer.yahoo.com/yql/console/*.

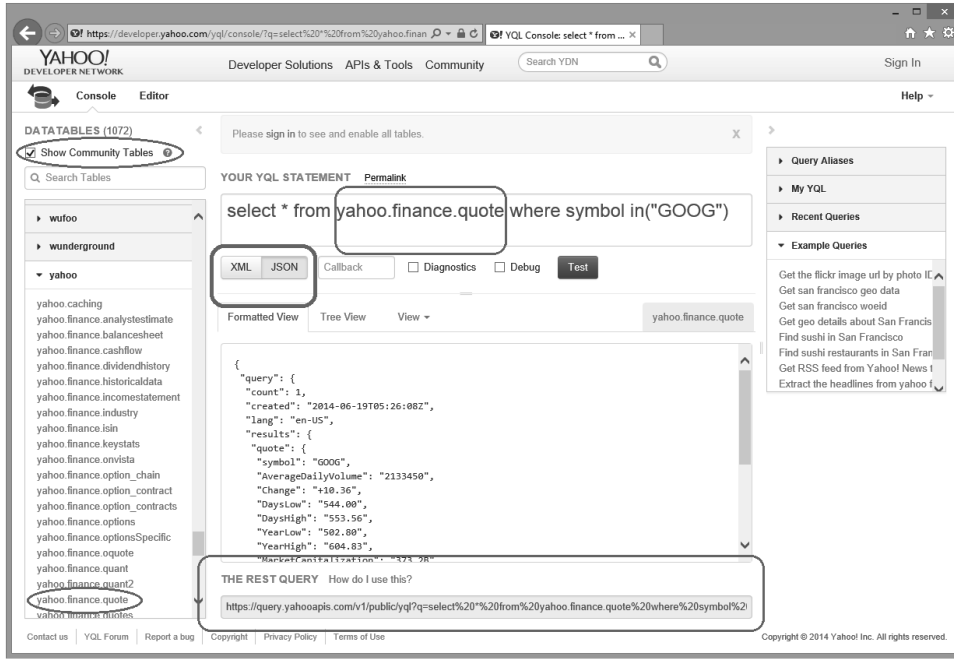
První, co vás napadne, bude pravděpodobně nejpopulárnější služba Yahoo Weather, která poskytuje informace o počasí. Jako parametr je potřeba specifikovat město a jednotky zobrazení teploty. Město se zadává nikoliv pomocí názvu, ale využívá se parametr WOEID (Where On Earth IDentifier). Pro zajímavost: WOEID Bratislavy je 818717 a Prahy 796597. Pokud tedy 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 zobrazování, 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 11.1: Nastavení generování zdrojových dat

Podle toho, zda stisknete tlačítko JSON nebo XML, vygeneruje aplikace údaje v požadovaném formátu.

Příklad údajů ve formátu JSON:

```
{
  "query": {
    "count": 1,
    "created": "2014-06-19T05:34:31Z",
    "lang": "en-US",
    "results": {
      "quote": {
        "symbol": "GOOG",
        "AverageDailyVolume": "2133450",
        "Change": "+10.36",
        "DaysLow": "544.00",
        "DaysHigh": "553.56",
        "YearLow": "502.80",
        "YearHigh": "604.83",
        "MarketCapitalization": "373.2B",
        "LastTradePriceOnly": "553.37",
        "DaysRange": "544.00 - 553.56",
        "Name": "Google Inc.",
        "Symbol": "GOOG",
        "Volume": "1737343",
        "StockExchange": "NasdaqNM"
      }
    }
  }
}
```

Příklad údajů ve formátu XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<query xmlns:yahoo="http://www.yahooapis.com/v1/base.rng"
  yahoo:count="1" yahoo:created="2014-06-23T12:02:59Z" yahoo:lang="en-US">
  <results>
    <quote symbol="GOOG">
      <AverageDailyVolume>2178110</AverageDailyVolume>
      <Change>+1.46</Change>
      <DaysLow>550.3915</DaysLow>
      <DaysHigh>557.58</DaysHigh>
      <YearLow>502.80</YearLow>
      <YearHigh>604.83</YearHigh>
      <MarketCapitalization>375.3B</MarketCapitalization>
      <LastTradePriceOnly>556.36</LastTradePriceOnly>
      <DaysRange>550.3915 - 557.58</DaysRange>
      <Name>Google Inc.</Name>
      <Symbol>GOOG</Symbol>
      <Volume>4497344</Volume>
      <StockExchange>NasdaqNM</StockExchange>
    </quote>
  </results>
</query>
```

Pro vás bude důležité pole **THE REST QUERY** v dolní části formuláře webového portálu, které obsahuje kompletní URL adresu 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 čárkou:

```
select * from yahoo.finance.quote where symbol in("GOOG,MSFT")
```

Pokud chcete možnosti aplikace rozšířit, můžete měnit textový řetězec s označením firmy uvnitř URL adresy. Řetězec s URL adresou můžete rozdělit na tři řetězce, přičemž prostřední z nich bude proměnná typu `string`, která bude obsahovat zkratku aktuálně vybrané firmy.

https://query.yahooapis.com/v1/public/yql?q=select%20%20from%20yahoo.finance.quote%20where%20symbol%20in(%22GOOG%22)&format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys&callback=*

https://query.yahooapis.com/v1/public/yql?q=select%20%20from%20yahoo.finance.quote%20where%20symbol%20in(%22GOOG%22)&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys*

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="${packageName}.${activityClass}" >

  <TextView
    android:layout_gravity="right"
    android:text="@string/zmena" />

  <TextView
    android:layout_column="1"
    android:id="@+id/akce"/>

  <TextView
    android:layout_column="0"
```

```
    android:layout_gravity="right"
    android:text="@string/min" />

  <TextView
    android:layout_column="1"
    android:id="@+id/min"/>

  <TextView
    android:layout_column="0"
    android:layout_gravity="right"
    android:text="@string/max" />

  <TextView
    android:layout_column="1"
    android:id="@+id/max"/>
</GridLayout>
```

Aplikace na demonstraci zobrazování údajů získaných ve formátu JSON je jednoduchá, proto ve zdrojích budete potřebovat definovat kromě záhlaví aplikace jen tři další textové řetězce:

```
<resources>
  <string name="app_name">Údaje JSON</string>
  <string name="zmena">Změna:</string>
  <string name="min">Denní minimum:</string>
  <string name="max">Denní maximum:</string>
</resources>
```



Obrázek 11.2: 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 stránce *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> ::= <řetězec> : <hodnota>
<pole> ::= [] | [ <prvky> ]
<prvky> ::= <hodnota> | <hodnota> , <prvky>
<hodnota> ::= <řetězec> | <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.

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 URL adresu 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 ze vstupu `InputStream`, 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, k 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.

Dříve než začnete JSON dokument v textovém řetězci parsovat, je potřeba provést jednu úpravu. Služba yahoo.finance vám vrátila údaje o akciích v těle funkce `cbfunc({...});`. Na parsování však potřebujete jen tělo JSON dokumentu, které začíná a končí složenou závorkou. Ze začátku dokumentu budete potřebovat odstranit sedmiznakový řetězec `cbfunc(` a z konce dokumentu dvojznakový řetězec `);`.

```
sJSON = sJSON.substring(7);
sJSON = sJSON.substring(0, sJSON.length()-2);
```

Pokud se podíváte na JSON údaje, 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 JSON dokumentu 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):

```
public class MainActivity extends Activity
{
  static String sURL = "http://query.yahooapis.com/v1/public/yql...";

  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 NactiUdajeAsyncTask().execute();
  }

  private class NactiUdajeAsyncTask extends AsyncTask<String,
    String, String>
  {
    protected String doInBackground(String... arg0)
    {
      DefaultHttpClient klientHTTP =
        new DefaultHttpClient(new BasicHttpParams());

      HttpPost httppost = new HttpPost(sURL);
      httppost.setHeader("Content-type", "application/json");
      InputStream inputStream = null;

      String sJSON = null;

      try
      {
        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();
      } catch (Exception e)
      {
        e.printStackTrace();
      }
      finally
      {
        try{if(inputStream != null)inputStream.close();}
        catch(Exception e){}
      }

      JSONObject jsonObject;
      try
      {
        // vymaž znaky navíc
        sJSON = sJSON.substring(7);
        sJSON = sJSON.substring(0, sJSON.length()-2);
        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 na Internetu, musíte v aplikačním manifestu v souboru *AndroidManifest.xml* deklarovat příslušná oprávnění.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Po doplnění oprávnění do aplikačního manifestu můžete aplikaci vyzkoušet.

Aplikace na načítání údajů ve formátu XML

Kromě srozumitelnosti je hlavní výhodou XML dokumentů univerzálnost a multiplatformovost, proto se velmi často využívají k výměně informací v heterogenním prostředí.

Na parsování údajů z XML dokumentu se využívá buď DOM (Document Object Model), nebo XmlPullParser. Proto i náš příklad bude mít dvě varianty, které se budou lišit použitou technologií na parsování XML. Uživatelské rozhraní obou aplikací bude stejné – shodné s aplikací pro načítání údajů v JSON.

http://is.muni.cz/th/324576/fi_b/Bakalarska_prace_-_Marcel_Broz.txt?lang=cs

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 XML dokumentu. Umožňuje načítat XML dokument do objektové stromové struktury, se kterou je potom možné pohodlně manipulovat podle potřeby. Každý dokument podle DOM má hierarchickou stromovou strukturu.

K XML údajům přistupuje aplikace přes připojení realizované objektem typu `HttpURL Connection`. 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 XML dokumentu je použit objekt `DocumentBuilder` z `DocumentBuilderFactory`. Instance `DocumentBuilderFactory` slouží ke správnému formátování stažených údajů při vytváření 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:

```
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 připojení úspěšné?
      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 XML elementu
- `TEXT` – indikuje, že parser je nastavený na 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 XML dokumentu `<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
```

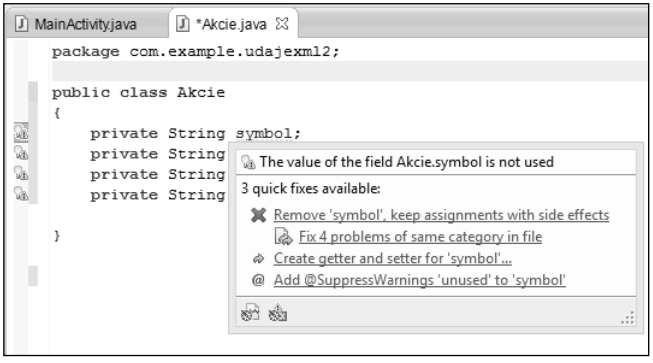
Pokud by bylo nutné načítat hodnotu atributu, použili byste metodu `getAttributeValue()`.

```
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 budeme v tomto nejjednodušším příkladu zobrazovat jen údaje pro jeden objekt, v tomto případě informace o akciích jedné firmy, budeme postupovat systematicky a vy-

tvoříme 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 Eclipse vygenerovat pro jednotlivé atributy automaticky.



Obrázek 11.3: 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 XML dokumentu 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 Activity
{
  static String sURL = "https://query.yahooapis.com/v1/public/yql..";

  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 (ClientProtocolException e)
    {
      e.printStackTrace();
    } catch (XmlPullParserException e)
    {
      e.printStackTrace();
    } catch (URISyntaxException e)
    {
      e.printStackTrace();
    } catch (IOException e)
    {
```

```
        e.printStackTrace();
      }
      finally {}
      return null;
    }

    public InputStream getUrlData(String url) throws URISyntaxException,
      ClientProtocolException, IOException
    {
      DefaultHttpClient client = new DefaultHttpClient();
      HttpGet method = new HttpGet(new URI(url));
      HttpResponse res = client.execute(method);
      return res.getEntity().getContent();
    }

    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());
    }
  }
}
```

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 XML dokumentu. 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 XML dokumentu prostřednictvím objektu `Resources` voláním metod `getResources()` a `openRawResources()`. Poslední ze jmenovaných metod vrátí objekt typu `InputStream`, který umožňuje údaje z XML dokumentu načítat, ale neumožňuje je měnit.

Vytvořte projekt aplikace pro Android a ve složce *assets* vytvořte soubor *data.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<prezidenti>
  <prezident>
    <id>1</id>
    <jmeno>George Washington</jmeno>
    <od>1789</od>
    <do>1797</do>
  </prezident>
  <prezident>
```

```
    <id>2</id>
    <jmeno>John Adams</jmeno>
    <od>1797</od>
    <do>1801</do>
  </prezident>
  ...
</prezidenti>
```

Texty ve zdrojích budou kromě názvu aplikace obsahovat jen jeden textový řetězec s textem záhlaví.

```
<resources>
    <string name="app_name">Statické XML</string>
    <string name="prezidenti">Prezidenti USA</string>
</resources>
```

Na zobrazení údajů z XML dokumentu využijeme prvek `ListView`.

Návrh uživatelského rozhraní v souboru *main_activity.xml*:

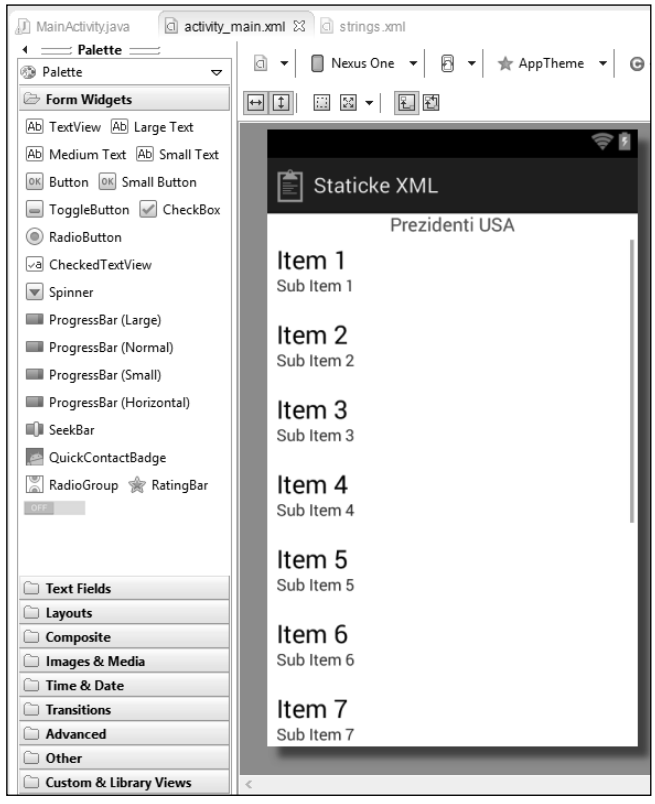
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${packageName}.${activityClass}" >

  <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>
```

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" />
```



Obrázek 11.4: Návrh uživatelského rozhraní

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 Eclipse 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 jmeno;
  private String odRok;
  private String doRok;

  public String getId()
  {
    return id;
  }
  ...

  @Override
  public String toString()
  {
    return id + ": " + jmeno + "\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 jen 44, nepotřebujeme realizovat čtení dokumentu ani jeho parsování v asynchronní třídě. Můžeme to pokojně provést v hlavním vlákně. Aby byl kód aplikace přehledně strukturovaný, budeme parsování realizovat v třídě `XMLPullParserHandler`.

Kód třídy *XMLPullParserHandler.java*:

```
public class XMLPullParserHandler
{
  List<PrezidentUSA> prezidenti;
  private PrezidentUSA prezident;
  private String text;

  public XMLPullParserHandler()
  {
    prezidenti = new ArrayList<PrezidentUSA>();
  }

  public List<PrezidentUSA> dajPrezidentov()
  {
    return prezidenti;
  }

  public List<PrezidentUSA> 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 PrezidentUSA();
            }
            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("jmeno"))
            {
              prezident.setJmeno(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;
  }
}
```

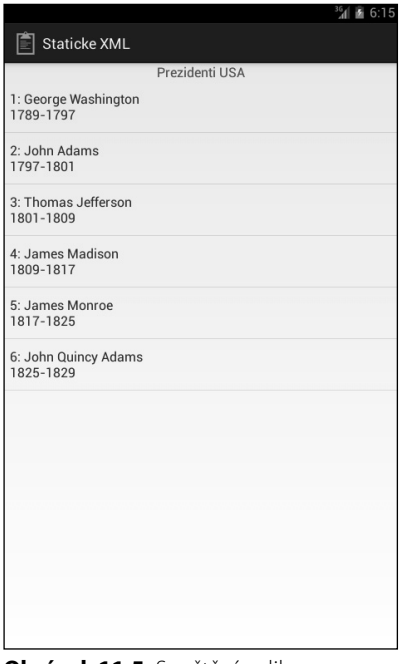
Jelikož jsme na parsování XML dokumentu vytvořili samostatnou třídu, kód hlavní aktivity bude jednoduchý a přehledný. Kód v souboru *main_activity.xml*:

```
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 11.5: Spuštění aplikace

KAPITOLA 12

Grafika a animace

V této kapitole:
- Zobrazení obrázku přes XML návrh
- Zobrazení obrázku programově
- Zobrazení obrázku z Internetu
- Vykreslování základních grafických tvarů
- Vykreslování na Canvas
- Kreslení dotykem na Canvas
- Dynamické vykreslování na Canvas s využitím View
- Dynamické vykreslování na Canvas s využitím SurfaceView
- Animace

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 k zobrazení statických nebo dynamicky se měnících grafických prvků.

Grafické prvky je možné vykreslovat buď přes `View`, nebo přes `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 XML souborech můžete vytvářet pomocí tříd `ColorDrawable`, `BitmapDrawable`, `ShapeDrawable` a `GradientDrawable`. XML soubory 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 XML souborech nebo programově.

Příklady v této kapitole budou postupně ilustrovat možnosti zobrazení obrázků a grafických tvarů jednak prostřednictvím XML definice, jednak programově v Java kódu. Ukážeme načítání a zobrazení obrázku z Internetu.

Zobrazení obrázku přes XML návrh

V prvním triviálním příkladu ukážeme zobrazení obrázku. Příklad má dvě varianty. V první zobrazíme obrázek pomocí prvku `View` definovaného v XML souboru. V druhé variantě zobrazíme obrázek pomocí Java kódu.

K zobrazení obrázku přidáme v návrhovém zobrazení do vizuálního kontejneru `Relative Layout` prvek `ImageView`.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/frame"
  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_centerInParent="true"
    android:contentDescription="@string/kitkat_desc"
    android:src="@drawable/kitkat3" />

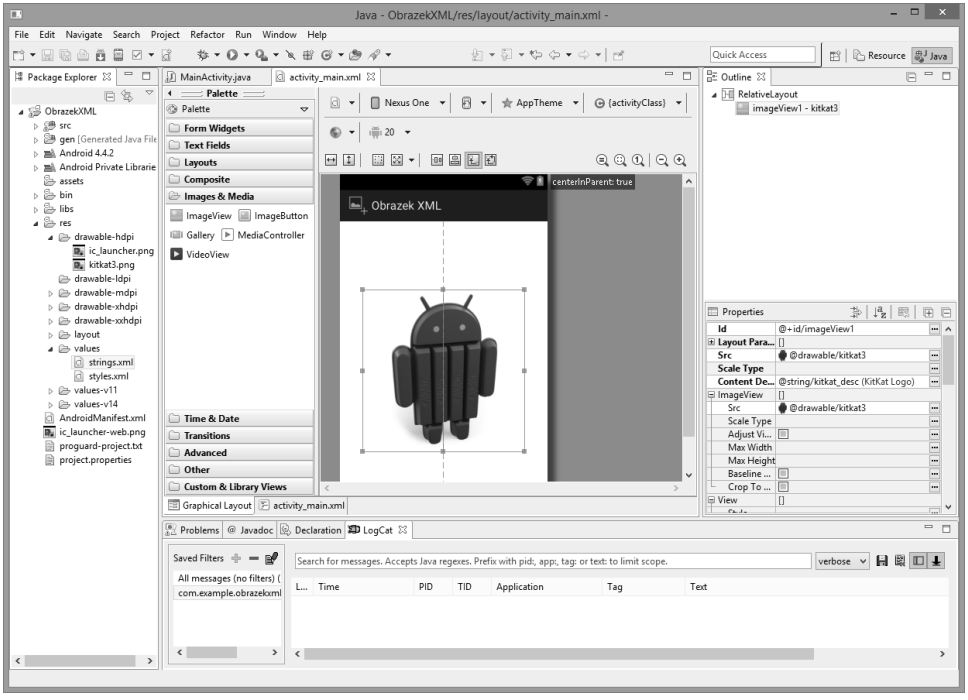
</RelativeLayout>
```

Obrázek přidejte do složky *res/drawable-hdpi*. Můžete ho zkopírovat přes schránku přímo do předmětné složky v Eclipse.

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/obrazek"/>
</LinearLayout>
```



Obrázek 12.1: Přidání obrázku do layoutu hlavní aktivity

Zobrazení obrázku programově

Pomocí metody `setContentView()` definujeme připojení na XML definiční soubor, 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.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/kontejner"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="${relativePackage}.${activityClass}" >
</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 XML definici.

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

    RelativeLayout relativeLayout = (RelativeLayout)
      findViewById(R.id.kontejner);

    ImageView imgView = new ImageView(getApplicationContext());
    imgView.setImageDrawable(
      getResources().getDrawable(R.drawable.kitkat3));

    int width = (int) getResources().getDimension(R.dimen.image_width);
    int height = (int) getResources().getDimension(R.dimen.image_height);

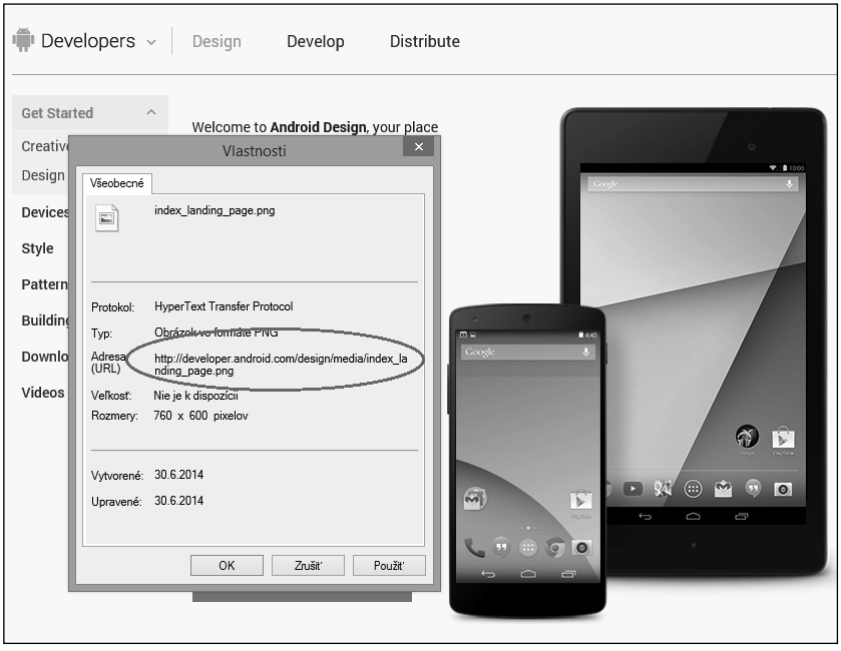
    RelativeLayout.LayoutParams params =
      new RelativeLayout.LayoutParams(width, height);
    params.addRule(RelativeLayout.CENTER_IN_PARENT);
    imgView.setLayoutParams(params);
    relativeLayout.addView(imgView);
  }
}
```

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. Uložení obrázku na webovém serveru není námětem této publikace, proto předpokládáme, že obrázek už na webu uložený je. Pro cvičnou aplikaci postačí získat URL adresu libovolného obrázku z libovolné webové stránky. Na obrázek v prohlížeči klepněte prvým tlačítkem myši. V zobrazené nabídce použijte příslušnou volbu ke zkopírování URL adresy a použijte ji v příkladu.



Obrázek 12.2: Získání URL adresy obrázku z webu pro cvičnou aplikaci

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`.

```
<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}" >

  <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="@string/hello_world"/>

</RelativeLayout>
```

Pokud byste potřebovali zobrazit obrázek ze zdrojů, nastavili byste hodnotu atributu.

```
android:src="@drawable/ic_launcher"
```

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 Activity
{
  public static final String URL = "http://developer.android...";
  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[] { URL });
  }

  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 httpConnection = (HttpURLConnection) connection;
        httpConnection.setRequestMethod("GET");
        httpConnection.connect();

        if (httpConnection.getResponseCode() == HttpURLConnection.HTTP_OK)
        {
          stream = httpConnection.getInputStream();
        }
      } catch (Exception ex)
      {
        ex.printStackTrace();
      }
      return stream;
    }
  }
}
```



Obrázek 12.3: Aplikace na zobrazení obrázku z webu

Vykreslování základních grafických tvarů

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.



Obrázek 12.4: Vytvoření definičního XML souboru grafiky

Postup vytvoření „XML grafiky“ je jednoduchý. Do příslušného adresáře umístěte XML soubor 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`. Definiční XML soubor grafiky nejsnáze vytvoříte pomocí nabídky **New** → **Android XML File**.

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 pouze 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 XML souboru 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" >

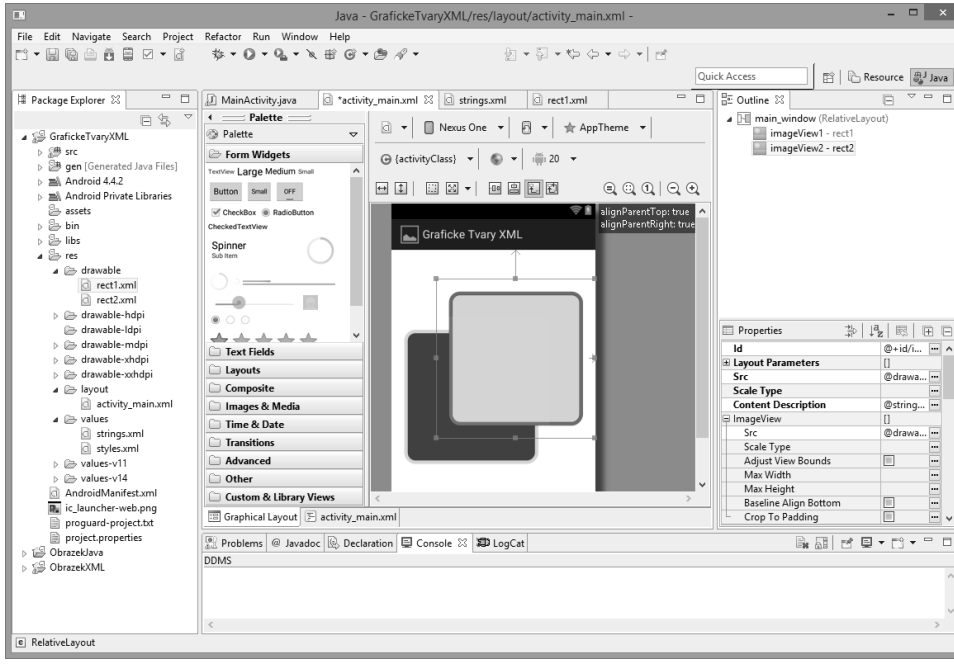
  <ImageView
    android:id="@+id/imageView1"
    android:layout_width="250dp"
    android:layout_height="250dp"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true"
    android:contentDescription="@string/rect1_desc"
    android:padding="20dp"
    android:src="@drawable/rect1" />

  <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 XML souboru 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 12.5: Statická grafika v návrhovém zobrazení

Zatímco se statická grafika definovaná v XML kódu 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. Aby byla nějaká změna a představili jsme více tvarů, namísto obdélníků vykreslíme barevné kruhy, tj. objekty `OvalShape`. XML definiční soubor 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/kontejner"
  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>
</resources>
```

Celé vykreslení obou prvků se odehrává v kódu hlavní aktivity:

```
public class MainActivity extends Activity
{
  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);

    // první modrý 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);

    // druhý zelený 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);

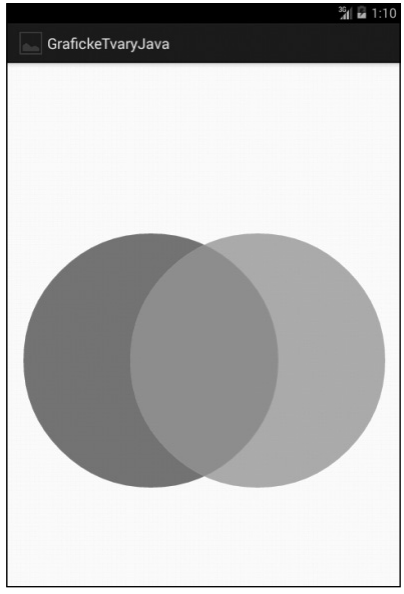
    // ... do RelativeLayout
    RelativeLayout.LayoutParams greenParams =
      new RelativeLayout.LayoutParams(height, width);
```

```
        greenParams.addRule(RelativeLayout.CENTER_VERTICAL);
        greenParams.addRule(RelativeLayout.ALIGN_PARENT_RIGHT);

        greenView.setLayoutParams(greenParams);

        rl.addView(greenView);

    }
}
```



Obrázek 12.6: Vykreslení grafických tvarů programovým kódem

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.

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.

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 i velikost textu. Můžete nastavit různé metody grafické optimalizace, například antialiasing na vyhlazení obrazu, aby se odstranily zubaté šikmé čáry, a podobně. K naznačeným účelům budete používat metody `setStrokeWidth()`, `setTextSize()`, `setColor()` či `setAntiAlias()`.

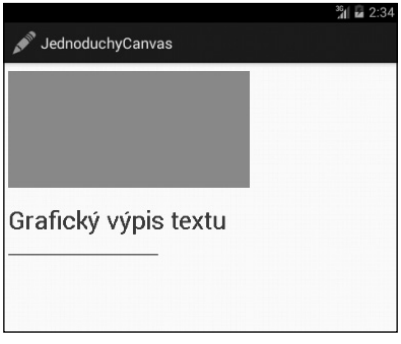
Vykreslování na `Canvas` budeme demonstrovat na triviálním příkladu, ve kterém nakreslíme obdélník, čáru a text.

Nejprve vytvoříme třídu `Kresleni`.

```
public class Kresleni extends View
{
  public Kresleni(Context context){super(context);}
  @SuppressLint("DrawAllocation") @Override
  protected void onDraw(Canvas canvas)
  {
    Paint paint = new Paint();
    paint.setColor(Color.GREEN);
    canvas.drawRect(10, 10, 300, 150, paint);
    paint.setColor(Color.BLUE);
    paint.setTextSize((float) 30.0);
    canvas.drawText("Grafický výpis textu", 10, 200, paint);
    canvas.drawLine(10, 230, 190, 230, paint);
  }
}
```

V kódu hlavní aktivity nastavíme tuto třídu jako kontextové `View` pro vykreslování v metodě `SetContentView()`. Z uvedeného vyplývá, že XML návrh hlavní aktivity se vůbec nepoužije.

```
public class MainActivity extends Activity
{
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(new Kresleni(this));
  }
}
```



Obrázek 12.7: 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 `Kresleni` 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 Kresleni extends View
{
  public Kresleni(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(4);
    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);
    }
  }
}
```



Obrázek 12.8: Příklad aplikace na kreslení prstem

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`.

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 javovém kódu hlavní aktivity. Figurka mění 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 je předtím vykreslený obrázek vymazán a nakreslen 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 Activity
{
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    final RelativeLayout rl =
      (RelativeLayout) findViewById(R.id.kontejner);
    final Bitmap bmp = BitmapFactory.decodeResource(getResources(),
      R.drawable.et);
    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("KresleniCanvas", "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);

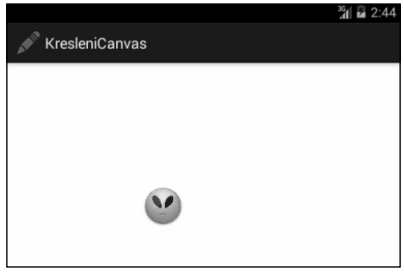
    mPainter.setAntiAlias(true);

  }

  @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 12.9: 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`. `Surface View` 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 SurfaceView pro zpětné volání.

