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

Vývoj aplikací pro Android

Ľuboslav Lacko

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

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.

1. vydání

ALBATROS MEDIA a.s.

Obsah

KAPITOLA 10

KAPITOLA 11

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.

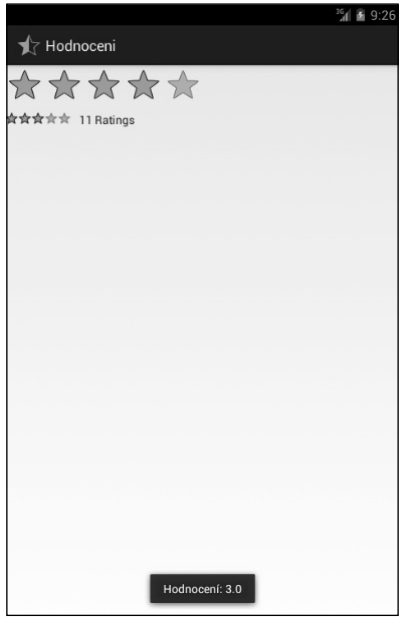
Publikace je určena i migrujícím vývojářům, kteří chtějí svou úspěšnou aplikaci, vytvořenou původně pro počítač nebo pro některou mobilní platformu, zpřístupnit i milionům uživatelů zařízení s Androidem.

Publikace předpokládá určité předběžné znalosti:

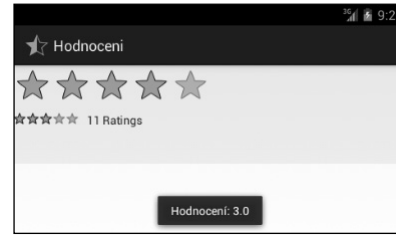
- Znalost programovacího jazyka Java. Postačí i znalost C#, jelikož tento jazyk je odvozený od Javy a je mu velmi blízký.
- Znalost základních principů objektově orientovaného programování.
- Zkušenosti s používáním moderních integrovaných vývojových prostředí.
- Znalost SQL (nejlépe SQLite, ale postačuje 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.

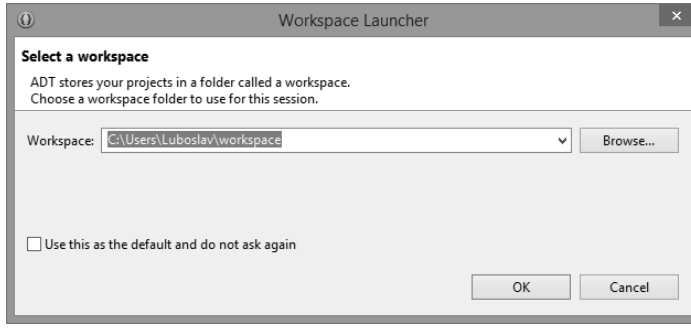
Existují čtyři edice jazyka Java zaměřené na různá prostředí s různě rozsáhlým obsahem.

- **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:

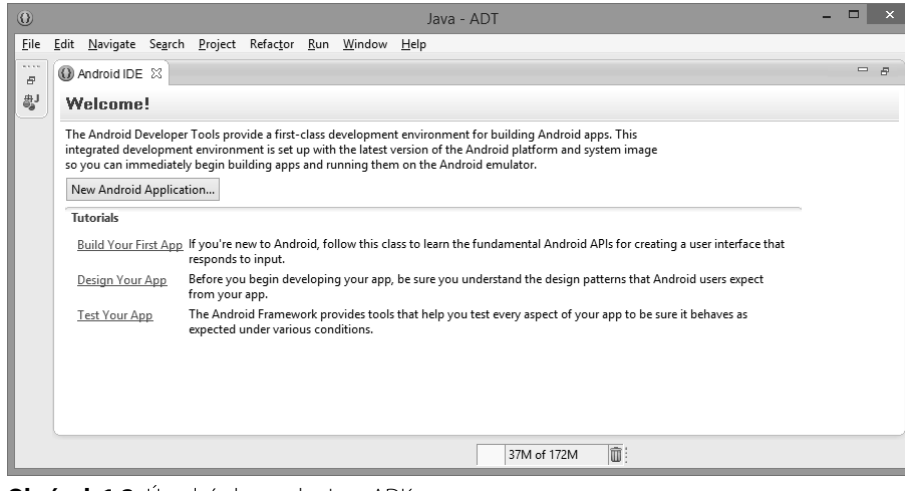
- Vývojové prostředí Eclipse + ADT plugin
- Android SDK Tools
- Android Platform-tools
- Obrazy verzí operačního systému pro emulátor

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.

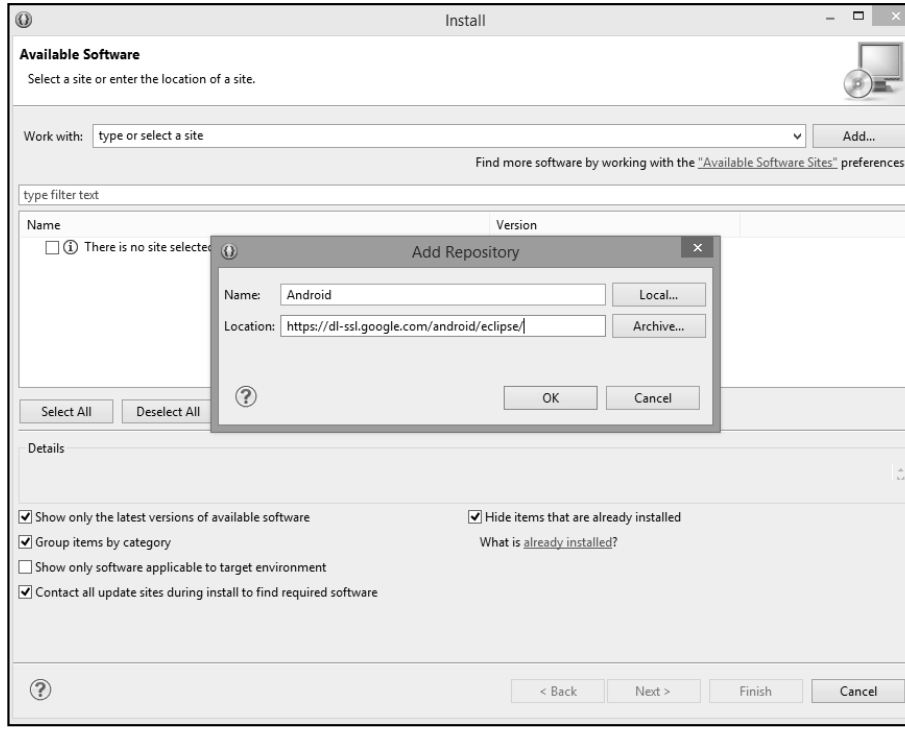
1. V archivu staženém z webu je složka se stejným názvem jako název souboru archivu. Tuto složku rozbalte na vhodné místo. Doporučujeme vytvořit složku s výstižným názvem, například *Android*, *Vývoj* apod. Ve složce *adt-bundle-<os_platforma-datum>* jsou vnořené složky *eclipse*, *sdk* a spustitelná 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.



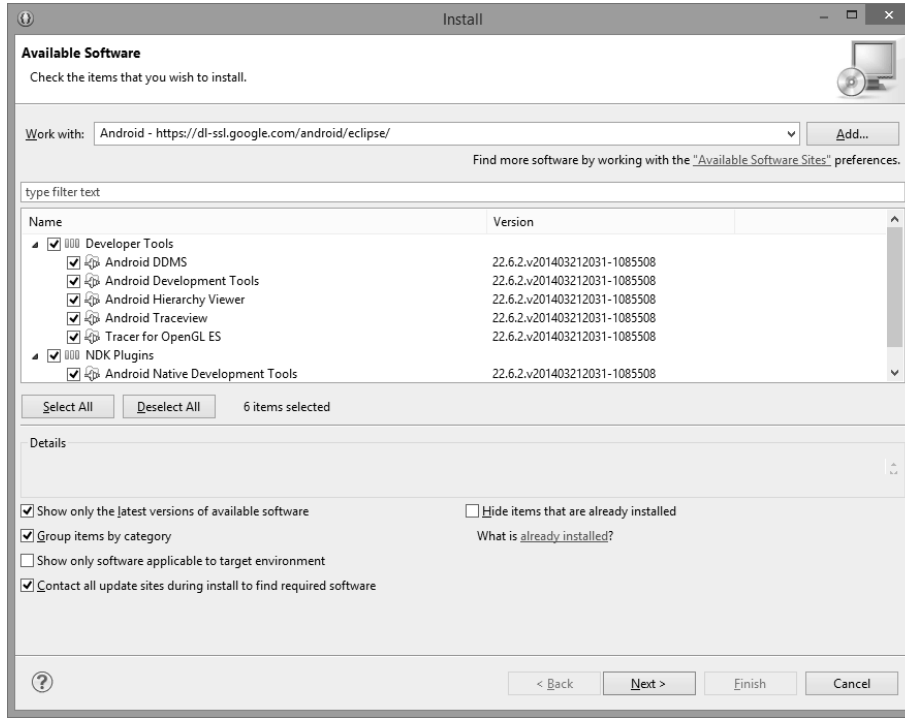
Obrázek 1.2: Úvodní obrazovka Java ADK

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).

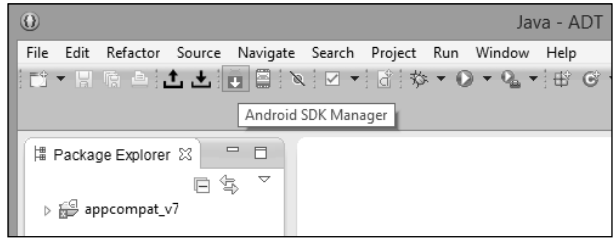


Obrázek 1.3: Instalace zásuvných modulů Android Developers Tools (ADT)

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).



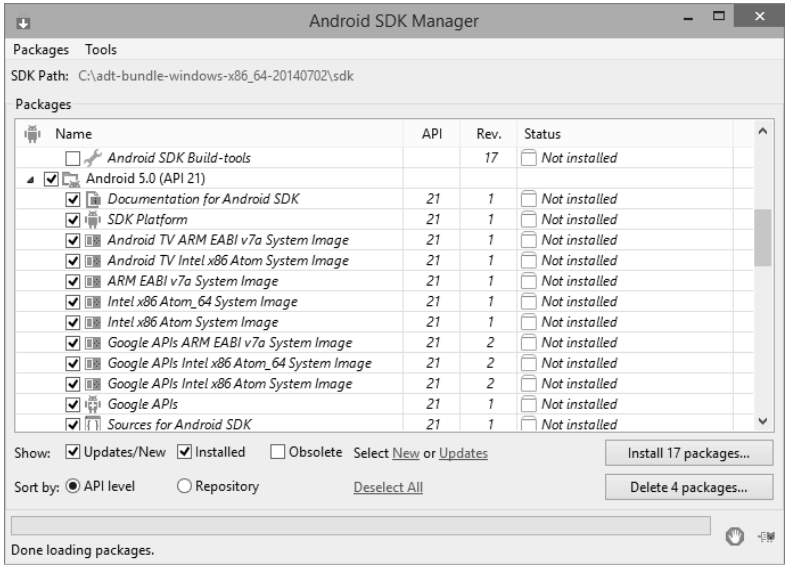
Obrázek 1.4: Konfigurace zásuvných modulů Android Developers Tools (ADT)



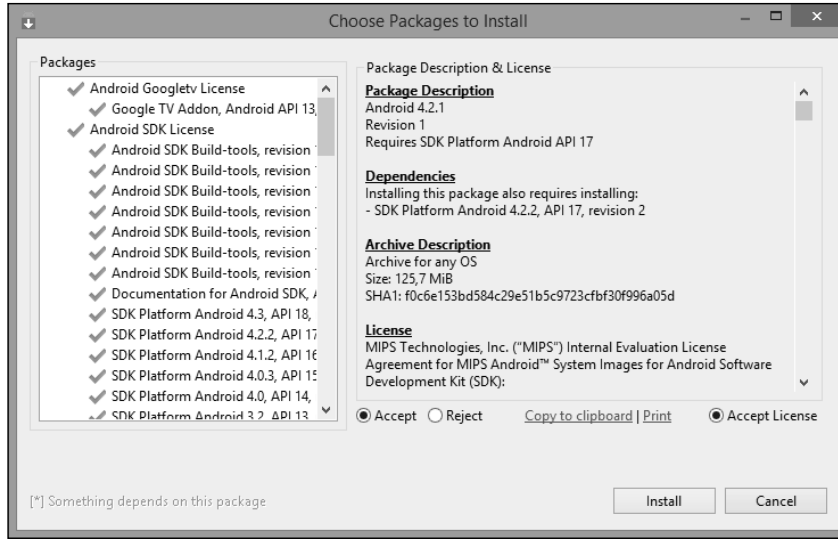
Obrázek 1.5: Spuštění nástroje Android SDK Manager přímo z prostředí Eclipse

