



RAZVOJ MOBILNIH APLIKACIJA

Radna bilježnica



Sadržaj:

0. Prije početka.....	4
0.1. Uvod.....	4
0.2. Potreban alat.....	6
0.3. Dodatni izvori za učenje.....	11
1. Laboratorijska vježba 1.....	14
1.1. Uvod.....	14
1.2. Kreiranje Android projekta	15
1.3. Manifest	20
1.4. Resursi.....	22
1.1. Kod.....	36
1.2. Primjeri.....	42
1.3. Zadaća.....	43
2. Laboratorijska vježba 2.....	46
2.1. Uvod.....	46
2.2. Intent	46
2.3. RecyclerView	55
2.4. Primjer	61
2.5. Zadaća.....	61
3. Laboratorijska vježba 3.....	64
3.1. Uvod.....	64
3.2. Dijeljene postavke	65
3.1. Korištenje baze podataka	67
3.2. Fragmenti.....	72
3.3. Primjeri	78
3.4. Zadaća.....	78
4. Laboratorijska vježba 4.....	81

4.1. Uvod	81
4.2. Pozadinska obrada zadataka	82
4.1. Primatelji odašiljanja (engl. <i>Broadcast Recievers</i>)	100
4.1. Primjeri	102
4.2. Zadaća	103
5. Laboratorijska vježba 5	106
5.1. Uvod	106
5.2. Senzori	106
5.3. Korištenje akcelerometra	108
5.1. Notifikacije	109
5.1. Soundpool	111
5.2. Geolokacijske usluge	113
5.3. Objava aplikacija	121
5.4. Potpisivanje	121
5.5. Google Play Store	124
5.6. Primjeri	125
5.7. Zadaća	126

0. Prije početka

Razvoj mobilnih aplikacija kolegij je koji se izvodi na prvoj godini diplomskih studija Računarstva – izborni blok Programsko inženjerstvo i Elektrotehnike – smjer Komunikacije i informatika na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Cilj je kolegija ponuditi studentima potrebna znanja kako bi samostalno mogli izrađivati programska rješenja namijenjena mobilnim platformama. Zbog raširenosti i dostupnosti, kao ciljana platforma odabran je Android. Riječ je o platformi koju je razvila malena tvrtka pod nazivom Android Inc. i namijenila ju digitalnim kamerama. Nedugo nakon toga tvrtku je kupio Google, a ostalo je povijest. Kakva je to točno povijest bila, moguće je saznati u detaljnem pregledu Rona Amadea objavljenom na Ars Thechnici.



Ars Technica – Povijest Androida – Ron Amadeo

<http://arstechnica.com/gadgets/2016/10/building-android-a-40000-word-history-of-googles-mobile-os/>

Kolegij je razložen u tri nastavne jedinice: predavanja, laboratorijske i konstrukcijske vježbe. Ovaj priručnik namijenjen je uporabi na laboratorijskim vježbama, te je shodno tome i organiziran. Cilj je kroz pet cjelina (u jednakom broju termina LV) približiti studentima osnove razvoja za Android platformu. Kako je riječ o osnovnim konceptima koje je nužno usvojiti, tako ovaj priručnik nosi naziv „radna bilježnica“. Ipak, neke teme bit će obrađene i pokrivene u dodatnom, posljednjem, poglavljju, koje će biti proširivano vremenom, dok su mogućnosti stjecanja naprednijih znanja pružene kroz vanjske poveznice i literaturu.

Za sve komentare, kritike, sugestije, pitanja i informacije, možete se javiti na bruno.zoric@ferit.hr

0.1. Uvod

Prilikom korištenja ovog priručnika moguće se susresti s različitim simbolima i okvirima. Svaki od njih označen je bojom i grafikama u ovisnosti o tome što predstavlja, kako je opisano u nastavku.

- **Obavijest**



Ovo je obavijest. Uobičajeno sadržava usputne informacije ili poveznice na vanjske resurse koji mogu pomoći prilikom učenja.

- **Napomena**



Ovo je napomena. Uobičajeno će se rabiti kada je važno zapamtiti i kasnije rabiti kakav implementacijski detalj ili informaciju.

- **Upozorenje**



Ovo je upozorenje. Uobičajeno će se rabiti u situacijama kada je nužno uputiti čitatelja na nekakav problem ili mu skrenuti pažnju na česte greške.

- **Primjer**



Primjer 1.