```
private class HraView extends SurfaceView implements SurfaceHolder.Callback
{
  mSurfaceHolder = getHolder();
  mSurfaceHolder.addCallback(this);
}
```

Po vytvoření objektu SurfaceView je volaná metoda surfaceCreated(), v 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() je volaná 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.

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 Activity
{

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

    RelativeLayout rl = (RelativeLayout) findViewById(R.id.kontejner);
```

```
    final HraView bubbleView = new HraView(getApplicationContext(),
      BitmapFactory.decodeResource(getResources(), R.drawable.et));
    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();
    }

  }
}
```

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 XML souborech *rect1.xml* a *rect2.xml* umístěných ve složce *drawable*.

Uvádíme jen obsah definičního souboru prvního obdélníka, druhý se liší pouze 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 XML souboru 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>
```

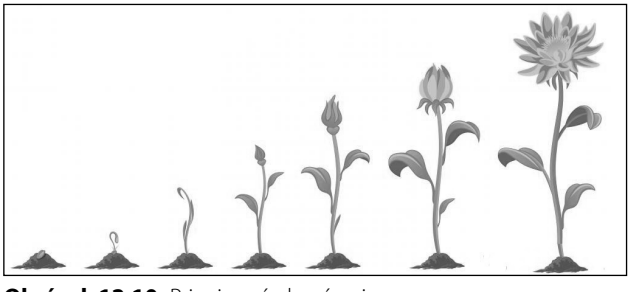
Animace v trvání dvou sekund je vytvořena a spuštěna v kódu hlavní aktivity. V metodě `onCreate` se načítá definiční soubor transformace *shape_transition.xml*.

```
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 naopak 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 12.10: 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 případě 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* ve složce *drawable* je 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();
  }
}
```



Obrázek 12.11: Test aplikace – finální fáze

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 a sérii animace ukončí zmizení objektu, tedy zvyšování průsvitnosti. Budeme měnit i tempo časového průběhu animace.

V XML kódu layoutu hlavní aktivity je jen jeden prvek `ImageView`.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/kontejner"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >

  <ImageView
    android:id="@+id/obrazek"
```

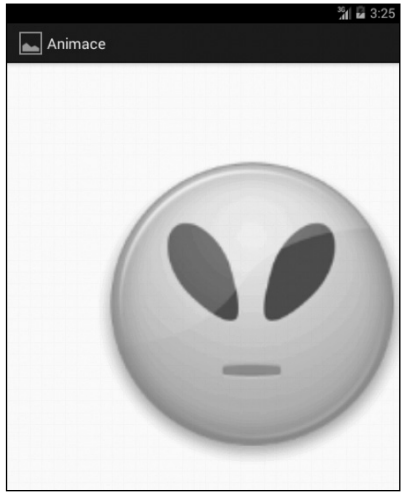
```
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:src="@drawable/et"
    android:contentDescription="@string/obrazek_popis"
    android:visibility="invisible"/>
</LinearLayout>
```

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 *animace.xml*. V krátkosti vysvětlíme jednotlivé fáze námi definované animace.

- V první sekci `<alpha>` se postupně zmenšuje transparentnost objektu z hodnoty `0,0` (úplně průsvitné) na hodnotu `1,0` (neprůsvitné). Fáze trvá 2 000 milisekund.
- V druhé sekci `<rotate>` se objekt otočí 3× dokola, tj. 3× 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.



Obrázek 12.12: Test aplikace – fáze zvětšování objektu

```
<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 spuštěná v metodě `onWindowFocusChanged()`.

```
public class MainActivity extends Activity
{
  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.animace);
  }

  @Override
  public void onWindowFocusChanged(boolean hasFocus)
  {
    super.onWindowFocusChanged(hasFocus);
    if (hasFocus) iw.startAnimation(anim);
  }
}
```

Property Animation

Dosud prezentované animace jsou velmi jednoduše realizovatelné, mají však 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. 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.

Property Animation 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 interval <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 XML kódu 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>
```

Celé dění 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 se využije `ArgbEvaluator`, jelikož se jedná o ARGB hodnotu.

```
ValueAnimator anim = ValueAnimator.ofObject(new ArgbEvaluator(),
  Color.YELLOW, Color.GREEN);
```

Evaluátor v tomto případě nemůže provést lineární mapování, protože 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.

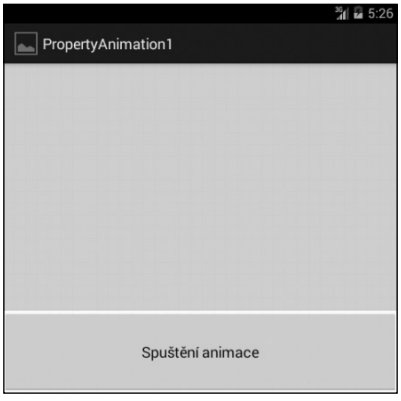
Kompletní kód hlavní aktivity:

```
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)
      {
        spustAnimaci();
      }
```

```
    });
  }

  public void spustAnimaci()
  {
    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();
  }
}
```



Obrázek 12.13: Příklad na ilustraci fungování Property Animation

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 XML kód hlavní aktivity aplikace obsahuje pouze jeden prvek typu `LinearLayout`:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/kontejner"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
  <ImageView
    android:id="@+id/obrazek"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:src="@drawable/et"
    android:alpha="0"
    android:contentDescription="@string/obrazek_desc"/>
</LinearLayout>
```

Animace se spustí automaticky v obsluze události `onWindowFocusChanged()`, kdy se spustí první úloha jako objekt `Runnable`. 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 Activity
{
  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.obrazek);
    if (hasFocus) pruhlednost.run();
  }

  Runnable pruhlednost = new Runnable()
  {
    public void run()
    {
      iv.animate().setDuration(2000)
        .setInterpolator(new LinearInterpolator()).alpha(1.0f)
        .withEndAction(rotace);
    }
  };

  Runnable rotace = 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(zvetseni);
    }
  };

  Runnable zvetseni = new Runnable()
  {
    public void run()
    {
      iv.animate().setDuration(2000)
        .setInterpolator(new AnticipateInterpolator())
        .scaleXBy(1.0f).scaleYBy(1.0f);
    }
  };
}
```

KAPITOLA 13

Multimédia

V této kapitole:
- Pasivní a aktivní konzumace multimédií
- Příklad – přehrání zvukového efektu a ovládání hlasitosti
- Příklad – přehrání videa
- Příklad – nahrávání zvuku
- Příklad – Snímání fotografie I
- Příklad – Snímání fotografie II
- Příklad – Snímání fotografie s využitím externí aplikace

Pasivní a aktivní konzumace multimédií

Moderní telefony a tablety mají kvalitní hardware nejen na zobrazování fotografií a přehrávání zvuku a videa, ale i na jejich snímání a zpracování. V přenosných zařízeních se často využívá možnost nahrávat hlasové poznámky. Multimédia 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` – třídy na ovládání zvuků vyzvánění a alarmů.
- `MediaRecorder` – umožňuje aplikacím záznam zvuku a obrazu.
- `Camera` – umožňuje aplikacím ovládání hardwaru na snímání fotografií.
- `MediaPlayer` – třída umožňuje aplikacím přehrávat audio- a videosoubory.

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 medi-

ální 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.

Příklad – přehrání zvukového efektu a ovládání hlasitosti

Zadání příkladu je kompletně obsaženo v jeho názvu. Vytvořte projekt aplikace pro Android, která bude přehrávat soubor s krátkým zvukovým efektem, v tomto případě 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"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <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 do 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 potom, 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 Activity
{
  private int hlasitost = 5, max = 10, min = 0;
  private SoundPool soundPool;
  private int id;
  private AudioManager audioManager;
  private boolean bPlay;

  @Override
  public 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 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 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.siren, 1);

soundPool.setOnLoadCompleteListener(new OnLoadCompleteListener()
{
  @Override
  public void onLoadComplete(SoundPool soundPool, int sampleId,
    int status)
  {
    if (status == ) playButton.setEnabled(true);
  }
});

playButton.setOnClickListener(new 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);

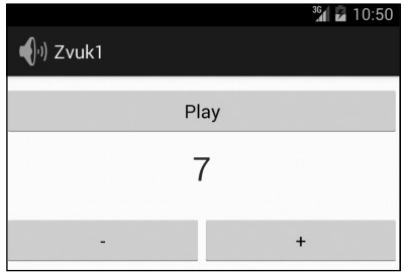
    bPlay = AudioManager.AUDIOFOCUS_REQUEST_GRANTED == result;

  }

  @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();
  }

  OnAudioFocusChangeListener afChangeListener =
    new OnAudioFocusChangeListener()
  {
    public void onAudioFocusChange(int focusChange)
    {
      if (focusChange == AudioManager.AUDIOFOCUS_LOSS)
      {
        audioManager.abandonAudioFocus(afChangeListener);
        bPlay = false;
      }
    }
  };
}
```



Obrázek 13.1: Jednoduchá aplikace na přehrání zvukového souboru a nastavování hlasitosti

Příklad – přehrání 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. 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 Activity
{
  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);

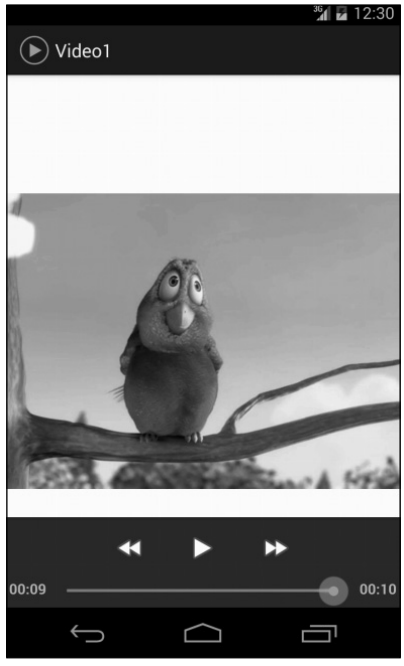
    mediaController.setEnabled(false);

    videoView.setMediaController(mediaController);

    videoView.setVideoURI(Uri.parse(
      "android.resource://com.example.video1/raw/demo"));

    videoView.setOnPreparedListener(new 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 13.2: Test aplikace

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 není moc co řešit. Uživatelské rozhraní sestává jen ze dvou přepínacích tlačítek. První aktivuje nahrávání a druhé přehrávání nahraného záznamu.

```
<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="Nahrávání STOP"
    android:textOff="Nahrávání 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="Přehrávání START"
    android:textOn="Přehrávání STOP"/>

</LinearLayout>
```

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 Activity
{
  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 OnCheckedChangeListener()
    {
      @Override
      public void onCheckedChanged(CompoundButton buttonView,
        boolean isChecked)
```

```
    {
      btPlay.setEnabled(!isChecked);
      onRecordPressed(isChecked);
    }
  });

  btPlay.setOnCheckedChangeListener(new 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) startRecording();
  else stopPlaying();
}

private void startPlaying()
{
  mPlayer = new MediaPlayer();
  try
  {
    mPlayer.setDataSource(soubor);
    mPlayer.prepare();
    mPlayer.start();
  } catch (IOException e)
  {
    Log.e("ZvukZaznam", "CHYBA MediaPlayer");
  }
}

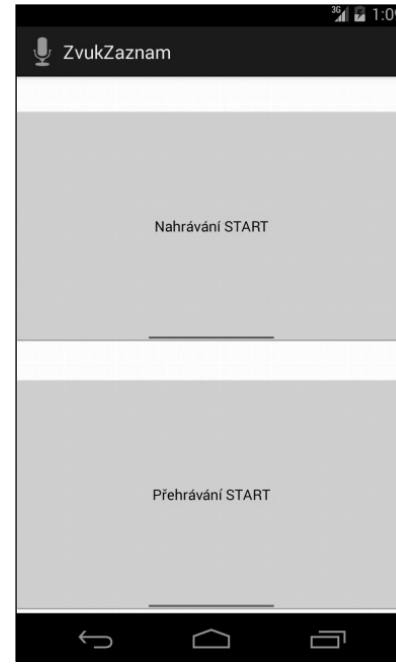
private void stopPlaying()
{
  if (null != canvas)
  {
    if (mPlayer.isPlaying()) mPlayer.stop();
    mPlayer.release();
    mPlayer = null;
  }
}


OnAudioFocusChangeListener afcListener = new 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;
    }
  }
}
```



Obrázek 13.3: Test aplikace

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.



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, jako například autofokus či blesk.

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name= "android.hardware.camera.autofocus" />
```

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 jen dvojsekundovým „zmrazením“ náhledu. Layout hlavní aktivity tvoří pouze prvek `SurfaceView`, jehož úlohou je indikovat klepnutí na displej.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/frame"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >

  <SurfaceView
    android:id="@+id/cameraView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
  </SurfaceView>
</LinearLayout>
```

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 LinearLayout 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 spuštěná. 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
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);

    //na celou 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();
    if (camera == null) finish();

    //nastavení dotyku
    layout = (LinearLayout) findViewById(R.id.frame);
    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;
      }
    });

    //nastaveni pro nahled
    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> instance 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: `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();}

    // ukládání obrázku

    // 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 – `PicrureCallback`, 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 nepřistupuje přímo 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();}
      }
  }
}
```

KAPITOLA 14

Senzory, mapové služby

V této kapitole:
- Integrované senzory chytrých telefonů a tabletů
- Lokalizační a mapové služby
- Zobrazení polohy na mapě

Integrované senzory chytrých telefonů 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 v tabletech velmi důležitou funkci. Umožňují nejen zjištění aktuální orientace přístroje (na výšku či na šířku), ale jelikož tablet nemá žádná kurzorová tlačítka ani žádný ekvivalent joysticku, využívá se naklápění přístroje velmi často i na ovládání her.

Pro ilustraci možností představíme senzorovou výbavu Samsungu Galaxy S4:

- **Proximity Sensor** (senzor přiblížení) – 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 klasický magnetický senzor.
- **Light Sensor** – měří intenzitu okolitého 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

`SensorManager` je systémová služba na řízení senzorů. Instanci objektu `SensorManager` získáte pomocí metody:

```
getSystemService(Context.SENSOR_SERVICE)
```

Na přístup ke konkrétnímu senzoru využijete metodu:

```
SensorManager.getDefaultSensor(int typ)
```

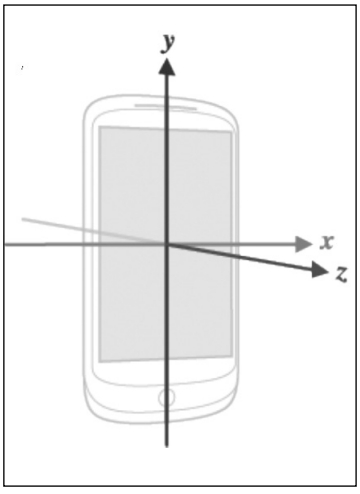
Senzor si vybírá pomocí parametru typ:

- `TYPE_ACCELEROMETER` – zrychlení v m/s² ve třech fyzických osách (x, y a z).
- `TYPE_AMBIENT_TEMPERATURE` – teplota vzduchu místnosti ve °C.
- `TYPE_GRAVITY` – gravitační zrychlení v m/s² 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/s² 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.

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()`.



Poznámka: Poloha zařízení se určuje vůči třem osám x, y a z. Mapování os na zařízení se po změně jeho polohy nemění.



Obrázek 14.1: Poloha zařízení se určuje vůči 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 na 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 zrychlení. 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>
```

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 `onAccuracy Changed()`, ignorujeme.

```
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();
```

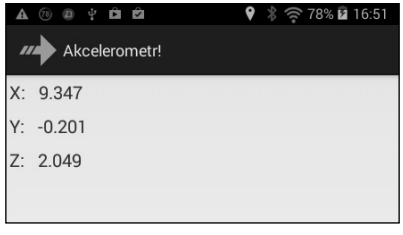
```
    sensorManager.registerListener(this, akcelerometr,
    SensorManager.SENSOR_DELAY_UI);
    posledniAktualizace = System.currentTimeMillis();
  }

  @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.



Obrázek 14.2: Zobrazení výstupů z akcelerometru



Poznámka: Pokud je zařízení v klidu, naměříte při použití senzoru `TYPE_ACCELEROMETER` v ose y hodnotu 9,81, což odpovídá gravitačnímu zrychlení.

Příklad – zobrazení filtrovaných údajů z akcelerometru

U předchozího příkladu platí modifikované úsloví, že za málo úsilí málo užitné hodnoty. Pozornější čtenáři si určitě na obrázku běhu aplikace všimli nesrovnalosti v hodnotách os X a Y. Obrázek 14.2 je totiž sejmutý z tabletu Lenovo Yoga 10 obráceného 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 je hodnota blízká 9,81 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. 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 Low Pass 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;
}
```

Zkrácený kód hlavní aktivity:

```
public class MainActivity extends Activity 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);
  ...

  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));
        ...
        tvXhi.setText(String.valueOf(aGravitace[0]));
        ...
        tvXlo.setText(String.valueOf(aAkcelerace[0]));
        ...
      }
    }
  }

  @Override
  public void onAccuracyChanged(Sensor sensor, int accuracy)
  {}
}
```

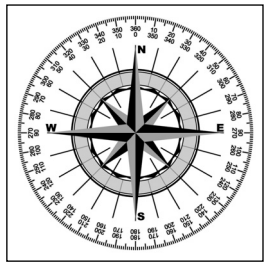
Jako ilustraci netradičního využití akcelerometru můžete vytvořit aplikaci, která promění tablet na fitness pomůcku, například na procvičení ztuhlých rukou při práci s počítačem. Stačí držet tablet v předpažených rukách displejem k sobě a pohybovat jím od sebe a k sobě, a to co nejrychleji. Na základě 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 (mikrotesla) 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 14.3: 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);
    }

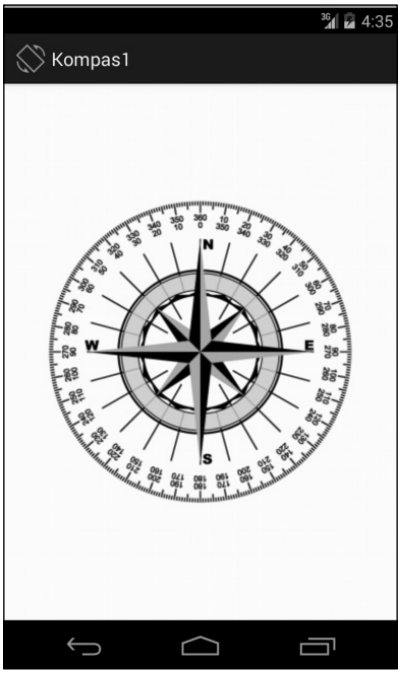
    if (zrychleni != null && magnet != null)
    {
      float rotMatrix[] = new float[9];

      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 14.4: Test aplikace

Senzor osvětlení

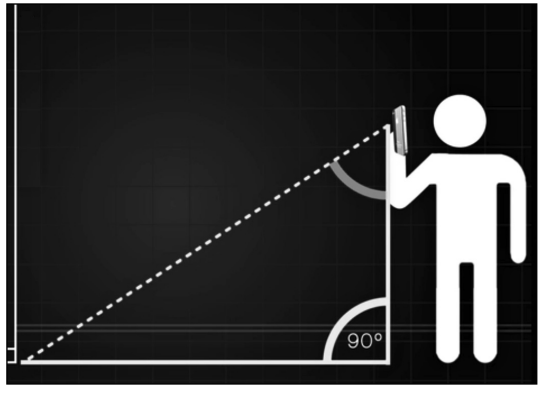
Pokud tablet disponuje senzorem měřícím intenzitu osvětlení, který implicitně využívá na automatickou regulaci jasu displeje, dá se využít i v aplikacích. Například může reagovat na to, že uživatel vyšel z místnosti ven na sluneční světlo (nebo opačně) a přizpůsobit tomu barevný motiv aplikace, případně vhodně zvýšit jas nebo kontrast mezi pozadím a písmem, podobně jako to dělají třeba GPS navigace 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 využití náklonoměru je 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 dis-

ponuje 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 skloníte zařízení tak, abyste záměrným bodem mířili na spodní okraj objektu, a provedete 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 14.5: Princip měření vzdálenosti tabletem

Rozměry menších předmětů můžete změřit tak, že je položíte na displej tabletu a pomocí kurzoru vhodné aplikace 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, jaký 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 – a aplikace snímá a vyhodnocuje změnu průsvitnosti vašeho prstu pro vnější světlo.

Lokalizační a mapové služby

Přístroje platformy Android, hlavně telefony, disponují obvody pro příjem signálu GPS z družic, takže po spojení 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 novu dimenzi, jelikož aplikace může poskytovat uživateli informace vzhledem k jeho momentální poloze.

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, 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 z GPS signálu, ale i triangulací GSM stanic či z Wi-Fi přístupových bodů.

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 callback metody, 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 ke 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 GPS polohy. 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.

Kód hlavní aktivity:

```
public class MainActivity extends Activity implements LocationListener
{
  private TextView latitute;
  private TextView longitude;
  private LocationManager locationManager;
  private String provider;


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

    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);
    Location location = locationManager.getLastKnownLocation(provider);

    if (location == null)
    {
      latitute.setText("---");
      longitude.setText("---");
    }
```

```
    else  onLocationChanged(location);
  }

  @Override
  protected void onResume()
  {
    super.onResume();
    locationManager.requestLocationUpdates(provider, 3000, 1, this);
  }

  @Override
  protected void onPause()
  {
    super.onPause();
    locationManager.removeUpdates(this);
  }

  @Override
  public void onLocationChanged(Location location)
  {
    double lat = location.getLatitude();
    double lng = location.getLongitude();
    latitute.setText("Zeměpisná šířka: " + String.valueOf(lat));
    longitude.setText("Zeměpisná délka" + String.valueOf(lng));
  }

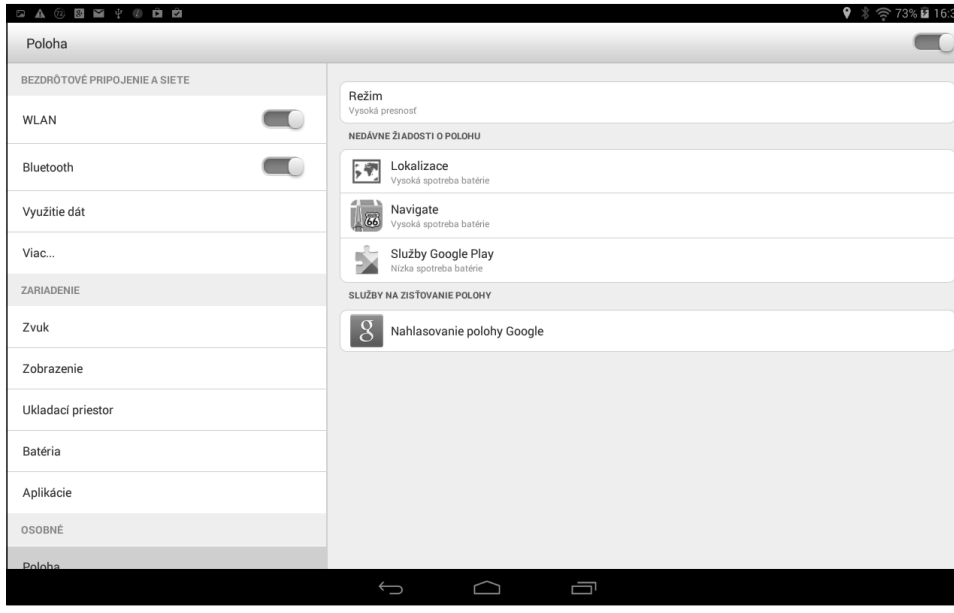
  @Override
  public void onStatusChanged(String provider, int status, Bundle extras) {}

  @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();
  }
}
```

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 z GPS signálu 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 v sekci **Nedávné žádosti o určení polohy** charakterizovaná jako aplikace s vysokou spotřebou energie z baterie. Všimněte si, že stejně dopadla i navigační aplikace *Navigate Route 66*, ta však nemá na výběr. Při navigaci během cesty autem je potřeba zjišťovat polohu z GPS neustále.



Obrázek 14.6: Informace o spotřebě baterie při používání GPS

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.

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 tato 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.

```
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)
  {
    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, tedy 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);
        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)
  {
    locManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
      POLLING, MIN_VZDALENOST, locListener);

    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
      POLLING, MIN_VZDALENOST, locListener);

    // naplánuj odregistrování listenerů po 30 sekundách
    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. šířka:" + 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 v2 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 14.7: Příklad zobrazení Normal, Satellite 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, a realizovat pootočení na gesto pootočení dvou prstů. API také umožňuje zobrazit aktuální polohu zařízení.

Google Maps Android v2 API využívá základní třídy.

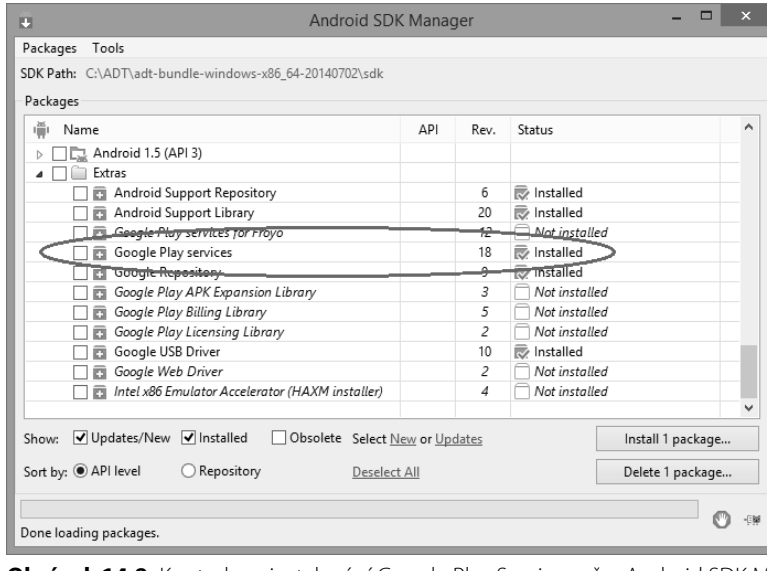
- `GoogleMap` – třída, která zastupuje a spravuje mapu
- `MapFragment` – zobrazení mapy ve fragmentech
- `Camera` – definuje část mapy, která se zobrazí na obrazovce, a bod, z kterého by tento pohled snímala fiktivní kamera
- `Marker` – třída reprezentuje ikony, někdy i vyskakovací okna, které ukazují 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 API klíč, který identifikuje vaši aplikaci. Parametry pro nastavení je potřeba vložit do manifestu aplikace.

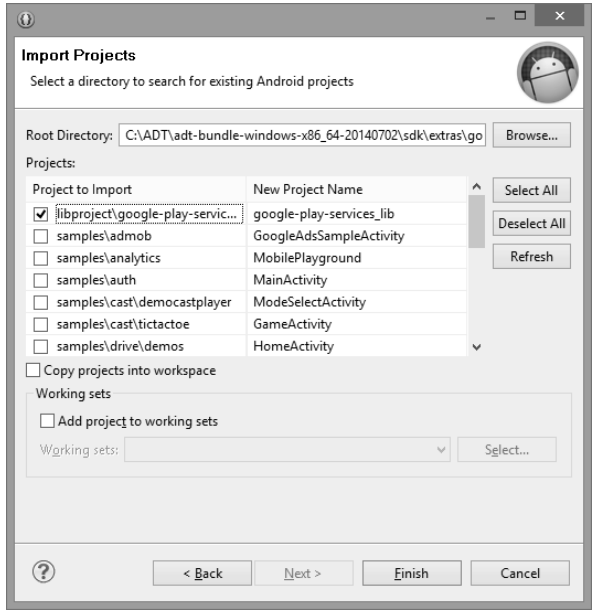
Instalace Google Play Services

Pokud nemáte nainstalované Google Play Services, je potřeba doinstalovat toto SDK přes Android SDK Manager.



Obrázek 14.8: Kontrola nainstalování Google Play Services přes Android SDK Manager

Pomocí nabídky **File** → **Import** → **Android** → **Existing Android Code** je potřeba importovat knihovnu Google Play Services do workspace, ve kterém vytváříte aplikaci využívající mapové služby.



Obrázek 14.9: Import Google Play Services do workspace

Do projektu je potřeba přidat referenci na Google Play Services. V okně **Package Explorer** vyberte v místní nabídce položku **Properties**. Ve složce **Android** přidejte referenci.

Referenci na Google Play Services je potřeba přidat do aplikačního manifestu:

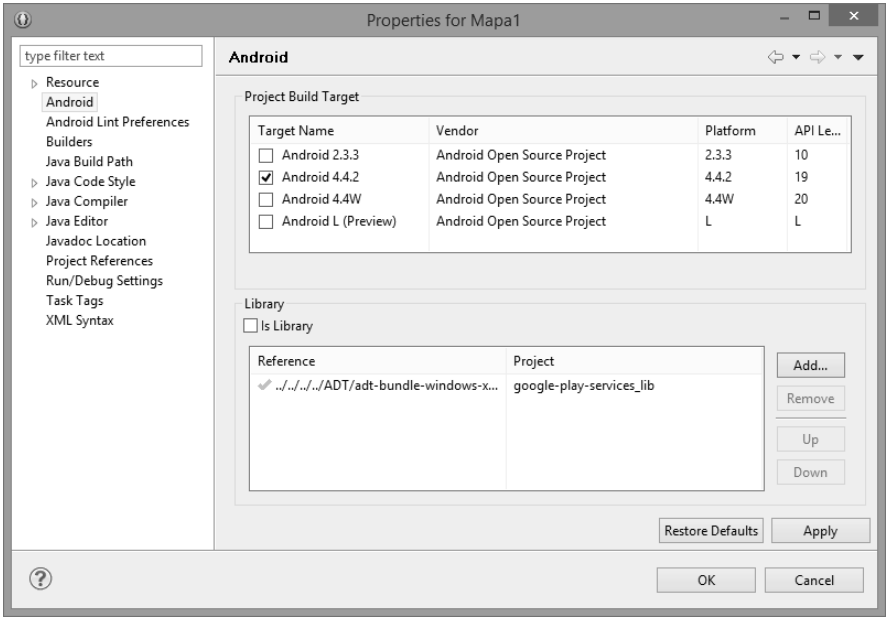
```
<meta-data
  android:name="com.google.android.gms.version"
  android:value="@integer/google_play_services_version" />
```

Vytvoření Google Maps API klíče

Pokud chcete v aplikaci využívat službu Google Maps, musíte si vytvořit platný Google Maps API klíč. 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.

K získání klíče je potřeba se přihlásit na Google API konzoli. Budete muset poskytnout podpisový klíč aplikace a název balíku.

Ladicí klíč vygenerovaný vývojovým prostředím Eclipse 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*.



Obrázek 14.10: Přidání reference

```
keytool -list -v -alias androiddebugkey -keystore
  <složka>debug.keystore -storepass android -keypass android
```

Aplikaci *keytool.exe* najdete v adresáři, do něhož jste nainstalovali Javu. V našem případě to byl adresář *C:\Program Files\Java\jre7\bin* a příkaz měl poté tvar:

```
keytool -list -v -alias androiddebugkey -keystore
  C:\Users\Luboslav\.android\debug.keystore
  -storepass android -keypass android
```

Potřebujete otisk SHA1, proto si ho zkopírujte z výstupu konzolové aplikace. V našem případě se jedná o řetězec:

2A:14:6B:BC:8F:F7:FA:16:55:79:11:DC:8F:82:FE:DB:8E:02:E7:15

Následně se přihlaste přes Google APIs Console na adrese *https://code.google.com/apis/console/*. Použijte svoje přístupové údaje do vývojářského účtu Google. Ve složce *Services* vyberte Google Maps Android API v2.



Obrázek 14.11: Vygenerování digitálního otisku klíče pomocí aplikace keytool.exe



Obrázek 14.12: Aktivování Google Maps Android API v2 přes Google APIs Console

Následně vytvořte klíč pro vaši aplikaci. Jako vstup zadejte SHA1 otisk vašeho klíče. Ve složce **API Access** klepněte na tlačítko **Crete new Android key**.

Zadejte řetězec obsahující SHA1 otisk vašeho klíče, středník a název balíčku vaší aplikace.

V našem případě jsme zadali řetězec:

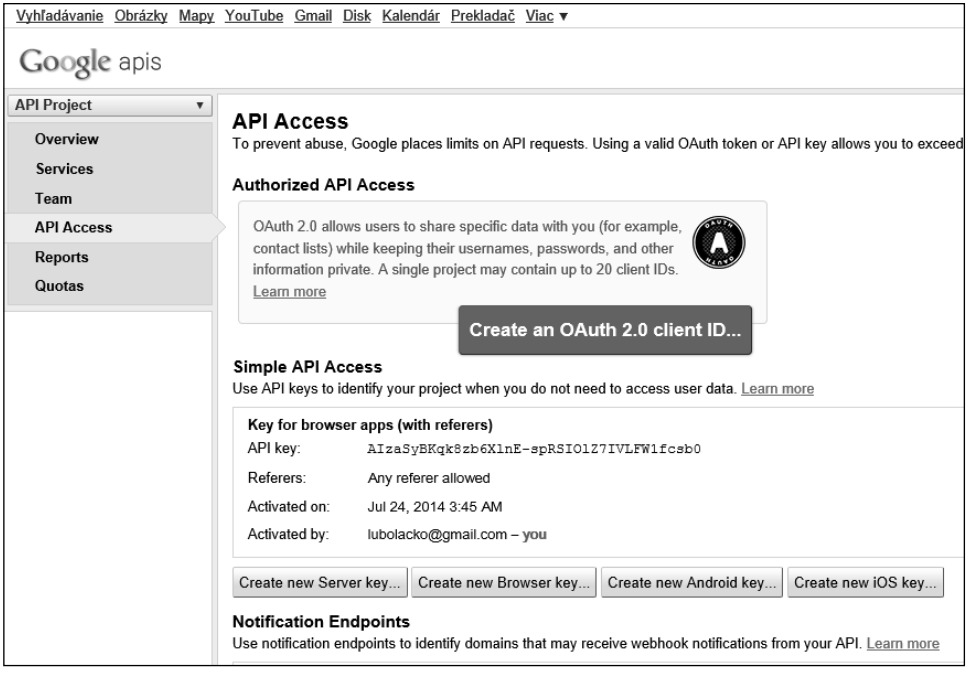
2A:14:6B:BC:8F:F7:FA:16:55:79:11:DC:8F:82:FE:DB:8E:02:E7:15;com.lubolacko.mapa1

Webová aplikace vám vrátí klíč ve tvaru:

AIzaSyAg3L82cOeUqogl7hTiuVDrJd5YqK14PfM

Klíč potom vložíte do aplikačního manifestu aplikace před koncový tag `</application>`.