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



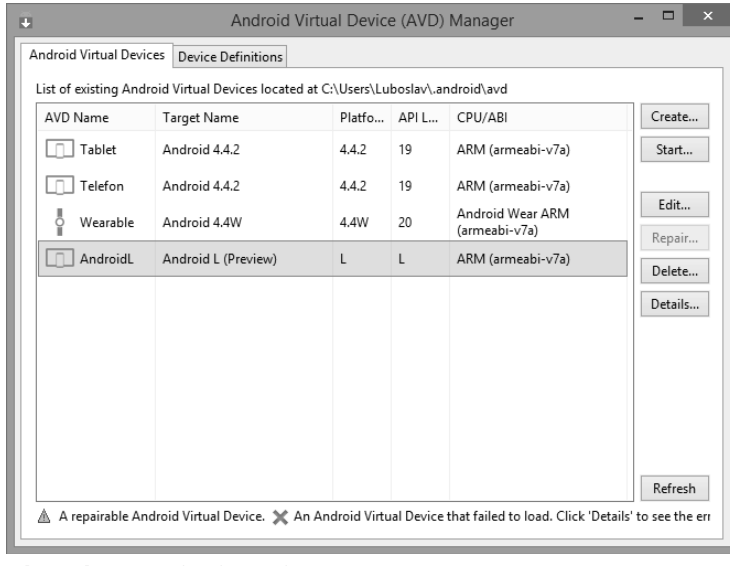
Obrázek 1.7: Souhlas s licenčními podmínkami při instalaci balíčků v aplikaci Android SDK Manager

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

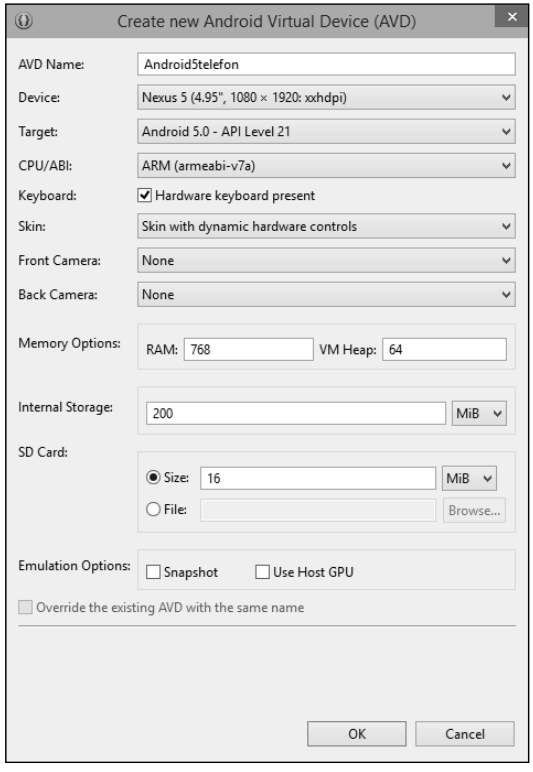
Zdálo by se, že pokud máte k dispozici jeden moderní chytrý telefon a jeden tablet s Androidem, dokážete pokrýt návrh, testování a ladění aplikací. Opak je pravdou. Výhodou a zároveň nevýhodou Androidu je variabilita různých zařízení s různými verzemi systému a různým rozlišením displeje. Neodmyslitelnou 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.

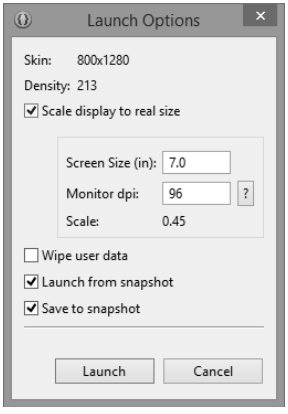


Obrázek 1.8: Android Virtual Device Manager

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.

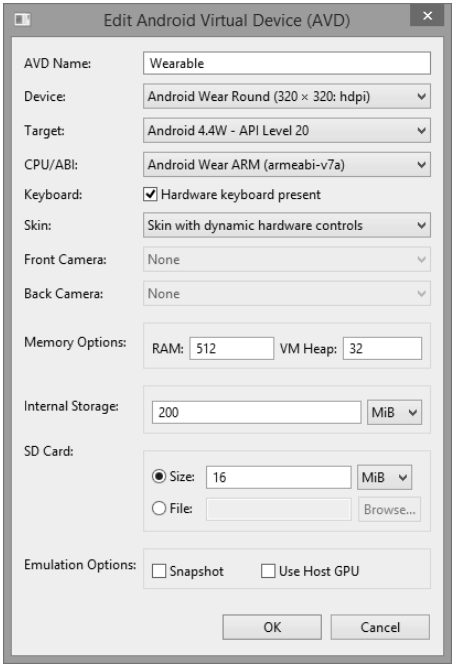
Obrázek 1.11: Spuštění emulátoru pro platformu Android 4.2.2

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

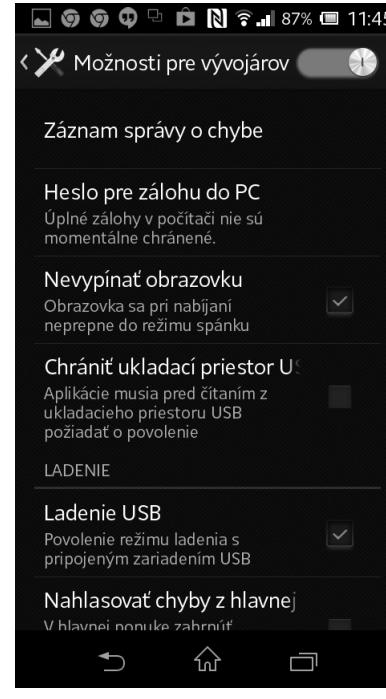
Obrázek 1.13: Emulátor Android 4.4 Wearable

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.

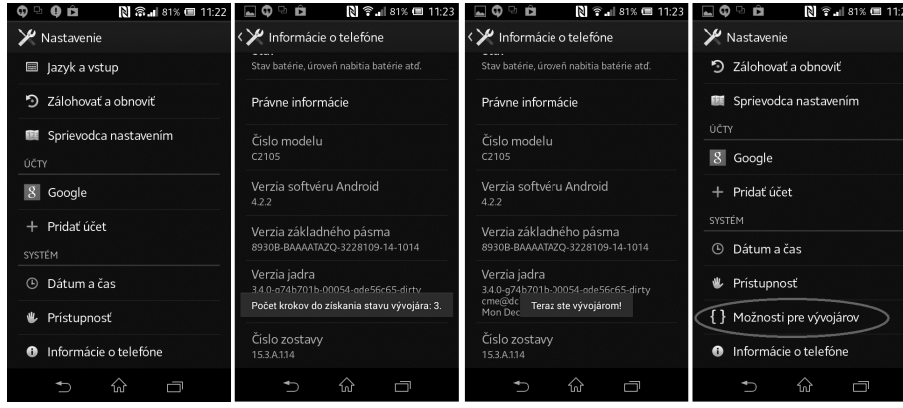
Na zařízeních s operačním systémem Android 4.2 a vyšším jsou možnosti pro vývojáře implicitně skryty a musíte je nejprve zobrazit:

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.

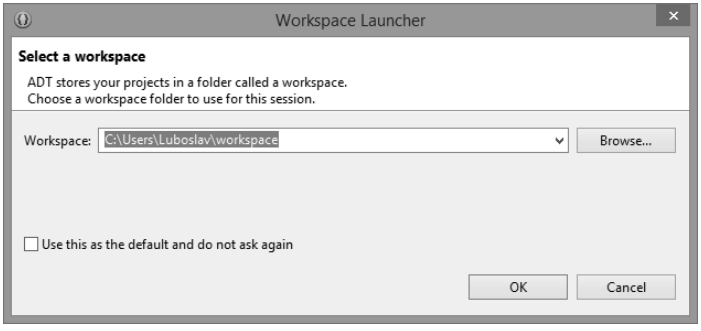
Poznámka: První projekt má v tomto případě i motivační význam, jelikož doslova na jedno klepnutí a bez jakéhokoliv programování vytvoříte aplikaci typu „Hello World", která vypíše na obrazovku text.

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

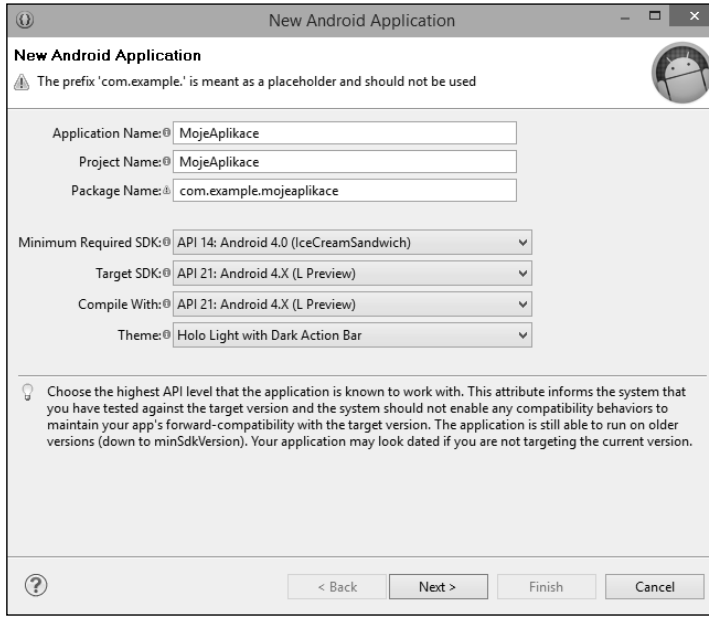
Pomocí nabídky **File → New → Android Application Project** vytvořte nový projekt. Můžete jej nazvat například MojeAplikace. Všimněte si tří polí pro zadání názvu. Název zadaný do pole **Application Name** se bude zobrazovat při spouštění projektu. Do pole **Package Name** se zadává název javového balíčku, 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**).

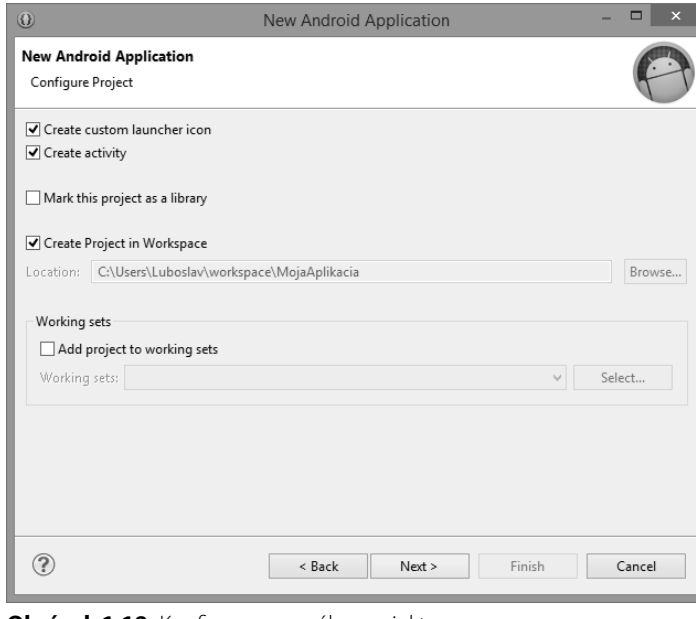
Vyberte verzi Androidu a vyplňte povinné položky **Project name**, **Application name** a **Package name** (ve tvaru *com.example.mojeaplikace*).