Kroz cijeli priručnik moguće je pronaći različite primjere koji pokazuju osnovne koncepte koji se usvajaju u danom poglavlju. Uobičajeno će se sastojati od opisa primjera, slike konačnog izgleda te izvornog koda. U slučaju većih segmenata koda, isti će biti postavljen na neki on-line GIT repozitorij.



```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

- **Zadatak**



Zadatak 1.

Za samostalan rad u priručniku su postavljeni brojni zadaci. Uobičajeno su zadani kao opis funkcionalnosti koje je nužno ugraditi, a dana je i slika konačnog izgleda aplikacije. Svakako se preporučuje rješavanje svih zadataka, ali i njihovo proširivanje i nadogradnja.

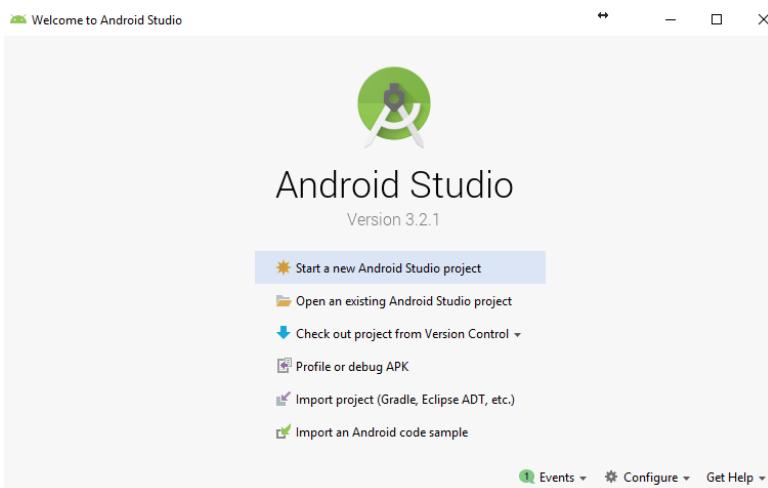


0.2. Potreban alat

Kako bi se moglo pristupiti razvoju mobilnih aplikacija, potreban je određeni alat. Cilj je ovog poglavlja predstaviti alate i resurse koji se koriste prilikom razvoja programskih rješenja za Android platformu.

0.2.1. Razvojno okruženje

Razvojno okruženje koje se koristi za razvoj Android aplikacija jest Android Studio. Iako nije jedini izbor, sasvim sigurno jest najpopularniji i postao je *de facto* standard za navedene potrebe. Riječ je o integriranom razvojnem okruženju zasnovanom na IntelliJ IDEA platformi koje nudi niz razvojnih alata poput uređivača teksta, sustava za prevodenje



Slika 0.1. Prikaz početnog sučelja Android studija

(build), emulator, integraciju s alatima za verzioniranje koda, alate za statičku analizu koda i brojne druge mogućnosti.



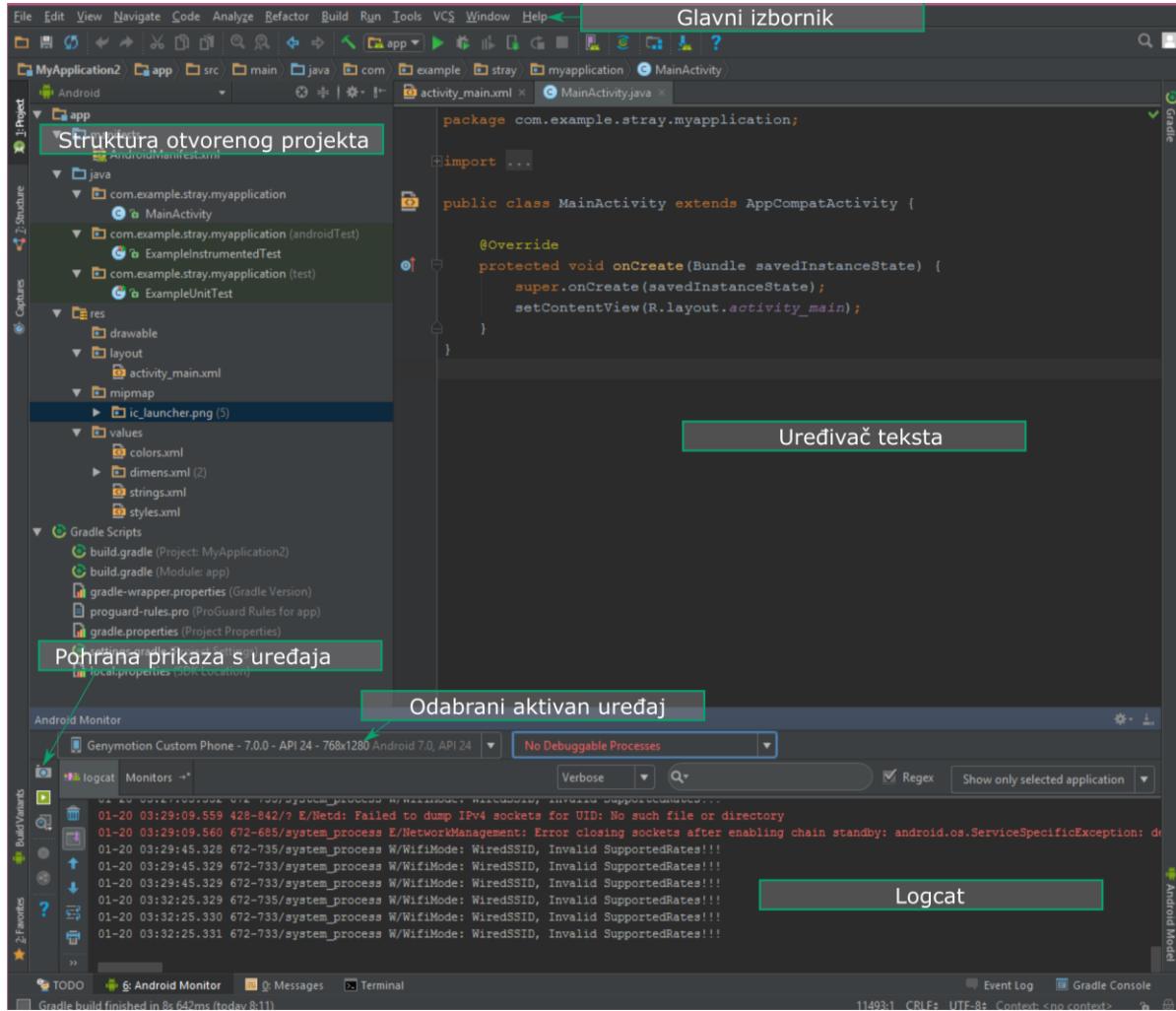
0.2.2. Android SDK

Kako bi se moglo razvijati aplikacije za Android platformu potrebna su dva razvojna paketa (engl. *development kit*), a riječ je o Java razvojnom paketu JDK i Android SDK. Kako je moguće zaključiti, programski jezici pomoću kojih se razvijaju Android aplikacije su Java i Kotlin (iako, za posebne namjene moguće je koristiti i C++).



Android SDK moguće je preuzeti uz razvojno okruženje, dok je Java JDK potrebno zasebno preuzeti i instalirati. Verzije Android SDK-a su brojne, a prate verzije Androida kako su

izlazile. SDK je u pravilu dostupan prije nego nova verzija Androida bude izdana za prve uređaje, kako bi se aplikacije mogle prilagoditi za novu platformu. Kod prve instalacije

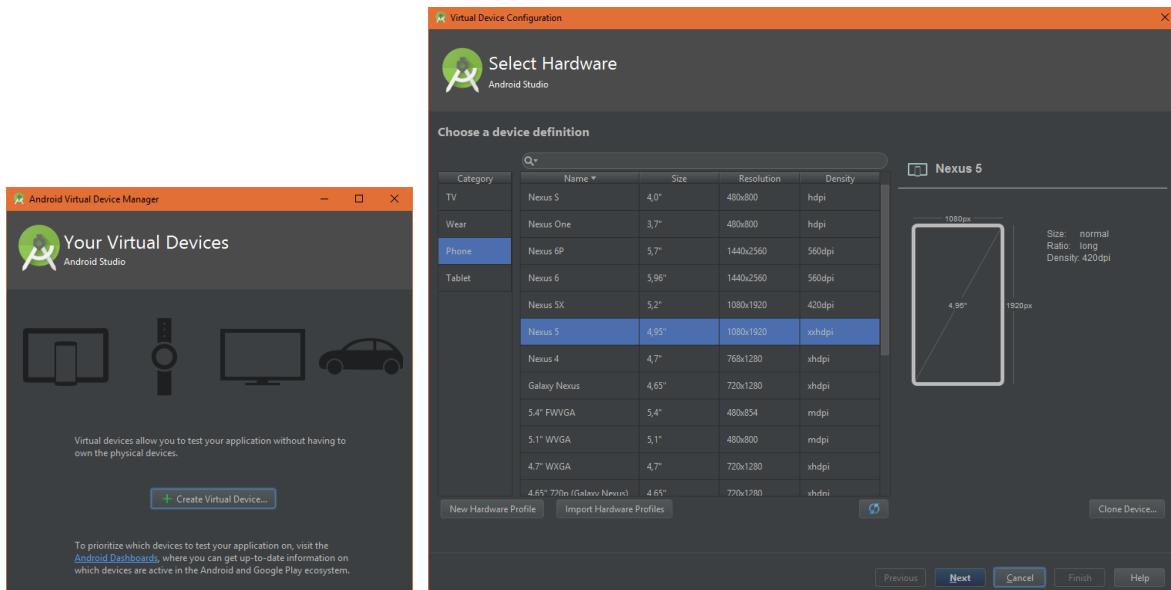


Slika 0.2. Izgled Android studija s ključnim dijelovima

dovoljno je instalirati najnoviju verziju SDK, a ako je kasnije potrebno ovo prilagoditi, isto je moguće kroz izbornik *Tools→Android→SDK Manager*. Instalacija i postavljanje razvojnog okruženja vrlo je jednostavna uz razumne podrazumijevane postavke te stoga neće biti detaljno opisana, dok je izgled sučelja s naznačenim bitnim dijelovima dan slikom 0.1.

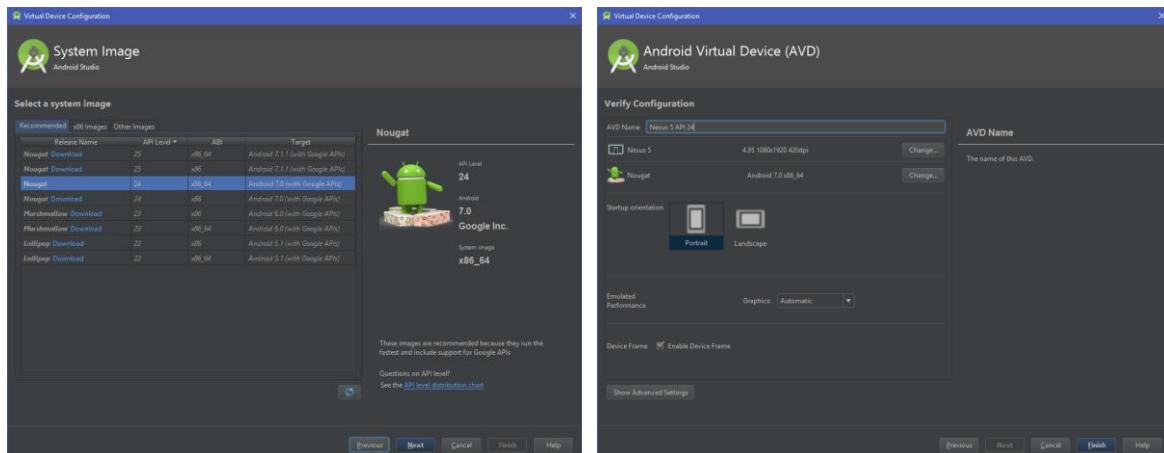
0.2.3. Uredaji

Kako bi se pokrenula Android aplikacija, nužno je rabiti fizički Android uređaj ili kreirati i pokrenuti virtualni Android uređaj (AVD). U prvom će slučaju biti nužni driveri koji uobičajeno dolaze kao dio PC alata za uređaj, a neki proizvođači nude ih i odvojeno. U



Slika 0.4. Kreiranje Android virtualnog uređaja

potonjem slučaju moguće se osloniti na virtualni uređaj koji dolazi kao dio Android SDK, čije je korištenje dano slikom 0.2. Za kreiranje virtualnog uređaja rabi se Android upravitelj virtualnim uređajima (engl. *Virtual device manager*). Najprije se odabire kategorija uređaja te profil sklopoljja. Ponuđeno je nekoliko različitih modela uređaja (Sl. 0.2), a moguće je kreirati i vlastiti uređaj sa specifikacijama po želji korisnika. Nakon odabira uređaja, prelazi se na izbor platforme (Sl. 0.3). Moguće je odabrati neku od platformi koja je preuzeta



Slika 0.3. Kreiranje Android virtualnog uređaja – postavke platforme

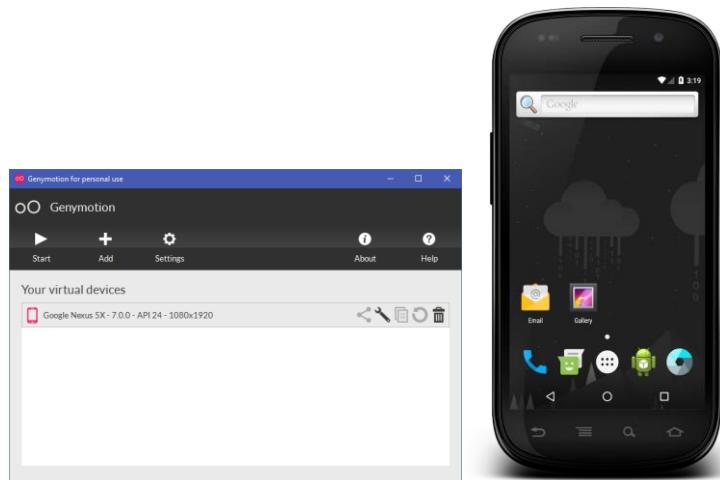
prilikom instalacije AS-a ili preuzeti i dodati novu platformu kroz Android SDK upravitelj (engl. *SDK manager*).



Slika 0.5. Izgled Android virtualnog uređaja

Za kreiranje virtualnog uređaja, nužno je skinuti sliku sustava pomoću gumba download pored pojedine platforme. Kada se odabere željena platforma, moguće je pregledati postavke kao i rabiti grafičku karticu za brži rad emulatora, postaviti parametre bežične mreže uređaja kao i to hoće li se rabiti web kamera kao kamera virtualnog uređaja. Klikom na „Finish“ kreira se virtualni uređaj kojega je moguće pokrenuti AVD upraviteljem.

Moguće je također rabiti emulator koji je razvio netko drugi. Jedan takav emulator prikazan je slikom 0.3. Riječ je o Genymotion emulatoru koji je besplatan za osobnu uporabu, a oslanja se na VirtualBox alat za virtualizaciju računalnih resursa. Virtualni uređaji dolaze u obliku slika (engl. *Image*) koje je moguće preuzeti sa Genymotionove stranice. Riječ je o vrlo moćnom i popularnom emulatoru, koji rabe brojni razvojni timovi diljem svijeta.



Slika 0.6. Izgled Genymotion virtualnog uređaja



Genymotion: <https://www.genymotion.com/>

0.2.4. Grafički elementi

Za pripremu grafičkih elemenata aplikacije moguće je rabiti bilo kakav alat za uređivanje slika. Android podržava i vektorsku grafiku u SVG formatu, a podržava i rastersku grafiku u JPEG i PNG formatima. Vektorsku je grafiku moguće uključiti kroz alat pod nazivom *Vector Asset Studio* koji omogućuje i kreiranje rasterskih grafičkih elemenata prilikom *build* procesa za starije verzije Androida (jer iste ne podržavaju vektorsku grafiku). Što se tiče same pripreme, čest izbor predstavljaju Adobeovi alati, Photoshop i Illustrator, no moguće se poslužiti i alternativama otvorenog koda.



Gimp - <https://www.gimp.org/>



Inkscape - <https://inkscape.org/en/>



Android Asset Studio - <https://goo.gl/bsdXu6>

Uz pripremu slika, važno se voditi i smjernicama dizajna definiranim za platformu. Android od inačice 5.0 rabi takozvani *Material* dizajn.



Material - <https://material.io/guidelines/>

0.2.5. Verzioniranje koda

Vrlo važan dio razvoja programske podrške, ne samo u kontekstu razvoja aplikacija za Android platformu, predstavljaju rad u timu, dijeljenje i održavanje koda. U tu se svrhu koriste različiti alati za verzioniranje koda, među kojima se ističe GIT. Riječ je o distribuiranom sustavu za verzioniranje koji se lako nosi, kako s malim, tako i s velikim projektima. Za potrebe ovih vježbi bit će dovoljno osnovno poznавanje GIT naredbi, kako bi se održavale promjene te omogućilo podizanje i preuzimanje koda s udaljenog repozitorija. Ipak, u svakom se slučaju preporučuje samostalno učenje i naprednjih mogućnosti koje ovaj sustav pruža, s obzirom da je poznавanje nekog od sustava za verzioniranje koda preduvjet za posao *developer*a i rad u razvojnog timu.



GIT - <https://git-scm.com/>



Naučiti GIT - <https://git-scm.com/book/en/v2>