```
<meta-data
  android:name="com.google.android.maps.v2.API_KEY"
  android:value="AIzaSyAg3L82cOeUqogl7hTiuVDrJd5YqK14PfM" />
```



Obrázek 14.13: Vytvoření klíče pro aplikaci

Příklad zobrazení polohy na mapě

Aplikace využívající mapové služby vyžaduje v manifestu několik typů oprávnění. Povolení přístupu k Internetu nevyžaduje komentář, aplikace přistupuje na servery Google Map. Oprávnění `ACCESS_NETWORK_STATE` je potřeba na zjištění, zda je možné ze serveru stahovat údaje. Budete nutně potřebovat i oprávnění `WRITE_EXTERNAL_STORAGE`, jelikož mapové údaje musí být uloženy do externího úložiště přístroje.

Oprávnění `READ_GSERVICES` umožní přístup ke Google Web Services. Prakticky všechny aplikace pracující s mapou potřebují i údaj o okamžité poloze zařízení, takže budete potřebovat i oprávnění `ACCESS_COARSE_LOCATION` a `ACCESS_FINE_LOCATION`.

Jelikož v aplikacích využívajících mapové služby hodně záleží na aplikačním manifestu, jednak kvůli povolení a také klíči pro identifikaci aplikace vůči službě, začneme příklad výpisem souboru *AndroidManixfest.xml*.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.lubolacko.mapa1"
  android:versionCode="1"
  android:versionName="1.0" >

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

  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name=
    "android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name=
    "com.google.android.providers.gsf.permission.READ_GSERVICES" />
  <uses-permission android:name=
    "android.permission.ACCESS_COARSE_LOCATION" />
  <uses-permission android:name=
    "android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name=
    "android.permission.ACCESS_NETWORK_STATE" />

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <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>

    <meta-data
      android:name="com.google.android.gms.version"
      android:value="@integer/google_play_services_version" />

    <meta-data
      android:name="com.google.android.maps.v2.API_KEY"
      android:value="AIzaSyAg3L82cOeUqogl7hTiuVDrJd5YqK14PfM" />

  </application>

</manifest>
```

Layout hlavní aktivity obsahuje fragment pro zobrazení mapového podkladu ze služby Google Maps.

```
<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=".MainActivity" >

  <fragment
    android:id="@+id/mapa"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.MapFragment" />

</RelativeLayout>
```

V kódu hlavní aktivity jsme přidali i importy, jelikož importy knihoven se nedají přidat přes klávesovou zkratku **Ctrl+Shift+O**. Zobrazíme mapu a označíme na ní dva markery, přičemž v druhém z nich nastavíme i vyskakovací okno s textem a jako obrázek markeru použijeme ikonu aplikace.

```
import android.app.Activity;
import android.os.Bundle;

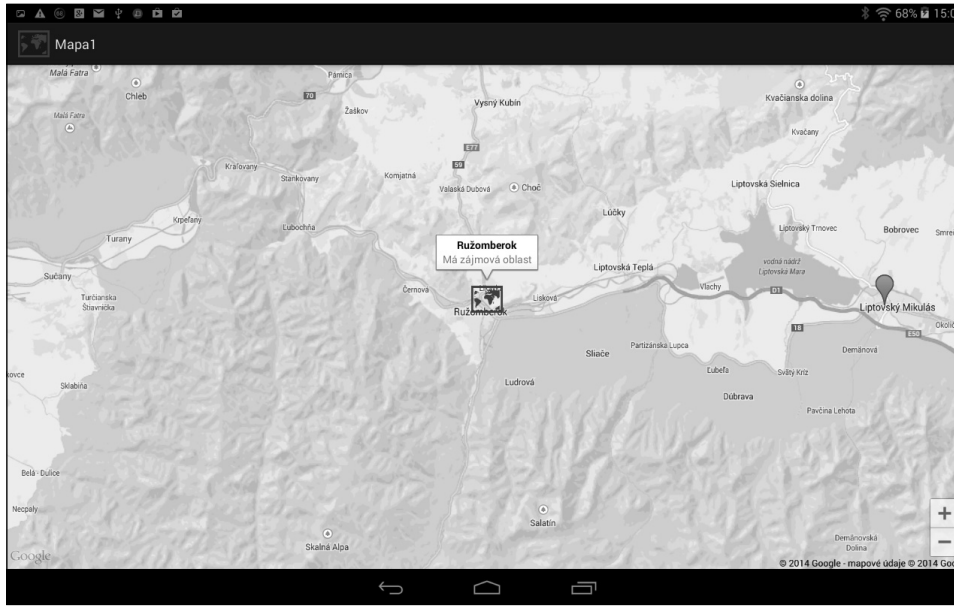
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.MapFragment;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;


public class MainActivity extends Activity
{
  static final LatLng LIPTOVSKY_MIKULAS = new LatLng(49.084167, 19.602222);
  static final LatLng RUZOMBEROK = new LatLng(49.078611, 19.308333);
  private GoogleMap map;

  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    map = ((MapFragment) getFragmentManager().findFragmentById(R.id.mapa))
      .getMap();
    Marker lm =
      map.addMarker(new MarkerOptions().position(LIPTOVSKY_MIKULAS)
      .title("Lipt. Mikuláš"));
    Marker rbk = map.addMarker(new MarkerOptions()
      .position(RUZOMBEROK)
      .title("Ružomberok")
      .snippet("Má zájmová oblast")
      .icon(BitmapDescriptorFactory
      .fromResource(R.drawable.ic_launcher)));

    map.moveCamera(
      CameraUpdateFactory.newLatLngZoom(LIPTOVSKY_MIKULAS, 15));
```

```
    map.animateCamera(CameraUpdateFactory.zoomTo(10), 2000, null);
  }
}
```



Obrázek 14.14: 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 ten 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.");
  }

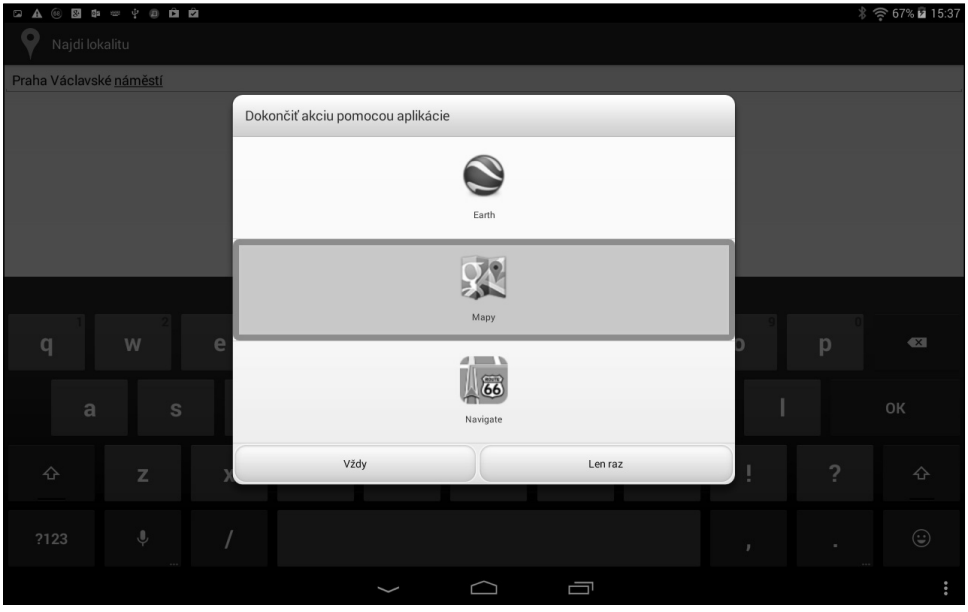
  @Override
  protected void onResume()
  {
    super.onResume();
    Log.i(TAG, "Aktivita má fokus a je v stave Resume");
  }

  @Override
  protected void onPause()
  {
    super.onPause();
    Log.i(TAG, "Jiná aktivita získala fokus");
  }

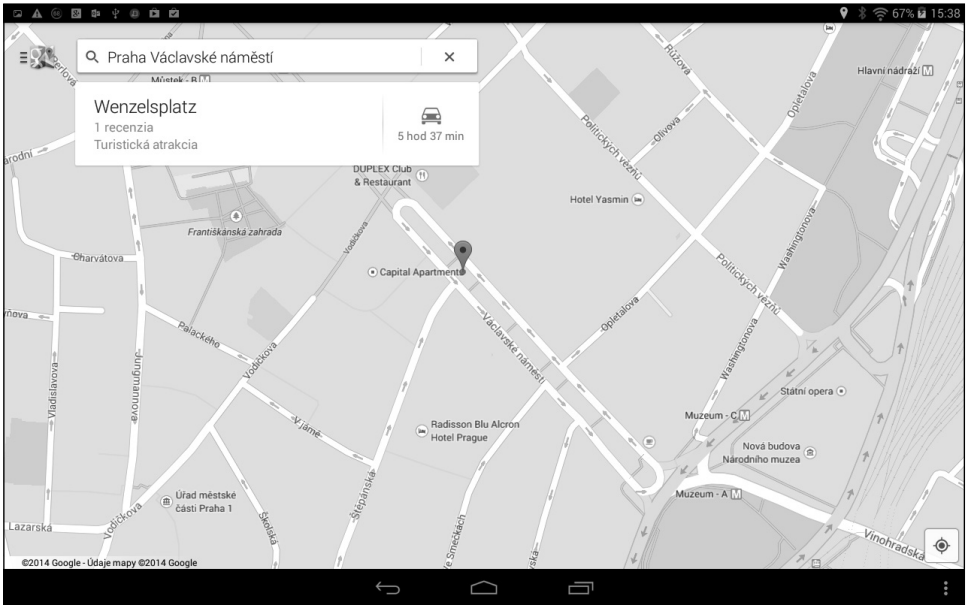
  @Override
  protected void onStop()
  {
    super.onStop();
    Log.i(TAG, "Aktivita zastavená");
  }

  @Override
  protected void onDestroy()
  {
    super.onDestroy();
    YLog.i(TAG, "Aktivita odstránená.");
  }
}
```

Po zadání názvu nebo poštovní adresy lokality se zobrazí nabídky aplikací, které jsou schopné zobrazit polohu lokality na mapě.



Obrázek 14.15: Zadání adresy a výběr aplikace



Obrázek 14.16: Zobrazení polohy na mapě ve vybrané aplikaci

KAPITOLA 15

Služby a broadcasty

V této kapitole:
- Služba a její životní cyklus

Tuto kapitolu začneme trochu netradičně – popisem scénářů z běžného života, ve kterých hrají mobilní zařízení klíčovou roli.

Pacient nebo senior s občasnými zdravotnými obtížemi je vybaven snímačem potřebných fyziologických parametrů s Bluetooth komunikací, modulem GPS a samozřejmě telefonem. Lékařský snímač zaznamenává změny fyziologických parametrů indikujících nastupující problémy. GPS modul určí nejen místo, na kterém k tomu došlo. Telefon v případě potřeby přivolá pomoc.

Diabetik využívá digitální glukometr, který je vybaven komunikačním rozhraním Bluetooth. Naměřené hodnoty se přenesou do chytrého telefonu a následně na server příslušné služby. Tyto údaje si potom může zobrazit nejen jeho praktický lékař či diabetolog, který v případě potřeby může změnit předpis dávkování inzulínu, ale z domácího počítače i sám pacient, aby případně korigoval svoje dietetické návyky.

Nevidomý je navigován na dálku příbuzným nebo dobrovolníkem přes web nebo mobil. Příkazy jsou transformovány do vibrací v různých směrech, takže sluch zůstává zachován jako nejdůležitější smysl nevidomých. Signály z mobilních zařízení jsou odesílány přes Bluetooth do vibračního pásu, takže navigovaný směr může být vyjádřen jako vibrace z daného směru. Senzory vzdálenosti automaticky generují varování také ve formě vibrace.

V dosud prezentovaných tématech jsme se zaměřovali většinou na činnost, jejíž 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.

Služby (Services) se používají na pravidelné a nepřetržitě vykonávané akce, které nevyžadují uživatelské rozhraní. Typickým příkladem je stahování souborů či RSS informací z Internetu, 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.



Poznámka: Služby je nutné registrovat v *AndroidManifest.xml*.

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, v kterém může služba realizovat svou 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 opět 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.



Poznámka: Podobně jako aktivity je potřeba i služby definovat v souboru *AndroidManifest.xml*.

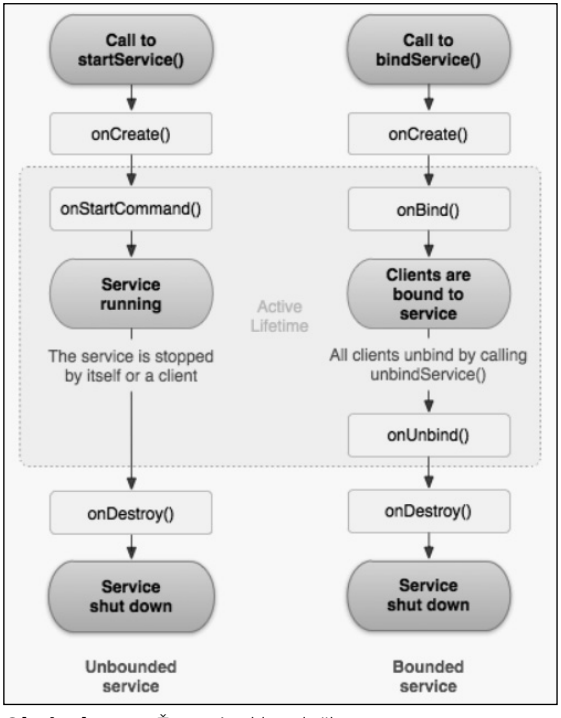
Služby mají dva základní stavy, přičemž služba se může nacházet i v obou zároveň.

- **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 všechno, co bylo zapotřebí.
- **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.

Metody navázané na stavy životního cyklu služby názorně vysvětluje obrázek.

- `onCreate()` – metoda je volaná při prvním vytvoření služby. Když už služba běží, volání metody se nerealizuje.
- `onBind()` – metoda se volá v případě, že některá aktivita nebo komponenta požaduje spuštění služby voláním metody `startService()`.

- `onUnbind()` – metoda je volaná, 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.



Obrázek 15.1: Životní cyklus služby

Metoda `onStartCommand()` 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`.

Na spuštění služby z aktivity je určena metoda `startService()`, která jako parametr dostane záměry. Volání je asynchronní. Nejprve je volána metoda `OnCreate()`. Následně je záměr odevzdán do metody `onStartCommand()`. Po spuštění služba běží, dokud není explicitně zastavena.

Na zpřístupnění API služby ostatním komponentám (klientům) je určena metoda `bind Service()`. 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, pokud 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`, je vytvořena nová služba, v opačném případě je vrácena hodnota `false`.

Příklad – přehrávání hudby na pozadí

Námětem příkladu je vytvoření aplikace, která bude přehrávat MP3 soubor. Soubor bude de facto přehrávat služba spuštěná aplikací, takže hudba bude hrát dál, i když uživatel činnost aplikace na popředí, která službu spustila, pozastaví nebo skončí.

Pro jednoduchost bude hudba součástí projektu aplikace. Po vytvoření projektu vytvořte ve složce *res* vnořenou složku *raw* a do ní umístěte MP3 soubor, 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"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">

  <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 dekladrovat v *AndroidManifest.xml*:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical">
```

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 Foreground služba. V principu je to obyčejná služba, ale s vysokou prioritou, takovou, která odpovídá zobrazené aktivitě. Foreground službu spustíte voláním metody `startForeground` v metodě `OnCreate`.



Poznámka: Nutná podmínka pro Foreground službu 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 Activity
{
  @Override
  public 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 OnClickListener()
    {
      public void onClick(View src)
      {
        startService(hudbaIntent);
      }
    });

    final Button stopButton = (Button) findViewById(R.id.stop_button);
    stopButton.setOnClickListener(new OnClickListener()
    {
      public void onClick(View src)
      {
        stopService(hudbaIntent);
      }
```

```
    });
  }
}
```

Kód služby v souboru *PrehravacService.java* přehraje MP3 soubor. 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 na notifikační centrum, odkud i po ukončení aplikace spouštějící službu lze tuto aplikaci znovu spustit a přehrávání ukončit.

```
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.beatles);
    if (mediaPlayer != null)
    {
      mediaPlayer.setLooping(false);
      mediaPlayer.setOnCompletionListener(new 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);
  }

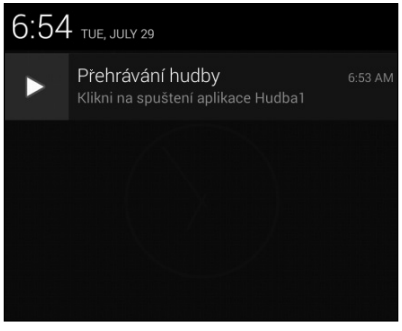
  @Override
  public int onStartCommand(Intent intent, int flags, int startid)
  {
```

```
    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;}

}
```



Obrázek 15.2: Po ukončení aplikace, která službu spustila, se k ní můžete vrátit přes notifikační centrum

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 systém potřebuje 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 komponentách. 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.

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 AIDL (Android Interface Definition Language) interface

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 16

Poskytování obsahu

V této kapitole:
- Třída ContentProvider
- Příklad – přístup ke kontaktům v zařízení
- Příklad – přístup ke kontaktům, načítání údajů na pozadí
- Příklad – aktualizace údajů
- Příklad – vytvoření aplikace, která bude poskytovat údaje

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.

Aplikace, které chtějí získat přístup ke sdíleným údajům, využívají třídu `ContentResolver`, která umožňuje aplikacím číst a zapisovat data, která jsou uložena v objektech `ContentProvider`. Podporuje metody `query`, `insert`, `update`, `delete` a také další služby, například informování registrovaných pozorovatelů o změnách.

Údaje mohou být fyzicky uloženy v SQLite databázi, v souborech, v cloudu a podobně, závisí to na konkrétní implementaci.



Poznámka: `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 URI řetězce 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 má tvar:

```
ContactsContract.Contacts.CONTENT_URI =
  "content://com.android.contacts/contacts/"
```

V tomto případě se jedná o URI na 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
- `getType` – vrácení MIME dat pro konkrétní URI. MIME formát je definován jako `vnd.X.cursor.dir/Y` nebo `vnd.X.cursor.item/Y`
- `query` – výběr požadovaných údajů
- `insert` – přidání údajů
- `update` – aktualizace údajů
- `delete` – smazání údajů

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`.

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.

Aplikace bude potřebovat v souboru *AndroidManifest.xml* deklarovat povolení přístupu k údajům v databázi kontaktů.

```
<uses-permission android:name="android.permission.READ_CONTACTS" />
```

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 pr-

vek `LinearLayout` s horizontální orientací uspořádání prvků. Návrh layoutu položky seznamu je v souboru *kontakt.xml*. Pro jednoduchost využijeme jako zástupný obrázek kontaktu ikonu aplikace.

```
<?xml version="1.0" encoding="utf-8"?>
<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="@string/ hello_world"
    android:background="@drawable/ic_launcher" >
  </ImageView>

  <TextView
    android:id="@+id/jmeno"
    android:layout_width="wrap_content"
    android:layout_height="60dp"
    android:textSize="30sp" >
  </TextView>

</LinearLayout>
```

V kódu hlavní aktivity v proměnné `atributy` definujeme dva atributy třídy `ContactsContract.ContactsColumns`: jméno kontaktu a URI adresu 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
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    String atributy[] = new String[] { Contacts._ID,
      Contacts.DISPLAY_NAME, Contacts.PHOTO_THUMBNAIL_URI};

    ContentResolver contentResolver = getContentResolver();
    String where = "((" + Contacts.DISPLAY_NAME + " NOTNULL) AND ("
      + Contacts.DISPLAY_NAME + " != '' ))";
    String sort = Contacts._ID + " ASC";
    Cursor cursor = contentResolver.query(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 definovaný předvolený obrázek pro kontakty, které nemají fotografii. Abychom nemuseli vkládat do aplikace další obrázek, například ikony postavičky, použili jsme ikonu aplikace. V metodě `newView()` vytvoříme přes `LayoutInflater` pohled s využitím layoutu *kontakt.xml*, ve kterém je definovaný 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.ic_launcher);
  }

  @Override
  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);
  }

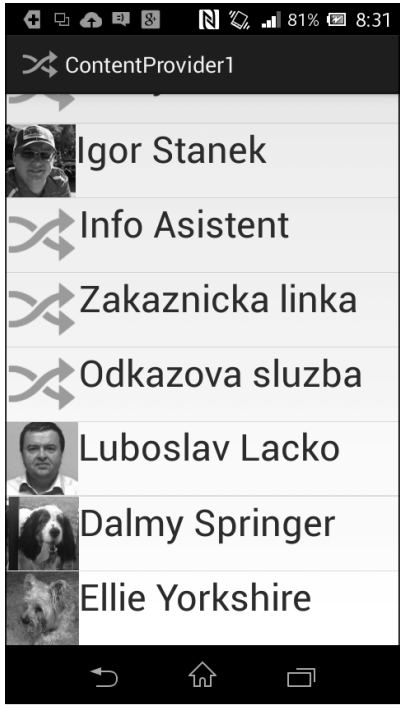
  @Override
  public void bindView(View view, Context context, Cursor cursor)
  {
    TextView textView = (TextView) view.findViewById(R.id.jmeno);
    textView.setText(cursor.getString(cursor
      .getColumnIndex(Contacts.DISPLAY_NAME)));

    ImageView imageView = (ImageView) view.findViewById(R.id.foto);
    Bitmap bmp = fakeBmp;

    String obrUri = cursor.getString(cursor
    .getColumnIndex(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);
  }
}
```



Obrázek 16.1: Zobrazení seznamu kontaktů

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 zhoršuje 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 objektu `ContentProvider` 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, během 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 nejen oprávnění 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/jmeno"
  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 `pridejZaznamDoDavky()`. Nejprve definujeme atributy, které pro nové záznamy chceme vložit, a následně hodnoty pro konkrétní záznam.

```
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[]
    {Contacts._ID, Contacts.DISPLAY_NAME};
  private static final String atributy1[] = new String[]
    { Contacts.DISPLAY_NAME };
  private static final int[] idRes = new int[] { R.id.jmeno };

  private Account[] konto;
  private String typ;
  private String jmeno;
  private SimpleCursorAdapter cursorAdapter;

  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    konto = AccountManager.get(this).getAccountsByType("com.google");
    typ = konto[0].type;
    jmeno = konto[0].name;
    vlozKontakty();

    cursorAdapter = new SimpleCursorAdapter(this, R.layout.kontakt, 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(RawContacts.CONTENT_URI)
      .withValue(RawContacts.ACCOUNT_TYPE, typ)
      .withValue(RawContacts.ACCOUNT_NAME, jmeno)
      .withValue(Contacts.STARRED, 1).build());

    ops.add(ContentProviderOperation.newInsert(Data.CONTENT_URI)
      .withValueBackReference(Data.RAW_CONTACT_ID, position)
      .withValue(Data.MIMETYPE, StructuredName.CONTENT_ITEM_TYPE)
      .withValue(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 ("
      + Contacts.DISPLAY_NAME + " != '' ) AND (" + Contacts.STARRED
      + "== 1))";

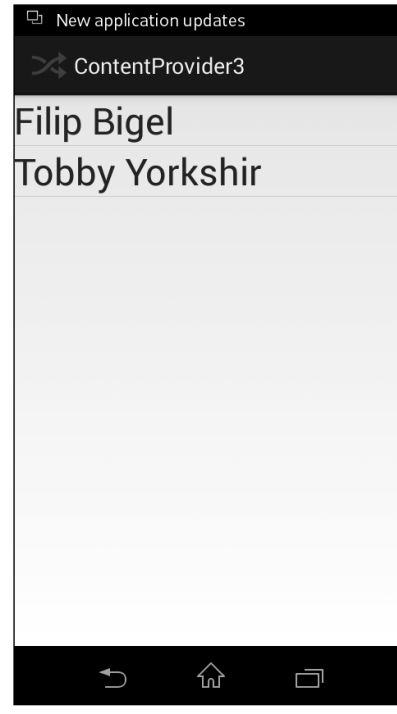
    return new CursorLoader(this, Contacts.CONTENT_URI, atributy,
      select, null, Contacts._ID + " ASC");
  }

  public void onLoadFinished(Loader<Cursor> loader, Cursor data)
  {
    cursorAdapter.swapCursor(data);
  }

  public void onLoaderReset(Loader<Cursor> loader)
  {
    cursorAdapter.swapCursor(null);
  }
}
```



Obrázek 16.2: Spuštění aplikace



Obrázek 16.3: Přidané kontakty se zobrazí i v aplikaci Lidé (Kontakty apod.), která je součástí operačního systému

