mistrovství

Ľuboslav Lacko

# Android

## KOMPLETNÍ PRŮVODCE VÝVOJÁŘE

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

computer press

# Mistrovství Android

Vyšlo také v tištěné verzi

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


**Ľuboslav Lacko**
**Mistrovství – Android – e-kniha**


ALBATROS MEDIA a.s.

Ľuboslav Lacko

# Mistrovství Android

Computer Press
Brno
2017

# Obsah


**Úvod** 15

Kapitola 1
**Nástroje pro vývoj** 17

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

Kapitola 2
**Anatomie Androidu** 65

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

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

Kapitola 3
**Anatomie aplikace** 79

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

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

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

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

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

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

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

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

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

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

Kapitola 11
**Grafika** 371

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

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

Kapitola 12
**Multimédia** 407

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

Kapitola 13
**Senzory a komunikace** 431

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

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

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

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

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

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

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

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

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

**Rejstřík 633**

# Úvod

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

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

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

Publikace předpokládá určité předběžné znalosti:

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

# Zpětná vazba od čtenářů

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

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

nebo

*sefredaktor.pc@albatrosmedia.cz*

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

# Errata

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

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

KAPITOLA

1

# Nástroje pro vývoj

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

- Java Development Kit (JDK)
- Android Studio

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

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

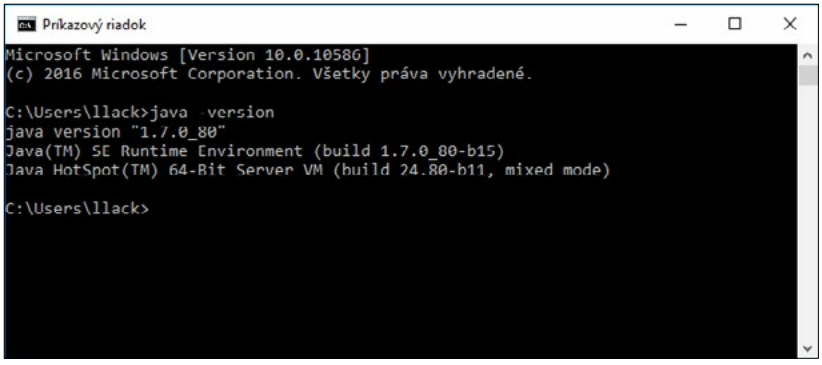
## Instalace Javy

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

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

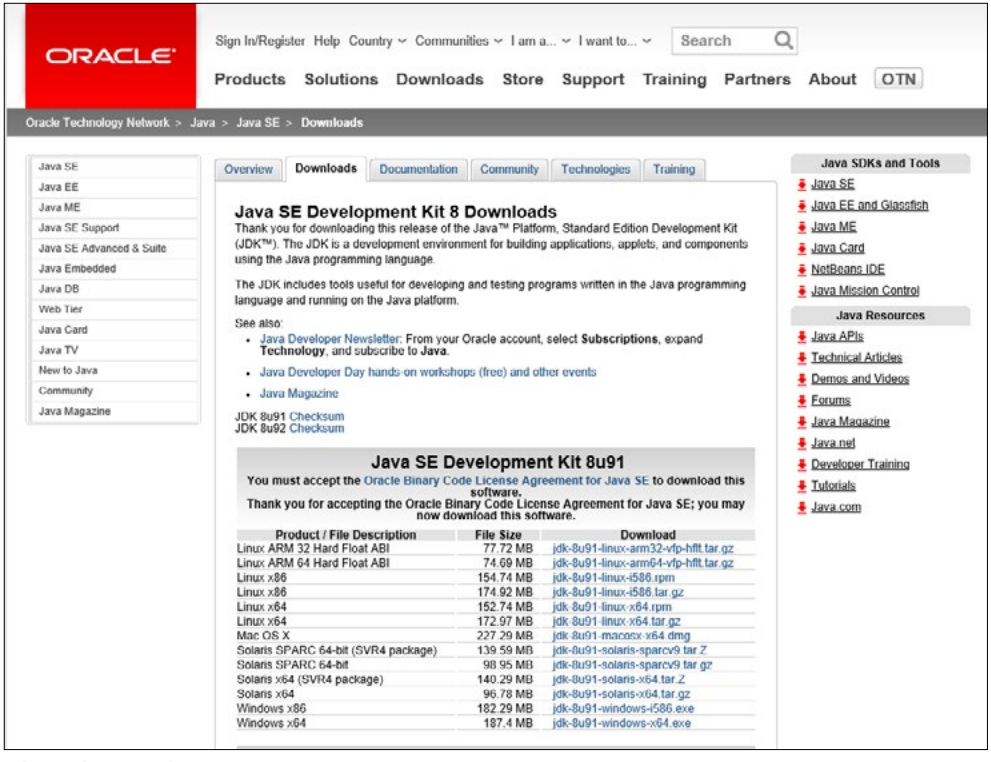
**Témata kapitoly:**

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

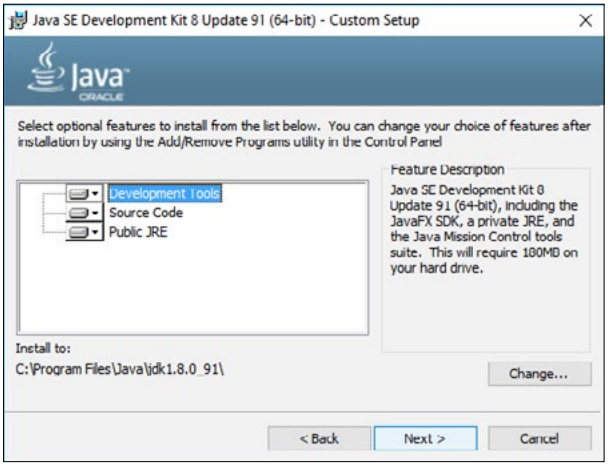



**Obrázek 1.1:** Identifikace nainstalované verze JDK

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

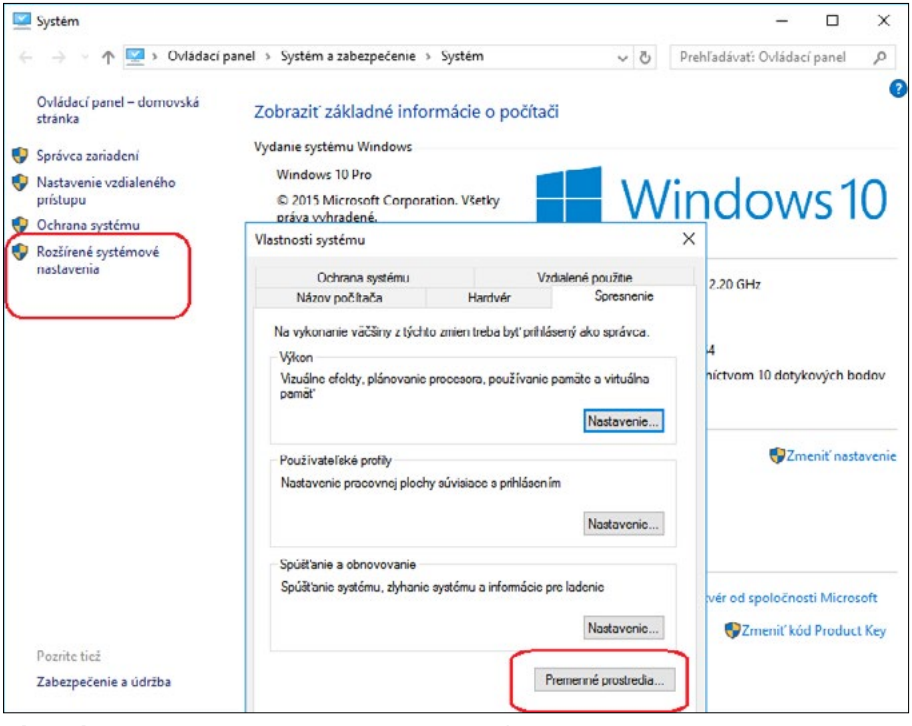



**Obrázek 1.2:** Vyberte správnou verzi JDK pro váš operační systém

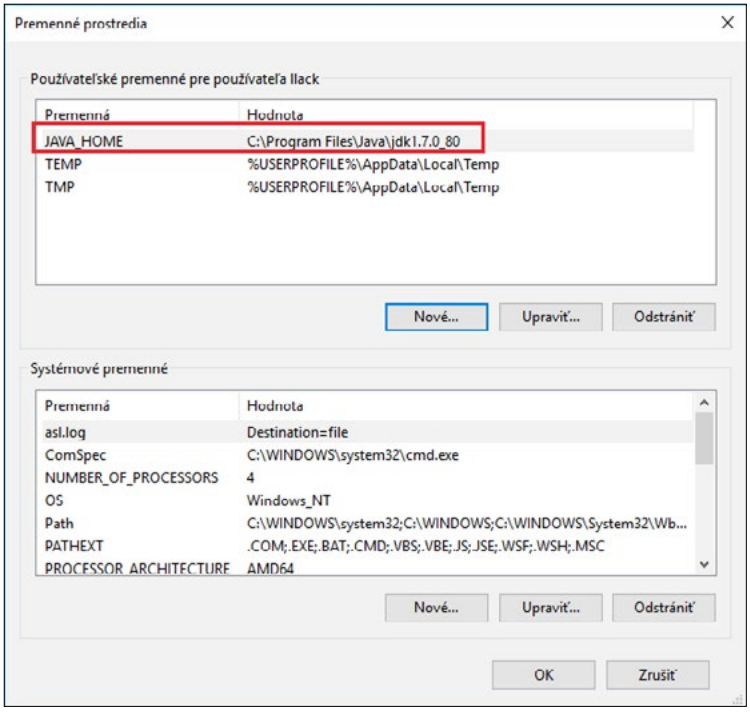



**Obrázek 1.3:** Instalace JDK

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




**Obrázek 1.4:** Rozšířené systémové nastavení Windows 10



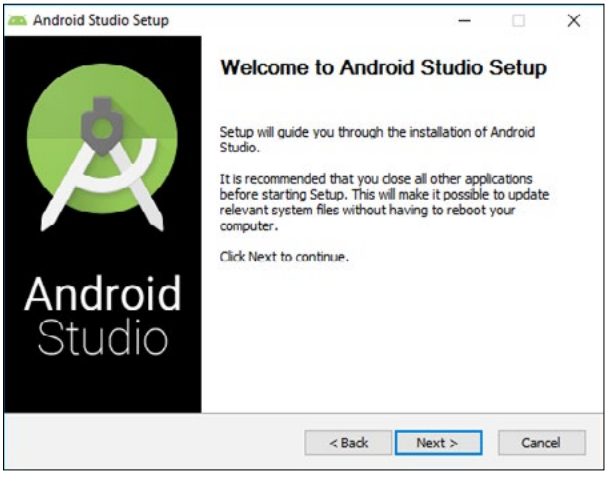

**Obrázek 1.5:** Nastavení uživatelské proměnné

# Instalace vývojového prostředí Android Studio

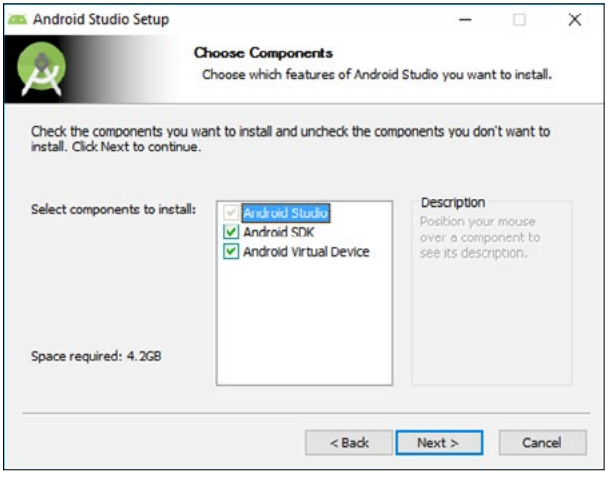
Android Studio jako oficiální nástroj na vývoj aplikací pro Android je k dispozici ke stažení na stránce *developer.android.com* pro platformy Windows, Linux a macOS. Dříve než zahájíte instalaci, nainstalujte balík Java SE Development Kit (viz předchozí podkapitolu). Instalace Android Studia je jednoduchá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

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

- Android Studio
- Android SDK
- Android Virtual Device



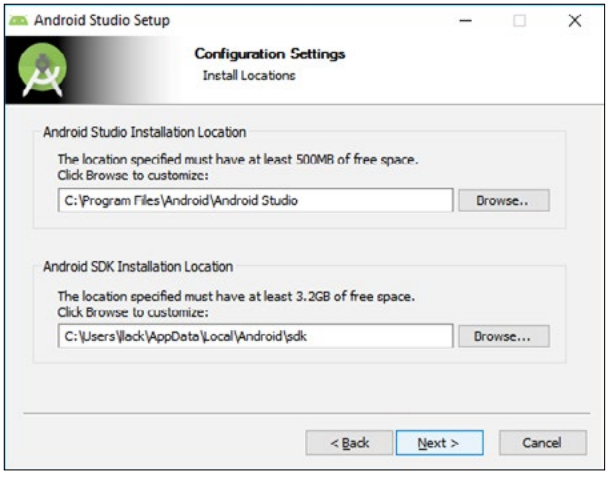

**Obrázek 1.6:** Instalace vývojového prostředí Android Studio




**Obrázek 1.7:** Android Studio – výběr komponent k instalaci

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

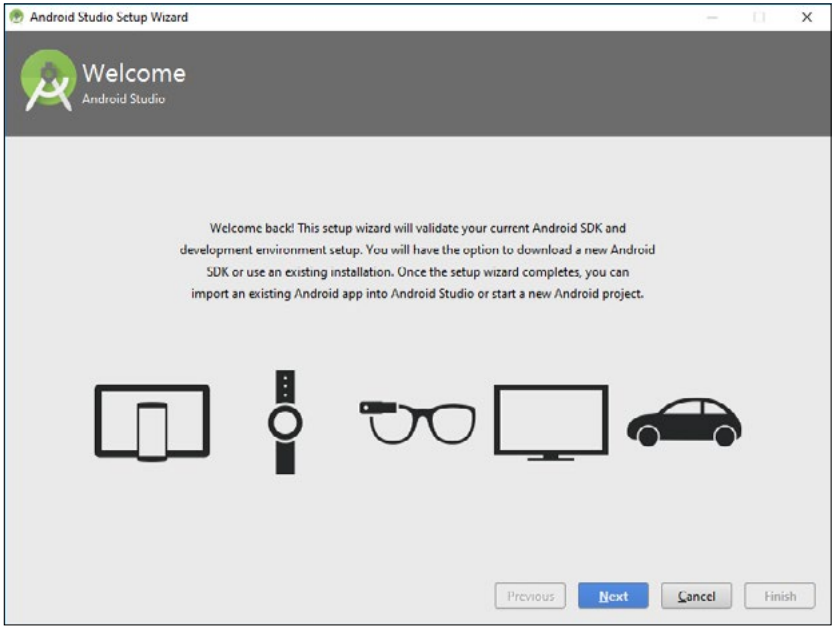
Během instalace se můžete setkat s oznámením firewallu, že Android Studio potřebuje využívat síťovou komunikaci, a firewall si od vás vyžádá povolení k přístupu.




**Obrázek 1.8:** Android Studio – implicitní adresáře k instalaci

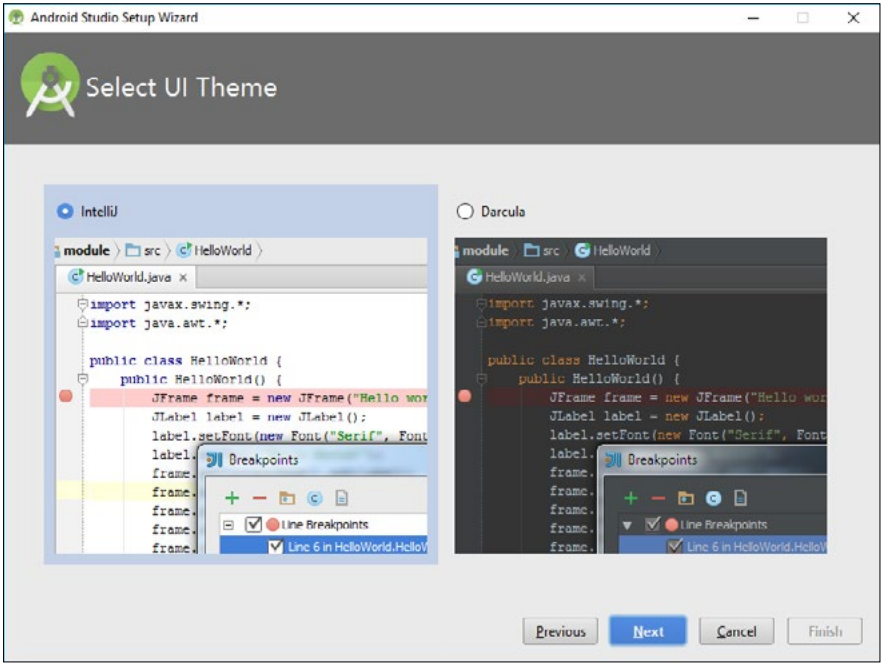
# První spuštění a konfigurace Android Studia

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




**Obrázek 1.9:** Android Studio Setup Wizard

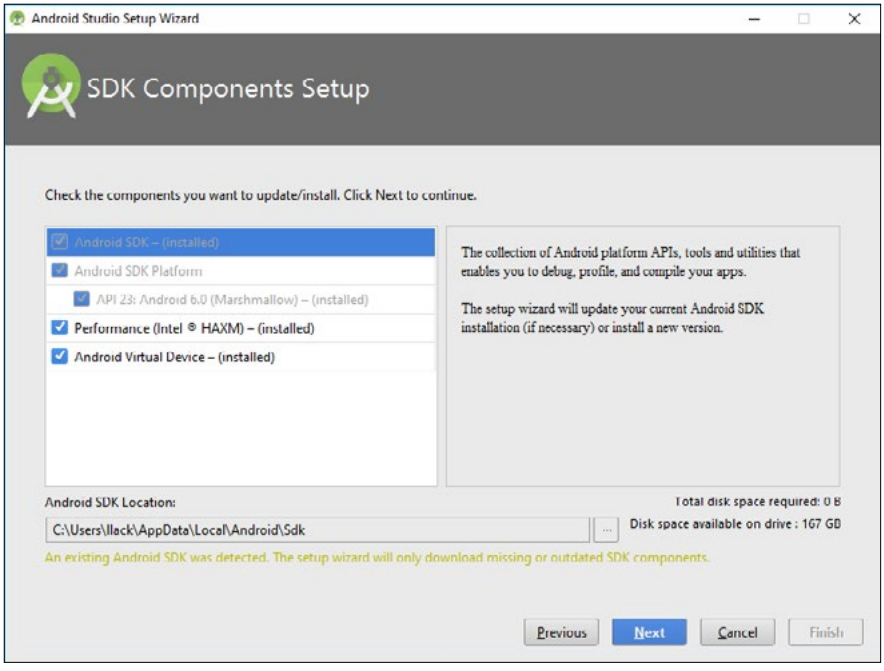
Můžete si zvolit typ instalace **Standard** nebo **Custom**. Pro běžné nasazení doporučujeme ponechat označenou volbu **Standard**.



