

Uvod u Android

Sadržaj predavanja

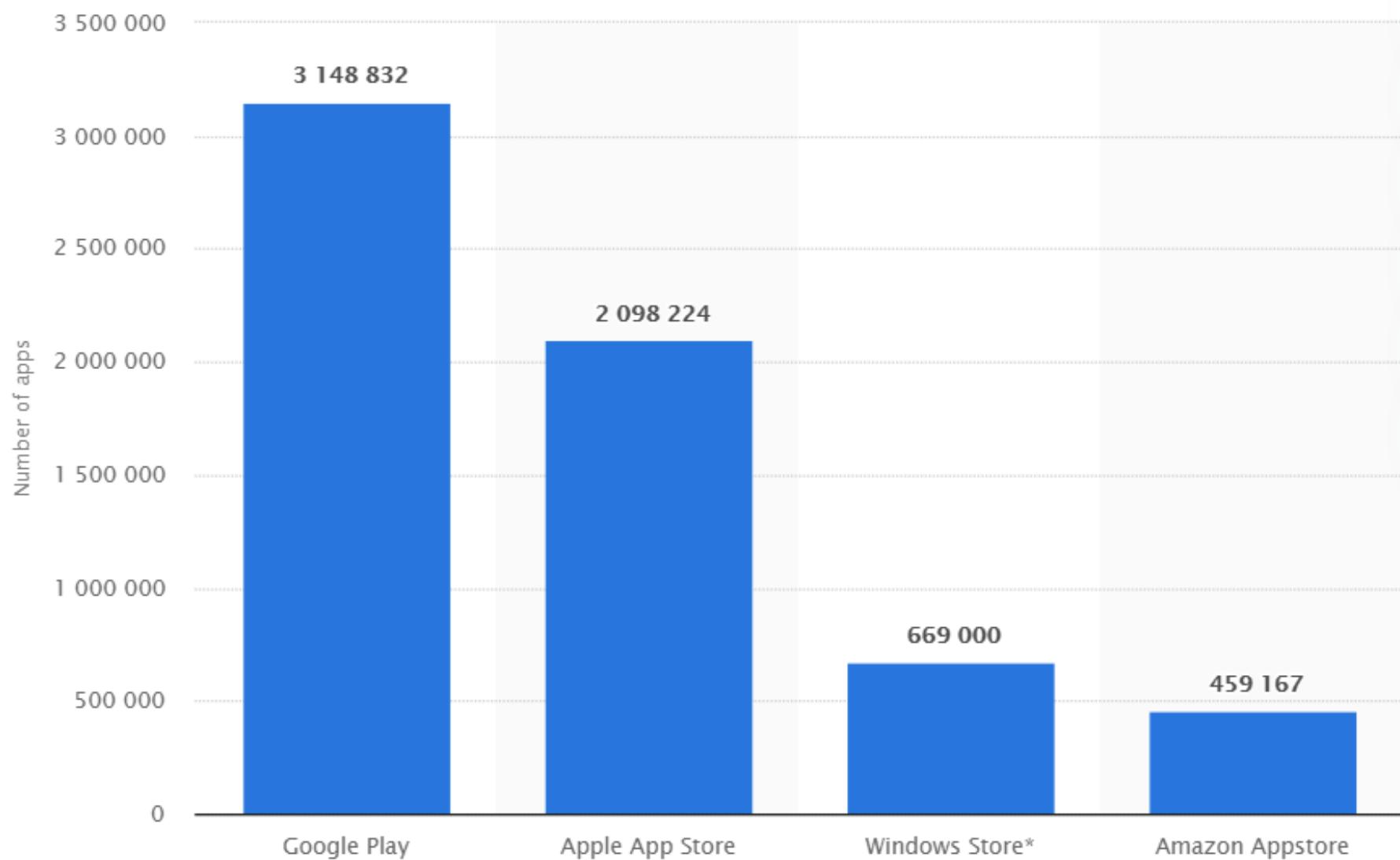
- Nepoznanice o Androidu
 - Razvoj i karakteristike OS-a
- Arhitektura Android sustava
 - Virtualni strojevi ART i Dalvik
- Razvoj aplikacija
 - Potrebni alati

Što je Android?

- Android – programsko okruženje za pokretne uređaje bazirano na otvorenom kodu
- Početak razvoja:
 - studeni 2007. godine
 - Google i OHA (engl. Open Handset Alliance)
- Karakteristike sustava:
 - otvorena platforma
 - automatsko upravljanje životnim ciklusom aplikacije
 - brz i jednostavan razvoj aplikacija
 - kompatibilnost s većinom sadašnjeg i budućeg hardvera
 - široki spektar primjene:
 - mobilni uređaji, televizori, pametni satovi, automobili...



Zašto je Android popularan?



Android verzije

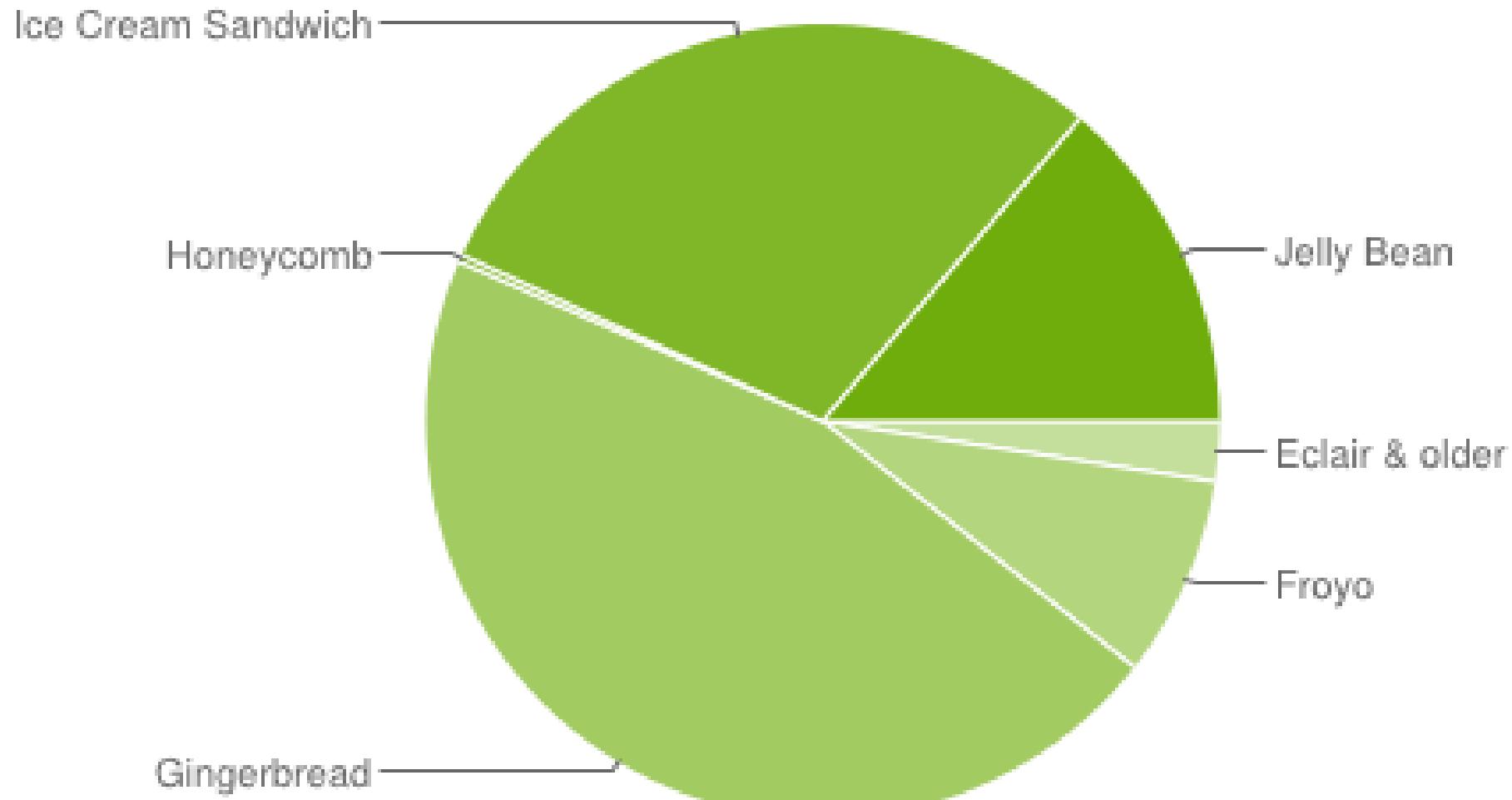
Cupcake
1.5Donut
1.6Eclair
2.0/2.1Froyo
2.2Gingerbread
2.3Honeycomb
3.0/3.1Ice Cream Sandwich
4.0Jelly Bean
4.1/4.2/4.3KitKat
4.4Lollipop
5.0Marshmallow
6.0Nougat
7.0Oreo
8.0Pie
9.0

10

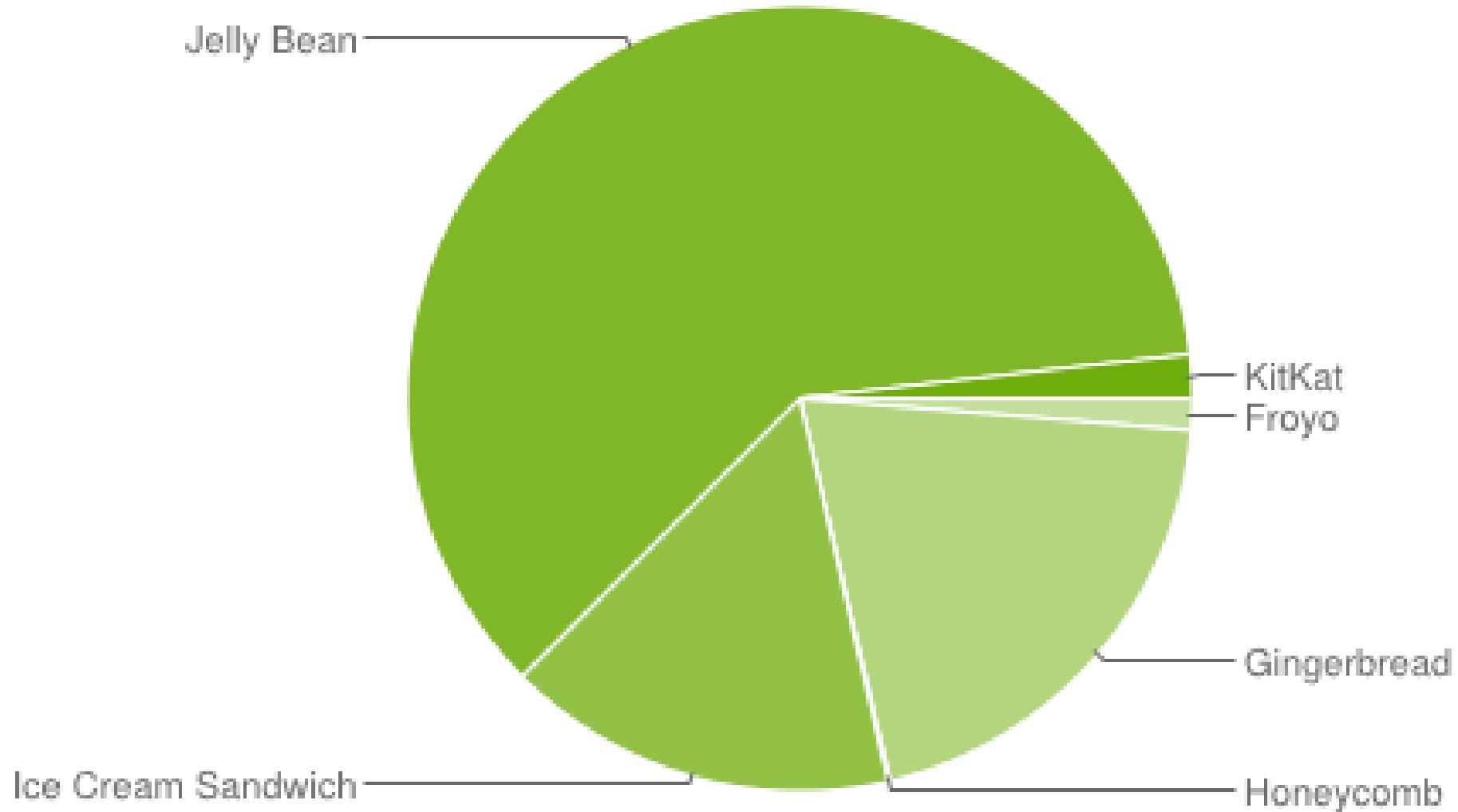


android

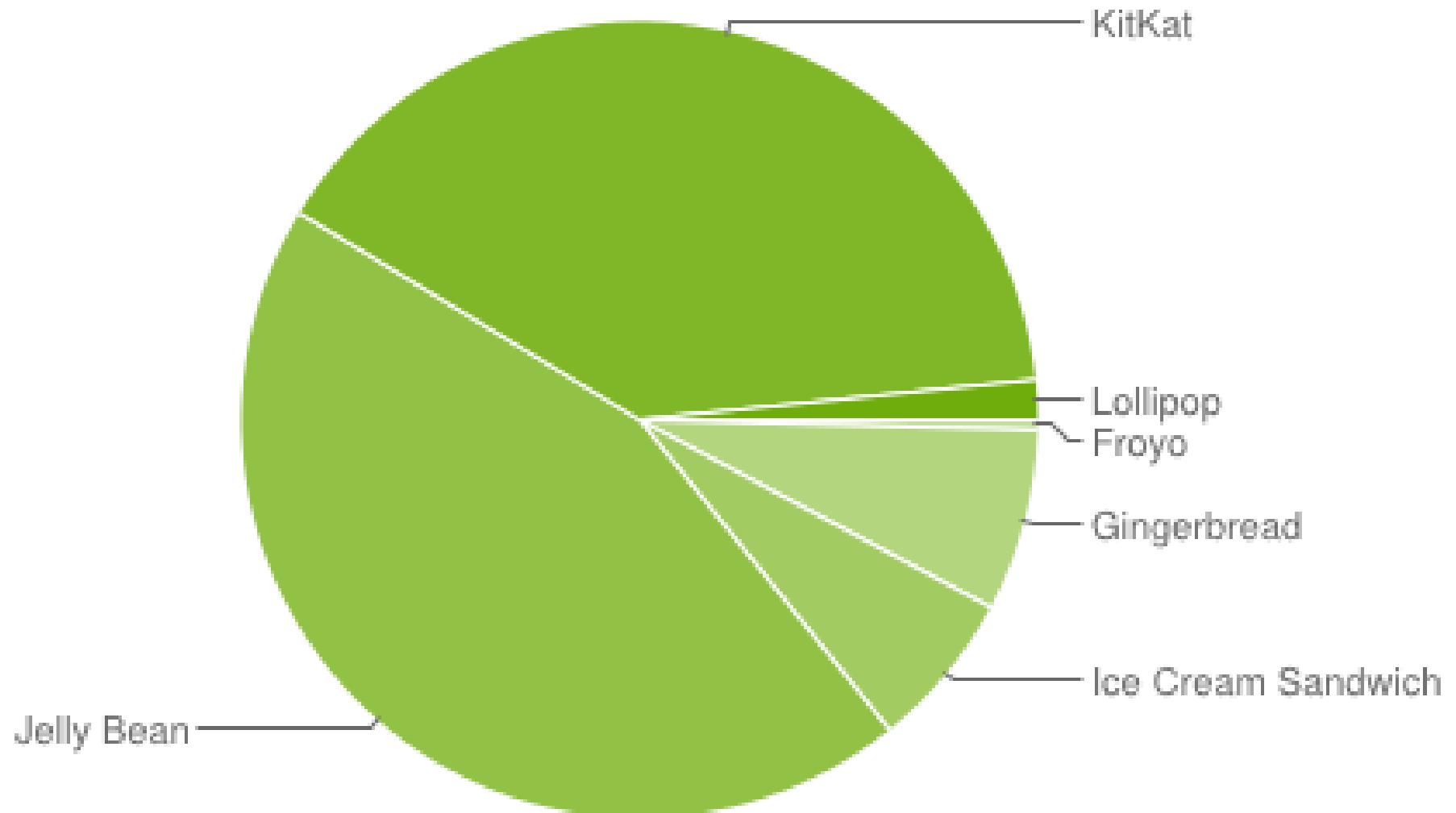
U drugom mjesecu 2012:



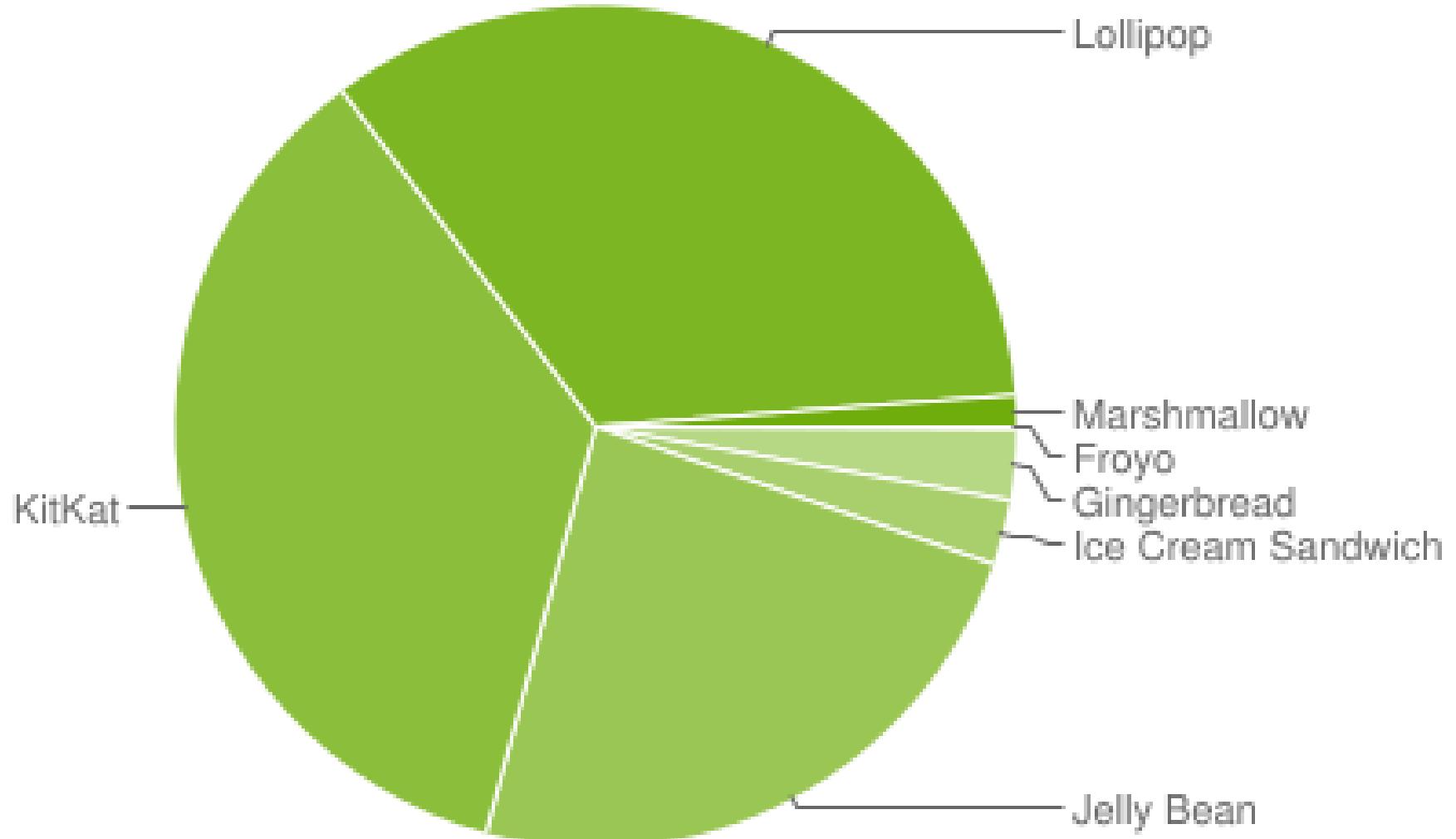
U drugom mjesecu 2013:



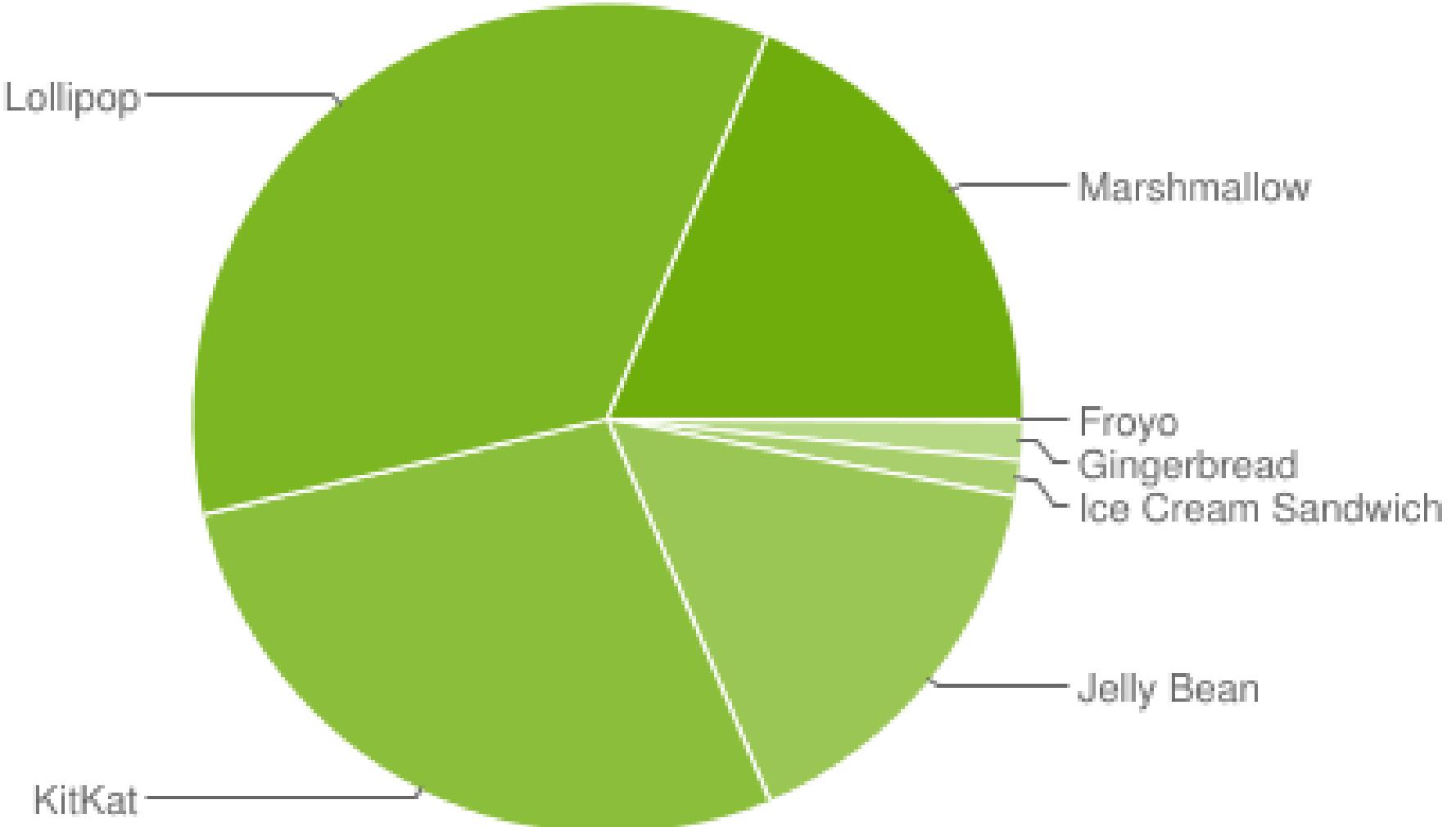
U drugom mjesecu 2014:



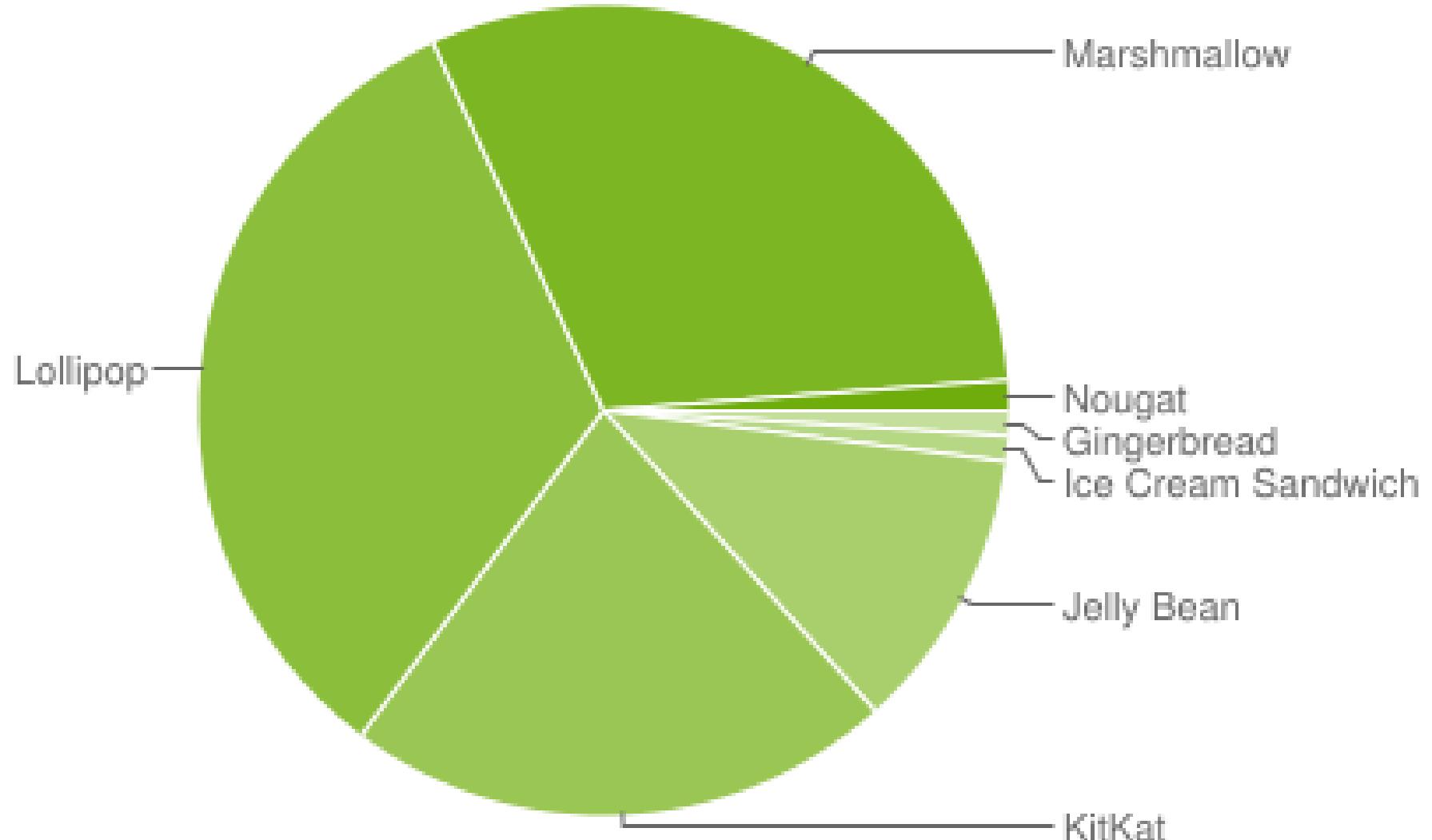
U drugom mjesecu 2015:



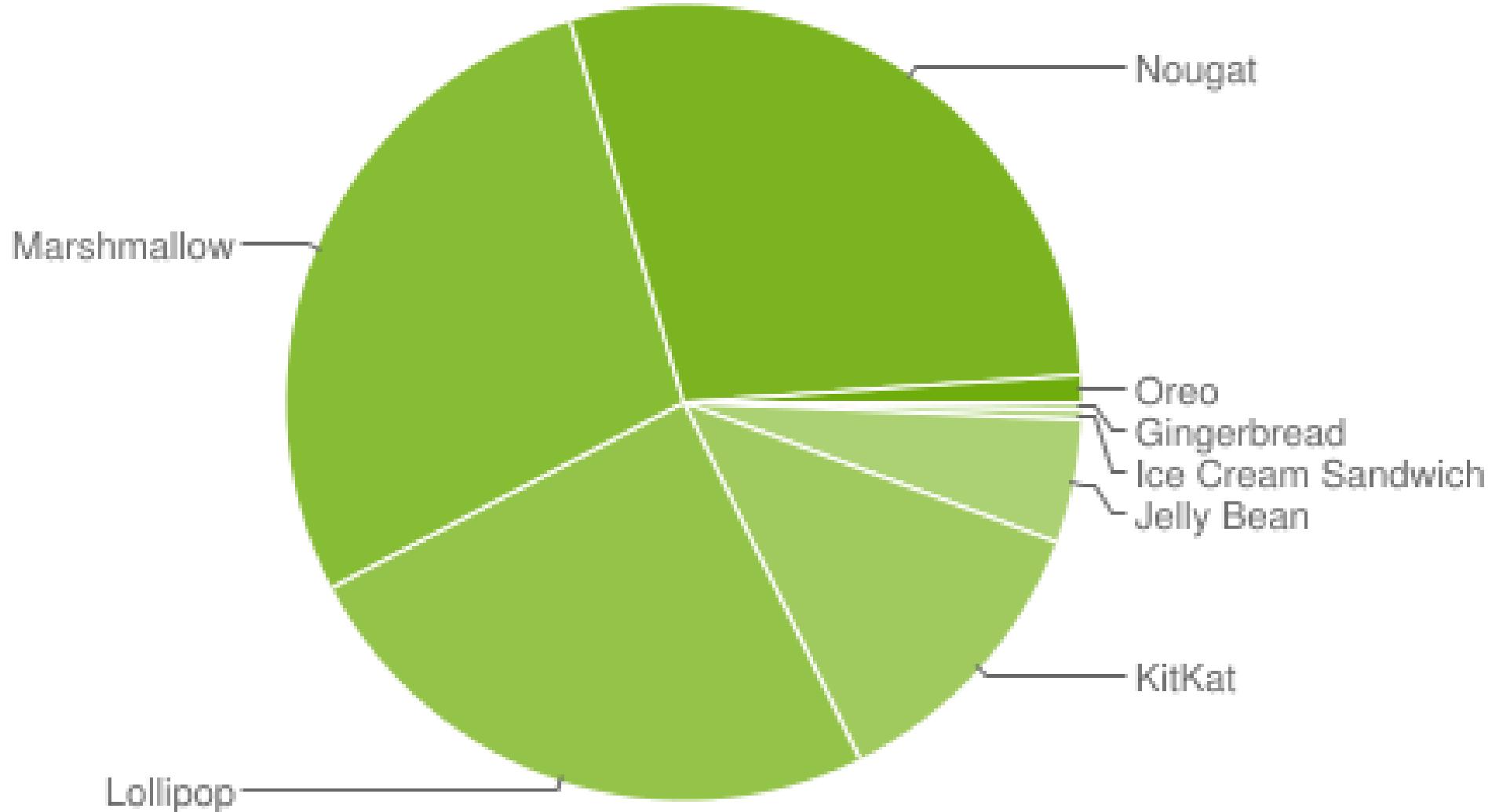
U drugom mjesecu 2016:



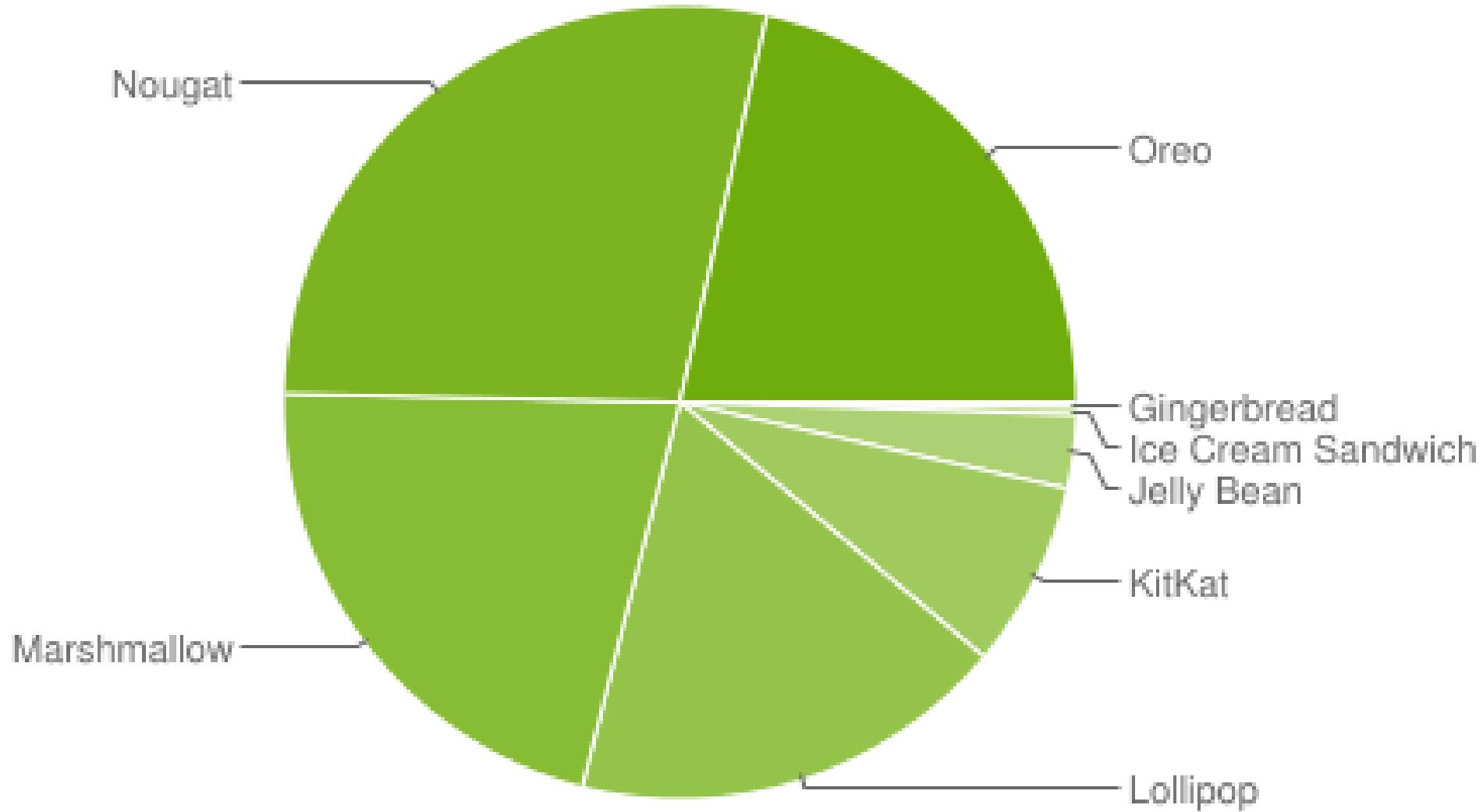
U drugom mjesecu 2017:



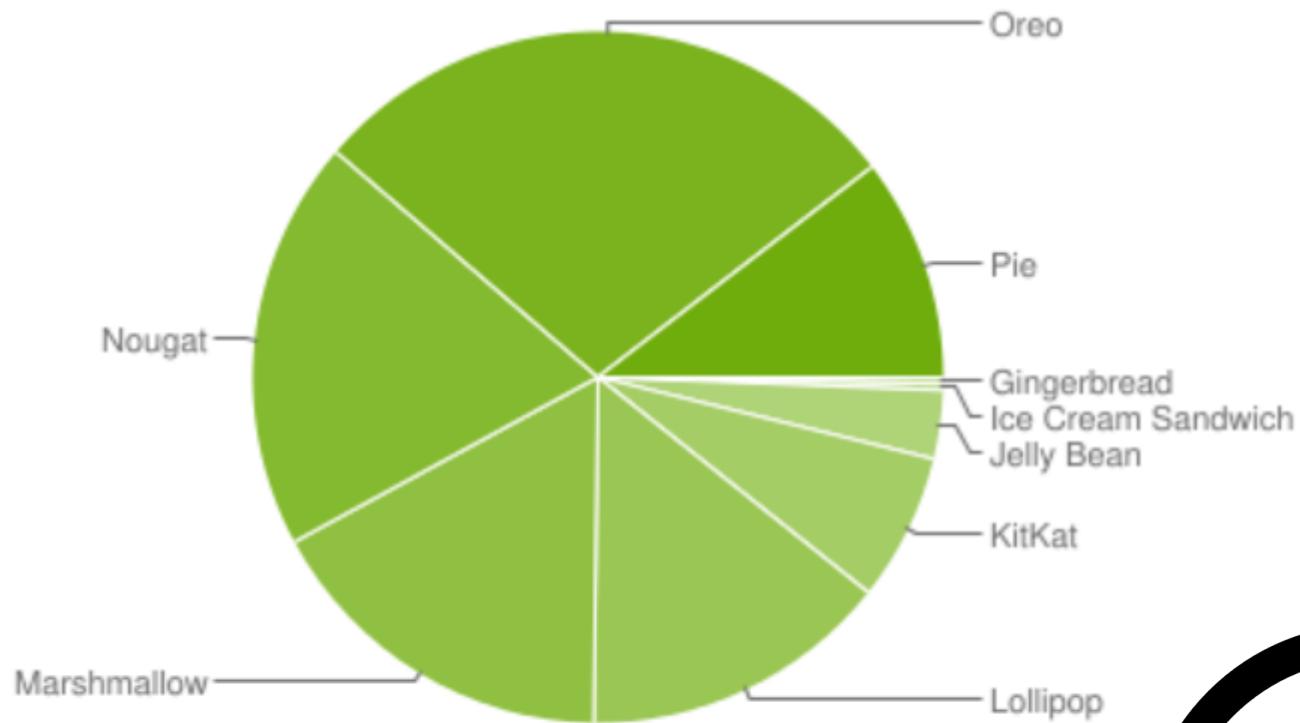
U drugom mjesecu 2018:



U drugom mjesecu 2019:



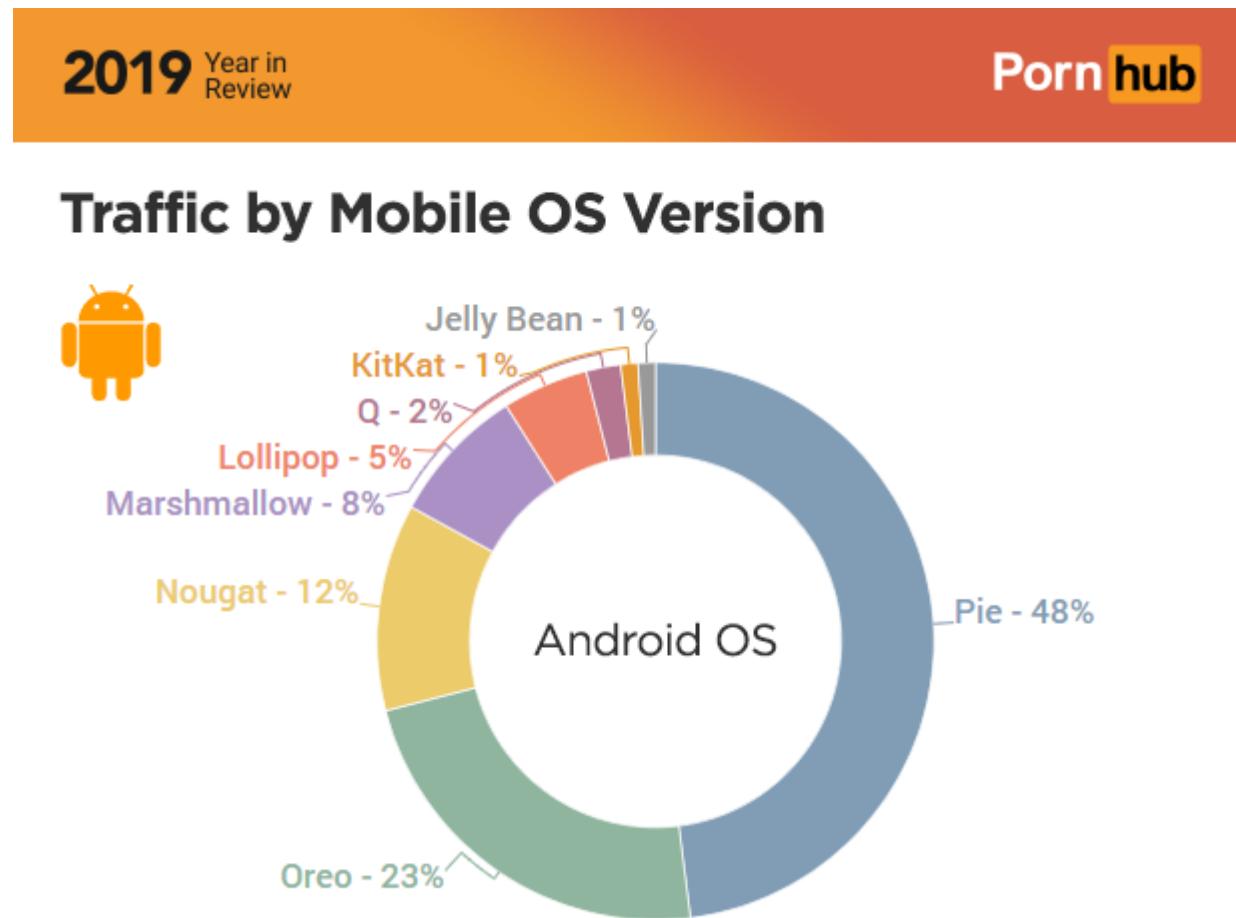
U drugom mjesecu 2020:



*Data collected during a 7-day period ending on May 7, 2019.
Any versions with less than 0.1% distribution are not shown.*

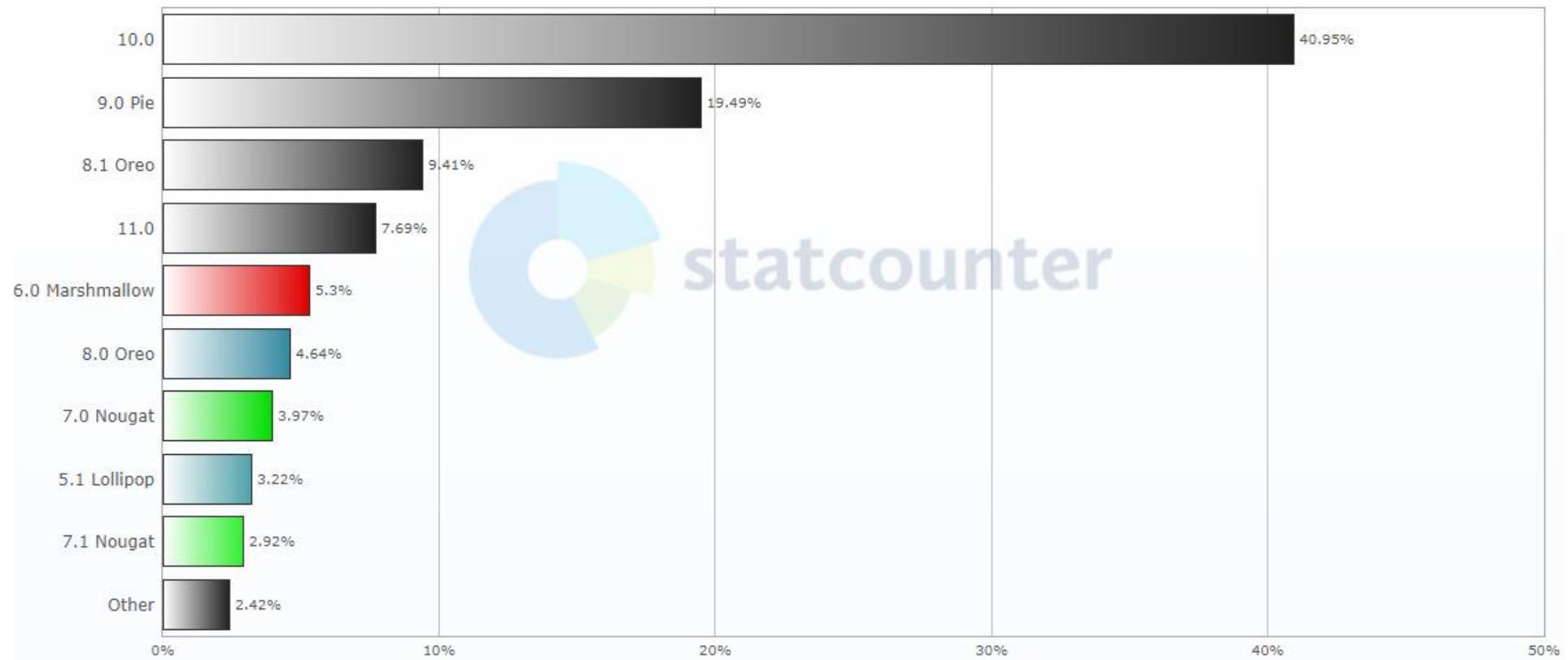


Third party statistika i dalje postoji!



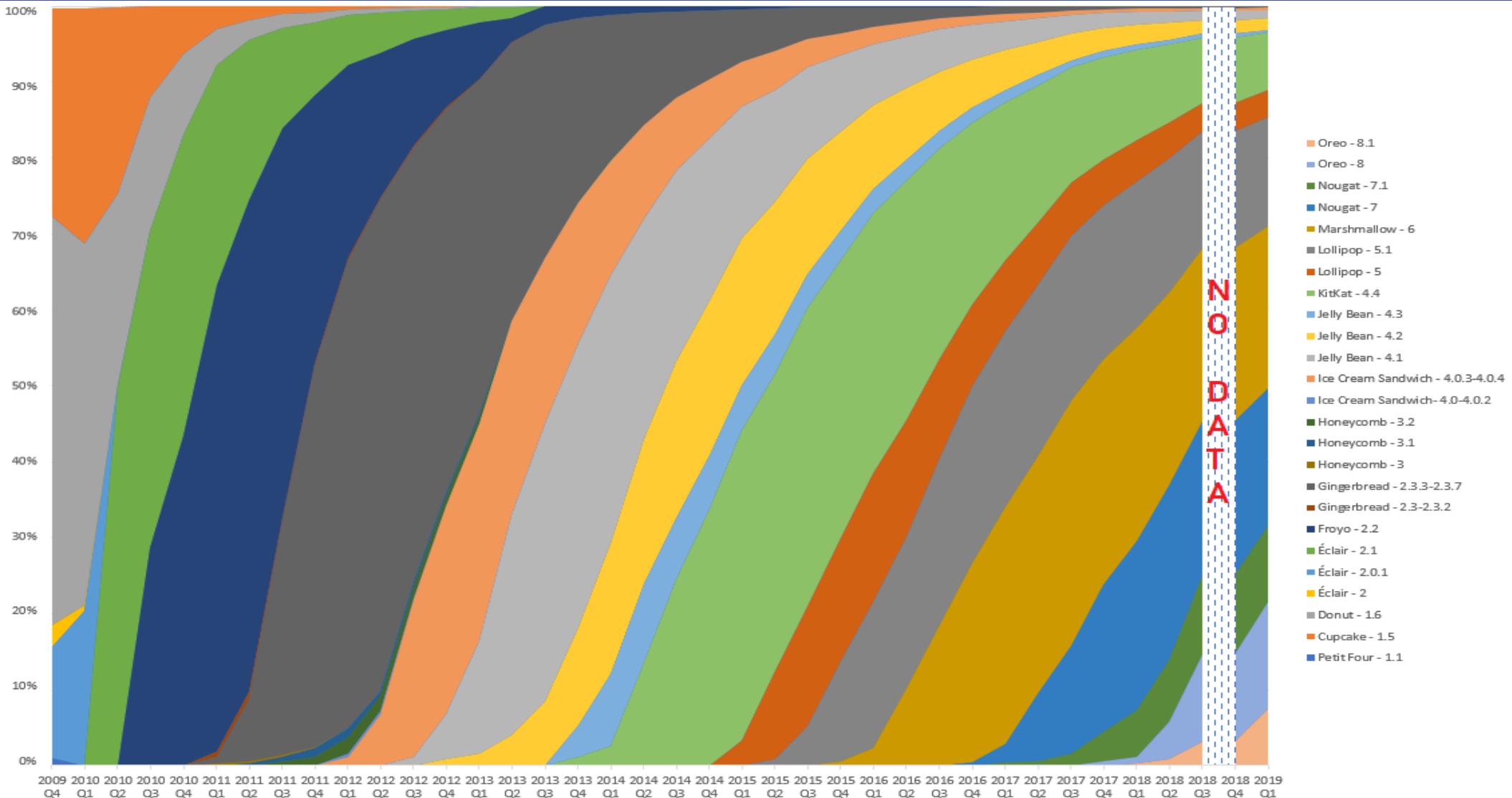
Više na: <https://www.androidpolice.com/2019/12/18/pornhub-does-what-google-wont-releases-android-version-stats-for-2019/>

U drugom mjesecu 2021:



Izvor: <https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide/#monthly-202103-202103-bar>

Cijela povijest:



Arhitektura Androida

Aplikacijski sloj

Izvorne sistemske aplikacije

Aplikacije neovisnih proizvođača

Aplikacije neovisnih razvojnih inženjera

Sloj aplikacijskog okvira

Upravljanje aktivnostima

Upravljanje prozorima

Pružatelji sadržaja

Sustav grafičkog prikaza

Upravljanje obavijestima

Upravljanje programskim paketima

Upravljanje pozivima

Upravljanje resursima

Upravljanje lokacijski baziranim uslugama

Sensor Manager

Biblioteke

Surface Manager

Media Framework

SQLite

OpenGL | ES

FreeType

WebKit

SGL

SSL

libc

Android okruženje

Jezgrene biblioteke

Virtualni stroj

Linux jezgra

Pogonski program za prikaz

Pogonski program za kameru

Pogonski program za Bluetooth

Pogonski program za flash memoriju

Pogonski program za meduprocesnu komunikaciju

Pogonski program za tipkovnicu

Pogonski program za WiFi

Pogonski program za USB

Pogonski programi za zvuk

Upravljanje napajanjem

Korisničke aplikacije pisane u programskom jeziku Java i Kotlin

Sloj aplikacijskog okvira pisan u programskom jeziku Java

Android okruženje – Dalvik Virtual Machine/ART

Izvorne programske biblioteke pisane u jezicima C i C++

Linux jezgra 4.9 / 4.14 / 4.19 / 5.4

Virtualne mašine

Najčešće

- Java – JVM – Java virtual machine (Stack bazirana)
- .NET – CLR – Common language runtime (Stack bazirana)
- Android – Dalvik i ART (Registarski bazirane)

Implementiraju

- Kompajliranje izvornog koda u neki od „bytecodova”
- Strukture podataka koje sadrže instrukcije i operande
- Call stack za pozive funkcija
- „Instruction pointer” koji pokazuje na sljedeću instrukciju koja se mora izvršiti
- Virtualni CPU
 - Dohvaća sljedeću instrukciju
 - Dekodira operande
 - Izvršava instrukciju

Battle of virtual machines

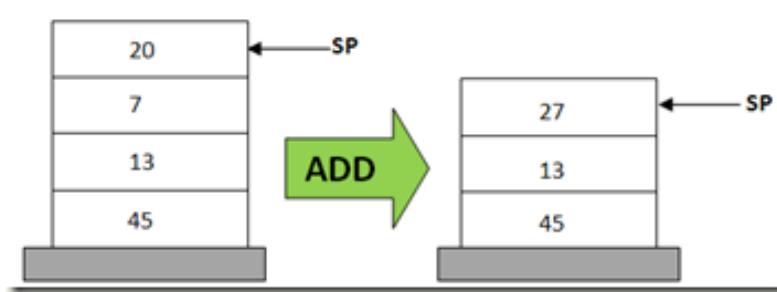
- Registr je najbrža memorija u procesoru
 - Mašina ima manje upita prema sporijem RAM-u
- Stack je memorija izvan procesora (sporija je) do koje se dolazi pop i push metodama
- Stack virtualnu mašinu je puno jednostavnije napisati i manje je podložna pogreškama

Battle of virtual machines

Stack bazirane

- Memorjska struktura u kojoj se nalaze operandi je stack
- Operacije se izvode na način da se podaci pop naredbom vade iz memorije, procesuiraju i nakon toga push naredbom stavlju nazad u LIFO
- Zbog push i pop naredbi, 4 linije instrukcija je potrebno napraviti kako bi se izvršila operacija zbrajanja

POP 20
POP 7
ADD 20, 7, result
PUSH result



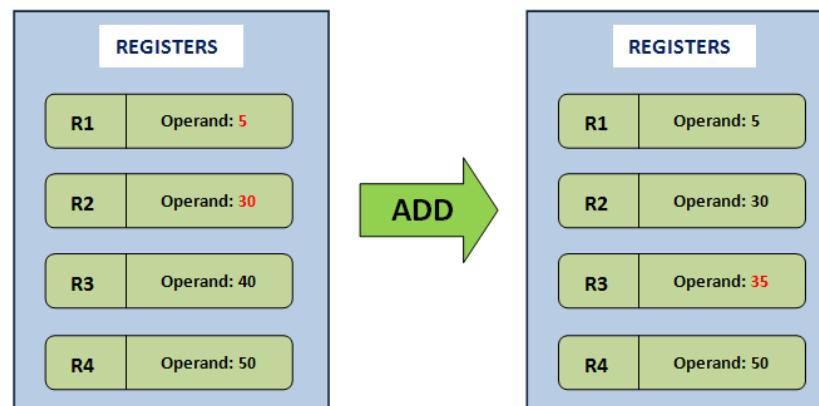
- Prednosti Stack bazirane virtualne mašine:
 - Operandi su adresirani implicitno od strane stack pointera
 - Virtualna mašina ne mora eksplisitno znati adresu operanda
 - Sve aritmetičke i logičke operacije se obavljaju „pushanjem” i „popanjem” operanada i rezultata na stack

Battle of virtual machines

Registarski bazirane

- Memorijska struktura u kojoj se nalaze operandi su registri na procesoru
- Nema push-a i pop-a, već instrukcije sadrže adrese registara operanada
- Zbog navedenog, zbrajanje se izvršava pomoću jedne naredbe

ADD R1, R2, R3

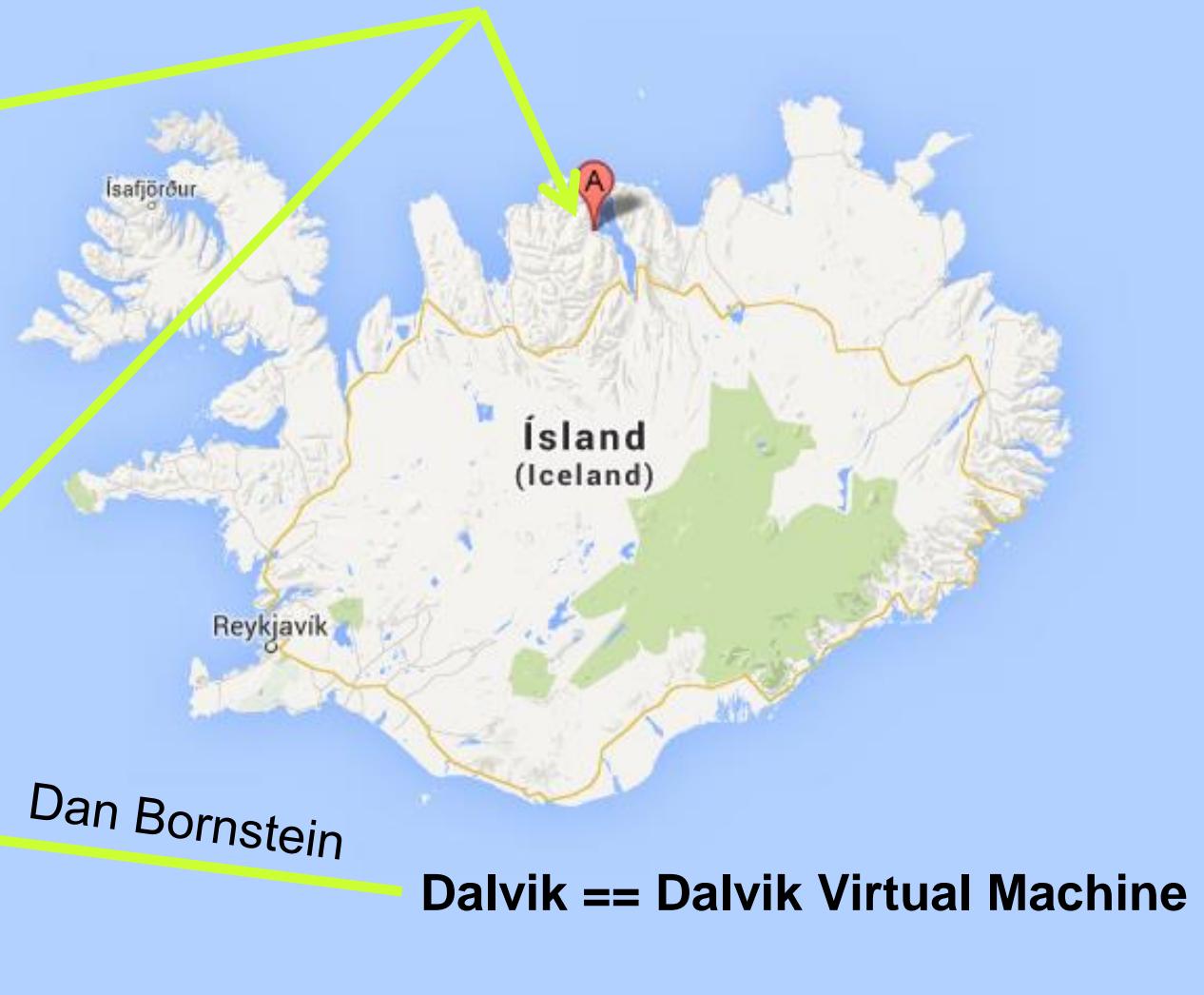


- Za razliku od stack VM-a, potrebno je eksplisitno odrediti adrese operanda (u ovom primjeru R1, R2 i R3)
- Prednosti Registarski bazirane virtualne mašine:
 - Nema nepotrebnog „pushanja“ i „popanja“
 - Mogućnosti dodatne optimizacije (npr. često korištene podizraze je moguće pohraniti u registar nakon prvog izračuna i ponovo koristiti)
- Mane se svode na veličinu instrukcije (potrebno je navesti eksplisitno operande)

Dalvik VM



Dalvik == grad na Islandu



Dan Bornstein

Dalvik == Dalvik Virtual Machine

Dalvik i ART VM

- Dalvik i ART su virtualne mašine
- Vrte većinu aplikacija na Androidu (Osim ako nije native)
- Mogućih ih je usporediti s JVM-om (Java virtualna mašina)
- Dalvik i ART su optimizirani od JVM-a

Dalvik i ART VM

Dalvik i ART su napravljeni da rade

- Na polaganom procesoru
- S malo RAM memorije
- Na OS-u bez swap memorije
- A sve to napaja baterija

Još o Dalvik-u i ART-u

Registarski bazirani

- (JVM je stack baziran)

Izvršavaju .dex datoteke (dalvik executable), a ne .class kao JVM

- Bolja prilagodba za low power procesore

Podržava višestruko instanciranje za razliku od JVM-a

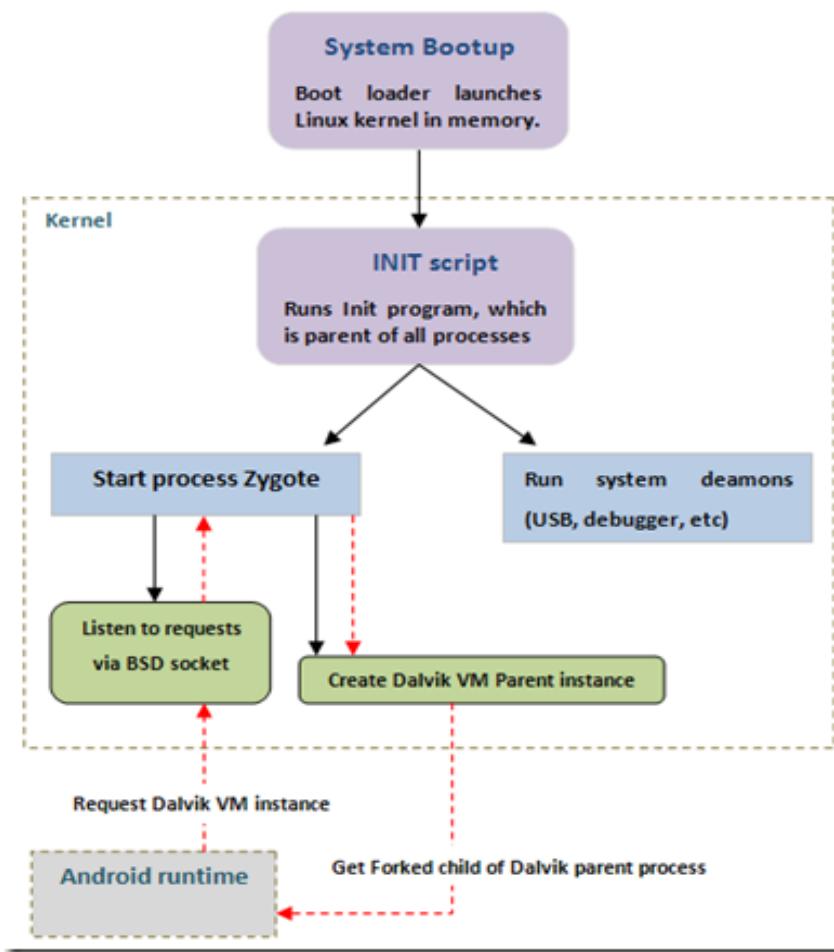
- To znači da se svaki program vrti u zasebnom procesu
- Moguće je „ubiti“ jedan proces bez utjecaja na ostale
 - (za razliku od JVM-a)

Under the hood

Ako želimo saznati kako se Dalvik i ART instanciraju, moramo krenuti od početka...
Na boot-anje Androida:

1. Boot loader učita kernel u memoriju
2. Kernel pokreće init program koji je „roditelj“ svim ostalim procesima
3. Zygote proces kreira instancu Dalvika ili ART-a koja će biti „roditelj“ svim ostaliminstancama Dalvika ili ART-a
4. Zygote također namješta BSD read socket i sluša dolazeće zahtjeve
5. Kada dođe novi zahtjev za Dalvik ili ART VM instancom, Zygote raščlaniti roditeljski proces i šalje ga aplikaciji djetetu na raspolaganje

Zygote – Inicijalna stanica novo kreiranog organizma (Grčki: ζυγωτός *zygōtos*, „spojen,“)

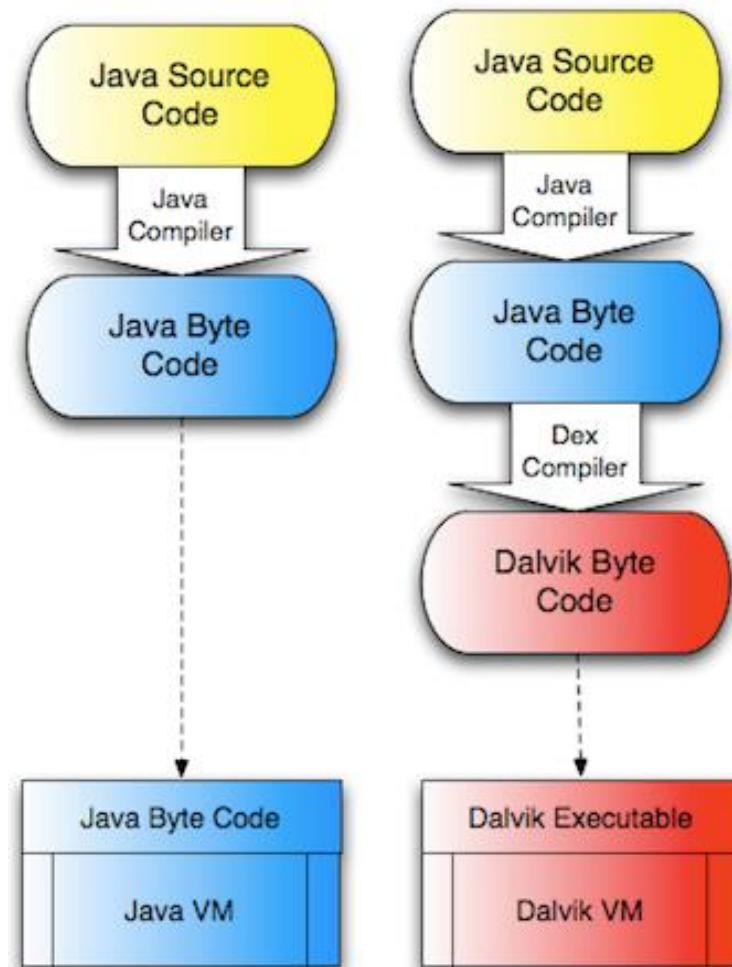
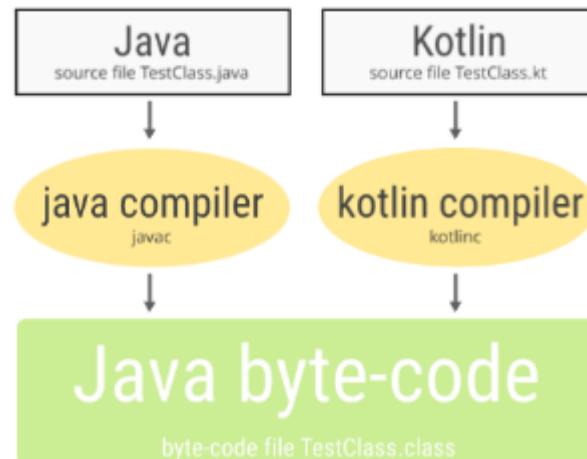


Više o BSD socketima: http://en.wikipedia.org/wiki/Berkeley_sockets

Under the hood

Zato što Dalvik i ART koriste Dalvik bytecode, za isti je kreiran poseban kompjajler koji pretvara Java byte code u Dalvik byte code

Taj proces obavlja DEX compiler koji .class datoteke pretvara u .dex datoteke koje su optimizirane za Dalvik ili ART VM



ART vs Dalvik

ART posjeduje ahead-of-time kompajliranje, dok Dalvik koristi just-in-time kompajliranje

To zapravo znači da ART kompajlira aplikaciju prilikom instalacije, dok je Dalvik kompajlira prilikom izvođenja

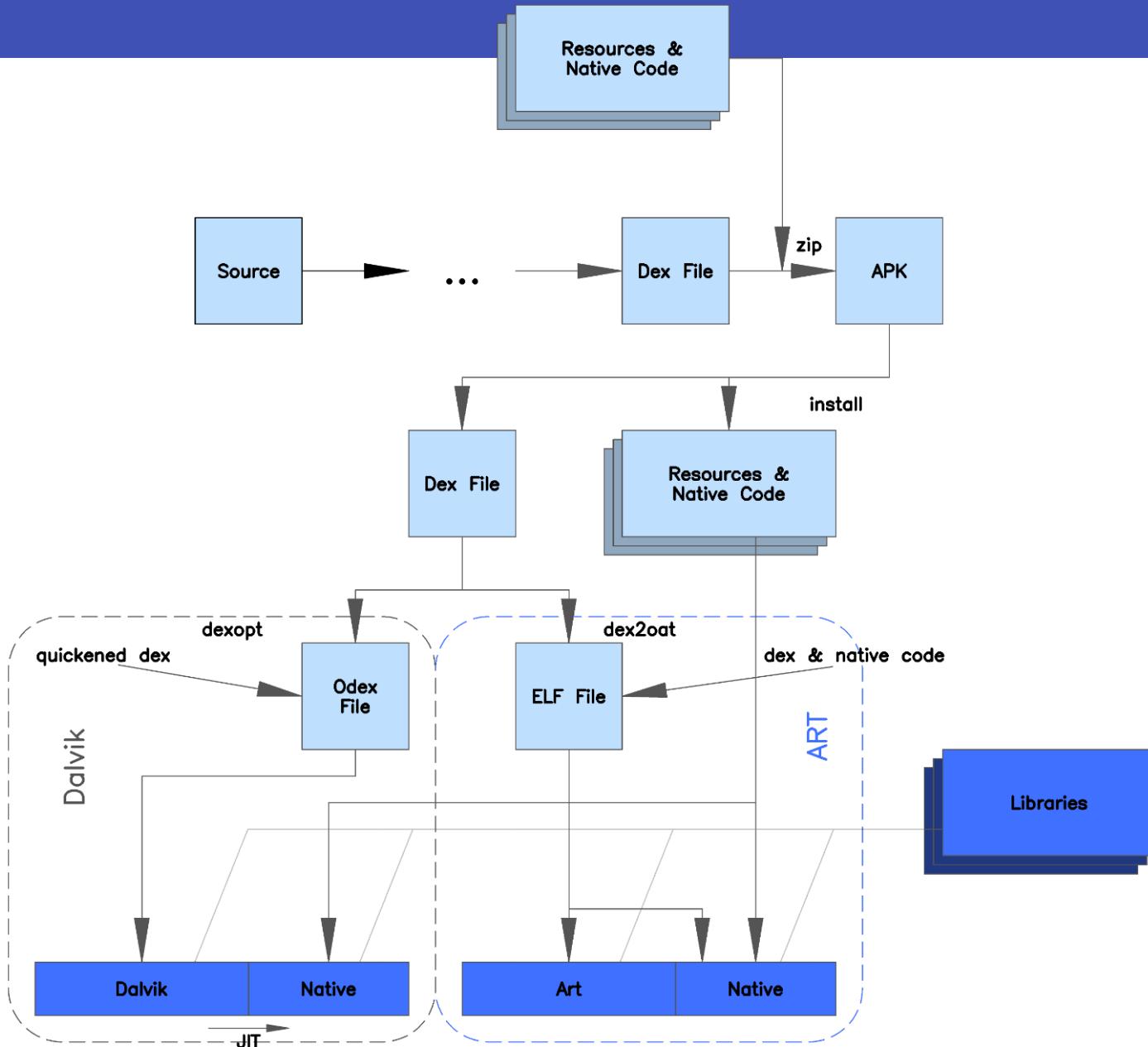
- Dulje trajanje baterije zbog manje kompajliranja
- Bolje performanse tijekom izvođenja aplikacije
- Sporija instalacija

Oba VM-a koriste dex datoteke

Odex datoteke (optimizacijske datoteke Dalvika) zamijenjene su ELF datotekama (Executable and Linkable Format)

- ART pokreće samo ELF datoteke nakon kompajliranja

ART vs Dalvik



Razvoj aplikacija

Android studio

- Baziran na IntelliJ Idea razvojnom okruženju

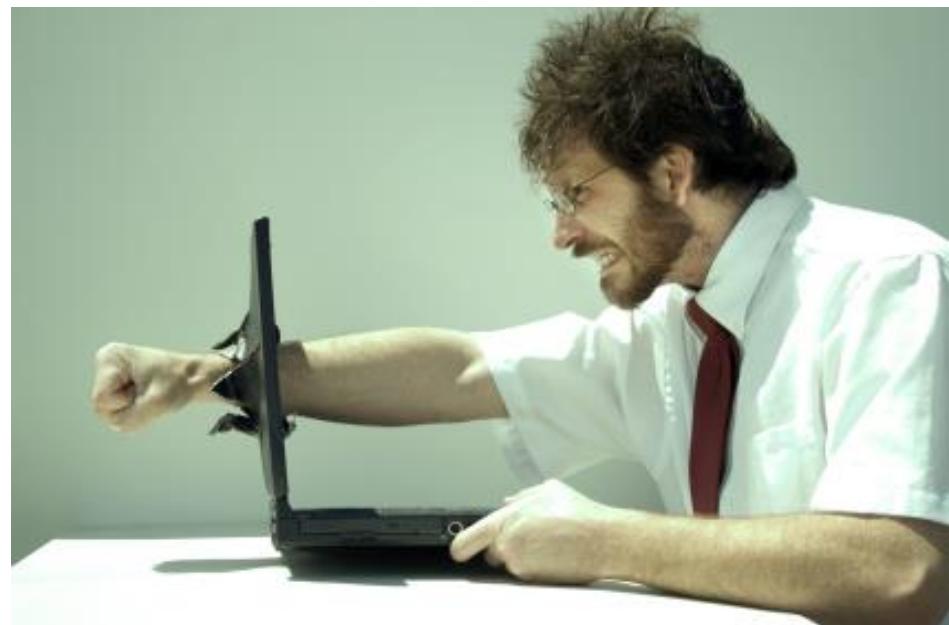
Upute i download:

<https://developer.android.com/studio/index.html>

Android emulator

AVD – Android Virtual Device

- služi emulaciji stvarnog uređaja
- najslabija karika Android SDK
 - brzina mu nije najbolja strana
 - ne posjeduje zamjenu za neke komponente stvarnih uređaja



Alternativa (nije savršena):

- **GENYMOTION**
- <http://www.genymotion.com/>

Upoznavanje



Izrada AVD-a

Pokretanje AVD-a

Izrada aplikacije

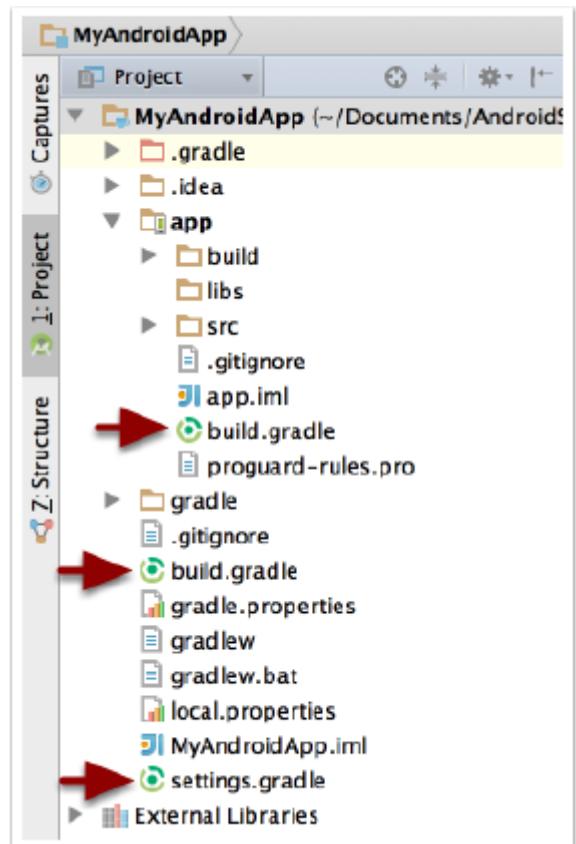
Gradle

- Pojavio se u svijet Androida s Android studiom
- Build sustav
- Open source
- Groovy jezik je baza (Maven i Ant imaju xml konfiguraciju)
 - Teško je reći da je neki build sustav bolji od drugoga

Gradle

Kako radi na Androidu?