Takto jednoduchá aplikace, která vloží kontakty z konstantních řetězců v metodě `onCreate`, je samozřejmě v praxi nepoužitelná. Stačí změnit orientaci zařízení a 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. 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 implementované 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);
  }
}
```

KAPITOLA 17

Připojení ke cloudovým službám a sociálním sítím

V této kapitole:

- Trendy a doporučení
- Google Cloud Messaging for Android
- Implementace na straně serveru
- Vytvoření aplikace pro Android vysílající a přijímající notifikace
- Připojení aplikace pro Android k mobilní službě
- Vytvoření aplikace pro Android využívající mobilní službu
- Úprava existující aplikace pro Android, aby mohla využívat mobilní službu
- Připojení aplikace k sociálním sítím
- Vytvoření aplikace pracující s Facebookem

Z hlediska principu fungování můžeme mobilní aplikace rozdělit na dvě základní skupiny.

- Aplikace fungující autonomně – aplikace se nepotřebují připojovat k externím zdrojům údajů, službám ani serverovým aplikacím.
- Aplikace využívající cloudové či mobilní služby, externí databáze, či připojující se k middlewarové aplikační vrstvě.



Poznámka: 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 neúspěšnějších 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 Google Cloud Messaging (GCM) ukážeme, jak posílat notifikaci (oznámení) z aplikačního serveru do aplikace na Androidu. V druhé cvičné aplikaci ukážeme připojení na mobilní službu s databází a odesílání a přijímání údajů z této služby. Takto je možné například implementovat už zmíněný žebříček top skóre nejúspěšnějších hráčů.

Trendy a doporučení

Jelikož v této kapitole překročíme pomyslnou hranici dělící hobby a profesionální aplikace určené do podnikové sféry, připomeneme doporučení nezávislé analytické společnosti Gartner ohledně vývoje mobilních aplikací. I když se doporučení týkají aplikací směřujících do podnikové sféry, určitě neuškodí jejich aspoň částečné akceptování pro jakékoliv aplikace. Jedním ze specifik mobilních aplikací je nutnost rychlého vývojového cyklu a rychlých iterací, aby mobilní aplikace dokázaly reagovat na inovace platforem, na které je aplikace portovaná.

- Je potřeba vytvářet aplikace napojené na jádro middlewaru, izolované od prezentační vrstvy obchodní logiky, jelikož musí podporovat více typů zařízení s různými prezentačními možnostmi.
- Zjednodušte proces změn a oprav, jelikož proces distribuce přes aplikační obchody je z důvodu testování a schvalování aplikace relativně nepružný.
- Určete, na které platformy a operační systémy budete aplikace portovat. Výběr závisí na vlastněných a používaných zaměstnaneckých zařízeních a zařízeních podporovaných v rámci BYOD. Některé platformy, typicky Android, jsou značně fragmentované a i pro tu samou verzi operačního systému jsou k dispozici stovky různých typů zařízení s různými displeji a možnostmi.
- Důležité je i definovat middleware vrstvy a služby na podporu fungování v mobilních aplikacích. Otázkou je, zda nasadit nové služby on premise nebo do cloudu, případně zda do cloudu migrovat existující služby.
- Samostatným atributem je bezpečnost při přístupu k podnikovým datům a back-office aplikacím.

Jsou oblasti, například CAD, DTP a podobně, ve kterých mobilní zařízení hrají „druhé housle" (pokud vůbec hraje). Naproti tomu klientské aplikace pro firemní aplikace, například CRM na mobilních zařízeních, svou operativností umožňují často vyšší efektivitu než aplikace na klasických desktopech.

Vzhledem ke specifikům uživatelského rozhraní mobilních platforem není možné mechanicky portovat klasické či webové aplikace do mobilního multidotykového prostředí. Limitujícím faktorem není ani tak rozlišení displeje, které je hlavně u tabletů stejné jako u notebooků, ale spíše možnosti ovládání, především jeho přesnost, a komfort při psaní delších textů. Proto musí být aplikace cílena na určitou vymezenou funkcionalitu.

Jiná je i filozofie použití, hlavně na chytrých telefonech. Uživatelé mají tendenci komunikovat s aplikací pouze velmi krátký čas, typicky minutu, nanejvýš dvě. Pokud aplikace v tomto čase neposkytne přidanou hodnotu v podobě požadovaných informací, uživatel ji ukončí a vrátí telefon do kapsy.

Google Cloud Messaging for Android

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), nebo to může být zpráva obsahující až 4 kB údajů, takže například chatovací aplikace mohou zprávu využívat přímo.



Poznámka: Služba GCM nahrazuje beta verzi služby C2DM (Android Cloud to Device Messaging). Google službu C2DM považuje za zastaralou a nové registrace do služby C2DM už nepřijímá. Vývojáři, kteří nepřejdou na službu GCM, mohou i nadále používat službu C2DM, ale bez zvyšování kvót.

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.

Získání klíče Simple API Access

Pokud chcete používat službu GCM, budete muset získat dlouhodobý token (jednoduchý klíč API). Pokud jste už uživatelem služby C2DM, můžete namísto toho použít svůj token služby C2DM. Pokud chcete získat jednoduchý klíč API, postupujte následovně.

V hlavní konzoli rozhraní Google API na adrese *https://code.google.com/apis/console/*, případně v inovovaném uživatelském rozhraní na adrese *http://cloud.google.com/console*, vyberte možnost **API Access**.

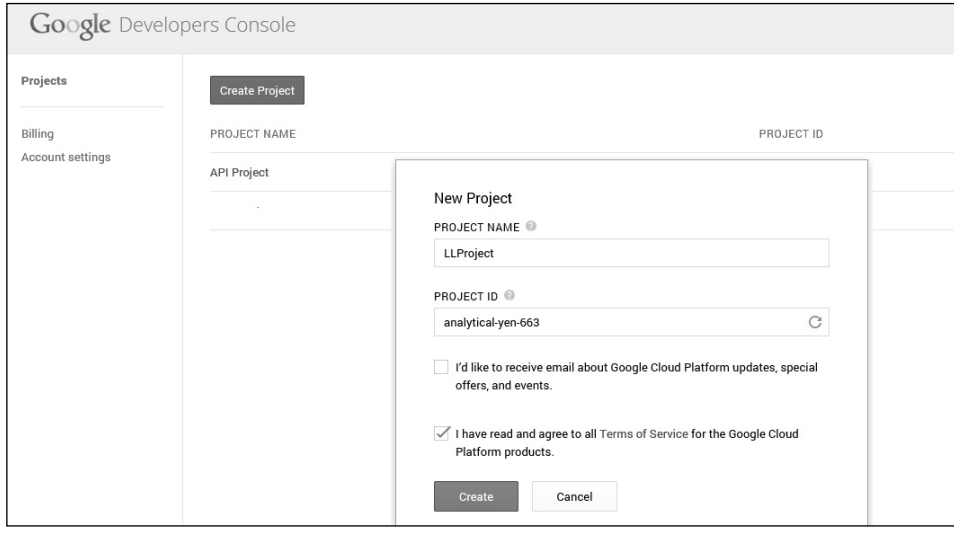


Obrázek 17.1: Konzole Google apis

Klepněte na možnost Simple API Access.

Vytvoření projektu na Google Cloud Console

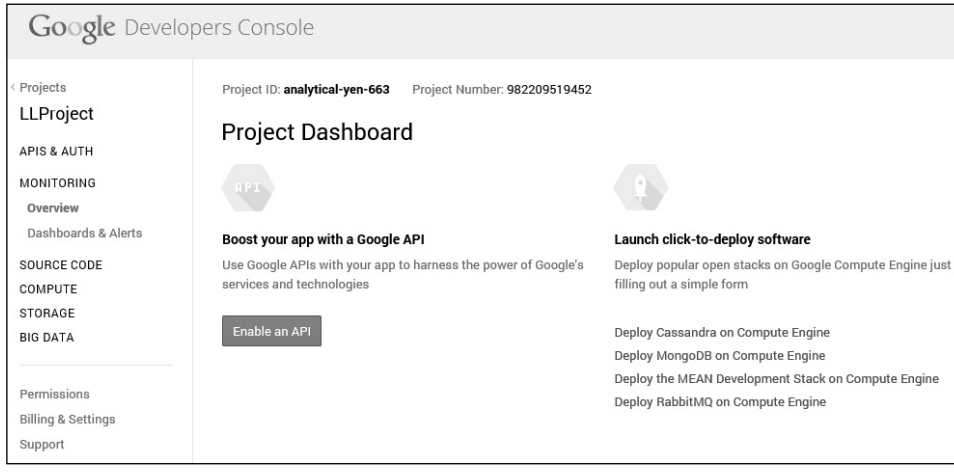
Otevřete webový portál Google Cloud Console na adrese *http://cloud.google.com/console*. Aktivujte funkci **Create Project**.



Obrázek 17.2: Vytvoření projektu přes Google Cloud Console

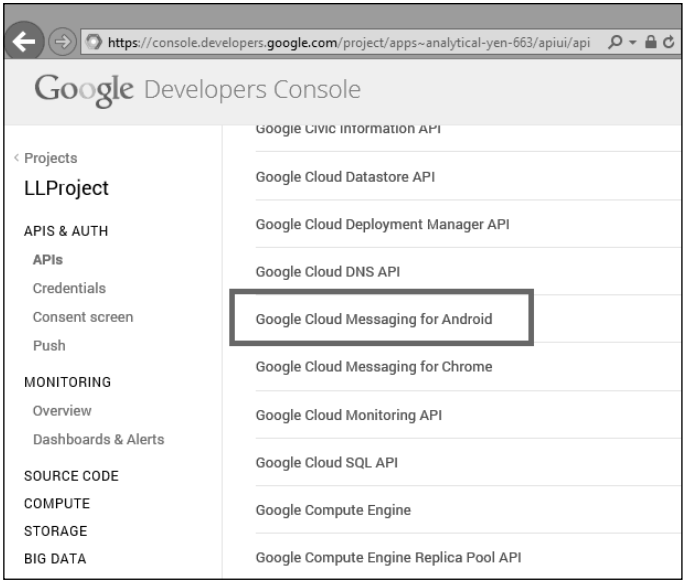
Kód v poli **PROJECT ID** si opište, později v projektu aplikace pro Android ho ve třídě *Main Activity.java* vložíte jako textový řetězec do proměnné `SENDER_ID`.

Po klepnutí na název projektu se zobrazí jeho stránka. Klepněte na tlačítko **Enable an API** nebo na liště vlevo na odkaz **APIs & Auth**.



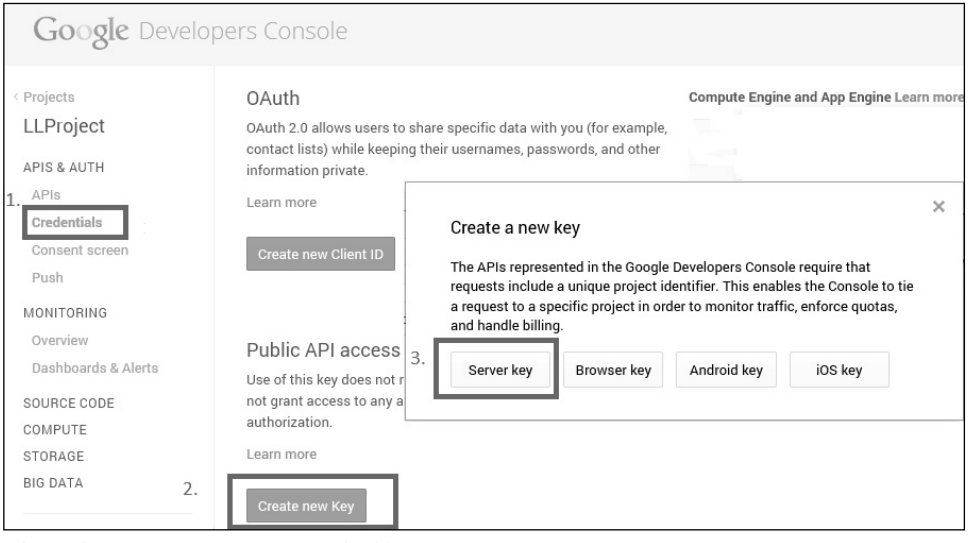
Obrázek 17.3: Stránka projektu

Ze seznamu API vyberte položku **Google Cloud Messaging for Android**. Zapněte API a akceptujte podmínky využívání.



Obrázek 17.4: Výběr API Google Cloud Messaging for Android

V levém úzkém okně klepněte na odkaz **Credentials** a potom stiskněte tlačítko **Create new Key**. Z nabízených možností vyberte **Server key**. V následujícím dialogovém okně vytvořte klíč tlačítkem **Create**.



Obrázek 17.5: Vytvoření serverového klíče

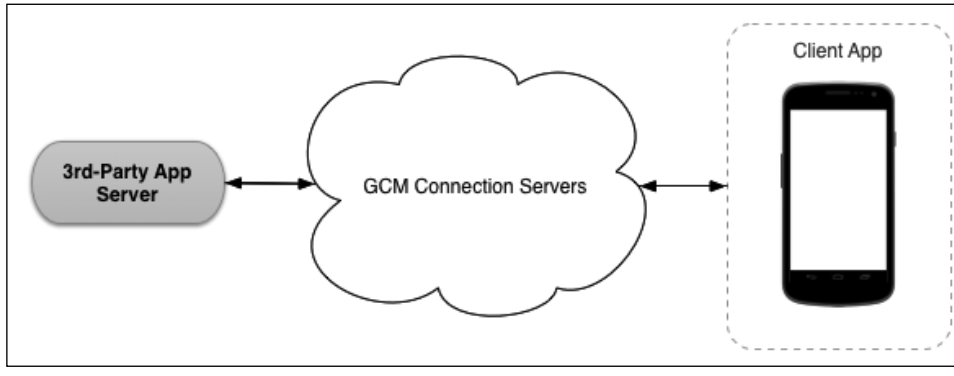
Vygenerovaný API klíč budete moct používat k autentizaci mobilní služby nebo notifikačního hubu vůči GCM a na posílání push notifikací.

Implementace na straně serveru

Aby mohlo fungovat posílání zpráv přes cloudové služby, musíte nejen vytvořit aplikaci, která pracuje s GCM, ale implementovat i službu na straně serveru, která bude údaje ukládat, zpracovávat, přeposílat a podobně.

Na straně serveru se implementace skládá ze dvou částí:

- **Google-provided GCM Connection Server**, který přijímá zprávy z aplikačního serveru třetí strany a posílá je do aplikací pro Android, které mají povolené GCM. Na připojení těchto serverů Google podporuje rozraní HTTP a CCS (XMPP).
- **Aplikační server třetí strany**, který je potřeba implementovat. Tato serverová aplikace odešle údaje aplikaci pro Android prostřednictvím GCM.



Obrázek 17.6: Architektura GCM

Abychom mohli vytvořit aplikaci pro Android, která pracuje s cloudovými notifikacemi, potřebujeme aplikační server třetí strany, případně cloudovou platformu, kde běží aplikace na pozadí, která poskytuje údaje pro mobilní aplikace. Tato aplikace potřebuje posílat notifikace do všech připojených aplikací, případně do aplikace konkrétního uživatele. Pro naši první cvičnou aplikaci jsme se rozhodli využít službu Azure Notification Hubs, která je součástí cloudové platformy Microsoft Azure.

V druhé cvičné aplikaci ukážeme připojení k mobilní službě s databází a 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, díky 30denní trial verzi dostupné 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 jeho syntaxe je na 95 procent shodná s Javou.

Vytvoření Azure Notification Hub

V této fázi se dění znovu přenese do vašeho Microsoft Azure konta, které musíte mít v tomto okamžiku vytvořené a aktivované. Můžete použít:

- 30denní Free trial konto
- Konto získané jako předplatitel MSDN
- Konto studenta přes DreamsPark
- Komerční konto

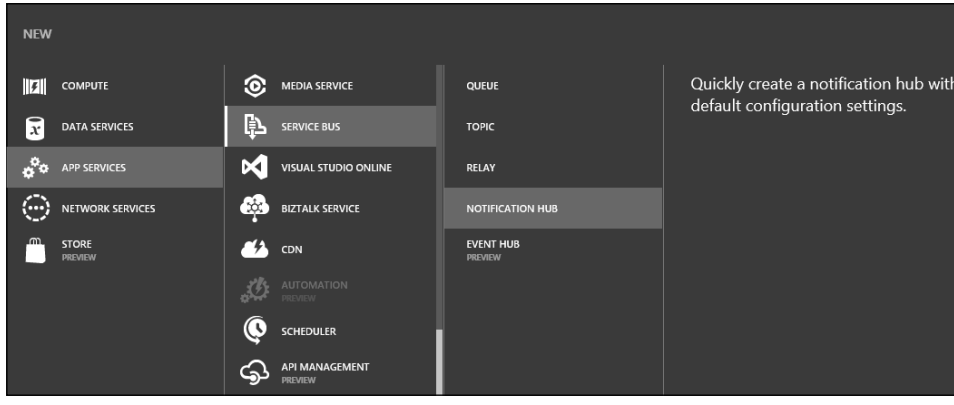
Vstupní bod do platformy je *www.windowsazure.com*. Můžete si vytvořit zkušební konto a během jednoho měsíce máte k dispozici na vyzkoušení libovolné služby v objemu 150 eur.

Notification Hub budete vytvářet přes **Microsoft Azure Management Portal** *https://manage.windowsazure.com/*.



Poznámka: Při prvním přístupu na Azure Management Portal se vám zobrazí krátký tutoriál, který vás seznámí s ovládáním uživatelského rozhraní portálu.

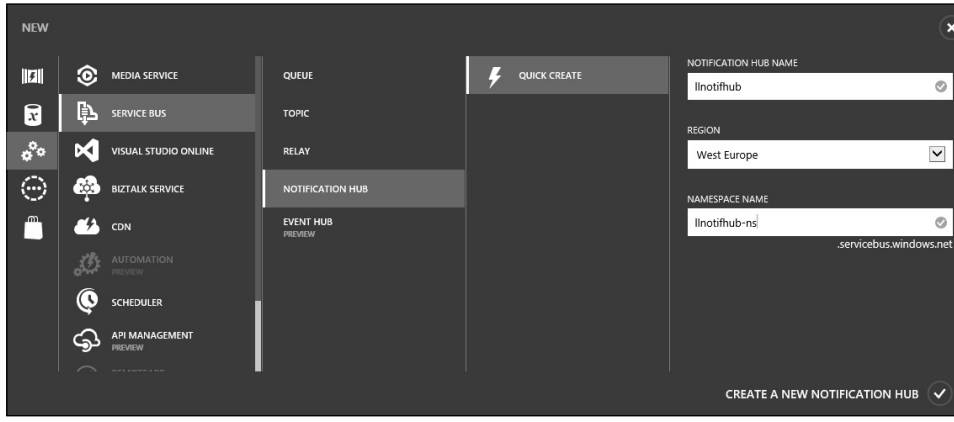
Nový Service Bus vytvoříte tlačítkem **New**, které najdete na dolní aplikační liště portálu. Zvolte položku **App Services** → **Service Bus** →**Notification Hub** → **Quick Create**.



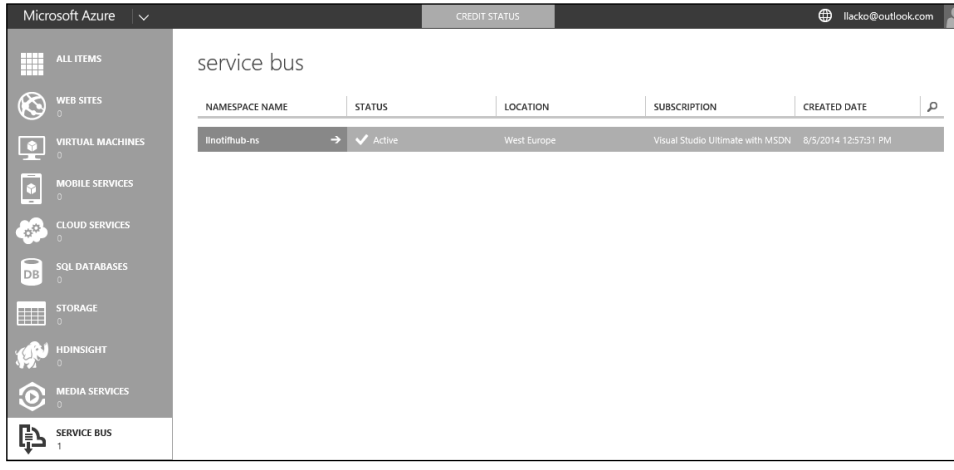
Obrázek 17.7: Vytvoření Notification Hub

Zadejte vhodný název a potvrďte klepnutím na odkaz **Create a new Notification Hub** v dolní části nabídky aplikační lišty.

Po vytvoření, které trvá několik sekund, najdete vámi vytvořený jmenný prostor pro Service Bus na portálu ve složce **SERVICE BUS**.



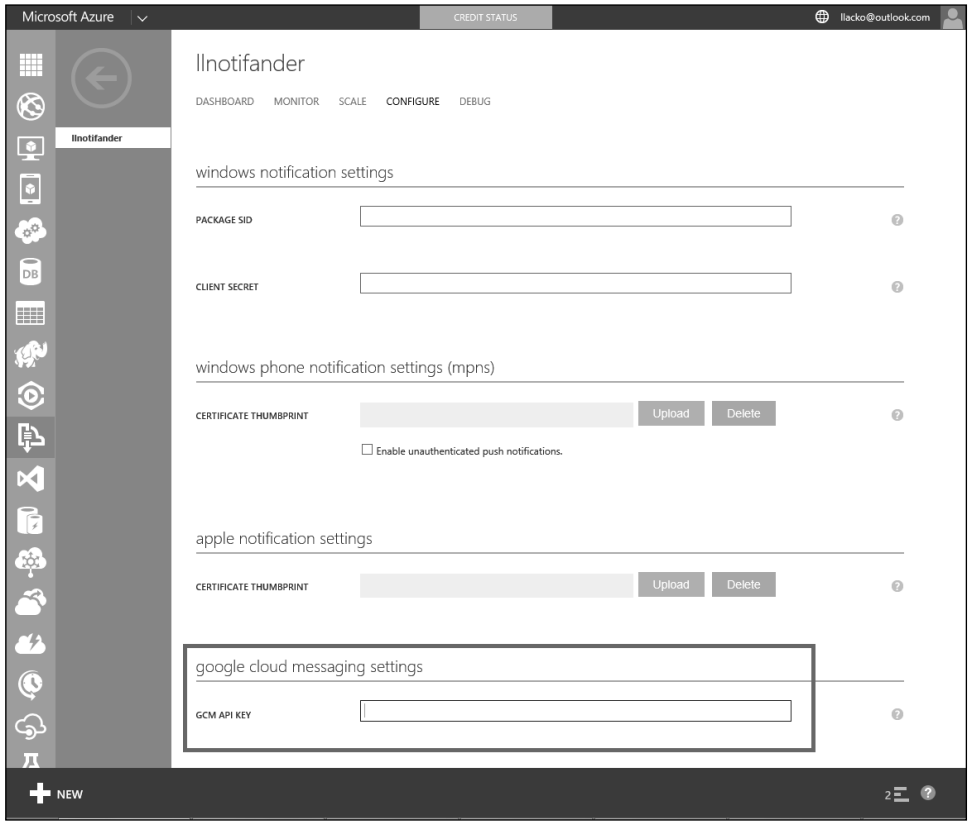
Obrázek 17.8: Zadání názvu pro Notification Hub



Obrázek 17.9: Nově vytvořený jmenný prostor Service Bus ve složce SERVICE BUS

Klepněte na jmenný prostor nově vytvořené služby, který má implicitně název `nazev_notifikacni_sluzby-ns`, a následně na kartu **Notifications Hubs**. Potom můžete vybrat konkrétní Notification Hub (jmenný prostor jich může obsahovat více). Pro vybraný hub se přepněte na kartu **Configure**.

Zobrazí se stránka, kde zadáváte notifikační parametry pro aplikace na požadovaných platformách. My jsme vytvořili Notification Hub s názvem `llnotifander`. Vyberte sekci **Google cloud messaging settings** a zadejte klíč **API Key**, který jste získali přes portál Google Developer Console.

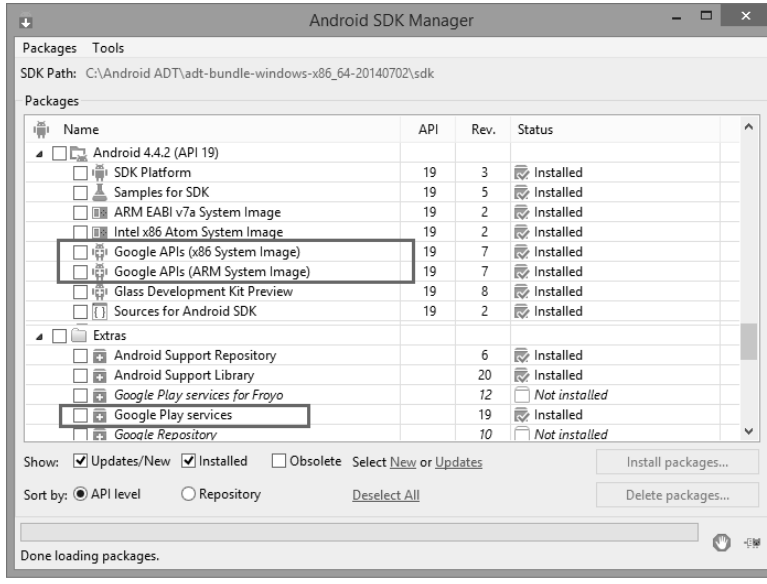


Obrázek 17.10: Zadání CGM API klíče

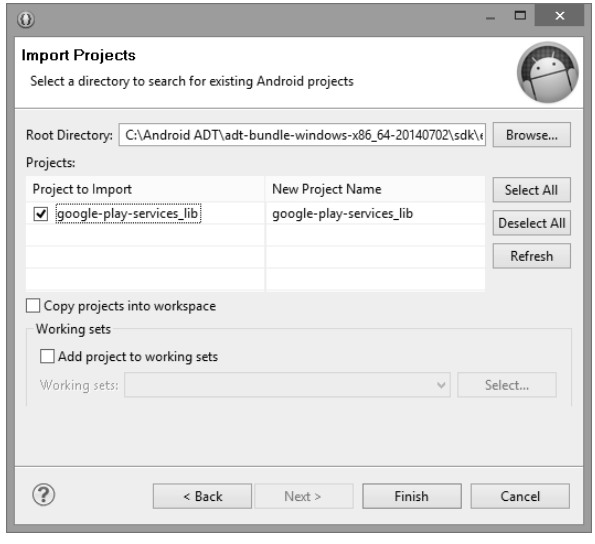
Vytvoření aplikace pro Android vysílající a přijímající notifikace

Po vytvoření projektu aplikace pro Android zkontrolujte v okně **Android SDK Manager**, zda máte nainstalované Google API pro příslušnou verzi Androidu (v tomto případě Android 4.4.2 KitKat API 19) a ve složce *Extras* Google Play Services. Pokud ne, doinstalujte nejprve tyto komponenty.

Po vytvoření projektu je potřeba ve vývojovém prostředí Eclipse pomocí nabídky **File** → **Import** → **Android** → **Existing Android Code** importovat knihovnu Google Play Services ze složky *\extras\google\google_play_services\libproject* do jmenného prostoru, ve kterém vytváříte aplikaci využívající služby.

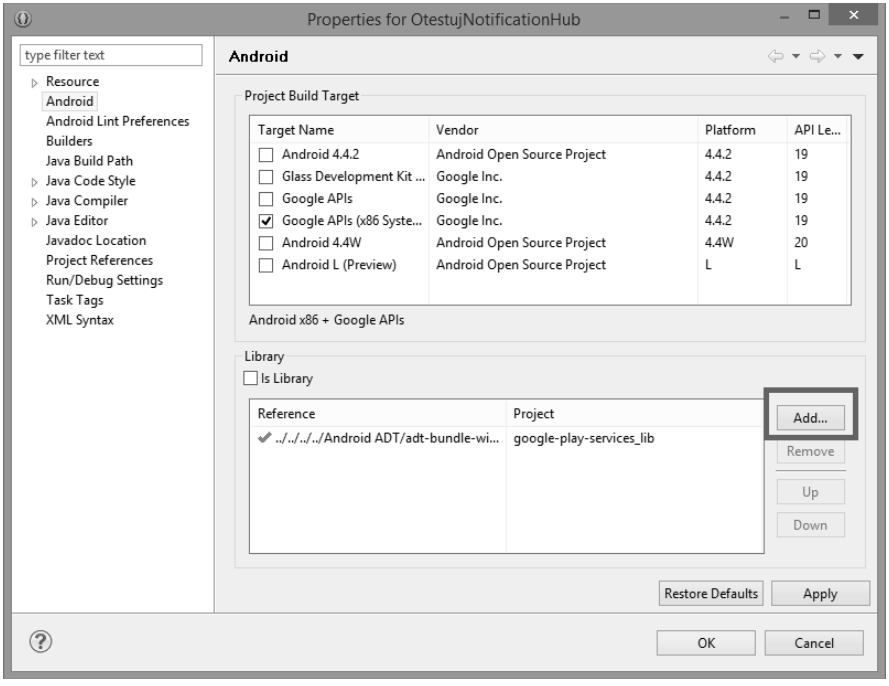


Obrázek 17.11: Instalace API



Obrázek 17.12: Import Google Play Services do jmenného prostoru

Do vašeho projektu je potřeba přidat referenci na Google Play Services. V okně **Package Explorer** vyberte v místní nabídce položku **Properties**. Ve složce Android přidejte referenci.



Obrázek 17.13: Přidání reference na Google Play Services do projektu aplikace pro Android

Následně je potřeba stáhnout Notification Hubs Android SDK. Najdete ho na adrese *https://go.microsoft.com/fwLink/?LinkID=280126&clcid=0x409*. Stáhnete soubor *azuresdk-android--1.1.5.zip*, který obsahuje složky:

- *mobileservices*
- *notificationhubs*
- *notifications*

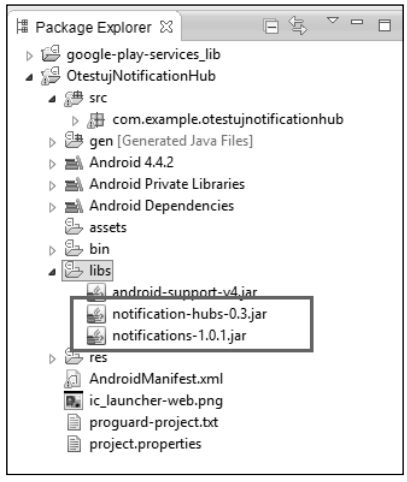
Ze složky *notificationhubs* vybalíte soubor s knihovnou *notification-hubs-0.3.jar* a zkopírujete ho do složky *lib* vašeho projektu. Soubor zkopírujete do schránky v Exploreru a vložíte ho do složky *libs* přímo ve vývojovém prostředí Eclipse v okně **Package Explorer**. Stejně ze složky *notifications* zkopírujte do složky *libs* soubor *notifications-1.0.1.jar*.



Poznámka: Čísla verzí souborů se mohou v dalších verzích lišit.

Referenci na Google Play Services je potřeba přidat do aplikačního manifestu aplikace v souboru *AndroidManifest.xml*.

```
<meta-data
  android:name="com.google.android.gms.version"
  android:value="@integer/google_play_services_version" />
```



Obrázek 17.14: Zkopírování knihoven notification-hubs-0.3.jar a notifications-1.0.1.jar do složky libs projektu vaší aplikace

Do aplikačního manifestu je potřeba přidat několik oprávnění, aby aplikace mohla přistupovat k Internetu, účtům a podobně.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name=
  "com.google.android.c2dm.permission.RECEIVE" />
<permission android:name="<your package>.permission.C2D_MESSAGE"
  android:protectionLevel="signature" />
<uses-permission android:name="<your package>.permission.C2D_MESSAGE"/>
```

Do elementu `<application>` je potřeba přidat deklaraci receiveru pro příjem notifikací.

```
<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>" />
  </intent-filter>
</receiver>
```



Poznámka: Nezapomeňte nahradit žolíka `<your package>` názvem balíku vaší aplikace.

Kompletní kód v souboru *AndroidManifest.xml* bude:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.otestujnotificationhub"
  android:versionCode="1"
  android:versionName="1.0" >
```

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

  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
  <uses-permission android:name="android.permission.WAKE_LOCK"/>
  <uses-permission android:name=
    "com.google.android.c2dm.permission.RECEIVE" />

  <permission android:name=
    "com.example.otestujnotificationhub.permission.C2D_MESSAGE"
    android:protectionLevel="signature" />
  <uses-permission android:name=
    "com.example.otestujnotificationhub.permission.C2D_MESSAGE"/>

  <meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />

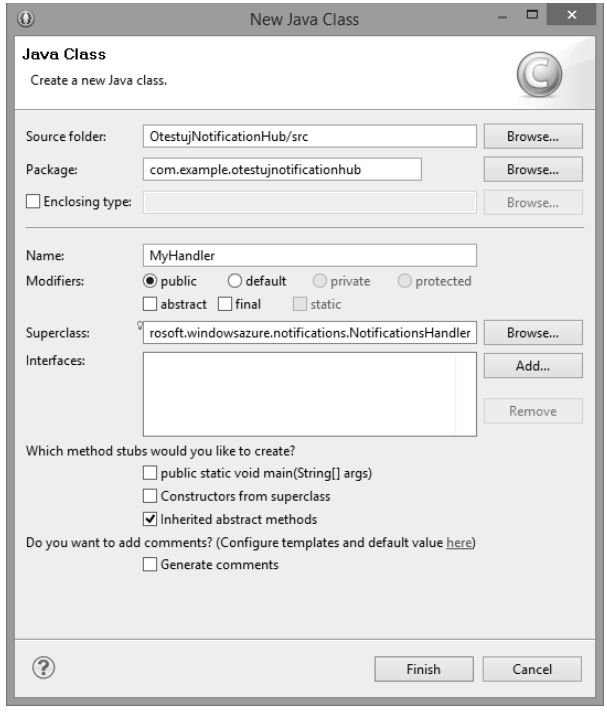
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".MainActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN"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

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <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="com.example.otestujnotificationhub" />
      </intent-filter>
    </receiver>
  </application>

</manifest>
```

Dříve než se dostaneme k úpravám kódu třídy hlavní aktivity v souboru *MainActivity.java*, vytvořte třídu `MyHandler`. Z místní nabídky složky *src/<nazev balicku>*, v našem případě složky *src/com.example.otestujnotificationhub*, vyberte položku **New** → **Class**. Do pole **Superclass** zadejte `com.microsoft.windowsazure.notifications.NotificationsHandler`.