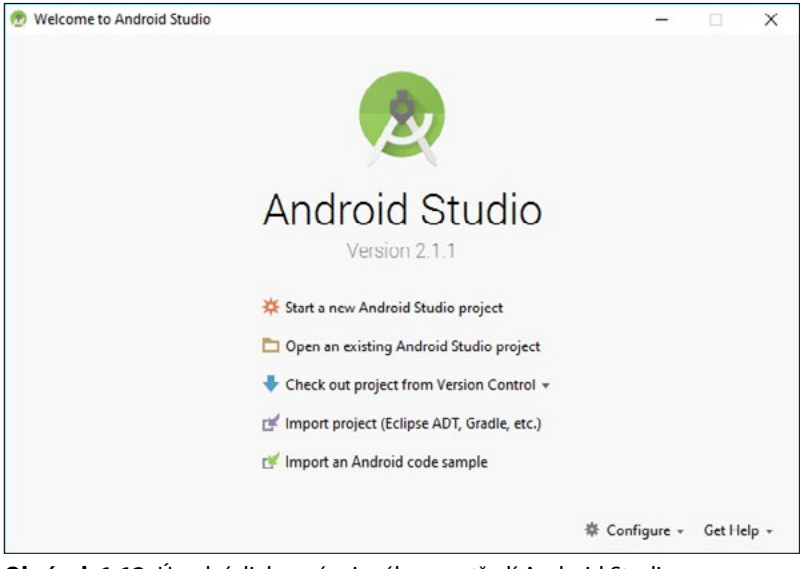

**Obrázek 1.10:** Volba Custom umožní vybrat barevné schéma uživatelského rozhraní Android Studia

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

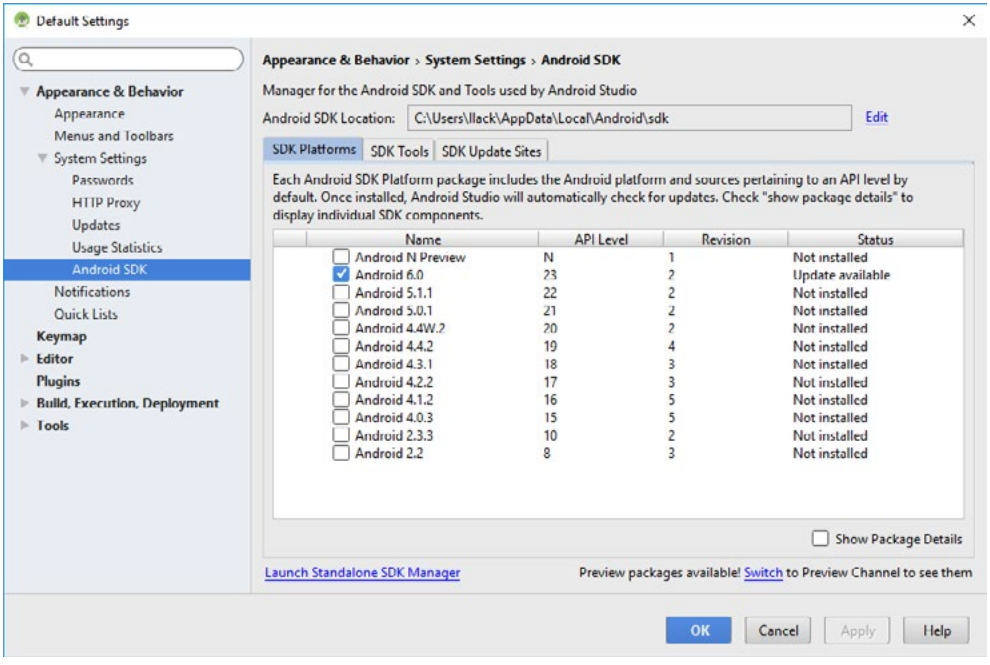
Po nainstalování a prvním spuštění doporučujeme zkontrolovat, pro které verze Androidu máte nainstalovanou podporu. V úvodním dialogu proto ve spodní části klepněte na položku **Configure** a z nové nabídky vyberte **SDK Manager**.




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




**Obrázek 1.12:** Úvodní dialog vývojového prostředí Android Studio




**Obrázek 1.13:** SDK Manager

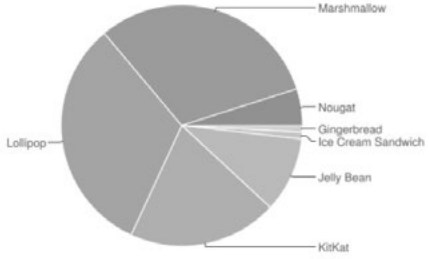
V okně **Android SDK Manager** si všimněte tří karet: **SDK Platforms**, **SDK Tools** a **SDK Update Sites**. Na kartě **SDK Platforms** je seznam SDK pro jednotlivé majoritní verze Androidu. Implicitně je nainstalované SDK pro Android 6.0 Marshmallow, jehož API má verzi 23. Jelikož v době psaní této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

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

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

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

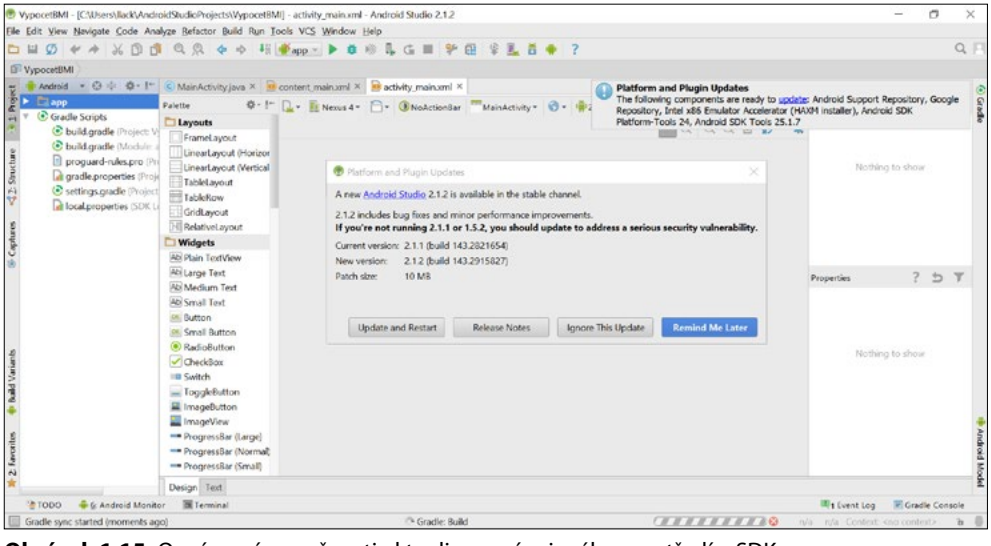
| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 0.9% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 0.9% |
| 4.1.x | Jelly Bean | 16 | 3.5% |
| 4.2.x | | 17 | 5.1% |
| 4.3 | | 18 | 1.5% |
| 4.4 | KitKat | 19 | 20.0% |
| 5.0 | Lollipop | 21 | 9.0% |
|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




**Obrázek 1.14:** Procentuální zastoupení verzí Androidu (duben 2017)

# Průběžná aktualizace

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




**Obrázek 1.15:** Oznámení o možnosti aktualizace vývojového prostředí a SDK

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

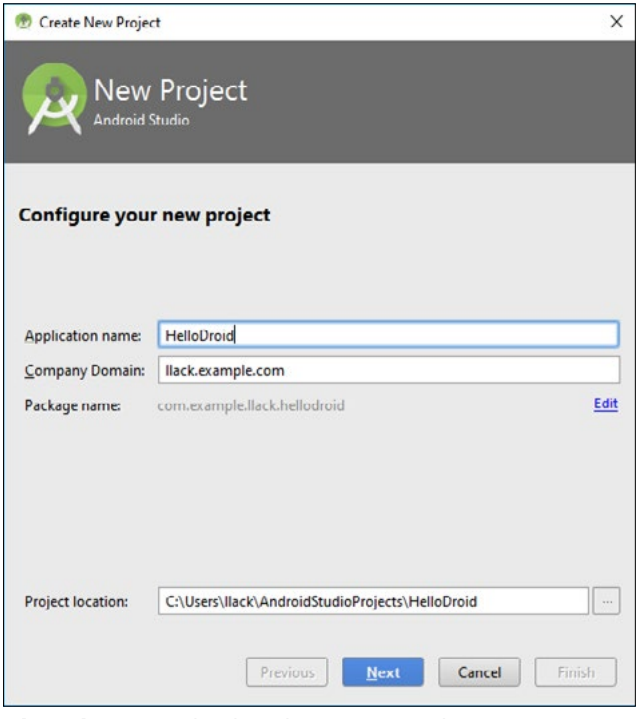
# Vytvoření projektu aplikace

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

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

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

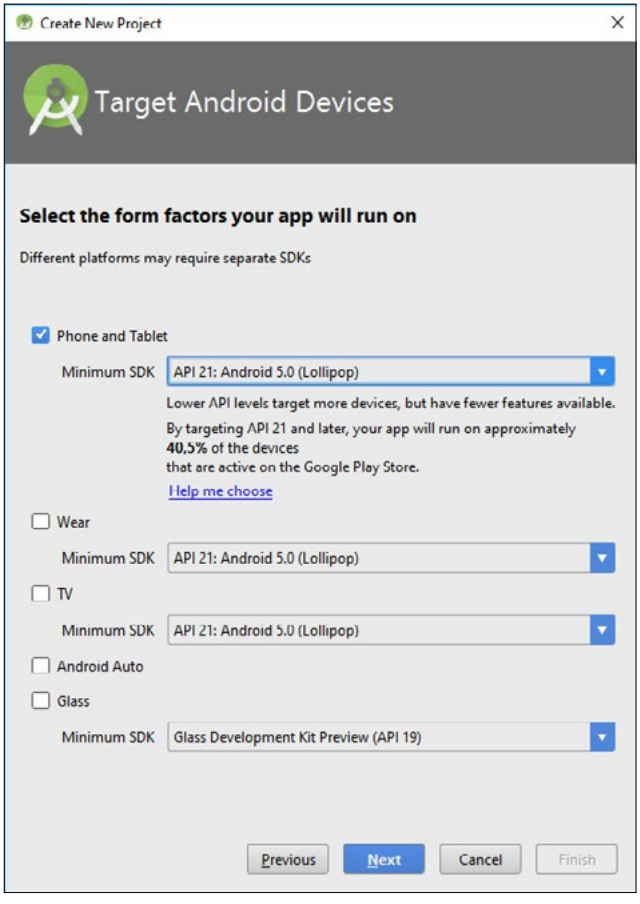
Spusťte vývojové prostředí Android Studio. V úvodním dialogu klepněte na položku **Start a new Android Studio project**. V úvodním dialogu průvodce vytvořením projektu je potřeba zadat název projektu a případně i firemní či vývojářovu doménu.




**Obrázek 1.16:** Android Studio Setup Wizard

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

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

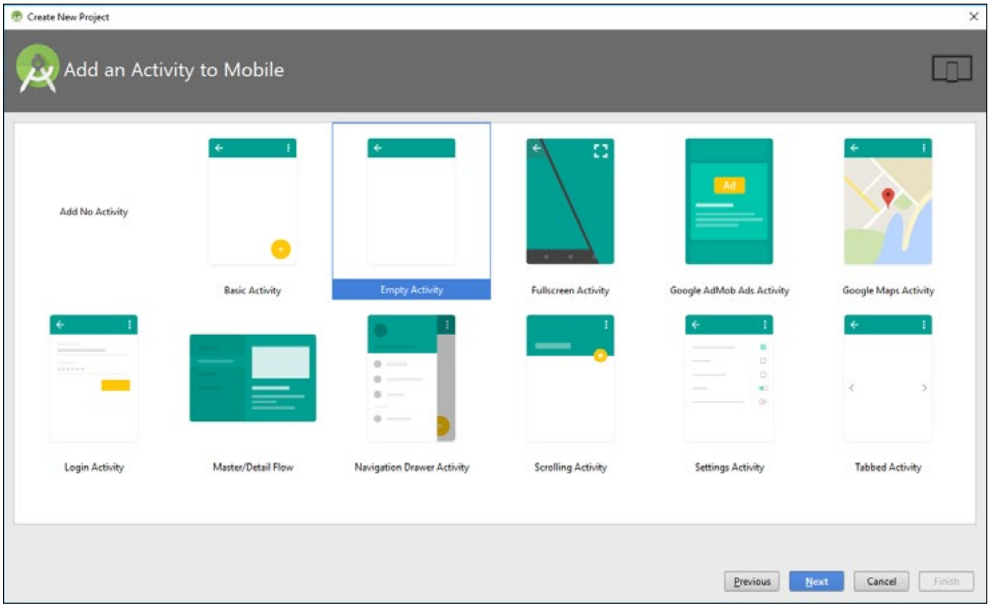



**Obrázek 1.17:** Výběr nejnižší podporované verze Androidu

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

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

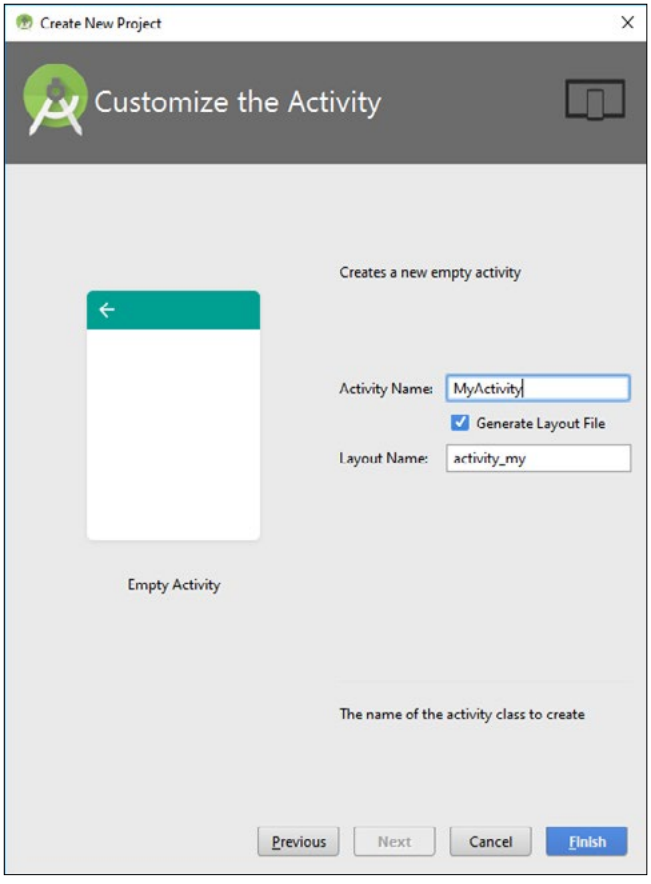
V dalším dialogu můžete zvolit několik typů pro hlavní aktivity aplikace včetně mapové či aktivity typu master-detail. Další postup se odvíjí od nastaveného typu hlavní aktivity. Například pokud jste zvolili typ master-detail, systém vás vyzve k definování názvu položky a skupiny položek. V našem příkladu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




**Obrázek 1.18:** Výběr typu hlavní aktivity

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

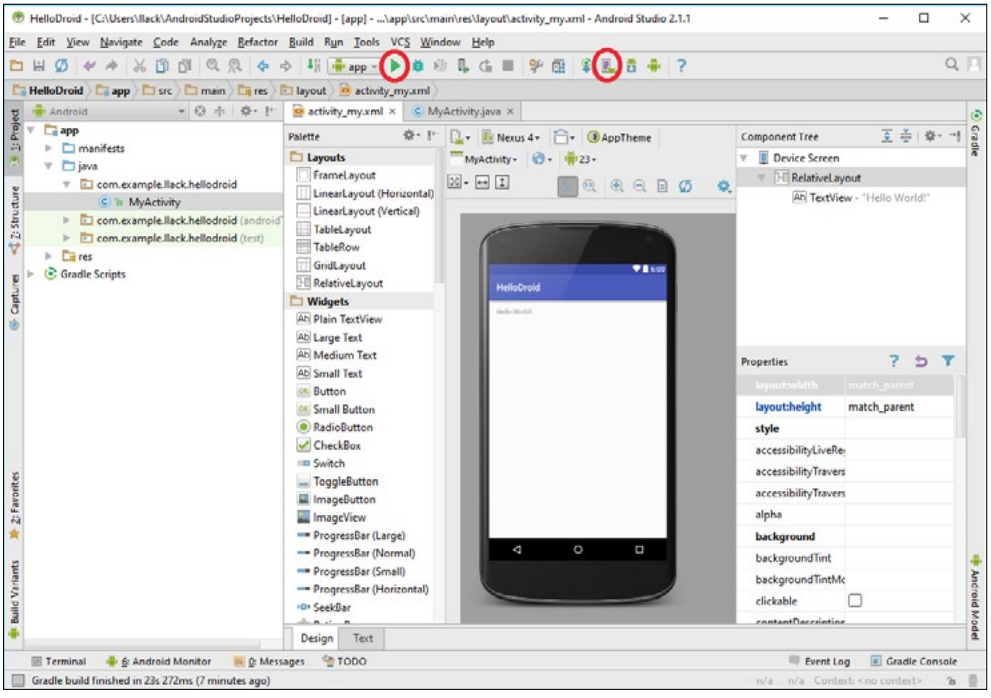
Než se vytvoří všechny potřebné soubory tvořící projekt, bude se několik sekund zobrazovat pouze šedé okno Android Studia s ukazatelem průběhu v dolní části a odhadem času, kolik sekund ještě příprava zabere. Pokud jste dosud nepovolili firevall pro Javu, zobrazí se vám dialog, ve kterém to můžete provést.