Gitlab - <https://about.gitlab.com/>



Bitbucket - <https://about.gitlab.com/>

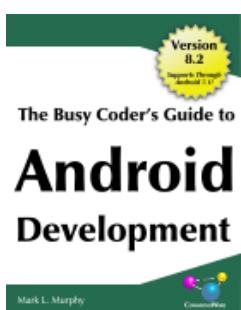


Github - <https://github.com/>

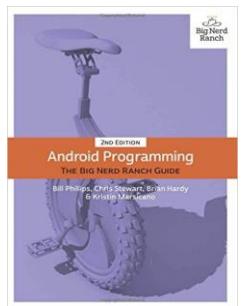
0.3. Dodatni izvori za učenje

Iako će se na brojnim mjestima u ovom priručniku pojaviti poveznice na vanjske resurse, kada to bude potrebno, u ovom potpoglavlju moguće je pronaći dodatne izvore informacija koji mogu pomoći u učenju. S obzirom da se preferirani načini učenja razlikuju od osobe do osobe, resursi su razloženi u knjige, tutorijale i kolegije koji su dostupni. Odmah je u početku važno naglasiti kako je cijelovita lista resursa nemoguć pothvat, a nije bila niti cilj ovog potpoglavlja. Ukoliko ipak smatrate da je neki kvalitetan resurs za učenje nepravedno izostavljen, javite se na ranije navedenu e-mail adresu kako bi bio uvršten.

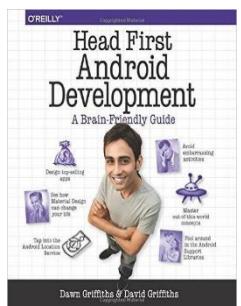
0.3.1. Knjige



S preko dvije stotine poglavlja, svako od kojih sadrži brojne primjere ova knjiga Marka L. Murphya ([Commonsware](https://commonsware.com/Android/)) predstavlja jedan od najboljih resursa za učenje razvoja Android aplikacija. S obzirom da je riječ o digitalnom izdanju, uspijeva pratiti izlaska novih platformi i ponuditi primjere prilagođene promjenama paradigme. Više informacija dostupno je na: <https://commonsware.com/Android/>



Iako zahtijeva određena predznanja o razvoju u objektno orijentiranoj paradigmi, kao i određeno poznavanje Java programskog jezika, knjiga je namijenjena početnicima u razvoju za Android platformu. Kroz brojne primjere kreće se od vrlo jednostavne do složenijih aplikacija. Više je informacija dostupno na: <https://www.amazon.com/Android-Programming-Nerd-Ranch-Guide/dp/0134171454/?tag=mak041-20>.



Knjiga iz serije „*Head first*“ donosi ponešto drugačiji pristup u izlaganju. Cilj joj je predstaviti ponekad vrlo kompleksne informacije na vrlo jednostavan i izravan način. Sadrži brojne primjere, a prezentacija materijala orijentirana je na snažan vizualni dojam. Više informacija dostupno je na: <https://www.amazon.com/Head-First-Android-Development-Brain-Friendly/dp/1449362184>

0.3.2. Kolegiji

- Coursera
 - <https://www.coursera.org/specializations/android-app-development>
 - <https://www.coursera.org/learn/android-programming>
- Edx

- <https://www.edx.org/course/android-app-development-beginners-galileox-caad002x>
- <https://www.edx.org/course/introduction-mobile-application-hkustx-comp107x-1>

0.3.3. Tutorijali

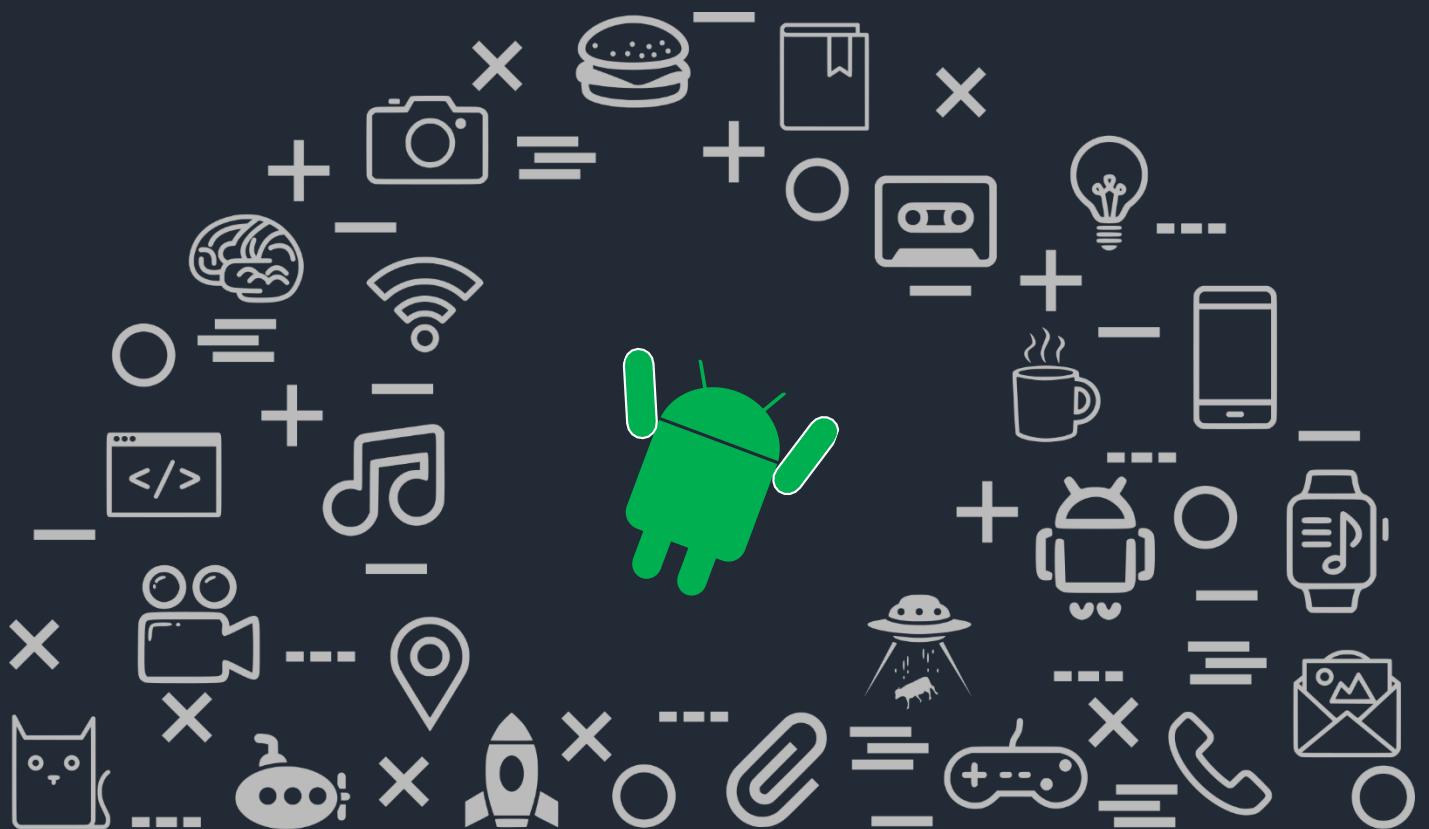
- Vogella - <http://www.vogella.com/tutorials/android.html>
- Android hive - <http://www.androidhive.info/>
- Android developers - <https://developer.android.com/training>
- Lynda - <https://www.lynda.com/Android-training-tutorials/>
- Codementor - <https://www.codementor.io/android/tutorial>
- CodePath - <https://guides.codepath.com/android>

0.3.4. Video

- Google I/O - <https://www.google.com/events/io>
- Android developers - <https://www.youtube.com/user/androiddevelopers>
- The new boston - <https://goo.gl/UUKRcx>

0.3.5. Poveznice

- Stack Overflow - <http://stackoverflow.com/tags/android>
- Android developer - <https://developer.android.com/index.html>
- Developer blogs - <https://android-developers.googleblog.com/>
- Developer resources - <https://goo.gl/IJzAZ7>
- Android arsenal - <https://android-arsenal.com/free>
- Gradle, please - <http://gradleplease.appspot.com/>



„The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time.“

- Tom Cargill



LV1:

Uvod u Android

1. Laboratorijska vježba 1

1.1. Uvod

U ovoj je vježbi prikazano kreiranje prve aplikacije za Android sustav. Razmotreni su njeni osnovni dijelovi, od osnovnih gradivnih elemenata do korisničkog sučelja (engl. *user interface*, UI). Kroz primjere je pokazana izgradnja UI-ja u XML opisnom jeziku (engl. *eXtensible Markup Language*) korištenjem različitih kontrola, povezivanje događaja poput klika na gumb s kodom koji ga obrađuje te pokretanje aplikacije.

1.1.1. Potrebna predznanja

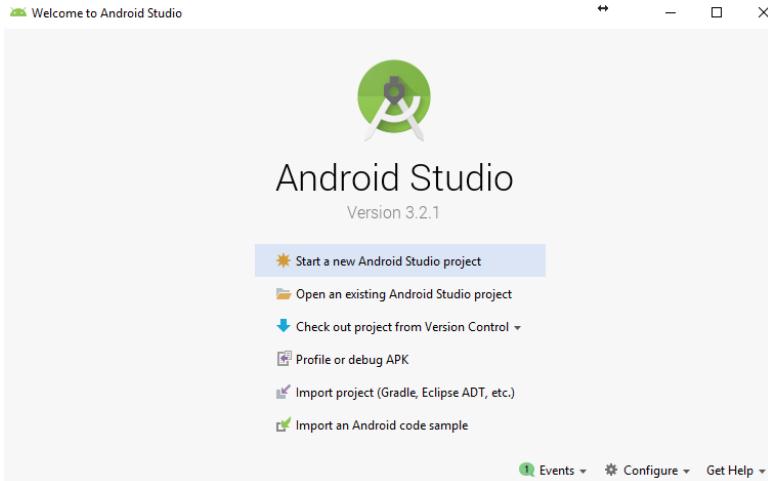
- Osnove programiranja
 - Kolegiji Programiranje I, Programiranje II, Algoritmi i strukture podataka
- Osnove objektno orijentiranog programiranja
 - Kolegij Objektno orijentirano programiranje
- Osnove Java programskog jezika
 - H. Schildt, Java, A Beginner's Guide, 5th Edition
 - B. Eckel, Thinking in Java, 4th edition
 - <http://docs.oracle.com/javase/tutorial/>

1.1.2. Korisna predznanja

- Input Controls, <http://developer.android.com/guide/topics/ui/controls.html>
- Layouts, <http://developer.android.com/guide/topics/ui/declaring-layout.html>
- Resursi, <http://developer.android.com/guide/topics/resources/index.html>
- App manifest, <http://developer.android.com/guide/topics/manifest/manifest-intro.html>
- Activity lifecycle, <http://developer.android.com/training/basics/activity-lifecycle>
- Begginer's guide (I/O), <https://www.youtube.com/watch?v=yqCj83leYRE>
- UI design patterns, <https://www.youtube.com/watch?v=M1ZBjICRfz0>
- Design for UI Devs, <https://www.youtube.com/watch?v=JI3-IzlzOJI>

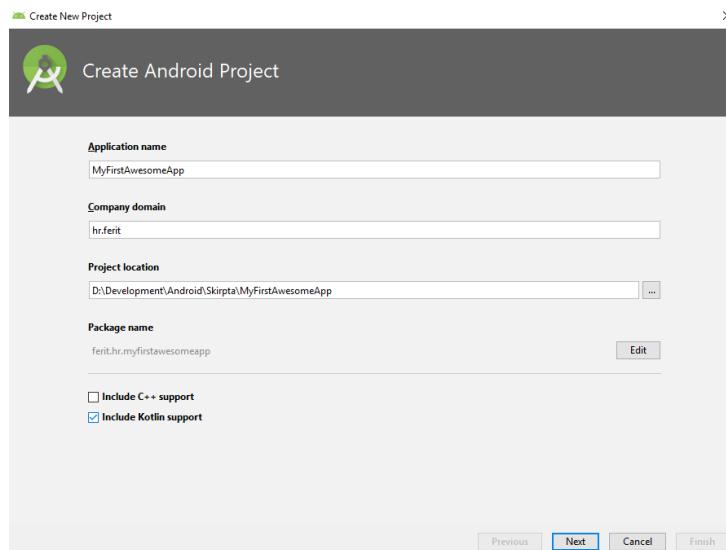
1.2. Kreiranje Android projekta

Kreiranje novog Android projekta moguće je obaviti iz glavnog izbornika kod pokretanja Android studija, kako je prikazano slikom 1.1. Nakon toga korisniku se prikazuje izbornik



Slika 1.2. Kreiranje Android virtualnog uređaja – postavke platforme

nalik onome na slici 1.2., koji omogućuje unos različitih detalja vezanih uz projekt. Prvenstveno se ovdje radi o nazivu aplikacije, nizu znakova koji će korisniku biti prikazani



Slika 1.1. Kreiranje Android virtualnog uređaja – postavke platforme

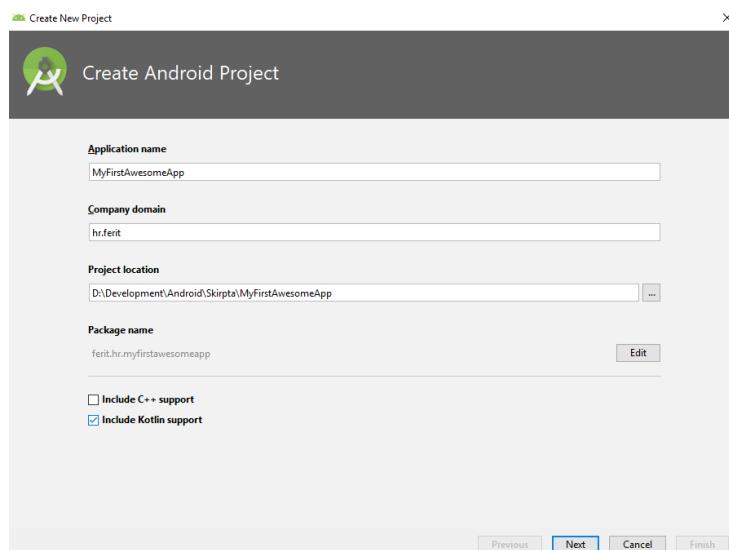
ispod ikone na uređaju, kao i na trgovini aplikacijama. Važno je reći kako ime aplikacije ne mora biti jedinstveno. Ono što mora biti jedinstveno je „*Company domain*“ (CD). Naime, kod je kod Java/Kotlin projekata organiziran u pakete, a osnovni paket mora biti jedinstven kako bi bilo moguće razlikovati projekte, odnosno aplikacije na uređaju (i na Googleovoj Play trgovini). Ime paketa određuje se stoga upravo kao obrnuto ime domene, s ciljem olakšavanja održavanja jedinstvenosti. Ukoliko ne posjedujete domenu, za učenje ovo ne

predstavlja nikakav problem. Možete unijeti bilo koje ime paketa (npr. `com.example.bruno`), a kasnije, ukoliko se odlučite na distribuciju, možete razmišljati o jedinstvenosti. Za potrebe ovih vježbi rabit će se **imeprezime.ferit.hr**. Ovim načinom CD autora ovih redaka bit će **brunozoric.ferit.hr**, dok će ime paketa biti **hr.ferit. brunozoric.myapplication**. Za potrebe ovih vježbi nije potrebno uključivati podršku za C++, jest potrebno uključiti podršku za Kotlin, a lokaciju projekta nužno je postaviti na **D:** disk, u direktorij naziva **RMA**, u poddirektorij vlastitog imena.