- Postoje tri datoteke:



Gradle

- settings.gradle datoteka
 - pokazuje koji poddirektoriji će biti sadržani u build-u
- Primjer datoteke:

```
include ':app'
```

- include indicira da je 'app' poddirektorij jedini podprojekt
- Kada bi dodali dodatni projekt (npr. biblioteku), on bi također morao stajati unutar 'include' klauzule
- Primjer datoteke:
- `include ':app', ':lib'`

Gradle

- build.gradle datoteka (na razini cijelog projekta, „top level“)
 - Dodavanje Gradle distribucije
- Primjer datoteke:

```
buildscript {  
    repositories {  
        google()  
        jcenter()  
    }  
    dependencies {  
        classpath "com.android.tools.build:gradle:4.1.2"  
    }  
}  
allprojects {  
    repositories {  
        google()  
        jcenter()  
    }  
}  
task clean(type: Delete) {  
    delete rootProject.buildDir  
}
```

Gradle

build.gradle datoteka

- Gradle ne uključuje Android funkcionalnost po standardnim postavkama (ili kako bi rekli defaultu)
- Google pruža plugin za Gradle za jednostavnu konfiguraciju
- U datoteci je opisano da se plugin skida s jcenter i google repozitorija
 - Ostali repozitoriji su isto uključeni (npr. mavenCentral)
- Sekcija allprojects indicira da svi projekti koriste google i jCentar kao repozitorij
- Gradle omogućava i posebne taskove
 - clean je dodan i u kombinaciji s delete omogućava brisanje build direktorija

Gradle

- build.gradle datoteka (na razini projekta – app direktorij)
 - Konfiguracija projekta
- Primjer datoteke:

```
plugins {  
    id 'com.android.application'  
}  
  
android {  
    compileSdkVersion 30  
    buildToolsVersion "29.0.3"  
  
    defaultConfig {  
        applicationId "hr.android.myapplication"  
        minSdkVersion 30  
        targetSdkVersion 30  
        versionCode 1  
        versionName "1.0"  
  
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
    }  
  
    buildTypes {  
        release {  
            minifyEnabled false  
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),  
            'proguard-rules.pro'  
        }  
    }  
    compileOptions {  
        sourceCompatibility JavaVersion.VERSION_1_8  
        targetCompatibility JavaVersion.VERSION_1_8  
    }  
}  
  
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation 'androidx.appcompat:appcompat:1.2.0'  
    implementation 'com.google.android.material:material:1.3.0'  
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'  
    testImplementation 'junit:junit:4.+'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'  
}
```

Gradle

build.gradle datoteka

- Plugins dodaje Android plugin na build system
- BuildTypes podešava tipove builda projekta
- Dependencies govori o čemu ovisi projekt
 - Prva linija govori da sve biblioteke moraju biti jar datoteke
 - implementation dodaje dependency za cijeli projekt
 - testImplementation znači da će junit testno okruženje biti podešeno
 - androidTestImplementation dodaje dependency za testno okruženje

Gradle

Pregled Gradle datoteka



finish();



.pitinja?

Android

Aktivnosti i osnove sučelja

Sadržaj predavanja

- Android manifest

- Što je to i čemu služi?

- Aktivnosti

- Što je aktivnost?
 - Životni ciklus aktivnosti

- Osnove sučelja

- Layouti
 - Widgeti

Arhitektura Androida

Osnovni pojmovi

• Activity Manager i Fragment Manager

- Kontroliraju životni ciklus aplikacije

• Views

- Konstruiraju UI za Activitye i Fragmente

• Notification Manager

- Mehanizam za prikaz bitnih obavijesti korisnicima

• Content Providers

- Rad s podacima

• Resource Manager

- Rad s resursima

• Intenti

- Omogućavaju mehanizam za prebacivanje podataka između jedne ili više aplikacija

Manifest

AndroidManifest.xml

- Sadrži opće postavke aplikacije
- Koji API level se koristi
- Što aplikacija smije raditi
 - Pisati po storage-u
 - Skidati s Interneta
 - Pristup mreži
 - Vibriranje
 - Itd...
- Popisuje Activity-e, Servise, Recievere i sl. koje aplikacija koristi

Demo



Izgled manifest datoteke

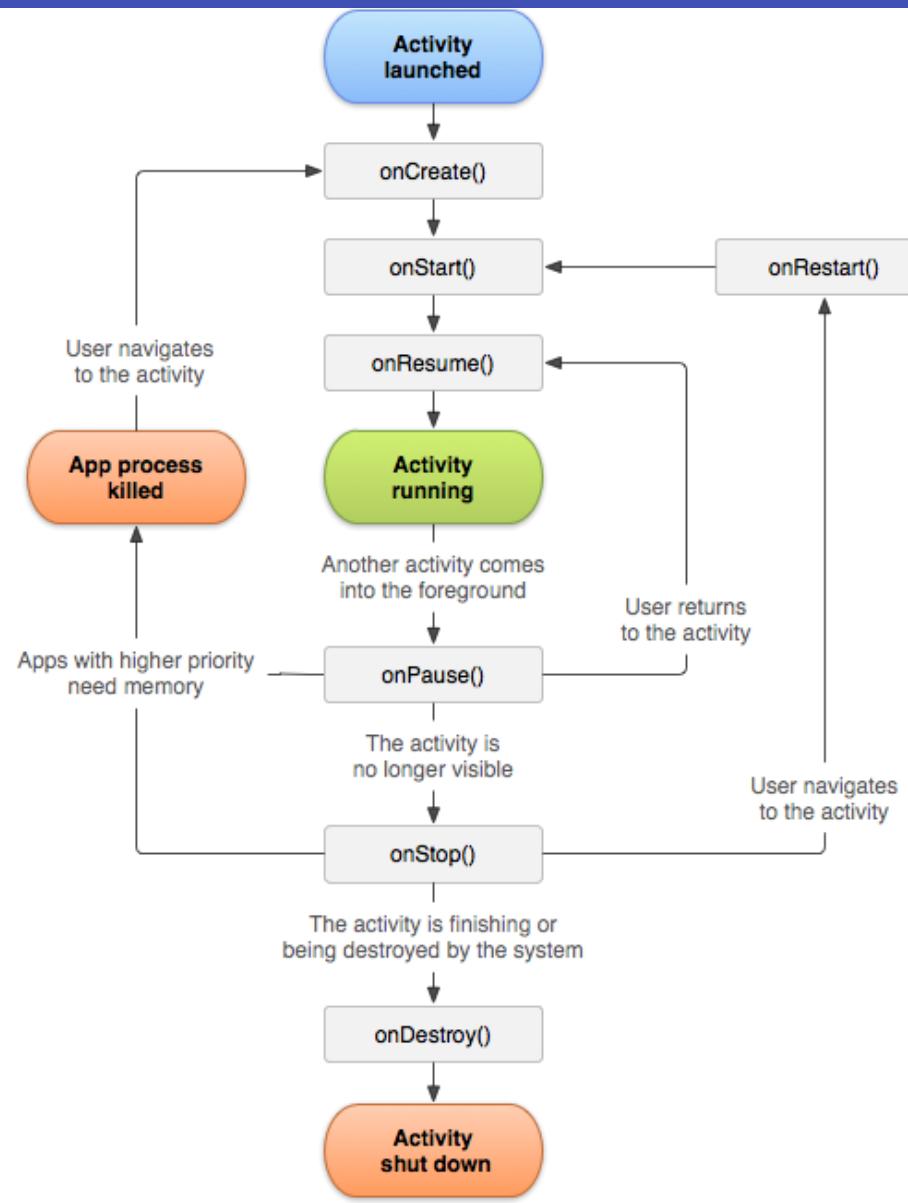
Aktivnosti

Engl. Activity

- Služi za interakciju s korisnikom
- Implementira funkcionalnost za GUI
- Svaka od aktivnosti prolazi kroz stanja koje poziva Android OS dok korisnik koristi aplikaciju
- Stanja pozivaju metode
 - onCreate
 - onStart
 - onResume
 - onPause
 - onStop
 - onRestart
 - onDestroy

Aktivnosti

Životni ciklus



Demo



Testiranje životnog ciklusa
aktivnosti

Metode aktivnosti

onCreate

- Inicijalizacija statičkih atributa
- Dohvat reference na grafičke komponente
- Kreiranje GUI-a pomoću setContentView metode
- Metoda prima klasu Bundle za pohranu prethodnih stanja – o njoj govora kasnije
- Najčešće korištena metoda
- Poziva se prva prilikom kreiranja aktivnosti

Metode aktivnosti

onStart

- Metoda se poziva kada se aktivnost prikazuje korisniku
- U ovoj metodi mogu se postaviti vrijednosti grafičkih komponenti ili stanja atributa
- Pristup posebnim podacima i resursima
 - Baza podataka
 - Mrežni resursi

Metode aktivnosti

onResume

- Metoda se poziva kada se aktivnost priprema za interakciju s korisnikom
- U ovom trenutku aktivnost je već na vrhu i ima fokus
- Unutar ove metode moguće je pokrenuti sve pauzirane resurse (npr. neki pozadinski thread)

Metode aktivnosti

onPause

- Poziva se kada neka druga aktivnost djelomično prekrije trenutnu aktivnost (nema fokus)
- U tom slučaju aktivnost prelazi u stanje "Paused"
- Potrebno je zaustaviti neke aktivne procese, pohraniti podatke i oslobođiti resurse
- Iz Paused stanja moguće je ići u stanja
 - Resumed
 - Stopped

Metode aktivnosti

onStop

- Poziva se kada aktivnost u potpunosti prestane biti vidljiva
- Poziva se nakon onPause metode
- Objekt Activity i dalje se nalazi u memoriji
- Potrebno je otpustiti resurse za rad s mrežom, datotekama, bazom podataka

Metode aktivnosti

onRestart

- Rijetko se koristi
- Unutar nje moguće je rekreirati resurse koji su otpušteni prilikom pozivanja onStop metode
- Najčešće se ti resursi kreiraju u onStart metodi koja će se pozvati nakon onRestart metode

Metode aktivnosti

onDestroy

• Poziva se prije uništavanja aktivnosti

• Aktivnost se može uništiti zbog:

- Pozivanja metode finish() iz aktivnosti
- Uklanjanje aktivnosti sa stoga (back tipka)
- Nedostatak resursa za normalno funkcioniranje uređaja

• Upozorenje za starije verzije (od verzije 8, API 26, na niže):

• Android ne garantira pozivanje ove metode prilikom oslobođanja resursa!

- Napraviti cleanup (npr. zatvoriti pozadinske dretve)
- Ne pohranjivati podatke unutar ove metode

Metode aktivnosti

Obvezno napraviti u svakoj metodi!

- Uvijek je potrebno pozvati implementaciju metode u nadklasi!
- Npr. unutar onDestroy() obavezno je potrebno pozvati super.onDestroy()

```
@Override  
protected void onDestroy() {  
    super.onDestroy();  
    Logger.info("onDestroy()");  
    Toast.makeText(this, "onDestroy()", Toast.LENGTH_SHORT).show();  
}
```

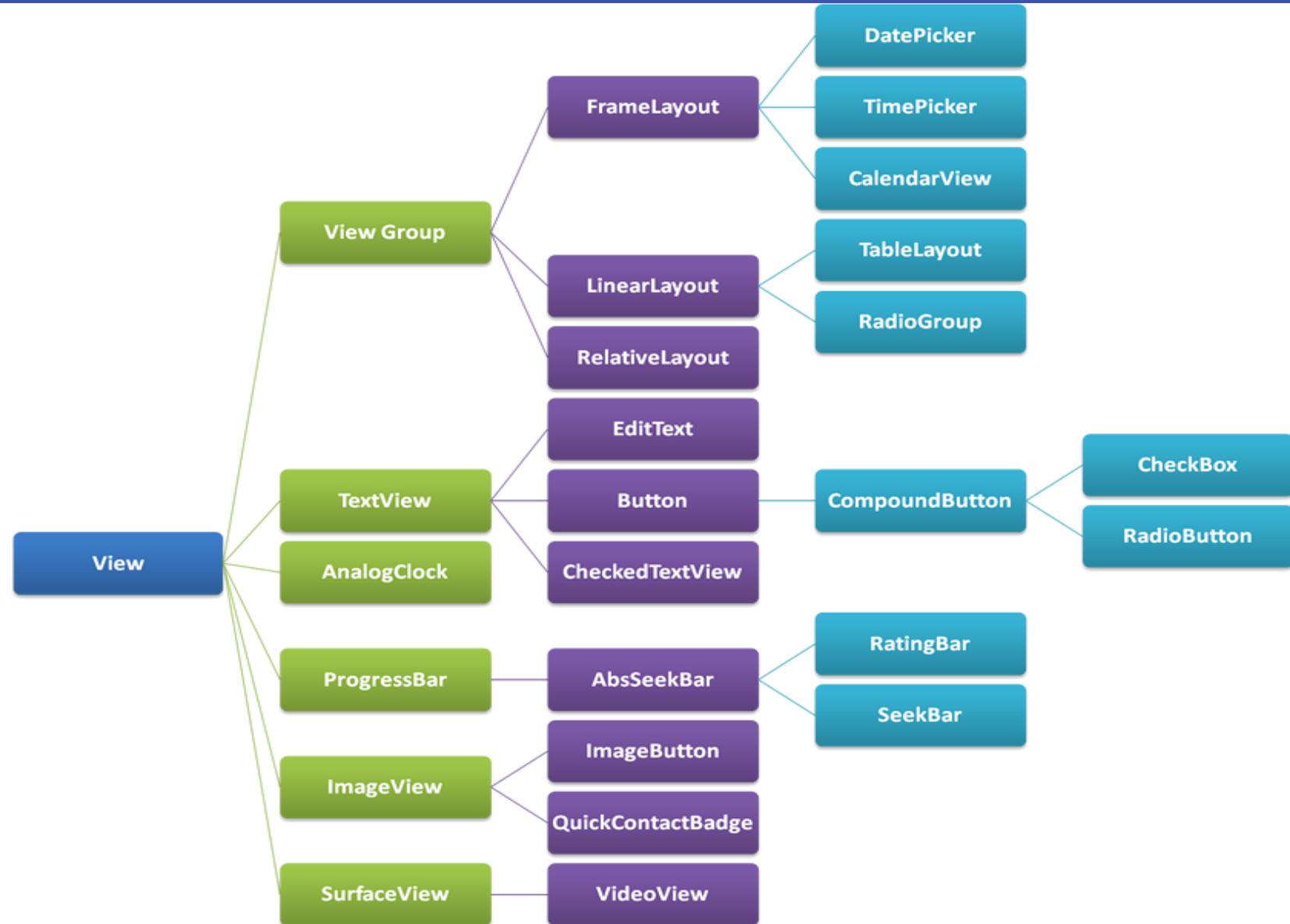
GUI

Osnovni elementi



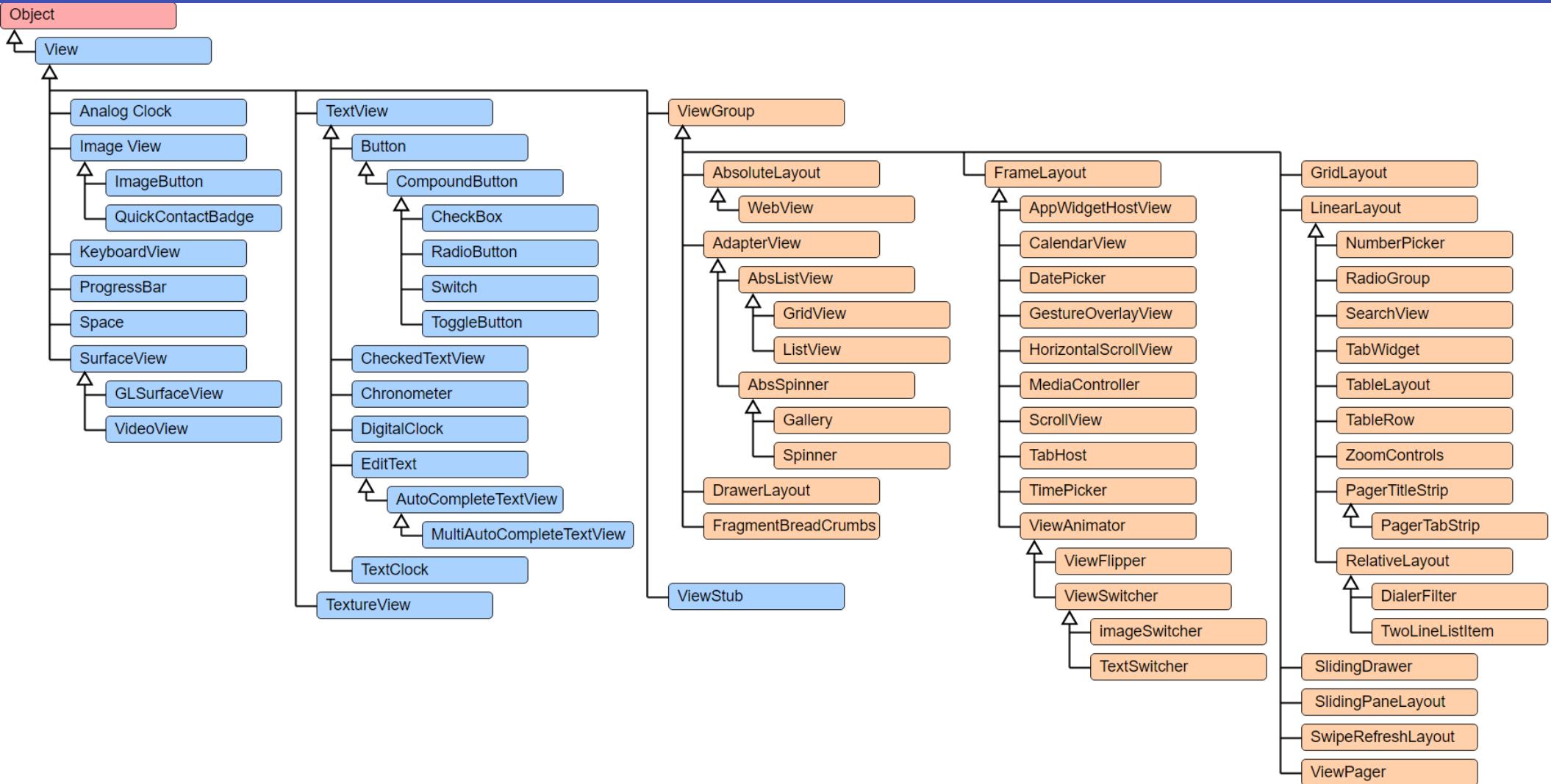
GUI

Nasljeđivanje nekih elemenata



GUI

Nasljeđivanje još nekih elemenata



View elementi

Svi elementi imaju

.ID – jedinstveno određuje element u cijeloj aplikaciji

.Pozicija s obzirom na element roditelj (px)

- getLeft(), getTop(), getWidth(), getHeight(), getRight(), getBottom()

.Padding – udaljenost od rubova

- setPadding(int, int, int, int), getPaddingLeft() getPaddingRight(),
getPaddingTop(), getPaddingBottom()

.Background – slika pozadine ili boja

.onClick – poziva metodu nakon klika

.Rotation – rotacija elementa

.Visibility – Vidljivost View elementa

- visible (vidljiv), invisible(nevidljiv, ali ostavlja prostor), gone(potpuno skriven, kao da ga se nikada nije ni dodalo)

Layout

Što radi?

- Definira izgled ekrana, widgeta i aplikacije

- Moguće ga je deklarirati pomoću

- xml datoteka (/res/layout)
- Kreiranjem klase unutar aktivnosti
- Oboje od navedenog

Layout

Mora imati definirane

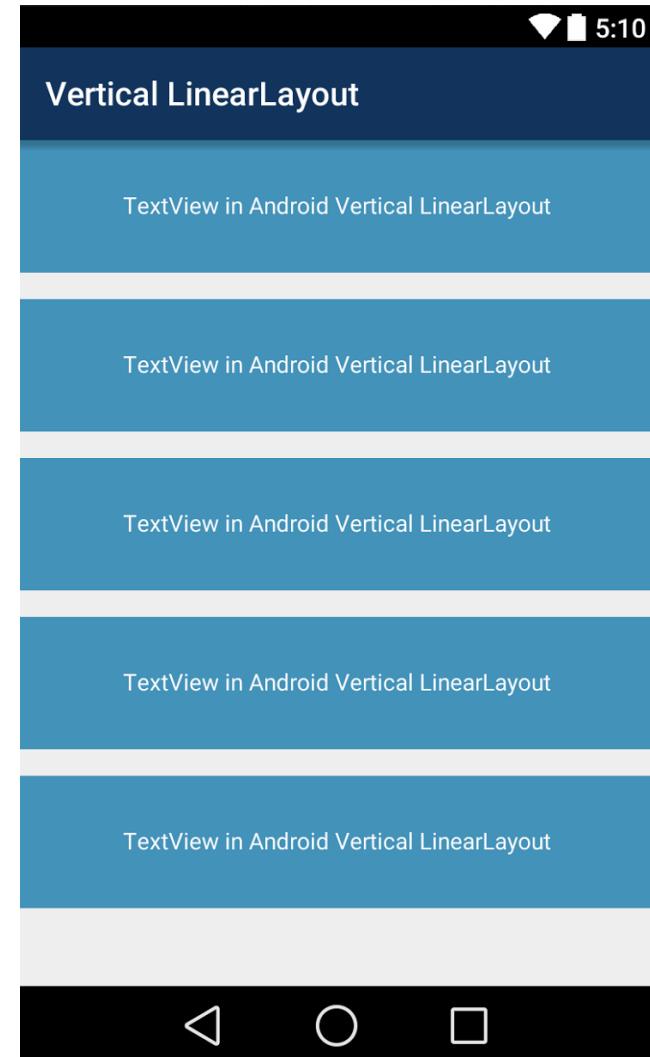
- Visinu i širinu

- match_parent – veličina kao i roditelj (minus padding)
- wrap_content – element je velik taman da obuhvati podelemente

Tipovi layouta

LinearLayout

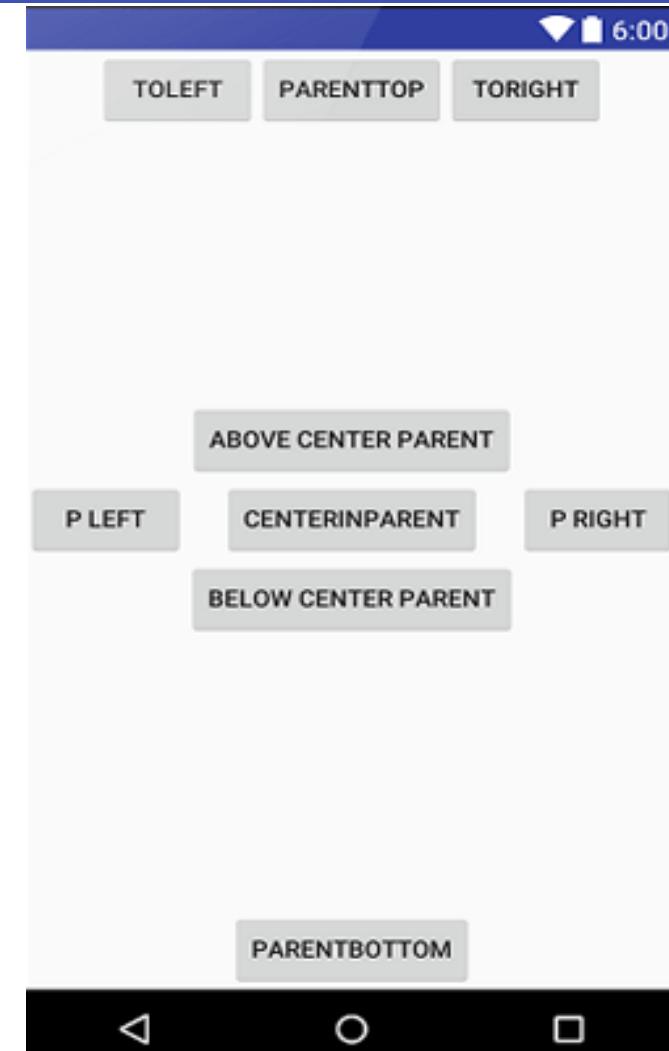
- Poravnava elemente u jednom smjeru
- Gravity – određuje poravnavanje elemenata
- Weight – koliko će prostora zauzeti neki element naspram ostalih elemenata



Tipovi layouta

RelativeLayout i ConstraintLayout

- Razmješta elemente u zavisnosti o nad elementu ili drugom elementu
- Ako se koristi nepravilno, GUI se lako može početi raspadati

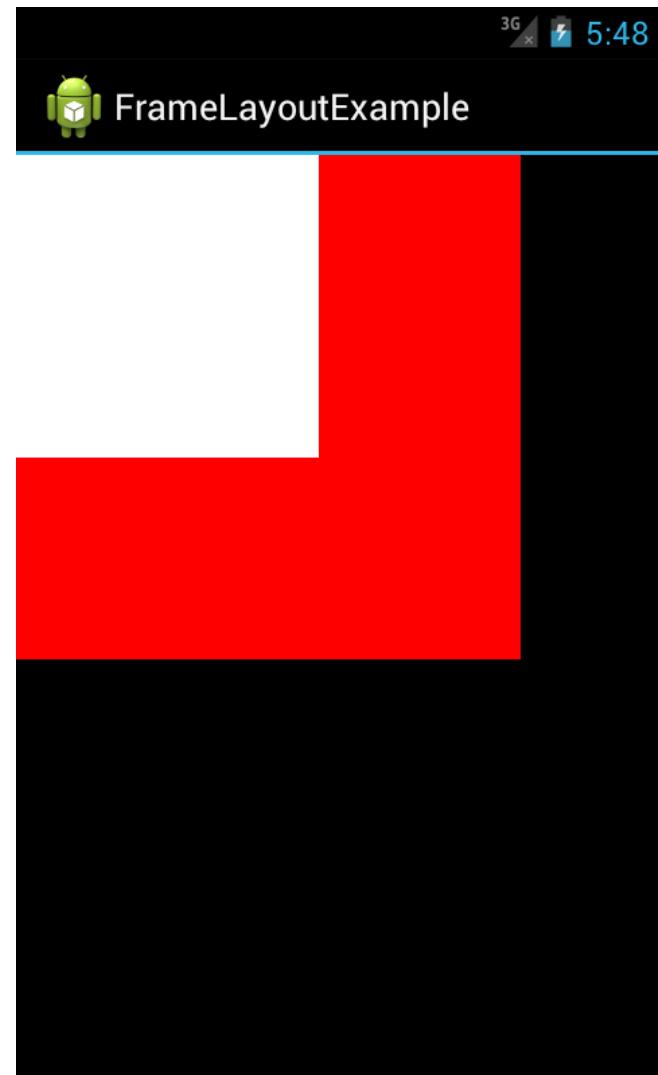


Više na <https://developer.android.com/reference/android/widget/RelativeLayout.html> i <https://developer.android.com/training/constraint-layout>

Tipovi layouta

FrameLayout

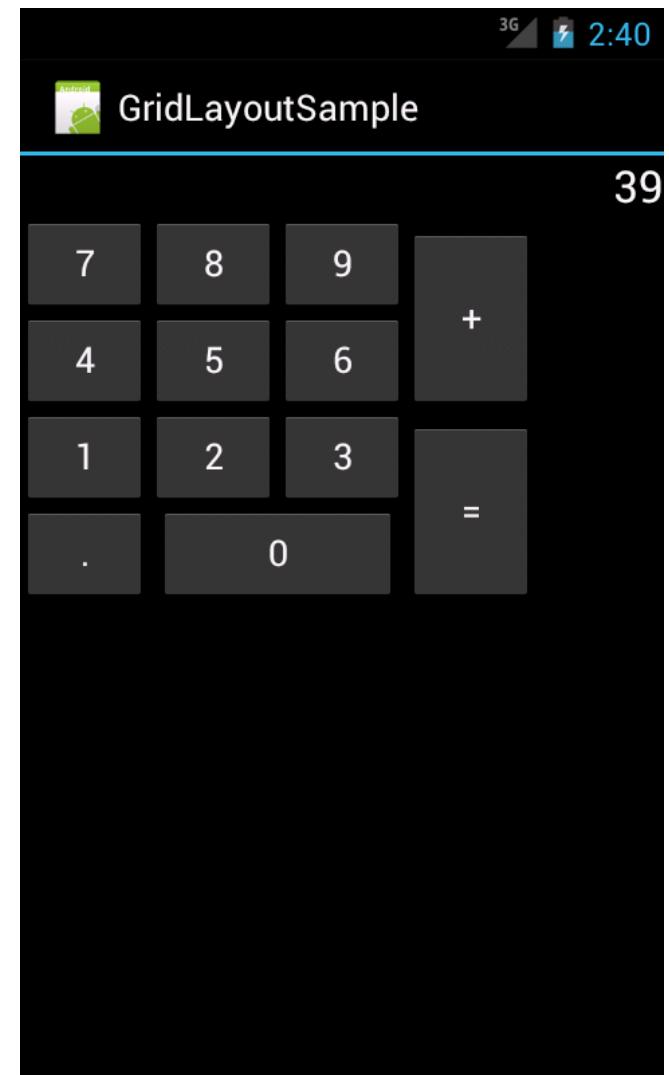
- Zamišljen je da sadrži samo jedan element
- Svaki novi element će biti nacrtan preko prethodnog u lijevi gornji ugao
- Koristi se najčešće kao *placeholder*



Tipovi layouta

GridLayout

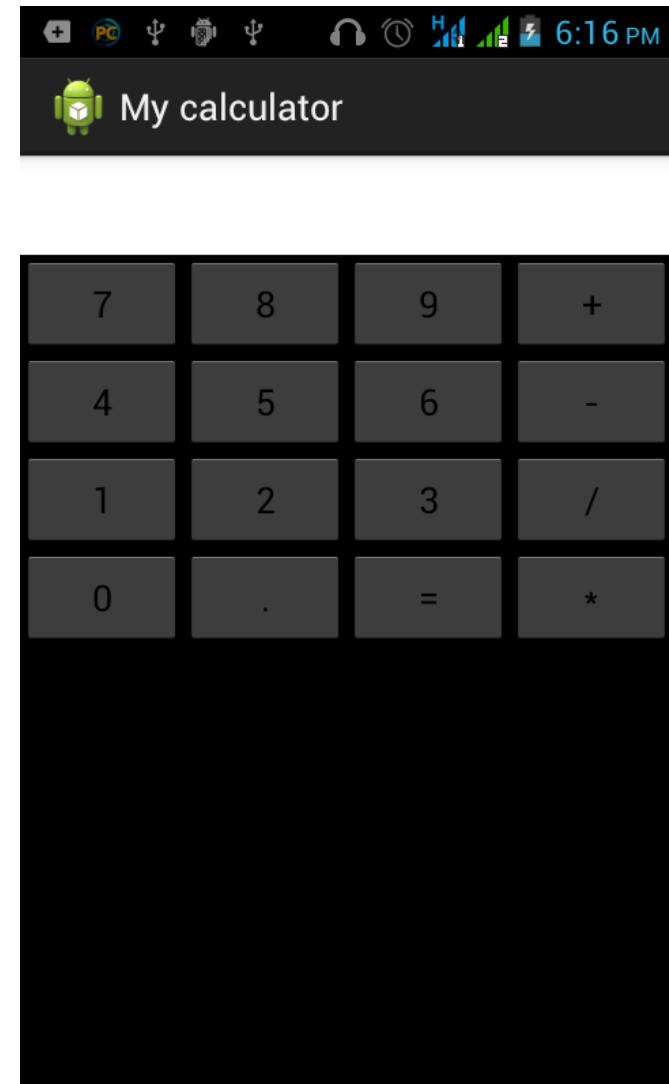
- Slaže elemente u pravokutnu mrežu
- Potrebno mu je odrediti broj redaka i kolona



Tipovi layouta

TableLayout

- Slaže elemente u pravokutnu mrežu
- Slično kao i GridLayout, ali drugačije pisan xml



Demo



Rad s layoutima kroz xml

Osnovni atributi layouta

Widgets

Neki od

- TextView
- EditText
- Button
- ToggleButton
- Switch
- CheckBox
- RadioButton
- ImageView
- ProgressBar

Widgets

TextView i EditText

- TextView služi za prikaz teksta
- EditText služi za mijenjanje teksta
- EditText je naslijeđen od TextView

Widgets

TextView i EditText parametri

| Atribut (XML) | Metoda (kod) | Opis |
|------------------|---------------------------|---|
| gravity | setGravity(int) | Orijentacija teksta u polju |
| height | setHeight(int) | Visina |
| hint | setHint(int) | Tekst dok je polje prazno |
| inputType | setRawInputType(int) | Vrsta podatka za unos |
| lines | setLines(int) | Broj linija |
| selectAllOnFocus | setSelectAllOnFocus(bool) | Označava cijeli tekst kada dođe u fokus |
| text | setText(CharSequence) | Tekst koji će se prikazati na kontroli |
| textColor | setTextColor(int) | Boja teksta |
| textSize | setTextSize(int, float) | Veličina teksta |
| textStyle | setTypeFace(Typeface) | Stil teksta (bold, italic, bolditalic) |
| typeface | setTypeFace(Typeface) | Izgled fonta (normal, sans, monospace) |
| width | setWidth(int) | Širina polja |

Demo



TextView i EditText primjeri

Widgets

Button

- Button može osim teksta prikazivati i sliku
- Standardna pozadina gumba je vezana uz temu iz API verzije
- Klasu Button nasljeđuje klasa CompoundButton iz koje proizlaze specijalni gumbi koji pamte stanja
 - CheckBox
 - RadioButton
 - Switch
 - ToggleButton

Demo



Button primjeri

finish();



.pitinja?