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

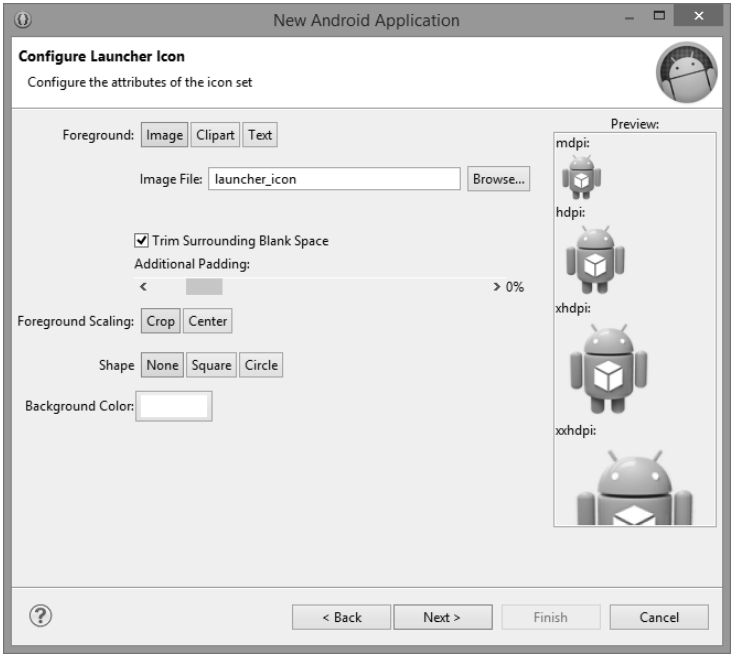
V následujícím dialogovém okně můžete změnit ikony aplikace. Jelikož je tato první aplikace cvičná a slouží k ověření správnosti instalace a konfigurace vývojového prostředí a SDK, není potřeba se v této fázi zdržovat návrhem ikon. Ponechejte implicitně nastavenou ikonu postavy Androida.



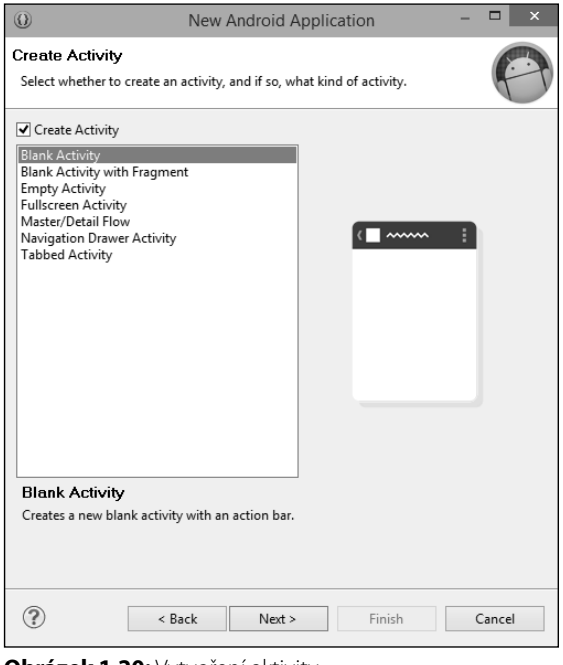
Obrázek 1.17: Vytvoření nového projektu



Obrázek 1.18: Konfigurace nového projektu



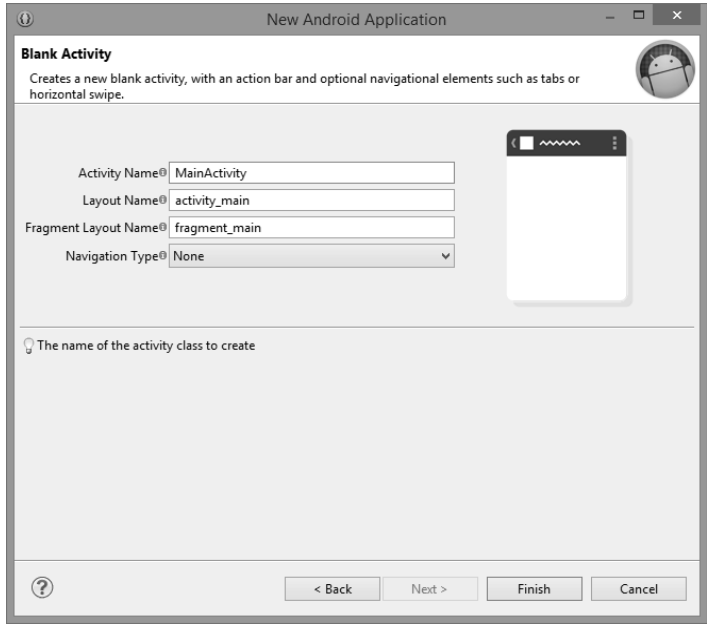
Obrázek 1.19: Ikony pro nově vytvořený projekt



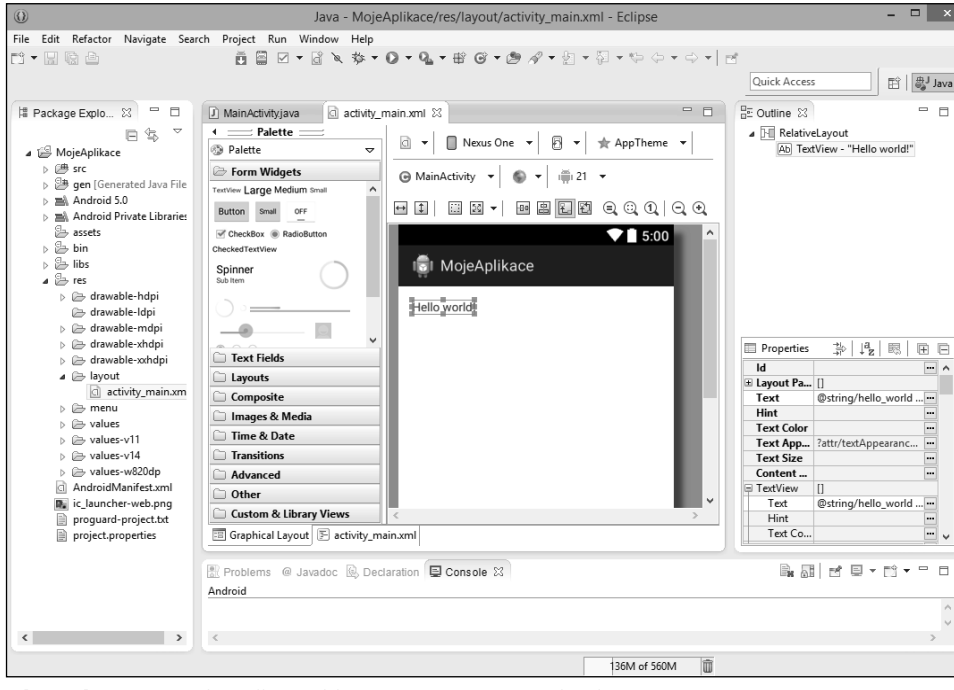
Obrázek 1.20: Vytvoření aktivity

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

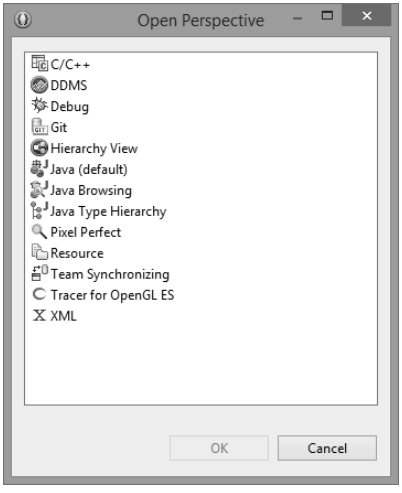


Obrázek 1.22: Projekt Hello World ve vývojovém prostředí Eclipse

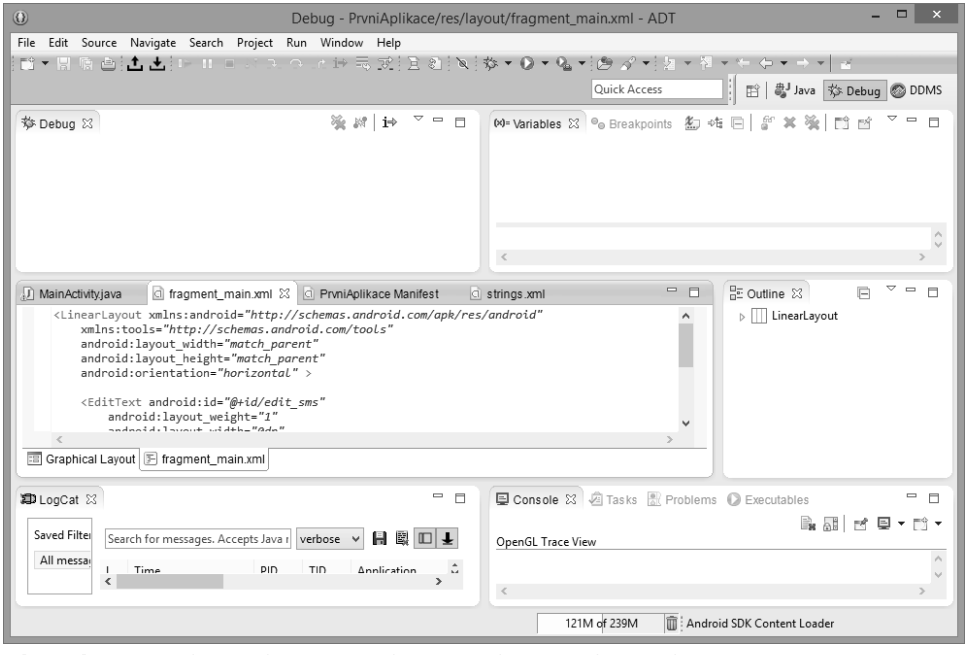
Tlačítkem **Finish** vytvořte projekt. S anatomií projektu se seznámíte v dalších kapitolách. V této fázi se pokusíte projekt spustit, nejprve na emulátoru a následně na reálném zařízení.

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

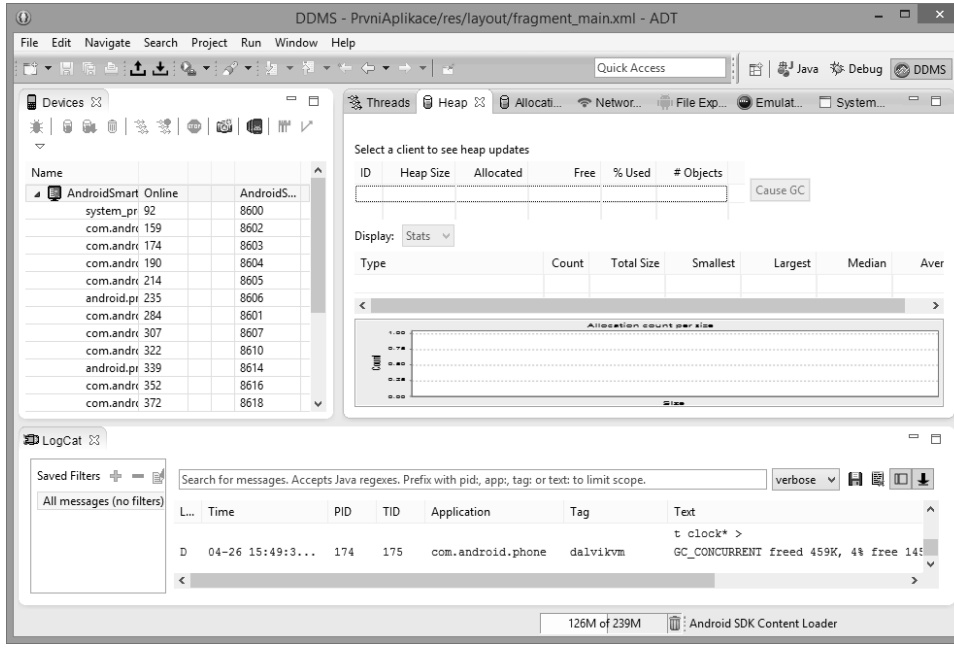


Obrázek 1.24: Rozložení oken vývojového prostředí v perspektivě Debug

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)

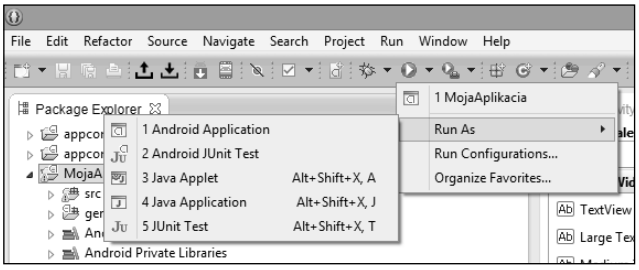
DDMS umožňuje sledovat fungování vaší aplikace na reálném zařízení se systémem Android. Vestavěný modul LogCat umožňuje v reálném čase sledovat všechny procesy, které probíhají na připojeném zařízení. Pro lepší přehlednost je možné události filtrovat a sledovat jen ty, které vás v daném okamžiku zajímají. 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)

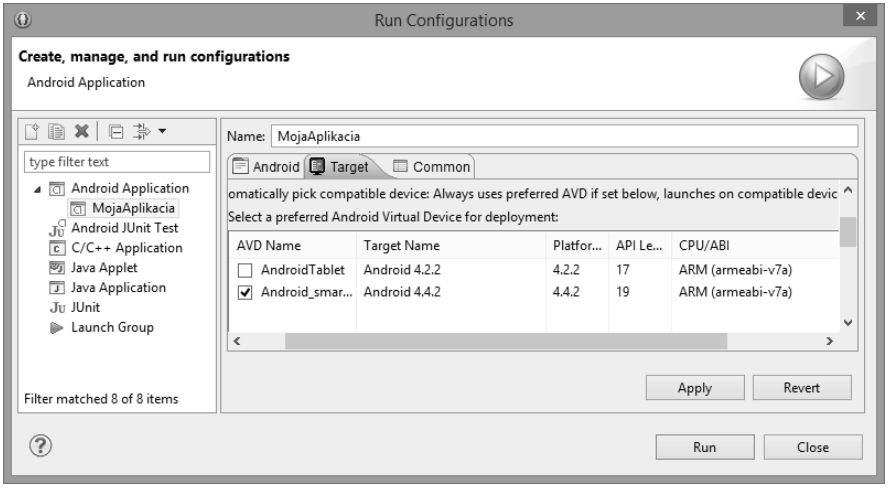
Spuštění aplikace v emulátoru

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.