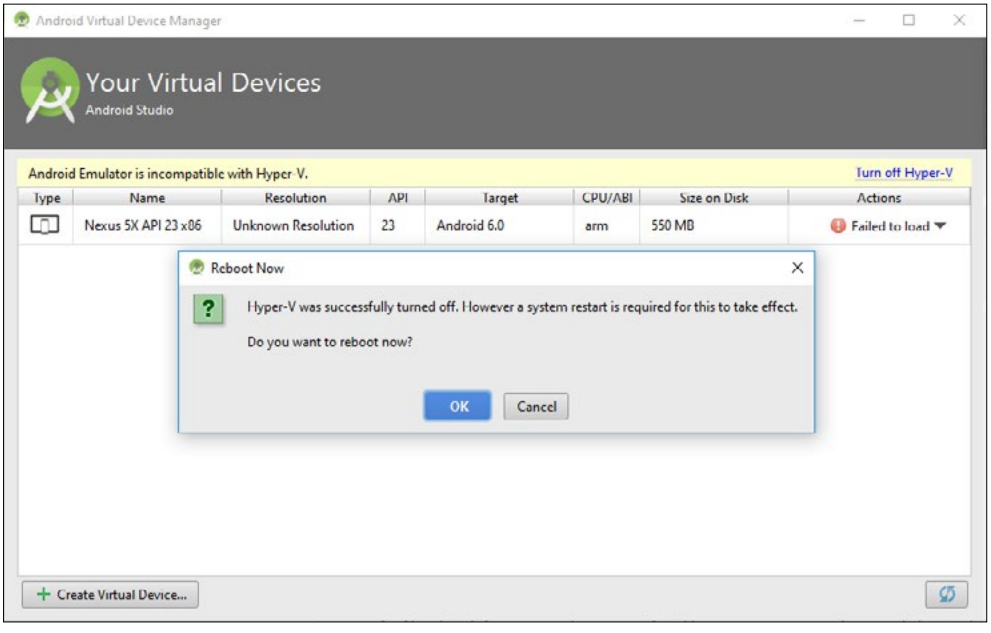

**Obrázek 1.19:** Vytvoření hlavní aktivity

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

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

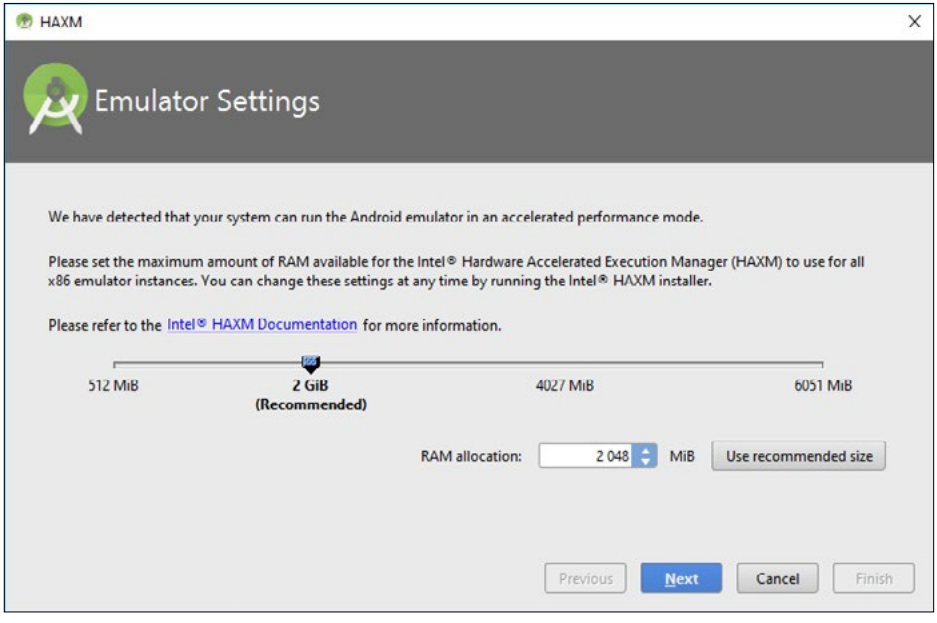



**Obrázek 1.20:** Nový projekt v Android Studiu. Označené jsou ikony na spuštění aplikace a ikona AVD manageru na konfiguraci emulátorů




**Obrázek 1.21:** Oznámení o nutnosti vypnutí Hyper-V

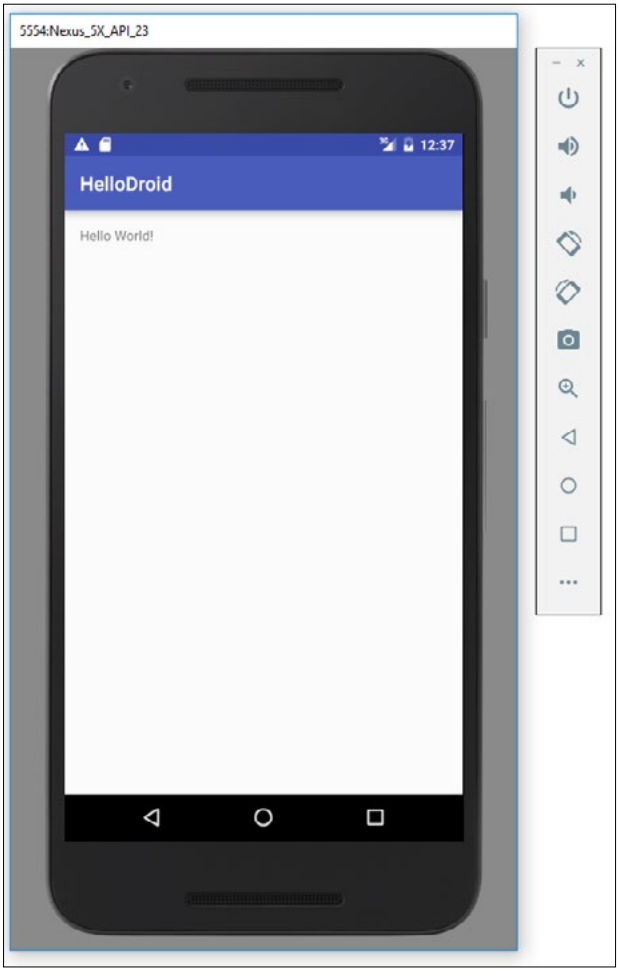
Po restartu vynuceném vypnutím hypervizoru Hyper-V bude potřeba nainstalovat Intel Hardware Accellerated Execution Manager (HAXM) a vyhradit mu adekvátní část paměti RAM. V našem případě měl počítač instalovaných 8 GB RAM a pro HAXM jsme vyhradili doporučenou velkost 2 GB.




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

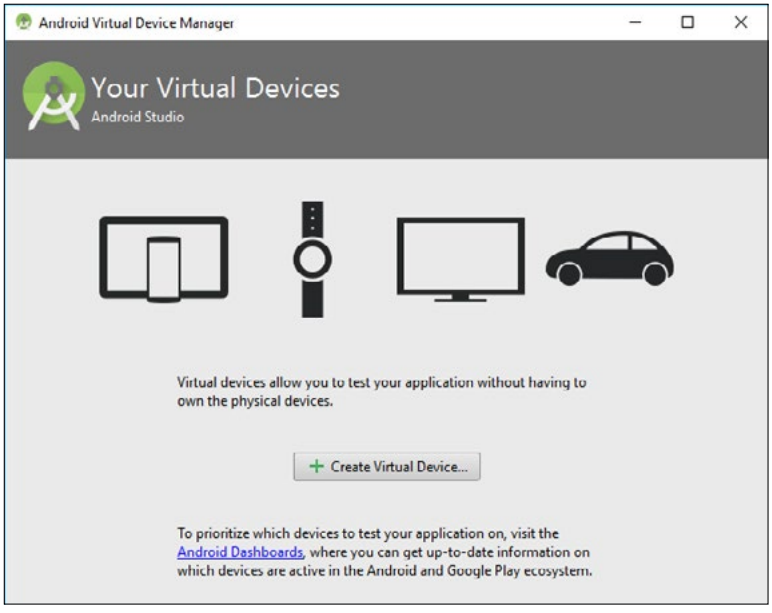
Tento úkon, tedy vypnutí Hyper-V a instalace HAXM, se samozřejmě provádí pouze jednou. Proto jsme zařadili projekt typu Hello World do kapitoly o instalaci, jelikož až po přeložení, linkovaní, spuštění na emulátoru a následně i spuštění projektu na reálném zařízení máte jistotu, že vývojové prostředí,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

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

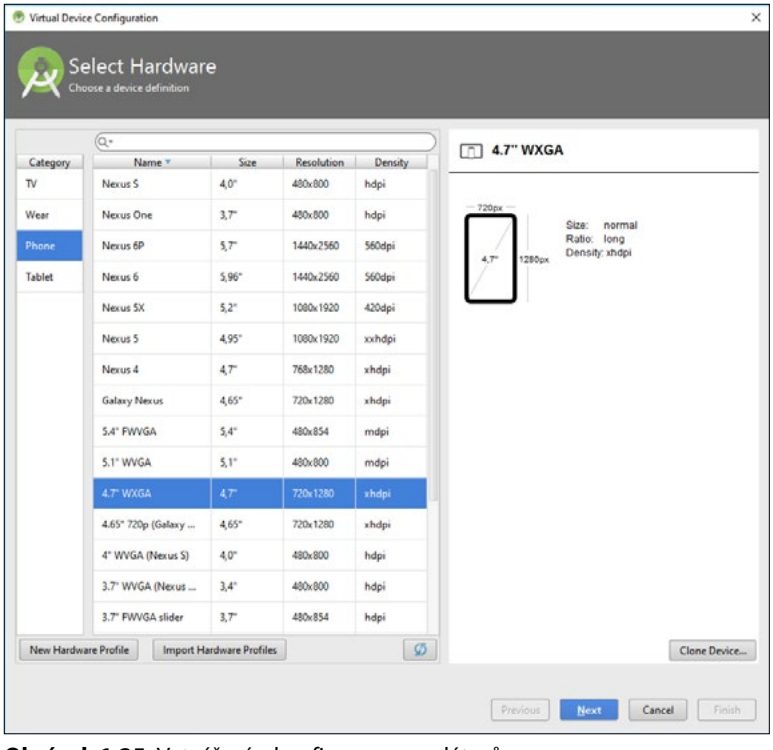



**Obrázek 1.23:** Spuštění aplikace v emulátoru

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




**Obrázek 1.24:** Android Virtual Device Manager

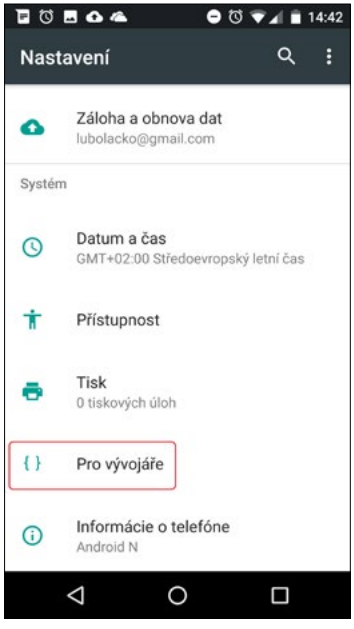



**Obrázek 1.25:** Vytváření a konfigurace emulátorů

# Odemknutí zařízení pro vývoj

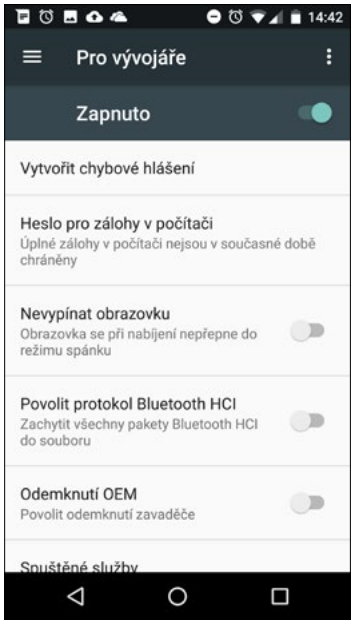
Abyste mohli vytvářet a spouštět aplikace na svém zařízení (bez ohledu na to, zda se jedná o telefon nebo tablet) musíte nejdříve aplikovat následující postup. V aplikaci **Nastavení** přejděte do nabídky **Informace o zařízení** a následně vyhledejte položku **Číslo sestavení**. U některých nadstaveb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


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




**Obrázek 1.27:** Položka Nastavení Pro vývojáře po odemknutí

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

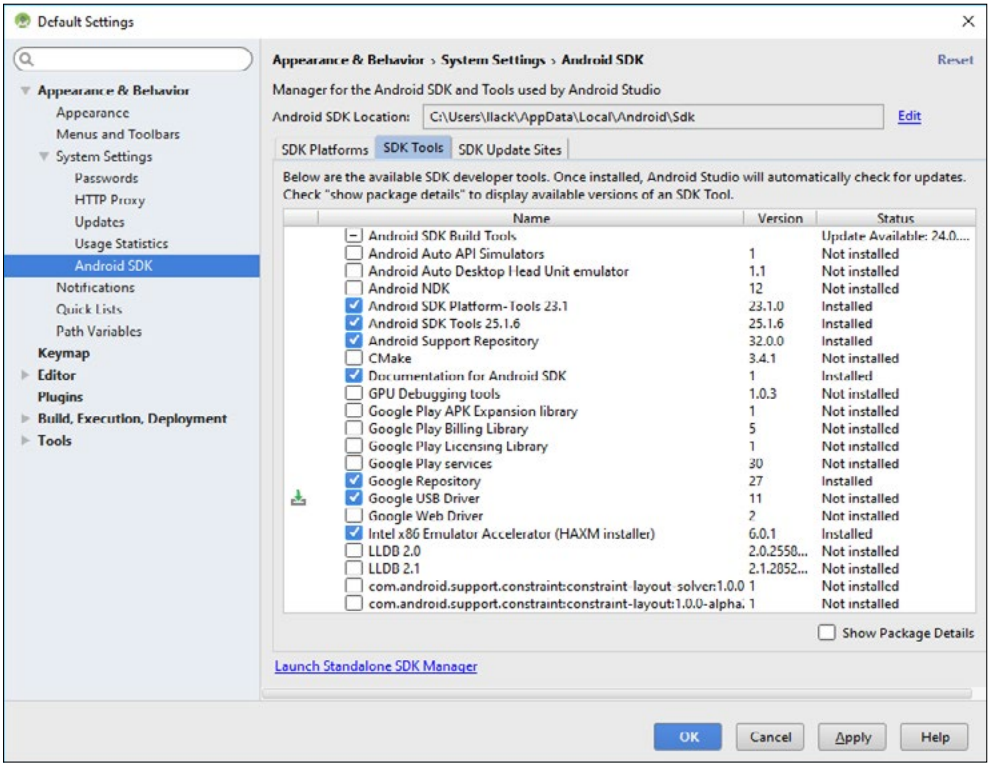



**Obrázek 1.28:** Seznam voleb, které můžete nastavit v nabídce Pro vývojáře

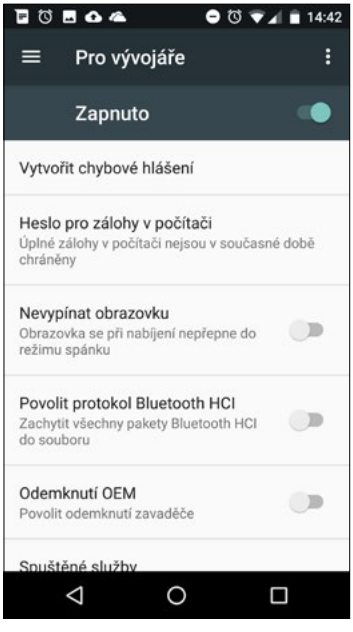
Na straně počítače potřebujete pro komunikaci se zařízením s Androidem USB ovladače pro ADB (Android Debug Bridge). Buď použijete ovladač Google USB Driver, který je součástí Android SDK, nebo ovladač dodaný výrobcem zařízení.