Android

Grafičko sučelje i resursi

Sadržaj predavanja

• Sučelja

- Toast, Snackbar
- View Binding
- Widgeti – nastavak

• Resursi

- Općenito
- Što ide u koju datoteku
- Alternativni resursi
- Assets

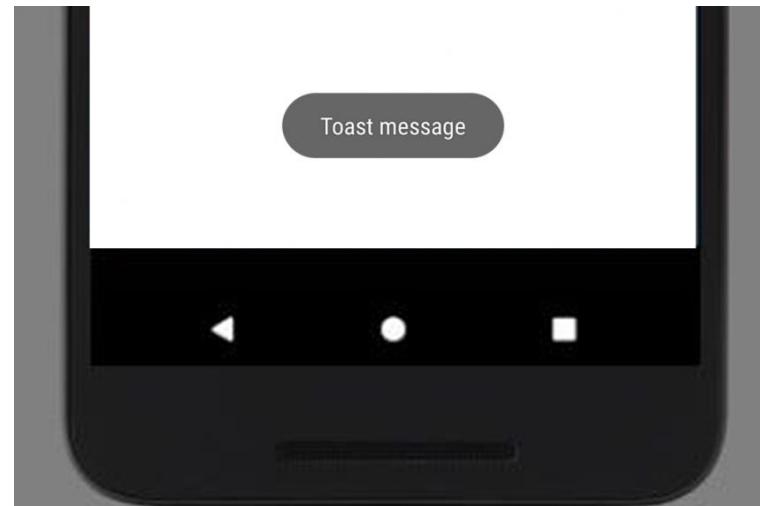
Kratka poruka

Toast

- Služi za prikazivanje kratkih poruka korisniku

- Jednostavni Toast se radi s metodom

- `Toast.makeText(context, text, duration)`
 - context – najbolje koristiti aktivnost, a moguće je koristiti i `getApplicationContext()`
 - text – tekst koji se prikazuje ili ID resursa
 - duration – konstanta iz Toast klase



Kratka poruka

Toast

- Toast je moguće napraviti i s proizvoljnim layoutom pomoću `toast.setView(layout)` metode
 - Deprecated od API verzije 30, moguće je samo napraviti u starijim verzijama!
 - Moguće koristiti Snackbar!

Kratka poruka

Snackbar

• Postoji nekoliko razlika između Toast-a i Snackbar-a:

- U starijim verzijama se Toast mogao prikazati i na vrhu ekrana, Snackbar ide uvijek na dno
- Toast nema akcija, a na Snackbar ih je moguće dodati (u obliku gumbiju, ali ih ne bi trebalo biti više od jednog)
- Kada se jednom Toast pokrene nije ga moguće maknuti dok se sam ne makne, Snackbar je moguće „swipe-ati“

Demo



Prikaz Toast-a

Prikaz Snackbar-a

Napredak

Progress bar

• ProgressBar

- Prikazuje napredak nečega

• SeekBar

- Moguće je ručno namještati vrijednosti

• RatingStar

- Izgled za rating u obliku zvjezdica (aplikacije npr.)

Ponavljanje

Dohvaćanje elemenata iz koda

• Ugrađena je metoda `findViewById`

- Potrebno joj je predati int kao parametar
- Do istog se dolazi preko generirane R klase
- Metoda vraća View kojeg je (najčešće) potrebno „kastati“ u odgovarajuću kontrolu iako kaže Android Studio da ne

• Primjer

- (SeekBar) `findViewById(R.id.seekBarWithListener);`

Demo



ProgressBar

SeekBar

RatingStar

Dohvat kontrola

Dohvaćanje elemenata iz koda

Ne postoji li nešto elegantnije?

View Binding

View Binding

- Jednostavniji pristup *View* elementima
- Generiraju se *binding* klase za svaki XML *layout* koje sadrže reference na sve elemente koji imaju ID
- Zašto je View Binding bolji?
 - **Null safety** – ne postoji rizik *null pointer exceptiona* ukoliko upišemo nepostojeći ID
 - **Type safety** – svaka *binding* klasa ima tipove koji odgovaraju XML-u i ne postoji rizik od *class cast exceptiona*

View Binding

Kako ga koristiti?

- Potrebno je uključiti View Binding u build.gradle datoteci na razini modula

```
buildFeatures {  
    viewBinding true  
}
```

View Binding

Kako ga koristiti?

- Unutar onCreate() metode je potrebno podesiti layout

```
private ActivityMainBinding binding;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    View view = binding.getRoot();
    setContentView(view);
}
```

- Gdje je ActivityMainBinding klasa vezana uz naziv aktivnosti/layout-a!

View Binding

Kako ga koristiti?

• Dohvat elemenata

```
binding.txtID.getText().toString();
```

```
binding.button.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) {
        }
});
```

Demo

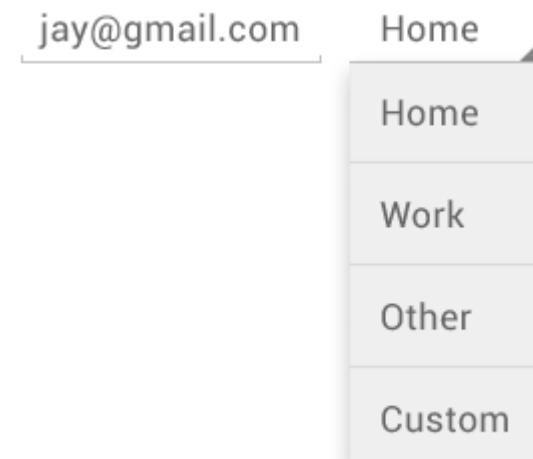


View Binding

Spinner

Nazad na widgete

- Ponašaju se kao JComboBox otprilike
- Služe odabiru jedne u nizu vrijednosti
- Moguće je dodati spinner-u predefiniranu listu vrijednosti kroz xml layout
- Također je moguće u programskom kodu definirati adapter koji koristi spinner



Demo



Predefinirani Spinner

Dinamički spinner

Prikaz slika

ImageView

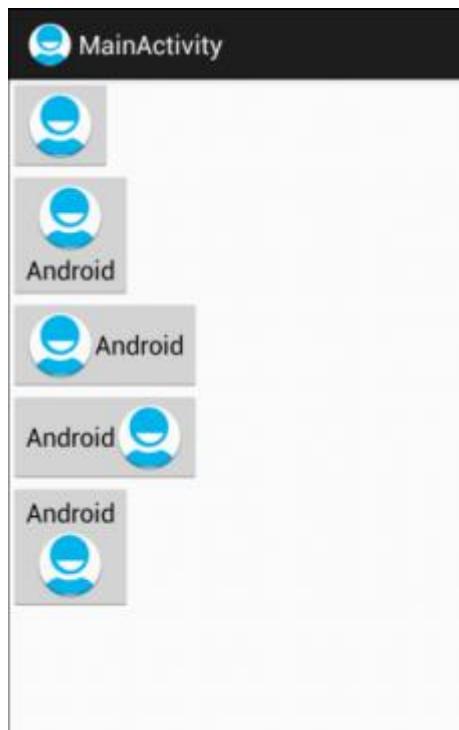
- Prikazuje sliku 😊
- Najvažniji atributi:

| Atribut (XML) | Metoda (kod) | Opis |
|---------------|--------------------------------------|---|
| cropToPadding | setCropToPadding(boolean) | Ako je true slika će biti odrezana da stane u okvir ImageView-a |
| maxHeight | setMaxHeight(int) | Maksimalna visina u pikselima |
| maxWidth | setMaxWidth(int) | Maksimalna širina u pikselima |
| src | setImageSource(int) | Resurs koji se prikazuje |
| tint | setColorFilter(int, PorterDuff.Mode) | Boja nijanse slike |

Prikaz slika

ImageButton

- ImageButton nasljeđuje ImageView
- Dodaje izgled gumba



Demo



ImageView

ImageButton

Resursi

Kako napraviti aplikaciju za sve te različite uređaje i ostali normalan?

Nikako. ☺

Resursi

Opća kultura

- Slike, stringovi, layouti, liste koji se koriste unutar aplikacije
- Svi resursi nalaze se u res/ direktoriju
- Dobivaju generirani id u R klasi
- Služe višejezičnosti i različitim tipovima uređaja
- Mogu biti univerzalni za sve uređaje
- Mogu biti različiti za različite uređaje
 - Veličina ekrana
 - Jezik

Resursi

Mjesta pohrane (nekih) resursa

.color/

- Definiranje boja nekog objekta ovisno o njegovom stanju

.drawable/

- Sadrži png, 9.png, jpg i gif datoteke (preporuča se png)

.mipmap/

- Ikone različitih gustoća

.layout/

- Izgled korisničkog sučelja

.menu/

- Sadrži xml datoteke koje opisuju izbornike

.raw/

- Sadrži datoteke koje se želi zadržati u izvornom formatu

Resursi

Mjesta pohrane (nekih) resursa

.values/

- xml datoteke koje sadrže neke jednostavne konstantne vrijednosti
- Imena datotekama su proizvoljna
- U svakoj je moguće definirati više vrsta vrijednosti
- Zbog standardizacije i jasnoće koriste se najčešće sljedeća imena
 - arrays.xml za nizove
 - colors.xml za boje
 - dimens.xml za dimenzije
 - strings.xml za tekstualne konstante
 - styles.xml za stilove

Resursi

Tablica dimenzija

- Dimenzijske jedinice koje se definiraju u XML-u sastoje se od broja i kratice mjerne jedinice (MJ)

MJ Opis

| | |
|----|---|
| dp | Density-independent pixel - mjerna jedinica kojom se definiraju svi elementi korisničkog sučelja. Jedan dp jednak je jednom pikselu na ekranu sa 160 piksela po inču (dpi). $px = dp * (dpi / 160)$ |
| sp | Scale-independent Pixels (sp) – slično kao dp ali je razmjerno s veličinom teksta, pa se i preporuča koristiti za definiranje veličine teksta. $sp = px * (dpi / 160) * scale$ |
| pt | Points - 1/72 od piksela |
| px | Pixels – stvarna vrijednost piksela na ekranu |
| mm | Millimeters - milimetar na temelju stvarne veličine ekrana |
| in | Inches – inč na temelju stvarne veličine ekrana |

Alternativni resursi

Zašto, čemu?

- Zbog velikih razlika u dizajnu i načinu upotrebe Android uređaja postoje alternativni resursi
- Omogućavaju povlačenje različitih resursa za različita okruženja
- Zbog alternativnih resursa postoji više različitih direktorija npr. drawable ovisno o dpi ekrana:
 - drawable-hdpi
 - drawable-ldpi
 - drawable-mdpi
 - drawable-xhdpi
 - drawable-xxhdpi

Alternativni resursi

Kvalifikatori direktorija

| Conf. | Opis |
|------------------|---|
| MCC i MCN | MCC (Mobilni kod zemlje), MNC(Mobilni kod operatora). Moguće je koristiti samo jedan od navedenih: npr. mcc219 ili mcc219-mnc01. Popis dozvoljenih moguće je naći na wikipediji: http://en.wikipedia.org/wiki/Mobile_country_code |
| jezik | Po iso standardu. Npr. hr-rHR ili samo hr, en-rUS ili samo en |
| Smjer layouta | ldrtl (layout-direction-right-to-left) ili ldltr (layout-direction-left-to-right) Za jezike koji se čitaju s desna na lijevo. (Api>17) |
| najmanja širina | sw<N>dp npr. sw320dp ili sw600dp. Najmanja moguća širina ekrana. Ako je za prikaz layouta potrebno barem 600dp-a, potrebno je napraviti direktorij res/layout-sw600dp (Api>13) |
| trenutna širina | w<N>dp npr. w320dp ili w600dp. Trenutna širina ekrana ovisno o landscape i portrait modu rada. (Api>13) |
| trenutna visina | h<N>dp npr. h320dp ili h600dp. Trenutna visina ekrana ovisno o landscape i portrait modu rada. (Api>13) |
| veličina ekrana | small (320x426 dp), normal (320x470 dp), large (480x640 dp), xlarge (720x960 dp) dp u zagradi označava minimalne vrijednosti nakon kojih će se resursi učitati. (Api>4) |
| omjer ekrana | long (WQVGA, WVGA, FWVGA) ili notlong (QVGA, HVGA, VGA) – podrška za widescreen ekrane. (Api>4) |
| Okrugli ekran | round, notround (Api>23) |
| Wide Color Gamut | Widecg (ima wide color gamut), nowidecg (nema wide color gamut) (Api > 26) |
| HDR | Highhdr (ima high dynamic range), lowdr (nema HDR) (Api > 26) |

Alternativni resursi

Kvalifikatori direktorija

| Conf. | Opis |
|---------------------|--|
| Orijentacija | port, land – je li uređaj u portretu ili landscapeu |
| UI mod | car, desk, television, appliance, watch – ovisi o tome na što je ukopčan uređaj (Api>8, television Api>13, watch Api>20) |
| Noćni mod | night, notnight – je li noć ili dan. (Api>8) |
| Gustoća piksela | ldpi (120dpi), mdpi (160dpi), hdpi (240dpi), xhdpi (320dpi, API>8), xxhdpi (420dpi, API>16), xxxhdpi (640dpi, API>18), nodpi (ne sklira slike), tvdpi (213dpi, API>13) |
| Touchscreen | notouch, finger |
| Tipkovnica | keysexposed (Uređaj ima tipkovnicu), keyshidden (Uređaj ima tipkovnicu, ali je sakrivena i uređaj nema softversku tipkovnicu), keyssoft (Softverska tipkovnica) |
| Metoda tipki | nokeys (uređaj nema hardver za unos), qwerty (uređaj ima tipkovnicu), 12key (uređaj ima tipkovnicu od 12 gumbića) |
| Navigacijske tipke | navexposed (Navigacijske tipke su dostupne korisniku), navhidden (Navigacijske tipke nisu dostupne. Npr, zatvoren je poklopac uređaja.) |
| Ne touch navigacija | nonav (Uređaj nema navigaciju), dpad (Uređaj ima d-pad), trackball (Uređaj ima trackball), wheel (Uređaj ima kotačić za navigaciju (rijetkost)) |
| Verzija APIja | verzija APIja na uređaju. Npr: v3, v4, v7, v14... |

Alternativni resursi

Pravila

- Moguće je specificirati više kvalifikatora odvojenih crticom
 - npr. drawable-hr-land (slike za uređaj na hrvatskom i u landscape modu rada)
- Kvalifikatori moraju biti po redoslijedu kao u tablici
 - npr. drawable-hr-land (slike za uređaj na hrvatskom i u landscape modu rada)
- Resursi se ne smiju ugnježđivati
- Vrijednosti su case insensitive
- Samo jedan kvalifikator je dozvoljen
 - Ne smije biti: drawable-hdpi-mdpi/

Kako resursi rade interno

R klasa

- Generira se na temelju kompletog res/ direktorija
- Svaki parametar je jedan element koji se rabi u aplikaciji i dodijeljena mu je generirana vrijednost
- Statičke konstante iz R klase potrebno je rabiti kao poveznicu između resursa i programskog koda
 - npr. setContentView(R.layout.main)

Resursi

Sintaksa

.Za dohvati iz programskog koda:

- Resources res = getResources();
- String hello = res.getString(R.string.hello);

.Za dohvati iz xml-a:

- android:textColor="@android:color/secondary_text_dark"

Demo



Pohrana resursa

Alternativni resursi

Pristup resursima

finish();



.pitinja?

Android

Stilovi, animacije, intenti, broadcast

Sadržaj predavanja

- Sučelja

- Stilovi
 - Animacije View elemenata

- Intenti

- Activity backstack
 - Taskovi

- Broadcast

- Slanje
 - Primanje

Stilovi

- Stil je kolekcija svojstava (propertija po hrv) koji specificiraju izgled i format View komponente ili prozora
- Stilovima je moguće odrediti sva svojstva poput visine, boje, boje pozadine i sl.
- Slična je logika kao kod CSS-a, ali je XML
- Stil može naslijediti neki drugi stil (parent)

Demo



Primjeri stilova

Animacije

View elemenata

- Animacije funkciraju slično kao u Flash-u i WPF frameworku
- Moguće je odrediti početnu vremensku točku, završnu vremensku točku i između istih promijeniti svojstva željenom elementu
- Moguće je mijenjati praktički sva svojstva View elemenata
- Animacije se pohranjuju u XML datotekama

Više o svim svojstvima datoteke na:

<http://developer.android.com/guide/topics/resources/animation-resource.html>

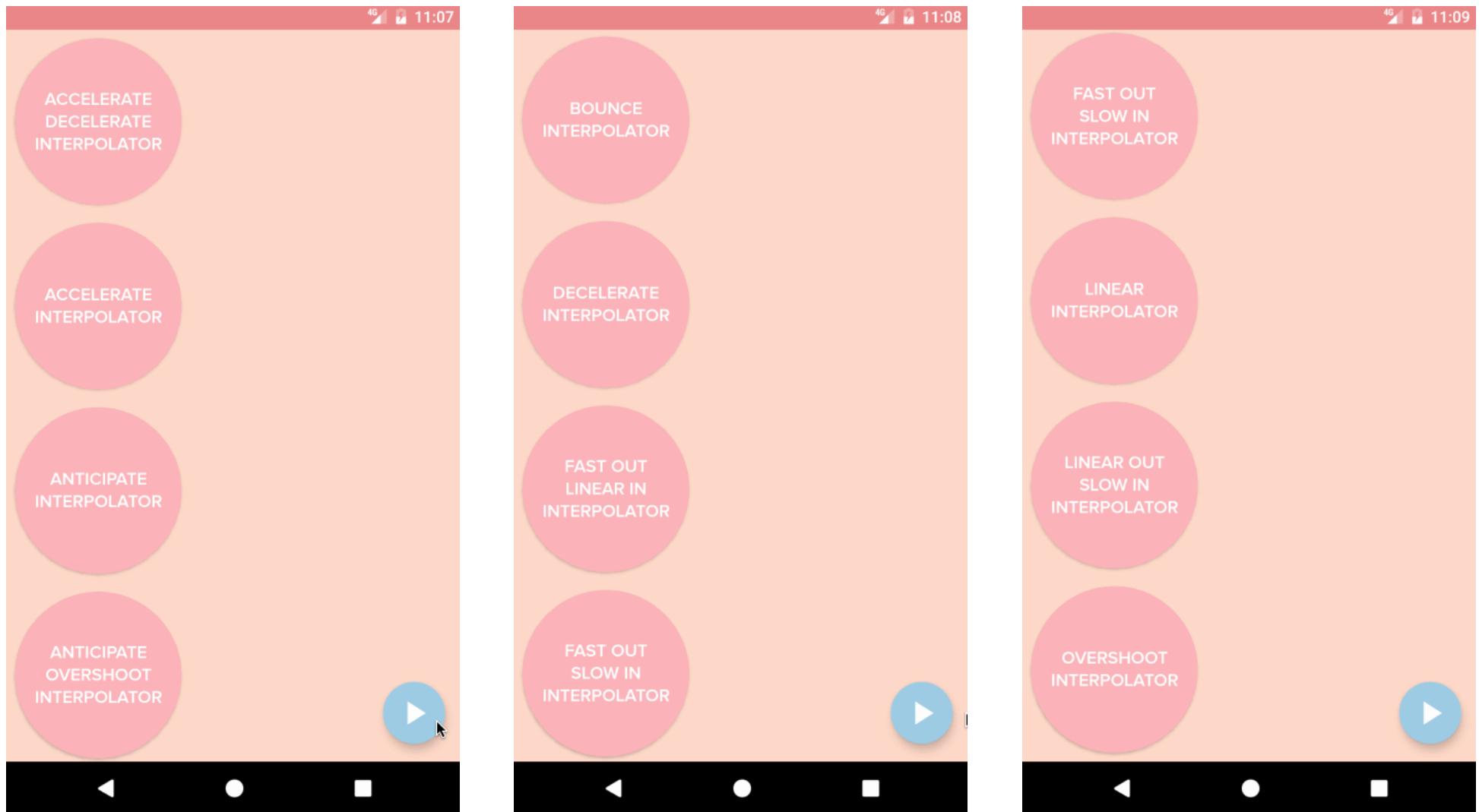
Animacije

View elemenata

- Animacije se pohranjuju u res/anim/ direktoriju
- Datoteka mora imati jedan element korijen!
- Animacije mogu biti sekvenčne ili istovremene
 - Npr. text je moguće prvo pomaknuti a potom rotirati, a također je moguće obje radnje napraviti istovremeno
- Moguće je odrediti interpolator (način na koji se transformacija obavlja)
 - Ugrađeni interpolatori: AccelerateDecelerateInterpolator, AccelerateInterpolator, AnticipateInterpolator, AnticipateOvershootInterpolator, BounceInterpolator, CycleInterpolator, DecelerateInterpolator, LinearInterpolator, OvershootInterpolator, PathInterpolator

Animacije

Interpolatori



Izvor: <https://thoughtbot.com/blog/android-interpolators-a-visual-guide>

Animacije

Svojstava (property-a)

- Moguće je animirati elemente koji nisu View
- Za razliku od View animacije, može animirati sva svojstva
- View animacije imaju bug zbog kojeg nije moguće kliknuti na animirani element dok je u procesu animacije
- Robusnije su
- Imaju više programskog koda ☺

Demo



Primjeri animacija

Intent

Namjera da se nešto učini

- Intent objekt je skup informacija koji se prosljeđuje sustavu
- Sadrži informacije o pokretanju aktivnosti ili servisa
- Postoje dvije grupe intenta:
 - **Eksplicitni** – točno se zna koju aktivnost će pokrenuti (koristi se unutar same aplikacije)
 - **Implicitni** – pokreću aktivnosti drugih aplikacija jer ne imenuju točnu aktivnost koju treba pokrenuti, već što bi sustav trebao napraviti (npr. nazvati neki broj ili otvoriti neki dokument)

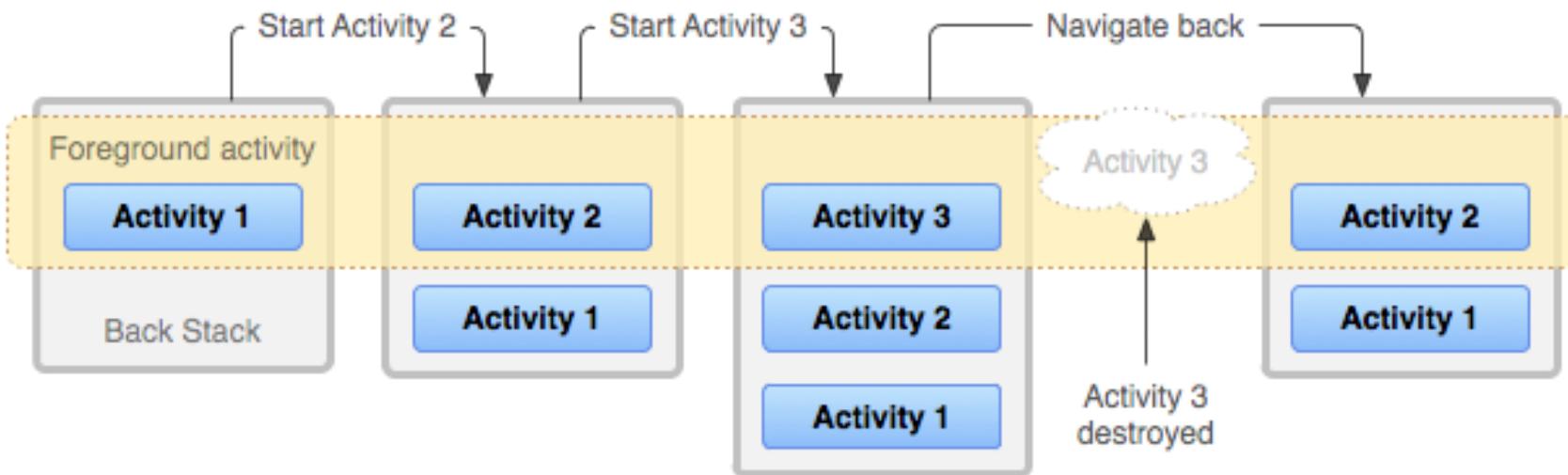
Task i back stack

Opće informacije

- Svaka aktivnost je dizajnirana oko neke funkcionalnosti
- Rad s aplikacijom podrazumijeva promjenu aktivnosti
- Task je kolekcija aktivnosti koje korisnik koristi kako bi obavio neku zadaću
- Aktivnosti su poredane u stog (back stack) po redoslijedu otvaranja aktivnosti

Back stack

- Back stack se ponaša po arhitekturi LIFO (last in first out)



Tasks

- Ako korisnik pritišće tipku “back” dok ne izađe iz aplikacije, sve aktivnosti će biti zatvorene
- Ako stisne tipku “home”, sve aktivnosti ostaju upaljene i to se naziva jednim taskom
- Nakon toga, korisnik se ponovo može vratiti gdje je stao, nastaviti i opet imati aktivnosti na back stacku (ukoliko ih Android ne uništi prije ☺)

Tasks

Upravljanje taskovima

.taskAffinity

- deklarira kojem tasku aktivnost preferira pripadati
- sve aktivnosti jedne aplikacije preferiraju pripadati istom zadatku, no moguće je definirati da aktivnosti različitih aplikacija preferiraju pripadati istom zadatku

.launchMode

- definira kako će aktivnost biti pokrenuta u zadatku

.allowTaskReparenting

- postavlja da li se aktivnost može prenijeti iz jednog zadatka (u kojem je pokrenuta) u drugi zadatak

Tasks

Upravljanje taskovima

.clearTaskOnLaunch

- aktivnost nikada ne zadržava stanje, bez obzira na trajanje neaktivnog stanja

.alwaysRetainTaskState

- aktivnost zadržava stanje i nakon dužeg perioda neaktivnog stanja, ali samo ukoliko je root aktivnost povratnog stoga

.finishOnTaskLaunch

- aktivnost ostaje dio zadatka samo u trenutnom korištenju (dok je zadatak aktivan). Ukoliko zadatak priđe u neaktivno stanje aktivnost se gubi

Tasks

Načini pokretanja - launchMode

.standard

- kreira se nova instanca aktivnosti i moguće je imati više instanci iste aktivnosti unutar zadatka
- aktivnost može biti instancirana unutar više zadataka

.singleTop

- ako instanca aktivnosti već postoji na vrhu povratnog stoga sustav preusmjerava na tu instancu i poziva metodu onNewIntent() aktivnosti
- Kada se nova aktivnost ne instancira, ne sprema se i ne može se dohvatiti u povratnom stogu.

Tasks

Načini pokretanja - launchMode

.singleTask

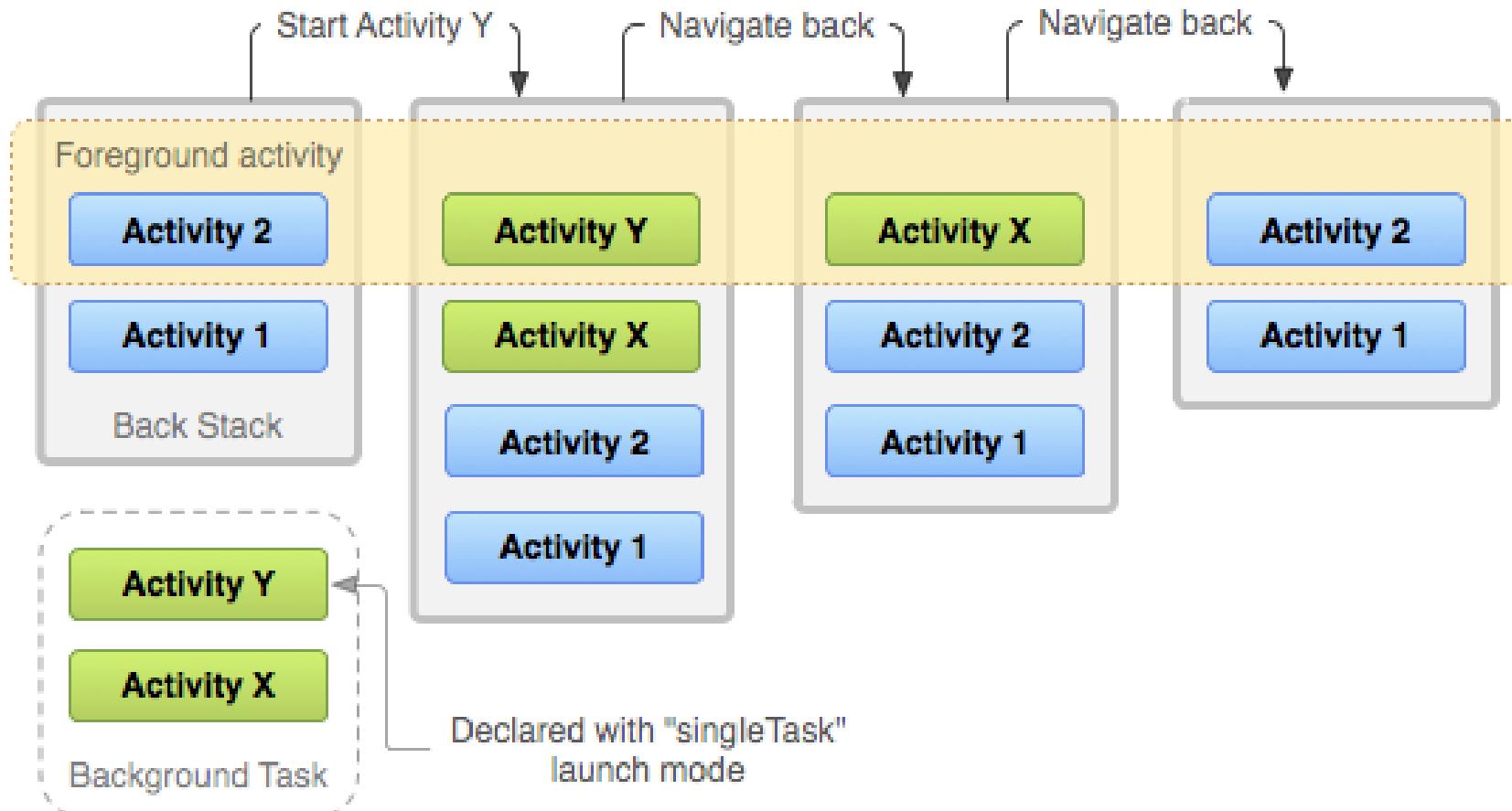
- Sustav kreira novi zadatak i kreira novu instancu na vrhu povratnog stoga tog zadatka
- Ako instance aktivnosti već postoji u nekom drugom tasku sustav ne kreira novi task, nego preusmjerava na već postojeći task na tu instancu i poziva metodu onNewIntent() aktivnosti

.singleInstance

- Slično kao i singleTask s razlikom da sustav ne pokreće niti jednu novu aktivnost u zadataku u kojem se nalazi aktivnost
- Aktivnost je uvijek jedini član tog zadatka

Tasks

SingleTask način pozivanja



Tasks

Pokretanje kroz Intent

.FLAG_ACTIVITY_NEW_TASK

- isto ponašanje kao i singleTask definiran u manifestu

.FLAG_ACTIVITY_SINGLE_TOP

- isto ponašanje kao i singleTop definiran u manifestu

.FLAG_ACTIVITY_CLEAR_TOP

- ako aktivnost koju se pokreće već postoji unutar povratnog stoga sve aktivnosti iznad nje u stogu se oslobođaju (destruction)
- aktivnost se postavlja kao aktivna na vrh povratnog stoga i poziva se metoda onNewIntent() aktivnosti

Demo



Kreiranje intenta

Pozivanje aktivnosti

Kako radi backstack?

Broadcast

Što je i čemu služi?

• Osnovna komponenta Android sustava

- (zajedno s aktivnostima, servisima i pružateljima sadržaja)
- aktivnost nikada ne zadržava stanje, bez obzira na trajanje neaktivnog stanja

• Služi praćenju promjena na sustavu (najčešće)

- Npr. stanje baterije, dostupnost mreže i slično

• Broadcast je moguće poslati i primiti

- Najčešće se koristi kod dobivanja nekih sistemskih promjena

• Često se koristi za neke notifikacije korisniku i gašenje procesa ako je baterija prazna

Broadcast receiveri koji se mogu registrirati (API level 26+):

<https://developer.android.com/guide/components/broadcast-exceptions>

Broadcast

Broadcast prijemnici

• Dva tipa:

- Definirani u manifest datoteci (više baš i ne rade. Od API>26 ☺)
- Definirani kroz programski kod

• Dodaje im se filter na koji se okida broadcast

• Potrebno je nadjačati onReceive metodu

• Mogu loviti custom broadcaste i sistemske broadcaste

Broadcast

Slanje

- Slično je kao i pozivanje aktivnosti
- Vrši se pomoću Intent klase
- Pomoću metode sendBroadcast se šalje iz aktivnosti
- Mogu se slati custom broadcasti

Demo



Kreiranje broadcasta

Hvatanje broadcasta

Hvatanje sistemskih
broadcastova

finish();



.pitinja?