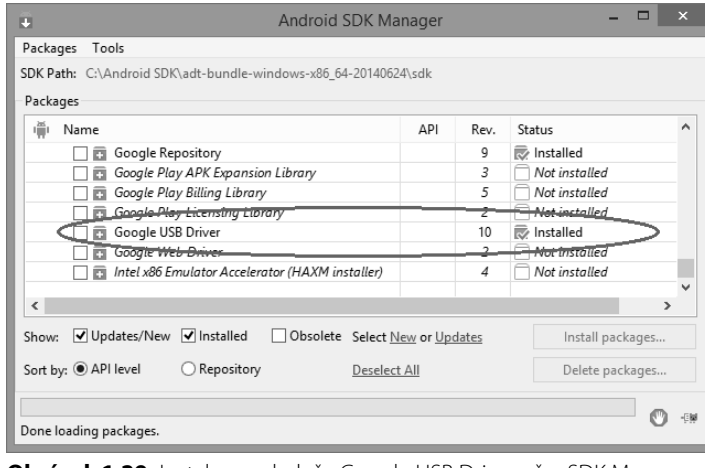
```
[2014-04-06 20:37:25 - MojeAplikace] ------------------------------
[2014-04-06 20:37:25 - MojeAplikace] Android Launch!
[2014-04-06 20:37:25 - MojeAplikace] adb is running normally.
[2014-04-06 20:37:25 - MojeAplikace] Performing
  com.example.mojeaplikace.MainActivity activity launch
[2014-04-06 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

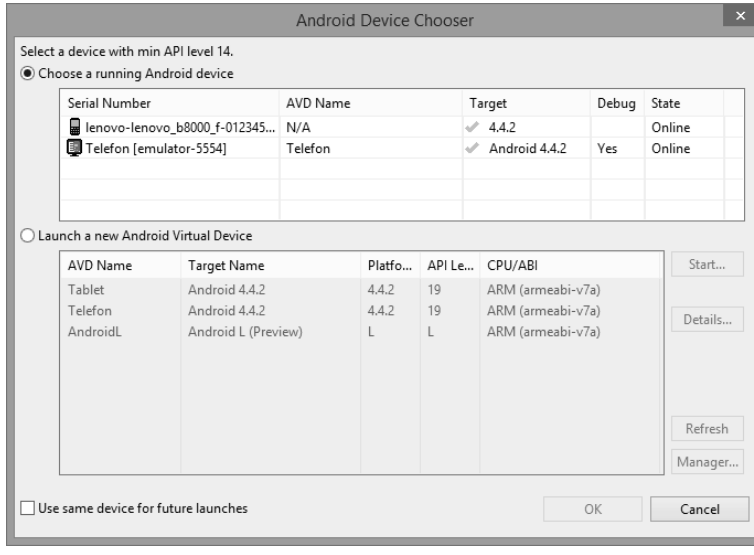
Aby bylo možné aplikaci spustit na reálném zařízení připojeném přes USB, je potřeba na vývojářském počítači nainstalovat USB ovladače pro ADB (Android Debug Bridge). Potom stačí



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

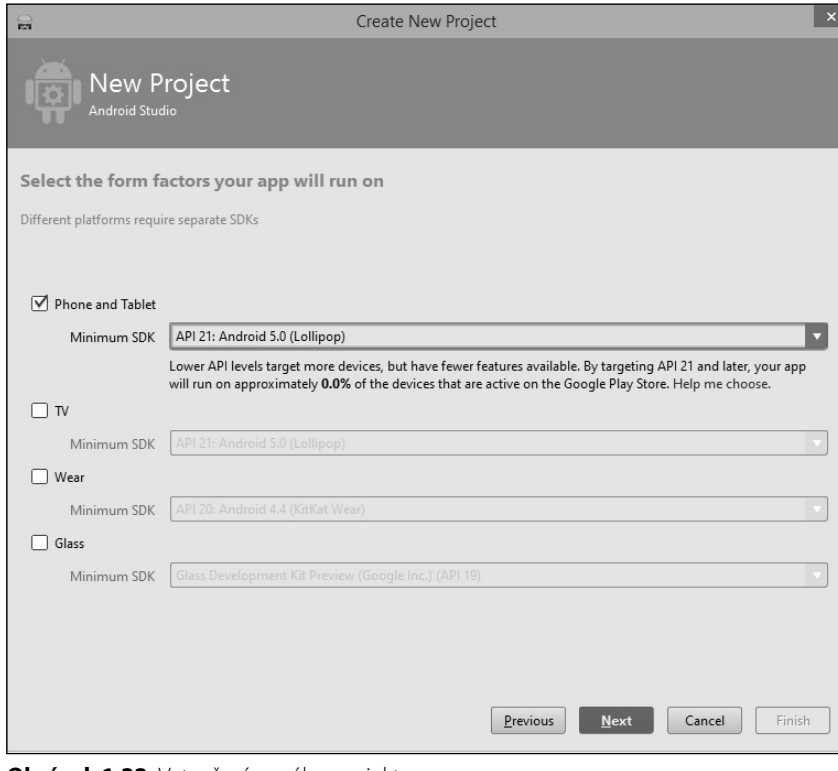
Android Studio

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

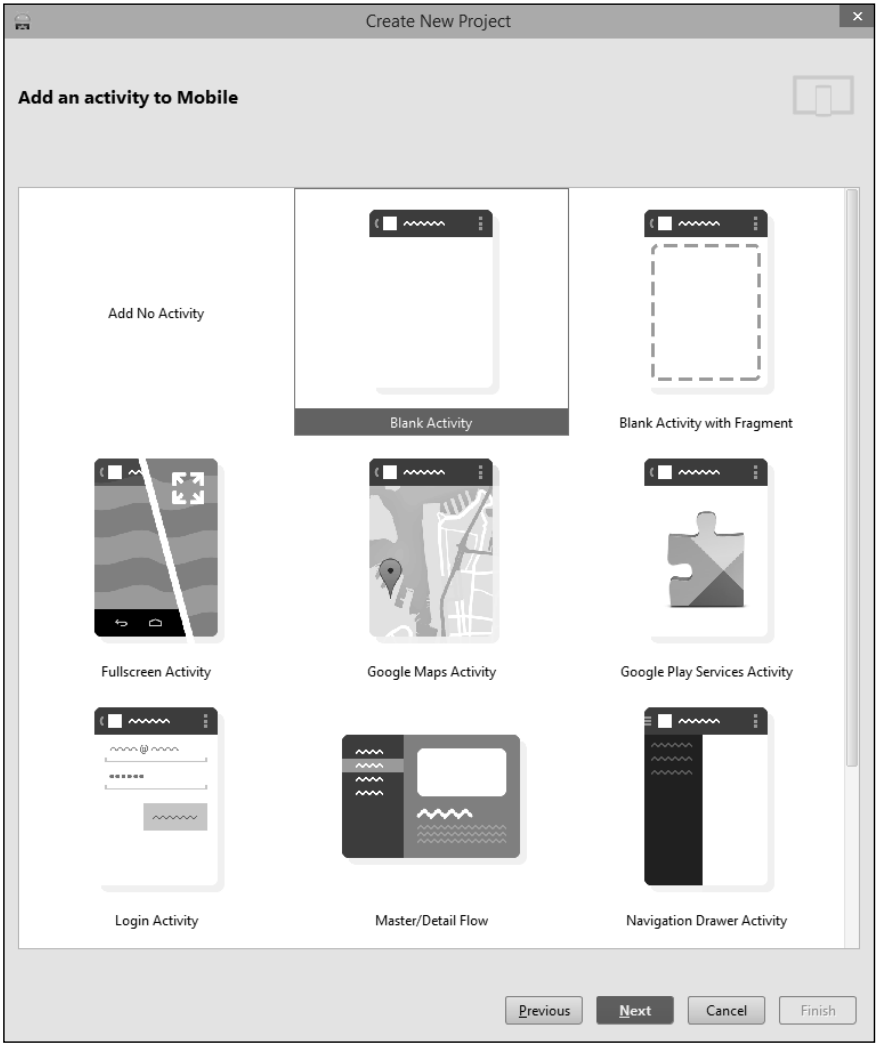


Obrázek 1.32: Vytvoření nového projektu

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.

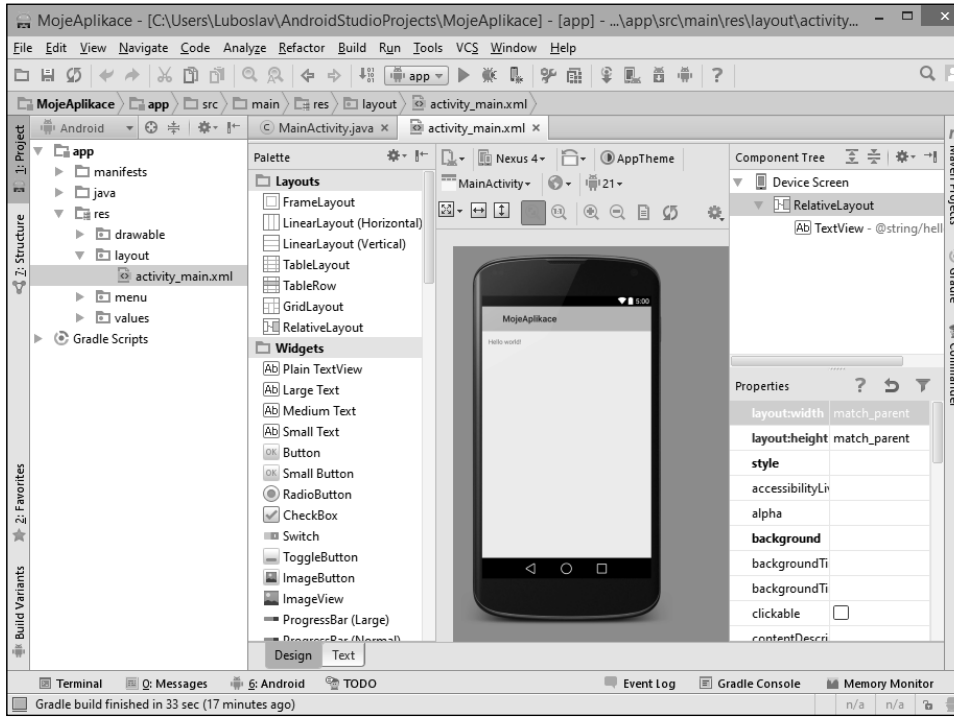
Můžete zvolit několik typů pro hlavní aktivitu aplikace včetně mapové či aktivity typu master-detail.



Obrázek 1.33: Výběr typu hlavní aktivity

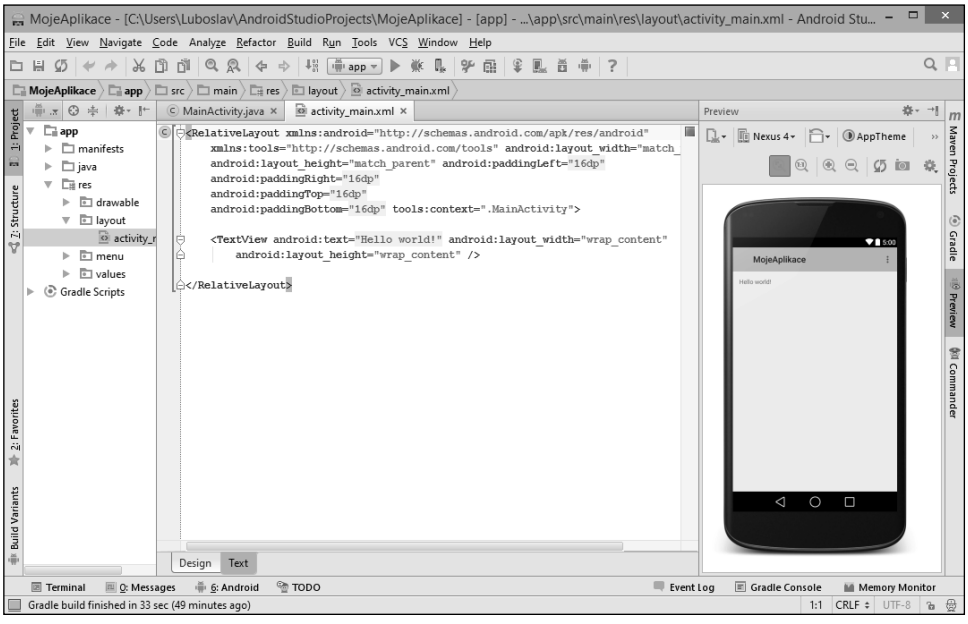
Další postup se odvíjí od nastaveného typu hlavní aktivity. Například pokud jste zvolili typ master-detail, systém vás vyzve k definování názvu položky a skupiny položek, přičemž skupina položek je plurálem od názvu položky; například objednávka/objednávky a podobně.

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.

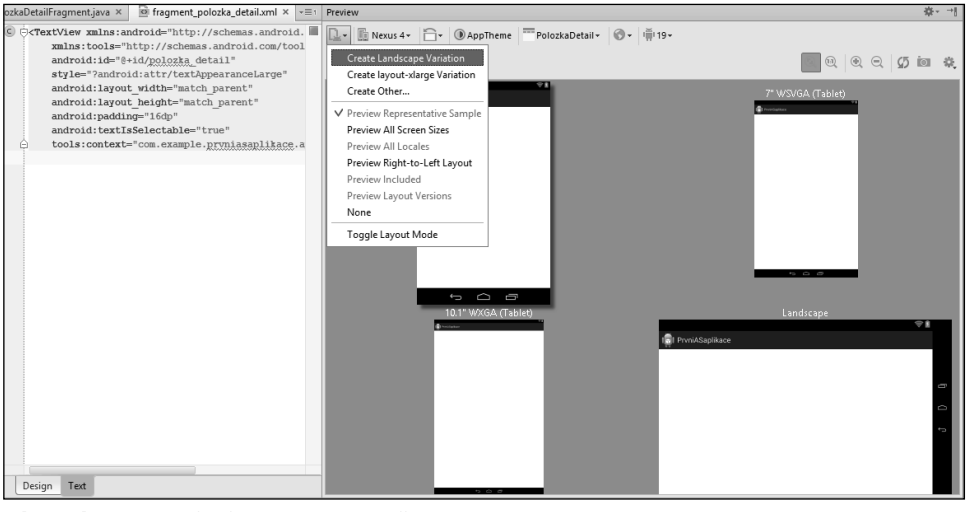


Obrázek 1.34: Uživatelské rozhraní Android Studia

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



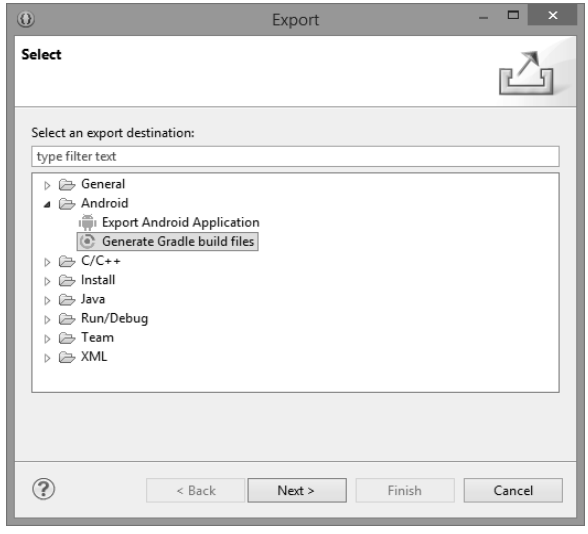
Obrázek 1.36: Mód zobrazení Preview All Screen Sizes

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.

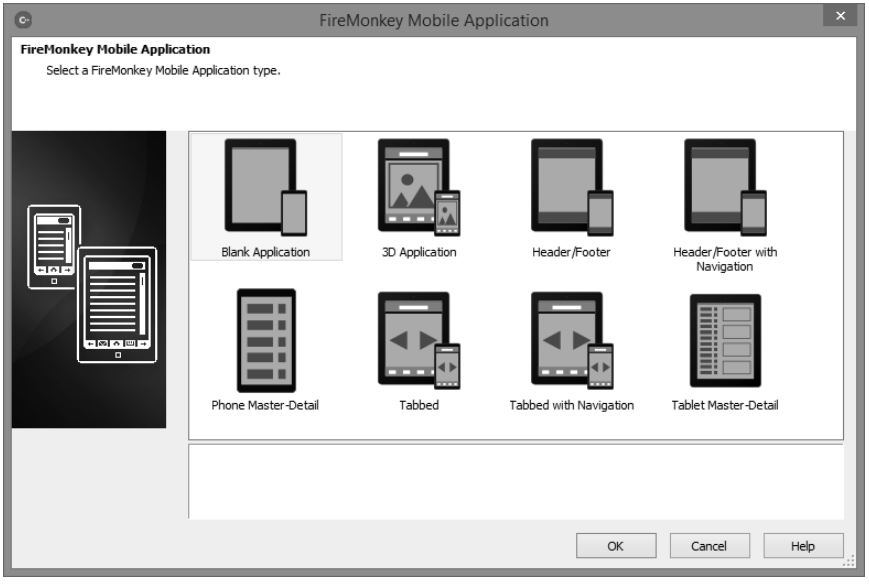