## Spuštění aplikace na reálném zařízení

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

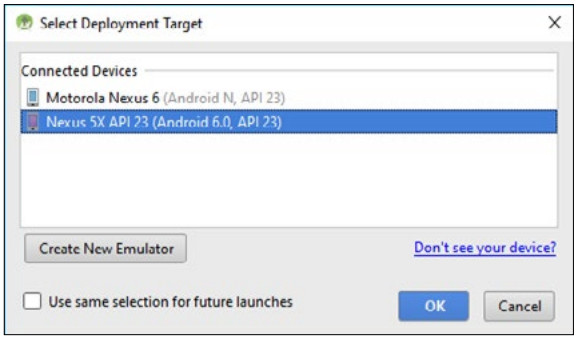



**Obrázek 1.29:** Instalace ovladače Google USB Driver přes SDK Manager




**Obrázek 1.30:** Zapnutí ladění přes USB na připojeném zařízení

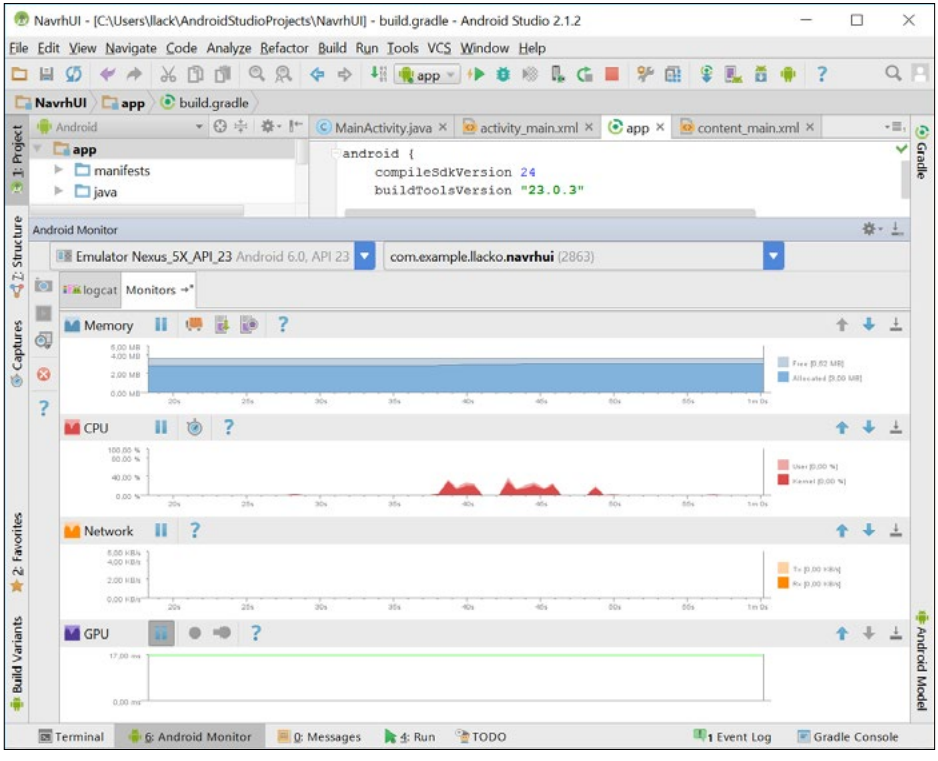
Po správném nakonfigurování USB ovladače pro ADB se při pokusu o spuštění aplikace zobrazí nabídka možností.




**Obrázek 1.31:** Výběr hardwarového zařízení ke spuštění aplikace

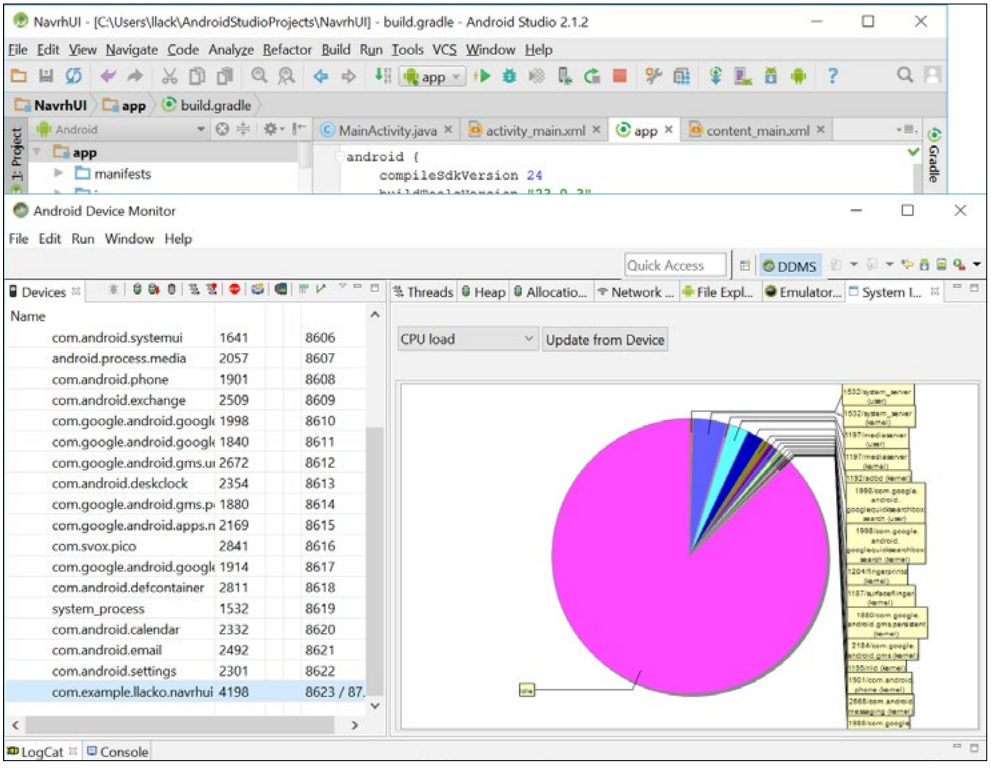
## Monitorování spuštěné aplikace

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




**Obrázek 1.32:** Monitorování spuštěné aplikace v okně Android Monitor

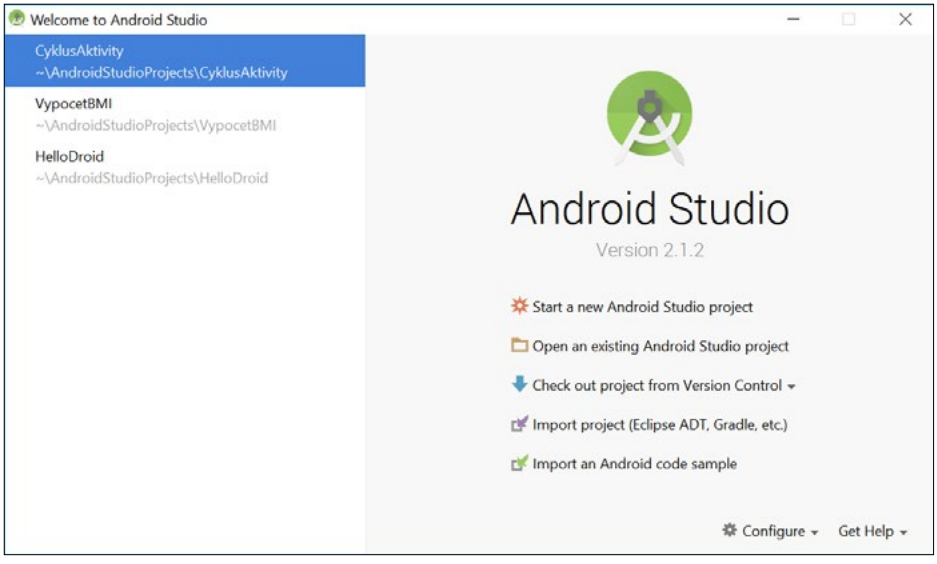
K monitorování spuštěné aplikace můžete využít i **Android Device Monitor**. Z Android Studia tento nástroj spustíte pomocí nabídky **Tools → Android → Android Device Monitor**.




**Obrázek 1.33:** Monitorování spuštěné aplikace pomocí nástroje Android Device Monitor

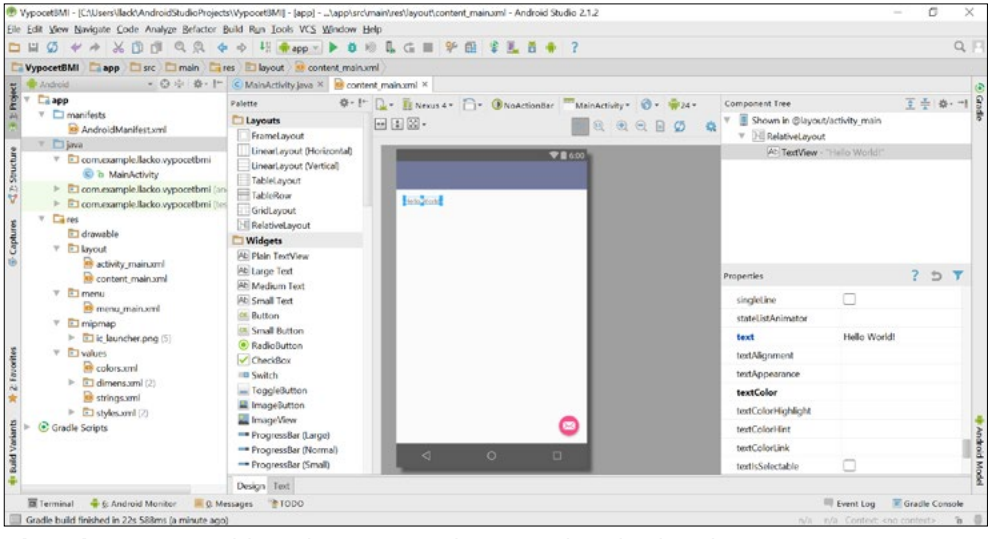
# Seznámení se s vývojovým prostředím

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



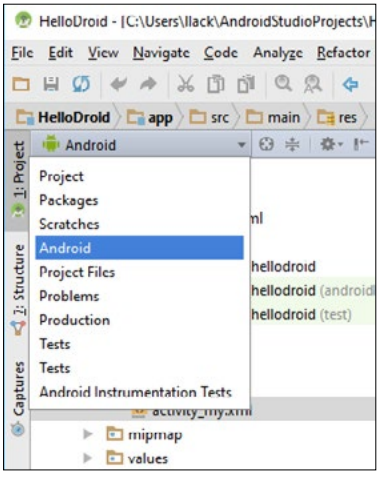

**Obrázek 1.34:** Úvodní dialog vývojového prostředí Android Studio

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



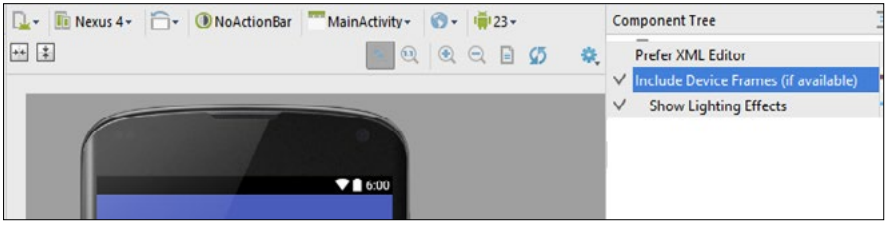
**Obrázek 1.35:** Uživatelské rozhraní vývojového prostředí Android Studio

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




**Obrázek 1.36:** Možnosti zobrazení struktury projektu

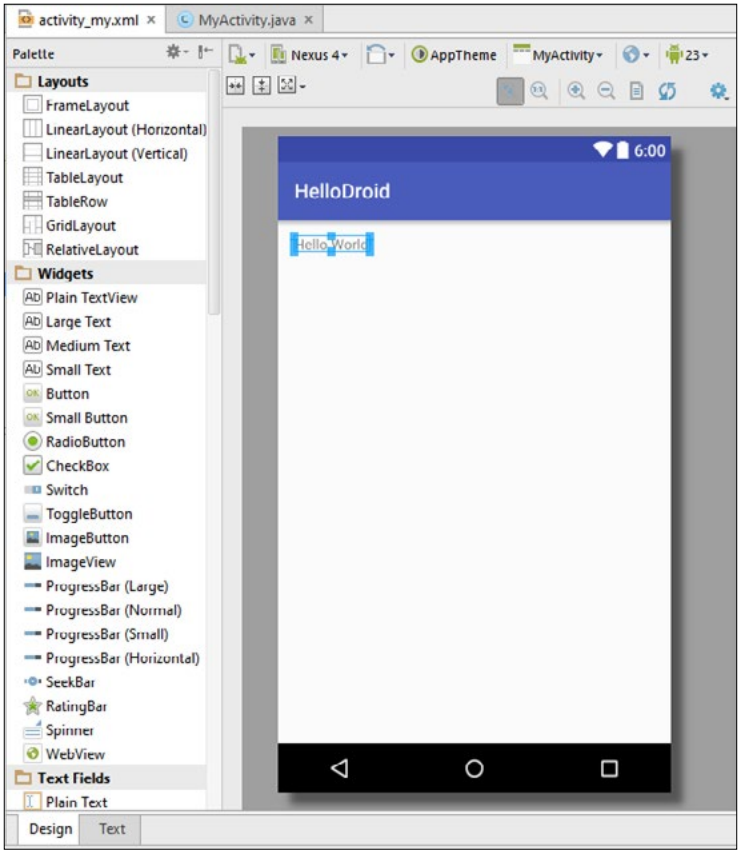
Dominantním při návrhu uživatelského rozhraní je náhledové okno. Po nainstalování se zobrazuje i s rámečkem znázorňujícím tělo telefonu. Na vývojářských noteboocích s menšími displeji je to tak trochu plýtvání místem. Tento rámeček lze ale naštěstí vypnout. Vpravo nahoře na liště okna najdete ikonu s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




**Obrázek 1.37:** Nastavení zobrazení – možnost vypnutí rámečku

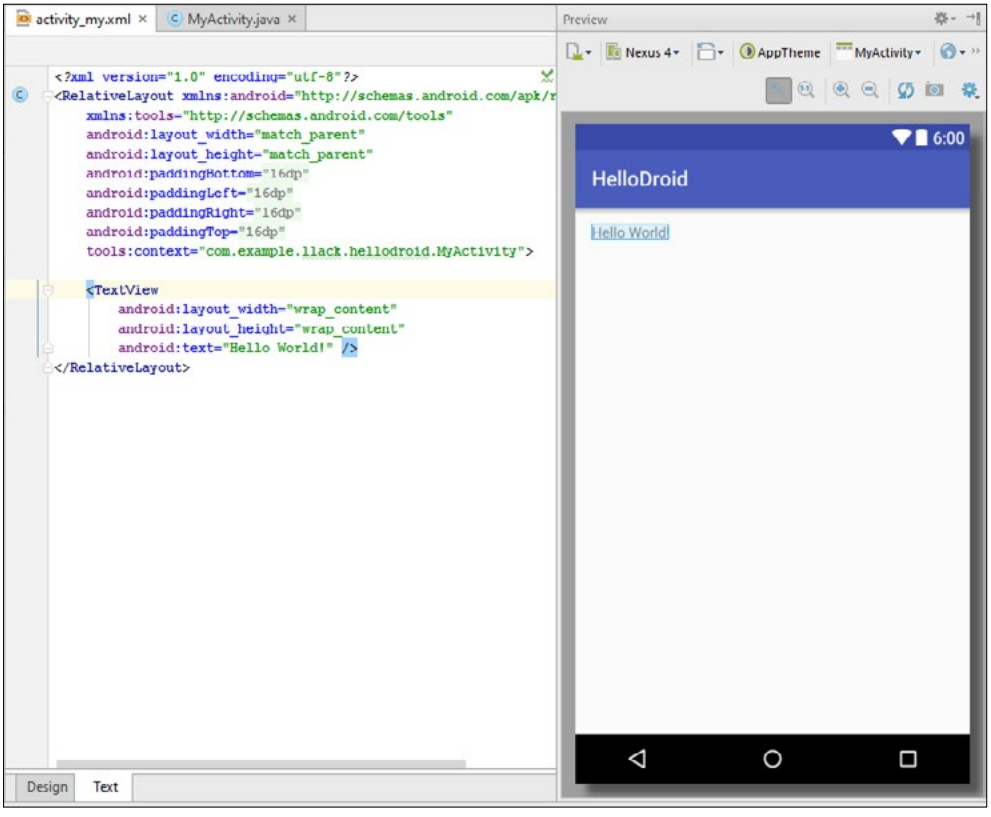
Uživatelské rozhraní můžete navrhnout ve dvou módech: **Design** a **Text**. Přepínají se pomocí karet ve spodní části okna.