Android

Dijalozi, izbornici, Bundle

Sadržaj predavanja

- Izbornici

- Context menu
 - Options menu

- Dijalozi

- Alert dijalog
 - Ostali dijalozi

- Bundle

Menu

Izbornici

- Izbornici su učestali način korisničke interakcije s aplikacijom jer pružaju poznato i jednostavno iskustvo.
- Za sve vrste izbornika koristi se xml format
- Izbornici se mogu raditi i direktno u Java kodu
 - Loša praksa zbog preglednosti
- XML se pohranjuje u res/menu direktorij

Menu

Tri vrste elemenata

- <menu>

- Glavni, root element koji definira jednu Menu klasu i može sadržavati jednu ili više item ili group elemenata

- <item>

- Jedna opcija menu-a (u kodu MenuItem klase)
 - Može sadržavati i menu element i tako se otvara podizbornik

- <group>

- Opcionalni element kojim se opcije (item-i) mogu grupirati kako bi se grupa mogla sakriti ili aktivirati

Item

Najvažniji atributi *item* elementa

| Atribut (XML) | Opis |
|---------------|---|
| icon | Postavlja ikonu na opciju iz drawable resursa |
| title | Naslov opcije |
| showAsAction | Definira da li će se i kako opcija prikazati u ActionBar-u. (ifRoom, withText, never, always, collapseActionBarView) |
| checkable | Hoće li opcija imati pridijeljen CheckBox |
| checked | Hoće li CheckBox biti označen (ako ga ima) |
| visible | Vidljivost elementa |
| enabled | Je li element omogućen za korištenje |

Group

Najvažniji atributi *group* elementa

| Atribut (XML) | Opis |
|-------------------|--|
| checkableBehavior | Definira ponašanje kod označavanja za sve elemente grupe: <ul style="list-style-type: none"> •none – neće imati opciju označavanja •all – svi elementi mogu biti označeni (koristit će se CheckBox elementi) •single – samo jedan element grupe može biti označen (radio button elementi) |
| menuCategory | Postavlja kategoriju grupe kojom se definira prioritet. (container, system, secondary, alternative) |
| orderInCategory | Definira redoslijed elemenata unutar kategorije |
| visible | Vidljivost elementa |
| enabled | Je li element omogućen za korištenje |

Tipovi menu-a

Options menu

- Najčešći, trebao bi sadržavati opcije kao što su pretraži, pošalji, postavke i sl.
- Pozicija ovisi o verziji androida
 - ver<3.0 – na dnu ekrana nakon pritiska “menu” tipke
 - ver>=3.0 – izbornik se integrira u akcijsku traku na vrhu ekrana
- Izbornik se postavlja na aktivnost pomoću MenuInflatera

```
@Override  
public boolean onCreateOptionsMenu(Menu menu) {  
    getMenuInflater().inflate(R.menu.activity_menu1, menu);  
    return true;  
}
```

Tipovi menu-a

Context menu

- Prikazuje se u obliku padajućeg izbornika
- Aktivira se na dugi pritisak na neki View element
- Registrira se pomoću registerForContextMenu() metode kojoj se predaje View
- Nakon dugog pritiska na element poziva se metoda

```
@Override  
public void onCreateContextMenu(Menu menu, View v,  
        ContextMenuInfo menuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo);  
    getMenuInflater().inflate(R.menu.activity_menu1, menu);  
}
```

Demo



Options menu

Context menu

Dijalozi

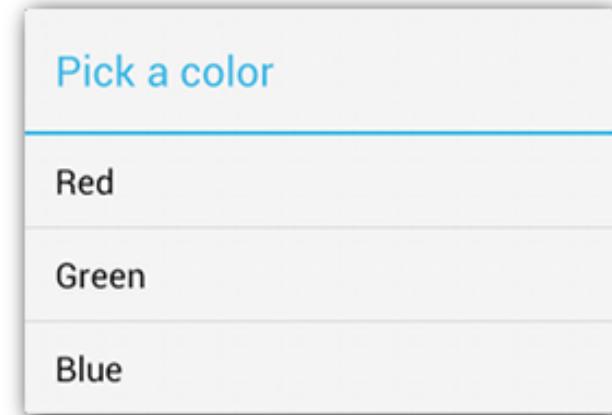
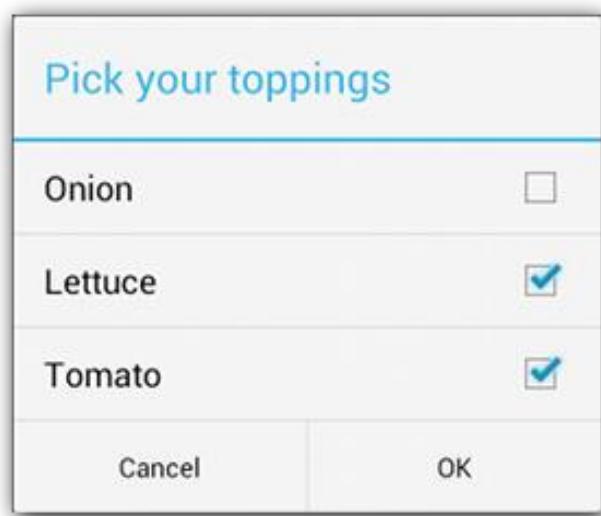
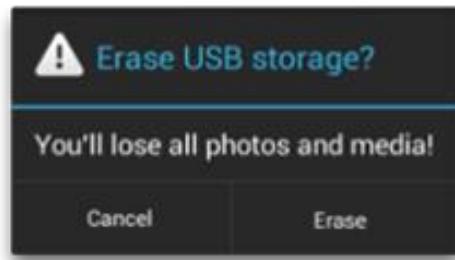
Kako izgledaju

Text message limit

Set number of messages to save:

499
500
501

Cancel Set



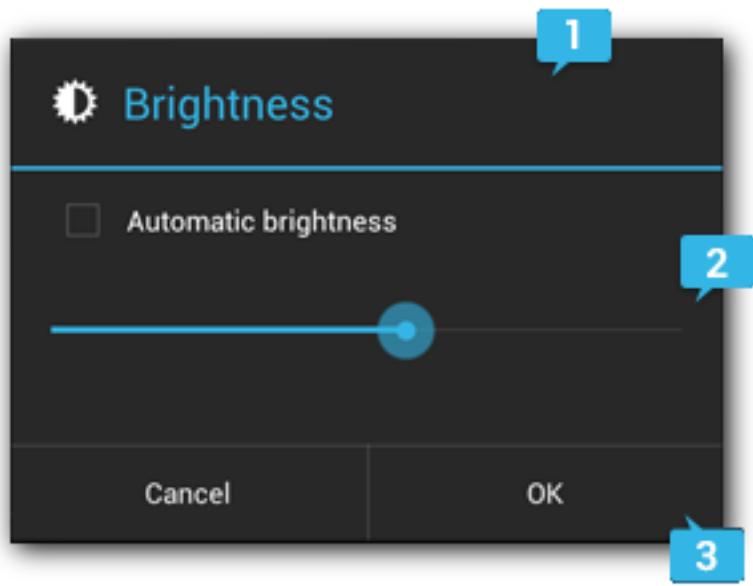
Dijalozi

Opće informacije

- Mali prozori koji se otvaraju da bi se korisnika upozorilo na neku odluku
- Bazna klasa svakog dijaloga je klasa Dialog
 - Treba je izbjegavati instancirati direktno (nasljeđivanje je o.k.)
 - Najčešće se koristi jedna od klasa za dijalog
 - AlertDialog – sadrži naslov, do tri tipke i layout za sadržaj
 - DatePickerDialog i TimePickerDialog – predefinirano sučelje za unos datuma i vremena
- Kontenjer za dialoge trebao bi biti DialogFragment

AlertDialog

Izgled



1. Naslov

- Opcionalan dio koji se treba koristiti kada se iz teksta ne može zaključiti o kakvom se dijalogu radi

2. Sadržaj

- Može prikazivati tekst, listu ili korisnički layout

3. Tipke

- Sadrži do tri gumba

AlertDialog

Korištenje

- .Unutar klase **AlertDialog** nalazi se klasa **Builder** pomoću koje se dijalog izrađuje po builder patternu
- .Metodama **setMessage()** i **setTitle()** postavlja se naslov i poruka, a metodama **setPositiveButton()**, **setNeutralButton()** i **setNegativeButton()** se postavljaju tipke.
- .Za prikaz samog dijaloga potrebno je nad njegovom instancom pozvati metodu **showDialog()**
- .Za dijalog koji će se sastojati od liste opcija za odabrat dovoljno je postaviti niz podataka za listu pomoću metode **setItems()** ili **setSingleChoiceItems()**, a za dijalog koji će imati listu za odabir više opcije postavlja se metodom **setMultiChoiceItems()**. Na dijalog se može postaviti i proizvoljan layout metodom **setView()**

Demo



AlertDialog

Bundle

Opće informacije

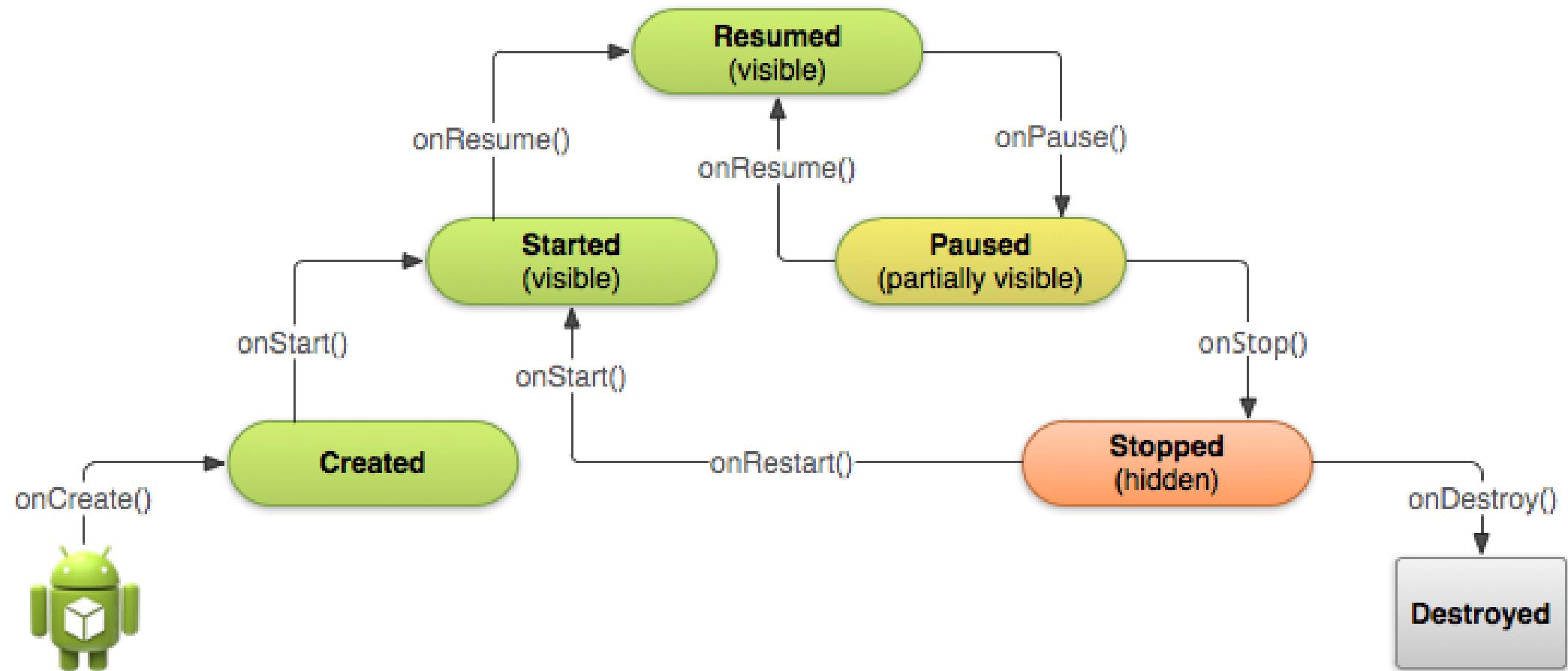
- Serijalizacija, komunikacija među procesima i predavanje vrijednosti se vrše Bundle objektom
- Serijalizacija u Javi je prespora za Android i zato je najbolje koristiti Parcelable sučelje
- Bundle je zapravo obična mapa ključeva povezana s objektima
- U Bundle je moguće koristiti i objekt koji implementira sučelje Serializable

Pohrana stanja aktivnosti

- Android može otpustiti sve aktivnosti koje se ne prikazuju i sve vrijednosti moraju biti pohranjene
- Prilikom npr. Rotacije aktivnost se rekreira
 - Polja imaju iste vrijednosti jer svi objekti view elemenata automatski pohranjuju svoje stanje
 - Potrebno je pozvati nadmetode koje automatski pohranjuju i rekreiraju GUI ukoliko elementi imaju definiran ID
 - `onRestoreInstanceState()`
 - `onSaveInstanceState()`

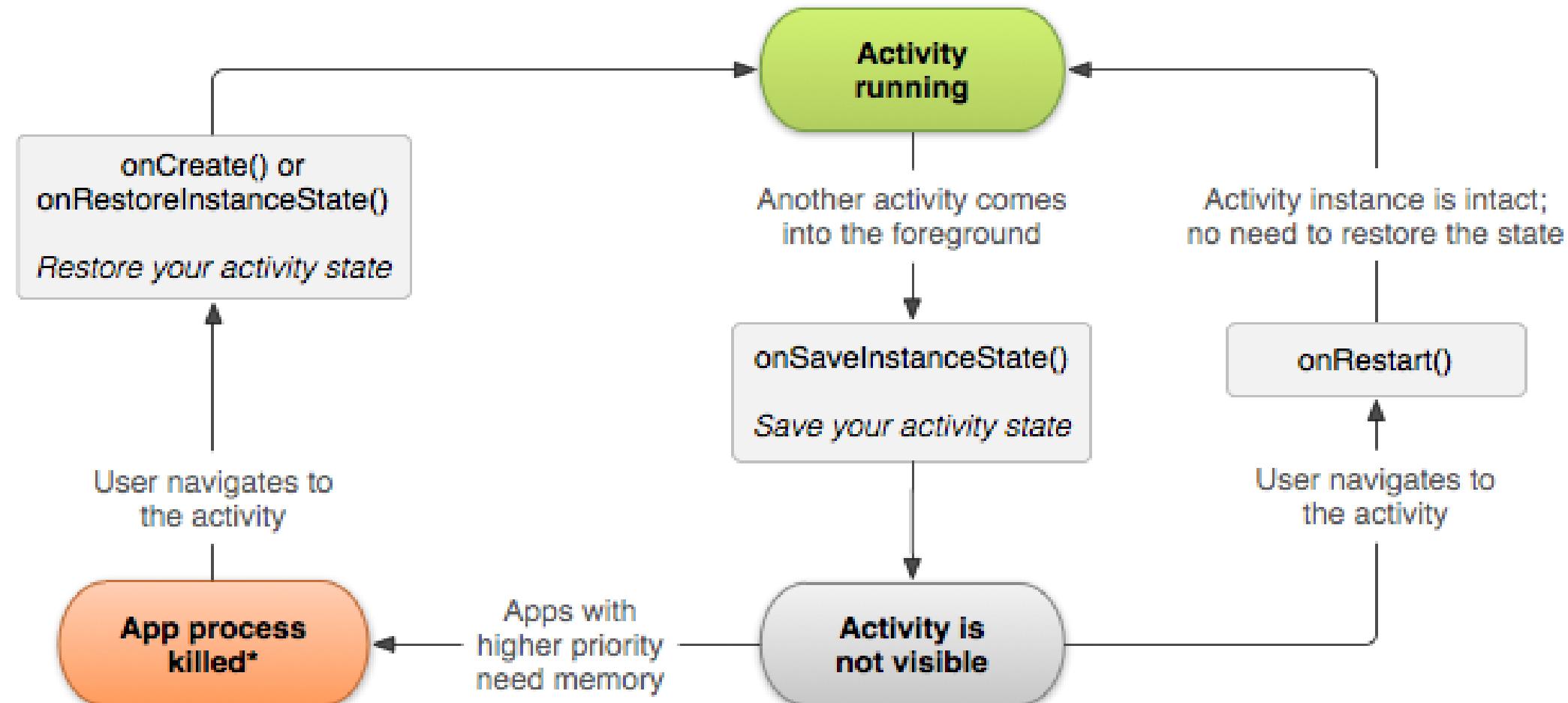
Pohrana stanja aktivnosti

Sjetimo se životnog ciklusa aktivnosti



Pohrana stanja aktivnosti

Metode onRestoreInstanceState i onSaveInstanceState



*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved

onSaveInstanceState()

Kada, što, kako?

- Metoda se poziva prije nego aktivnost ode u pauzirano stanje
- Metodi se prosljeđuje Bundle objekt u kojeg je moguće sve pospremiti
- Poziv metodi nije garantiran
 - npr. vraćanje korisnika po back stacku tipkom back
 - samim time u Bundle se ne bi trebalo spremati bitne podatke, već prvenstveno podatke za UI

onRestoreInstanceState()

Kada, što, kako?

• Poziva se kod rekreiranja aktivnosti nakon što je bila otpuštena

- promjena konfiguracije okoline (orientacija ekrana, prikaz tipkovnice, promjena jezika i sl.)
- oslobođenje resursa

• Metoda se ne zove kada je aktivnost prvi put pozvana

• Ako je potrebno nešto obaviti svaki put kada se aktivnost poziva koristi se onCreate()

Intent putExtra i getExtra

Pohranjivanje u bundle prilikom promjene stanja

- Služi za dodavanje dodatnih informacija Intentu kroz Bundle objekt
- Moguće je dodavanje raznih tipova podataka uključujući i Parcelable i Serializable
- Metode za preuzimanje se razlikuju (nažalost) Npr:
 - getInt(key)
 - getParcelable(key)
 - getString(key)

Parcelable

Android style serijalizacija

• Klasa koja implementira Parcelable mora implementirati sljedeće metode

- describeContents() – vraća hashCode kako bi se znao vratiti točno određeni objekt
- writeToParcel(Parcel dest, int flags) – zapisuje zapis u dest varijablu

• Mora implementirati CREATOR:

```
public static final Parcelable.Creator<MyObject> CREATOR = new Parcelable.Creator<MyObject>() {  
    @Override  
    public MyObject createFromParcel(Parcel source) {  
        return new MyObject(source);  
    }  
    @Override  
    public MyObject[] newArray(int size) {  
        return new MyObject[size];  
    }  
};
```

Demo



Bundle

Aktivnost s rezultatom

Kako vratiti informaciju nazad?

- Pozivanje aktivnosti ne mora biti jednosmjerno
- Aktivnost se može pokrenuti pomoću metode „startActivityForResult”
- Npr. aplikacija može pokrenuti kameru i tražiti fotografiju nazad.
- Prilikom dobivanja povratnih rezultata potrebno je nadjačati onActivityResult metodu

Demo



Dohvat podataka iz imenika

finish();



.pitinja?

Android

Postavke, ListView i GridView kontrole

Sadržaj predavanja

- Postavke
- ListView i GridView općenito
- Data adapteri
- ListView kontrola
- GridView kontrola
- RecyclerView kontrola

Postavke aplikacije

- U svrhu postavki postoji Preference API
- Sučelje se definira kroz XML ili pomoću Java koda
- .xml datoteka se sprema u res/xml direktorij pod imenom preferences.xml jer aplikacija najčešće ima jedne postavke
- Glavni element xml-a mora biti

```
<PreferenceScreen  
    xmlns:android="http://schemas.android.com/apk/res/android">
```

- Preference API je deprecated od verzije 29
 - dodati u projekt AndroidX Preference Library

Elementi postavki

| Element (XML) | Opis |
|---------------------------|--|
| CheckBoxPreference | Prikazuje element s CheckBox elementom |
| ListPreference | Otvara dijalog s listom za odabir samo jednog elementa |
| EditTextPreference | Otvara dijalog sa EditText elementom |
| MultiSelectListPreference | Otvara dijalog sa listom za odabir više elemenata |
| SwitchPreference | Prikazuje element sa Switch tipkom |
| PreferenceCategory | Prikazuje naslov podgrupe u koju je onda moguće dodavanje pod elemenata |
| Preference | Za pokretanje nekog Intenta ili neke slične akcije |
| PreferenceScreen | Za kreiranje novog izbornika postavki koje se nalaze unutar tog elementa |

SharedPreferences

- Svakoj od postavki potrebno je postaviti atribute key, title i defaultValue
- Key je ključ pod kojim će se vrijednosti pohraniti u SharedPreferences objekt u kojem se pohranjuju sve postavke
- DefaultValue je vrijednost na koju će biti postavljena postavka
- Te iste postavke moguće je dodavati ručno kroz programski kod ;)

Demo



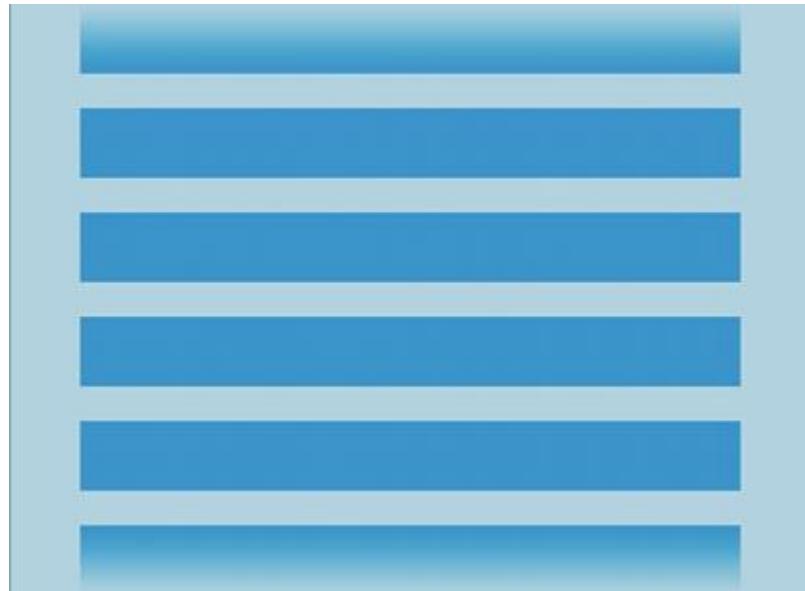
XML postavke

Dohvaćanje postavki

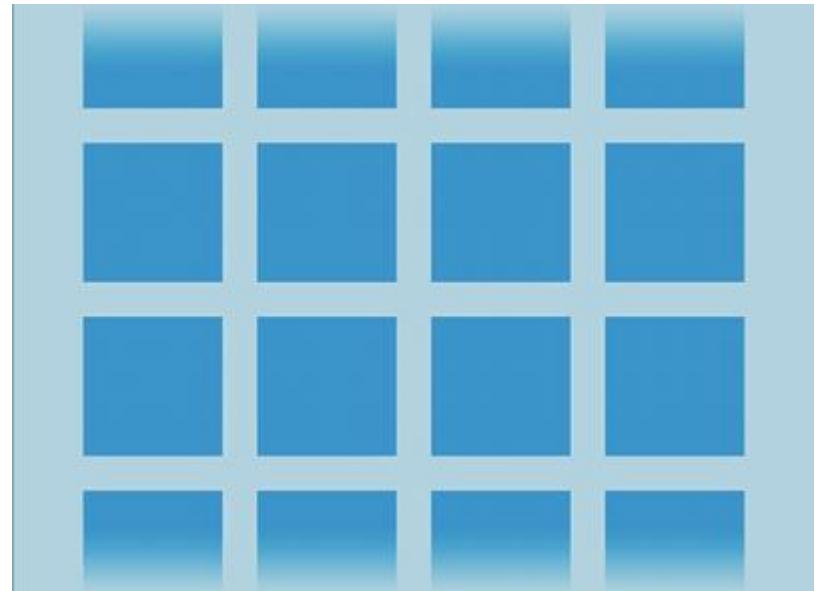
ListView i GridView

Izgled

ListView

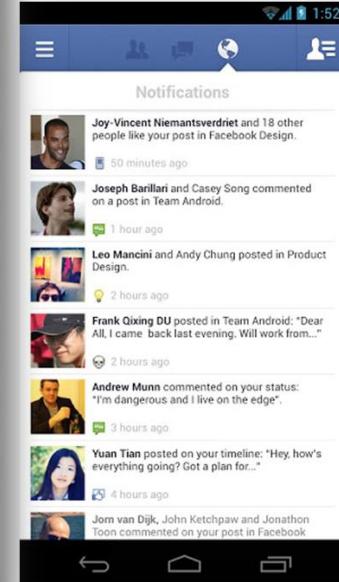
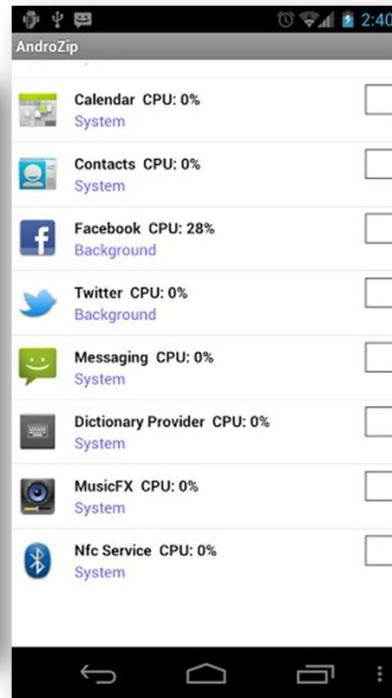
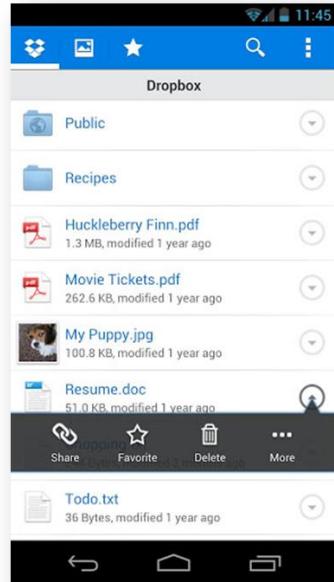
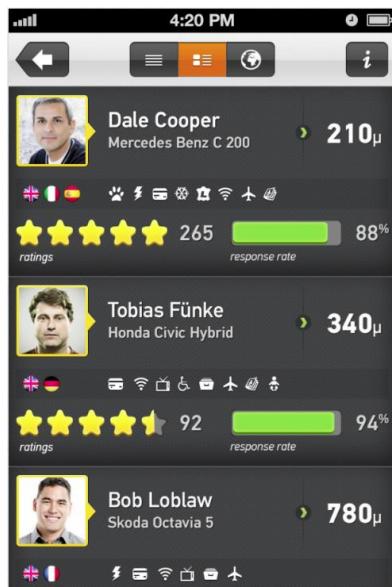


GridView



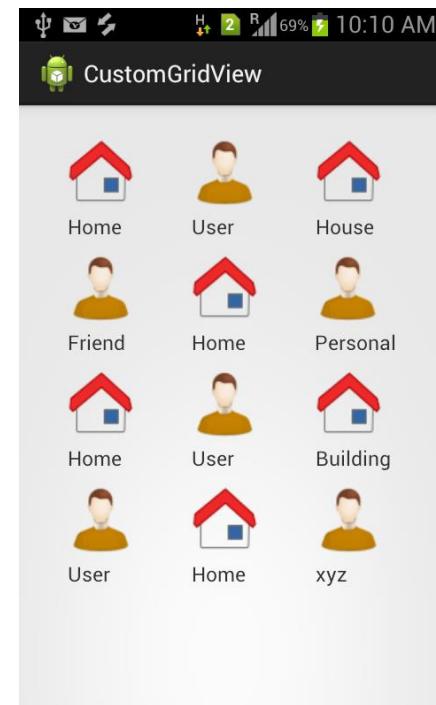
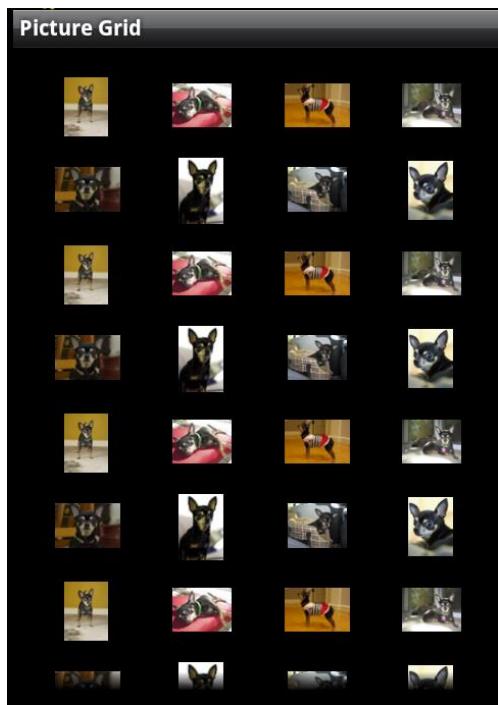
ListView

Izgled



GridView

Izgled



ListView i GridView

Sličnosti

• Nasljeđuju AdapterView klasu

- Osim GridView i ListView kontrola je nasljeđuju još i:
 - AdapterViewFlipper, ExpandableListView, Gallery, Spinner i StackView
 - Sve gore navedene kontrole funkcioniraju slično kao GridView i ListView

• Koriste se prilikom prikaza više podataka na ekranu
(Jednodimenzionalno ili dvodimenzionalno)

• Da bi prikazali podatke potreban im je Adapter

• Moguće ih je animirati

Adapter je?

- Adapter služi prikazivanju jednog podatka u listi
- Adapter je zapravo komponenta koja definira dizajn liste ili grida.
- Lista i Grid su na neki način okviri koji multipliciraju adapter
 - Npr. kao Repeater kontrola u ASP .NET-u

Adapter

Unaprijed pripremljeni adapteri

- Postoje unaprijed pripremljeni adapteri unutar samog framework-a Androida
- **ArrayAdapter, CursorAdapter, BaseAdapter i mnogi drugi**
 - Svaki od navedenih ima određeni dizajn kojeg je moguće donekle izmijeniti
 - (Više na <http://developer.android.com/reference/android/widget/Adapter.html>)

Demo



Izgled adaptera

Rad s adapterima

ListView

Značajke

- Pokazuje jednodimenzionalno podatke pomoću Adaptera
- Izuzetno se često koristi kada je korisniku potrebno prikazati veliku količinu podataka
- Moguće je koristiti ~~ListViewActivity~~ klasu
 - Deprecated

ListView

Značajke

- Svaki stavak ima svoj zasebni Layout

- Ugrađeni
 - Vlastiti

- Podatke i izgled same kontrole određuje adapter

- Svakoj listi potrebno je predati adapter

- Bez adaptera lista nema nikakvog smisla

Demo



ListView bez dodataka

ListView s pozivom aktivnosti

Lista kontakata

GridView

Značajke

- Pokazuje dvodimenzionalnu mrežu podataka koju je moguće pomicati (Engl. scroll)
- Najčešće se koristi prilikom prikaza slika (tu i tamo za prikaz osoba u imeniku)

GridView

Značajke

• Koristi identične adapttere kao i ListView

- ArrayAdapter – za nizove i liste
- SimpleAdapter – za adaptiranje malo komplikiranijih lista koristeći listu mapa za mapiranje
- SimpleCursorAdapter – za adaptiranje skupa podataka dobivenih upitom na bazu ili prema sistemu (npr. lista kontakata)
- Vlastiti adapteri (Nasljedivanje BaseAdapter klase)

Demo



GridView s običnim
adapterom

GridView s napravljenim
adapterom

Animacija GridView kontrole

RecyclerView

Prednosti

- Nasljednik ListView-a
- Ima manje obveza nego ListView
 - Ne pozicionira elemente na ekran
 - Ne animira View elemente
 - Ne obrađuje evente osim scrolling-a
- Jednostavnije je ponovo iskoristiti ćelije prilikom scroll-a
- Odvojiva lista od kontenjera
- Pojednostavnjuje animiranje
- Ima dosta gotovih biblioteka

RecyclerView

Kako radi

• Koristi layout manager

- LinearLayoutManager – vertikalni ili horizontalni scroll
- GridLayoutManager – kreira grid
- StaggeredGridLayoutManager – uređeni grid

• Adapter mora naslijediti RecyclerView.Adapter

• Animacije se mogu raditi nasljeđivanjem klase
RecyclerView.ItemAnimator

Demo



RecyclerView primjeri

finish();



.pitinja?

Android

Fragmenti

Sadržaj predavanja

- Što su fragmenti?
- Čemu služe?
- Izrada
- Korištenje na više aktivnosti
- Puno primjera ☺

Općenito

O fragmentima

- Fragmenti su kao „podaktivnosti“ aktivnosti
- Fragment predstavlja ponašanje ili dio korisničkog sučelja unutar Aktivnosti
- Moguće je kombinirati više fragmenata za jednu aktivnost kako bi se izradilo više panelno korisničko sučelje
- Moguće je isti fragment koristiti unutar više aktivnosti
- Fragment je modularan dio aktivnosti koji ima vlastiti životni ciklus, i prima svoje vlastite evente

Općenito

Još o fragmentima

• Fragment se mora nalaziti na aktivnosti

- Životni ciklus fragmenta direktno ovisi o životnom ciklusu aktivnosti
- Npr. Ako je aktivnost pauzirana i fragmenti su pauzirani
- Ako se aktivnost uništi, fragmenti se unište

• Ali, dok se aktivnost izvršava moguće je manipulirati svakim fragmentom zasebno

• Nasljeđuju AdapterView klasu

- Dodavati ih, uklanjati ih i općenito vršiti transakcije nad njima
- Dodavati ih na backstack (stack na aktivnosti) kojim upravlja aktivnost pomoću FragmentTransaction klase (back button radi)

Općenito

Još malo...

- Kada dodate fragment kao dio layouta aktivnosti on „živi” u ViewGroup-u (bazna klasa za layoute i kontejnere)
- Unutar ViewGrupe definira svoj layout
- Fragmente je moguće napraviti „statičkim” preko xml-a, a moguće ih je i dinamički dodavati kroz programski kod
- Fragment ne mora imati UI, već se može koristiti kao worker

Filozofija fragmenata

Zašto su nastali?

- Fragmenti su uvedeni s verzijom 3.0 (API 11)
- Tada je android počeo službeno podržavati tablete
- Potreba za fleksibilnijim korisničkim sučeljem
- Tableti imaju veći ekran, tako da ima više mesta
 - Tablet se češće drži vodoravno ;)

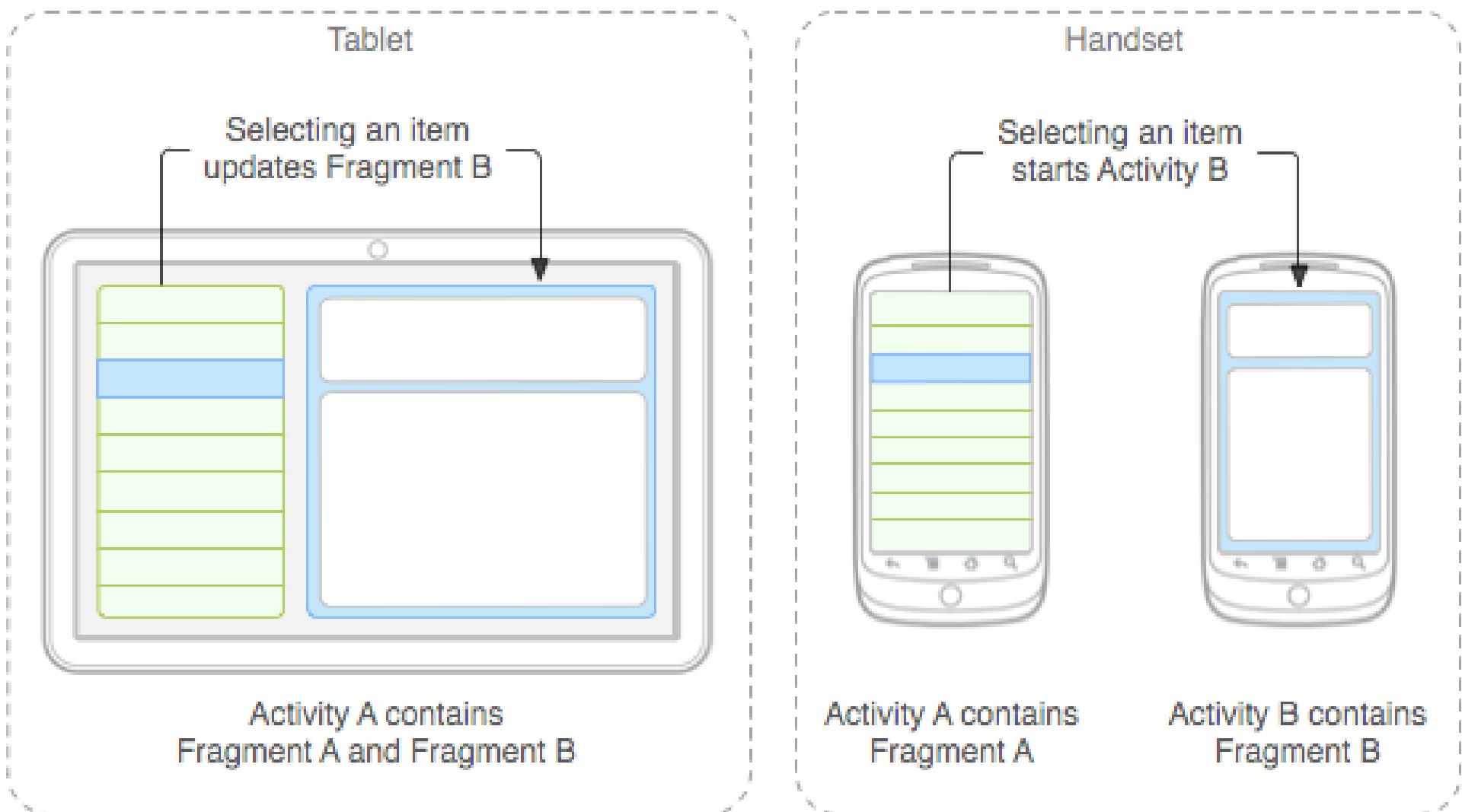
Filozofija fragmenata

Zašto ih koristiti?