Firmy zabývající se vývojem aplikací si snadno dokáží spočítat, jakým koeficientem by se u nich násobila produktivita vývojářských týmů, pokud by mohly pro nejrůznější platformy použít jeden nástroj, jednotný programovací jazyk a jednotný framework. Multiplatformovost oceníte i po ukončení vývoje v dalších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

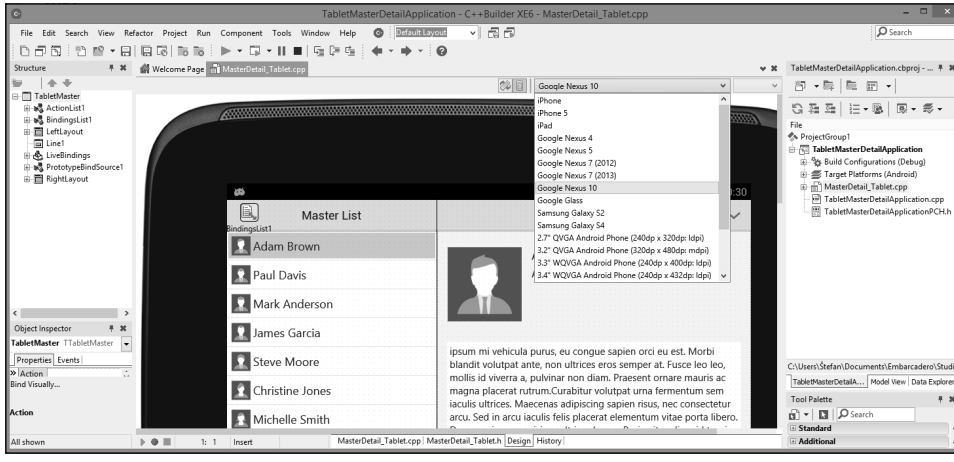
FireMonkey

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

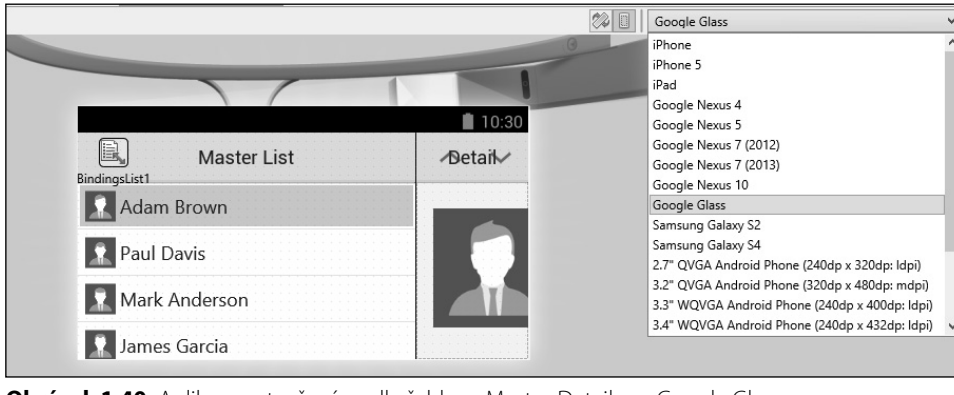


Obrázek 1.38: Šablony pro FireMonkey mobilní aplikace

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.

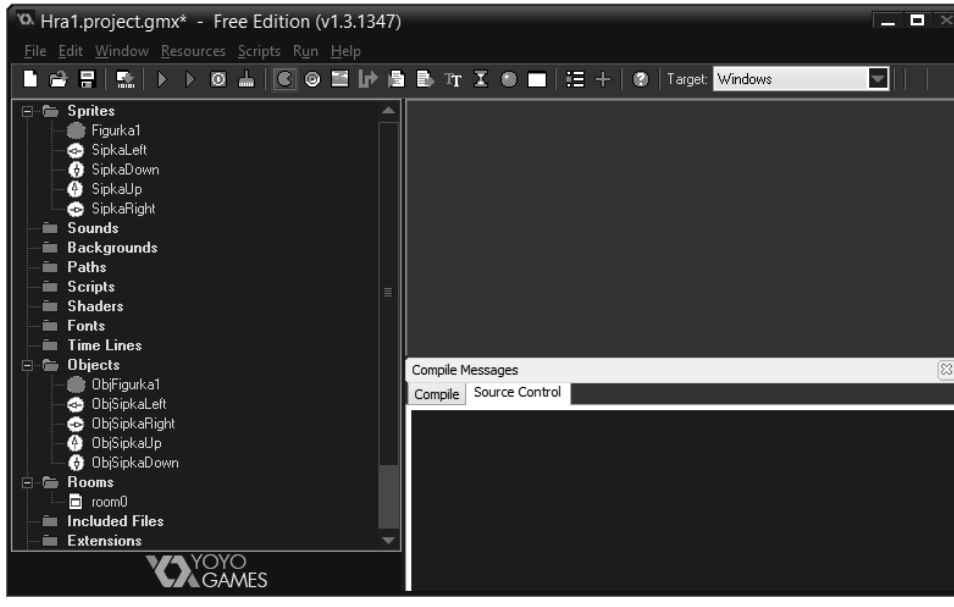
Poznámka: Game Maker Studio Edice Standard je bezplatná, verze Pro, která disponuje pokročilejšími vlastnostmi, například správou textur či možností ladění na mobilním zařízení s Androidem, je k dispozici za 99 dolarů.

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.

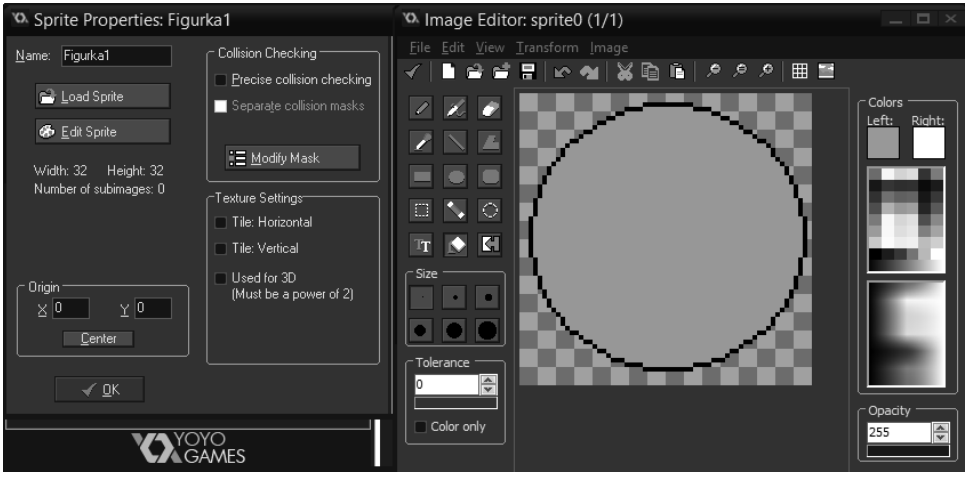
Vytvořte nový projekt s vhodným názvem. Po spuštění si všimněte vlevo složek pro jednotlivé typy objektů, které je možné v nástroji Game Maker Studio vytvořit. Nejprve vytvořte hráče. Klepněte na panelu nástrojů na ikonku se symbolem zelené figurky PacMan.



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

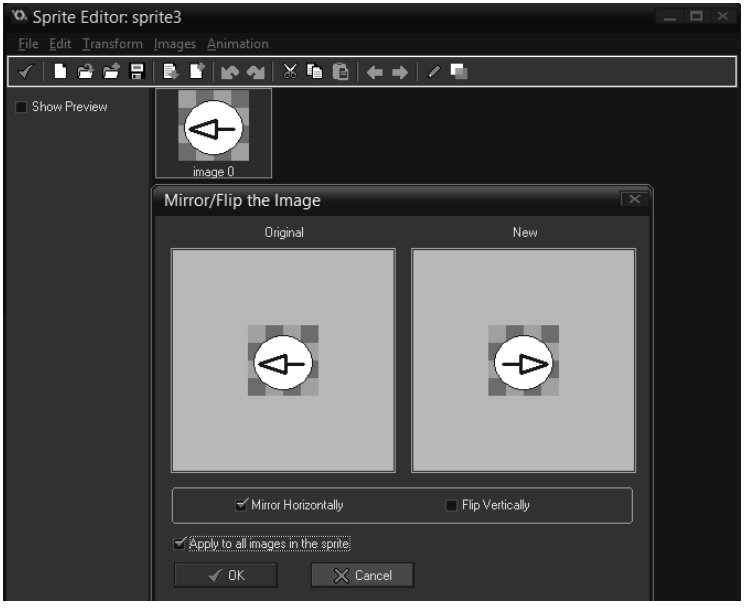
Zobrazí se dialogové okno pro vytvořený sprite (v doslovném překladu skřítek, v terminologii Game Maker Studia grafický návrh herních figurek). Dialogové okno obsahuje i informace o parametrech. Objekt hry, který právě vytváříte, nejprve vhodně pojmenujte. Pomocí tlačítka **Edit Sprite** a položky nabídky **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

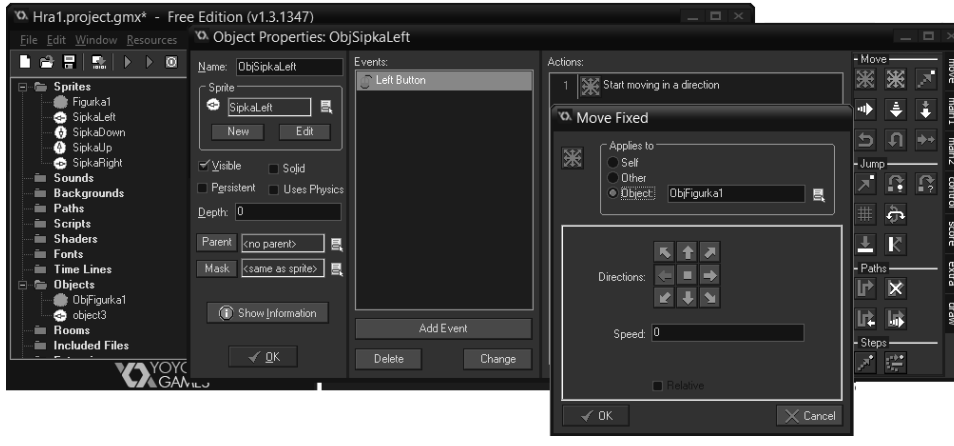
Dalším krokem je na základě figur (sprite), tedy grafických návrhů prvků hry, vytvořit objekty. Objekty je možné vytvářet pomocí ikony se symbolem zeleného kruhu. Do objektu přidejte předtím vytvořený sprite. Parametr **Depth** udává polohu hráče v ose Z.