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



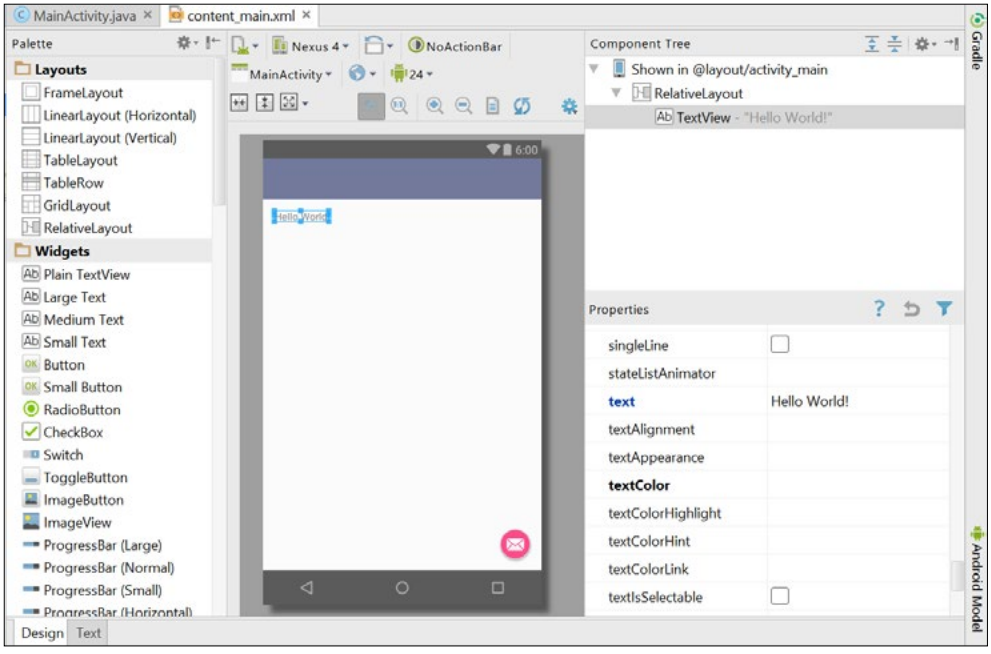

**Obrázek 1.38:** Návrh uživatelského rozhraní – mód Design

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



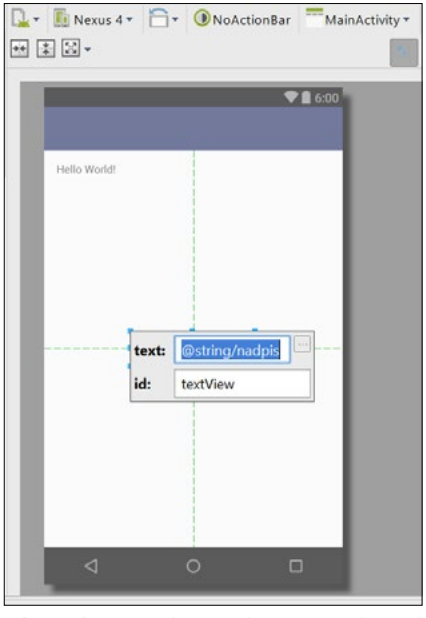

**Obrázek 1.39:** Návrh uživatelského rozhraní – mód Text

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



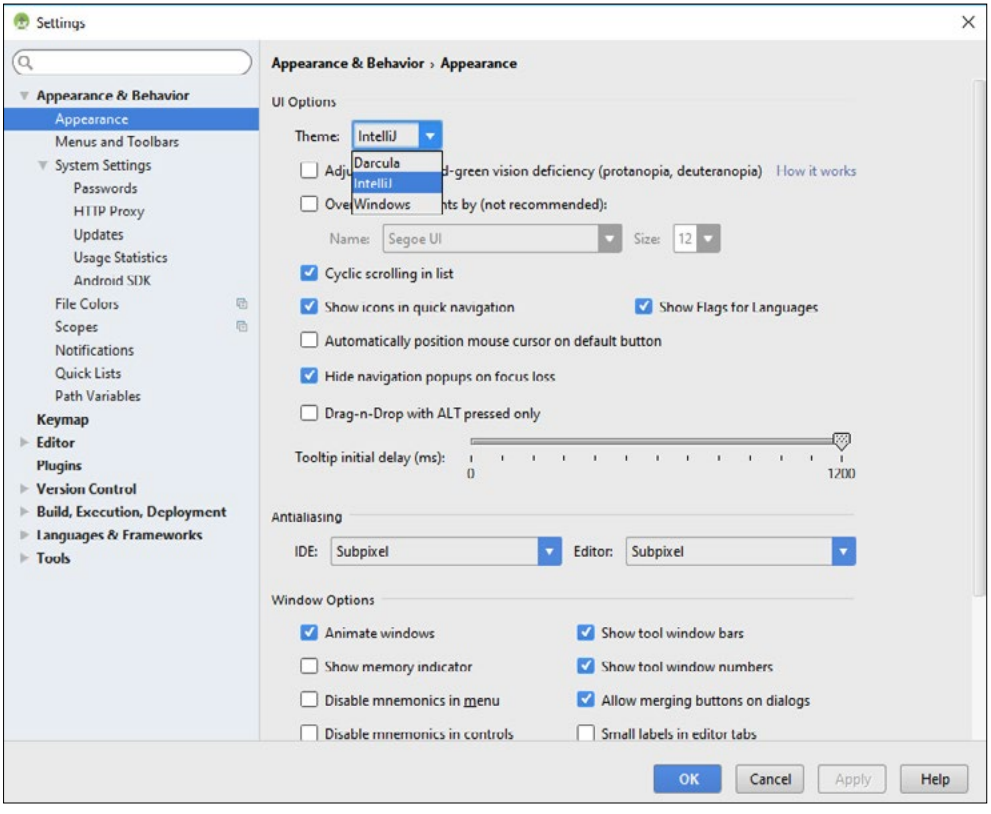

**Obrázek 1.40:** Vlastnosti vybraného prvku v okně Properties

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



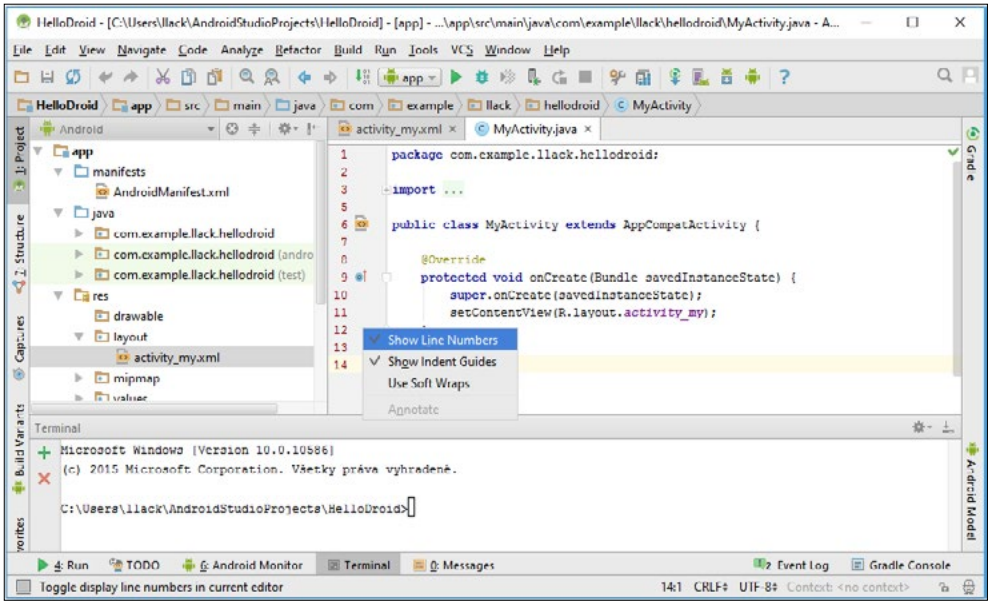

**Obrázek 1.41:** Klíčové vlastnosti vybraného prvku

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




**Obrázek 1.42:** Změna barevného schématu vývojového prostředí Android Studio

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

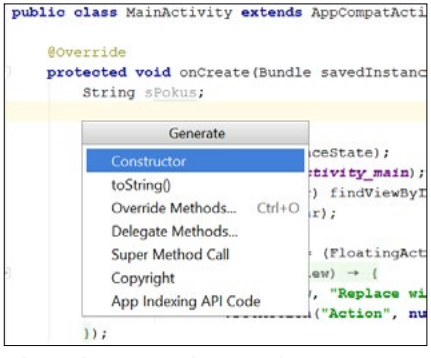



**Obrázek 1.43:** Zobrazení zdrojového kódu v programovacím jazyce Java

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

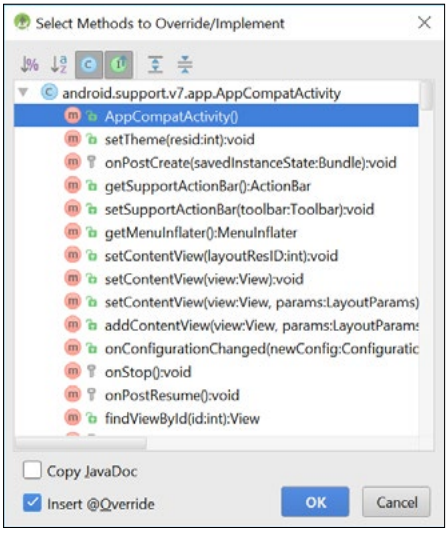
## Editor kódu v jazyce Java

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



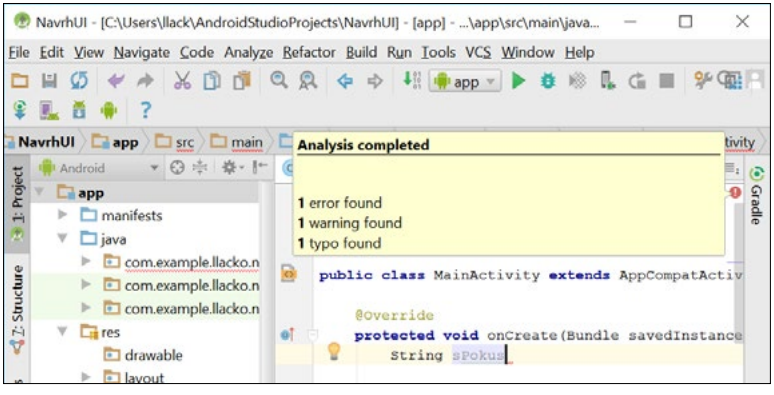

**Obrázek 1.44:** Dialogové okno s automatickým generováním bloků kódu

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



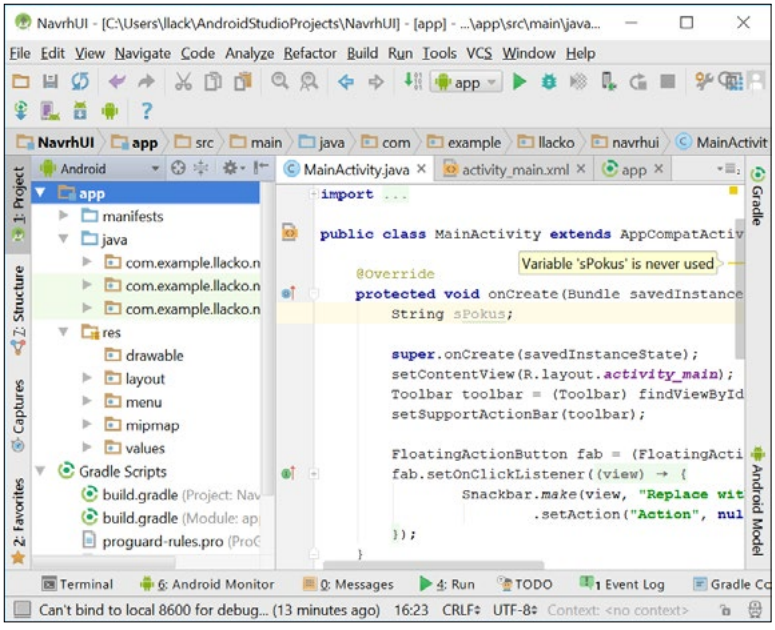

**Obrázek 1.45:** Nabídka metod Override

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




**Obrázek 1.46:** Výsledek průběžné analýzy kódu

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




**Obrázek 1.47:** Upozornění, v tomto případě na nepoužitou proměnnou

## Java pro migranty

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

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

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

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

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

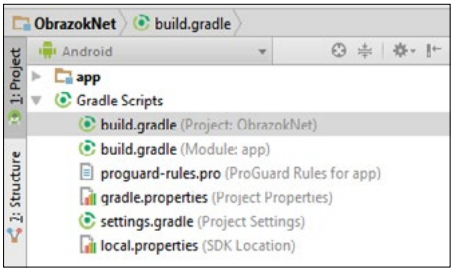
nujete přístupovou úroveň `protected`, můžete k takto definovaným členům třídy přistupovat z libovolného místa v rámci balíčku, avšak mimo něj pouze z podtříd dané třídy. Rodičovská třída je definovaná pomocí klíčového slova `extends` a metody rodičovské třídy voláme pomocí klíčového slova `super`. Toto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

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

## Gradle

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

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




**Obrázek 1.48:** Struktura zobrazení složky Gradle v Android Studiu

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

Příklad skriptu:

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

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

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

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

allprojects {
  repositories {
    jcenter()
  }
}

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

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

```
include ':app'
```

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

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

android {
  compileSdkVersion 24
  buildToolsVersion "23.0.3"

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

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

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

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

android {
  compileSdkVersion 24
  buildToolsVersion "23.0.3"
```

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

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

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

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