- Fragmenti omogućuju takav dizajn bez velike potrebe za kompleksnim promjenama u hijerarhiji UI-a
- Potrebno je „samo“ UI podijeliti po fragmentima

Filozofija fragmenata

Grafički prikaz



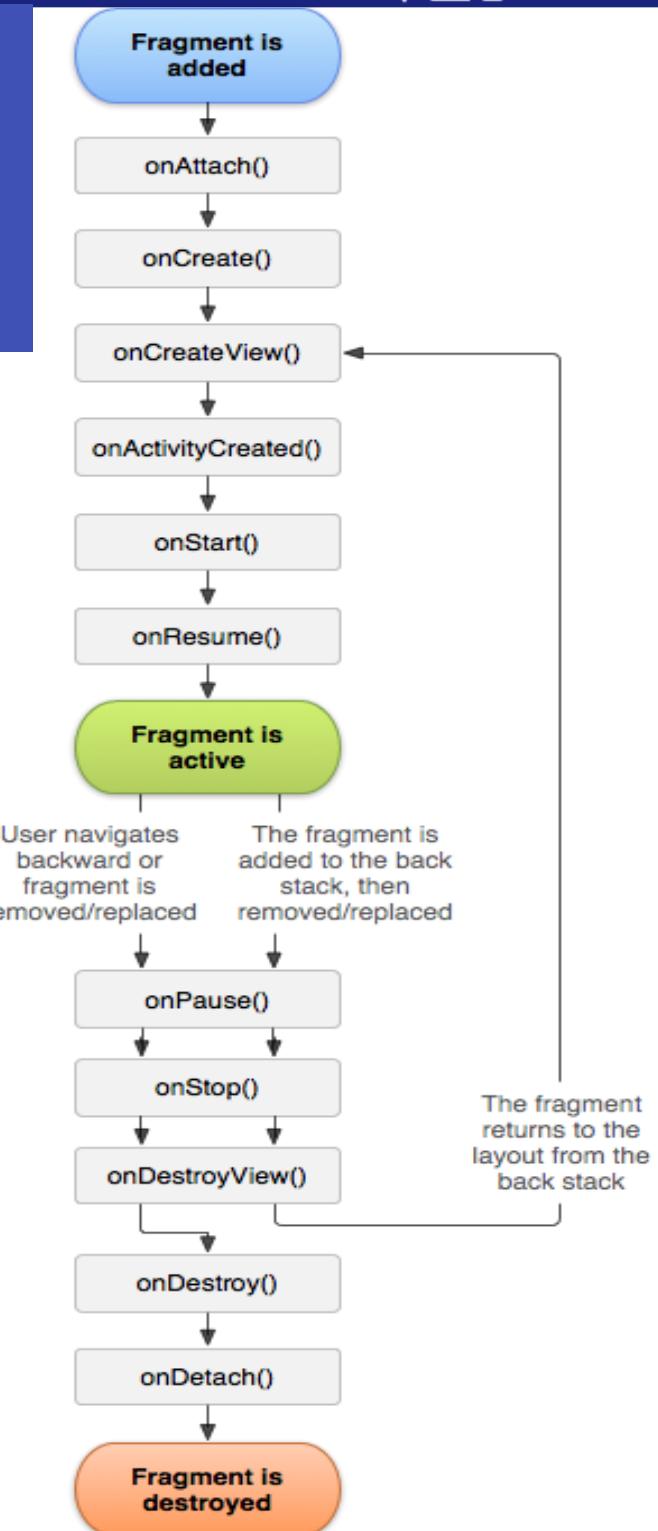
Kompatibilnost

Što s verzijama prije 3.0

- Budimo realni... I onako nećemo podržati ranije verzije ☺
- Postoji Android Support library
 - Moguće ga je dohvatiti kroz SDK Manager
- Prilikom korištenja support library-a potrebno je voditi računa o importima
 - klase se dohvaćaju iz support library-a
- Po novom su deprecated u androidu i potrebno se prebaciti na androidx ili compatibility biblioteke (api>28)

Životni ciklus fragmenta

- Vezan je za životni ciklus aktivnosti
- Malo drugačije metode nego kod aktivnosti



Metode

Prilikom paljenja

- `onAttach(Activity)` – poziva se kada je fragment asociran s aktivnosti
- `onCreate(Bundle)` – inicijalno kreiranje fragmenta
- `onCreateView(LayoutInflater, ViewGroup, Bundle)` – kreira i vraća view fragmenta
- `onActivityCreated(Bundle)` – aktivnost je izvršila onCreate()
- `onViewStateRestored(Bundle)` – pohranjeno stanje je vraćeno
- `onStart()` – fragment postaje vidljiv korisniku
- `onResume()` – fragment postaje interaktivan prema korisniku

Metode

Prilikom gašenja

`.onPause()` – fragment više nije interaktivan

`.onStop()` – fragment više nije vidljiv

`.onDestroyView()` – fragment treba počistiti resurse koji su vezani za View komponente

`.onDestroy()` – počistiti do kraja

`.onDetach()` – zove prije nego što fragment ne bude više povezan s aktivnosti

Demo



Životni ciklus fragmenata

Kreiranje fragmenata

Što je potrebno napraviti?

- Potrebno je naslijediti klasu Fragment ili neke od njenih derivata
 - DialogFragment, ListFragment, PreferanceFragment, WebViewFragment
- Unutar metode onCreateView pomoću LayoutInflater klase kreirati layout i zlijepiti ga na kontejner View

Kreiranje fragmenata

Dva načina kreiranja

- Unutar layouta aktivnosti moguće je kroz xml „zalijepiti“ fragment
- Moguće ga je zalijepiti dinamički unutar programskog koda (preporučljivo)
 - Vrši se pomoću FragmentManager klase
 - Pomoću transakcije prilikom koje je moguće dodati fragment, promijeniti fragment, dodati ga na backstack i slično

Demo



Fragment u XML-u

Fragment kroz kod

„Problemčići“ s XML-om

Problemi u raju (problema)

- Fragment koji je kreiran unutar XML-a je nemoguće zamijeniti drugim fragmentom
- Jedini način da se zamjeni fragment s fragmentom jest da ga se inicijalno dodaje dinamički!

Rad s fragmentima

Klasa FragmentManager

- `.getSupportFragmentManager()` iz aktivnosti
- Sadrži metode za dohvaćanje i postavljanje fragmenata
- Služi za dohvaćanje `FragmentTransaction` klase (`beginTransaction()`)
- Više na
<http://developer.android.com/reference/android/app/FragmentManager.html> i
<https://developer.android.com/reference/androidx/fragment/app/FragmentManager>

Rad s fragmentima

Transakcije

- Moguće je dodavati, premještati, uklanjati fragmente u jednoj transakciji
- Transakcija je set promjena koje će se izvršiti istovremeno
- Na kraju svake transakcije potrebno je pozvati commit() metodu da se promjene izvrše

Fragment & aktivnost

Komunikacija

- Fragment može do svoje aktivnosti doći pomoću metode `getActivity()`
- Aktivnost do fragmenta može doći pomoću metoda
 - `getSupportFragmentManager().findFragmentById()`
 - (Ako je dodan statički)
 - `getSupportFragmentManager().findFragmentByTag()`
 - (Ako je dodan dinamički)
 - `findViewById()`
 - Također funkcionira

Demo



Fragment mijenja aktivnost

Aktivnost mijenja fragment

Izmjena fragmenata

Programska izmjena

- Fragmente je moguće programski izmjenjivati i staviti na backstack kako bi radila „back“ tipka
- Jedini način da se zamjeni fragment s fragmentom jest da ga se inicialno dodaje dinamički!

Izmjena fragmenata

Transakcije

- Izmjena se također vrši pomoću FragmentManager-a i transakcija
- Ako se pozove metoda addToBackStack() onda će raditi tipka „back“

Demo



Lijepljene fragmenata na isto
mjesto i backstack

Najčešće korištenje

List - detail

- Fragmenti se najčešće koriste kada je potrebno na različitim uređajima različito prikazati podatke
- Na tabletu je zgodnije pored liste u landscape načinu rada odmah prikazati detalje
- Na mobilnom uređaju je zgodnije prikazati listu i na klik novu aktivnost s detaljima
- Ovdje na scenu stupaju fragmenti zbog čega su i zamišljeni

Najčešće korištenje

List - detail

- ListFragment klasa se koristi slično kao i ListActivity
- Ima ugrađene metode za rad s listama i Callback prema aktivnosti na kojoj se nalazi kako bi aktivnost preuzela većinu rada s podacima a Fragment samo prikazuje podatke

Najčešće korištenje

Tabovi, swipe, drawer

- Fragmenti se također koriste prilikom izrade tabova, swipe-a i drawer-a
- Većinu koda za tabbed izgled aktivnosti s fragmentima moguće je jednostavno kreirati (generirati) kroz Android Studio

Demo



List – detail

Tabovi

Swipe

Drawer

finish();



.pitinja?

Android

Notifikacije, servisi, push

Sadržaj predavanja

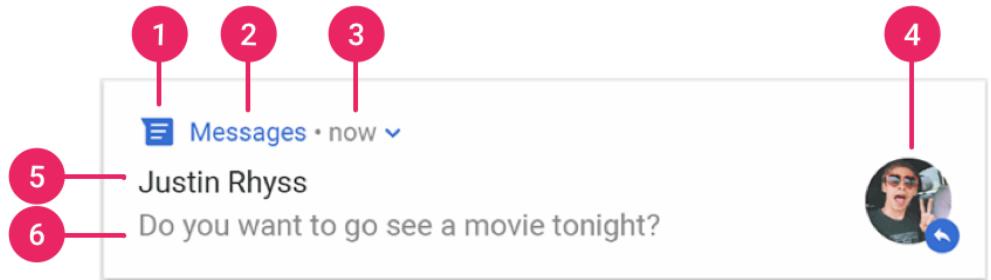
- Notifikacije
- Servisi
- Firebase push

Notifikacije

- Notifikacija je poruka koja se prikazuje izvan normalnog korisničkog sučelja aplikacije
- Prvo se prikazuje kao ikona u statusnoj traci koja se gestom može proširiti na cijeli ekran
- Postoje više vrsta notifikacija
 - Normalne veličine visine 24dp
 - Velika – od ver. Androida 4.1
 - LockScreen od Androida 5.0

Dijelovi notifikacije

1. Mala ikonica
2. Naziv aplikacije
3. Vrijeme
4. Velika ikonica
5. Naslov
6. Tekst



Stilovi notifikacija

.Velika notifikacija može biti prikazana u nekom od stilova:

- Big picture style
 - prostor s detaljima sadrži sliku visoku do 256 dp
- Big text style
 - prostor s detaljima sadrži veliki tekst
- Inbox style
 - prostor s detaljima sadrži linije nalik na inbox
- Media style
 - Stil media playera
- MessagingStyle
 - Dodavanje poruka

Stilovi notifikacija

• Svaki od stilova ima opcije:

- Big content title
 - dozvoljava postavljane posebnog naslova koji će biti prikazan samo kada se notifikacija ekspandira
- Summary text
 - dozvoljava dodatnu liniju teksta ispod prostora s detaljima

Kreiranje notifikacija

Od ver. Androida 8 (API >= 26) potrebno je kreirati kanal

Kanal služi tome da je moguće ugasiti notifikacije pojedine aplikacije



Kreiranje kanala

Kreiranje notifikacija

- Kreira se Builder patternom unutar Notification klase
- Metoda build() vraća objekt Notification koji se predaje NotificationManageru
- Svaka notifikacija mora imati:
 - Malu ikonu – setSmallIcon()
 - Naslov – setContentTitle()
 - Tekst – setContentText()
- Nakon što se notifikacija kreira ona se predaje NotificationManageru metodom notify(), koji notifikaciju objavljuje pod određenim identifikacijskim brojem. Ako broj već postoji, postojeća notifikacija se osvježava(update)

Progress notifikacija

- Od Android ver. 4.0 moguće je prikazivati progres notifikaciju
- To se obavlja metodom `setProgress()`
- U starijim verzijama je to bilo moguće napraviti pomoću korisničkog sučelja koje se implementiralo u notifikaciju
- Metodom `setProgress()` postavlja se maksimalna vrijednost, trenutna vrijednost i je li progressbar određen ili neodređen

Notifikacije



Osnovna notifikacija

Velike notifikacije

Progress notifikacija

Notifikacija s gumbima

Servisi

- .Servis je komponenta koja se koristi kada je potrebno izvoditi neke dugotrajne operacije u pozadini
- .Servisi ne omogućuju UI
- .Postoje tri tipa servisa:
 1. Foreground
 2. Backgdound
 3. Bound

Servisi

.Foreground

- Izvodi neku operaciju koje je vidljiva korisniku
- Primjerice *audio player* pokreće glazbu korisniku

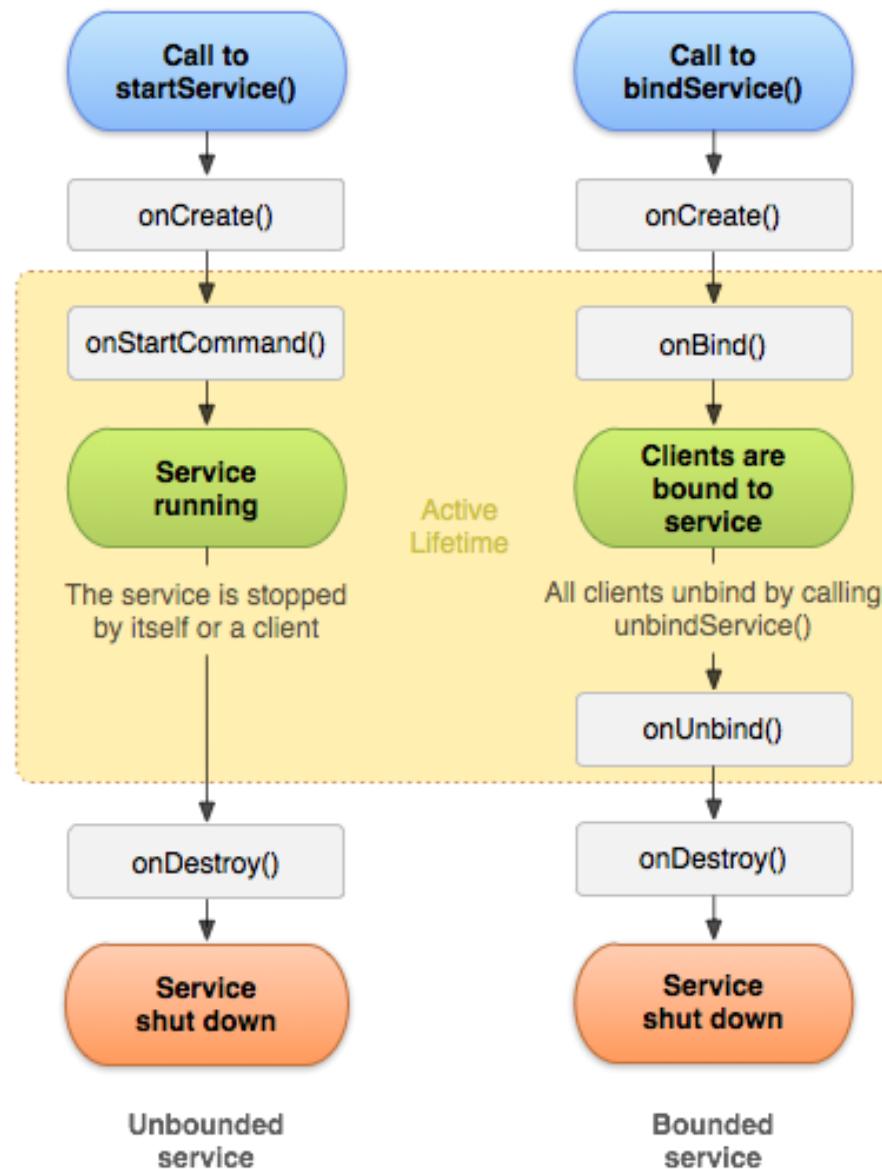
.Background

- Izvodi neku operaciju koja nije direktno vidljiva korisniku
- Primjerice pohrana slike

.Bound

- Servis je vezan kada se komponenta veže za njega pomoću „bindService()” metode
- Vezani servis omogućava komponentama interakciju sa servisom (slanje zahtjeva, preuzimanje rezultata, IPC (*interprocess communication*))

Životni ciklus servisa



Servisi

- Isti servis može raditi kao Bound i Started servis ukoliko se implementiraju obje metode
- Uvijek mora biti deklariran u manifestu
 - ```
<application ... >
 <service android:name=".ExampleService" />
</application>
```
- „po defaultu“ ne pokreće novu dretvu već se vrati u host dretvi
- Servis nije odvojeni proces ako nije drugačije specificirano

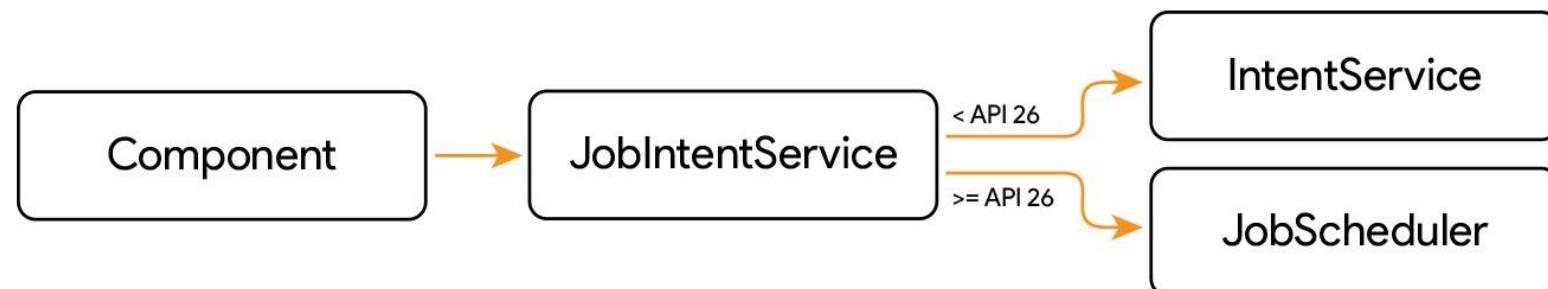
# Servisi

• Klasa Service je bazna klasa svakog servisa

- Traži override svih metoda
- Briga o zasebnoj dretvi

• Klasa JobIntentService je omotač oko Service klase

- Kreira worker dretvu
- Kreira work queue
- Stopira servis kada završi radnju
- Implementira onHandleWork()



# Bound servisi

• Moraju omogućiti Ibinder *interface* kojeg klijenti koriste za interakciju sa servisom

- Ako je servis privatni na razini aplikacije potrebno je vratiti instancu Binder klase i vratiti je kao *interface* kroz onBind() metodu
- Ako je servis javan među procesima tada je potrebno koristiti instancu Messenger klase
- AIDL (Android Interface Definition Language)
  - Najčešće nije potreban
  - Messenger je baziran na AIDL-u
  - Potrebno je samostalno implementirati *multithreading* i *thread safety*

# Servisi



Klasični servis

Bound servis kao  
generator brojeva

# Push

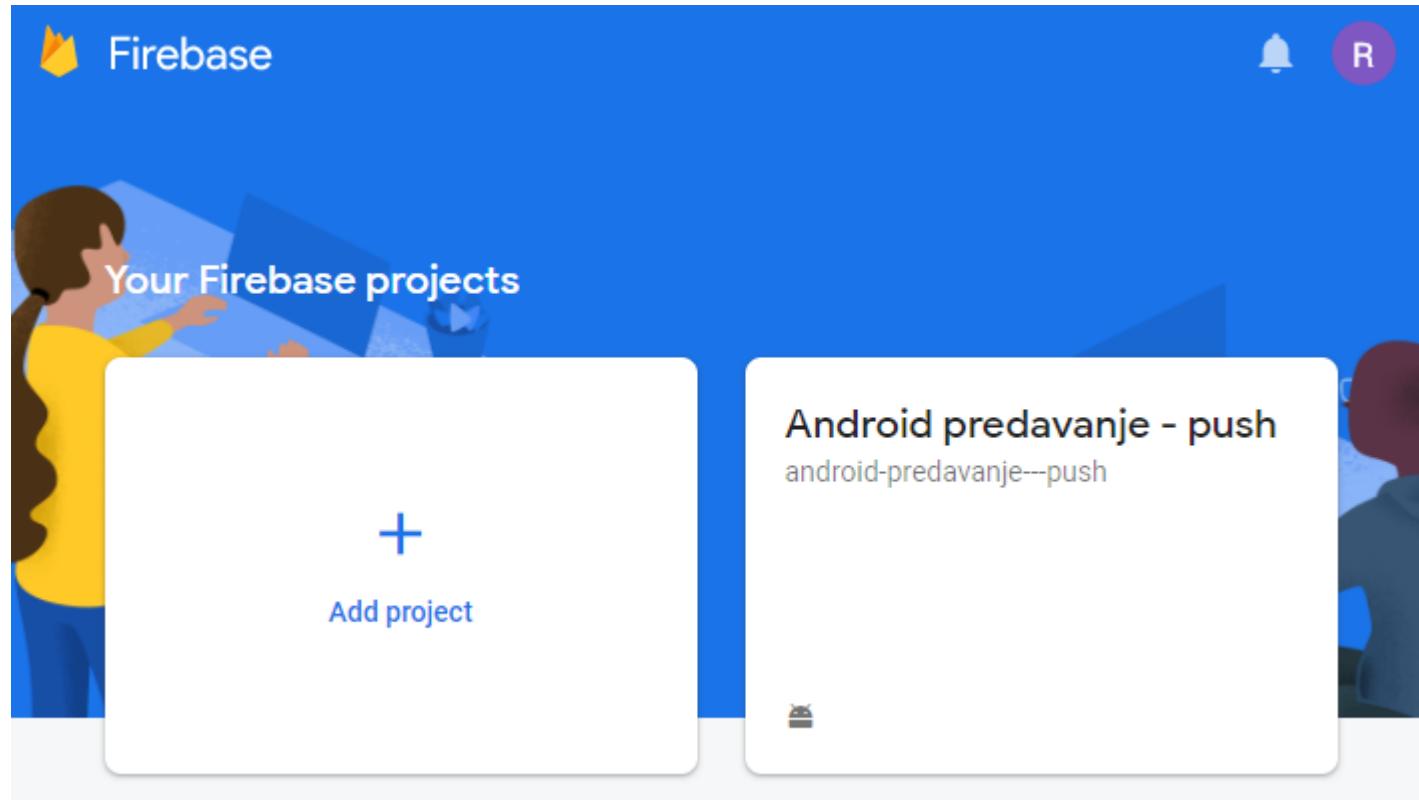
- Push poruka je bilo koja poruka koja se prikazuje dok aplikacija nije trenutno u upotrebi
- Postoji mnogo servisa i oblaka koji na jednostavan način omogućuju takvo ponašanje
  - Amazon AWS SNS (Amazon web services simple notification service)
  - Google Firebase
  - Cloud messaging (depricated)
  - Vlastita implementacija preko servisa
  - ...
- Mi ćemo koristiti Firebase ☺

# Push

Otvaranje Firebase projekta

Prvo je potrebno kreirati račun na [firebase.google.com](https://firebase.google.com)

Na konzoli ([console.firebaseio.google.com](https://console.firebaseio.google.com)) kreirati novi projekt

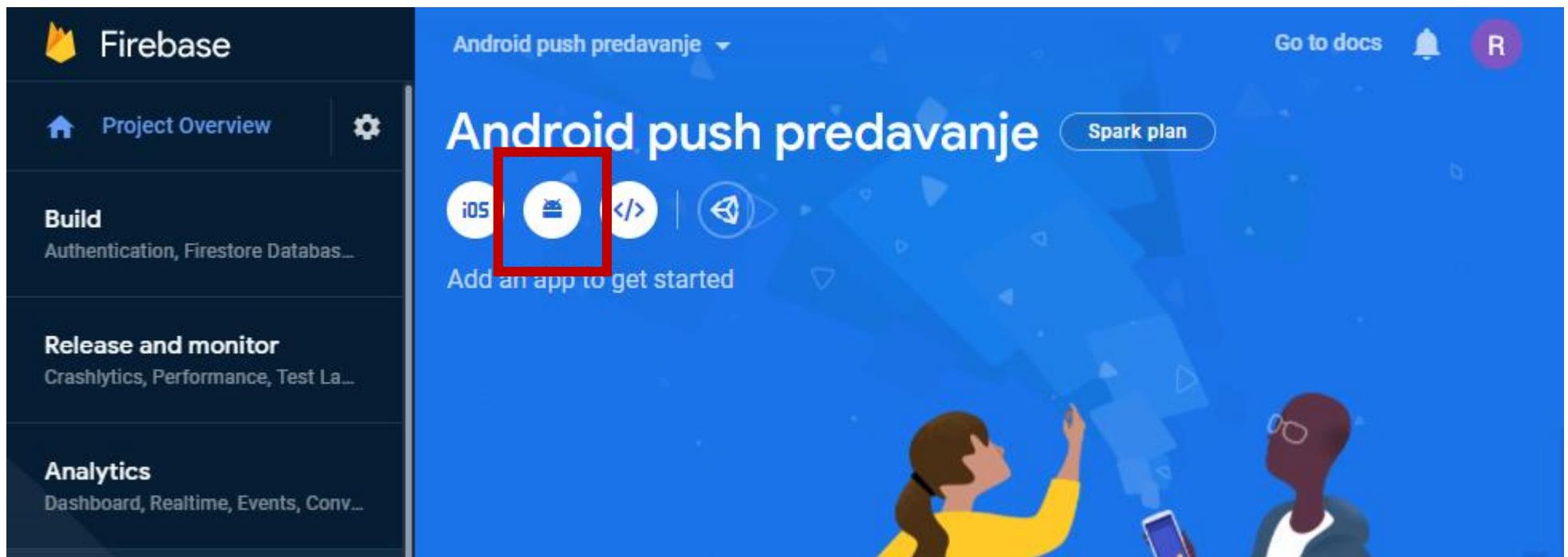


# Push

Otvaranje Firebase projekta

Dodati Firebase android aplikaciji

Klik na projekt -> Add Firebase to your Android app



# Push

## Otvaranje Firebase projekta

### × Add Firebase to your Android app

#### 1 Register app

Android package name ?

App nickname (optional) ?

Debug signing certificate SHA-1 (optional) ?

Required for Dynamic Links, Invites, and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.

**REGISTER APP**

2 Download config file

3 Add Firebase SDK

4 Run your app to verify installation

### × Add Firebase to your Android app

#### 1 Register app

Android package name ?

App nickname (optional) ?

Debug signing certificate SHA-1 (optional) ?

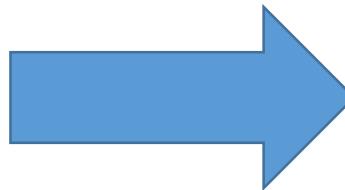
Required for Dynamic Links, Invites, and Google Sign-In or phone number support in Auth. Edit SHA-1s in Settings.

**REGISTER APP**

2 Download config file

3 Add Firebase SDK

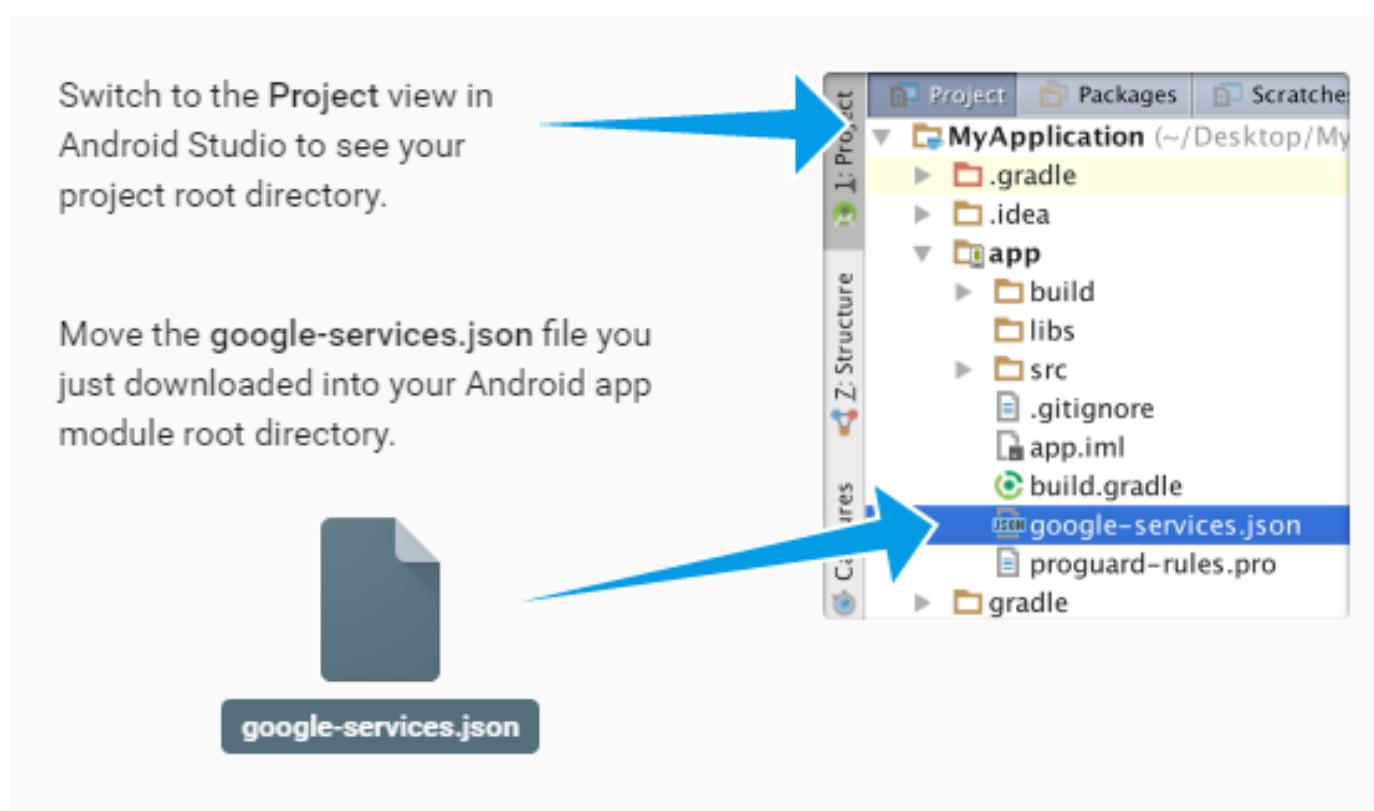
4 Run your app to verify installation



# Push

Otvaranje Firebase projekta

Preuzeti json konfiguracijsku datoteku i staviti je kao na slici:



# Push

Otvaranje Firebase projekta

Dodati plugin za google services:

3 Add Firebase SDK Instructions for Gradle | [Unity](#) [C++](#)

The Google services plugin for [Gradle](#) loads the google-services.json file that you just downloaded. Modify your build.gradle files to use the plugin.

Project-level build.gradle (<project>/build.gradle):

```
buildscript {
 repositories {
 // Check that you have the following line (if not, add it):
 google() // Google's Maven repository
 }
 dependencies {
 ...
 // Add this line
 classpath 'com.google.gms:google-services:4.3.3'
 }

 allprojects {
 ...
 repositories {
 // Check that you have the following line (if not, add it):
 google() // Google's Maven repository
 }
 }
}
```

App-level build.gradle (<project>/<app-module>/build.gradle):

```
apply plugin: 'com.android.application'
// Add this line
apply plugin: 'com.google.gms.google-services'

dependencies {
 // add SDKs for desired Firebase products
 // https://firebase.google.com/docs/android/setup#available-libraries
}
```

Finally, press 'Sync now' in the bar that appears in the IDE:

Gradle files have changed since last sync [Sync now](#)

# Push

Otvaranje Firebase projekta

• U build.gradle (app) dodati dependencies:

```
implementation 'com.google.firebaseio:firebase-analytics'
implementation 'com.google.firebaseio:firebase-messaging'
```

• Ovaj dio neće raditi ukoliko se kroz SDK manager ne preuzme najnoviji Google repozitorij i play services  
Prijevod: neće raditi na genymotionu!

Ponaša se tako da ne baci nikakvu grešku, samo ne radi

# Push

Otvaranje Firebase projekta

- Nadjačati „FirebaseMessagingService“ i u njemu slušati poruke i raditi s tokenom ☺

# Push

Nakon pokretanja

Kada se aplikacija prvi puta pokrene (zafrkava u emulatoru) Trebala bi se ispisati poruka na firebaseu

4 Run your app to verify installation

 Congratulations, you've successfully added Firebase to your app!

[PREVIOUS](#) [CONTINUE TO CONSOLE](#)

# Push

Slanje poruka kroz cloud sučelje

Screenshot of the Firebase Cloud Messaging interface.

The interface includes:

- Left sidebar:** Lists various Firebase services: Cohorts, StreamView, Latest Release, DebugView, User Properties, GROW, Predictions, Cloud Messaging (selected), Remote Config, Dynamic Links, and AdMob.
- Top navigation:** Shows "Super projekt" dropdown, "Go to docs", a notification bell, and user profile.
- Header:** "Cloud Messaging" title and "A/B Testing" button.
- Illustration:** An illustration of a woman with curly hair wearing yellow headphones, holding a smartphone, with a speech bubble above her.
- Text:** "Manage notification campaigns and send messages to engage the right users at the right moment".
- Links:** "Learn more" and "View the docs".
- Call-to-action button:** "SEND YOUR FIRST MESSAGE".
- Bottom left:** "Spark" plan information ("Free \$0/month") and an "UPGRADE" button.

# Push



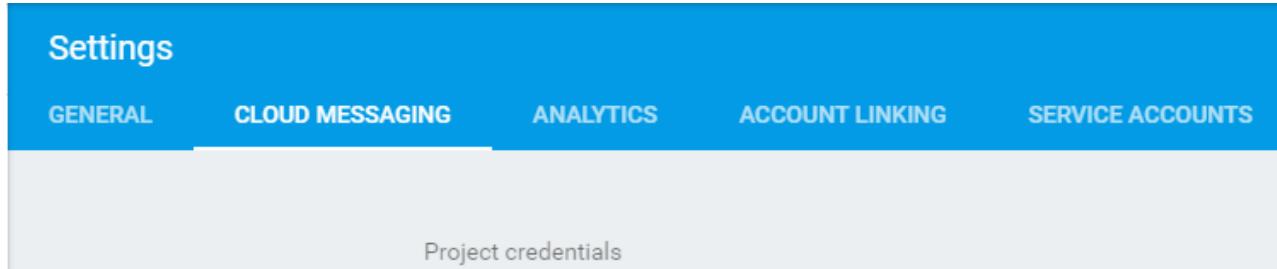
Push poruka preko  
firebase-a

# Push

Otvaranje Firebase projekta

Moguće je poslati poruku na samo jedan uređaj s vanjskog servera

Potrebno je slati poruke na  
<https://fcm.googleapis.com/fcm/send> koristeći api ključ koji se nalazi na cloud messaging tabu konzole



Sve mogućnosti moguće je pronaći na  
[firebase.google.com/docs/cloud-messaging/server](https://firebase.google.com/docs/cloud-messaging/server)

finish();



.pitinja?

# Android

Baze podataka

Threading, SQLite, DbFlow

# Sadržaj predavanja

## *.Multithreading*

- Što raditi, a što izbjegavati
- Tehnike multithredinga na Androidu

## **.SQLite**

- Kreiranje baze podataka
- Upiti
- DBflow

# Osnovni pojmovi

## Dretva

- Izvršavaju se unutar procesa (kod većine OS-a)
- Dijele resurse (Memorija tj. adresni prostor)
  - Istoj varijabli mogu pristupiti različite dretve
- Imaju zaseban povratni stog (engl. stack)
- Na sustavu s jednim procesorom događa se *time-division multiplexing*
  - Prebacivanje s jedne na drugu dretvu
  - Korisnik ima dojam da se operacije izvode simultano
- Na višeprocesorskim sustavima dretve se izvode simultano ovisno o broju procesora / jezgara

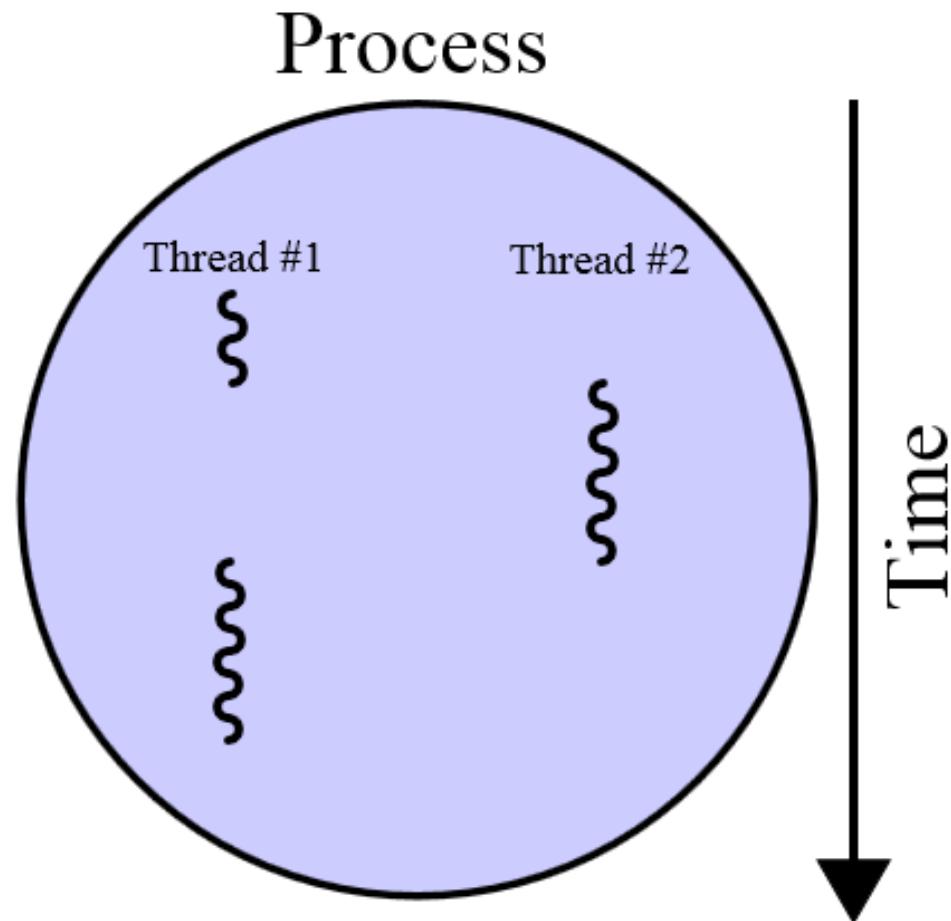
# Osnovni pojmovi

## Proces

- Instanca programa koji se izvodi
- Jedan program može imati više instanci
- OS brine da procesi imaju odvojene resurse (nemoguće je pristupiti varijabli drugog procesa iz prvog procesa)

# Osnovni pojmovi

Proces vs. Dretva



# Višedretvenost na Androidu

# Višedretvenost

Moguće je koristiti Klasičan Java kod kako bi osvježili UI iz druge dretve:



```
public void onClick(View v) {
 new Thread(new Runnable() {
 public void run() {
 Bitmap b = loadImageFromNetwork(
 "http://example.com/image.png");
 mImageView.setImageBitmap(b);
 }
 }).start();
}
```

NE MODIFICIRAJTE UI IZ DRUGE DRETVE!!!