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

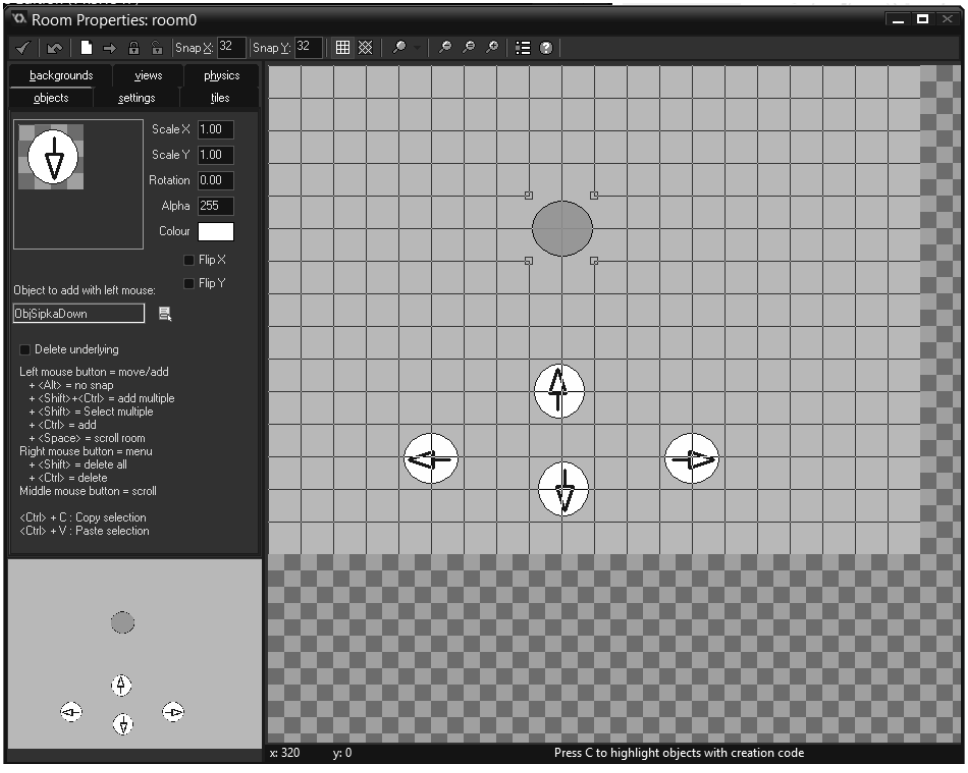
Akci definujete vizuálně v okně **Actions**. Přesuňte na plochu objekt **Move** (ikona se zelenými šipkami, první v sekci **Move**). Následně specifikujte požadovaný směr. Definujte rychlost pohybu, například 5, parametr **Applies to** nastavte na volbu **Object** a zadejte název objektu – figurky, která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

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



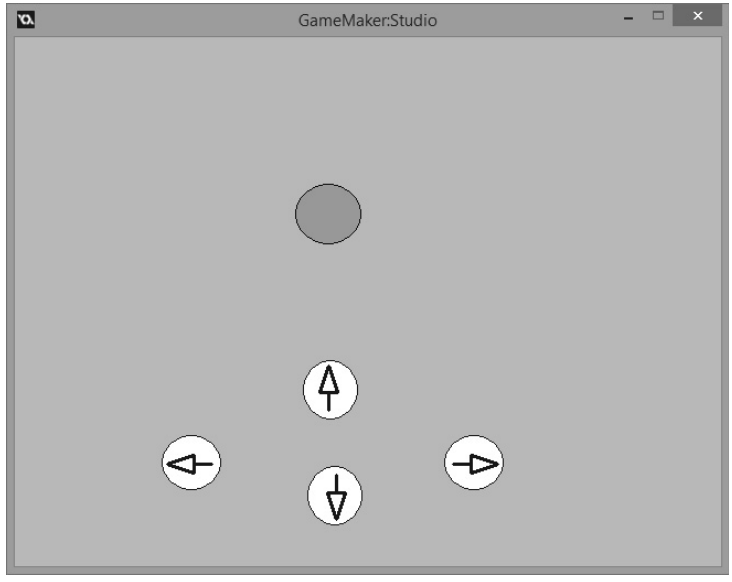
Obrázek 1.45: Vytváření objektů a definování akcí

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.

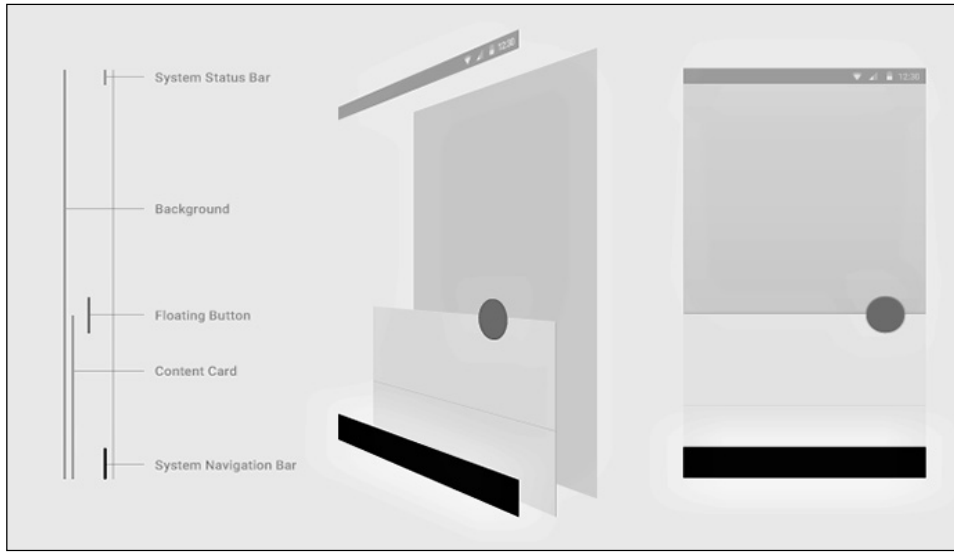
Jednou z velkých změn v nových verzích Androidu je nahrazení dosud používaného virtuálního stroje Dalvik, pod kterým běžely aplikace, novým enginem ART (Android Runtime).

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.

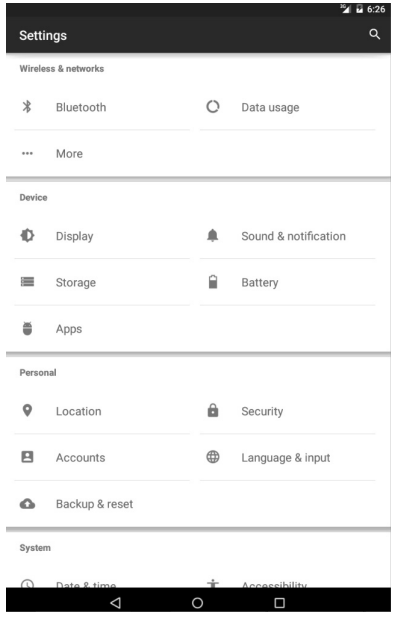
Hlavním přínosem ART je možnost kompilovat část kódu aplikací už při jejich instalaci. To významně urychlí spouštění aplikací a sekundárně prodlouží výdrž baterie, jelikož Dalvik proces kompilace vykonával při každém spuštění.

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.

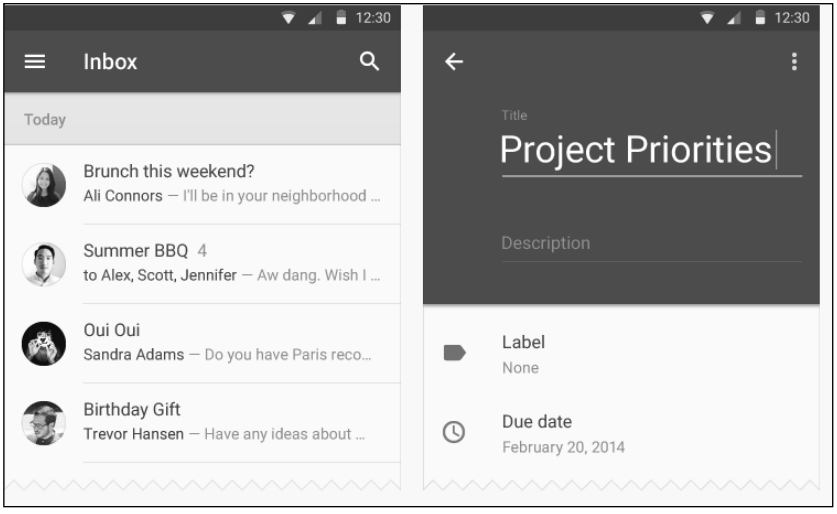


Obrázek 2.3: Design nové verze označovaný jako Material Design je přehlednější a pomocí různých vizuálních efektů, hlavně stínu, se snaží vzbudit zdání třetího rozměru

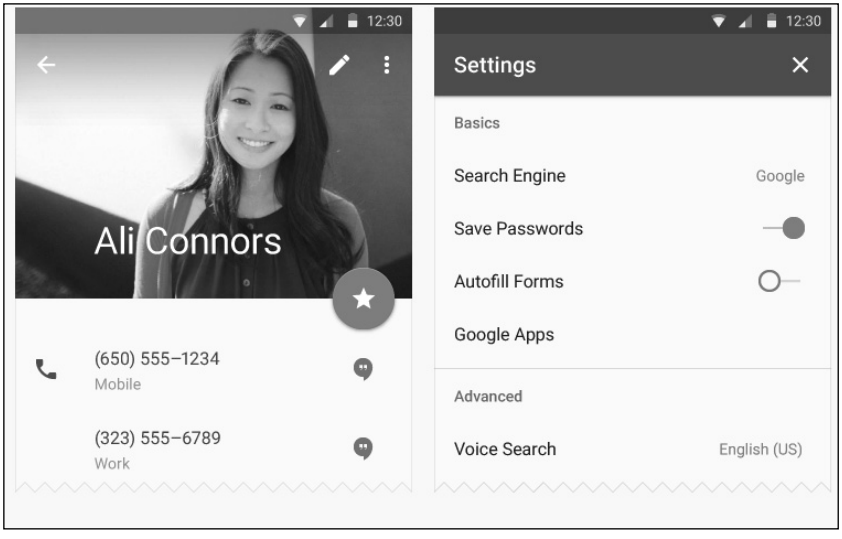


Obrázek 2.4: Design aplikace Nastavení je praktickou ukázkou designových doporučení pro aplikace pro Android 5.0

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.

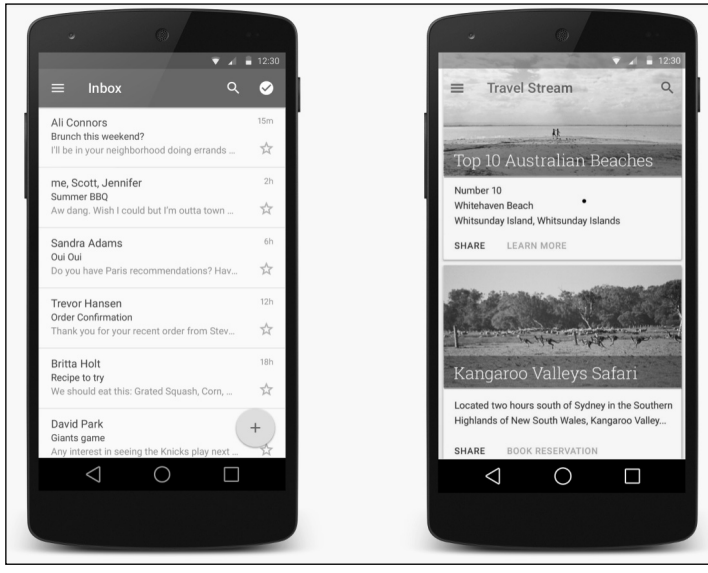


Obrázek 2.5: Příklad designu aplikace



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.

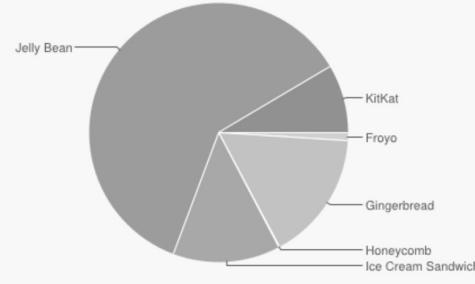


Obrázek 2.7: Nové prvky uživatelského rozhraní RecyclerView vlevo a CardView vpravo