Obrázek 17.15: Dialogové okno na vytvoření třídy MyHandler

Do kódu třídy *MyHandler.java* je potřeba přidat importy:

```
import com.microsoft.windowsazure.notifications.NotificationsHandler;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.NotificationCompat;
```

Kompletní kód v souboru *MyHandler.java* (bez importů) bude:

```
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("msg");
    sendNotification(nhMessage);
```

```
  }

  private void sendNotification(String msg)
  {
    mNotificationManager = (NotificationManager)
      ctx.getSystemService(Context.NOTIFICATION_SERVICE);

    PendingIntent contentIntent = PendingIntent.getActivity(ctx, 0,
      new Intent(ctx, MainActivity.class), 0);

    NotificationCompat.Builder mBuilder =
      new NotificationCompat.Builder(ctx)
      .setSmallIcon(R.drawable.ic_launcher)
      .setContentTitle("Moja notifikácia z AZURE")
      .setStyle(new NotificationCompat.BigTextStyle()
      .bigText(msg))
      .setContentText(msg);

    mBuilder.setContentIntent(contentIntent);
    mNotificationManager.notify(NOTIFICATION_ID, mBuilder.build());
  }
}
```

Vraťme se ke kódu hlavní aktivity. Do kódu třídy hlavní aktivity v souboru *MainActivity.java* je potřeba přidat importy.

```
import com.google.android.gms.gcm.GoogleCloudMessaging;
import com.microsoft.windowsazure.messaging.NotificationHub;
import com.microsoft.windowsazure.notifications.NotificationsManager;
```

Do třídy `MainActivity` před metodu `onCreate` přidejte deklarace:

```
private String SENDER_ID = "<your project number>";
private GoogleCloudMessaging gcm;
private NotificationHub hub;
```

Místo „žolíka“ `"<your project number>"` vložte kód zobrazený v poli **PROJECT ID** (více v části Vytvoření projektu na Google Cloud Console v této kapitole).

Do metody `onCreate` přidejte řádky kódu:

```
NotificationsManager.handleNotifications(this, SENDER_ID, MyHandler.class);
gcm = GoogleCloudMessaging.getInstance(this);
String connectionString = "<your listen access connection string>";
hub = new NotificationHub("<your notification hub name>",
  connectionString, this);
registerWithNotificationHubs();
```

Do třídy hlavní aktivity dále přidejte kód metody:

```
@SuppressWarnings("unchecked")
  private void registerWithNotificationHubs()
  {
    new AsyncTask()
    {
      @Override
```

```
      protected Object doInBackground(Object... params)
      {
        try
        {
          String regid = gcm.register(SENDER_ID);
          hub.register(regid);
        } catch (Exception e) {return e;}
        return null;
      }
    }.execute(null, null, null);
  }
```

Kompletní kód hlavní aktivity v souboru *MainActivity.java* (bez importů) bude:

```
public class MainActivity extends Activity
{
  private String SENDER_ID = "analytical-yen-663";
  private GoogleCloudMessaging gcm;
  private NotificationHub hub;

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

    NotificationsManager.handleNotifications(this, SENDER_ID,
      MyHandler.class);
    gcm = GoogleCloudMessaging.getInstance(this);
    String connectionString =
      "<***your listen access connection string***>";
    hub = new NotificationHub("llnotifander", connectionString, this);
    registerWithNotificationHubs();
  }

  @SuppressWarnings("unchecked")
  private void registerWithNotificationHubs()
  {
    new AsyncTask()
    {
      @Override
      protected Object doInBackground(Object... params)
      {
        try
        {
          String regid = gcm.register(SENDER_ID);
          hub.register(regid);
        } catch (Exception e) {return e;}
        return null;
      }
    }.execute(null, null, null);
  }
}
```

Nyní teoreticky můžete aplikaci spustit, ideálně 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 však neudělá nic užitečného, jelikož očekává oznámení od serveru, které je potřeba poslat.

Posílání notifikací

Abyste mohli v aplikaci přijmout notifikaci, musí ji někdo přes něco poslat. Reálně by to dělala serverová aplikace, vytvoření takovéto aplikace je však mimo téma této publikace. My budeme posílání notifikace simulovat nejjednodušším možným způsobem, pomocí konzolové aplikace pro Windows připojené k Azure Notification Hub.



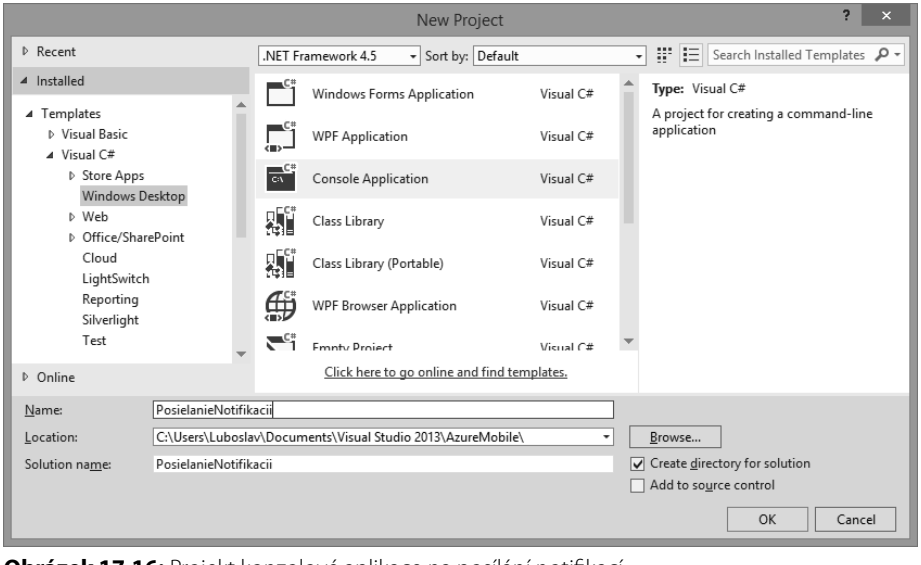
Poznámka: Takovouto aplikaci vytvoříte podle návodu ve volně šiřitelné verzi vývojového prostředí Visual Studio 2013 Express Edition za méně než 5 minut.

Pokud nemáte Visual Studio 2013, nainstalujete si edici Express ze stránky *http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx*.

Ve Visual Studiu vytvořte projekt aplikace na posílání notifikací. Bude to nejjednodušší konzolová aplikace bez uživatelského rozhraní. Projekt vytvoříte pomocí nabídky **File → New Project**. Z nabídky šablon pro vámi vybraný programovací jazyk vyberte složku WindowsDesktop a v ní položku Console Application.

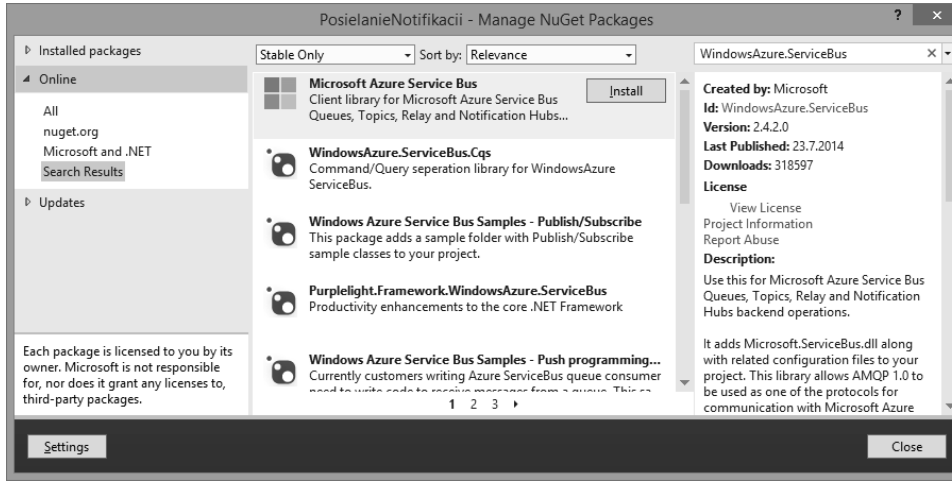


Poznámka: My jsme vybrali programovací jazyk C#, který vychází ze syntaxe Javy a jeho syntaxe je na 95 procent shodná s Javou.



Obrázek 17.16: Projekt konzolové aplikace na posílání notifikací

Do projektu přidejte **WindowsAzure.ServiceBus NuGet**. V místní nabídce první složky projektu vyberte položku **Manage NuGet Packages**. Zobrazí se dialogové okno **Manage NuGet Packages**. Do okna online vyhledávání zadejte název `WindowsAzure.ServiceBus`. Po nalezení balíčku ho nainstalujte.



Obrázek 17.17: Přidání NuGet balíčku WindowsAzure.ServiceBus

Do souboru *Program.cs* přidejte referenci na jmenný prostor:

```
using Microsoft.ServiceBus.Notifications;
```

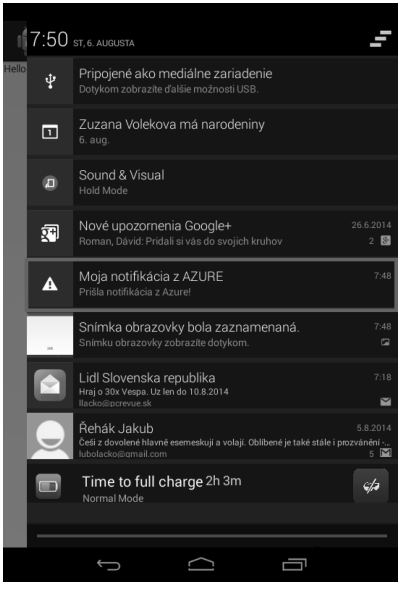
Ve třídě `Program` vytvořte metodu:

```
private static async void SendNotificationAsync()
{
  NotificationHubClient hub =
    NotificationHubClient.CreateClientFromConnectionString
    ("Endpoint=sbxx", "llnotifander");
  await hub.SendGcmNativeNotificationAsync
    ("{ \"data\" : {\"msg\":\"Přišla notifikace z Azure!\"}}");
}
```

Volání metody na vysílání notifikační zprávy umístěte do těla metody `Main()`.

```
static void Main(string[] args)
{
  SendNotificationAsync();
  Console.ReadLine();
}
```

Nyní můžete aplikaci spustit, a pokud je vše v pořádku, zobrazí se v aplikaci notifikační zpráva. Po stažení panelu oznámení v levém horním rohu obrazovky zařízení dolů se zobrazí okno notifikačního centra. Oznámení vaší aplikace sestává z ikony, názvu, textu oznámení a data.



Obrázek 17.18: Notifikace z Azure v seznamu notifikací

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 Services. V reálné aplikaci můžete využít tuto nebo jinou komerční službu nebo si napsat serverovou aplikaci sami. Výhodou Azure Mobile Services je, že hned po vytvoření mobilní služby v cloudu si můžete stáhnout hotový projekt ukázkové aplikace pro všechny nejpoužívanější mobilní platformy včetně Androidu.

Námětem ukázkové aplikace bude seznam úkolů, které přes mobilní službu můžete sdílet mezi více zařízeními toho samého uživatele, a to i napříč platformami.

Microsoft Azure Mobile Services

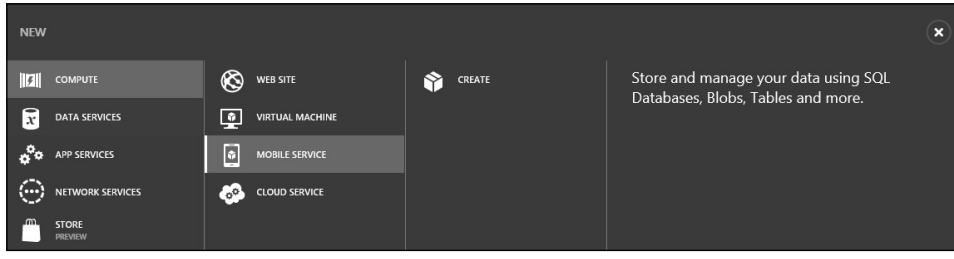
Mobilní služby se využívají pro specifický okruh často se opakujících úloh v mobilních aplikacích, typicky k ukládání dat, autentizaci, posílání notifikací a podobně. Použitím mobilní služby si ulehčíte a zrychlíte vývoj, jelikož funkcionalitu, kterou přesunete na mobilní službu, byste museli pracně vytvářet a provozovat ji na vlastním serverovém prostředí. Mobilní službu

vytvoříte rychle, doslova na několik klepnutí. Mobilní služby mohou využívat aplikace pro nejpoužívanější tabletové a mobilní platformy Windows 8 a 8.1, Windows Phone, iOS a Android. Provoz mobilní 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.

Cenové kalkulace najdete na *http://azure.microsoft.com/en-us/pricing/calculator/?scenario=mobile*. Můžete provozovat až 10 služeb zdarma a zdarma ukládat údaje do objemu 20 MB.

Vytvoření nové mobilní služby

K vytvoření mobilní služby použijte webový nástroj Azure Management Portal na adrese *https://manage.windowsazure.com*. V dolní části levého sloupce s přehledem aktuálně využívaných služeb zvolte položku **New**. Postupně klepněte na volby **Compute** → **Mobile Service** a následně aktivujte tlačítko **Create.**



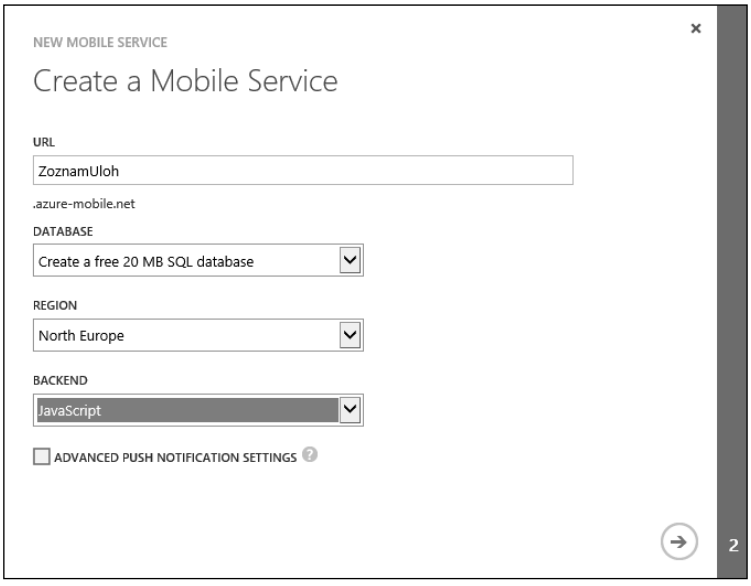
Obrázek 17.19: Vytvoření nové mobilní služby

Většina mobilních služeb se neobejde bez ukládání údajů, proto bude prvním krokem vytvoření databáze. Pokud vám stačí 20 MB, můžete si vytvořit bezplatnou 20MB SQL databázi. K vytvoření databáze stačí do pole URL zadat její název, v našem případě třeba `ZoznamUloh`. 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 zvětšení šířky pásma a přístup služby k databázi bude mít samozřejmě vyšší latenci.

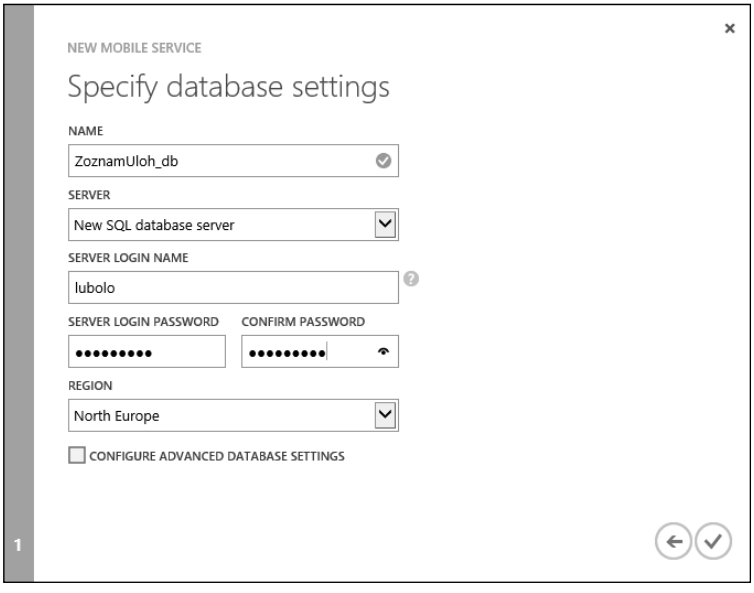
V našem případě jsme vybrali region **North Europe**. V poli **Backend** zvolte pro tento příklad možnost **JavaScrip**t. Alternativní volbou je .NET.

Tuto free 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 použít i na právě vytvářenou mobilní službu.

V následujícím dialogu buď vyberte existující databázový server, nebo podobně jako v našem případě zvolte položku **New SQL database server**. Zadejte přístupové jméno a heslo. U hesla dbejte, aby bylo dostatečně silné.

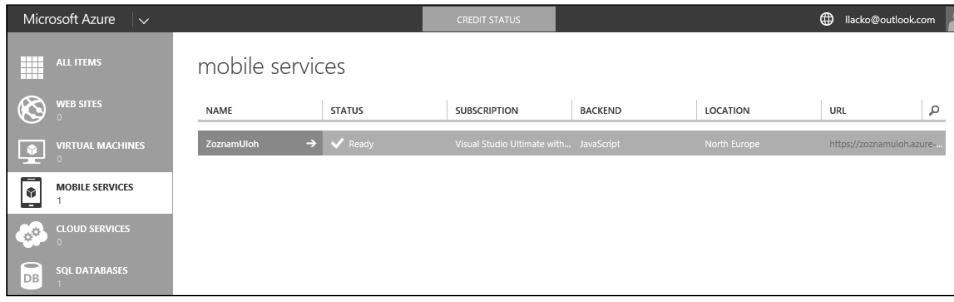


Obrázek 17.20: Vytvoření 20MB SQL databáze



Obrázek 17.21: Konfigurace databázového serveru

Po potvrzení se začne vytvářet mobilní služba a databáze. Vytvoření mobilní služby bude trvat nějaký čas, při našich pokusech to bylo přibližně půl minuty. Po vytvoření se zobrazí v seznamu mobilních služeb Microsoft Azure portálu.



Obrázek 17.22: Nově vytvořená mobilní služba

Vytvoření aplikace pro Android využívající mobilní službu

Předpokládejme výchozí situaci po vytvoření mobilní služby. Pokud na tuto službu v seznamu **Mobile Services** klepnete, zobrazí se nabídka vytvoření mobilních aplikací pro různé platformy. Z nabízených možností vyberte Android. Zobrazí se aktualizovaná nabídka.

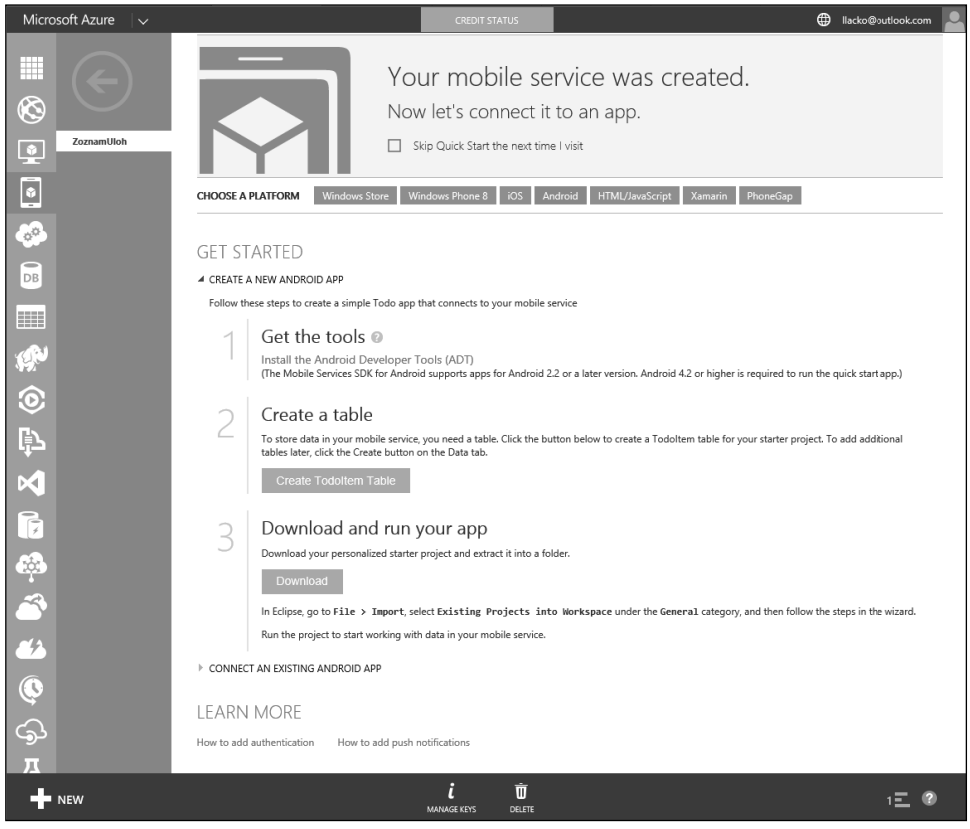
- **Create a new Android App** – vytvoří šablonu projektu aplikace využívající mobilní službu.
- **Connect an existing Android App** – vygeneruje postup a fragmenty kódu, které je potřeba přidat do aplikace, aby mohla komunikovat s mobilní službou.

V prvním příkladu zvolte **Create a new Android App.** Aplikace je kompletní, její součástí je i cvičná databázová tabulka, kterou můžete spustit, prozkoumat a případně modifikovat.

Pokud nemáte vytvořenou tabulku `TodoItem`, klepněte na tlačítko **Create TodoItem Table**. V kroku číslo 3 si stáhněte projekt ukázkové aplikace pro Android využívající mobilní službu. Stáhněte kompletní projekt aplikace zabalený v archívu ZIP. V našem případě bude mít archív název *ZoznamUloh.Zip*. V Eclipse zvolte **File → Import → Existing Projects into Workspace.**

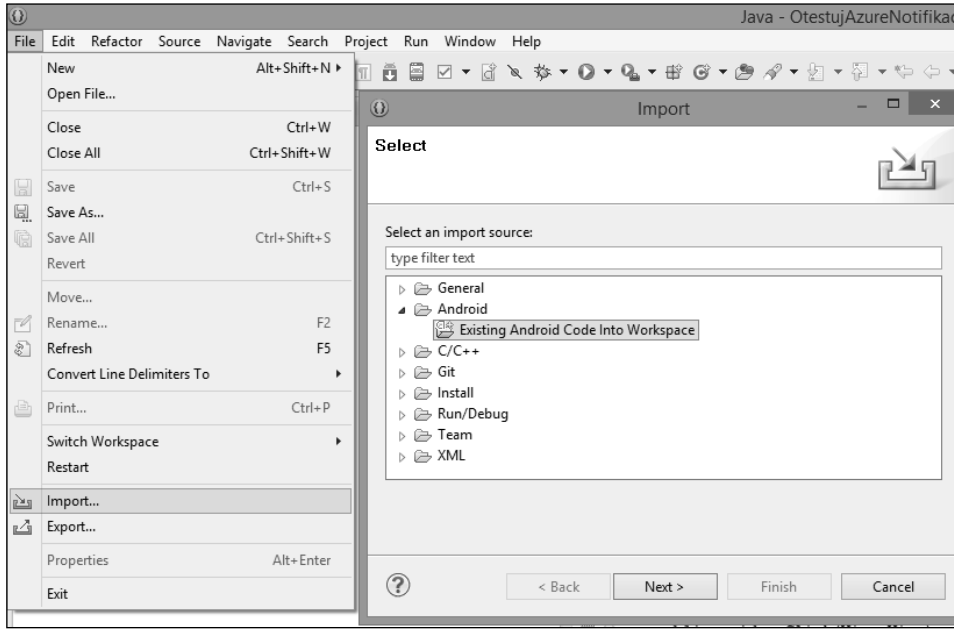
Všimněte si, že aplikace má v manifestu v souboru *AndroidManifest.xml* přidané oprávnění přístupu na Internet.

```
<uses-permission android:name="android.permission.INTERNET" />
```

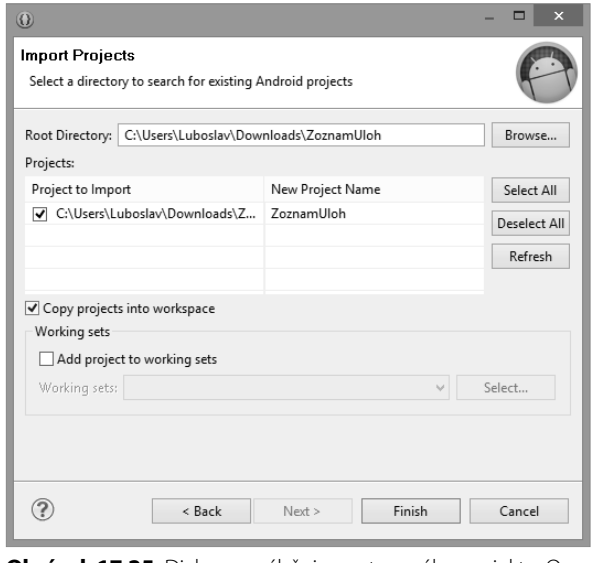


Obrázek 17.23: Postup vytvoření projektu aplikace pro Android

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.



Obrázek 17.24: Inicializace importu projektu aplikace pro Android ve vývojovém prostředí Eclipse



Obrázek 17.25: Dialog na výběr importovaného projektu. Označte pole Copy project into workspace.

```
//Mobile Service Client reference
private MobileServiceClient mClient;
//Mobile Service Table used to access data
```

```
private MobileServiceTable<ToDoItem> mToDoTable;

//Adapter to sync the items list with the view
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);
  mProgressBar.setVisibility(ProgressBar.GONE);

  try
  {
    mClient = new MobileServiceClient(
      "https://zoznamuloh.azure-mobile.net/",
      "iHlKtHakccfDYIlWfgXZdSQSAPzOxA99",
      this).withFilter(new ProgressFilter());

    // Get the Mobile Service Table instance to use
    mToDoTable = mClient.getTable(ToDoItem.class);
    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();

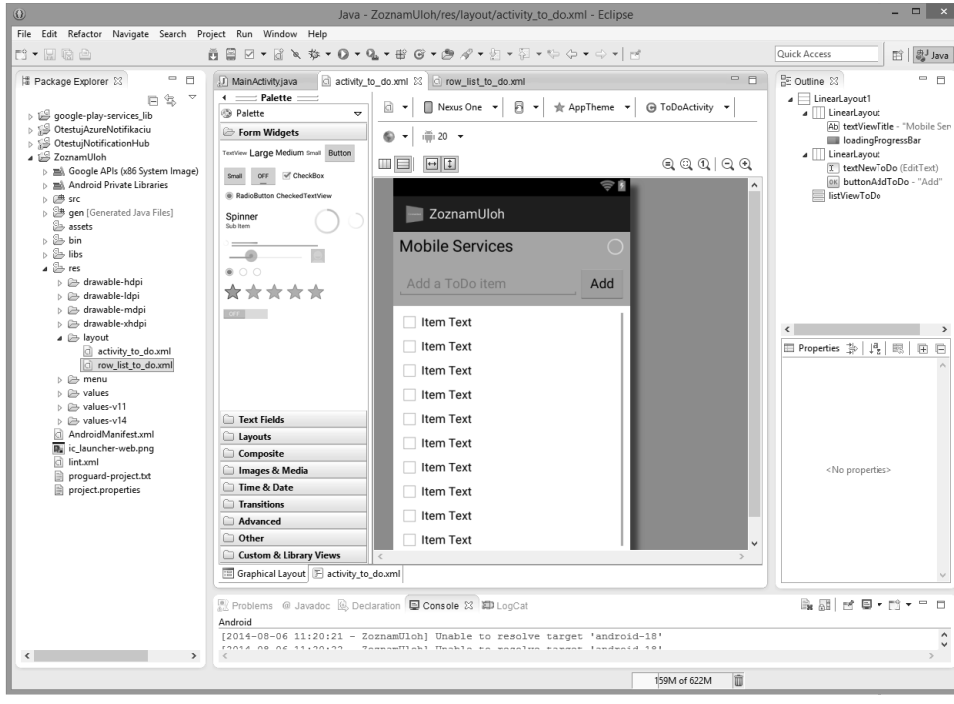
  } catch (MalformedURLException e) {
    createAndShowDialog(new Exception("There was an error creating the
      Mobile Service. Verify the URL"), "Error");
  }
}

public void addItem(View view)
{
  if (mClient == null) return;
  // Create a new item
  ToDoItem item = new ToDoItem();

  item.setText(mTextNewToDo.getText().toString());
  item.setComplete(false);
  // Insert the new item
  mToDoTable.insert(item, new TableOperationCallback<ToDoItem>()
  {
```

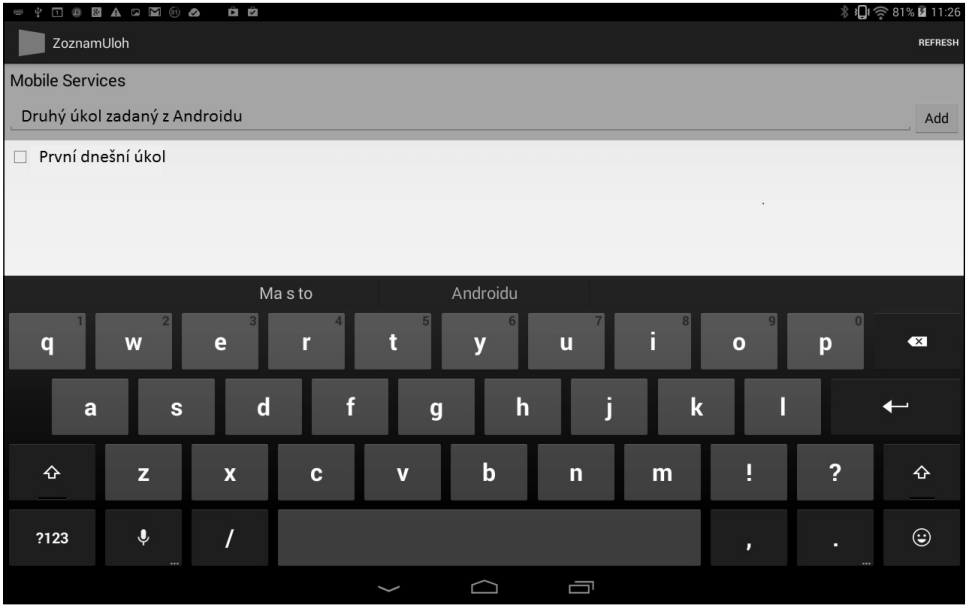
```
    public void onCompleted(ToDoItem entity, Exception exception,
      ServiceFilterResponse response)
    {
      if (exception == null)
      {
        if (!entity.isComplete())
        {
          mAdapter.add(entity);
        }
      } else
      {
        createAndShowDialog(exception, "Error");
      }
    }
  });
  mTextNewToDo.setText("");
}
```

Ve vývojovém prostředí Eclipse můžete případně projekt přizpůsobit svým požadavkům.