```
> gradlew.bat assembleRelease
```

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

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

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

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

# Možnosti současného hardwaru

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

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

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

# Referenční zařízení

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

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

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

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

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

Technické údaje Nexusu 5X:

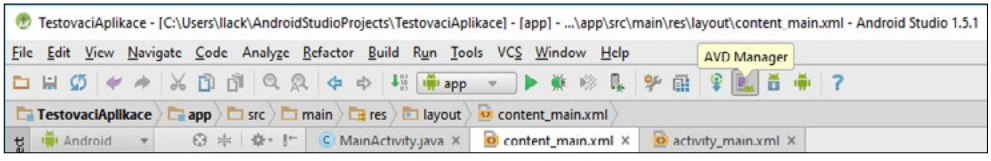
| | |
|---|---|
| **Procesor a grafika:** | Šestijádrový Qualcomm Snapdragon 808, 1,8 GHz, GPU Adreno 418 |
| **Paměť:** | 2 GB RAM, úložný prostor 16 GB nebo 32 GB |
| **Displej:** | 5,2“ IPS s rozlišením 1 920 × 1 080, 423 ppi |
| **Fotoaparát:** | Zadní: 12,3 MP, Přední: 5 MP, oba se světelností f/2.0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


**Obrázek 1.49:** Referenční telefon NEXUS 5X

# Vytvoření emulátoru

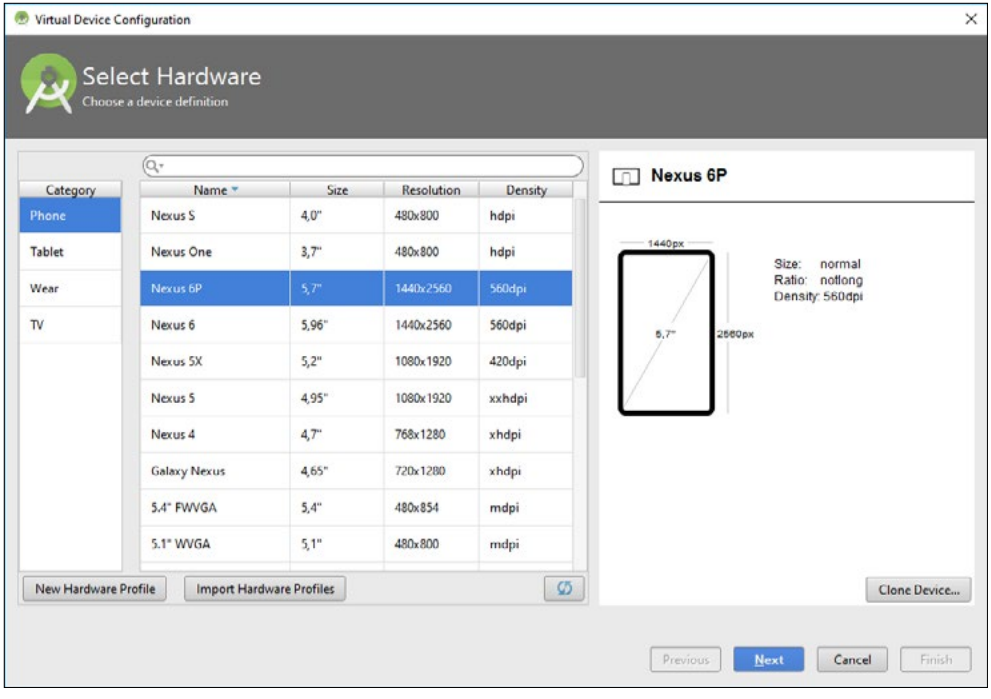
Zdálo by se, že pokud máte k dispozici jeden moderní chytrý telefon a jeden tablet s Androidem, dokážete pokrýt návrh, testování a ladění aplikací. Opak je pravdou. Výhodou a zároveň nevýhodou Androidu je variabilita různých zařízení s různými verzemi systému a různým rozlišením displeje. Neodmyslitelnou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

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



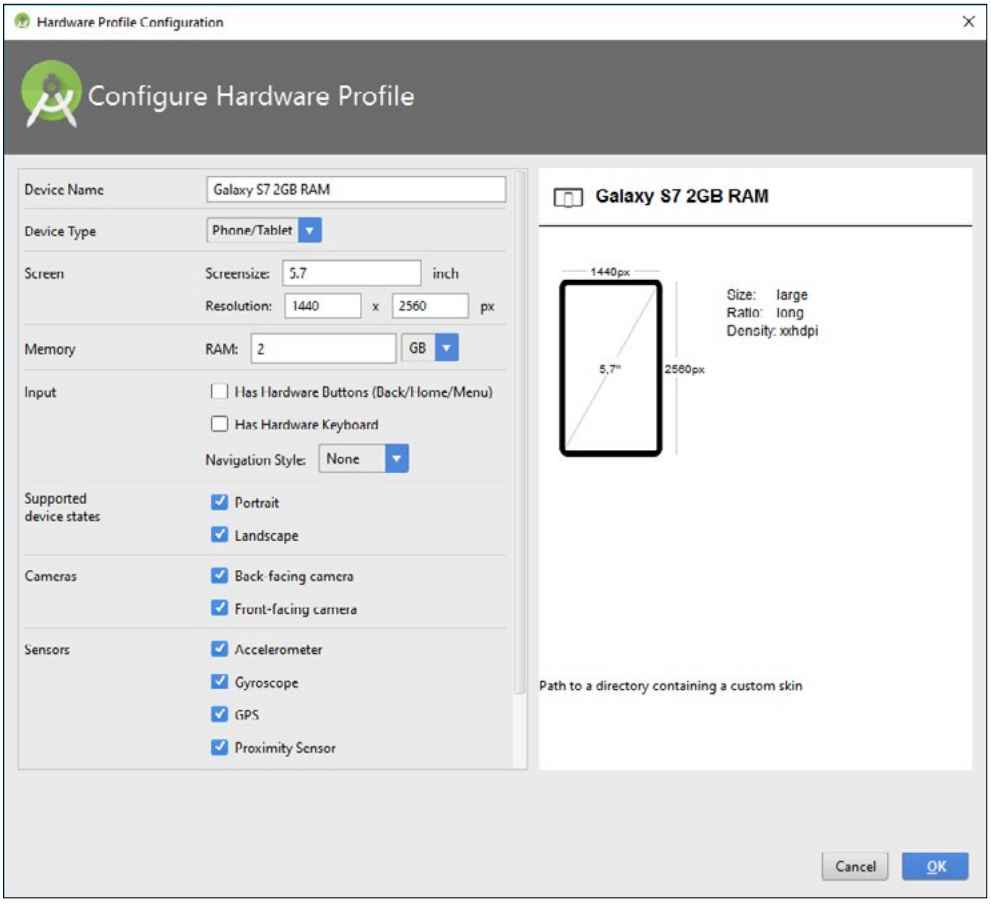

**Obrázek 1.50:** Ikona AVD Managera v aplikaci Android Studio

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



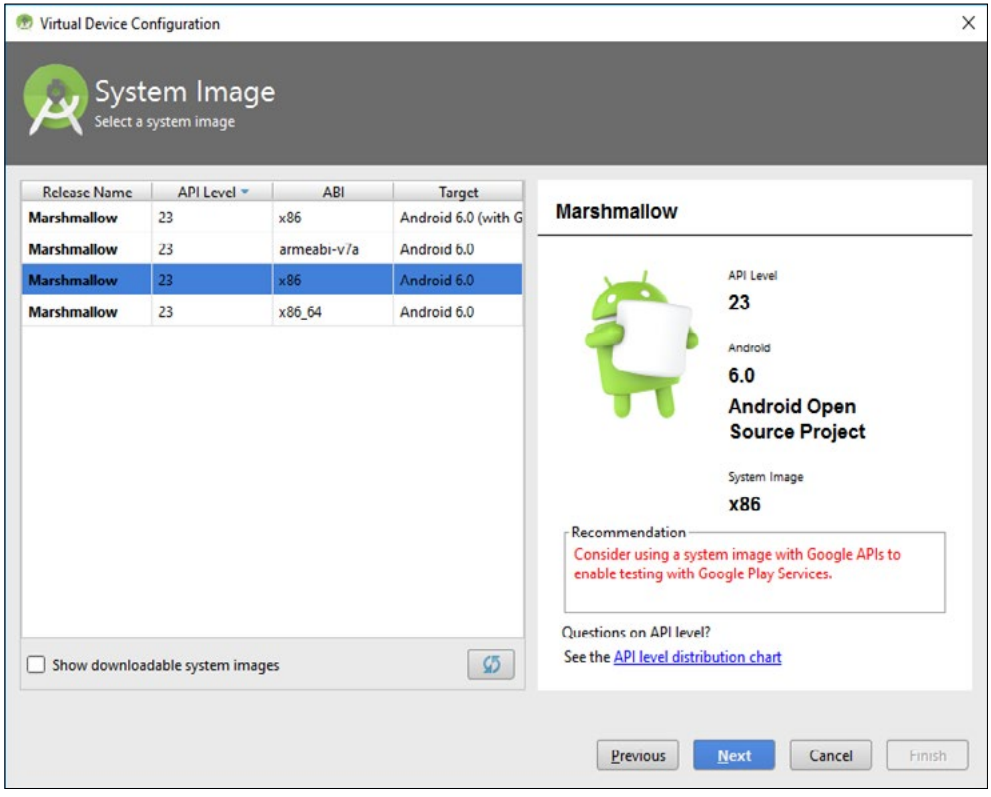

**Obrázek 1.51:** Vytvoření emulátoru

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




**Obrázek 1.52:** Vytvoření emulátoru klonováním

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

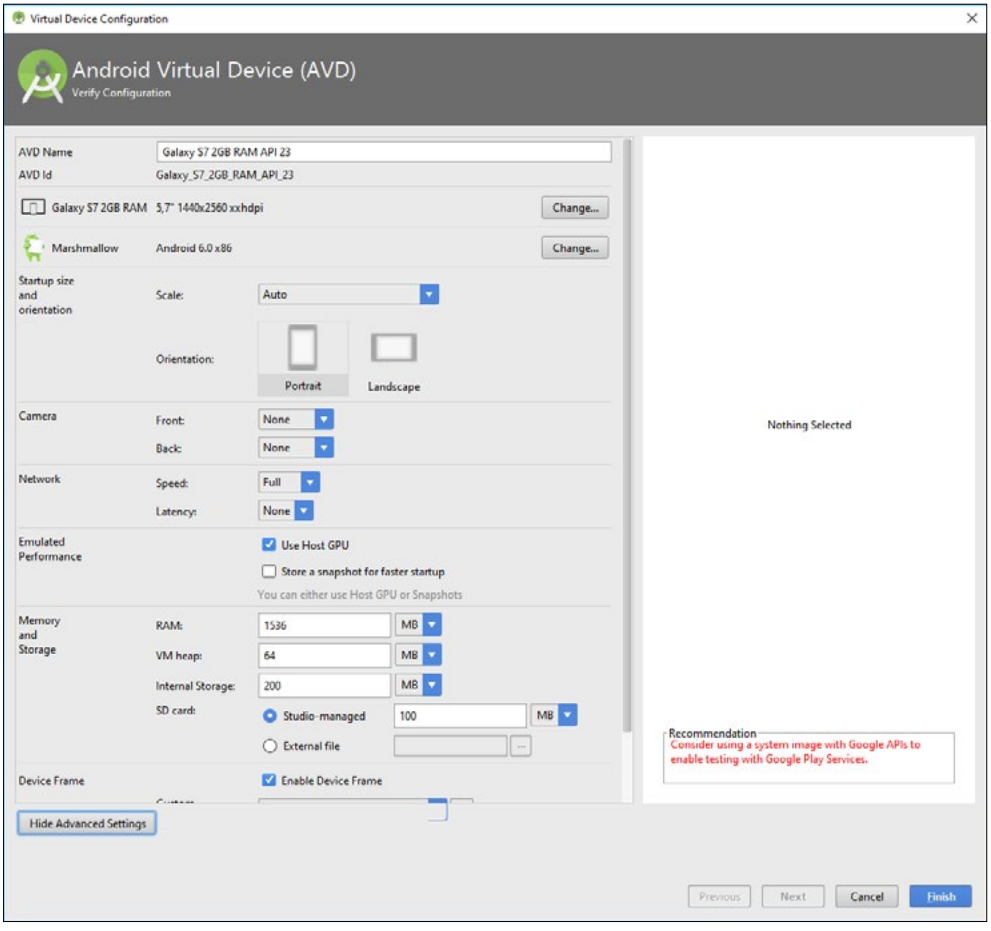



**Obrázek 1.53:** Vytvoření emulátoru – výběr verze operačního systému

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

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

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




**Obrázek 1.54:** Vytvoření emulátoru – nastavení parametrů

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

# GitHub

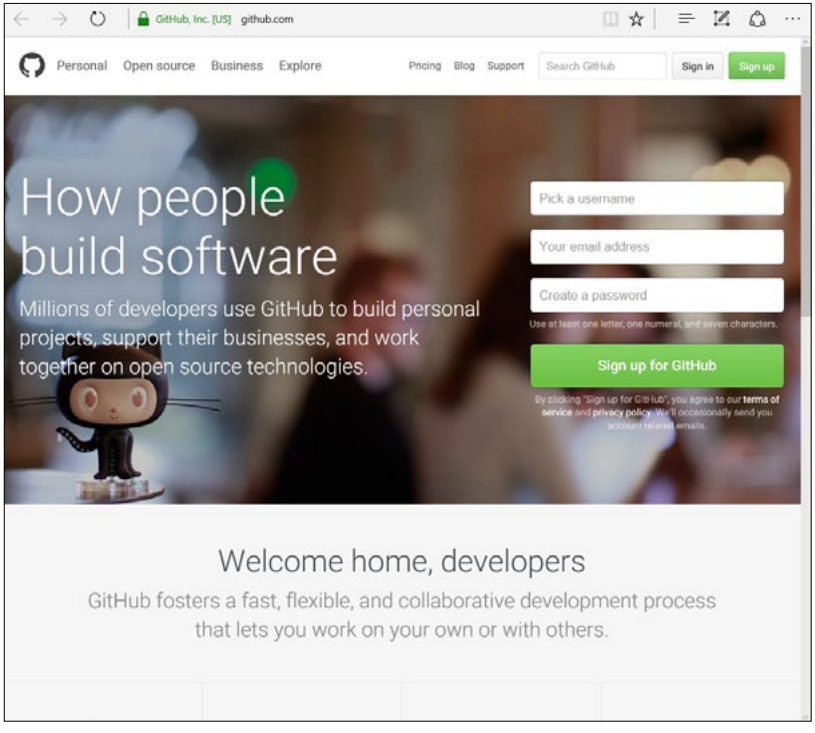
Správa verzí zdrojového kódu aplikací je důležitá především u týmového vývoje, ale díky řadě výhod ji využívají i samostatní vývojáři. Předchází tak mnohým problémům souvisejícím se ztrátou zdrojových a jiných souborů, případně jejich přepsáním. I samostatní vývojáři často pracují na projektu z více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

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

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

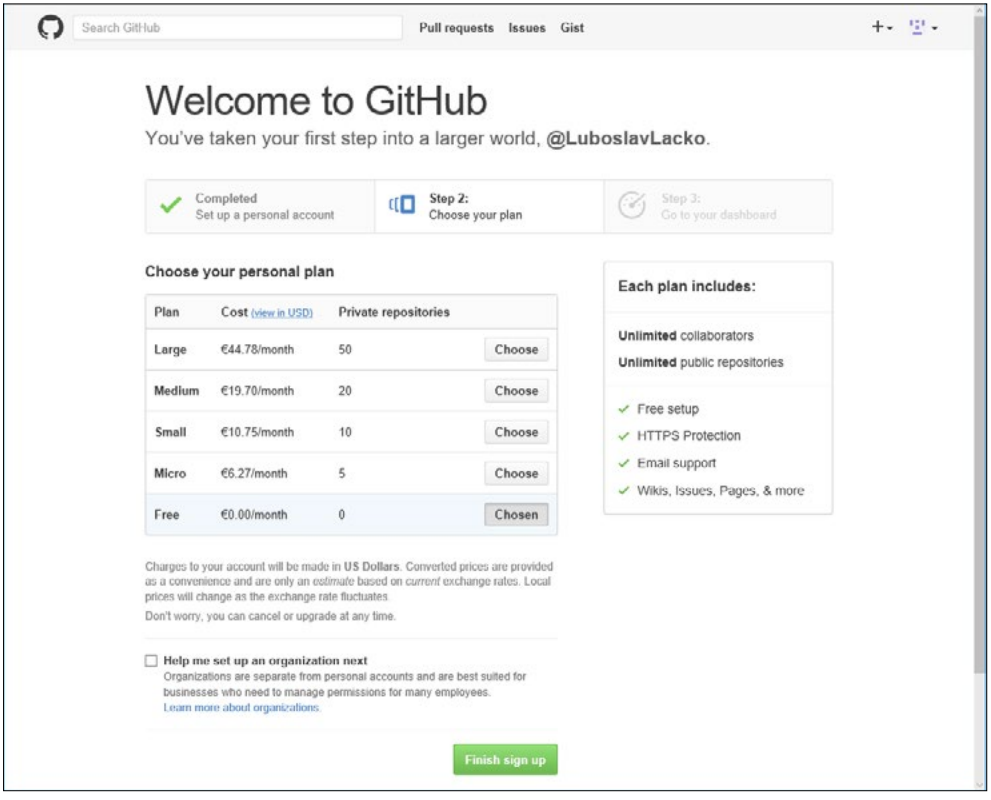
## Vytvoření účtu

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



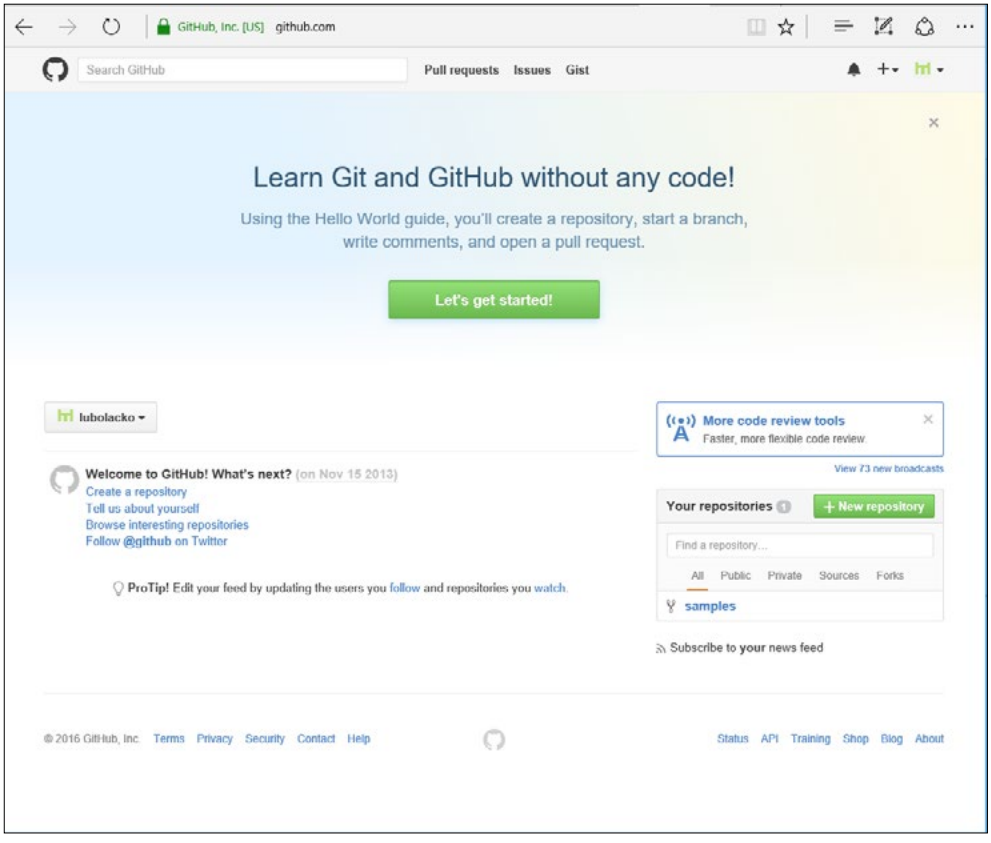

**Obrázek 1.55:** Vytvoření účtu ve službě github.com

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




**Obrázek 1.56:** Cenové varianty služby

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




**Obrázek 1.57:** Úvodní stránka zobrazená po vytvoření služby

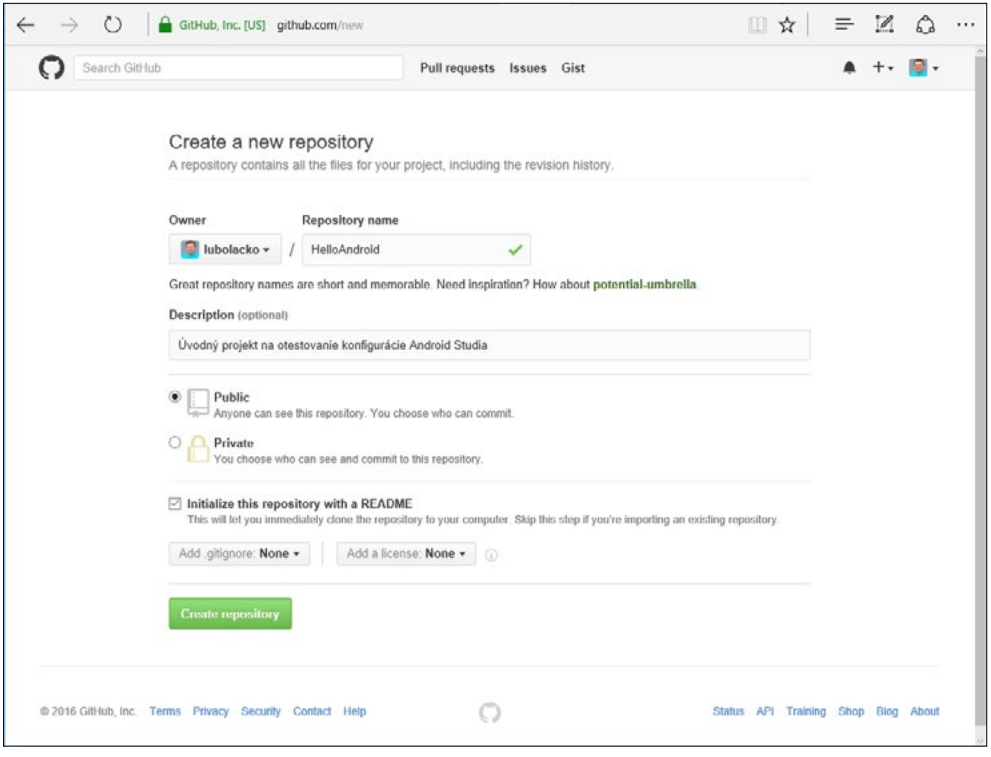
Doporučujeme v profilu aktualizovat fotku vašeho avatara, případně zveřejnit jméno či e-mailovou adresu. Pro komunitu vývojářů je přece důležitá především vzájemná spolupráce a sdílení poznatků.

## Vytvoření repozitáře

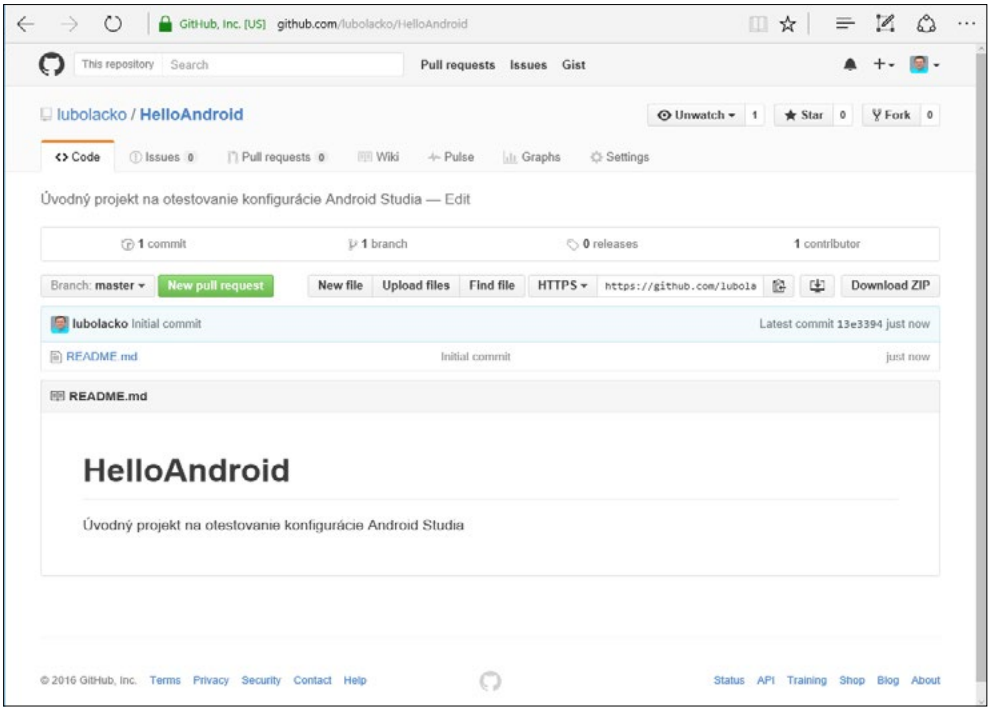
Pokud jste server GitHub dosud nevyužívali, doporučujeme absolvovat krátký, přibližně 10minutový tutoriál, který nabízí hlavní stránka projektu po prvním přihlášení. Nejprve je potřeba vytvořit nový repozitář. Repozitář si nejjednodušeji představíte jako složku, ve které máte umístěný jeden projekt,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

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

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



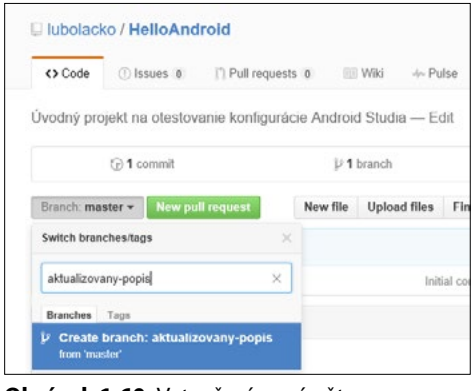

**Obrázek 1.58:** Vytvoření repozitáře




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

## Vytvoření větve (branch)

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




**Obrázek 1.60:** Vytvoření nové větve

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

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

## Úprava souborů

Git spravovaným souborům přiděluje tři základní stavy:

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

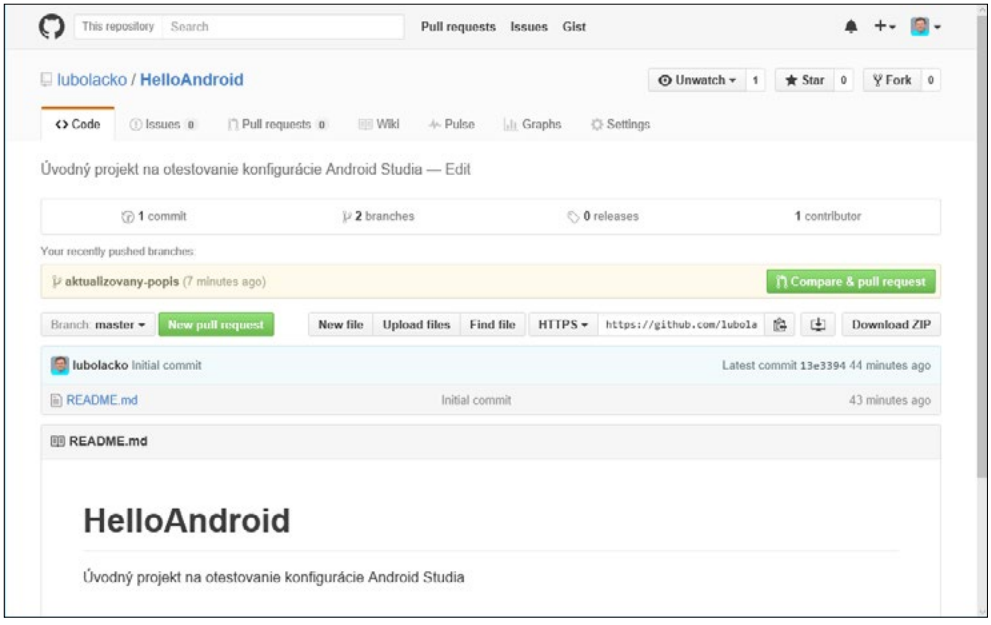
Změny si vyzkoušejte na editování textového souboru *README.md*. Klepněte na název souboru a následně na ikonu tužky v pravém horním rohu. Otevře se okno, ve kterém můžete editovat obsah souboru. Změny potvrdíte tlačítkem **Commit changes**. Operace **Commit** umožňuje uložení provedených úprav, během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

Na synchronizaci zdrojových kódů jsou určeny operace:

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

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

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




**Obrázek 1.61:** Aktualizovaný popis

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

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

KAPITOLA

2

# Anatomie Androidu

Témata kapitoly:

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

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

## Multiplatformní operační systém

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

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

# Historie verzí

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

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


**Obrázek 2.1:** Loga verzí Androidu

# Starší verze

Nejprve představíme historické verze:

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

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

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

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

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



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

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

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

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

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

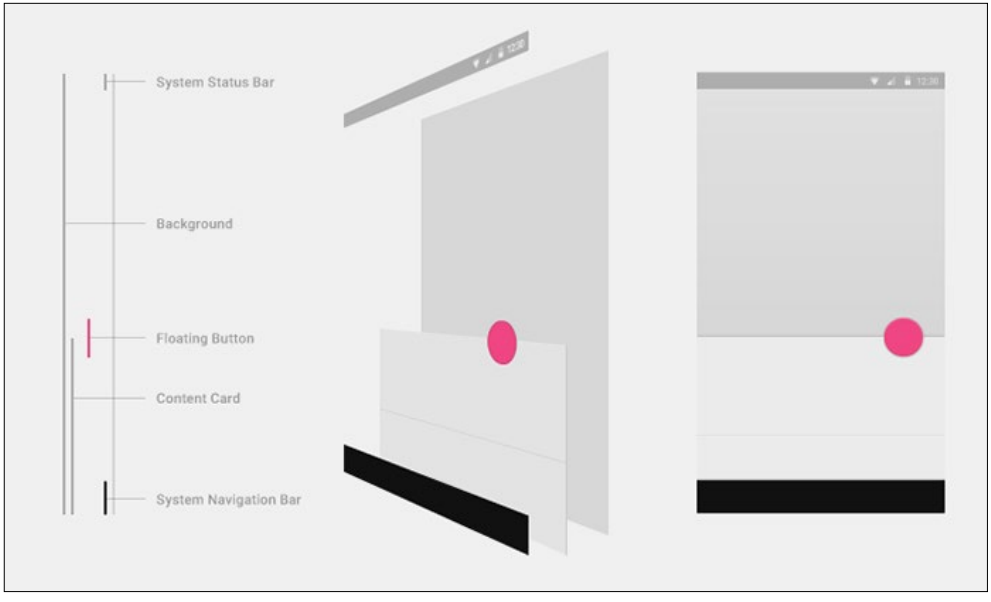
| Verze OS | Verze API |
|---|---|
| Android 1.0 | 1 |
| Android 1.1 | 2 |
| Android 1.5 Cupcake | 3 |
| Android 1.6 Donut | 4 |
| Android 2.0 Eclair | 5 |
| Android 2.0.1 Eclair | 6 |
| Android 2.1 Eclair | 7 |
| Android 2.2–2.2.3 Froyo | 8 |
| Android 2.3–2.3.2 Gingerbread | 9 |
| Android 2.3.3–2.3.7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

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

Nejnovější, v současnosti nejpoužívanější verze Androidu představíme podrobněji.

# Android 5.0 Lollipop

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

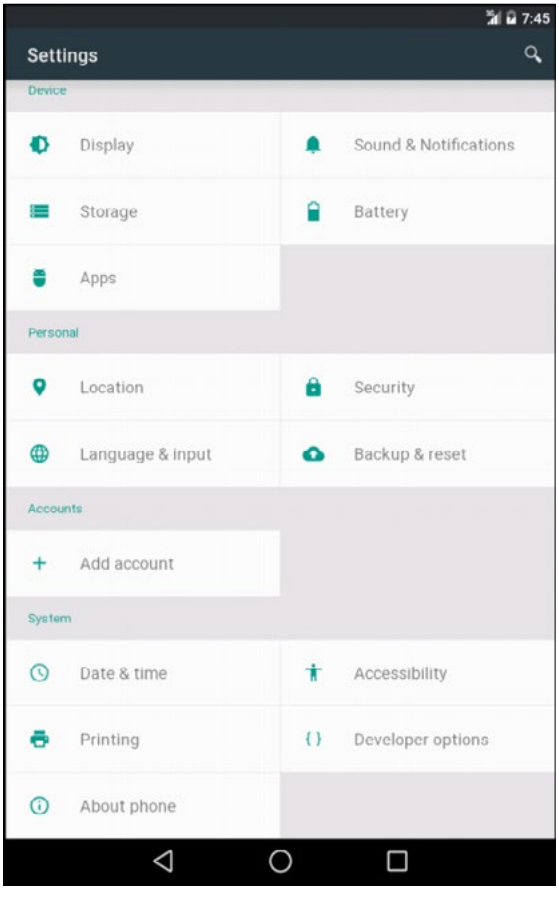



**Obrázek 2.2:** Design nové verze označovaný jako Material Design je přehlednější a pomocí různých vizuálních efektů, hlavně stínu, se snaží vzbudit zdání třetího rozměru

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

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



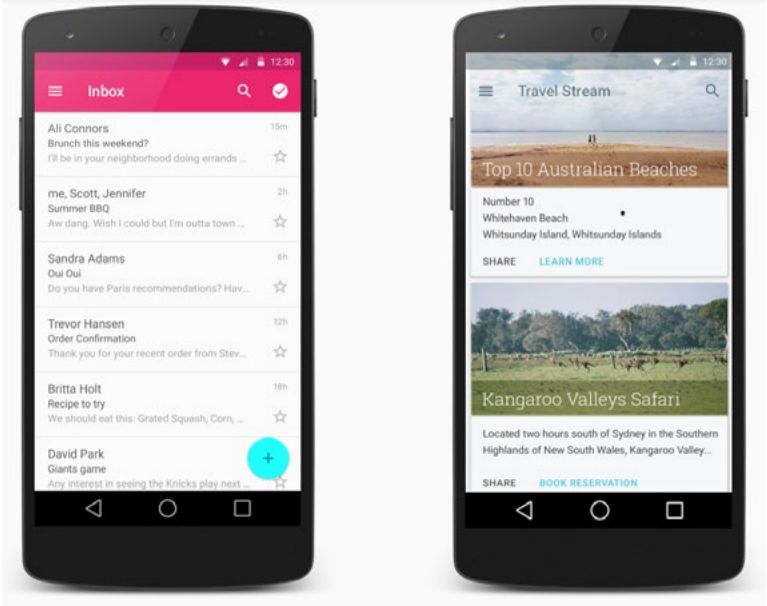



**Obrázek 2.3:** Design aplikace Nastavení je praktickou ukázkou designových doporučení pro aplikace pro Android 5.0

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

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

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




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

# Android 6.0 Marshmallow

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

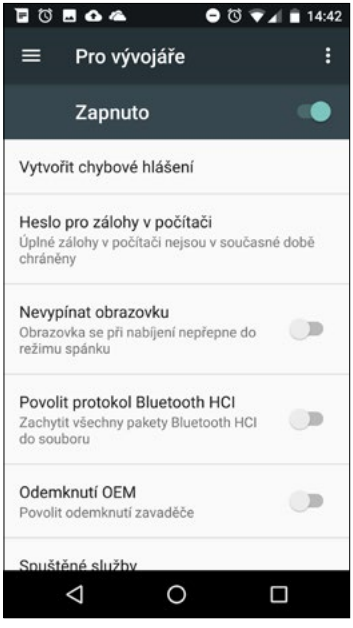
Mírně inovovaný design uživatelského rozhraní, který přináší Marshmallow, charakterizujeme jako barevně živější, dynamičtější a intuitivnější. Nejpoužívanější aplikace jsou soustředěny na hlavní obrazovce. Jejím posunem doprava se zobrazí Google asistent, který kromě vyhledávání poskytuje i další důležité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