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.

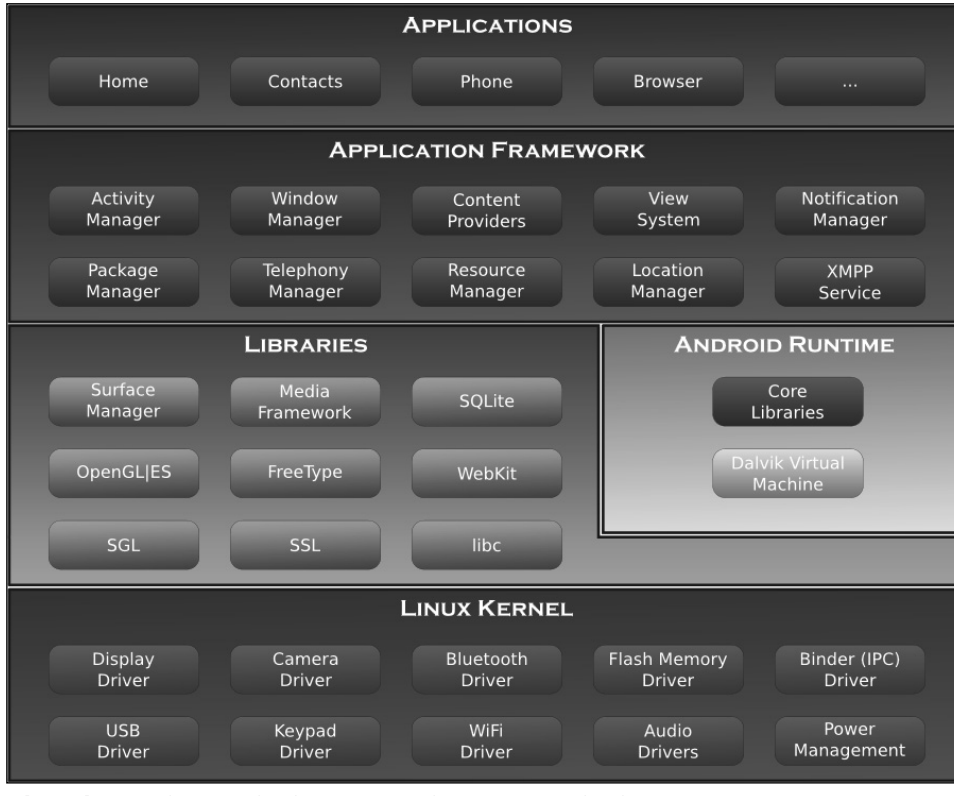
K dispozici jsou knihovny:

- **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:

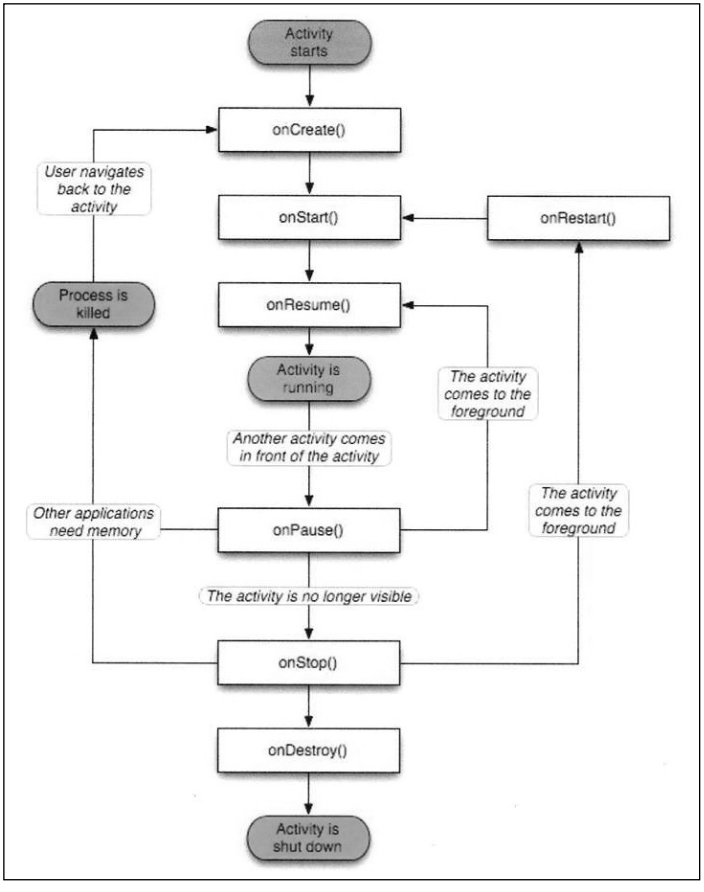
- **Aktivita na popředí** – zobrazuje se na displeji a má takzvaný fokus, to znamená, že interaguje s uživatelem. Aktivity běžící na popředí jsou ve stavech Running nebo Resumed.
- **Pozastavená aktivita** – je stále v paměti, je zpravidla částečně viditelná, ztrácí ale fokus, to znamená, že uživatel 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`.

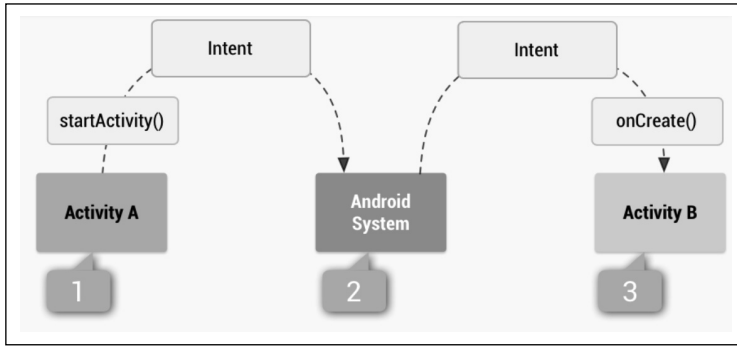
Intent Filter

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:

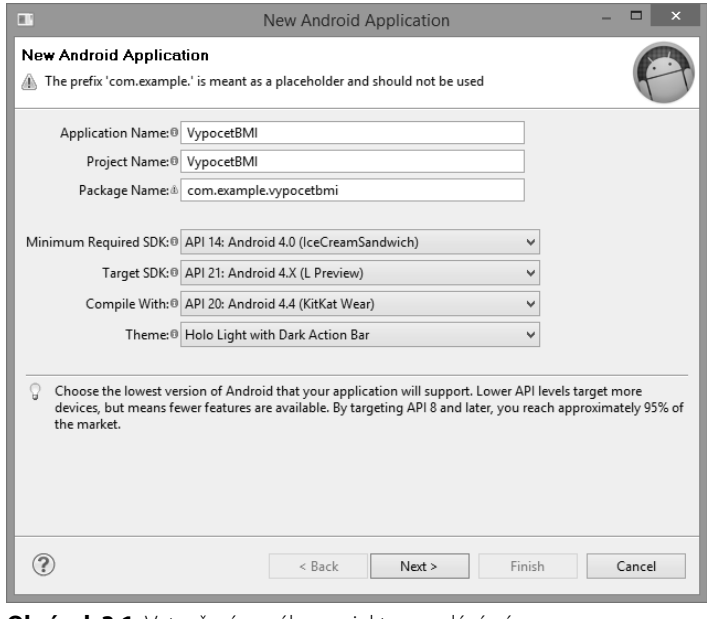
- **Application Name** – název zadaný do tohoto pole se bude zobrazovat při spouštění aplikace. Napište do pole vhodný název, v tomto případě „Výpočet BMI". Jelikož se jedná

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:

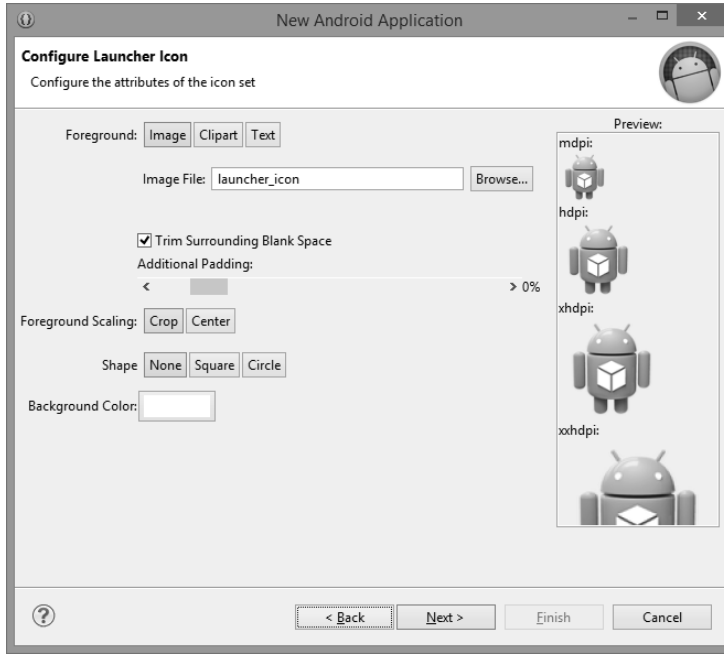
- None
- Holo Dark
- Holo Light
- Holo Light with Dark Action Bar

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.



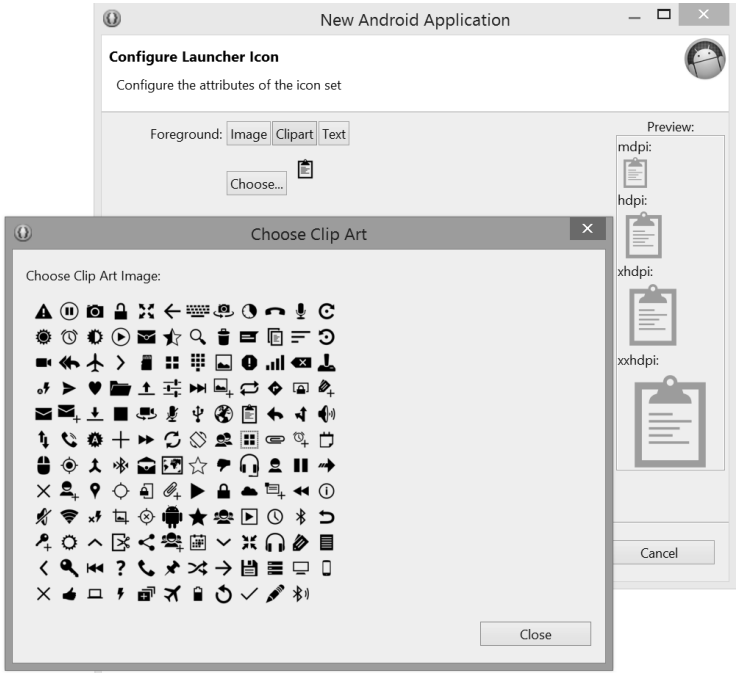
Obrázek 3.2: Ikony pro nově vytvořený projekt

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.

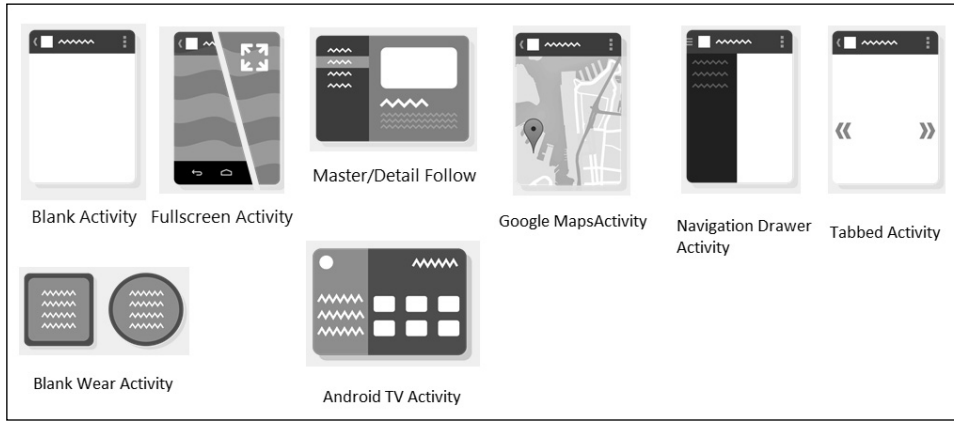


Obrázek 3.3: Nabídka ikon z galerie clipartů

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:

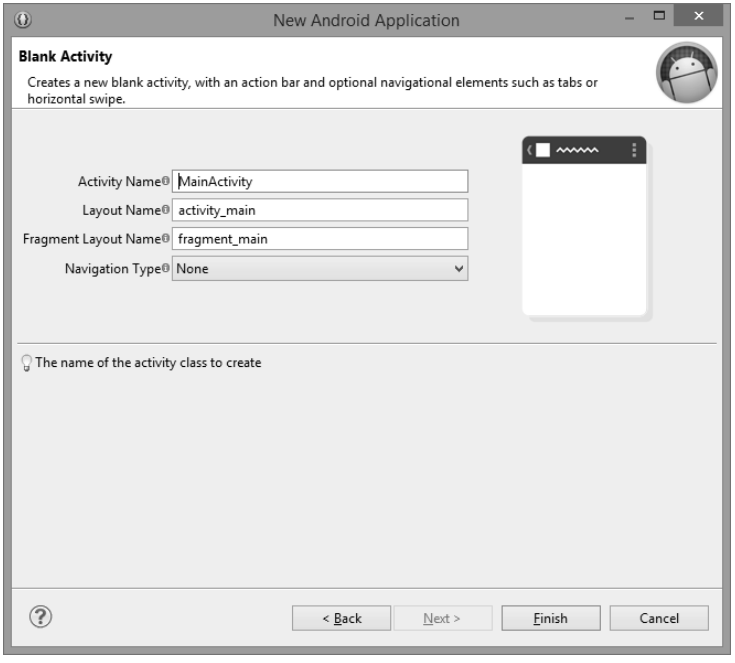
- Activity Name
- Layout Name
- Fragment Layout Name
- Navigation type

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



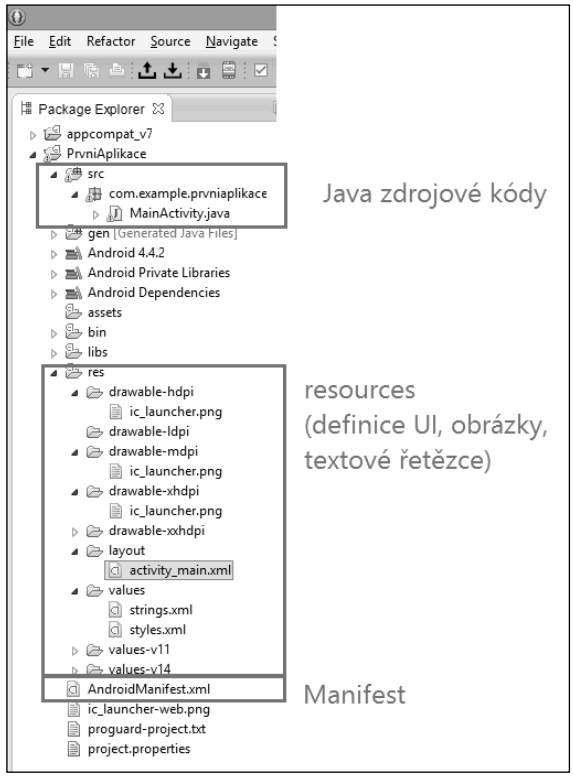
Obrázek 3.6: Konfigurace vybraného typu aktivity

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

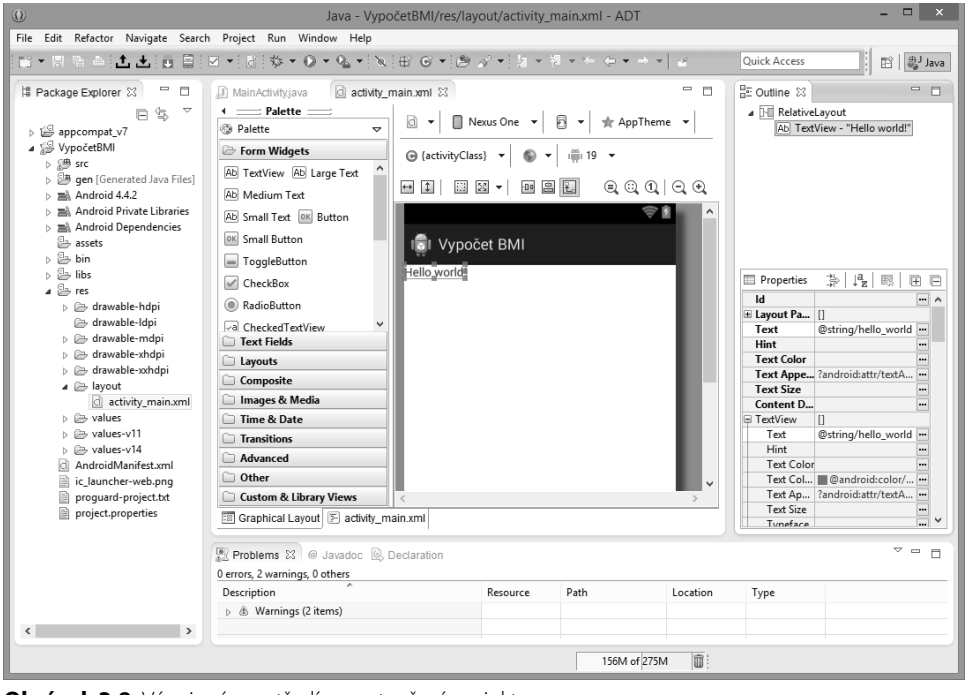
Jednotky dp znamenají density independent pixels.

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"
```