# Višedretvenost

Iz druge dretve UI je potrebno osvježavati na sljedeći način:

```
public void onClick(View v) {
 new Thread(new Runnable() {
 public void run() {
 final Bitmap bitmap = loadImageFromNetwork(
 "http://example.com/image.png");
 mImageView.post(new Runnable() {
 public void run() {
 mImageView.setImageBitmap(bitmap);
 }
 });
 }
 }).start();
}
```

# Višedretvenost

Dva pravila

**.NIKADA SE NE SMIJE BLOKIRATI UI DRETVA**

- Android nakon 5 sekundi gasi aplikaciju

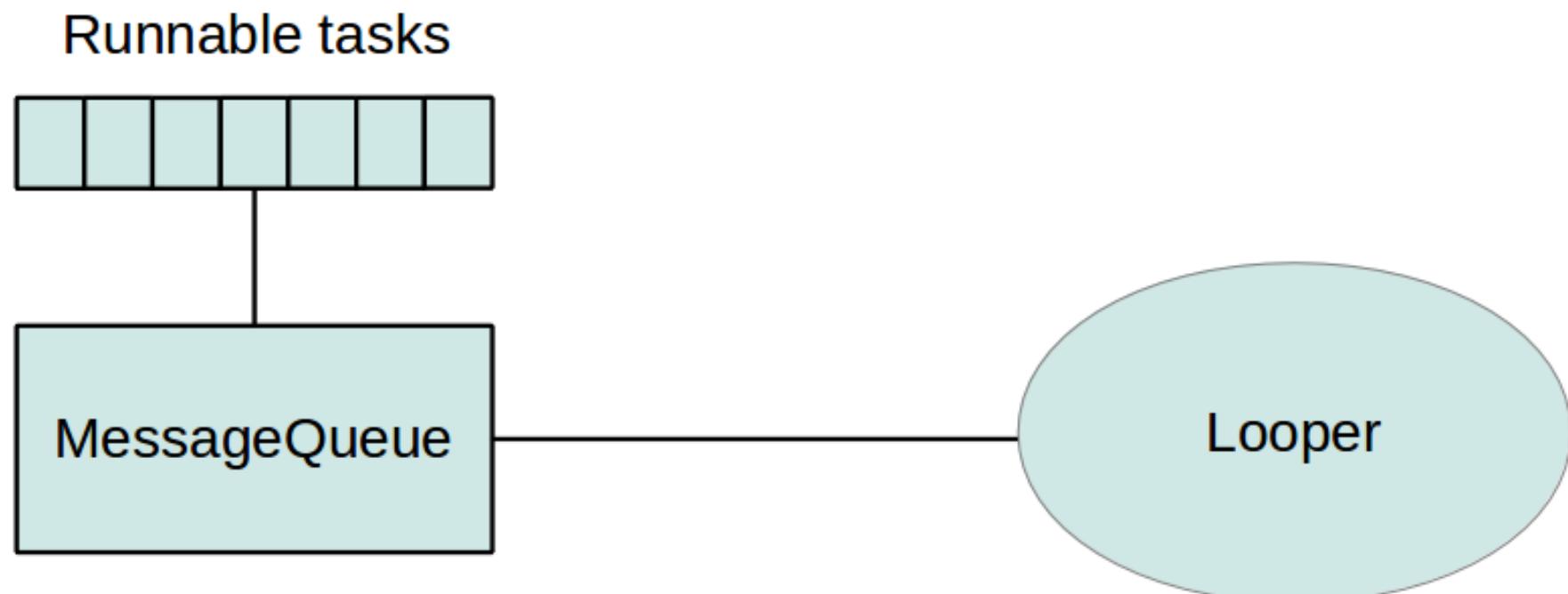
**.NIKADA SE NE SMIJE PRISTUPITI UI DRETVI IZ DRUGE DRETVE DIREKTNO**

- Može se dogoditi neregularno i nasumično ponašanje koje je jako teško pronaći i ukloniti

# A zašto?

Zbog *loopera*

- Android vrti jednu glavnu UI dretvu
- Unutar nje postoji Looper koji izvršava zadatke iz message queue-a



# Pojašnjenje

Zašto baš dretve?

- Ako se program piše u istoj dretvi, naredbe se izvršavaju jedna nakon druge
- Ako je radnja dulja, aplikacija se blokira dok god se radnja ne izvrši (prošli slajd)
- Kako bi korisnikov doživljaj aplikacije bio što bolji, blokiranje korisničkog sučelja nije dobrodošlo

# Demo



Ponašanje aplikacije sa i  
bez dretve (C#)

# Što koristiti na Androidu?

- Android posjeduje iste klase za dretve kao i Java +
- Handler
- Headless fragmenti (fragmenti bez GUI-a)
- Loader
- Servisi
- Cache
- LiveData...

# Klasične dretve

(Kao u Javi)

- Rade identično kao u javi
- Moguće je koristiti Thread, ThreadPools i Executor
- Nedostaci:
  - Sinkronizacija s glavnom UI dretvom
  - Nema automatskog zaustavljanja dretve
  - Nema automatskog thread poolinga
  - Nema automatskog praćenja promjena stanja Androida

# Klasične dretve



Kada će se raspasti

Korištenje post metode

# Handler klasa

(Ubacivač poruka u red poruka)

- Koristi se kao pomoćna klasa za ubacivanje poruka u message queue iz druge dretve
- Dretva i dalje postoji, a Handler samo pomaže oko sinkronizacije s GUI-jem
- Moguće je koristiti na više načina
  - Preko Bundlea dodavati poruke
  - Pomoću post metode (slično kao kod obične dretve)

# Handler



Načini korištenja Handler klase

# Loader

(Prati stanja)

- Olakšava asinkronizaciju na aktivnosti i fragmentu
- Automatski se spajaju na aktivnost prilikom rotacije
- Omogućavaju praćenje novih rezultata kada se promjeni sadržaj
- Užasno komplikiran za korištenje

# Loader



Primjer dretve s Loader  
klasom

# SQLite

- SQLite je relacijska baza podataka
- Posebna je po tome što je idealna za embedded sustave (ne vrti zaseban proces)
- Ponaša se kao library (Zapravo i je library)
- Napisan u c-u, wrappan u Javu
- Radi pomoću SQL jezika

# SQLite

Na Androidu

- Svaka aplikacija ima svoju bazu
- Baze aplikacija se nalaze u data/data direktoriju
- Nevjerojatno, ali moguće im je pristupiti preko ADB-a (na emulatoru)
- Također ih je moguće kopirati i onda raditi s njima na računalu

# SQLite

## Hacking the SQLite

• Vezanje na uređaj:

- adb –e shell

• Prelazak u direktorij i kreiranje novog:

- cd /data/data
- cd hr.android.primjeri/
- cd databases

• Igranje s bazom:

- sqlite3 ./naziv\_baze

• Moguće je kreirati i vlastitu bazu

# Baze iz Android aplikacije

• Zato što su baze wrappane u java kod, to znači da im je moguće pristupiti bez nekih većih poteškoća

## • SQLiteOpenHelper

- Sadrži gotove metode za rad s bazom
- Nju je potrebno naslijediti za lakši rad

## • Cursor

- Klasa za pristup resultsetu (Kod select naredbe npr.)

## • SQLiteDatabase

- Klasa koja služi kao interface između programskog koda i baze
- Uključuje funkcije za SQL operacije poput selecta, inserta i deletea

## • Model klase

# Baze iz Android aplikacije

(Divota u Javi ☺)

- Za insert, update i delete postoje metode koje generiraju upit
- Moguće je izvršavati praktički bilo koji SQL upit
- Idealno bi bilo da se rad s bazama izvodi u zasebnoj dretvi kako se aplikacija ne bi „smrznula”
- Potrebno je naslijediti SQLiteOpenHelper u kojoj se nalaze metode za pristup bazi i rad s bazom
- Postoji i odličan framework za rad s bazama - Dbflow

# Baze iz Android aplikacije



Izrada nove baze

Kreiranje upita

# ORM

Object-relational mapper

- Tehnika pretvaranja podataka između nekompatibilnih tipova u objektno orijentiranim jezicima
- Kreira se virtualna objektna baza podataka
- Npr:

```
String sql = "SELECT ... FROM persons WHERE id = 10";
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["FIRST_NAME"];
```

Prevedeno u OOP:

```
Person p = repository.GetPerson(10);
String name = p.getFirstName();
```

# DBFlow

Removing the ugliness

- Većina ORM (object-relation mapping) biblioteka koriste Java reflection kako bi definirali model, tablice i relacije
- DBFlow je jedan od rijetkih koji koristi procesiranje anotacija da bi generirao Java programski kod
  - Bolje performanse

# DBFlow

## Setuping root build.gradle

```
buildscript {
 repositories {
 jcenter()
 google()
 }
 dependencies {
 classpath 'com.android.tools.build:gradle:4.1.2'
 }
}
allprojects {
 repositories {
 jcenter()
 google()
 //DBFlow je samo na githubu
 maven { url "https://www.jitpack.io" }
 }
}
```

# DBFlow

## Setuping app build.gradle

```
//Dodati varijablu da bude lakše verzioniranje (sve biblioteke moraju biti
iste verzije!!!
def dbflow_version = "4.2.4" //Varijabla za verziju da bude na jednom
mjestu ;)

//Dodati ovisnosti, tj. Dependency-je
dependencies {
 compile fileTree(dir: 'libs', include: ['*.jar'])

 annotationProcessor "com.github.Raizlabs.DBFlow:dbflow-
processor:${dbflow_version}"
 compile "com.github.Raizlabs.DBFlow:dbflow-core:${dbflow_version}"
 compile "com.github.Raizlabs.DBFlow:dbflow:${dbflow_version}"
}
```

# DBFlow

## Setuping database

```
@Database(version = BazaPodataka.VERSION)
public class BazaPodataka {
 public static final int VERSION = 1;
}
```

# DBFlow

## Setuping table

```
@Table(database = BazaPodataka.class)
public class ToDo extends BaseModel {

 @Column
 @PrimaryKey(autoincrement = true)
 int id;

 @Column
 String title;

 @Column
 String note;
 //Geters & setters
}
```

# DBFlow

That's it

Studenti: Ček, samo to je potrebno podešiti?

Ja: Da.

# DBFlow



## Kreiranje upita

finish();



.pitinja?

# Rad s web servisima

ASP .NET WEB API, autentikacija, autorizacija, Fresco, Retrofit

Kako raditi s web servisima na Androidu

# Razvojno okruženje

- Windows OS (7, 8, 8.1, 10)

Linux, Mac OS - .NET core

<https://docs.asp.net/en/latest/tutorials/your-first-mac-aspnet.html>

- SQL Server express 2008 R2 na dalje

- Linux, Mac OS - SQL Server, Snalaženje s PostgreSQL || MySQL

- Visual studio 2017+

# C# - Solution i projekti

Why do Java Programmers wear glasses? Because they don't C#.

• Aplikacija se često dijeli na nekoliko slojeva

• **Solution** – ukupno rješenje

• **Projekt** – sloj aplikacije

# C# vs Java

- C# ima svojstva (properties) i događaje (evente)
- Korištenje refleksije nad generičkim klasama
- Jednostavnije iznimke (upitno je li dobro ili loše)
- Java je neovisna o OS-u
- C# ima dinamičke varijable
- Bolja enumeracija (yield)
- Generalno, C# je moderniji jezik od Java

# C#

## Svojstva (properties)

```
public class Student
{
 //Primativni tip
 private string _ime;

 //Objekt
 private String _prezime;

 public String Ime //Property
 {
 get { return this._ime; }
 set { this._ime = value; }
 }
 public string Prezime //Property
 {
 get { return this._prezime; }
 set { this._prezime = value; }
 }
 public string Oib { get; set; } //Brže pisanje propertija (auto property)
}
```

# C# vs. Java

```
public class Student
{
 private string _ime;
 private String _prezime;

 public String Ime
 {
 get { return this._ime; }
 set { this._ime = value; }
 }
 public string Prezime
 {
 get { return this._prezime; }
 set { this._prezime = value; }
 }
 public string Oib { get; set; }
}
```



```
public class Student
{
 private String _ime;
 private String _prezime;

 public String getIme()
 {
 return this._ime;
 }
 public void setIme(String Ime)
 {
 this._ime = Ime;
 }
 public String getPrezime()
 {
 return this._prezime;
 }
 public void setPrezime(String Prezime)
 {
 this._prezime = Prezime;
 }
 //Nema ekvivalenta
}
```

# C# vs. Java

## Dohvaćanje svojstava

```
Student stud = new Student();
stud.Ime = "Neko ime";
string ime = stud.Ime;
```

# C# - napredniji koncepti

LINQ i lambda izrazi

- Slične kao u Javi
- Anonimne ili inline funkcije
- Najčešće se koriste na kolekcijama

- Filtriranje
- Sortiranje
- Join

# C# - napredniji koncepti

LINQ i lambda izrazi - primjeri

• Potrebno je ispisati sve objekte koji imaju id < 3

p => p.Id < 3

• Ekvivalent:

```
p =>
{
 if (p.Id < 3)
 return true;
 return false;
}
```

# C# - napredniji koncepti

LINQ i lambda izrazi - primjeri

• Potrebno je ispisati sve objekte koji imaju id < 3

```
var novaLista = lista.Where(p => p.Id < 3)
```

• Potrebno je ispisati sve objekte koji imaju id < 3 i sortirati listu po koloni id

```
var novaLista = lista.Where(p => p.Id <
3).OrderBy(p => p.Id);
```

# Entity framework

• C# ima nekoliko načina za pristup bazi

- ADO.NET adapter
- SqlConnection
- LINQtoSQL
- Entity framework

• Entity framework je najnoviji i najčešće se koristi u velikim web aplikacijama

- Ne znači da je najbolji ☺

# Entity framework

## Konfiguracija

- Instalirati s nuget-a paket EntityFramework preko package manager konzole

```
Install-Package EntityFramework
```

- U **web.config** datoteku **web projekta** potrebno je dodati odgovarajuće podatke za spajanje

```
<connectionStrings>
 <add name="MojDbContext" connectionString="Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\Companies
Db.mdf;Integrated Security=True" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

# Entity framework

## Konfiguracija

- Kreirati klasu/e modela (tablica)

```
namespace MojWebApi.Models
{
 public class ToDo
 {
 [Key]
 [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
 public int Id { get; set; }
 [Required]
 public string Note { get; set; }
 [Required]
 public string Title { get; set; }
 }
}
```

# Entity framework

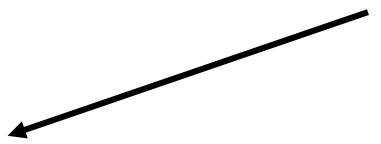
## Konfiguracija

- Kreirati klasu konteksta baze podataka

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Naziv isti kao i u web.config datoteci (zbog jednostavnosti)!

```
namespace MojWebApi.Models
{
 public class MojDbContext : DbContext
 {
 }
}
```



# Entity framework

## Konfiguracija

• Dodati klasu modela u DbContext:

```
public class MojDbContext : DbContext
{
 public DbSet<ToDo> Todos { get; set; }
}
```

# Entity framework

## Konfiguracija

- Uključiti migraciju na bazu preko package manager konzole  
`enable-migrations`
- Dodati migraciju nakon svake promjene u modelu na bazu  
`add-migration naziv_migracije`
- Spustiti migraciju na bazu podataka  
`update-database`

# Entity framework

## Dohvaćanje

- Dohvat svih vrijednosti:

```
MojaBaza db = new MojaBaza();
db.Todos;
```

- Dohvat jedne vrijednosti:

```
MojaBaza db = new MojaBaza();
db.Todos.Find(id);
```

# Entity framework

Dodavanje, brisanje

• Brisanje vrijednosti:

```
MojaBaza db = new MojaBaza();
ToDo todo = db.Todos.Find(id);
db.Todos.Remove(todo);
db.SaveChanges();
```

• Dodavanje vrijednosti:

```
MojaBaza db = new MojaBaza();
db.Todos.Add(todo);
db.SaveChanges();
```

# Web API

Općenito

- ASP.NET WebApi – rad isključivo s podacima (JSON, XML)
- ASP.NET MVC – generira HTML stranice (najčešće)
  - API projekt je moguće imati uz MVC projekt
- Najčešće se koristi u tri scenarija:
  - Javascript zove API i prikazuje podatke korisnicima
  - Server komunicira s drugim serverom putem API metoda
  - Mobilni uređaj ili desktop aplikacija komunicira s API servisima

# Web API

## Način rada

• U direktoriju controllers potrebno je kreirati klasu NazivController koja nasljeđuje ApiController

• Moguće je atributima podesiti na što se metoda javlja

• Sve metode rade preko http-a i poziva funkcija

- **GET** – dohvati podataka
- **POST** – unos podataka
- **PUT** – izmjena podataka
- **DELETE** – brisanje podataka

# Web API



Osnovne funkcionalnosti

# Autentikacija

... na poslužitelju

- Koristi se kada je potrebno otkriti tko pristupa informacijama
- Zapravo se radi o „loginu“ na stranice kako bi se potvrdio identitet klijenta
  - Korisničko ime i lozinka
  - Token
  - Raspoznavanje glasa
  - Kartice
  - Otisak prsta

# Autentikacija

... na klijentu

- Autentikacija se koristi na klijentu kada je potrebno znati je li server onaj koji tvrdi da jest.
- Najčešće se server autenticira izdavanjem vjerodostojnog certifikata klijentu

# Autentikacija

Zapamtite

- Autentikacija ne određuje koje radnje korisnik može napraviti, koje datoteke vidjeti ili koje servise zvati, već samo identificira i verificira osobu!
- Za gore navedeno odgovorna je autorizacija

# Tipovi autentikacije

- HTTP Basic authentication
- Cookies
- Tokens
- Signatures
- One-Time Passwords

# Autorizacija

- Proces kojim server određuje ima li klijent dozvole za neku radnju ili resurs
- Najčešće je vezana uz autentikaciju kako bi server imao neke informacije o korisniku
- U nekim slučajevima ne postoji autorizacija (većina resursa na internetu)

# OAuth 2.0

- OAuth 2.0 je framework koji se koristi prilikom autorizacije za web aplikacije, desktop aplikacije i mobilnih uređaja
- Koristi se kako bi se omogućio pristup preko HTTP servisa kao npr. Facebook, GitHub i sl.
- Može delegirati autentikaciju na servis koji drži korisnički račun (Facebook)

# Enkripcija

A što je onda enkripcija?

- To je proces koji transformira podatke na način da ne budu čitljivi nekome tko ne posjeduje dekripcijski ključ
- Za enkripciju i dekripciju se najčešće koriste SSH (Secure Shell), SSL (Secure Socket Layer) i TLS (Transport Layer Security)
- Kod SSL-a i TLS-a svi su podaci kriptirani između klijenta (browser, mobilna aplikacija) i servera (web server najčešće) ([https](https://)**s**)

# Projekt s autentikacijom



Kako ASP .NET obavlja  
posao autentikacije i  
autorizacije?

# Klijentski dio aplikacije

(Retrofit & Fresco)

# Dohvaćanje s weba

Kako dohvatiti nešto s web-a na Androidu?

• `HttpURLConnection` klasu koja je ugrađena i u Javu

- Problem nastaje kod parsiranja podataka

• `DefaultHttpClient` koja se nalazi u androidu u paketu  
`org.apache.http.impl.client.DefaultHttpClient`

- Uklanja boilerplate kod, parsiranje je i dalje nužno

• Neki od frameworka za asinkrono dohvaćanje  
podataka

- U potpunosti automatizirane neke stvari
- Asinkroni pozivi metodama

# Dohvaćanje s weba

Na što paziti?

- Dohvaćanje je uvijek potrebno napraviti asinkrono, nikako u glavnoj dretvi!
- Najjednostavnije je koristiti Loader, AsyncTask ili neki of frameworka (npr. Retrofit ili Volley) kako bi se postigla asinkronizacija

# Dohvaćanje s weba

Usporedba brzine dohvata podataka s weba

|                  | Jedan dohvat | Dohvat Android dashboard-a | 25 dohvata |
|------------------|--------------|----------------------------|------------|
| <b>AsyncTask</b> | 941 ms       | 4.539 ms                   | 13.957 ms  |
| <b>Volley</b>    | 560 ms       | 2.202 ms                   | 4.275 ms   |
| <b>Retrofit</b>  | 312 ms       | 889 ms                     | 1.059 ms   |

# Retrofit

- Retrofit koristi deklaracije servisa preko interface-a
- Anotacije nad parametrima i metodama kako bi kreirao custom request
- Posjeduje automatske serijalizatore i deserijalizatore requesta preko plugin-a
  - Znači da je format zapravo nebitan
- Call klasa služi za jednu request/response interakciju

# Retrofit

Interface i anotacije

```
public interface GitHubService {
 @GET("users/{user}/repos")
 Call<List<Repo>> listRepos(
 @Path("user") String user);
}
```

# Retrofit

Call

- Modelira jedan request/response par
- Odvaja kreiranje requesta od rukovanja s responseom
- Svaka instanca se može koristiti samo jednom
- Instance se mogu klonirati (jeftinije kreiranje)
- Podržava asinkrono i sinkrono izvršavanje
- Može se odustati od zahtjeva

# Retrofit

Call na što paziti

```
Call<List<Repo>> repos = service.listRepos("square");
response = call.execute();
```

```
// Ovo će baciti grešku:
response = call.execute();
```

```
// Ovo će proći:
response = call.clone().execute();
```

P.S. Nemojte koristiti sinkrono dohvaćanje ako ne morate. ;)

# Retrofit

Call asinkrono

```
Call<List<Repo>> repos = service.listRepos("square");
call.enqueue(new Callback<List<Contributor>>() {
 @Override
 void onResponse(Response<List<Contributor>> cont) {
 // ...
 }
 @Override void onFailure(Throwable t) {
 // ...
 }
});
```

# Retrofit

## Ostale mogućnosti

- Retrofit 2 zna koristiti paging (hrv. Straničenje)
- Parametriziran Response objekt (kako bi se izvukli dodatni podaci)
- Dinamički URL parametri
- Višestruki i efikasni konverteri (može istodobno parsirati JSON i XML npr.)

# Retrofit upit



Jednostavan retrofit upit

# A kako dohvatiti slike?

• Postoje mnoge biblioteke

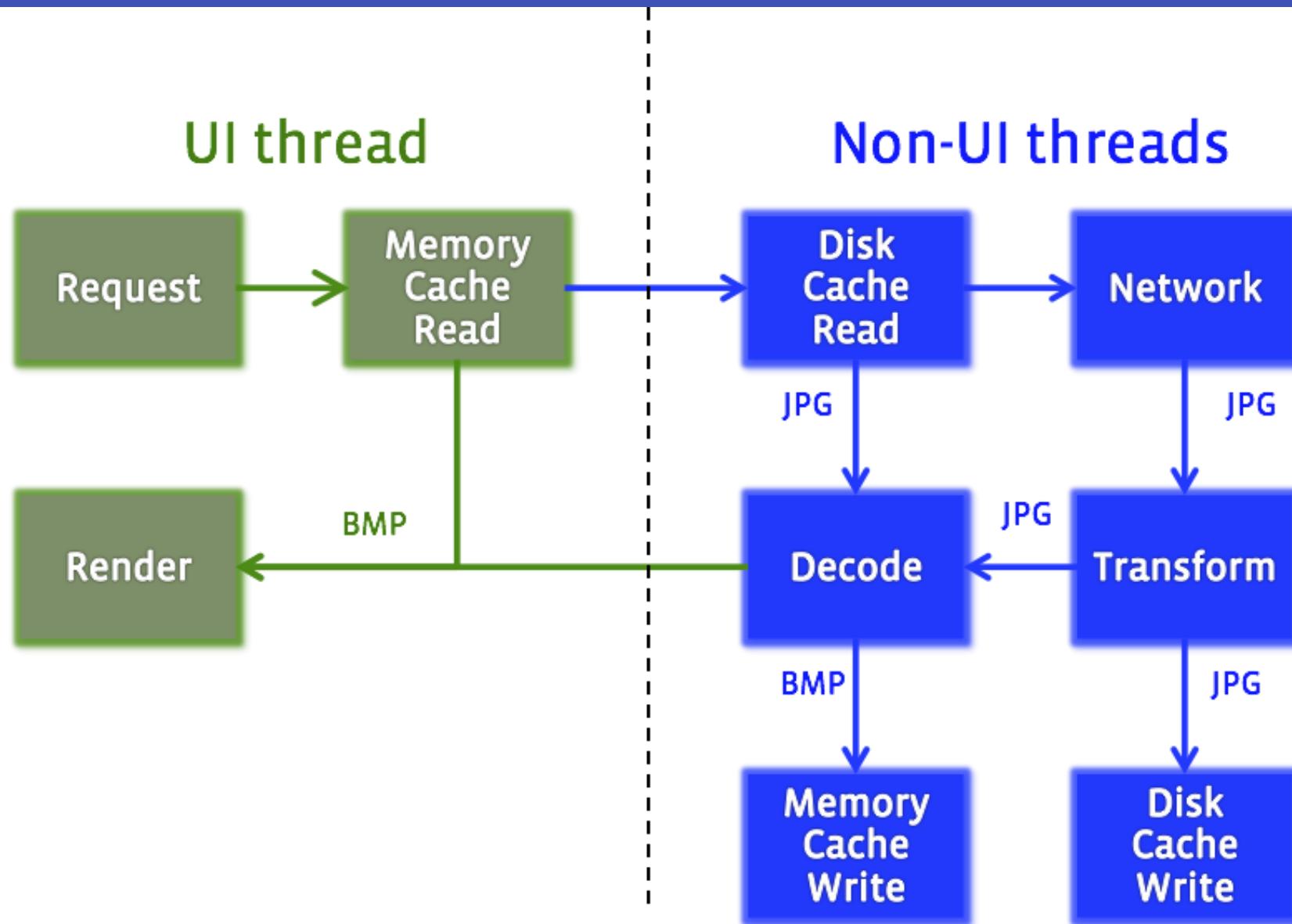
- Fresco
- Picasso
- Universal Image Loader
- Glide
- Volley Image Loader

# Koji odabrat?

Mi ćemo koristiti fresco

- Zašto što je najbrži. ☺
- Slike ne drži na Java heap-u nego na ashmem heapu
  - Sličan kao nativni heap, ali s mogućnošću lakšeg oslobođanja memorije
- Progresivno učitavanje slika
- Bolji crop
- Nativna promjena veličine slike

# Kako Fresco radi?



# Retrofit + Fresco



Jednostavan retrofit sa  
Fresco slikom

# Retrofit

... na localhostu

- Kako bi se moglo izaći iz virtualne mašine na localhost potrebno je podesiti nekoliko stvari

- U config datoteku \$(solutionDir)\.vs\config\applicationhost.config dodati unutar binding elementa web-a sljedeće:

```
<binding protocol="http" bindingInformation="*:8080:*" />
```

- Pokrenuti komandu

```
.netsh http add urlacl url=http://*:8080/ user=everyone
```

- Dodati IIS Express na firewall inbound rule

```
%ProgramFiles%\IIS Express\iisexpress.exe
```

# Retrofit

... na localhostu

• Postaviti link iz android aplikacije

• Ako se pristupa iz AVD-a:

• `http://10.0.2.2:8080/api/`

• Ako se pristupa iz Genymotion-a:

• `http://10.0.3.2:8080/api/`

# Retrofit

... debugging

- Da se omogući debugging na retrofitu potrebno je u projekt uključiti

compile '**com.squareup.okhttp3:logging-interceptor:3.14.1**'

- Potrebno je dodati dio koda kod kreiranja servisa:

```
HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
logging.setLevel(HttpLoggingInterceptor.Level.BODY);
OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
httpClient.addInterceptor(logging);
```

```
Retrofit retrofit = new Retrofit.Builder()
 .baseUrl(baseUrl)
 .addConverterFactory(GsonConverterFactory.create())
 .client(httpClient.build())
 .build();
```

# Retrofit + ToDo api



Jednostavan retrofit sa  
ToDo apijem

# Retrofit + Autentikacija

- Autentikaciju na Retrofitu je moguće napraviti na nekoliko načina.
- Najjednostavnije je da se u lokalnu bazu podataka pohranjuje korisničko ime i lozinka, a zatim se s istima dohvaća token sa servera kako se korisnik ne bi morao svaki put ulogiravati

# Retrofit + ToDo api + Auth



Jednostavan retrofit sa  
ToDo apijem i  
autentikacijom

finish();



.pitinja?

# Android

Homescreen i multimedija

# Sadržaj predavanja

## • Widgeti

- Što su widgeti
- Kako funkcioniraju
- Malo kompleksniji primjer sa satom

## • Multimedija

- Podržani audio formati
- Android Media API
- MediaStore
- MediaPlayer
- Primjeri

# Widgeti

Što su widgeti?

- Widgeti su mali aplikacijski view-ovi koji mogu biti ukomponirani u druge aplikacije (kao što je npr. Home screen) te periodično primati update-ove
- Aplikacija koja sadrži widgete se zove app widget host
- Kako bi pravilno dizajnirali widgete potrebno je proučiti smjernice dizajna:
  - <https://developer.android.com/guide/topics/appwidgets/overview>

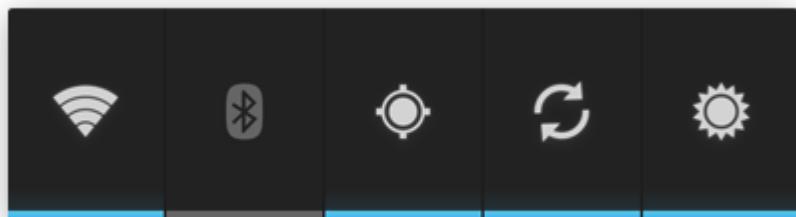
# Widgeti

Tipovi widgeta?

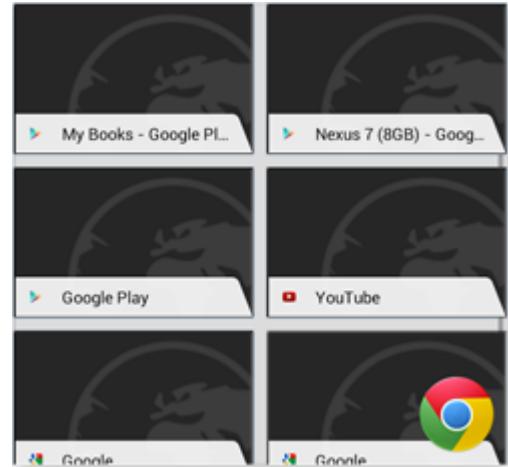
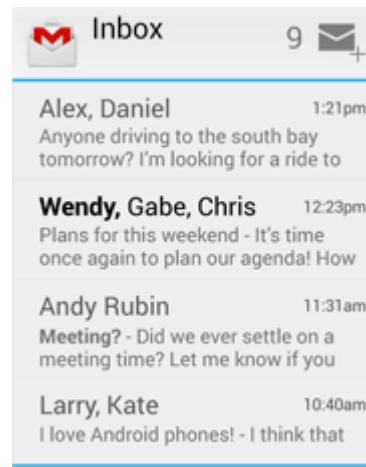
## Informacije



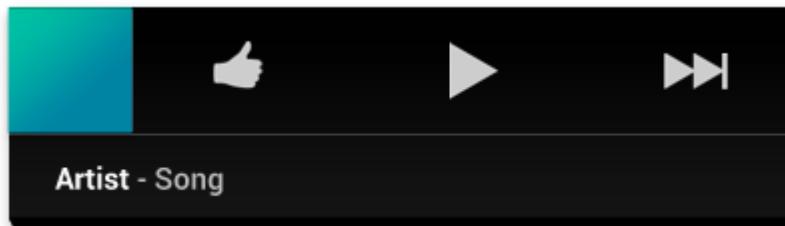
## Kontrolni



## Kolekcija (slika/mail/poruka...)



## Hibridni



# Widgeti

Što su potrebno da bi se kreirao widget?

- XML layout datoteka

- XML datoteka koja opisuje meta podatke vezane uz widget

- AppWidgetProviderInfo

- Dodavanje widgeta u manifest
- Definira se u XML-u

- AppWidgetProvider

- Broadcast receiver za widget
- Definira osnovne metode za povezivanje sa widgetom

# Manifest

U manifestu se deklarira kao broadcast receiver

```
<receiver android:name="ExampleAppWidgetProvider" >
 <intent-filter>
 <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
 </intent-filter>
 <meta-data android:name="android.appwidget.provider"
 android:resource="@xml/example_appwidget_info" />
</receiver>
```

# AppWidgetProviderInfo

Definira widget

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
 android:minWidth="40dp"
 android:minHeight="40dp"
 android:updatePeriodMillis="86400000"
 android:previewImage="@drawable/preview"
 android:initialLayout="@layout/example_appwidget"
 android:configure="com.example.android.ExampleAppWidgetConfigure"
 android:resizeMode="horizontal|vertical"
 android:widgetCategory="home_screen|keyguard"
 android:initialKeyguardLayout="@layout/example_keyguard">
</appwidget-provider>
```

# AppWidgetProviderInfo

## Atributi

### .minWidth i minHeight

- Definiraju prostor koji zauzima widget po default vrijednostima
- Ne bi trebali prekoračiti 4x4 polje

### .minResizeWidth i minResizeHeight

- Minimalan prostor koji widget može zauzeti

| Broj ćelija<br>(Kolona ili redaka) | Veličina (dp)<br>(minWidth ili minHeight) |
|------------------------------------|-------------------------------------------|
| 1                                  | 40dp                                      |
| 2                                  | 110dp                                     |
| 3                                  | 180dp                                     |
| 4                                  | 250dp                                     |

# AppWidgetProviderInfo

## Atributi

### • updatePeriodMilis

- Koliko često widget zahtjeva update od AppWidgetProvider-a

### • initialLayout

- pokazuje na resurs layout-a

### • configure

- Definira aktivnost koju widget zove prilikom dodavanja na ekran (za konfiguraciju i sl.)

### • previewImage

- Specificira sliku koja će se pokazivati gotov widget kako bi ga korisnik lakše prepoznao i odvukao na ekran
- Ako se ne specificira, pokazuje se ikona programa

# AppWidgetProviderInfo

## Atributi

### • autoAdvanceViewId

- Specificira view element unutar widgeta koji treba auto-advance-ati (Npr. neku listu ili slično)

### • resizeMode

- Može li widget mijenjati veličinu ili ne i u kojem smjeru

### • widgetCategory

- Definira gdje se može dodati widget. (homescreen, lock screen ili oboje)
- Može imati vrijednosti "home\_screen" ili "keyguard,"
- Verzije 5.0 na više podržavaju samo home screen

### • initialKeyguardLayout

- Pokazuje na layout za lock screen
- Moguće od Androida 4.2

# Layout widgeta

• Moguće je koristiti layoute:

- FrameLayout, LinearLayout, RelativeLayout, GridLayout

• Widget klase

- AnalogClock, Button, Chronometer, ImageButton, ImageView, ProgressBar, TextView, ViewFlipper, ListView, GridView, StackView, AdapterViewFlipper

• Potomci ovih klasa NISU dozvoljeni!

• Također se može koristiti ViewStub koji je nevidljiv view kako bi se inflateao layout iz resursa u trenutku izvođenja.