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

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

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






**Obrázek 2.5:** Povolení nebo zákaz oprávnění pro konkrétní aplikaci

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

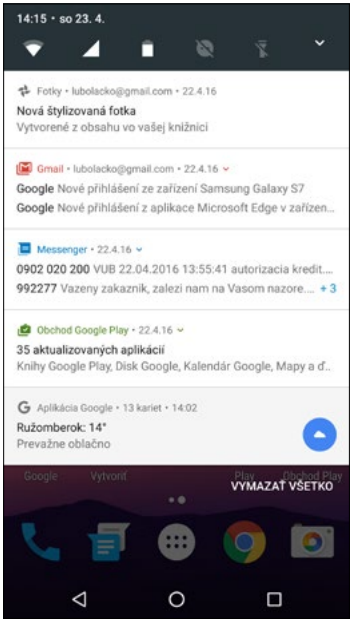
Pokud často používáte více současně spuštěných aplikací a služeb na pozadí, oceníte vylepšený engine ART (Android Runtime), který lépe hospodaří s operační pamětí. Z dalších vylepšení týkajících se hardwaru zmíníme nativní podporu snímání otisků prstů, podporu USB typu C jako nového standardu s oboustranným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

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

# Android 7.0 Nougat

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

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



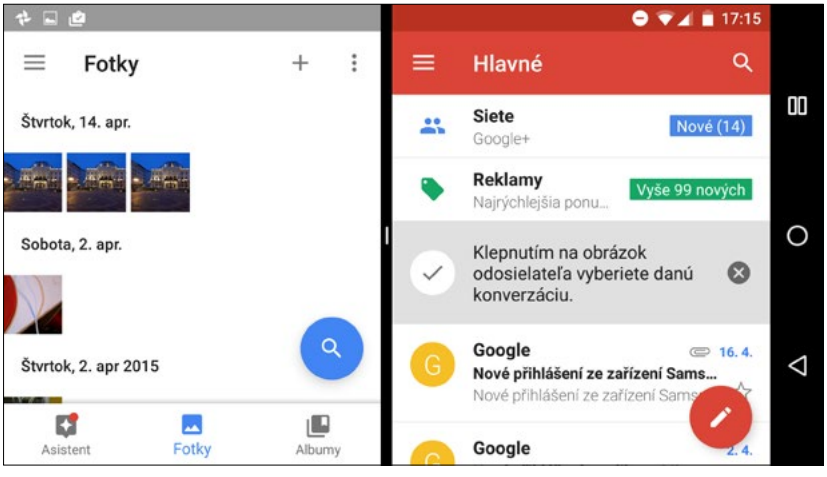

**Obrázek 2.6:** Nový design oznamovacího panelu

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

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

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

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

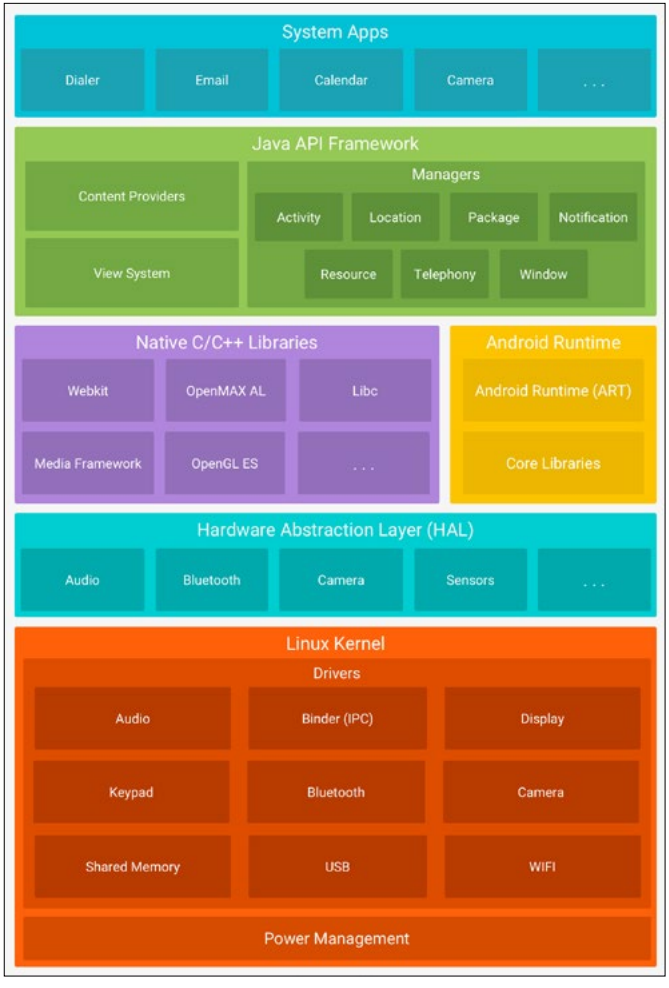



**Obrázek 2.7:** Režim Multi-Window

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

# Stručně o architektuře Androidu

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




**Obrázek 2.8:** Schéma architektury operačního systému Android

## Linux Kernel

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

**POZNÁMKA**

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



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

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

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

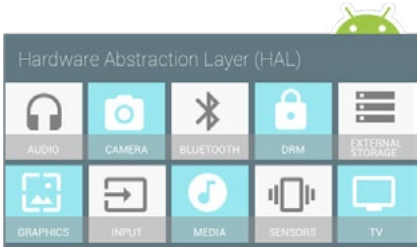
Proto je na platformě Android implementován meziprocesorový ovladač komunikace a volání metod zvaný Binder, který umožňuje více procesům sdílet údaje. Využívá sdílenou paměť, přičemž údaje se odevzdávají jako balíčky. Komunikace procesů je řízena, Binder počítá a mapuje vzájemné reference napříč systémem,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

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

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

## Hardware Abstraction Layer (HAL)

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




**Obrázek 2.9:** Moduly vrstvy HAL

## Nativní knihovny

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

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

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

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

## Android Runtime – ART

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

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

## Java API Framework

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

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

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

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

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

## Aplikace

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

KAPITOLA

3

# Anatomie aplikace

Témata kapitoly:

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

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

## Dříve než začnete vyvíjet – filozofie aplikace

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

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

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

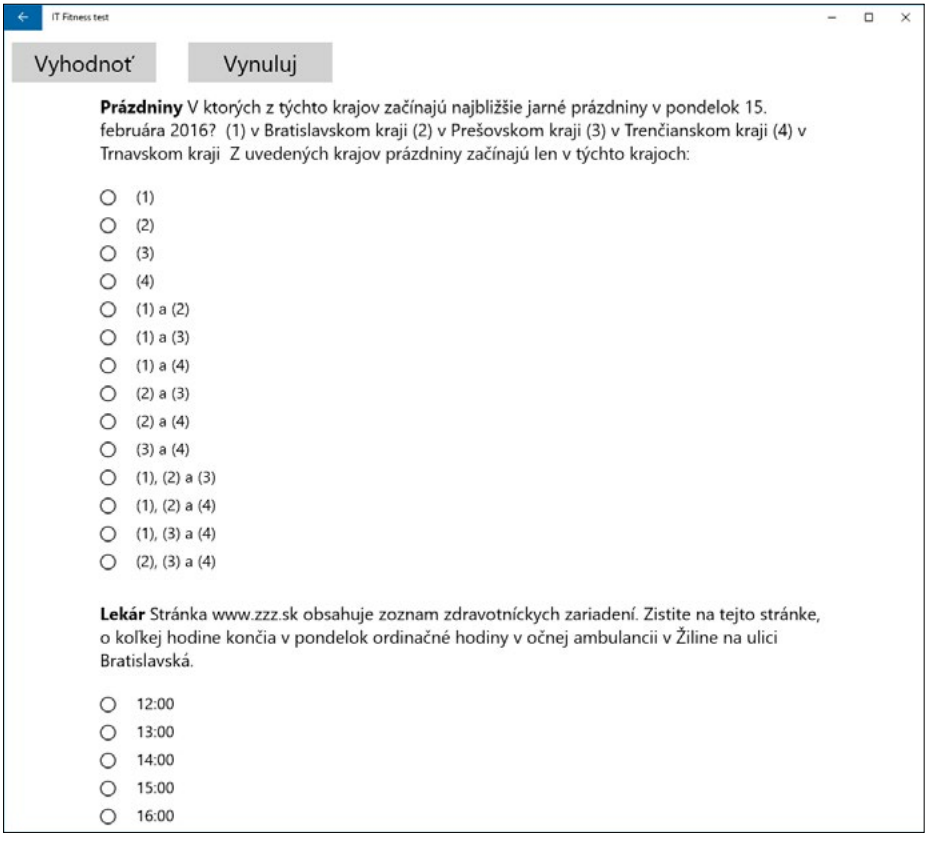
velké části dokumentu. V rozlišení displeje problém není, většina telefonů vyšší střední třídy má displej s rozlišením full HD a vlajkové lodě QHD, tedy 2 560 × 1 440, případně i vyšší. O takovémto rozlišení může řada majitelů notebooků, ale i externích displejů jen snít, oni však mají k dispozici úhlopříčku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

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

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

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

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




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

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

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


**Obrázek 3.2:** Zábavné spoření v bankovní mobilní aplikaci

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

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

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

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


**Obrázek 3.3:** Reklamní podtext populární mobilní aplikace



# Základní součásti aplikace pro Android

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

## Aktivity (Activity)

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

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

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

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

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

### Služby (Services)

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

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

### Broadcast receivers

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

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

### Poskytovatelé obsahu (Content providers)

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

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

## Aktivita a její životní cyklus

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

**POZNÁMKA**

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

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

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

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

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

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

**POZNÁMKA**

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

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

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

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

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

*Aktivita může spouštět jinou aktivitu včetně aktivit jiné aplikace.*

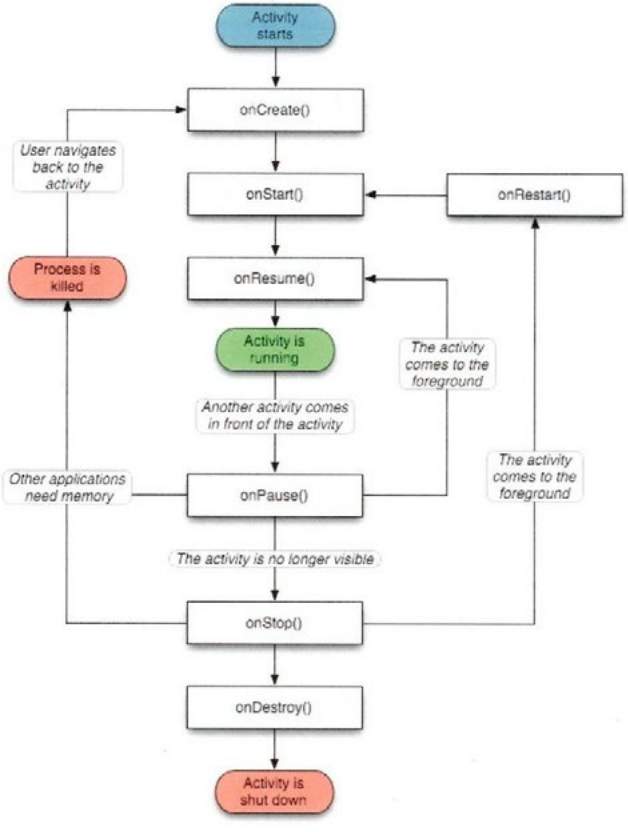
## Životní cyklus aktivity

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

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

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

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




**Obrázek 3.4:** Schéma životního cyklu aktivity

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

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

### onCreate()

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

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

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

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

### onStart() a onResume()

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

### onPause()

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

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

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

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

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

### onStop()

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

**POZNÁMKA**

Metoda `onStop()` nemusí být nikdy volána.

### onDestroy()

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

**POZNÁMKA**

Metoda `onDestroy()` nemusí být nikdy volána.

## Příklad – životní cyklus aktivity

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

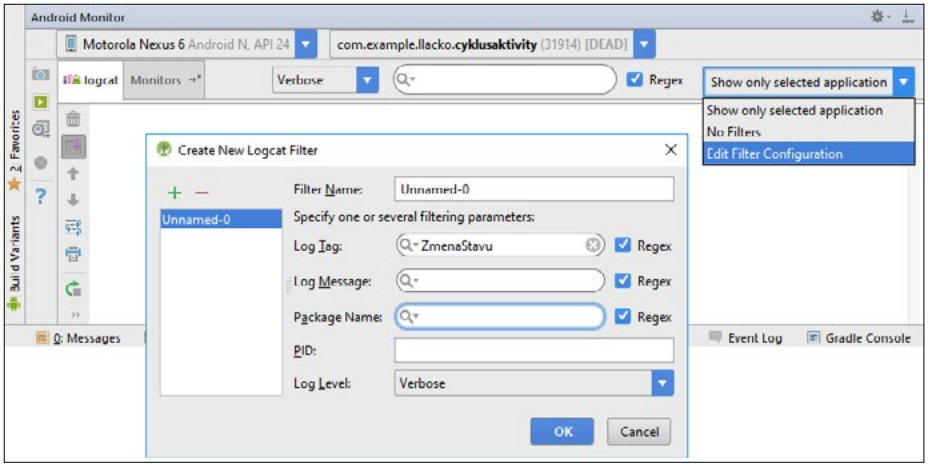
Otevřete soubor *StateChangeActivity.java* s kódem aktivity. Zatím jediná metoda, která se vztahuje k životnímu cyklu, je metoda `OnCreate()`. Jejím úkolem je volat superinstanci metody před nastavením uživatelského rozhraní pro aktivitu. Budeme modifikovat tuto metodu tak, abychom získali ladicí výstup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

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