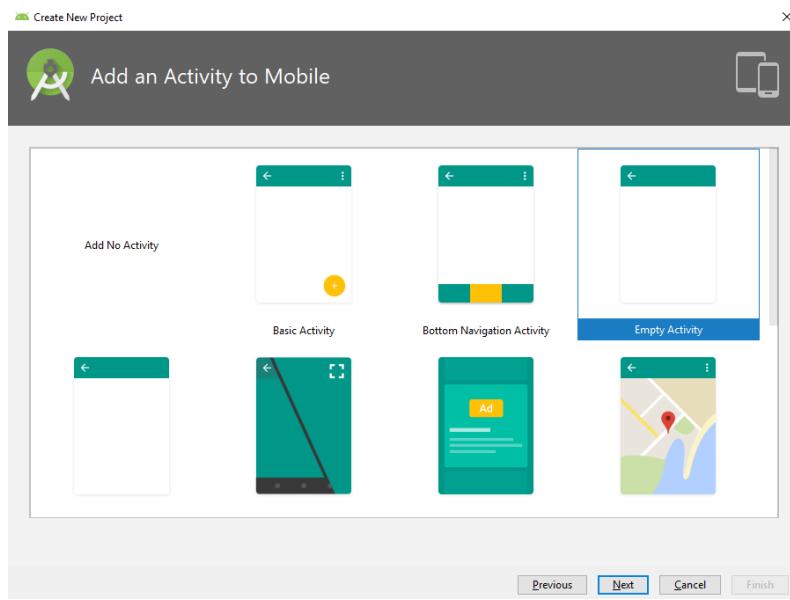
Sve projekte potrebno je u laboratoriju smjestiti u `D:\RMA\ImePrezime`

Idući zaslon, prikazan slikom 1.3., nudi izbor platforme za koju se želi razvijati aplikacija. Android je od originalnog sustava za digitalne kamere prerastao u operacijski sustav za širok spektar najrazličitijih uređaja, od telefona do automobila. Ukoliko se želi razvijati za pametne telefone, odabire se prva opcija i određuje se najniža razina (verzija) Androida koja se želi podržati, kako je prikazano slikom 1.3. Različite inačice Android sustava donosile su različite novitete koji u ranijim inačicama nisu u potpunosti podržani ili je potrebno na njih obratiti posebnu pozornost. Za potrebe ovih vježbi kao najniža inačica rabit će se API razine 16, odnosno najniža podržana verzija Androida bit će 4.1. (*IceCreamSandwich*). Ciljanjem navedene inačice pokriveno je otprilike 99.6% uređaja, dok ostalima aplikacija neće biti vidljiva. Danas je za većinu potreba dovoljno ciljati značajno više inačice Android operacijskog sustava (npr. KitKat ili Marshmallow), no zbog opreme dostupne u laboratoriju, tableta s OS inačicom 4.1. rabit će se upravo API 16.



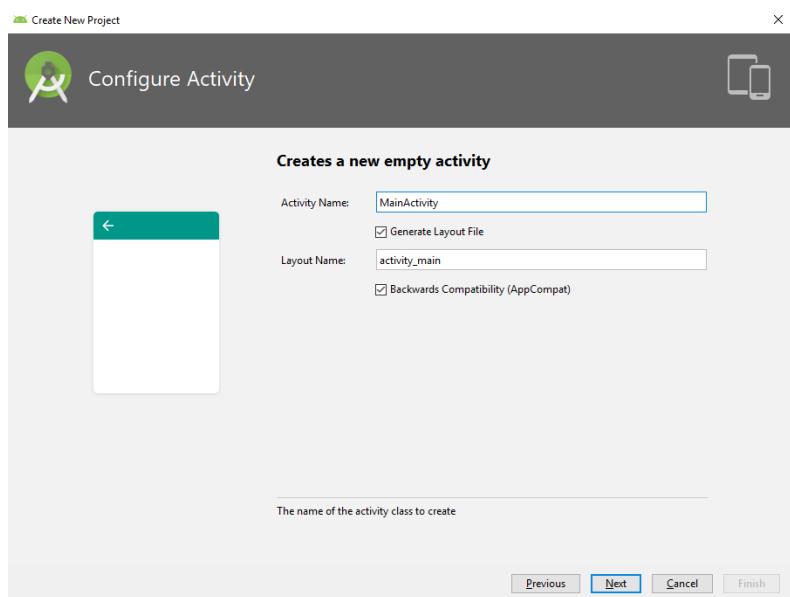
Slika 1.3. Kreiranje Android virtualnog uređaja – postavke platforme

Kod kreiranja novog projekta čarobnjak Android studija omogućuje dodavanje novog *Activitya*. *Activity* je nalik formi u klasičnom programiranju desktop aplikacija, a predstavlja jedan zaslon aplikacije. Ponuđeni su različiti predlošci, čak i mogućnost izostanka *Activitya*, a za potrebe ovih vježbi svi će novi projekti započinjati dodavanjem praznog *Activitya* (*Empty Activity*), kako je označeno slikom 1.4.



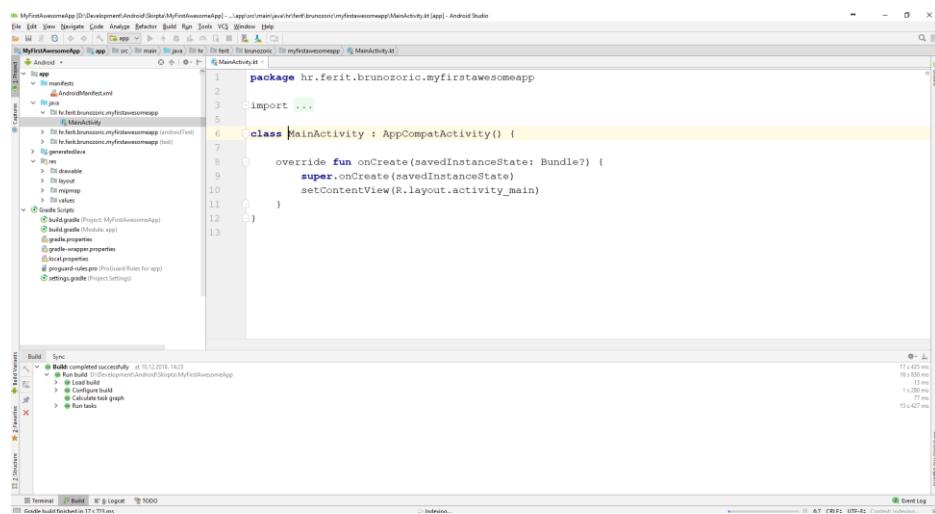
Slika 1.4. Kreiranje Android virtualnog uređaja – postavke platforme

Dodavanjem praznog *Activitya*, IDE generira nužan kod (zapis u manifest datoteci, kreiranje nove klase itd.), no ipak je nužno unijeti neke osnovne informacije, kako je prikazano slikom 1.5.



Slika 1.5. Kreiranje Android virtualnog uređaja – postavke platforme

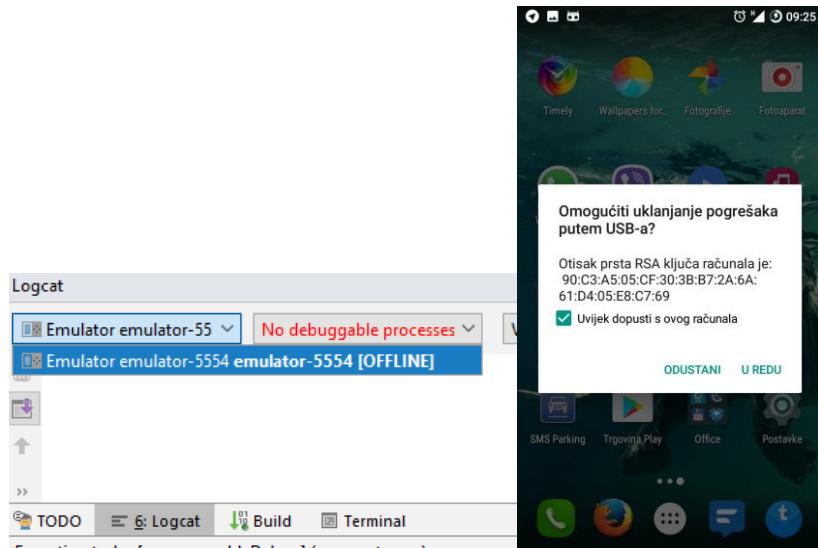
Naime, potrebno je reći naziv *Activity* i naziv datoteke koja definira njegov izgled. Imenom *Activity* određuje se ime klase koja predstavlja taj *Activity* i unutar koje će biti pisan kod, a s obzirom da Android sustav razdvaja funkcionalnost od prikaza, u datoteci koja definira izgled (*layout*) uporabom XML opisnog jezika bit će definirani svi vizualni elementi (kontrole na korisničkom sučelju). Za sada je dovoljno ostaviti sve na podrazumijevanim postavkama. Klikom na *Finish* kreiran je (nakon kraćeg čekanja) prvi projekt koji ima osnovnu funkcionalnost, ispisuje na zaslonu poruku „Hello world“. Izgled projekta u Android studiju prikazan je slikom 1.6.



Slika 1.6. Kreiranje Android virtualnog uređaja – postavke platforme

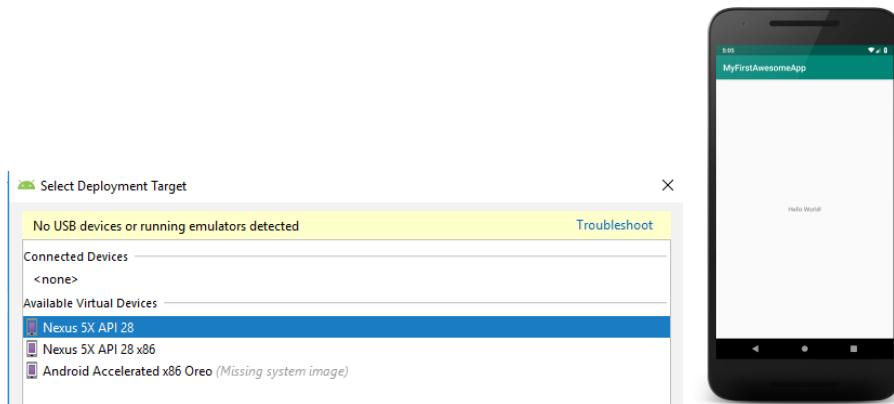
1.2.1. Pokretanje Android aplikacije

Da bi se Android projekt pokrenuo, potrebno je imati spojen i uključen uređaj. Ovdje može biti riječ ili o stvarnom ili o virtualnom uređaju. Svi spojeni uređaji dostupni preko ADB-a vidljivi su unutar Android monitora na dnu sučelja razvojnog okruženja, kako je prikazano slikom 1.7. Virtualni uređaji radit će bez ikakvih problema, dok je za korištenje fizičkih uređaja potrebno dati dozvolu na samom uređaju. Klikom na gumb u alatnoj traci ili pritiskom na kombinaciju tipku *Shift* i *F10* pokreće se aplikacija i nudi se izbor uređaja na



Slika 1.7. Odabir uređaja

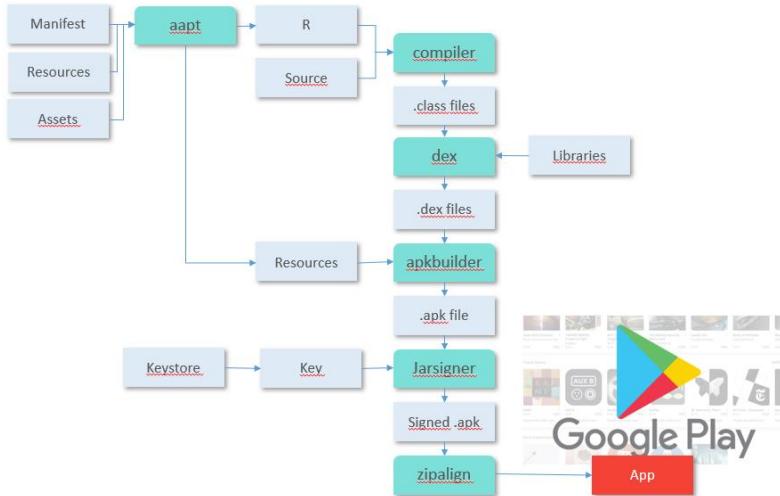
kojem ju se želi pokrenuti. Ovakvim pokretanjem koristi se podrazumijevana konfiguracija za pokretanje, dok je istu moguće urediti ili pak kreirati potpuno novu korištenjem *Run→EditConfigurations* izbornika. Slikom 1.8. prikazan je izbornik uređaja, a uređaj je moguće učiniti podrazumijevanim tako da svako iduće pokretanje aplikacije bude na istom uređaju. Aplikacija u radu prikazana je istom slikom.



Slika 1.8. Odabir uređaja

Iako se cijelokupni postupak izrade aplikacije čini vrlo jednostavan, u pozadini se odvija velik broj koraka koje IDE prikriva, obavi i njima ne opterećuje programera. Sustav za izgradnju (engl. *build system*) odgovoran je za sve od prevođenja izvornog koda do pakiranja aplikacije, a čini to uporabom različitih alata kojima upravlja (prevoditelj, povezivač i sl.) te skripti za izgradnju. Sav izvorni kod prevodi se u Java class datoteke koje se zatim pretvaraju u .dex datoteke (Dalvik executable) uporabom alata *dx*. Ova se datoteka zatim zajedno sa svim resursima pakira uporabom alata AAPT pakiraju u .apk datoteku. Ova se datoteka u slučaju *debug* inačice potpisuje takozvanim *debug* ključem što omogućuje da ju se uporabom alata *adb* postavi i instalira na uređaj. Od verzije

Androida 5.0 koristi se ART, pa postoje neke razlike u ranije opisanom postupku. Prikaz cjelokupnog postupka prevođenja aplikacije skiciran je slikom 1.9.



Slika 1.9. Odabir uređaja



Više informacija o gradleu moguće je pronaći na

<https://rominirani.com/announcing-gradle-tutorial-series-5fd134223bf8#.gwyelvyc1>

1.3. Manifest

Jedna od temeljnih datoteka unutar Android aplikacije jest Manifest datoteka, koja se nalazi unutar *app->manifests* mape u strukturi projekta. Riječ je o datoteci pisanoj XML opisnim jezikom koja sadrži meta-podatke o aplikaciji, definira strukturu aplikacije, sve aktivnosti, servise, pružatelje sadržaja, dozvole, definira ikonu, verziju aplikacije i slično. Izgled osnovne manifest datoteke dan je primjerom 1. Pregledom datoteke uočava se osnovni manifest element koji govori da se radi o manifest datoteci. Gotovo svaki gradivni element aplikacije mora biti prijavljen i definiran u ovoj datoteci.



Primjer 1. - Manifest datoteka

Prikaz manifest datoteke, njena smještaja u strukturi projekta te njena sadržaja za osnovnu aplikaciju s jednim Activityem.

The screenshot shows the Android Studio project structure. The main window displays the project tree under the 'app' module. The 'manifests' folder is expanded, showing the 'AndroidManifest.xml' file, which is highlighted with a blue selection bar. To the right of the project tree, a list of Gradle scripts is shown:

- Gradle Scripts
 - build.gradle (Project: RMA)
 - build.gradle (Module: app)
 - gradle-wrapper.properties (Gradle Version)
 - proguard-rules.pro (ProGuard Rules for app)
 - gradle.properties (Project Properties)
 - settings.gradle (Project Settings)
 - local.properties (SDK Location)

Gradle skripta

Gradle skripta:

```
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "brunozoric.ferit.hr.myfirstawesomeapp"
        minSdkVersion 16
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        // omitted for brevity...
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="brunozoric.ferit.hr.myfirstawesomeapp">

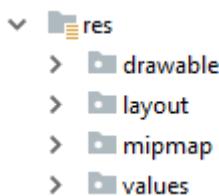
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">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

Unutar oznake manifest smještaju se nužne informacije. Prvenstveno se ovoj oznaci pridodaje atribut *package* koji predstavlja ID aplikacije kao i mjesto gdje će biti smješten generirani kod, poput primjerice R klase s resursima. Važno je još jednom naglasiti kako na trgovini Play ne mogu postojati dvije aplikacije koje imaju isti ID aplikacije. Unutar oznaka manifest moguće je navesti i minimalnu podržanu verziju SDK kao i ciljanu verziju. Kod Android studija potonji se detalji navode u *gradle build* skripti. U istoj skripti navode se trenutna inačica aplikacije, i to na dva načina. Ime verzije predstavlja korisniku čitljiv niz znakova, dok kod verzije predstavlja cjelobrojnu vrijednost koja se ne prikazuje korisnicima, ali ju primjerice trgovina *Play* rabi za provjeru i ažuriranje aplikacije. Informacije o verziji mogu se navesti i u manifest datoteci. U manifest datoteci nalazi se potom oznaka *application* unutar koje se definira tema, ikona, te se registrira svaka komponenta aplikacije. U trenutnoj inačici postoji samo jedan *Activity*, a uočljivo je kako je dano ime (*.MainActivity*) te je uporabom *intent filter-a* navedeno da se ova aktivnost

pokreće prilikom pokretanja aplikacije. U slučaju da se nove komponente dodaju uporabom čarobnjaka Android studija, manifest datoteka bit će uglavnom automatski ažurirana.

1.4. Resursi

Kod razvoja aplikacija za Android platformu, kod se u pravilu razdvaja od izgleda, slika i drugih podataka koji se zajednički nazivaju resursima. Ovo razdvajanje u značajnoj mjeri olakšava izmjene, ali i prilagodbe različitim uređajima, tržištima i slično, jer je vrlo lako imati više različitih inačica istog resursa. Ove prilagodbe sustav će odraditi bez zahtjeva za dodatnim pisanjem koda ili intervencijama programera. Resursi za aplikaciju smješteni su u *res* mapi unutar projekta, a pakiraju se u .apk datoteku prilikom njena generiranja.



Slika 1.10. Resursi unutar Android aplikacije

1.4.1. Izgled

Izgled Android aplikacija definira se resursima koji se nazivaju *layout*. Riječ je o datotekama napisanim uporabom XML opisnog jezika unutar kojih se definiraju elementi korisničkog sučelja. Unutar aplikacije se ovi elementi „napuhuju“ (engl. *Inflate*) te se na temelju njih stvaraju objekti poput gumba, polja za unos i sl. sa svojstima definiranim u *layout* datoteci. Svaki layout sadrži jedan korijenski element, koji mora biti ili *View* ili *ViewGroup* (derivat *View* klase, implementiraju *ViewManager* sučelje) objekt. U korijenski se element mogu zatim ugnijezditi drugi elementi, odnosno moguće je izgraditi hijerarhiju koja sačinjava korisničko sučelje (engl. *user interface*, UI). Čest slučaj predstavlja korištenje nekog objekta klase *ViewGroup* kao korijenskog elementa, s obzirom da je riječ o takozvanim *layout managerima*. Odnosno, oni nude mogućnost smještanja i organizacije drugih *View* objekata unutar sebe.

1.4.1.1. View

View je osnovna klasa koja predstavlja UI kontrolu. Iz nje su izvedene brojne korisne kontrole poput *TextViewa*, *EditTexta*, *ImageViewa* i slično. Osnovne i najčešće korištene klase koje su derivati *Viewa* bit će dane u nastavku, dok je za potpunu listu i opis potrebno pogledati u dokumentaciju. Kod dodavanja bilo kakvih *View* objekata u *layout* datoteku, obavezno je navesti njihove dimenzije. Dimenzije se mogu dati eksplicitno ili putem dviju vrijednosti koje ih prilagođavaju roditelju ili sadržaju, a riječ je o *match_parent* i

wrap_content. Detaljnije će ovo biti prikazano u nastavku. Dodatno svojstvo koje se u pravilu dodjeljuje jest *id*. Uporabom ovog svojstva moguće je programski pristupiti elementu sučelja, ali i unutar jednog resursa referencirati drugi (primjerice slika na gumbu), a dodaje se tako da se u XML-u svojstvo *id* postavi na „@+id/some_custom_id“.



Više informacija o *View* klasi i njenim derivatima moguće je pronaći na
<https://developer.android.com/reference/android/view/View.html>

- **TextView**

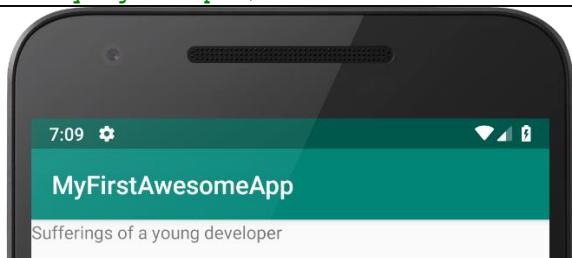
TextView je klasa koja predstavlja UI kontrolu za prikaz tekstualnog sadržaja koji se ne može izravno uređivati (zapravo može, ali samo u ograničenoj mjeri i to mu nije osnovna namjena). Ekvivalent je labeli iz desktop aplikacija. Sadrži brojna svojstva, među kojima je ključna *text*, a primjer njegova korištenja dan je primjerom 2.