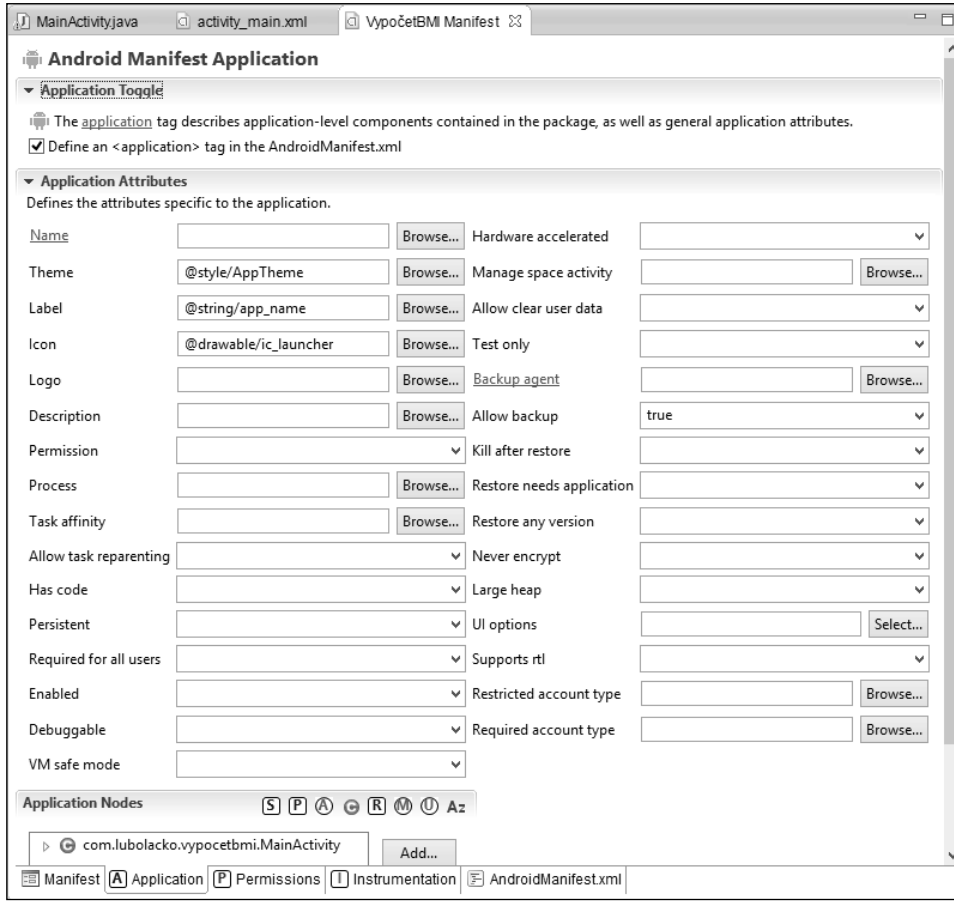
```
    android:targetSdkVersion="19" />

  <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

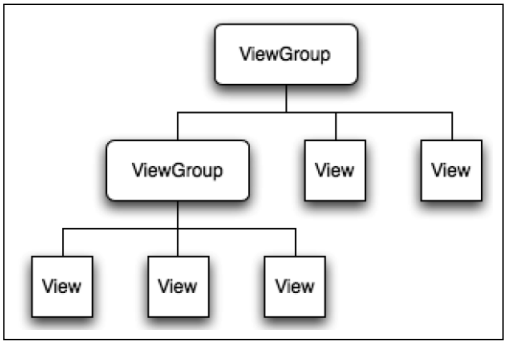
Zařízení, na kterých budou aplikace spouštěny, mají různou velikost obrazovky a různé rozlišení. Proto by programátoři migrující z klasických aplikací pro Windows měli zapomenout na praxi definování přesných souřadnic polohy a konkrétních rozměrů ovládacích prvků. Na Androidu definujete vzájemné uspořádání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