```
import android.util.Log;

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

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

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

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

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

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

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

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

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

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

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

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

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



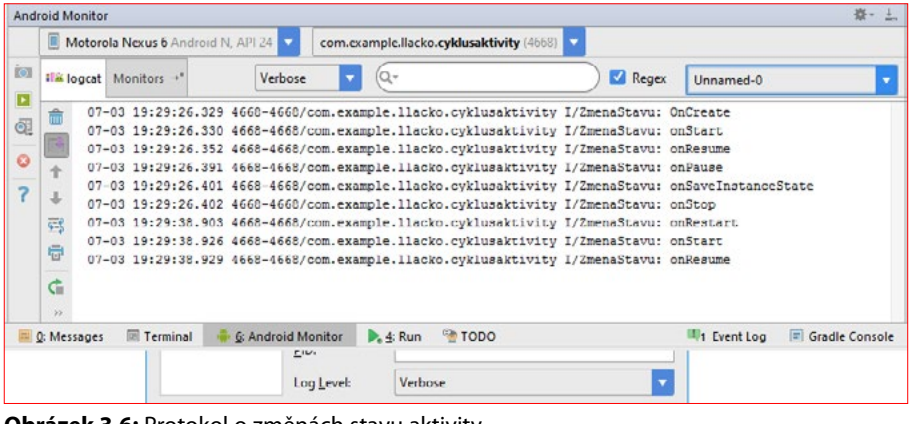

**Obrázek 3.5:** Definování filtru pro logovací výpisy do okna Android Monitor

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

```
onCreate
onStart
onResume
```

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

```
onPause
onSaveInstanceState
onStop
```




**Obrázek 3.6:** Protokol o změnách stavu aktivity

## Příklad – ukládání a obnovení stavu aktivity

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

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

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

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

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

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

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

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

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

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

V metodě `onRestoreInstanceState()` text obnovíte:

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

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

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

**POZNÁMKA**

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

## Intent

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

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

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

**POZNÁMKA**

Intent může spouštět i aktivity jiných aplikací.

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

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

V souvislosti s intenty se nejčastěji používají metody:

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

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

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

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

Způsob spuštění aktivity je definovaný pomocí příznaků:

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

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

Příznaky se nastavují metodou:

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

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

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

**POZNÁMKA**

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



## Odevzdávání údajů a výsledků

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

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

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

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

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

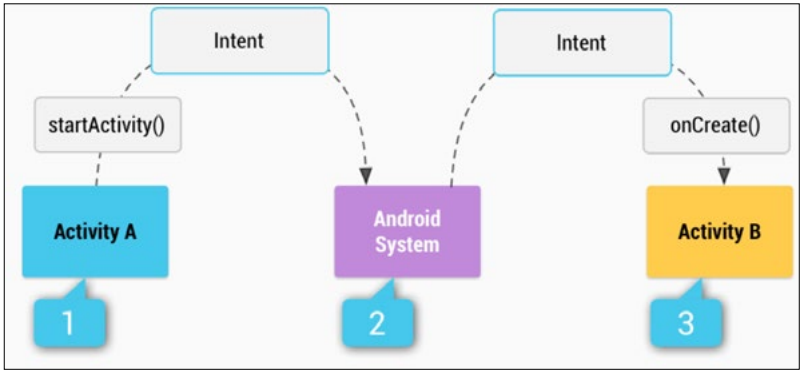
## Intent Filter

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

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

Například:

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




**Obrázek 3.7:** Princip fungování záměru

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

Základní rozdělení záměrů vychází z jejich směrování:

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

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

Třída `Intent` má několik důležitých atributů.

**Action**

Deklaruje typ požadované akce. Například:

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

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



**Data**

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

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

Nebo pro vytočení telefonního čísla:

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

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

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

**Category**

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

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

**Type**

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

**Component**

Konkrétní objekt, který by měl provést požadovanou operaci.

**Extras**

Umožňuje uložit další informace související se záměrem.

**Flags**

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

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

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

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

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

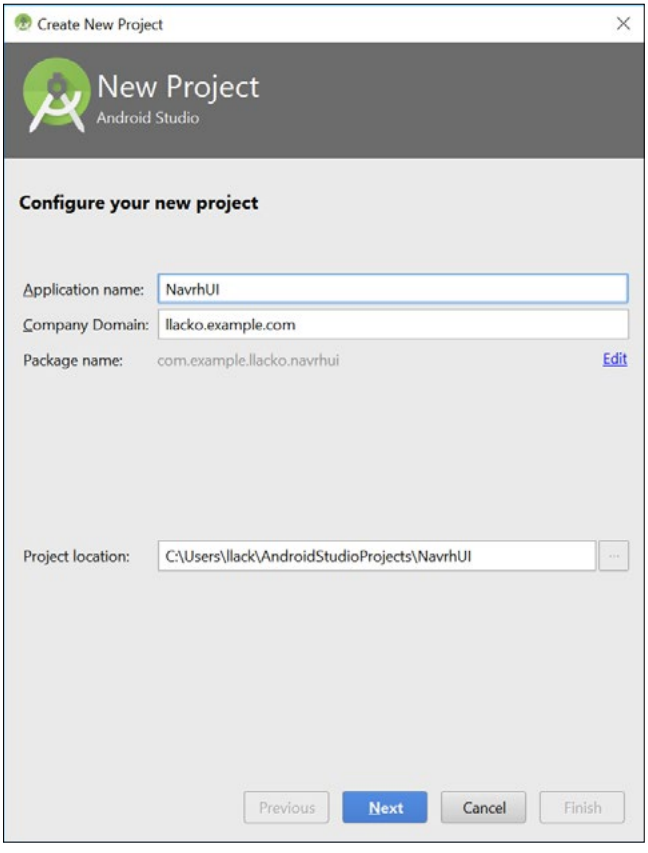
# Příklad – Anatomie projektu

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

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

## Vytvoření projektu

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




**Obrázek 3.8:** Vytvoření nového projektu – zadání názvu

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