# Margine

- Widgeti ne bi smjeli biti kreirani da budu od ruba do ruba
- Od Ver. 4.0 widgetima se automatski dodaje padding (najbolje je namjestiti target SDK na 14 ili više)
- Kako bi radile sve aplikacije moguće je koristiti alternativne resurse

```
<FrameLayout
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:padding="@dimen/widget_margin">
```

- res/values/dimens.xml: <dimen name="widget\_margin">8dp</dimen>
- res/values-v14/dimens.xml: <dimen name="widget\_margin">0dp</dimen>

# AppWidgetProvider

(Bez Info)

• Nasljeđuje BroadcastReciever

• Lovi samo evenete vezane uz broadcaste koji su relevantni widgetu:

## **.onUpdate()**

- Zove se u intervalu koji je podešen s atributom updatePeriodMillis
- Također se zove nakon dodavanja widgeta osim ako nije podešena konfiguracijska aktivnost

## **.onAppWidgetOptionsChanged()** (od Androida 4.1)

- Poziva se prilikom prvog smještanja widgeta ili promjene veličine
- Veličinu je moguće dobiti pomoću getAppWidgetOptions() koji vraća Bundle s konstantama:

- OPTION\_APPWIDGET\_MIN\_WIDTH,
- OPTION\_APPWIDGET\_MIN\_HEIGHT,
- OPTION\_APPWIDGET\_MAX\_WIDTH,
- OPTION\_APPWIDGET\_MAX\_HEIGHT

# AppWidgetProvider

(Bez Info)

## **.onDeleted(Context, int[])**

- Poziva se kada se widget obriše s widget host-a

## **.onEnabled(Context)**

- Poziva se kada se instanca widgeta kreira po prvi puta. Npr. ako korisnik kreira dvije instance widgeta, ova metoda se poziva samo prvi puta. Koristi se za kreiranje baze ili nekih postavki aplikacije

## **.onDisabled(Context)**

- Poziva se kada je zadnja instanca widgeta obrisana s host-a.
- U ovoj metodi bi se trebao napraviti cleanup resursa koji su kreirani u onEnabled metodi

## **.onReceive(Context, Intent)**

- Ova metoda se poziva za svaki broadcast i prije svake od navedenih metoda.
- Najčešće nije potrebno implementirati ovu metodu jer AppWidgetProvider automatski filtrira korisne broadcastove

# App widgets



Prikaz sata

# Multimedija

# Podržani audio formati

(Neki od)

- AAC - .m4a i .3gp - audio kodek unutar MPEG-4 videozapisa
- MP3 - .mp3 – najrašireniji, najbolji omjer kvalitete/veličine zapisa
- AMR - .amr i .3gp – niske kvalitete, dobar za pozive, loš za glazbu
- OGG - .ogg - open source, konkurent .mp3 formatu
- PCM - .wav – nekompresirani audio, 3 min pjesme = cca. 45 MB
- FLAC - .flac - visoka kvaliteta s malenim zauzećem memorije
- MIDI - .midi - sekvenca tonova, zapis koji sadrži samo note

# Android media API

1/2

- Sve klase za rad s multimedijom smještene u paketu android.media
- Sadrži klase za snimanje i reprodukciju audio i video zapisa te rad sa slikama
- Klase za upravljanje parametrima zvuka se nalaze u paketu android.media.audiofx, koriste se za Equalizer, Bass Boost i slično
- android.media.effect sadrži klase za editiranje videa i slika, zahtjeva OpenGL podršku

# Android media API

2/2

• Osim standardnih klasa postoje i tri posebne klase:

- FaceDetector - klasa koja služi za detektiranje lica na slikama
- MediaRouter - klasa za upravljanje gdje se reproducira multimedija, npr. eksterni zvučnik, slušalice ili nešto treće
- AudioManager - upravljanje zvukovima alarma, poziva, sistemskih događaja i slično

# MediaStore

• Podijeljen na 4 klase i 1 interface:

- Audio - sadrži sve audio datoteke
- Files - indeks svih datoteka na uređaju
- Images - sadrži metapodatke o svim slikama
- Video – sadrži sve o videozapisima
- MediaColumns - zajedničke značajke svih tipova multimedije

# MediaStore.Audio

- Baza svih pjesama na uređaju
- Sadrži tablice svih albuma, izvođača, žanrova i playlista
- Pomoću SQL upita filtriramo sve pjesme po npr. određenom žanru
- Problemi s lošom hijerarhijom:
  - MediaStore.Audio.Albums - svi albumi svih izvođača
  - MediaStore.Audio.Artists.Albums - svi albumi jednog izvođača

# Dohvat podataka iz MediaStore-a

1/4

- Prije samog upita treba "pripremiti" sve parametre za query:

```
context.getContentResolver
 .query(uri, projection, selection, selectionArgs,
 sort);
```

- uri - kakav sadržaj se traži
- projection - koje stupce treba prikazati
- selection - koje podatke je potrebno preuzeti
- selectionArgs - argumenti za upite s "=?"
- sort - način sortiranja

# Dohvat podataka iz MediaStore-a

2/4

.uri – traži se sadržaj s korisničkog prostora na uređaju:

`MediaStore.Audio.Media.EXTERNAL_CONTENT_URI`

.projection - polje stringova s popisom stupaca koje je potrebno izvući:

```
String[] projection = {
 MediaStore.Audio.Media.TITLE,
 MediaStore.Audio.Media.ARTIST,
 MediaStore.Audio.Media.DATA };
```

# Dohvat podataka iz MediaStore-a

3/4

- **.selection** - dohvati svu glazbu (ne zvukove alarma, notifikacija i sl.)

```
String selection =
 MediaStore.Audio.Media.IS_MUSIC + " != 0";
```

- **.selectionArgs** – Argumenti selekcije

- **.sort** - sortiraj po nazivu pjesme uzlazno

```
String sort =
 MediaStore.Audio.Media.TITLE + " ASC";
```

# Dohvat podataka iz MediaStore-a

4/4

- Kada su parametri pripremljeni, potrebno je izvesti query koji vraća ResultSet u Cursoru

```
while(cursor.moveToNext()){
 // rad s podacima
}
```

# MediaPlayer

1/3

- Glavna klasa za sviranje i manipulaciju audio datoteka
- Uvijek imati samo jednu instancu objekta MediaPlayer
- MediaPlayer objekt od početka do kraja reprodukcije prolazi kroz nekoliko od ukupno 10 stanja (state)
- Sve metode imaju stanja u kojima se smiju ili ne smiju pozivati
- Pozivanje metode u krivom stanju prebacuje objekt u ERROR state i baca IllegalStateException

# MediaPlayer

2/3

- `create()` - kreira objekt s fiksnim audio zapisom za reprodukciju
- `pause()` - zaustavlja reprodukciju ali ostaje na trenutnoj poziciji zapisu
- `prepare()` - priprema zapis, ako je lokalni izvede se odmah
- `prepareAsync()` - `prepare()` ali u novoj dretvi, preporuča se za stream
- `release()` - otpušta sve zauzete resurse i sam objekt
- `reset()` - stavlja objekt u stanje prije inicijalizacije (null)
- `setDataSource()` - daje objektu putanju do zapisu ili streama
- `start()` - ovisno o prethodnom stanju pokreće ili nastavlja reprodukciju
- `stop()` - zaustavlja reprodukciju

# MediaPlayer

3/3

- IDLE - stanje nakon deklaracije new MediaPlayer() i metode reset()
- INITIALIZED - stanje nakon inicijalizacije i setDataSource()
- PREPARING - nakon prepareAsync(), player buffera stream
- PREPARED - nakon prepare(), spreman za početak reprodukcije
- STARTED - nakon start(), reproducira se audio zapis
- PAUSED - nakon pause(), zaustavlja se reprodukcija na trenutnoj poziciji
- STOPPED - nakon stop(), potrebno ponovo pozvati prepare() prije start()
- PLAYBACKCOMPLETED - kada reprodukcija sama završi
- END - nakon release(), za ponovo korištenje objekta proći sve ispočetka
- ERROR - kad napravite glupost ☺

# Pozivanje ugrađenog playera

- Implicitni intent
- URI (putanja) audio datoteke
- MIME tip podatka (u ovom slučaju mp3 datoteka)

```
Intent intent = new Intent(android.content.Intent.ACTION_VIEW);
intent.setDataAndType(PUTANJA_DO_AUDIO_DATOTEKE, "audio/mp3");
startActivity(intent);
```

- Implicitno pozivanje samo za lokalne datoteke!

# Kreiranje media playera

1/3

- Kreirati objekt media playera
- Pozvati metodu create() te u nju predati kontekst i URI datoteke
- Pozvati metodu start za pokretanje reprodukcije

```
MediaPlayer mp = mp.create(this, R.raw.some_audio_file);
mp.start();
```

- Koristiti samo s audio datotekama iz same aplikacije (raw)

# Kreiranje media playera

2/3

```
mp = new MediaPlayer();
mp.setDataSource("PATH_TO_LOCAL_FILE");
mp.prepare();
mp.start();
```

• Koristiti samo s lokalnim datotekama na uređaju

# Kreiranje media playera

3/3

- Za online stream dodati vrstu streama:

```
mp.setAudioAttributes(new AudioAttributes
 .Builder()
 .setLegacyStreamType(AudioManager.STREAM_MUSIC)
 .build());
```

- S obzirom da se stream mora prvo bufferati potrebno je asinkrono pripremiti sadržaj:

```
mp.prepareAsync();
```

- I dodati onPreparedListener() da reprodukcija započne kada se završi bufferanje

# Media player



Audio datoteke  
Video datoteke  
Streaming

finish();



.pitinja?

# Android

## Obrasci programiranja

# Sadržaj predavanja

## .Obrasci programiranja

- MVC
- MVP
- MVVM
- Clean

## .Dagger2

# Zašto je arhitektura aplikacije bitna?

- Jednostavnost koda

- Definiranje uloge svake komponente u aplikaciji omogućuje preglednost

- Testiranje

- Jednostavnije pisanje testova (Unit testiranje)

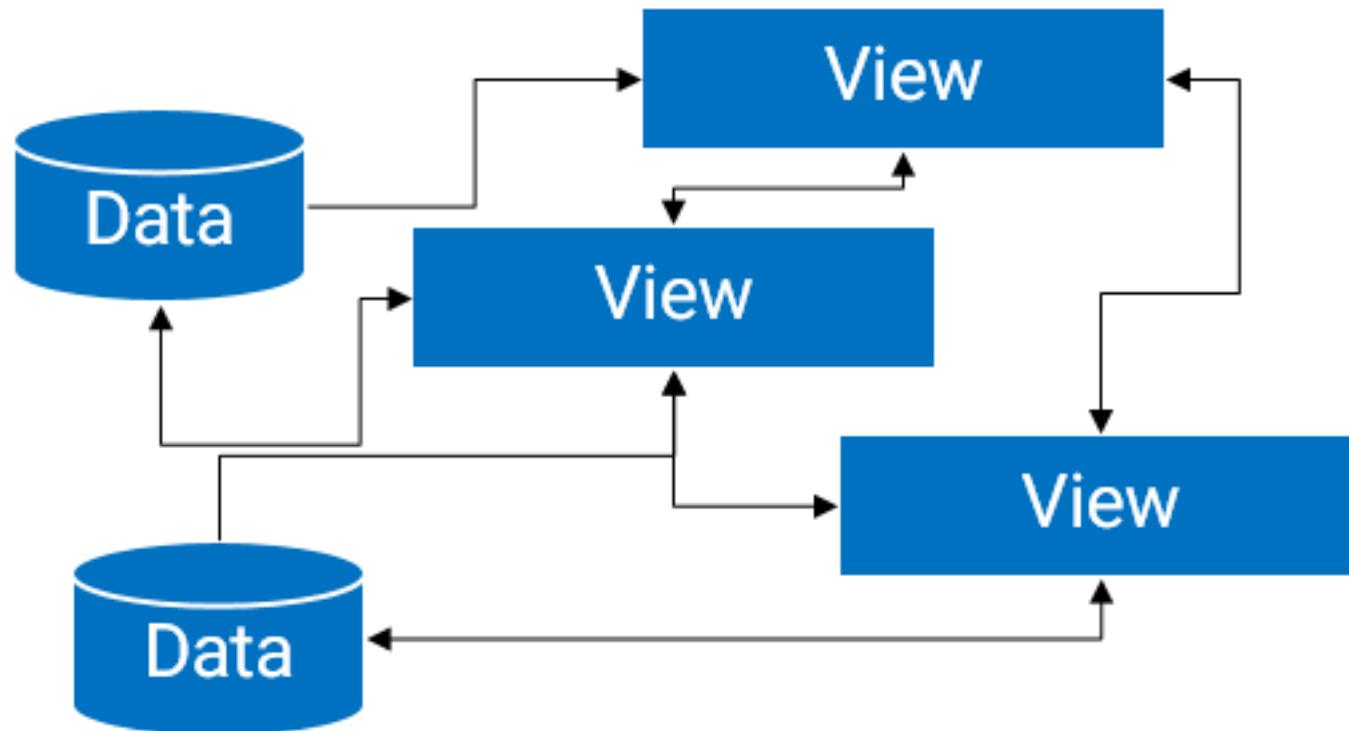
- Održavanje

- Jednostavnost prilikom dodavanja i uklanjanja značajki aplikacije

# God object

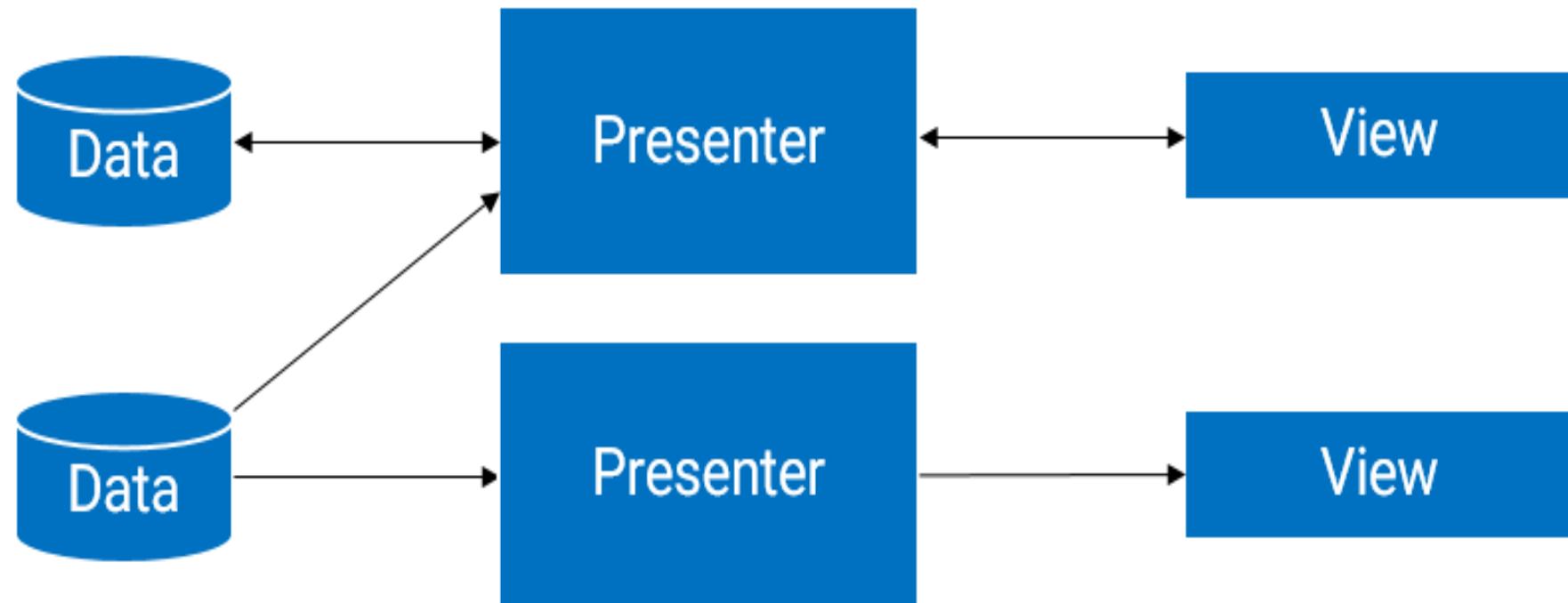
"A year from now you may wish you had started today."

• aktivnosti su usko vezane za sučelje i mehanizme za dohvaćanje podataka – *God object*



# Rješenje

- potrebno definirati dobro odvojene slojeve



# SOLID principi

1/2 SO

- Idućih 5 principa koji čine dizajn aplikacija razumljivijim, fleksibilnijim i održivijim su:
  - **Single responsibility** – svaka komponenta bi trebala imati jednu odgovornost (zadatak)
  - **Open-closed** – potrebno je moći proširiti komponentu na način da se i dalje jednako koristi i da se njena proširenja na diraju

# SOLID principi

2/2 LID

- **Liskov substitution** – ako je klasa A podklasa klase B, trebali bismo moći zamijeniti klasu B sa klasom A bez narušavanja aplikacije
- **Interface segregation** – bolje je imati više manjih sučelja nego jedno veliko kako bi se spriječilo da klasa implementira nepotrebne metode
- **Dependency inversion** – komponente bi trebale ovisiti o apstrakcijama, a ne o konkretnim implementacijama

# KISS princip

Keep It Simple, Stupid

- Jedan od glavnih principa kod arhitekture aplikacija
- Raščlanjivanje kompleksnog problema i implementacija istog pomoću manjih jednostavnih dijelova

# Koje arhitekture postoje?

• Ne postoji jedna arhitektura koja odgovara svim aplikacijama.

• Najpoznatije:

- MVC (Model – View – Controller)
- MVP (Model – View –Presenter)
- MVVM (Model – View – ViewModel)
- Clean arhitektura

# Model – View – Controller

MVC

.Kod je podijeljen na 3 dijela:

- . Model**

- . sve klase koje se povezuju sa podatcima

- . View**

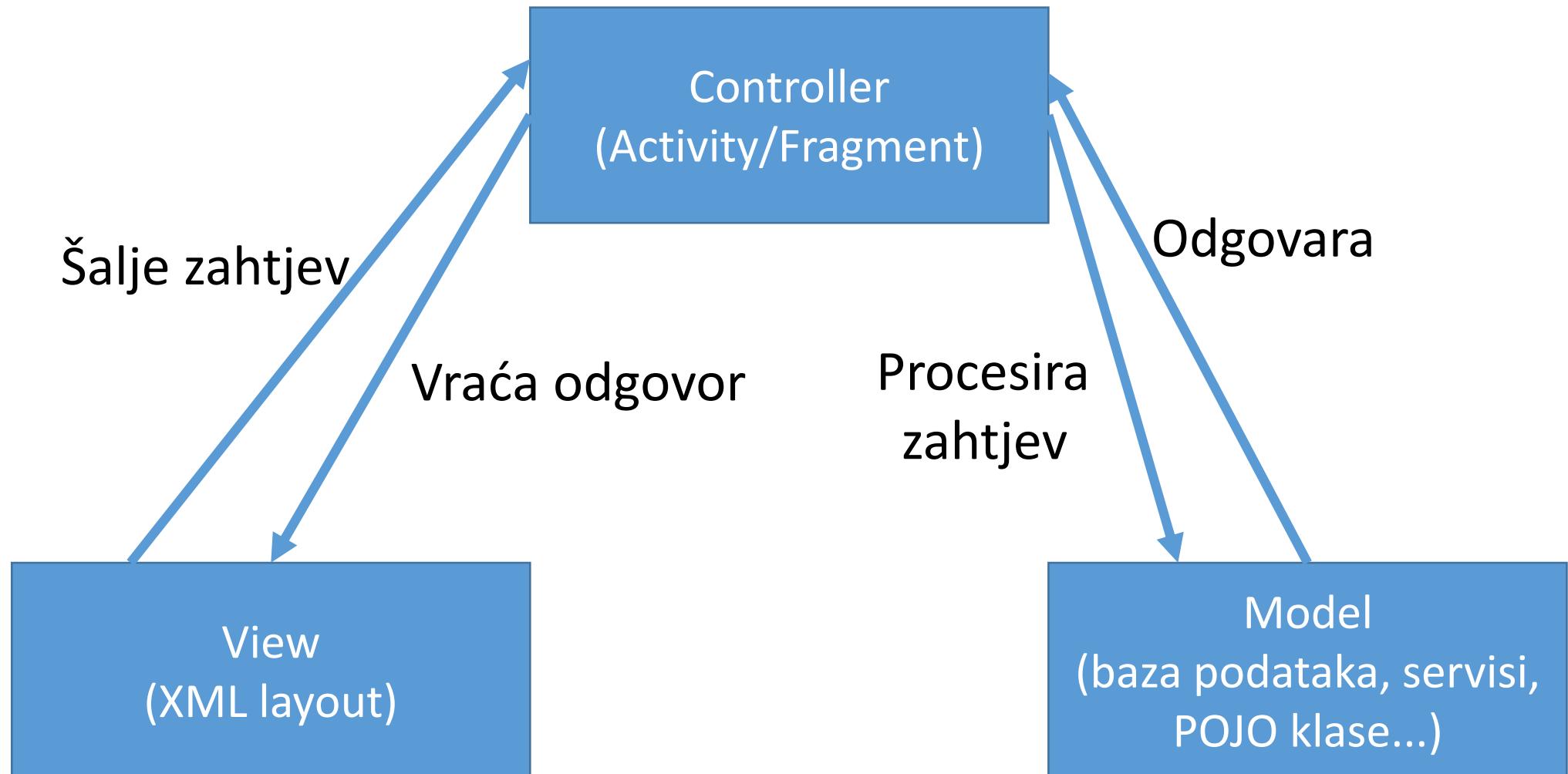
- . korisničko sučelje
  - . XML datoteke

- . Controller**

- . komunikacijski medij između Model i View elemenata
  - . uzima korisnički unos iz UI-a, procesira zahtjev i vraća podatke od Model elementa
  - . aktivnost ili fragment

# Model – View – Controller

MVC



# Model – View – Controller

MVC

## .Prednosti:

- brži razvoj
- jednostavno surađivanje na aplikaciji (više developera)
- jednostavno ažuriranje aplikacije
- jednostavnije otklanjanje pogrešaka (debugging)

## .Nedostatci:

- teže razumijevanje arhitekture
- stroga pravila o metodama

# Demo



MVC arhitektura

# Model – View – Presenter

MVP

• Kod je podijeljen na 3 dijela:

- **Model**

- sve klase koje se povezuju sa podatcima

- **View**

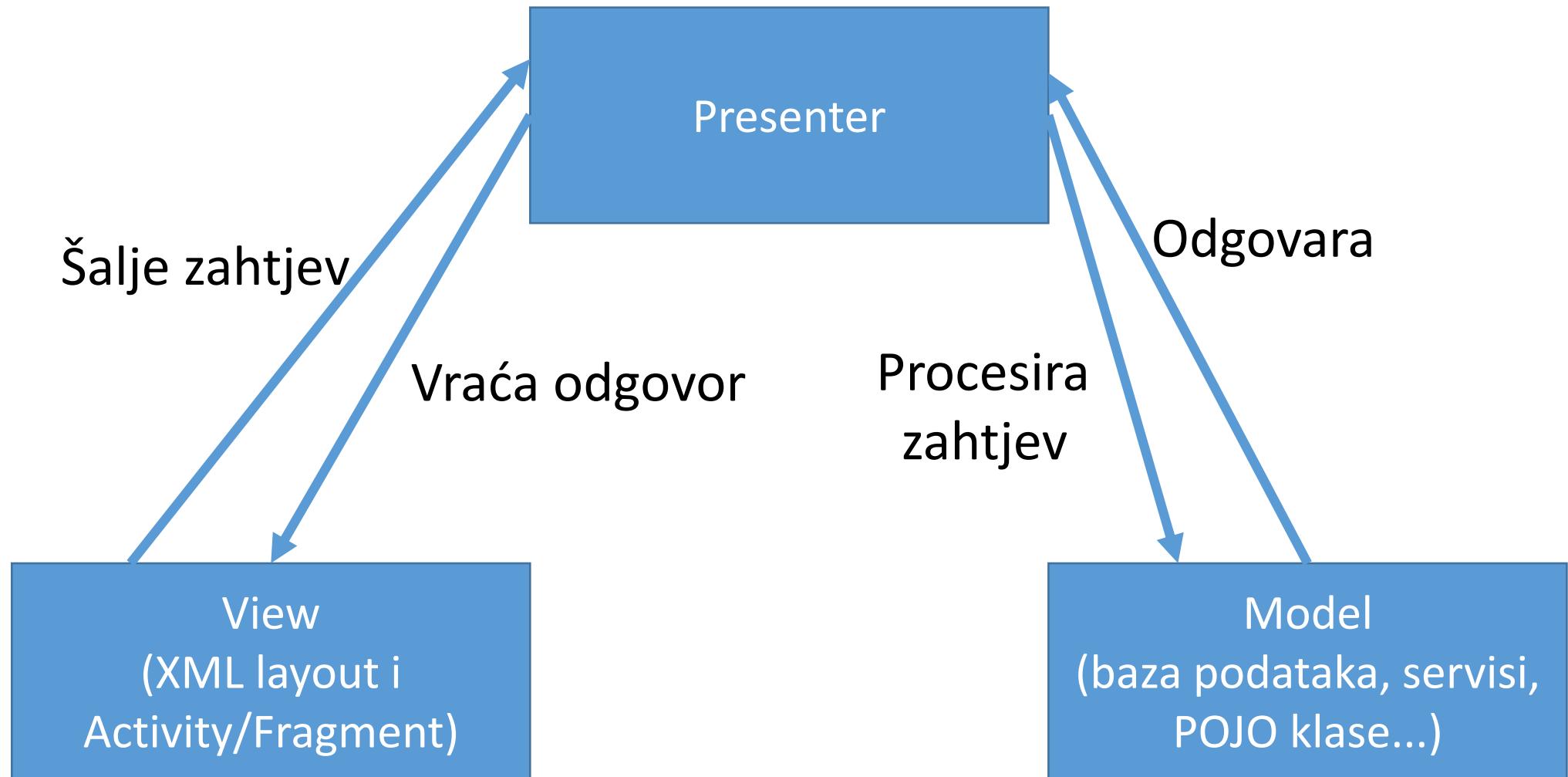
- korisničko sučelje (XML layout) zajedno sa aktivnostima/fragmentima
  - implementiraju interface za Presenter

- **Presenter**

- komunikacijski medij između Model i View elemenata
  - nema poveznice za View elementima

# Model – View – Presenter

MVP



# Model – View – Presenter

MVP

## .Prednosti:

- brži razvoj
- jednostavno surađivanje na aplikaciji (više developera)
- jednostavno ažuriranje aplikacije
- jednostavnije otklanjanje pogrešaka (debugging)
- kod je neovisan o korisničkom sučelju
- manje koda u klasama

## .Nedostatci:

- teže razumijevanje arhitekture
- više datoteka

# Demo



MVP arhitektura

# Model – View – ViewModel

MVVM

.Kod je podijeljen na 3 dijela:

- **Model**

- sve klase koje se povezuju sa podatcima

- **View**

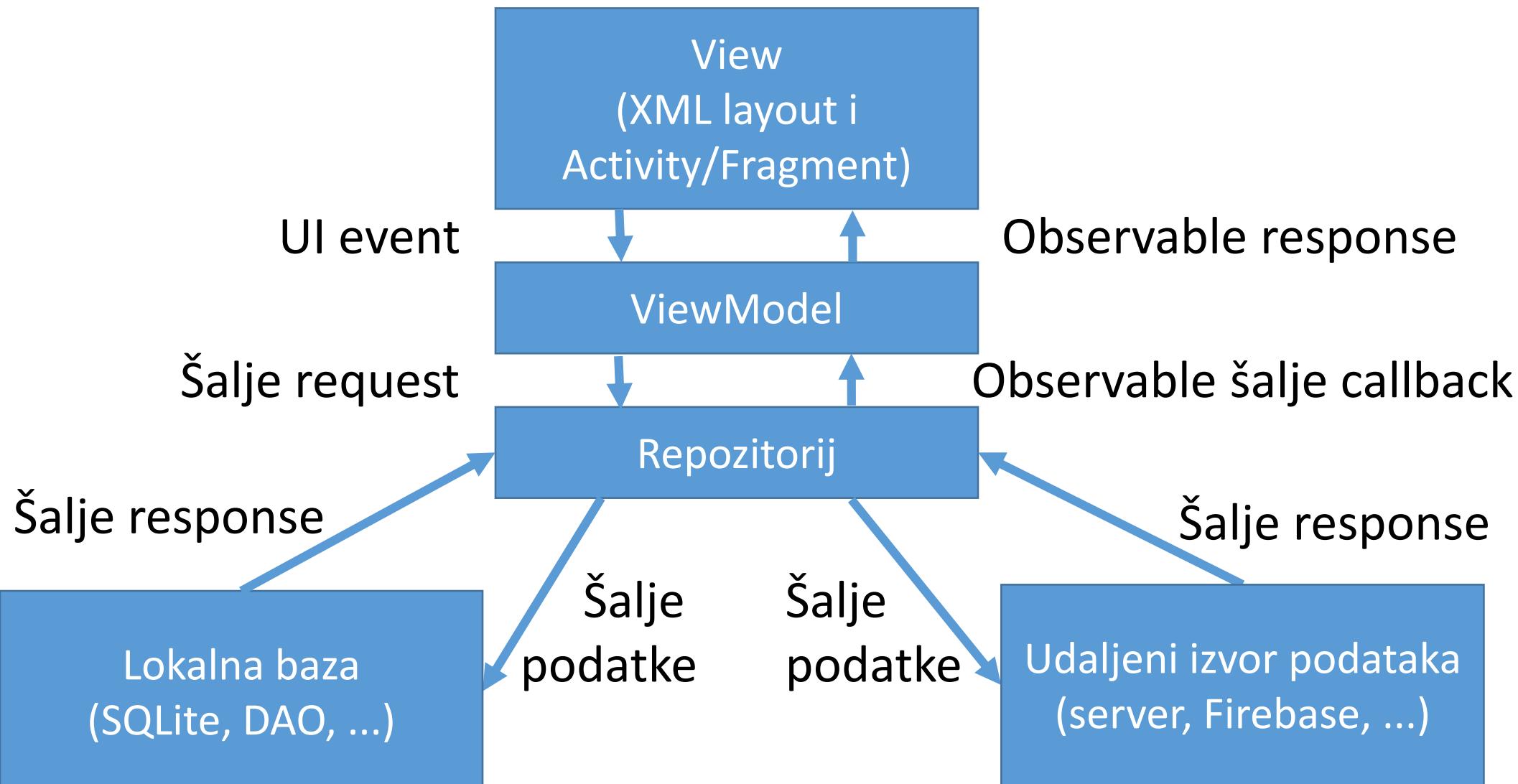
- korisničko sučelje (XML layout) zajedno sa aktivnostima/fragmentima

- **ViewModel**

- komunikacijski medij između Model i View elemenata
- sadrži Model
- kada se desi ažuriranje, relevantni View elementi će dobiti nove podatke (Data Binding)

# Model – View – ViewModel

MVVM



# Model – View – ViewModel

MVVM

## .Prednosti:

- manje koda u klasama
- jednostavno ažuriranje aplikacije
- individualne komponente omogućuju brže testiranje

## .Nedostatci:

- u nekim slučajevima kod je u XML-u što može biti kompleksno za developera
- moguće se raditi sa View-om samo na dva načina: Data Binding ili pomoću View metoda

# Demo



MVVM arhitektura

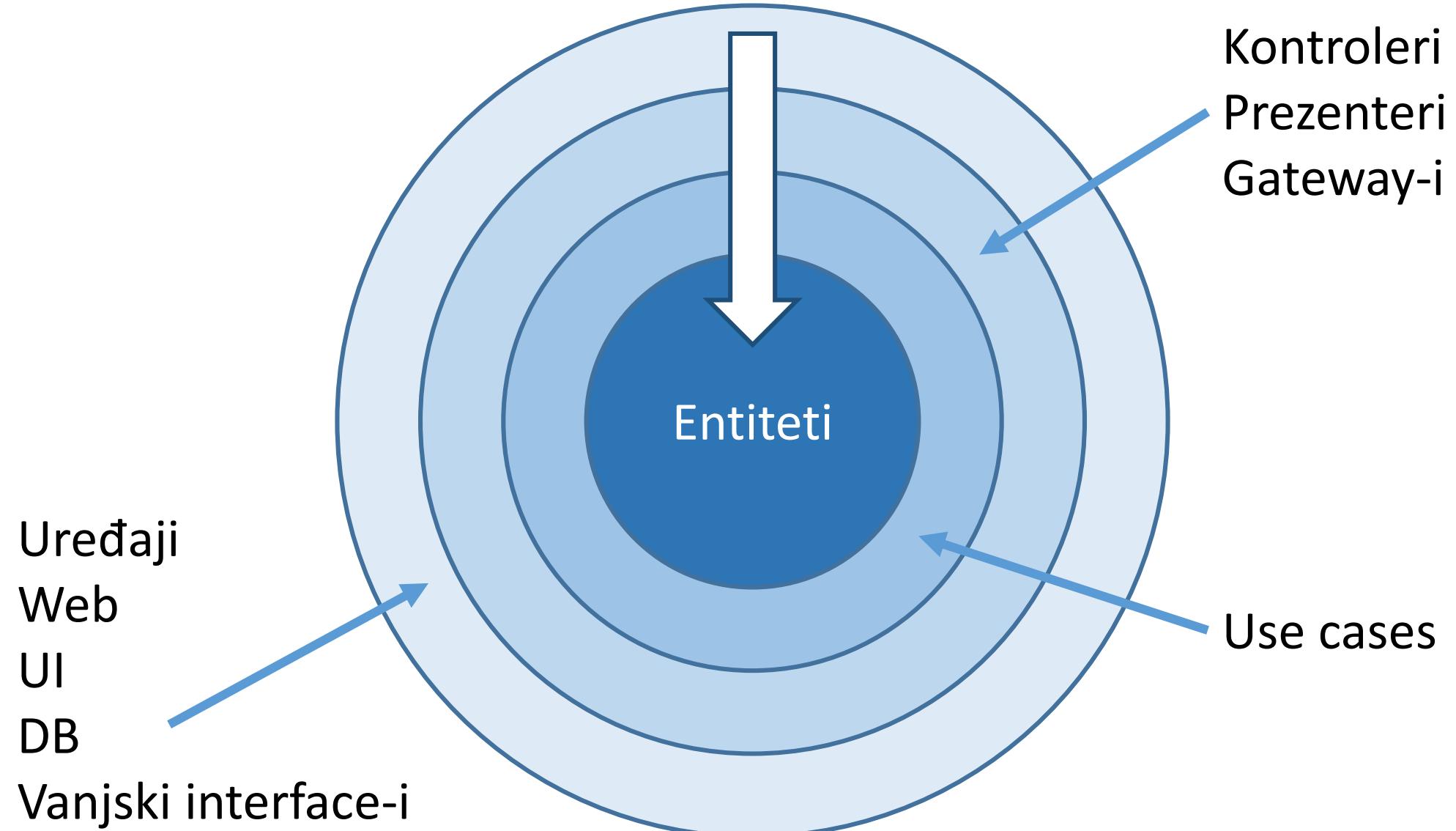
# Clean arhitektura

Clean

- Definira na koji način će slojevi aplikacije vršiti međusobnu interakciju
- Više koncept nego arhitektura

# Clean arhitektura

Clean



# Clean arhitektura

Clean

## .Prednosti:

- jednostavno testiranje
- neovisnost elemenata
- jednostavna struktura paketa
- jednostavnost održavanja aplikacije
- jednostavno ažuriranje aplikacije

## .Nedostatci:

- strma krivulja učenja
- više datoteka

# Dagger2

- Framework za dependency injection
- Kreiran od strane Square-a (Jake Wharton ☺)
  - Dagger2 je fork kojeg razvija Google
- Baziran na JSR 330 standardu (Java Specification Request)
- Generirani kod je jednostavno za čitati i debugirati

# Dagger2

Glavne anotacije

• @Inject

• @Module

• @Provides

• @Component

• @Scope

• @Qualifier

# Dagger2

@Inject

- Zahtjev za dependency-em
- U prijevodu, njome se govori Dagger-u da anotirana klasa ili varijabla želi sudjelovati u injektiranju
- Dagger će konstruirati instance ovako anotirane klase kako bi zadovoljio njene dependency-je

# Dagger2

@Module

- Moduli su klase čije metode pružaju dependency-je
- Pomoću modula Dagger zna gdje može pronaći dependency-je

# Dagger2

@Provides

- Nalazi se unutar modula iznad metode i govori Dagger-u kako se želi kreirati dependency

# Dagger2

@Component

- Component je zapravo injektor, odnosno most između Inject-a i Module-a
- Njegova glavna namjena je da spoji Inject i Module

# Dagger2

@Scope

- Nije potrebno da svaki objekt bude menadžer svojih instanci
- Npr. @PerActivity anotacijom je moguće dobiti ponašanje da objekt živi dok god i aktivnost živi

# Dagger2

@Qualifier

- Koristi se kada tip klase nije dovoljan da se identificira dependency
- U Androidu je često potreban drugačiji tip konteksta

# Demo



MVP dagger2 arhitektura

finish();



.pitinja?