 Primjer 2. – TextView

U XML datoteku dodaje se oznaka `<TextView>`. Istu je moguće zatvoriti odgovarajućom zatvarajućom oznakom, ili koristiti `</>`. Parametrima *layout_width* i *layout_height* definiraju se dimenzije koje govore da širina treba biti jednaka širini roditelja (*View* objekta u koji je smješten *TextView*), dok se visina treba prilagoditi visini sadržaja. Svojstvo *gravity* govori da se tekst treba smjestiti u središte kontrole, dok svojstvo *text* određuje sadržaj. Ovakav način navođenja sadržaja nije dobar, već se tekst treba izdvojiti kao zaseban resurs, što će biti kasnije pokazano. **Napomena:** Ovaj *TextView* smješten je unutar linearног *layouta*, što će biti pojašnjeno u kasnijem primjeru.

Layout:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    android:id="@+id>ShowTitle"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Sufferings of a young developer"/>
```



- **EditText**

EditText je klasa koja nasljeđuje *TextView*, a riječ je o UI kontroli za unos alfanumeričkih znakova. Moguće ga je postaviti da dozvoljava samo određenu vrstu unosa (primjerice, samo brojeve), da prikazuje zamaskirani unos lozinke (****), da prikazuje prijedloge (engl. *hint*) itd. Primjer korištenja *EditText* kontrole dan je primjerom 3. Važna svojstva su *hint*

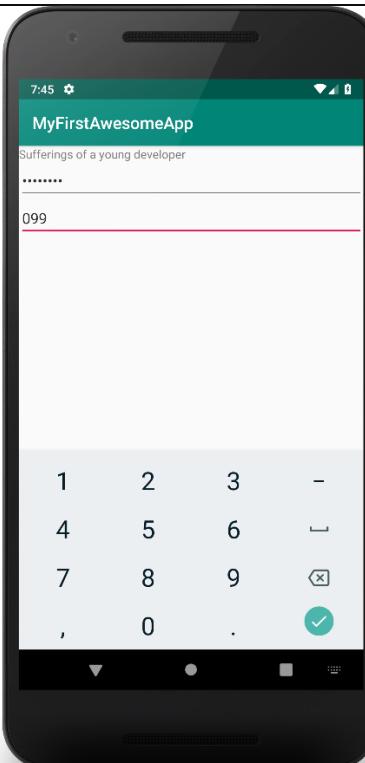
koji definira naputak korisniku te *inputType* koji govori kakav će biti tip unosa podataka što će primjerice utjecati na prikaz ili na oblik tipkovnice koja će se ponuditi korisniku.

</>Primjer 3. – EditText

U XML datoteku dodaje se oznaka `<EditText>`. Istu je moguće zatvoriti odgovarajućom zatvarajućom oznakom, ili koristiti `/>`. Parametrima `layout_width` i `layout_height` definiraju se dimenzije koje govore da širina treba biti jednaka širini roditelja (*View* objekta u koji je smješten *EditText*), dok se visina treba prilagoditi visini sadržaja. Svojstvo `hint` govori koji se tekst treba prikazati dok je kontrola prazna, dok svojstvo `inputType` određuje vrstu sadržaja. Kako je vidljivo sa slike, lozinka se ne prikazuje, dok je za unos telefonskog broja prikazana numerička tipkovnica. Kao i u prethodnom primjeru, kontrole su smještene u linearni `layout`.

Layout:

```
<TextView  
    android:id="@+id/textview_main_showtitle"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Sufferings of a young developer"/>  
<EditText  
    android:id="@+id/edittext_main_password"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Enter password"  
    android:inputType="textPassword"/>  
<EditText  
    android:id="@+id/edittext_main_phone"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="Enter phone number"  
    android:inputType="numberDecimal"/>
```



- **ImageView**

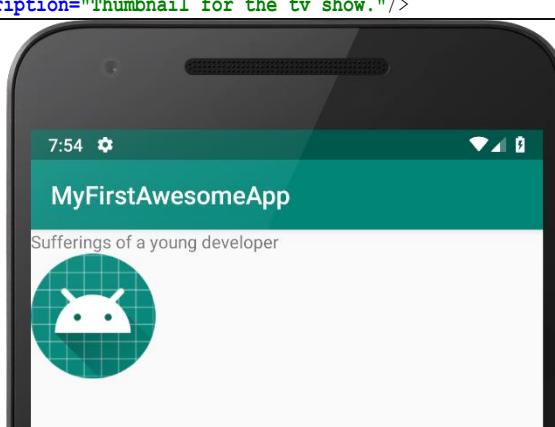
Klasa *ImageView* predstavlja UI kontrolu za prikaz slika na zaslonu. Nudi mogućnost prikaza slika iz različitih izvora, omogućuje njen prikaz u bilo kojem *layout* upravitelju te nudi brojne mogućnosti vezano uz prikaz slike. Ključan atribut u ovom slučaju je *src* koji govori iz kojeg se izvora dohvata slika. Odabrana vrijednost u primjeru 4 govori da se prikazuje ikona aplikacije. Ona se nalazi u resursima, konkretno u *mipmap* mapi i zove se *ic_launcher*. Ovo se i navodi korištenjem @ simbola i navođenjem identifikatora resursa, koji je u ovom slučaju njegovo ime.

Primjer 4. – ImageView

U XML datoteku dodaje se oznaka *<ImageView>*. Istu je moguće zatvoriti odgovarajućom zatvarajućom oznakom, ili koristiti *>*. Parametrima *layout_width* i *layout_height* definiraju se dimenzije koje govore da širina treba biti jednaka širini roditelja (*View* objekta u koji je smješten *ImageView*), dok se visina treba prilagoditi visini sadržaja. Svojstvo *src* govori koja se slika treba prikazati, dok svojstvo *contentDescription* određuje opis slike u slučaju da istu nije moguće prikazati. Svojstvo *scaleType* govori na koji način će slika biti skalirana u slučaju da njene dimenzije ne odgovaraju dimenzijama kontrole. Kao i u prethodnom primjeru, kontrole su smještene u linerni *layout*.

Layout:

```
<ImageView  
    android:id="@+id/imageview_main_showicon"  
    android:layout_width="100dp"  
    android:layout_height="100dp"  
    android:src="@mipmap/ic_launcher"  
    android:scaleType="centerCrop"  
    android:contentDescription="Thumbnail for the tv show."/>
```



- **Button**

Klasa *Button* predstavlja standardni gumb pritiskom na koji je moguće obaviti neku radnju. Stil gumba moguće je definirati unutar datoteke koja određuje stilove, ukoliko podrazumijevani izgled iz bilo kojeg razloga nije zadovoljavajuće rješenje. Definiranje

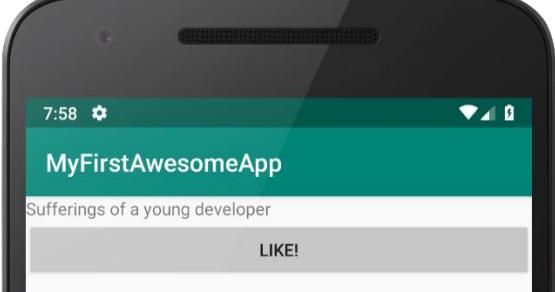
ponašanja moguće je postići na više različitih načina, neki od kojih su opisani u kasnijem poglavlju. Primjer korištenja gumba dan je primjerom 5.

Primjer 5. – Button

U XML datoteku dodaje se oznaka `<Button>`. Istu je moguće zatvoriti odgovarajućom zatvarajućom oznakom, ili koristiti `/>`. Parametrima `layout_width` i `layout_height` definiraju se dimenzije koje govore da širina treba biti jednaka širini roditelja (`View` objekta u koji je smješten `Button`), dok se visina treba prilagoditi visini sadržaja. Svojstvo `text` govori koji se tekst treba prikazati na površini gumba, dok svojstvo `onClick` određuje metodu koja će biti pozvana pritiskom na gumb. Ova metoda mora imati točno određen potpis, a kao argument prima objekt `View` klase koji je generirao događaj. Kao i u prethodnom primjeru, kontrole su smještene u linerni `layout`.

Layout:

```
<Button  
    android:id="@+id/button_main_like"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="Like!"  
    android:onClick="onLikeClick"/>
```



1.4.1.1. ViewGroup

Klasa `ViewGroup` izvedena je iz klase `View`, a klase koje nju nasleđuju su upravitelji `layouta`. Objekti ovih klasa omogućuju unutar sebe organiziranje i grupiranje drugih `View` (ili `ViewGroup`) objekata. Uz nekoliko opisanih klasa izvedenih iz klase `ViewGroup`, postoje i druge, dio kojih će biti prikazan u kasnijim vježbama (primjerice `RecyclerView`).

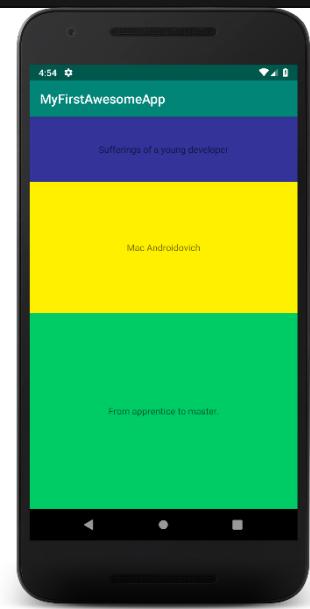
- **LinearLayout**

Linearni `layout` je jedan od najjednostavnijih i osnovnih `layouta`. Elementi se unutar ovog `layouta` slažu jedan ispod drugog, ako je orijentacija postavljena na vertikalnu, ili jedan pored drugoga, ako je orijentacija postavljena na horizontalnu. Ako je potrebno rasporediti elemente unutar ovog `layouta` tako da neki od njih zauzimaju određeni udio ekrana, to je moguće učiniti uporabom svojstva `weight`. Prikaz linearne `layouta` i načina korištenja dan je primjerom 6.



Primjer 6. – Linear Layout

U XML datoteku dodaje se oznaka `<LinearLayout>`. Istu je moguće zatvoriti odgovarajućom zatvarajućom oznakom. Parametrima `layout_width` i `layout_height` definiraju se dimenzije koje govore da i širina i visna trebaju biti jednaki širini roditelja. S obzirom da je ovo korjenski element, on će zauzeti čitav ekran. Orientacija je vertikalna, pa će svi elementi biti složeni jedan ispod drugoga. Svojstvo koje određuje težinu svakog elementa, i to tako da gornji zauzima $1/(1+3+2)$ dijela ekrana, središnji $3/6$, a doljnji $2/6$ dijelova ekrana. Kao što je jasno iz prethodno navedenog, svaki od elemenata uzima onoliko dijelova u ukupnoj sumi svih težina kolika je njegova težina.



Layout:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textview_main_showtitle"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="center"
        android:background="#333399"
        android:text="Sufferings of a young developer"/>
    <TextView
        android:id="@+id/textview_main_showcast"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:gravity="center"
        android:background="#FFF000"
        android:text="Mac Androidovich"/>
    <TextView
        android:id="@+id/textview_main_showdescription"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="3"
        android:gravity="center"
        android:background="#00cc66"
        android:text="From apprentice to master."/>
</LinearLayout>
```

▪ RelativeLayout

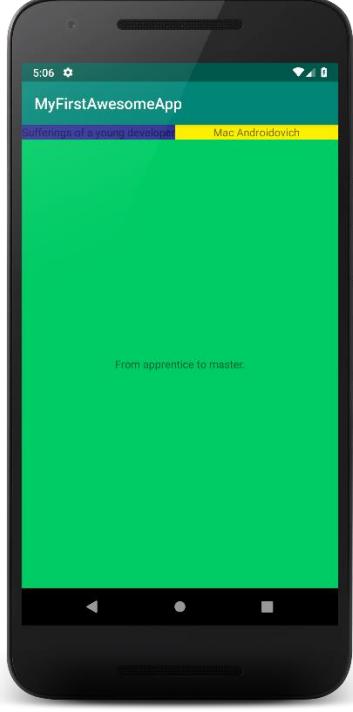
Za razliku od linearnog, relativni `layout` omogućuje smještanje elemenata unutar sebe relativno u odnosu na druge elemente ili u odnosu na vlastite točke vezanja (primjerice,

gore lijevo). Kako bi se pojedini elementi mogli smjestiti u odnosu na druge elemente koriste se njihovi identifikatori. Prikaz korištenja relativnog *layouta* dan je primjerom 7.