Obrázek 17.26: Projekt aplikace pro Android ve vývojovém prostředí Eclipse

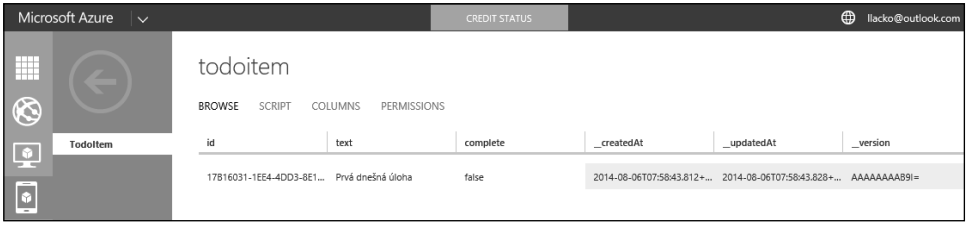
V našem případě jsme se k mobilní službě připojili z aplikace pro Windows 8.1 a přidali jsme položku „První dnešní úkol“ (viz předchozí příklad). Následně jsme spustili aplikaci pro Android, kde se tento úkol zobrazil, a přidali jsme položku „Druhý úkol zadaný z Androidu“.



Obrázek 17.27: Test aplikace na tabletu s Androidem

Co se stalo na pozadí? Po potvrzení zadaného textu ho aplikace pošle jako POST mobilní službě v cloudu a ta uloží údaje do tabulky `ToDoTable`. Po zápisu se údaje vrátí na mobilní zařízení a zobrazí se aplikaci napojené na mobilní službu.

Přes Azure Management Portal se můžete přesvědčit (karta **Browse**), že vámi zadaný text je v tabulce databáze, na kterou je napojená mobilní služba.



Obrázek 17.28: Údaje zadané v aplikaci byly zapsány do tabulky databáze, na kterou je napojená mobilní služba

Pokud v aplikaci zaškrtnete políčko u splněného úkolu, položka se přestane zobrazovat. Tímto úkonem jste v tabulce změnili hodnotu atributu `complete` z `false` na `true`.

Úprava existující aplikace pro Android, aby mohla využívat mobilní službu

Do manifestu v souboru *AndroidManifest.xml* přidejte povolení:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Do souboru s kódem aktivity přidejte import:

```
import com.microsoft.windowsazure.mobileservices.*;
```

Do kódu aktivity přidejte deklaraci:

```
private MobileServiceClient mClient;
```

Do metody `onCreate()` přidejte kód:

```
public class Item
{
  public String Id;
  public String Text;
}
```

Do projektu přidejte třídu `Item`:

```
public static MobileServiceClient MobileService = new MobileServiceClient(
  "https://zoznamuloh.azure-mobile.net/",
  "iHlKtHakccf...zbytek klíče"
);
```

Na přidání záznamu použijte fragment kódu:

```
Item item = new Item();
item.Text = "Awesome item";
mClient.getTable(Item.class).insert(item,
  new TableOperationCallback<Item>()
{
  public void onCompleted(Item entity, Exception exception,
    ServiceFilterResponse response)
  {
    if (exception == null)
    {
      // Insert succeeded
    } else
    {
      // Insert failed
    }
  }
});
```

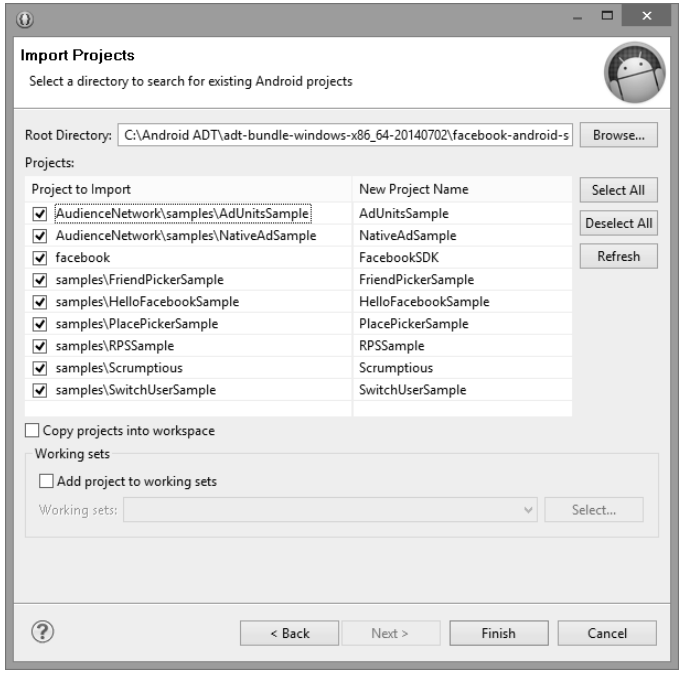
Připojení aplikace k sociálním sítím

Několikrát v této kapitole rezonovalo téma top skóre ve hře. Pokud vytváříte hru, určitě přidáte funkcionalitu evidování nejvyššího dosaženého skóre. Buď v centrálním žebříčku pro všech-

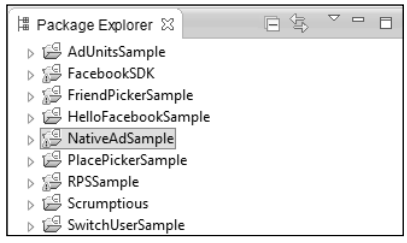
ny hráče, kteří využívají vaši aplikaci (tehdy potřebujete cloudovou službu nebo serverovou aplikaci s databází), nebo jen lokálně, kde se zaznamenává pouze vaše skóre. Tak či onak, když hráč dosáhne vysokého skóre, možná se bude chtít dosaženým úspěchem pochlubit na sociální síti. Může tak učinit přes web sociální sítě, pro uživatele však je jednodušší a interaktivnější poslat status na sociální síť přímo z vaší aplikace pro Android.

Z adresy *https://developers.facebook.com/resources/facebook-android-sdk-current.zip* si stáhněte Facebook SDK for Android a Facebook APK. Je to archív ZIP, v našem případě soubor *facebook-android-sdk-3.17*. Číslo se bude lišit v závislosti na aktuální verzi. Po rozbalení získáte složku se stejným názvem jako archivní soubor. Složku umístěte na vhodné místo, například tam, kde máte Android SDK. Samotné SDK je ve vnořené složce *facebook*.

Ve vývojovém prostředí Eclipse zvolte **File → Import** a vyberte položku **General/Existing Projects into Workspace**. V dialogu na import projektu označte celou složku SDK staženou z webu Facebooku pro vývojáře; v našem případě *facebook-android-sdk-3.17.1*. Všimněte si, že v seznamu je několik zajímavých příkladů. Položku **Copy projects into workspace** musíte ponechat neoznačenou, aby SDK a příklady zůstaly v původní lokalitě a nezměnila se adresářová struktura.



Obrázek 17.29: Import SDK na připojení aplikace pro Android k Facebooku



Obrázek 17.30: Importované SDK a příklady v okně Package Explorer

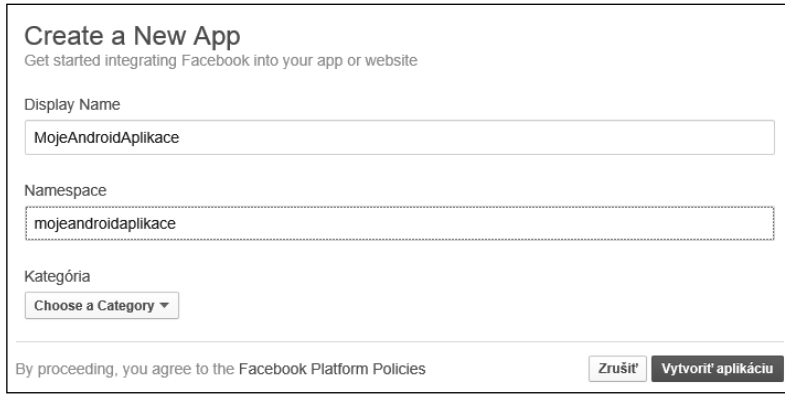
Nejprve doporučujeme prostudovat příklad `HelloFacebookSample`, který demonstruje základní funkcionalitu na Facebooku – přístup k uživatelskému profilu a update statusu a upload fotografie.

http://blog.doityourselfandroid.com/2011/02/28/30-minute-guide-integrating-facebook- android--application/

Vytvoření aplikace pracující s Facebookem

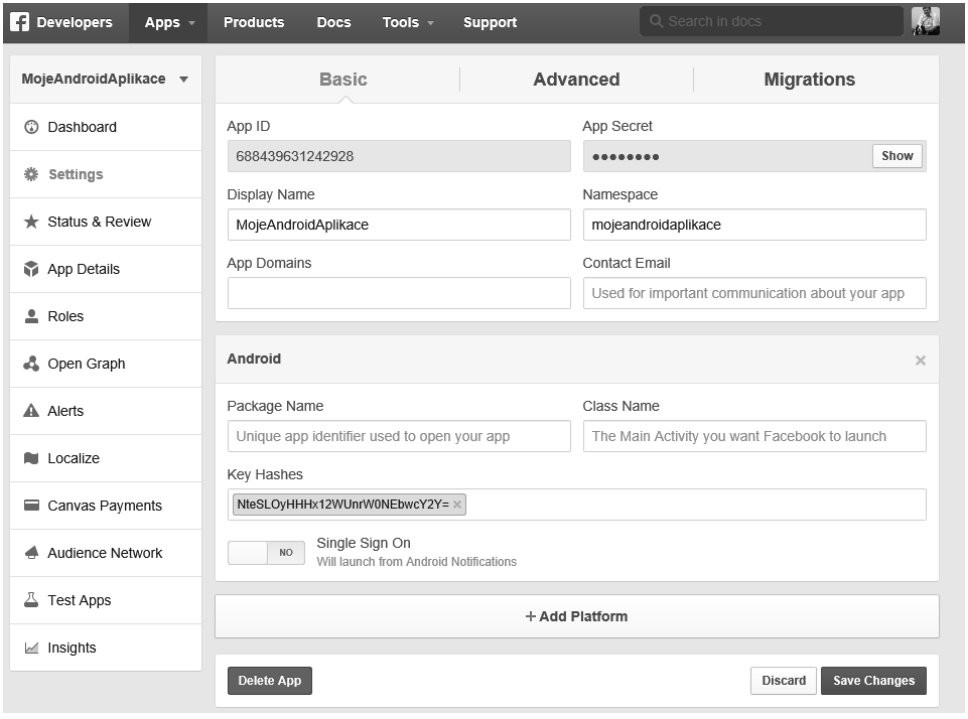
Aby se mohla vaše aplikace přihlašovat k účtu Facebook, je potřeba realizovat několik přípravných úkonů. Na vývojářském portálu Facebooku *https://developers.facebook.com/apps* se musíte nejprve zaregistrovat jako vývojář. Zvolte na liště v horní části obrazovky funkce **Apps → Register as a developer**. Následně proběhne zpřísněná kontrola vaší identity přes SMS na vaše telefonní číslo.

Následně aktivujte na liště v horní části obrazovky funkci **Apps → Create a New App**. Zadejte název aplikace a jmenný prostor.



Obrázek 17.31: Vytvoření aplikace na vývojářském portálu Facebooku

Po vytvoření aplikace se na její dashboard stránce zobrazí parametry **App ID** a **App Secret**. Parametr **App ID** budete potřebovat vložit do projektu vaší aplikace pro Android. Přejděte do záložky **Settings** a klepněte na tlačítko **Add Platform**.



Obrázek 17.32: Přidání platformy pro vaši aplikaci

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.exe* z balíčku Java. Postup je stejný, jaký jste použili v kapitole věnované mapovým službám.

Ladicí klíč vygenerovaný vývojovým prostředím Eclipse 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 | openssl base64
```

Aplikaci *keytool.exe* najdete v adresáři, do kterého se nainstalovala Java. V našem případě to byl adresář *C:\Program Files\Java\jre7\bin*. V našem případě byl příkaz ve tvaru:

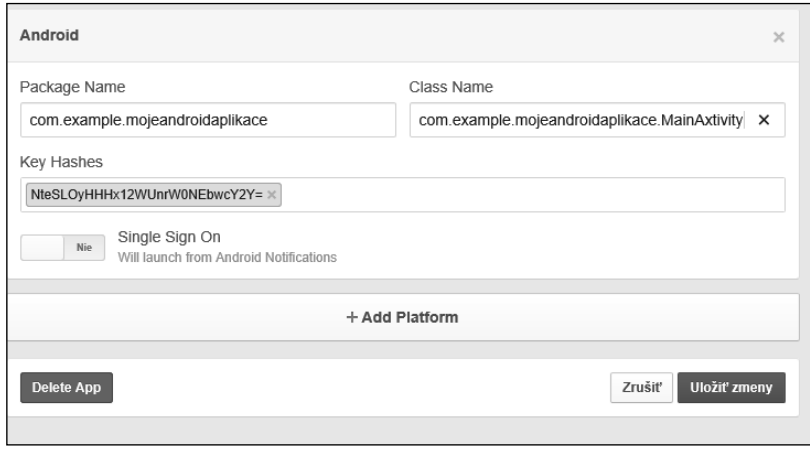
```
keytool -exportcert -alias androiddebugkey -keystore
  "C:\Users\Luboslav\.android\debug.keystore" |
  openssl sha1 -binary | openssl base64
```

Abyste mohli příkaz spustit, musíte mít nainstalované openssl-for-windows z adresy *https://www.openssl.org/related/binaries.html*.

Vytvoření aplikace pro Android přihlašující se k Facebooku

Vytvořte projekt aplikace pro Android. Nemusíte používat stejný název aplikace a jmenného prostoru, jaký jste zadali, když jste registrovali aplikaci na vývojářském portálu Facebooku, je to však vhodné kvůli konzistenci.

Po vytvoření projektu se vraťte na dashboard vývojářského portálu Facebooku a vyplňte dvě dosud nevyplněná pole: **Package Name** a **Class Name**.



Obrázek 17.33: Doplnění údajů do polí Package Name a Class Name

Znovu se vraťte do vývojového prostředí Eclipse, vaše nová aplikace musí být spojená s projektem knihovny Facebook SDK. V místní nabídce projektu aktivujte položku **Properties** a přejděte na kartu **Android**. Ve spodní části dialogového okna klepněte na tlačítko **Add** a vyberte projekt **FacebookSDK** z pracovního prostoru vaší aplikace, kam jste ho předtím importovali.

Následně přidejte Facebook app ID jako řetězec do souboru *strings.xml* ve složce *res/values*.

```
<string name="app_id">688439631242333</string>
```

Do aplikačního manifestu v souboru *AndroidManifest.xml* přidejte oprávnění přistupovat k Internetu.

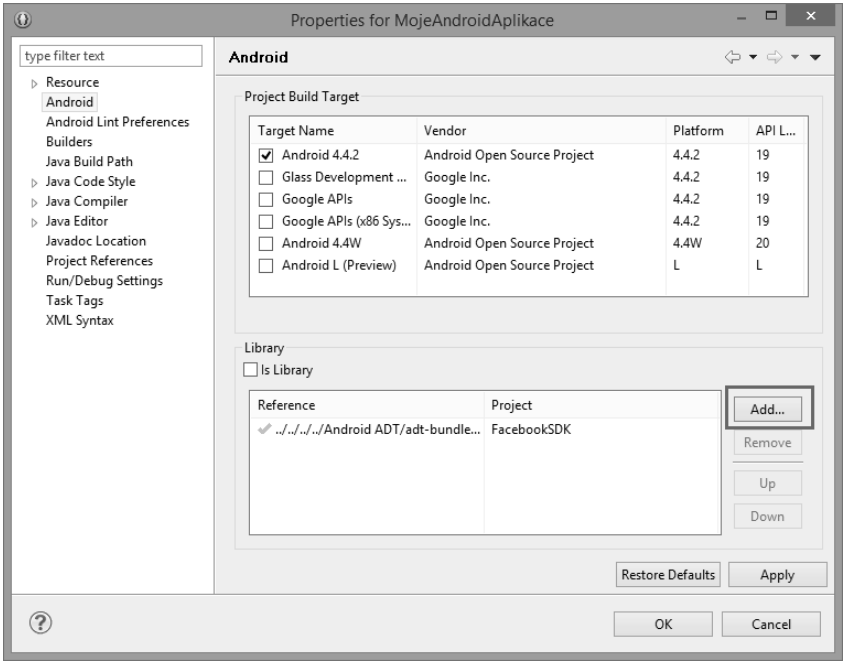
```
<uses-permission android:name="android.permission.INTERNET"/>
```

Do elementu `application` přidejte metadata:

```
<meta-data android:name="com.facebook.sdk.ApplicationId"
  android:value="@string/app_id"/>
```

Nakonec přidejte do seznamu aktivit novou aktivitu na přihlašování aplikace k účtu Facebook.

```
<activity android:name="com.facebook.LoginActivity"/>
```



Obrázek 17.34: Přidání knihovny Facebook SDK do projektu

Layout hlavní aktivity obsahuje po vytvoření projektu jeden prvek typu `TextView`, který slouží na vypsání textu „Hello World“. Abychom tento prvek mohli použít na výpis textu z aplikačního kódu, je potřeba přiřadit mu ID.

```
<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}" >
  <TextView
    android:id="@+id/welcome"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />
</RelativeLayout>
```

V kódu hlavní aktivity v souboru *MainActivity.java* přidejte importy:

```
import com.facebook.*;
import com.facebook.model.*;
```

Důležitou součástí Facebook SDK je třída `Session`, která spravuje velkou část procesu ověřování a autorizace uživatelů. Vzhledem k tomu, že po přihlášení bude požadovat návrat do hlavní aktivity, je potřeba přidat do kódu rozšíření metody `onActivityResult`.

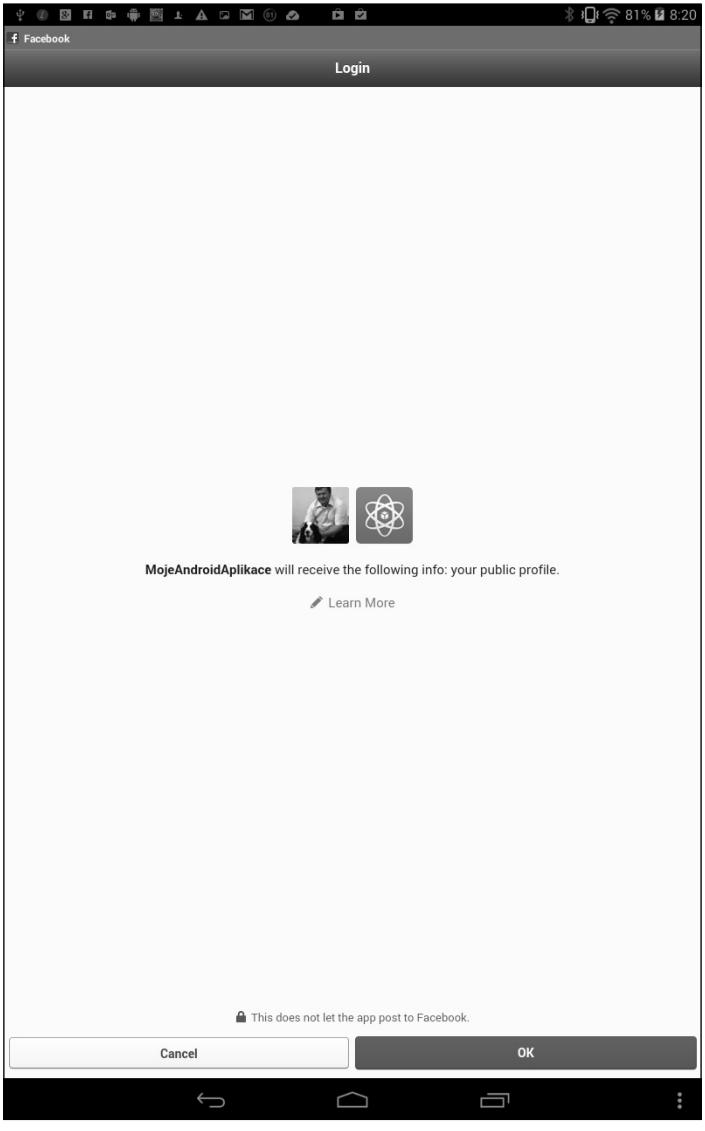
```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data)
{
  super.onActivityResult(requestCode, resultCode, data);
  Session.getActiveSession().onActivityResult(this, requestCode,
    resultCode, data);
}
```

Musíte doplnit kód i do metody `onCreate`.

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

  // Facebook Login
  Session.openActiveSession(this, true, new Session.StatusCallback()
  {
    // callback při změně stavu session
    @Override
    public void call(Session session, SessionState state,
      Exception exception)
    {
      if (session.isOpened())
      {
        Request.newMeRequest(session, new Request.GraphUserCallback()
        {
          @Override
          public void onCompleted(GraphUser user, Response response)
          {
            if (user != null)
            {
              TextView welcome = (TextView) findViewById(R.id.welcome);
              welcome.setText("Hello " + user.getName() + "!");
            }
          }
        }).executeAsync();
      }
    }
  });
}
```

K vývoji komplexní aplikace využívající Facebook doporučujeme tutoriál na adrese *https://developers.facebook.com/docs/android/scrumptious/*.



Obrázek 17.35: Test aplikace na připojení k Facebooku

KAPITOLA 18

Publikování aplikací do služby Google Play

V této kapitole:
- Registrace vývojářského účtu
- Vytvoření balíčku aplikace na publikování
- Publikování aplikace

Google Play je nejnavštěvovanější obchod s aplikacemi pro Android. Umožňuje uživatelům vyhledat a stahovat vaše 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.

Registrace vývojářského účtu

Nejprve se musíte zaregistrovat jako vývojář *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.



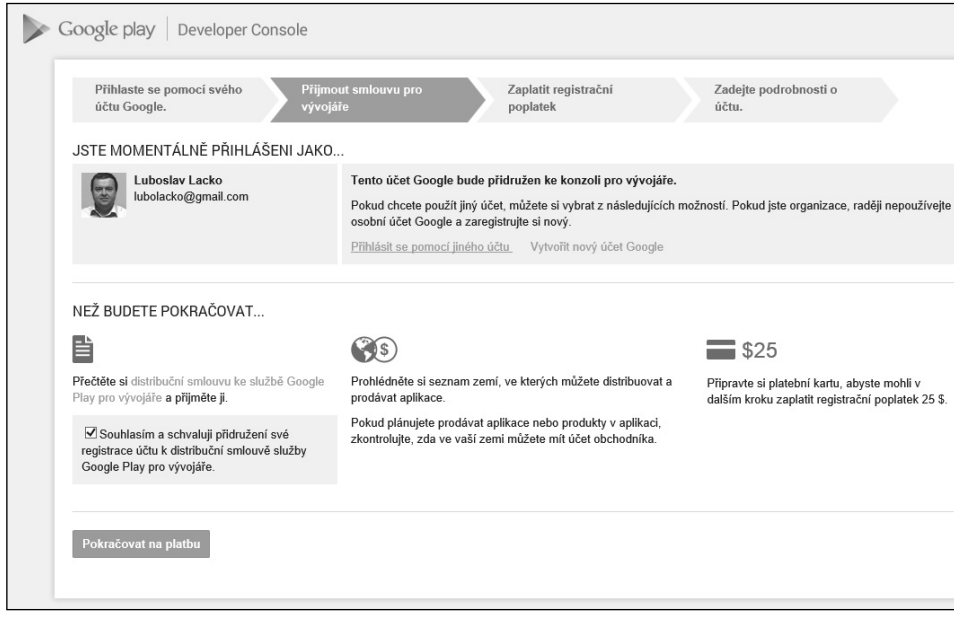
Pokud váš účet v minulosti tuto smlouvu porušil, registrace vývojáře do služby Google Play nebude možná.

Pokud chcete stanovit ceny svých aplikací, musíte mít účet služby Google Wallet Merchant Center. Služba Google Play nabízí několik způsobů na 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 18.1: Registrace vývojářského účtu

Vytvoření balíčku aplikace na publikování

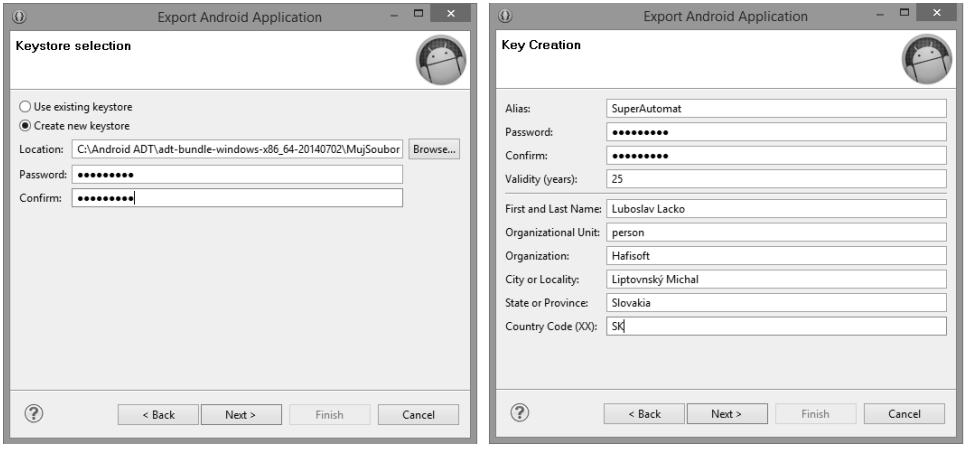
Upozornění: Aplikaci před publikováním důkladně otestujte.

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 Eclipse klepněte pravým tlačítkem na projekt v seznamu projektů a vyberte z místní nabídky položku **Android Tools → Export Signed Application Package**. Pokud nemáte vytvořený klíč, označte v dalším dialogu položku **Create new Keystroke** a vyberte cílové umístění souboru s klíči. 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 a soubor aplikace s příponou APK.



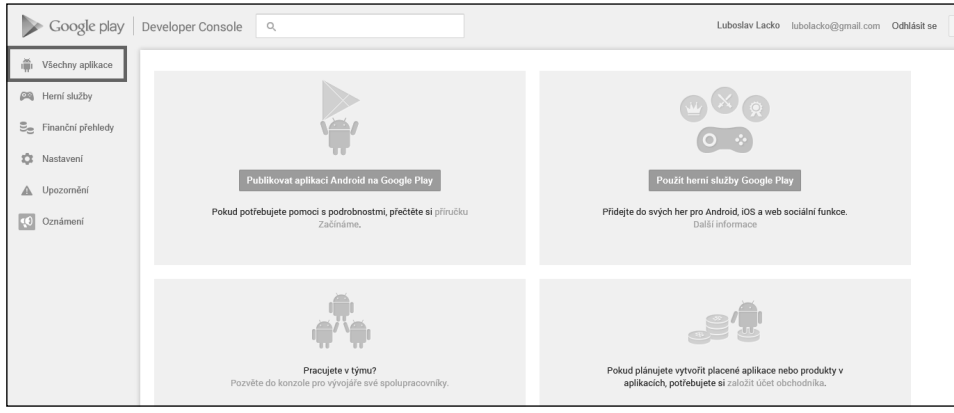
Obrázek 18.2: Vytvoření klíče na podepsání aplikace

Publikování aplikace

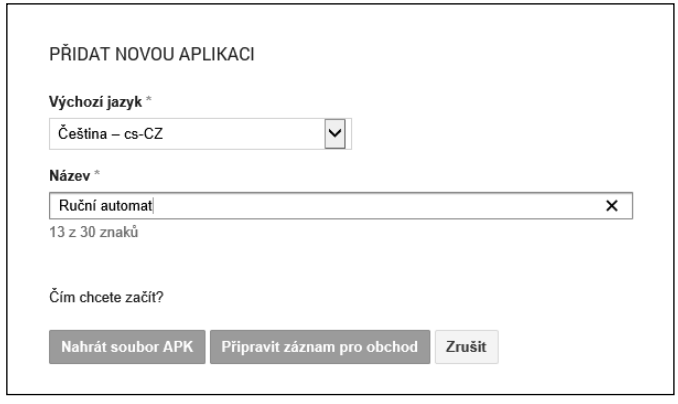
Pomocí Konzole pro vývojáře Google Play můžete nahrát svoje soubory APK, spravovat záznam v obchodě a nakonfigurovat integrované služby a funkce. Na stránce **Všechny aplikace** klepněte na tlačítko **Přidat novou aplikaci** a nahrajte soubor APK. Také je potřeba vyplnit údaje o aplikaci, takzvaný záznam aplikace pro obchod.

Aplikace i její záznam v obchodě po nahrání projdou 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 dotehdy, dokud nebudete 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.

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.



Obrázek 18.3: Rozhraní vývojářské konzole na publikování aplikací



Obrázek 18.4: Publikování aplikace

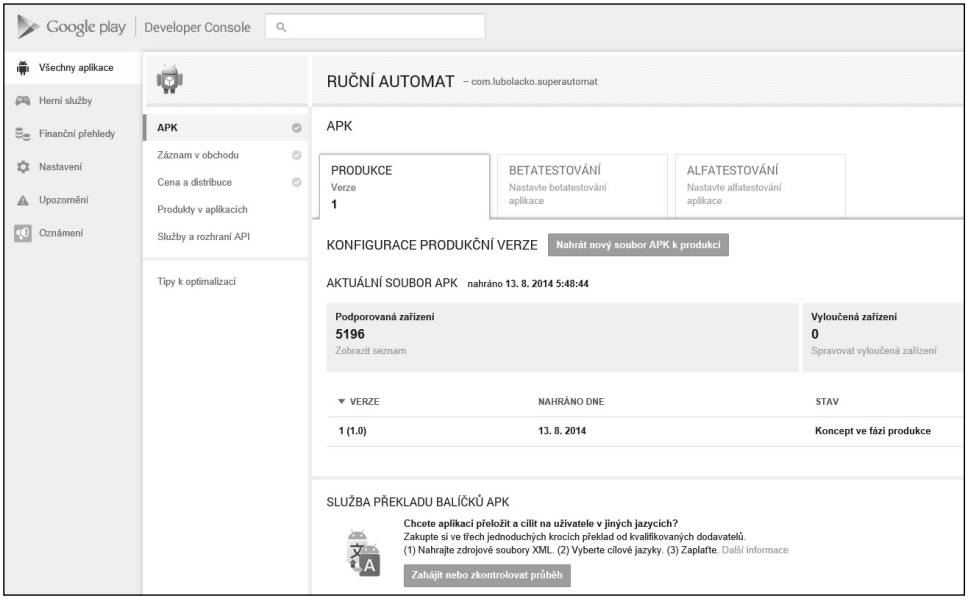
Upozorňujeme, že názvy balíčků pro soubory aplikací jsou jedinečné a trvalé. Vybírejte je proto s rozvahou. Názvy balíčků se nedají odstranit ani opětovně použít v budoucnosti. Nezapomínejte, že pokud jste ztratili svoje úložiště klíčů, budete muset aplikaci zveřejnit s novým názvem balíčku a novým klíčem. Měli byste také aktualizovat popis původní aplikace a zrušit její zveřejnění.