</>

U XML datoteku dodaje se oznaka <RelativeLayout>. Istu je moguće zatvoriti odgovarajućom zatvarajućom oznakom. Parametrima *layout_width* i *layout_height* definiraju se dimenzije koje govore da i širina i visna trebaju biti jednaki širini roditelja. S obzirom da je ovo korjenski element, on će zauzeti čitav ekran. Svakom se elementu zadaju točke roditelja u odnosu na koje se pozicionira, kao i odnosi prema drugim elementima. Da bi se navedeno postiglo, nužno je odrediti id svojstva za pojedine elemente.

Primjer 7. – Relative Layout



Layout:

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

    <TextView
        android:id="@+id/textview_main_showtitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:gravity="center"
        android:background="#333399"
        android:text="Sufferings of a young developer"/>

    <TextView
        android:id="@+id/textview_main_showcast"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@+id/textview_main_showtitle"
        android:gravity="center"
        android:background="#FFF000"
        android:text="Mac Androidovich"/>

    <TextView
        android:id="@+id/textview_main_showdescription"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/textview_main_showtitle"
        android:layout_alignParentBottom="true"
        android:gravity="center"
        android:background="#00cc66"
        android:text="From apprentice to master."/>

</RelativeLayout>
```

▪ ConstraintLayout

Novi *layout manager* naziva ConstraintLayout postao je standardni način organiziranja sadržaja na ekranima aplikacija radi fleksibilnosti i uklanjanja potrebe za prekomjernim ugnježđivanjem elemenata. Riječ je o organizatoru sadržaja koji dolazi kao dio biblioteke podrške i koji za organizaciju sadržaja unutar sebe i u odnosu na druge prisutne elemente rabi takozvane „constraintove“. Kako bi se uopće mogao rabiti, potrebno je u *gradle* datoteku modula dodati vanjsku ovisnost, kako je prikazano primjerom 8, a pojašnjeno u kasnijem potpoglavlju ove vježbe.



Više informacija o *ConstraintLayout*-u i njegovu korištenju moguće je pronaći na
<https://developer.android.com/reference/android/support/constraint/ConstraintLayout>
<https://android.jlelse.eu/constraints-layout-best-layout-ever-230175272c0f>



Primjer 8. – Constraint Layout

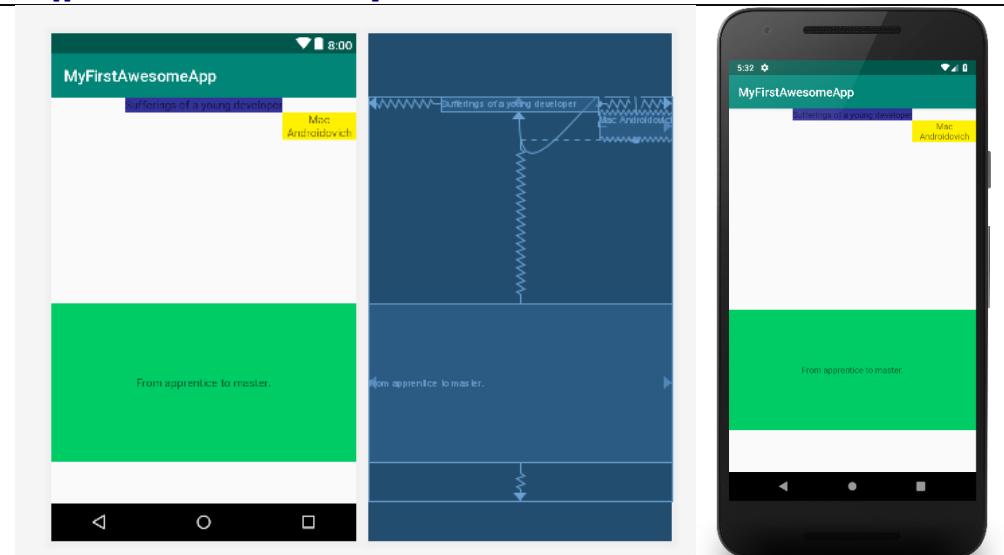
U XML datoteku dodaje se oznaka `<ConstraintLayout>`. Istu je moguće zatvoriti odgovarajućom zatvarajućom oznakom. Parametrima *layout_width* i *layout_height* definiraju se dimenzije koje govore da i širina i visina trebaju biti jednaki širini roditelja. S obzirom da je ovo korijenski element, on će zauzeti čitav ekran. Svakom se elementu mogu dodati točke kojima se postavljaju „sidra“ za njegove dijelove. Primjerice, ako se za prvi *textview* zada *layout_constraintTop_toTopOf="parent"*, ovo označava da je vrh elementa vezan za vrh roditelja (roditelj je ovdje *ConstraintLayout*). Ako se zadaju *constraintovi* i za vrh i za dno, onda će se element postaviti u središte prikaza, a ako je u istom slučaju zadana visina 0dp, element će zauzeti sav slobodan prostor. Moguće je zadati i pristranost prema nekom od *constraintova*, i to vertikalnu i horizontalnu.

Layout:

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <TextView
        android:id="@+id/textview_main_showtitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:gravity="center"
        android:background="#333399"
        android:text="Sufferings of a young developer"/>
    <TextView
        android:id="@+id/textview_main_showcast"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/textview_main_showtitle"
        app:layout_constraintLeft_toRightOf="@+id/textview_main_showtitle"
        app:layout_constraintRight_toRightOf="parent"
        android:gravity="center"
        android:background="#FFFF00"
        android:text="Mac Androidovich"/>
    <TextView
        android:id="@+id/textview_main_showdescription"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        app:layout_constraintTop_toBottomOf="@+id/textview_main_showcast"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintVertical_bias="0.8"
        android:gravity="center"
        android:background="#00cc66"
        android:text="From apprentice to master."/>
</android.support.constraint.ConstraintLayout>

```



1.4.1. Stringovi

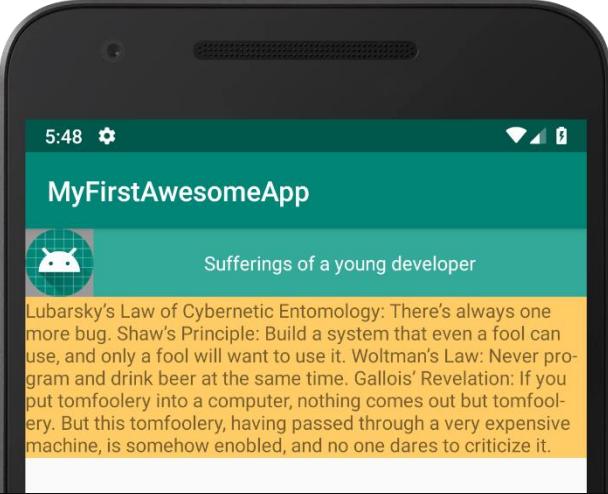
Dobra praksa prilikom razvoja aplikacija jest razdvajanje stringova od koda i prikaza. Naime, česta je potreba distribuirati aplikaciju na različitim jezičnim područjima pa se

stoga pojavljuje potreba za prevodenjem (lokalizacijom). Ako je sav tekst, uključujući onaj koji se pojavljuje na elementima UI-ja, izdvoji u posebnu datoteku, onda je dovoljno prevoditeljima dati tu datoteku i oni ne moraju brinuti o detaljima sučelja. Dodatno, programeri ne moraju voditi računa (osim u specifičnim situacijama) o učitavanju odgovarajuće skupine stringova, već će to učiniti sam sustav u ovisnosti o postavljenom *localeu*. U slučaju stringova, ali i drugih resursa poput boja, dimenzija i sl. nije potrebno posebno dodavati id svojstvo, već se ime samog resursa rabi kao id. Primjer izdvajanja stringova u posebnu datoteku i njihova referenciranja unutar *layouta* dan je primjerom 9.

A small circular icon representing Android Studio or Java development, featuring a stylized '</>' symbol inside a purple circle.

Primjer 9. - Izdvajanje stringova

U primjeru se rabi jednostavno sučelje kreirano u ConstraintLayoutu. Stringovi se izdvajaju kao resursi u strings.xml datoteku, smještenu u res/values mapu unutar hijerarhije projekta. Svaki je string smješten unutar oznaka <string> i </string>, a unutar otvarajuće oznake dodjeljeno mu je svojstvo name. Upravo ovo svojstvo predstavlja jedinstveni identifikator resursa. Stringovima sada pristupa preko njihova imena korištenjem oznake @.



The screenshot shows a mobile application interface. At the top is a green header bar with the text "MyFirstAwesomeApp". Below it is a white area containing a small Android icon in a rounded square and the text "Sufferings of a young developer". The main content area has a white background with a yellow rectangular callout box containing text. The text in the callout box reads:
Lubarsky's Law of Cybernetic Entomology: There's always one more bug. Shaw's Principle: Build a system that even a fool can use, and only a fool will want to use it. Wolman's Law: Never program and drink beer at the same time. Gallois' Revelation: If you put tomfoolery into a computer, nothing comes out but tomfoolery. But this tomfoolery, having passed through a very expensive machine, is somehow ennobled, and no one dares to criticize it.

Strings:

```
<resources>
    <string name="app_name">MyFirstAwesomeApp</string>
    <string name="main_thumbnaildescription">Show thumbnail.</string>
    <string name="main_showtitle">Sufferings of a young developer</string>
    <string name="main_showdescription">Lubarsky...</string>
</resources>
```

Layout:

```

<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <ImageView
        android:id="@+id/imageview_main_showthumbnail"
        android:layout_width="50dp"
        android:layout_height="50dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        android:background="#999999"
        android:src="@mipmap/ic_launcher"
        android:contentDescription="@string/main_thumbnaildescription"/>
    <TextView
        android:id="@+id/textview_main_showtitle"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="@+id/imageview_main_showthumbnail"
        app:layout_constraintLeft_toRightOf="@+id/imageview_main_showthumbnail"
        app:layout_constraintRight_toRightOf="parent"
        android:gravity="center"
        android:background="#33AA99"
        android:textColor="#FFFFFF"
        android:text="@string/main_showtitle"/>
    <TextView
        android:id="@+id/textview_main_showdescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageview_main_showthumbnail"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintVertical_bias="0"
        android:background="#FFCC66"
        android:text="@string/main_showdescription"/>
</android.support.constraint.ConstraintLayout>

```

1.4.1. Boje

Boje predstavljaju još jedan važan resurs koji je moguće izdvojiti. Ovim načinom moguće je rabiti boje prema smislenim imenima koje imaju značaj u kontekstu aplikacije, izbjegava se nečitak kod te je uvelike olakšana izmjena sheme boja u slučaju redizajna. Primjer korištenja boja dan je primjerom 11.



Moguće je pronaći i gotove xml datoteke s bojama, poput primjerice
<https://gist.github.com/kalehv/bae765c756e94455ed88>



Primjer 10. - Izdvajanje boja

Boje se izdvajaju kao resursi u colors.xml datoteku, smještenu u res/values mapu unutar hijerarhije projekta. Svaka je boja smještena unutar oznaka `<color>` i `</color>`, a unutar otvarajuće oznake dodjeljeno joj je svojstvo *name*. Upravo ovo svojstvo predstavlja jedinstveni identifikator resursa. Za kreirani projekt već su definirane neke boje, poput primjerice „colorPrimary“ i „colorAccent“, koje se koriste u resursnoj datotesi koja definira stil aplikacije.