Příprava záznamu pro obchod

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.



Obrázek 18.5: Rozhraní na nahrání APK souboru

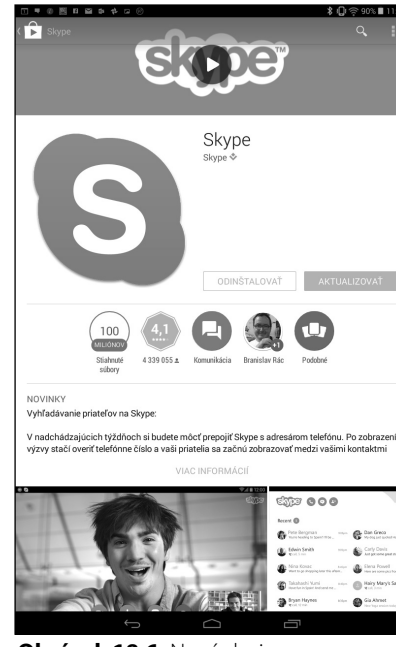
Textové popisy

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 se 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 jsou povoleny 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.

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.



Obrázek 18.6: Nový design aplikačního obchodu Google Play. Ikona ve vysokém rozlišení je nad tlačítkem Instalovat

> **Poznámka:** 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 je 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 aplikace v obchodě 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 pixelů není zapotřebí na uložení a zveřejnění záznamu v obchodě.



Obrázek 18.7: Přidání grafických prvků do záznamu aplikace pro obchod. Všimněte si možnosti Přidat překlady, kdy se popisy aplikace a obrázků budou automaticky překládat do vybraných jazyků pomocí služby Google Translator.

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. Poklud máte zařízení se systémem Android 4.4 Kitkat nebo vyšším, můžete pomocí příkazu prostředí `adb shell screenrecord` zaznamenat video přímo ze zařízení.



Tip: Pokud chcete omezit propagaci aplikace v internetových nebo mobilních službách Google, musíte na stránce **Cena a distribuce** zaškrtnout políčko **Deaktivace marketingu**.

KAPITOLA 19

Nové platformy – Android Wear a Google Glass

V této kapitole:
- Android Wear
- Google Glass

Android Wear

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. Hodně lidí přestalo nosit hodinky a na zjišťování času používají telefon, takže značný podíl interakce s telefonem představuje tento účel. Vývoj však jde po spirále a operativnost hodinek ke zjišťování informací upoutala pozornost tvůrců mobilních platforem.

Jednou z prvních vlaštovek, která se vyznačuje komplexností a univerzálností, je platforma Android Wear. Základem filozofie nové platformy je operativní poskytnutí relevantních informací při co nejjednodušším a nejintuitivnějším ovládání.

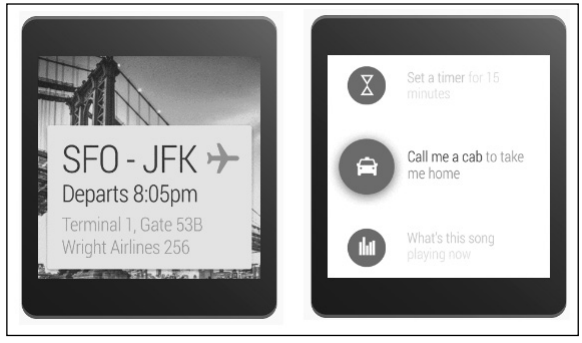
Hodinky vám poskytnou množství drobných, ale v daném okamžiku velmi potřebných informací. Popíšeme typický scénář při cestě letadlem. Děj začíná upozorněním na nutnost odchodu na letiště, sejmutím QR kódu při nástupu do letadla – samozřejmě z displeje hodinek, hodinky vám poskytnou informaci o hotelu 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. Scénář může pokračovat – zjistíte, ve které restauraci ve „walking distance" mají nejlepší pizzu, v předstihu si ji objednáte, hodinky vás navigují k restauraci, kde vám přinesou objednané jídlo, za které jediným klepnutím na displej hodinek zaplatíte.

Platforma Android Wear je příležitostí pro vývojáře na vytvoření užitečných informací pro nové typy zařízení.



Poznámka: 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, které 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 realizovaná 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.



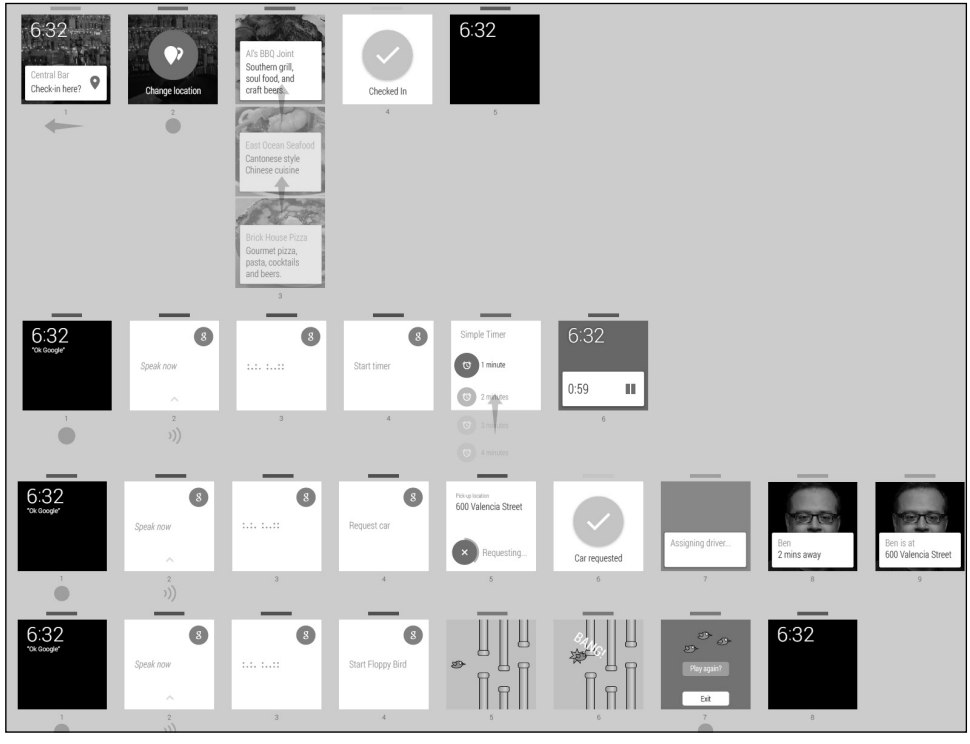
Poznámka: Aplikace pro platformu Android Wear může vytvářet karty a aplikovat je do kontextového streamu vždy, když mohou být s vysokou pravděpodobností uživateli užitečné.

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“, resp. „OK Glass“ u populárních brýlí, nebo poklepáním na pozadí domovské obrazovky či dotykovou plošku na rámu brýlí. Gestem, tedy posouváním prstu po obrazovce hodinek nebo po dotykové plošce brýlí, se zobrazí seznam navrhovaných hlasových příkazů, které mohou být aktivovány hlasem nebo poklepáním.

Z hlediska vývojáře takovéto gesto nebo hlasový příkaz aktivuje 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.

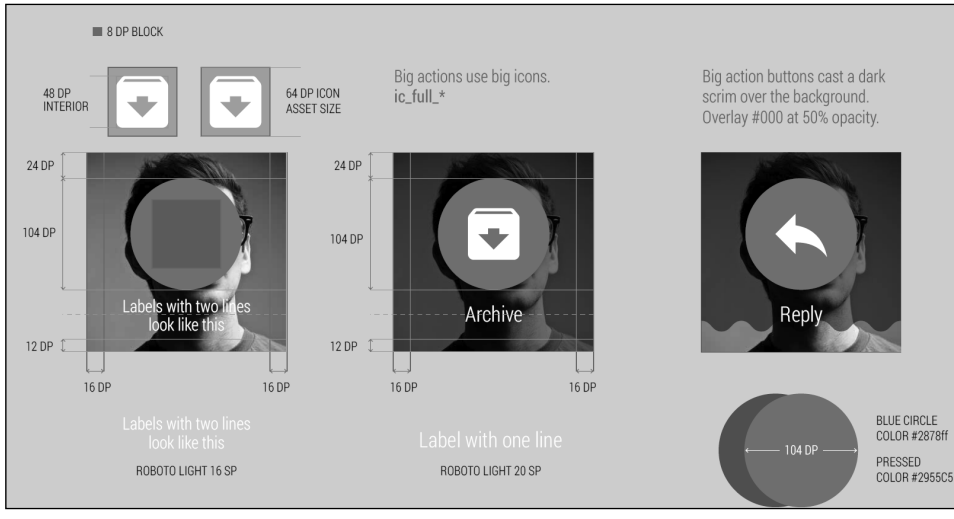


Obrázek 19.2: Princip ovládání aplikací by měl vycházet z ovládání operačního systému, aby se dosáhlo konzistentní uživatelské zkušenosti

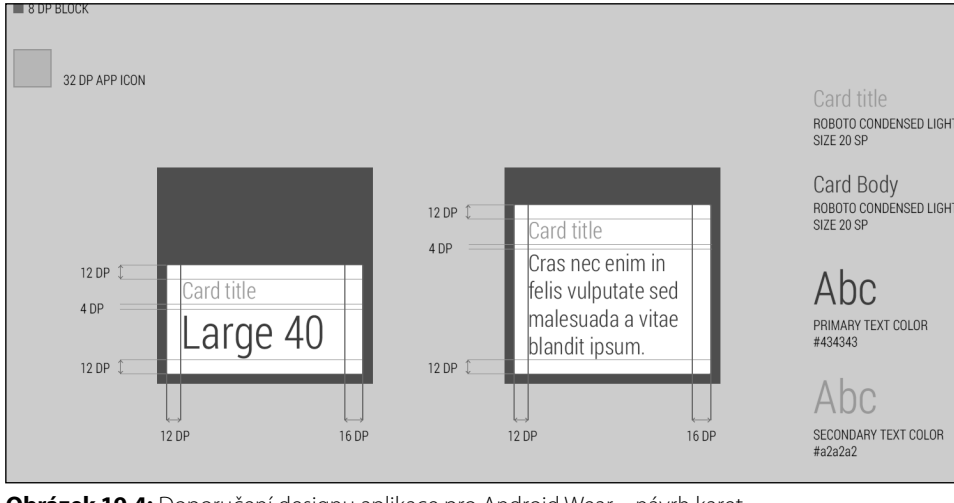
Výchozím stavem zařízení je domovská obrazovka, která zobrazuje buď obsah vztahující se k první kartě, nebo hodinky ukazující čas s takovým designem ciferníku, jaký si uživatel zvolil.

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 uživatel vybere 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 optimalizované na zobrazení v tomto stavu. Uživatelé mohou opustit tento režim klepnutím na obrazovku, nakloněním zařízení.



Obrázek 19.3: Doporučení designu aplikace pro Android Wear – ikony pro jednoduché a intuitivní ovládání

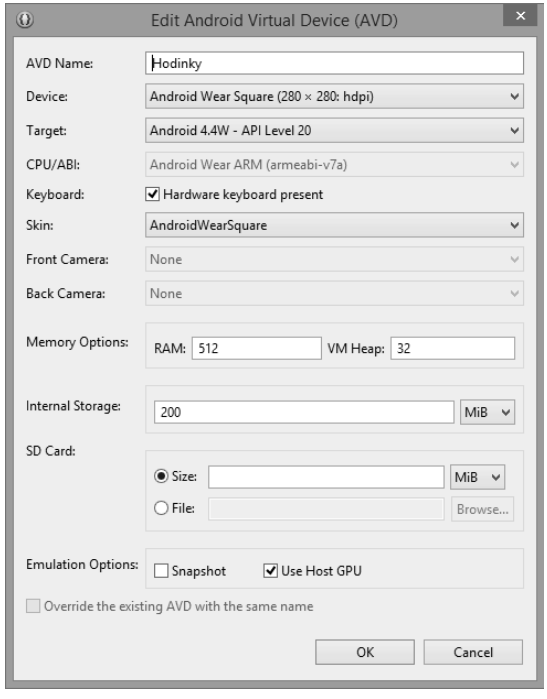


Obrázek 19.4: Doporučení designu aplikace pro Android Wear – návrh karet

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. K vytvoření emulátoru spusťte AVD Manager. Nastavte hodnoty pro jednotlivá pole:

- **Device** – Android Wear Round nebo Square
- **Target** – Android 4.4W - API Level 20
- **CPU/ABI** – Android Wear ARM (armeabi-v7a)
- **Keyboard** – Select Hardware keyboard present
- **Skin** – AndroidWearRound nebo AndroidWearSquare depending
- **Snapshot** – nenastavené
- **Use Host GPU** – nastavené



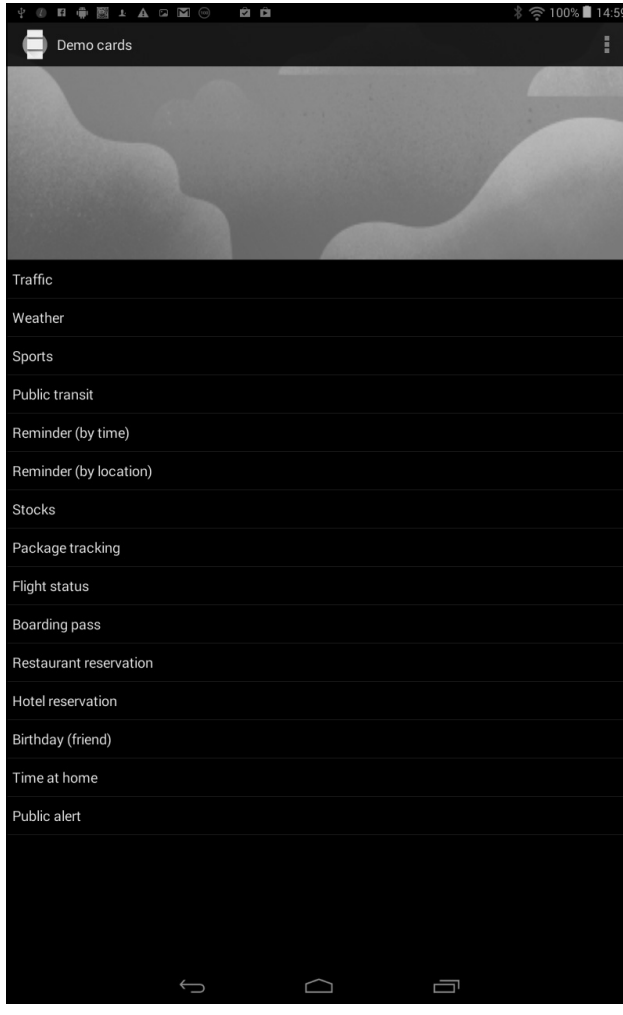
Obrázek 19.5: Nastavení parametrů pro virtuální zařízení Android Wear

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 s Androidem přes USB port. 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 Adntroid SDK, v podadresáři *\sdk\platform-tools* (*C:\Android ADT\adt-bundle-windows-x86_64-20140702\sdk\platform-tools*).

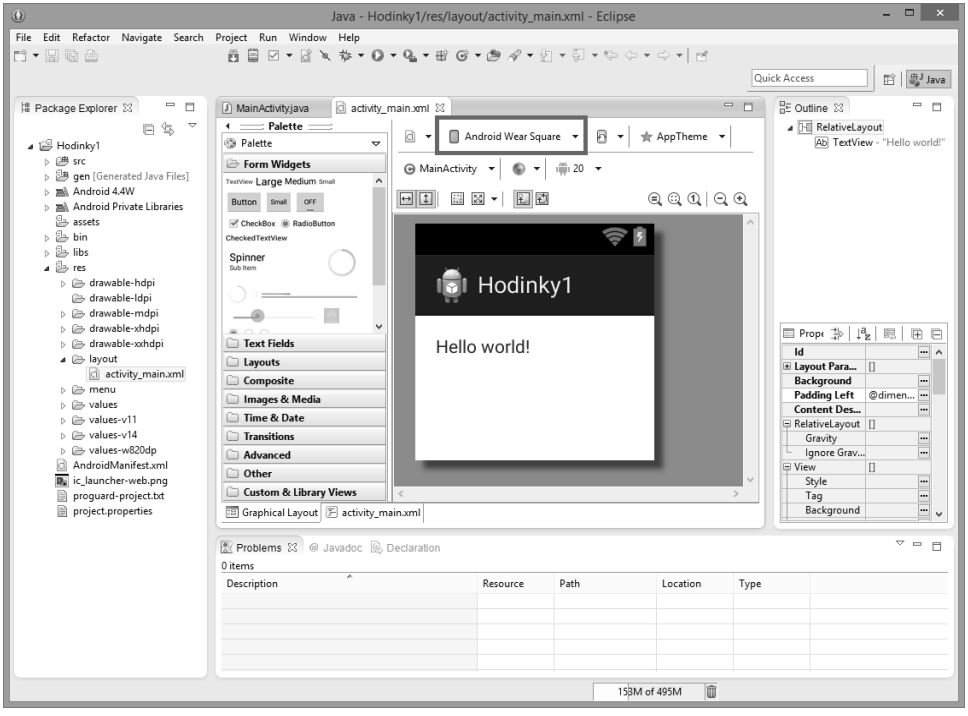


Obrázek 19.6: Aplikace Android Wear na posílání notifikací do emulátoru nebo wereable zařízení

Spusťte na připojeném zařízení aplikaci Android Wear a připojte se z ní k emulátoru. Po připojení vyberte v nabídce v pravém horním rohu funkci **Demo Cards**. Zobrazí se seznam notifikací, které můžete posílat do emulátoru.

Projekt aplikace pro Android Wear

Aplikace pro platformu se dají vytvářet a ladit v Eclipse i v Android Studiu. V Eclipse vytvoříte projekt standardním postupem, jako cílové API nastavte API 20: Android 4.4 (KitKat Wear). Při návrhu uživatelského rozhraní přepněte zobrazování layoutu na Android Wear (Square, nebo Round).



Obrázek 19.7: Projekt aplikace pro Android Wear v Eclipse

V předposlední verzi Eclipse byly dokonce šablony aplikací s aktivitou typu **Blank Wear Activity**, avšak v poslední verzi v době psaní rukopisu (z 2. 7. 2014) byla tato nabídka odstraněna. Změnila se i dokumentace k Android Wear, kde se popisuje vývoj aplikací v Android Studiu. Proto i náš další projekt vytvoříme v tomto vývojářském nástroji. Popis instalace a konfigurace Android Studia je v první kapitole.

Vytvořte nový projekt. Všimněte si, že dialog má momentálně tři sekce:

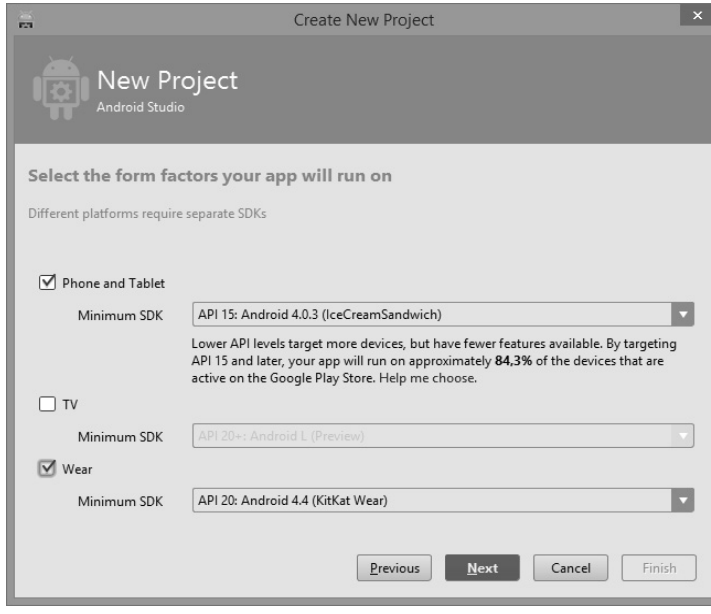
- **Phone and Tablet**
- **TV**
- **Wear**

Z toho logicky vyplývá, že z projektu můžete vytvořit aplikace pro tři sesterské platformy. My vytvoříme aplikaci, kterou bude možné spustit nejen na hodinkách nebo jiném zařízení platformy Android Wear, ale i na klasickém telefonu či tabletu.



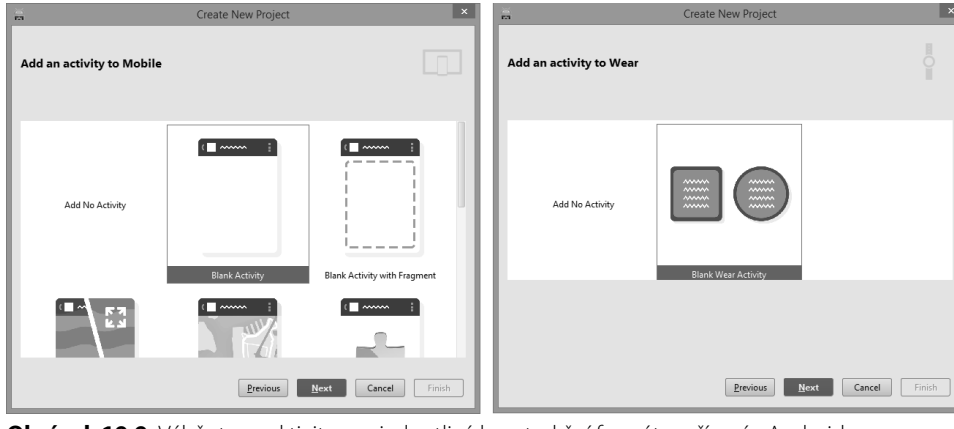
Poznámka: Vytvoření tandemu dvou projektů je logické. Uživatel bude mít s sebou wearable zařízení zároveň s telefonem a tato zařízení budou mezi sebou komunikovat.

Ponechejte označenou sekci **Phone and Tablet** a jako minimální API ponechejte API 15: Android 4.0.3 IceCream Sandwich. Označte platformu Wear a vyberte jako minimální API verzi API 20: Android 4.4 (KitKat Wear).



Obrázek 19.8: Parametry projektu pro platformu Android Wearable

V dialogu **Add an Activity** přidejte pro telefon a tablet aktivitu typu **Blank Activity** a následně pro hodinky aktivitu **Blank Wear Activity**.



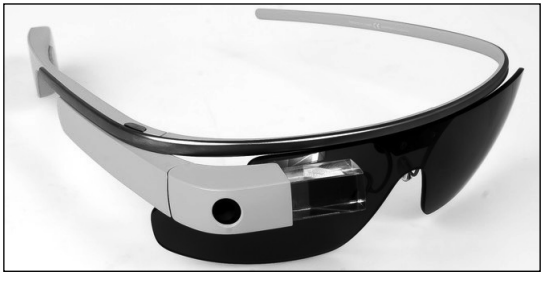
Obrázek 19.9: Výběr typu aktivity pro jednotlivé konstrukční formáty zařízení s Androidem

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 wearable zařízení. Jak rozdělit úlohy pro tento spolupracující tandem? Aplikace na telefonu bude řešit náročnější úlohy, například síťovou komunikaci nebo intenzivní zpracování údajů, nebo úlohy, které vyžadují intenzivní interakci s uživatelem. Na wearable zařízení se budou posílat aktuální notifikace o ukončení činnosti, připomínky a podobně.

Modul wear obsahuje po vytvoření projektu aktivitu „Hello World", která využívá `Watch ViewStub`. Ten přizpůsobí návrh uživatelského rozhraní aktivity podle toho, zda je obrazovka zařízení kulatá nebo hranatá.

Google Glass

Pravděpodobně nejzajímavějším zařízením využívajícím platformu Android byly v době psaní publikace brýle Google Glass. Jsou vybaveny procesorem, fotoaparátem a různými senzory, se kterými se setkáváme i u dnešních inteligentních telefonů. Disponují základním uživatelským rozhraním, které je jednoduché a dá se intuitivně ovládat pomocí hlasu nebo dotykové plochy na rámu brýlí. Zařízení je umístěno na rámu brýlí nad zorným polem uživatele.



Obrázek 19.10: Brýle Google Glass

K dispozici jsme měli vývojářskou edici Google Glass společně se sluneční clonou. Ve výbavě je plochý kabel USB na spojení s počítačem a nabíjení přes microUSB port a speciální sluchátko s připojením k portu USB. V běžném provozu sluchátko nepotřebujete, jelikož zvuk se dostává do ucha přes lebeční kost. Brýle mají nad pravým okem maličký displej a k dispozici je i připojení k Internetu přes Wi-Fi. První dojem je zajímavý, vizuálně se zdá, jako by byl displej ve větší vzdálenosti, než ve skutečnosti je.

Baterie Google Glass při šikovném používání vydrží na větší část dne. V předvoleném případě se vypnou, když si je sundáte z hlavy, a zapnou se, když se přes ně podíváte pod úhlem asi 30° nahoru. K ovládání se používá hlasový povel „OK Glass" následovaný možnostmi z nabídky nebo dotyková plocha. Google Glass je možné spárovat s chytrým telefonem, tabletem s Androidem, ale i s notebookem s Windows.

K využití možností brýlí Google Glass je potřeba účet Google s Google+ a Gmailem a zařízení (telefon nebo tablet) s operačním systémem Android nebo iOS, na které nainstalujete aplikaci

MyGlass. Aplikace si vyžádá potvrzení oprávnění přístupu k údajům o aktuální poloze a možnost přímého uploadu na Google+.

Technické údaje

Verze Explorer má implementovaný operační systém Android 4.4. Pro vývojáře je zajímavé hlavně rozlišení displeje 640 × 360 a rozlišení vestavěné kamery (5megapixelové fotografie a 720p video).

Procesor Texas Instruments OMAP 4430 SoC 1.2Ghz Dual (ARMv7) využívá 2GB RAM. Operační systém má k dispozici 16GB SSD, z toho má uživatel k dispozici 12 GB. Zařízení disponuje komunikačním rozhraním Wi-Fi 802.11b/g a Bluetooth. Senzorová výbava zahrnuje:

- Trojosý gyroskop
- Trojosý akcelerometr
- Trojosý magnetometr (kompas)
- Snímač okolního osvětlení (ambient light sensor)
- Snímač přiblížení k oku (proximity sensor)

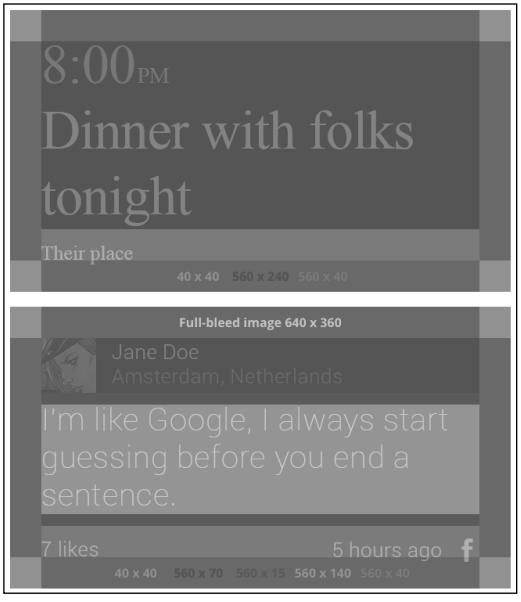
Zajímavý je přenos zvuku přes spánkovou kost.

Návrh designu aplikací

Podobně jako pro platformu Android Wear je i pro Google Glass potřeba brát při návrhu uživatelského rozhraní aplikace v úvahu specifika zobrazování a ovládání zařízení. Uživatel očekává zobrazování obvyklých věcí na obvyklých místech.



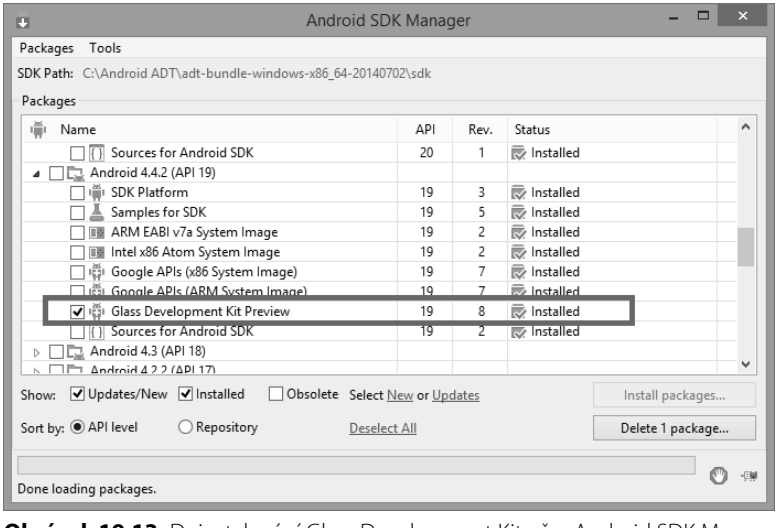
Obrázek 19.11: Standardní velikost a umístění polí na zobrazování obsahu



Obrázek 19.12: Příklad návrhu uživatelského rozhraní

Vývoj aplikací

Google Glass v době psaní publikace ještě nebyly v komerčně dostupné verzi. Měli jsme k dispozici verzi Explorer pro vývojáře. Všechny aktuální informace k vývoji aplikací najdete na webu *https://developers.google.com/glass/*.



Obrázek 19.13: Doinstalování Glass Development Kit přes Android SDK Manager

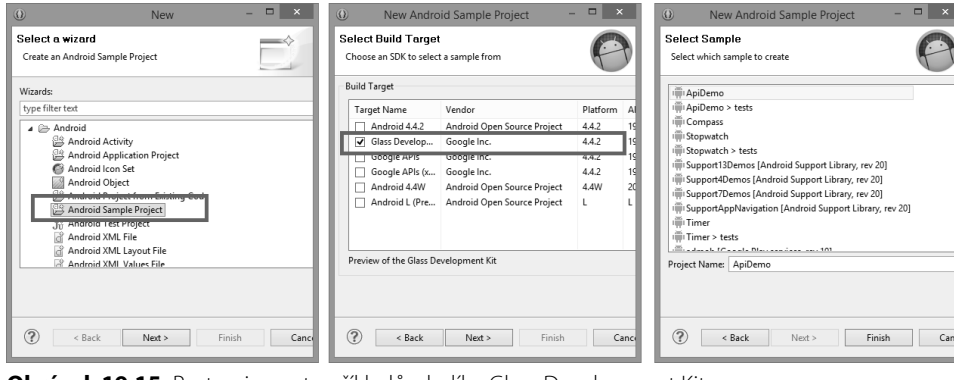
K vývoji aplikací je potřeba do vývojového prostředí doinstalovat Glass Development Kit (GDK). V době psaní publikace bylo dostupné pro Android 4.4.2 (API 19).

Abyste mohli spouštět aplikace na reálném zařízení, potřebujete brýle „odemknout", aby se daly připojit přes USB k vývojářskému počítači a ladit přes ADB. Postupně se navigujte přes obrazovky **Settings**, **Device info** na volbu **Turn on debug**. Tuto volbu aktivujte.

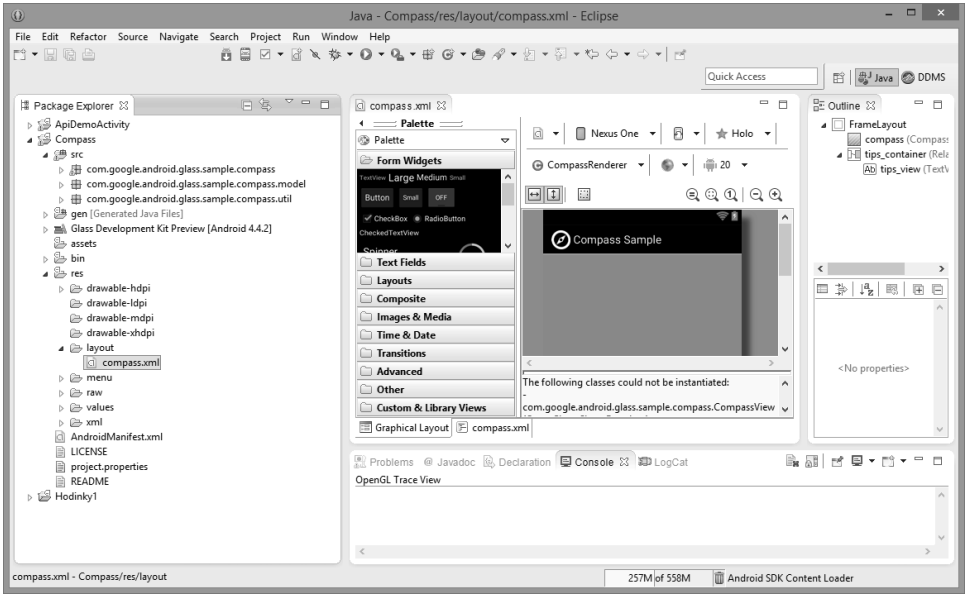


Obrázek 19.14: Postup odemknutí brýlí Google Glass

Nejlépe princip vývoje pochopíte na příkladech. Příklad importujete pomocí tlačítka **New** na panelu nástrojů vývojového prostředí Eclipse (ne nabídka) a pak vyberte položku **Android Sample Project**. V dalším dialogu označte položku **Glass Development Kit** a importujte příklady, které vás zajímají. Doporučujeme projekty Compass, Stopwatch, případně Timer.



Obrázek 19.15: Postup importu příkladů z balíku Glass Development Kit

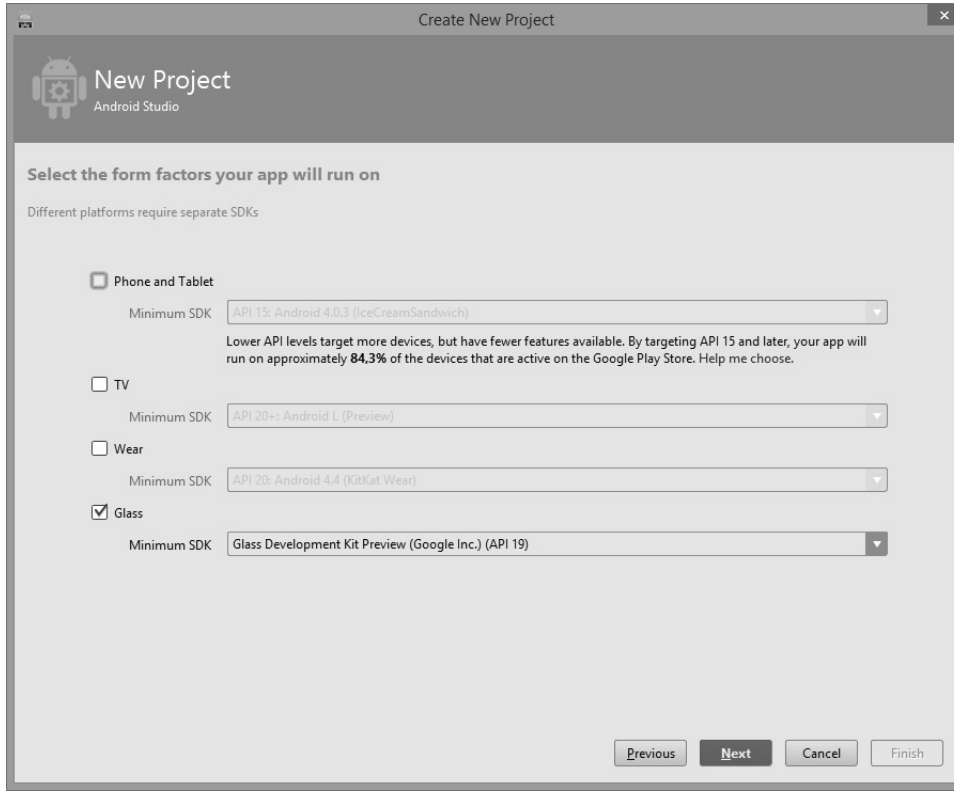


Obrázek 19.16: Projekt aplikace pro Google Glass

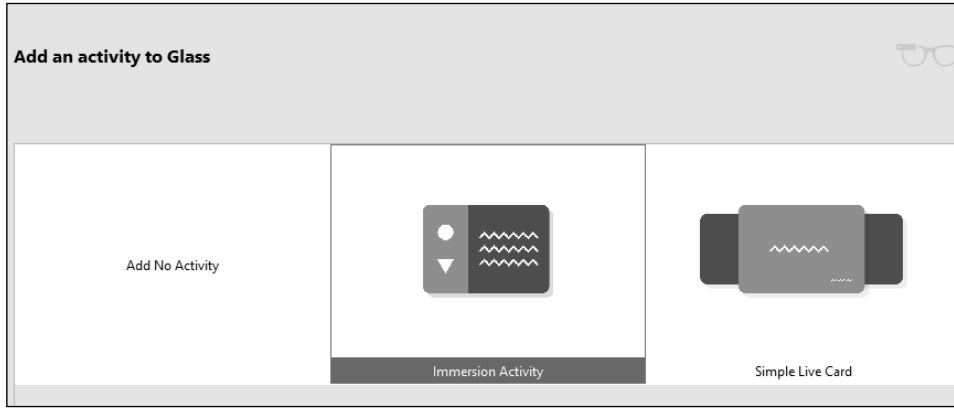
Projekt pro Google Glass můžete po doinstalování Glass Development Kit (GDK) pro Android 4.4.2 (API 19) vytvářet i v Android Studiu. V době psaní publikace bylo v beta verzi. K dispozici byly dvě připravené šablony pro hlavní aktivitu **Immersion Activity** a **Simple Live Card**.



Poznámka: Jelikož se na rozdíl od Wear nejedná o produkční verzi, může se ve finální verzi produktu ledasco změnit.



Obrázek 19.17: Vytvoření projektu v Android Studiu



Obrázek 19.18: Nabídky typů aktivit

Rejstřík

#

3D, 320
3gp, 335
9-patch, 95

A

ACCESS_COARSE_LOACTION, 375
ACCESS_COARSE_LOCATION, 363, 366
ACCESS_FINE_LOCATION, 363, 366, 375
ACCESS_NETWORK_STATE, 375
ACID, 244
ACL, 211
Action, 72
Action Bar, 143
– přidání widgetu, 150
– přizpůsobení, 147
– rozdělení lišty, 147
ACTION_BATTERY_LOW, 225
ACTION_BOOT_COMPLETED, 225
ACTION_CAMERA_BUTTON, 225
ACTION_CANCEL, 155
ACTION_DOWN, 155
ACTION_MEDIA_BUTTON, 225
ACTION_MEDIA_EJECT, 225
ACTION_MEDIA_MOUNTED, 225
ACTION_MEDIA_UNMOUNTED, 225
ACTION_MOVE, 155
ACTION_POINTER_DOWN, 155
ACTION_POINTER_UP, 155
ACTION_SCREEN_OFF, 225
ACTION_SCREEN_ON, 225
ACTION_UP, 155
Activities, 93
activity, 64, 210
Activity Manager, 61
Activity Name, 83
Activity.startActivity, 118
adaptivní jas, 53
ADB (Android Debug Bridge), 59
ADT (Android Development Tools), 15
AIDL, 390
akce, 72
akcelerometr, 349
– filtrované údaje, 354
– zobrazení údajů, 351
aktivita, 64, 93
– hlavní, 80, 245, 255, 262
– konfigurace, 30
– na editaci záznamu, 259, 264
– na popředí, 66
– na přidání záznamu, 258, 263
– na zapouzdření fragmentů, 218
– navigace, 148
– pozastavená, 66
– spuštění, 114
– výběr typu, 82
– vytvoření, 29, 118
– zastavená, 67
– životní cyklus, 65–66
aktualizace
– údajů, 397
– záznamů, 252
alarm, 169, 174
– ověření, 175
– vytvoření, 174
Alarm Timers, 59
AlarmManager, 177
AlertDialog, 197
anatomie projektu, 83
Android, verze, 50–52
Android 5.0 Lollipop, 52
Android Activity, 118
Android Runtime, 60
Android Studio, 15, 35
Android TV Activity, 81
Android Virtual Devices, 22
Android Wear, 449
– emulátor, 453
– projekt, 455
Android Wear Round, 24
Android Wear Square, 24
Android XML File, 301
AndroidManifest.xml, 71, 90, 225
animace, 293, 314
– čas interpolace, 320
– doba trvání, 320

– chování a počet opakování, 320
– klasická, 314
– seskupování, 321
– Tween, 317
– zpoždění obnovování, 321
AnimationDrawable, 314–315
AnimatorUpdateListener, 321
antialiasing, 305
API, verze, 90
API 21, 52
API Access, 374
aplikace, 61
– design, 53
– ikona, 79
– ladění, 121
– název, 78
– oprávnění, 61
– připojení k mobilní službě, 423
– publikování, 441, 443
– spuštění na reálném zařízení, 34
– spuštění v emulátoru, 32
– typ, 445
– vizuální návrh, 41
– základní principy, 77
– základní součásti, 64
aplikační
– kód, 86
– logika, 115, 137, 262
– manifest, 90
App Tethering, 43
Application Framework, 60
Application Name, 78
Application Not Responding, 74
architektura, 49, 58
ArrayAdapter, 180
ArrayList, 288
AsyncTask, 75, 396
AsyncTaskLoader, 76
asynchronní
– programování, 74
– třída, 283
– zpráva, 223
AudioManager, 327, 329
Azure Notification Hub, 411

B

balíček aplikace, 442
barometr, 349
BaseAdapter, 184
BaseExpandableList, 195
beforeTextChanged, 139–140
bezpečnost, 61
BitmapDrawable, 293
Blank Activity, 81, 93
Blank Wear Activity, 455
bound, 384
Broadcast, 223, 383
Broadcast Receiver, 223
– registrovaný dynamicky, 227
– registrovaný v aplikačním manifestu, 225
Broadcast Receivers, 65
BroadcastReceiver, 176–177, 226
BufferedReader, 275
Bundle, 193
Button, 88, 93
buttonClicked, 115
BYOD, 406

C

C#, 11
C2DM, 407
CALL_PHONE, 62
Callback, 340
Camera, 327, 338, 371
CameraCallback, 341
Canvas, 293
– dynamické vykreslování, 310
– kreslení dotykem, 306
– vykreslování, 304
– zámek, 311
CardView, 55–56
Category, 72
Clipart, 80
close, 251
cloudová služba, 405
CMS, 244
colorDrawable, 293
commit, 232
Component, 73
Content providers, 65
ContentProvider, 391
ContentResolver, 391, 397
Context, 249
CREATE TABLE, 246
Cue cards, 451
Cursor, 191
cursor query, 253
CursorLoader, 396

Č

číslo, množné, 89
čtení stavu telefonu, 229

D

Dalvik Debug Monitor, 165
data, 72
databáze
– SQLite, 244
– vložení záznamu, 251
databázová tabulka, 250
DataModel, 249

datový
– model, 248
– typ, 231
DDMS (Dalvik Debug Monitor Server), 32
Debug, 122
definice
– objektů ve zdrojích, 86
– rozložení prvků, 107
– uživatelského rozhraní, 84
delete, 252
design aplikace, 53
DetailActivity, 221
DetailFragment, 216
detailní informace, 216
detekce, gest, 160
dimens.xml, 295
Disk Google, 76
DisplayMetrics, 128
dočasný soubor, 243
doInBackground, 298
DOM (Document Object Model), 278
doporučení, 406
dopředná kompatibilita, 57
dotyk, 154
– vícenásobný, 156
drawable-hdpi, 206
drawBitmap, 304, 308, 311
drawColor, 304
drawOval, 304
drawpoints, 304
drawText, 304
DROP TABLE, 247
Dropbox, 76
dynamické vykreslování, 307, 310

E

Eclipse, 15, 17
– ADT, 19
edit, 233
editace záznamu, 245, 259, 264
EditText, 88, 93
EditView, 264
efekt, zvukový, 328
Embarcadero RAD Studio XE6, 40
Empty Activity, 81–82
emulátor, 32
– Android 4.4 Wearable, 24
– Android Wear, 453
– vytvoření, 22
END_DOCUMENT, 281
END_TAG, 281
ES File Explorer, 243
execSQL, 250
ExpandableListView, 195
External Storage, 76
externí
– paměť, 240
– úložiště, 237
Extras, 73

F

Facebook, 434
– přihlášení, 436
FacebookSDK, 436
File Explorer, 166, 239
FileInputStream, 238
FileOutputStream, 238
fill_parent, 190
findPointerIndex, 155
findViewById, 116
finish, 71
FireMonkey, 41
firstChild, 279
Flags, 73
fotografie, 338, 343
– externí aplikace, 346
fragment, 209–211
– paused, 214
– pozastavený, 214
– přidání, 209
– resumed, 214
– spuštěný, 214
– stopped, 214
– zastavený, 214
– životní cyklus, 212
FragmentManager, 209
FrameLayout, 100, 219
frekvence, procesoru, 128
Full HD, 94
funkcionalita, 143

G

Game Maker Studio, 43
GCM (Google Cloud Messaging), 405
– architektura, 410
GDK (Glass Development Kit), 461
generování zdrojových dat, 270
geomagnetický senzor, 349
gesto, 154
– detekce, 160
– rozpoznání, 168
– vlastní, 164
Gesture Sensor, 349
GestureDetector, 160
Gestures Builder, 165
GET_TASKS, 63
getActionIndex, 155
getActionMasked, 155
getAll, 232

getBestProvider, 364
getBoolean, 232
getContentResolver, 391
GetElementByTagName, 279
getEventType, 281
getExternalCacheDirs, 243
getExternalFilesDirs, 243
getExternalStorageState, 240
getExtras, 70
getHolder, 311
getInclination, 357
getListView, 186
getObbDirs, 243
getOrientation, 357
getPointerCount, 155
getPointerId, 155
getPreferences, 231
getReadableDatabase, 251
getResources, 286
getResponseCode, 278
getRotationMatrix, 357
getString, 232, 275
getSystemService, 174, 350
getter, 184
getText, 281
getWritableDatabase, 251
getX, 155
getY, 155
Google Cloud Console, 408
Google Cloud Messaging for Android, 407
Google Glass, 449, 457
– design aplikací, 458
– vývoj aplikací, 459
Google Maps, 370
– API klíč, 372
Google Play, 441
Google Play Services, instalace, 371
Google USB Driver, 35
Google Waller Merchant Center, 441
Google-provided GCM Connection Server, 410
GoogleMap, 371
GPS, 362–363, 368
GPS navigace, 59
GPS_PROVIDER, 364
GradientDrawable, 293
grafický
– návrh, 85
– tvar, 300
grafika, 293
– hlavní, 446
– propagační, 447
Graphical Layout, 85, 110
GridLayout, 101, 103, 272
GSM, 59, 368
gyroskop, 349

H

Hall Sensor, 349
HashMap, 252
hierarchická struktura, 195
historie verzí, 50
hlasitost ovládání, 328
hlavní
– aktivita, 80, 245, 255, 262
– grafika, 446
hovor, indikace, 229
hra, vytvoření, 44
HttpResponse, 275
HttpURLConnection, 278
hudba na pozadí, 386

CH

CheckBox, 93

I

ikona, 29
– aplikace, 79
– ve vysokém rozlišení, 446
ilustrace, 321
ImageView, 93, 170, 241, 297, 322
import projektu, 39
indikace příchozího hovoru, 229
informace o zařízení, 126
initLoader, 396
Input Type, 135
InputStream, 275, 286
INSERT, 246
instalace SDK, 17
integer, 206
Intent, 69, 118
Intent Filter, 71
IntentFilter, 71
IntentService, 76
interakce
– s databází, 244
– s uživatelem, 143
Internal Storage, 76
INTERNET, 366
Internet, povolení přístupu, 297
interní
– paměť, 238
– úložiště, 237
INTERVAL_DAY, 175
INTERVAL_FIFTEEN_MINUTES, 175
INTERVAL_HALF_DAY, 175
INTERVAL_HALF_HOUR, 175
INTERVAL_HOUR, 175

J

jas, adaptivní, 53
Java, 11
– instalace, 15
Java 8, 16
Java EE, 16
Java ME, 16
Java SE, 16
JavaCard, 16
jazyk, 90, 445
JDK (Java Development Kit), 15
JSON, 217, 269, 274

K

Key Hashes, 435
keytool.exe, 373
KILL_BACKGROUND_PROCESSES, 63
knihovna pro dopřednou kompatibilitu, 57
kód, aplikační, 86
kompas, magnetický, 357
kompatibilita, dopředná, 57
kompatibilní projekt, 214
kompilace části kódu, 53
konfigurace
– ověření, 26
– projektu, 28
– SDK, 17
kontakty, 392
– načítání na pozadí, 395
kontejner
– na rozmístění prvků, 95
– vnoření, 105
kontextový stream, 450
krátký popis, 445
kreslení dotykem, 306

L

ladění aplikace, 121
Layout Name, 83
layout_gravity, 96
layout_height, 190
layout_weight, 96, 111
layout_width, 190
Layouts, 96
LENGTH_LONG, 113
LENGTH_SHORT, 113
LibC, 60
Libraries, 60
Light Sensor, 349
LinearLayout, 96, 105, 114, 182
Linux Kernel, 59
ListActivity, 179
– formátované zobrazení položek, 182
– s kontextovým menu, 186
ListAdapter, 180
listener, 125
ListView, 55, 188, 204
LoaderCallback, 396
LocalBroadcastManager, 227
Location, 363
Location Manager, 61
LocationProvider, 363
lockCanvas, 311
logika, aplikační, 115, 137, 262
lokalizační služba, 362

M

magnetický kompas, 357
main, 65
MainActivity, 83, 113
manifest, aplikační, 90
mapa
– typy, 370
– zobrazení polohy, 370
MapFragment, 371
mapová služba, 349, 362
mapy, zobrazení, 378
Marker, 371
match_parent, 109
MEDIA_MOUNTED, 240
MEDIA_MOUNTED_READ_ONLY, 240
MEDIA_REMOVED, 240
MediaPlayer, 327, 332
MediaRecorder, 327, 334
Messenger, 390
Microsoft Azure, 410
Microsoft Azure Management Portal, 411
Microsoft Azure Mobile Services, 423
miniaktivita, 210
Minimum Required SDK, 144
množné číslo, 89
Mobile Services, 426
mobileservices, 415
mobilní služba, 423, 432
model, datový, 248
modifikátor, 89
Mono for Android, 43
možnosti,
– ukládání údajů, 76
– pro vývojáře, 26
MP3, 386
multimédia, 327
– aktivní, 327
– pasivní, 327
Multitouch, 154
MySQL, 244

N

načítání,
– na pozadí, 395
– údajů, 274
nahrávání, 335
– zvuku, 334
náklonoměr, 361
nákupy v aplikaci, 441
nástroje pro vývoj, 15
navigace, 143
– na jinou aktivitu, 148
navigační tlačítko, 55
návrh
– grafický, 85
– uživatelského rozhraní, 92, 132
– vizuální, 41
název
– aplikace, 78, 445
– javového balíčku, 78
– projektu, 78
newView, 394
next, 282
NotifActivity, 172
Notification Manager, 61
notificationhubs, 415
NotificationManager, 172
notifications, 415
Notifications Hubs, 412
notifikace, 169
– posílání, 421
– poslání, 171
– přijímání, 413
– vysílání, 413
notify, 75
nový projekt, 28
NuGet, přidání balíčku, 422
numberdecimal, 135

O

objekt
– seznam, 179
– zvětšování, 318
obrázek
– 9-patch, 95
– programové zobrazení, 295
– z Internetu, 296
– zobrazení, 294
obrazovka, rozlišení, 94
obsah, poskytování, 391
obsluha události, 113
odběr, 442
ohodnocení, 123
onAccuracyChanged, 350, 352
onActivityCreated, 213, 217
onActivityResult, 71
onAttach, 213
onBind, 384
onCompletion, 388
onContextItemSelected, 187
onContextMenuItem, 144
onCreate, 68, 138, 149, 248, 384
onCreateContextMenu, 144, 186
onCreateOptionsMenu, 144, 151
onCreateView, 213, 216
onDestroy, 69, 385
onDestroyView, 213
onDetach, 213
onDraw, 308
OneDrive, 76
onFling, 163
onItemClick, 189, 262
onItemLongClick, 181
onListItemClick, 181
onLocationChanged, 363
onOptionsItemSelected, 146, 149
onOptionsMenuItem, 144
onPause, 68, 212
onPostExecute, 75, 276, 298
onPreExecute, 75
onProviderDisabled, 363
onProviderEnabled, 363
OnRatingChangeListener, 125
onReceive, 223
onResume, 68
onSaveInstanceState, 138, 214
onSensorChanged, 350, 352
onStart, 68
onStartCommand, 385
onStatusChanged, 363
onStop, 69
onTextChanged, 139
onTouch, 157
onTouchEvent, 160, 306
onUnbind, 385
onUpgrade, 248
onWindowFocusChanged, 324
openFileInput, 238
openFileOutput, 238
OpenGL ES, 60
openRawResources, 286
operační systém, 49
oprávnění pro aplikaci, 61
Oracle, 244
Ordered Broadcast, 225
orientace zařízení, 90
orientation, 96
osvětlení, 361
OutputFormat, 335
OvalShape, 300, 302
ověření konfigurace, 26
ovládání hlasitosti, 328

P

Package Explorer, 86
Package Manager, 61
Package Name, 78
PacMan, 44
paměť, externí, 240
Parcelable, 70
PathShape, 300
pause, 328
PictureCallback, 343
Plain Text, 135
plocha, 68
PNG, 295
počítadlo, 160
pohled, 92
pole řetězců, 88
poloha
– zařízení, 363, 366
– zobrazení, 370
popis, textový, 445
posílání notifikací, 421
poskytování
– obsahu, 391
– údajů, 400
poskytovatel obsahu, 65
poslání notifikace, 171
pracovní prostor, 27
PreferenceFragment, 234
PreferenceScreen, 235
prepare, 327, 334
PRIMARY KEY, 246
procesor, frekvence, 128
prodej, aplikace, 441
programování, asynchronní, 74
ProgressBar, 93
Project Name, 78
projekt
– anatomie, 83
– ikony, 29
– import, 39
– název, 78
– vytvoření, 77
propagační
– grafika, 447
– video, 448
Property Animation, 320–321, 323
propojení
– na vyhledávání, 152
– s databází, 251
prostředí, vývojové, 31
Proximity Sensor, 349
prvky, rozmístění, 95
předávání
– údajů, 70
– výsledků, 70
přehrávání
– hudby na pozadí, 386
– videa, 332
přidání záznamu, 245, 258, 263
přihlášení k Facebooku, 436
příchozí hovor, 229
přijímání notifikací, 413
připojení
– k sociální síti, 432
– ke cloudovým službám, 405
přístup k Internetu, 297
přizpůsobení rozlišení obrazovky, 94
publikování aplikace, 441, 443
putExtra, 116
putInt, 233

Q

query, 397

R

RadioButton, 93
RandomAccessFile, 128
RatingBar, 123
READ_CALENDAR, 63
READ_CONTACTS, 63
READ_GSERVICES, 375
READ_LOGS, 63
reálné zařízení, 25, 34
RECEIVE_BOOT_COMPLETED, 63, 225
receiver, 178, 226
rectangle, 301
RectShape, 300
RecyclerView, 55–56
region, 90
registerReceiver, 224
registrace vývojářského účtu, 441
reklama v aplikaci, 441
RelativeLayout, 98, 192, 203, 296
relativní umisťování, 99
release, 328, 334
Resource Chooser, 133
resources, 86
REST API, 274
Ringtone, 327
RingtoneManager, 327
rooms, 46
rozhraní, uživatelské, 84, 131
rozlišení
– displeje, 90
– přizpůsobení, 94
rozložení prvků, 107
rozmístění prvků, 95
RTC_WAKEUP, 175

Ř

řetězec
– pole, 88
– textový, 88

S

ScrollView, 100, 106
SDK (Android Software Development Kit), 15
– instalace, 17
– konfigurace, 17
searchable, 150
SearchView, 151
SeekBar, 140
seekTo, 328
SELECT, 244, 247, 253
SEND_SMS, 62
sendBroadcast, 226
sendOrderedBroadcast, 225
SensorEventListener, 350
senzor, 349
– osvětlení, 361
– přiblížení, 349
– základní údaje, 350
Serializable, 70
serverový klíč, 409
services, 64
setAntiAlias, 305
setAudioSource, 334
setColor, 305
SetContentView, 305
setDataSource, 327
setDisplayHomeAsUpEnabled, 149
setGravity, 107
setHeaderTitle, 186
setImageURI, 241
setInexactRepeating, 174
setOnItemClickListener, 262
setOnLoadCompleteListener, 329
setOrientation, 107
setOutputFormat, 334
setPreviewDisplay, 340
setResult, 71
setStrokeWidth, 305
setter, 184
setTextSize, 305
setVideoSource, 334
seznam
– objektů, 179, 217
– objektů s obrázky, 202
– oprávnění, 62
– výběr více položek, 192
Shape, 300
ShapeDrawable, 293
Shared Preferences, 76
SharedPreferences, 231, 234
showAsAction, 145, 148
ShutterCallback, 341
Simple API Access, 407
síť, sociální, 432
síťový úložný prostor, 76
skica uživatelského rozhraní, 132
skupina pohledů, 93
sleep, 74, 308
služba, 383
– spuštěná, 384
– vázaná, 384
– životní cyklus, 384
služby, 64
snímek obrazovky, 446
sociální síť, 405, 432
soubor, dočasný, 243
SoundPool, 327, 329
SparseArray, 400
specifikace verze, 79
Spinner, 93
splitActionBarWhenNarrow, 147
sprite, 44
spuštěná služba, 384
spuštění
– jiné aktivity, 114
– v emulátoru, 32
SQL, 11
SQL Server, 244
SQLite, 244
– databáze, 76
SQLiteOpenHelper, 248
start, 328, 334
START_DOCUMENT, 281
START_FLAG_REDELIVERY, 386
START_GLAF_RETRY, 386
START_NOT_STICKY, 385, 389
START_REDELIVER_INTENT, 385
START_STICKY, 385
START_TAG, 281
startActivity, 91
startActivityForResult, 70, 224
started, 384
startForeground, 387, 389
startService, 384
statické údaje, 188
stop, 328, 334
stream, kontextový, 450
StringBuilder, 275
strings.xml, 87
struktura, hierarchická, 195
Surface Manager, 60
surfaceCreated, 311
surfaceDestroyed, 311

surfaceChanged, 311
SurfaceView, 307, 310, 332
SYSTEM_ALERT_WINDOW, 63
systémová událost, 223

Š

šířka displeje, 90
škálování zařízení, 23

T

Tabbed Activity, 81
TableLayout, 101
TableRow, 101, 255
tabulka, databázová, 250
TagName, 279
takePicture, 344
technické údaje, 458
TEXT, 281
textový
– popis, 445
– řetězec, 88
TextView, 87, 93, 124, 133
THE REST QUERY, 272
Thread, 308
TimeInterpolator, 321
Toast, 113
– s vlastním designem, 169
toString, 203, 288
TransitionDrawable, 314
trendy, 406
Tween, animace, 317
TweenAnimation, 314
typ
– aplikace, 445
– datový, 231
– hlavní aktivity, 80
Type, 73
TYPE_ACCELEROMETER, 350–351
TYPE_AMBIENT_TEMPERATURE, 350
TYPE_GRAVITY, 350
TYPE_GYROSCOPE, 350
TYPE_LIGHT, 350
TYPE_LINEAR_ACCELERATION, 350
TYPE_MAGNETIC_FIELD, 350
TYPE_ORIENTATION, 350
TYPE_PRESSURE, 350
TYPE_PROXIMITY, 350
TYPE_RELATIVE_HUMIDITY, 350
TYPE_ROTATION_VECTOR, 350
TYPE_TEMEPRATURE, 350

U

účet, vývojářský, 441
údaje
– aktualizace, 397
– načtení, 274
– předávání, 70
– technické, 458
– uložení, 76, 231
– v kódu, 190
– výběr, 253
událost
– obsluha, 113
– systémová, 223
uložení
– do externí paměti, 240
– do interní paměti, 238
– do souboru, 237
– dočasných souborů, 243
– skóre, 232
– údajů, 76, 231
úložiště
– externí, 237
– interní, 237
unikátní identifikátor, 246
unlockCanvasAndPost, 311
unregisterReceiver, 227
update, 252
určení polohy zařízení, 363
URLConnection, 278
úroveň hlasitosti, 328
uses-permission, 297
uživatel, interakce, 143
uživatelské rozhraní, 84, 92, 131

V

ValueAnimator, 321
vázaná služba, 384
verze, 50
– API, 90
– specifikace, 79
vícenásobný dotyk, 156
video
– propagační, 448
– přehrávání, 332
View System, 61
ViewFlipper, 161–162
ViewGroup, 95
ViewGroups, 93
Views, 92
vizuální návrh, 41
vlastní gesto, 164
vložení záznamu do databáze, 251
vnoření kontejnerů, 105
výběr
– pracovního prostoru, 27

– údajů, 253
– více položek, 192
vyhledávání, 153
vykreslování
– dynamické, 307, 310
– grafických tvarů, 300
– na Canvas, 304
vymazání záznamu, 252
vysílání notifikací, 413
výsledek, předávání, 70
vytvoření
– aktivity, 118
– balíčku aplikace, 442
– mobilní služby, 424
– nového projektu, 28
– projektu, 77
vývojářský účet, 441
vývojové prostředí, 15, 17, 31

W

wait, 75
Wakelocks, 59
Wi-Fi, 368
widget, 92
– přidání, 150
Window Manager, 61
withEndAction, 324
WOEID, 270
Workspace, 27
wrap_content, 109
WRITE_CALENDAR, 63
WRITE_CONTACTS, 63
WRITE_EXTERNAL_STORAGE, 63, 375
WRITE_EXTERNAL_STORGE, 240
WRITE_SMS, 62

X

Xamarin MonoTouch, 43
XML, 11, 269, 278
– návrh, 294
– statická data, 286
XmlPullParser, 281
XMLPullParserHandler, 289

Z

základní součásti aplikace, 64
záměr, 69
– explicitní, 72
– implicitní, 72
– princip fungování, 71
zamykání, 53
zařízení
– informace, 126
– škálování, 23
– určení polohy, 366
záznam
– aktualizace, 252
– editace, 245, 259, 264
– pro obchod, 444
– přidání, 245, 258, 263
– vymazání, 252
zdroj údajů, 270
zdrojová data, 270
zobrazení obrázku, 294
zpráva, asynchronní, 223
zvětšování objektu, 318
zvuk, nahrávání, 334
zvukový efekt, 328

Ž

životní cyklus, 65
životnost mobilního telefonu, 56