Colors:

```

<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="color_background">#999999</color>
    <color name="color_titlebackground">#33AA99</color>
    <color name="color_titletext">#FFFFFF</color>
    <color name="color_contenttext">#FFcc66</color>
</resources>

```

Layout:

```

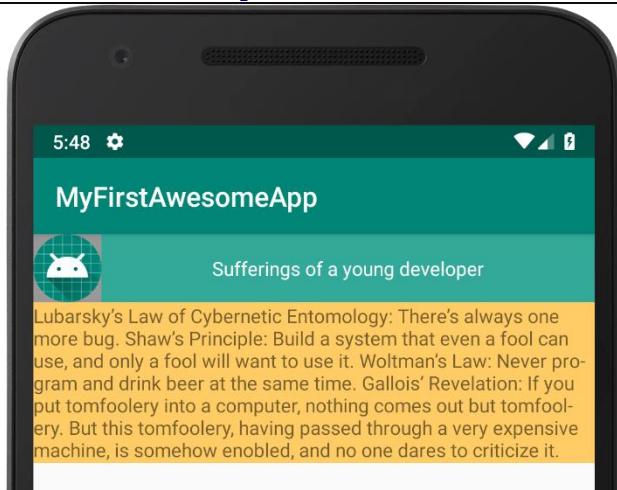
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <ImageView
        android:id="@+id/imageview_main_showthumbnail"
        android:layout_width="50dp"
        android:layout_height="50dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        android:background="@color/color_background"
        android:src="@mipmap/ic_launcher"
        android:contentDescription="@string/main_thumbnaildescription"/>

    <TextView
        android:id="@+id/textview_main_showtitle"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="@+id/imageview_main_showthumbnail"
        app:layout_constraintLeft_toRightOf="@+id/imageview_main_showthumbnail"
        app:layout_constraintRight_toRightOf="parent"
        android:gravity="center"
        android:background="@color/color_titlebackground"
        android:textColor="@color/color_titletext"
        android:text="@string/main_showtitle"/>

    <TextView
        android:id="@+id/textview_main_showdescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageview_main_showthumbnail"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintVertical_bias="0"
        android:background="@color/color_contenttext"
        android:text="@string/main_showdescription"/>
</android.support.constraint.ConstraintLayout>

```



1.4.1. Dimenzije

Dimenzije se u Android aplikacijama mogu definirati na nekoliko načina. Već su ranije spomenute dimenzije koje pojedini element prilagođavaju roditelju ili sadržaju, a riječ je o često korištenim izborima. Ukoliko se želi eksplisitno reći kolika je dimenzija nekog elementa, onda se to čini u jedinicama pod nazivom *density independent pixel* za sve vizualne elemente osim teksta, dok se za tekst koristi *scale independent pixel*. Iako je moguće rabiti i piksele (px) kao jedinicu, ovo se izbjegava i nikako ne preporučuje. Korištenje ranije navedenih jedinica osigurava da vizualni elementi izgledaju jednak na ekranima različitih gustoća piksela. Primjer korištenja ovih jedinica dan je primjerom 11.

Primjer 11. - Izdvajanje dimenzija

Dimenzije se izdvajaju kao resursi u xml datoteku proizvoljnog imena koju je nužno kreirati i smjestiti u res/values mapu unutar hijerarhije projekta. Svaka je dimenzija smještena unutar oznaka `<dimen>` i `</dimen>`, a unutar otvarajuće označke dodjeljeno joj je svojstvo *name*. Upravo ovo svojstvo predstavlja jedinstveni identifikator resursa. Dimenzije se definiraju u dp i sp, ovisno o potrebi.

Dimens:

```
<resources>
    <dimen name="main_thumbnailsize">50dp</dimen>
    <dimen name="all_none">0dp</dimen>
</resources>
```

The screenshot shows a smartphone displaying an application titled "MyFirstAwesomeApp". The screen has a green header with the app's name. Below the header is a card with a small icon and the text "Sufferings of a young developer". A yellow box contains the text: "Lubarsky's Law of Cybernetic Entomology: There's always one more bug. Shaw's Principle: Build a system that even a fool can use, and only a fool will want to use it. Wolman's Law: Never program and drink beer at the same time. Gallois' Revelation: If you put tomfoolery into a computer, nothing comes out but tomfoolery. But this tomfoolery, having passed through a very expensive machine, is somehow enabled, and no one dares to criticize it." The phone's status bar shows the time as 5:48.

Layout:

```

<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <ImageView
        android:id="@+id/imageview_main_showthumbnail"
        android:layout_width="@dimen/main_thumbnailsize"
        android:layout_height="@dimen/main_thumbnailsize"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        android:background="@color/color_background"
        android:src="@mipmap/ic_launcher"
        android:contentDescription="@string/main_thumbnaildescription"/>
    <TextView
        android:id="@+id/textview_main_showtitle"
        android:layout_width="@dimen/all_none"
        android:layout_height="@dimen/all_none"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="@id/imageview_main_showthumbnail"
        app:layout_constraintLeft_toRightOf="@+id/imageview_main_showthumbnail"
        app:layout_constraintRight_toRightOf="parent"
        android:gravity="center"
        android:background="@color/color_titlebackground"
        android:textColor="@color/color_titletext"
        android:text="@string/main_showtitle"/>
    <TextView
        android:id="@+id/textview_main_showdescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageview_main_showthumbnail"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintVertical_bias="0"
        android:background="@color/color_contenttext"
        android:text="@string/main_showdescription"/>
</android.support.constraint.ConstraintLayout>

```

1.4.1. Slike

Drawables je zajednički naziv za slike koje se koriste u aplikaciji. Podržani rasterski formati su: JPEG, GIF, PNG i NinePatch (*.9.png, rastezljivi PNG). Iako su podržani, dobra je praksa ne koristiti JPEG i GIF formate. Rasterske slike se uvijek kreiraju u nekoliko različitih veličina kako bi se podržale različite gustoće ekrana. Svaku sliku dobro je kreirati barem u 5 veličina i postaviti u odgovarajuće mape unutar hijerarhije projekta (mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi). U slučaju da za određenu rezoluciju nisu pruženi resursi, sustav će odabrati najbliži resurs i skalirati ga. Odnedavno je u Android studio uvedena i podrška za vektorsku grafiku, pa je kroz *Android Asset Studio* moguće kreirati grafičke resurse u vektorskome formatu. Dodatno, grafičke je elemente moguće kreirati i uporabom XML-a. Primjer korištenja slike koja dolazi kao dio Android SDK dan je primjerom 12.



Više informacija o *Drawable* resursima moguće je pronaći na
<https://developer.android.com/guide/topics/resources/drawable-resource.html>



Šalabahter za Android dizajnere

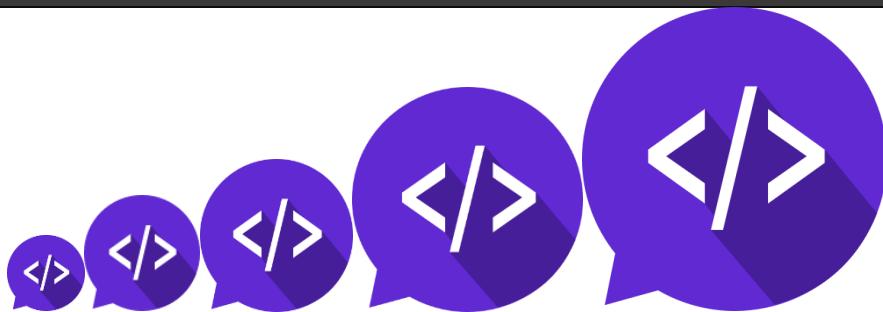
<https://possiblemobile.com/wp-content/uploads/2014/01/Android-Design-Cheat-Sheet.png>



Primjer 12. – Rad sa slikama

U programu za obradu slika GIMP kreirane su 4 inačice grafičkog resursa i postavljene su u odgovarajuće mape unutar hijerarhije projekta. Android će sam, već prema specifikaciji uređaja na kojemu se aplikacija pokreće odabrat i rabiti određeni grafički element. Uz vlastite grafičke elemente, kao dio SDK dolazi velik broj grafičkih elemenata, a prikazan je primjer korištenja i takvih elemenata. Kod kreiranja slike, mdpi je takozvani baseline, odnosno veličina koja se želi postići, hdpi je 1.5xmdpi, xhdpi je 2xmdpi, xxhdpi je 3xmdpi, a xxxhdpi je 4xmdpi. Alternativa ovom pristupu je imati slike/ikone u višoj rezoluciji i pustiti sustav da sam skalira sliku.

Drawable: example.png:



Layout:

```
...
<ImageView
    android:id="@+id/imageview_main_showthumbnail"
    android:layout_width="@dimen/main_thumbnailsize"
    android:layout_height="@dimen/main_thumbnailsize"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    android:background="@color/color_titlebackground"
    android:src="@drawable/example"
    android:contentDescription="@string/main_thumbnaildescription"/>
...
```



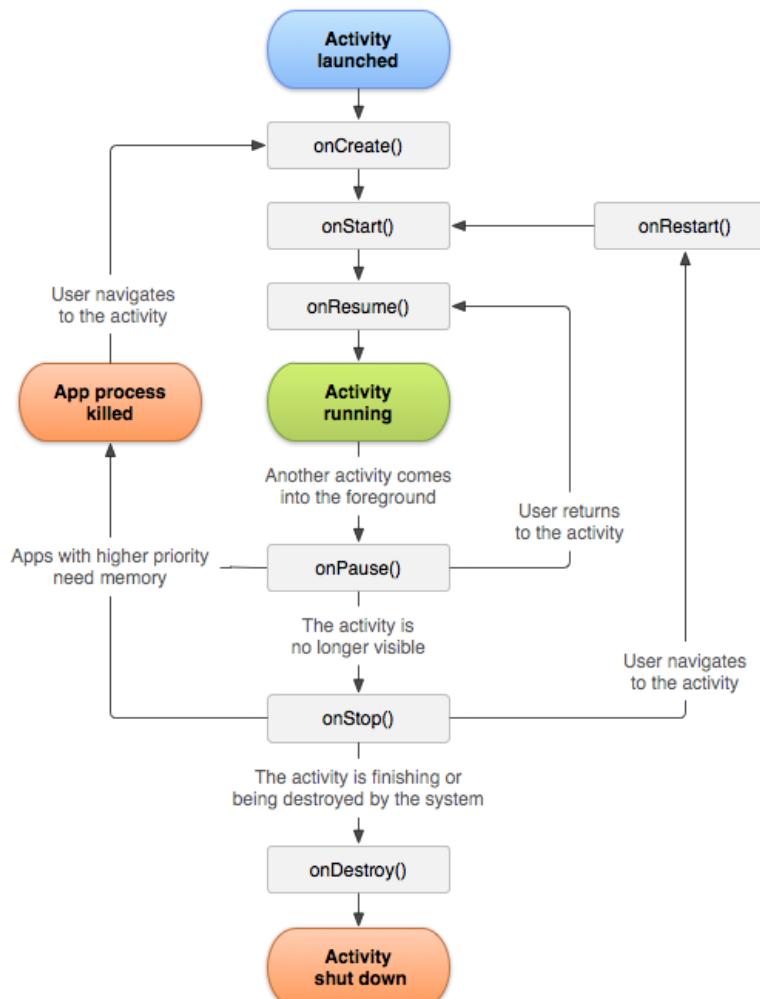
1.1. Kod

Sve do sada prikazano odnosilo se na baratanje XML datotekama i resursima. Iako sami nismo napisali niti jednu liniju koda, ostvareni rezultat je bila (donekle) funkcionalna

Android aplikacija. Razlog tomu je odabir kod kreiranja projekta da se stvori i novi Activity, pri čemu je generirano nešto koda kako bi se korisniku prikazalo sučelje.

1.1.1. Activity

Activity je klasa koja predstavlja jedan zaslon aplikacije. Prvi *Activity* u ovoj je vježbi kreiran prilikom kreiranja projekta, a pri tome je automatski postavljen kao početni ekran koji se prikazuje pokretanjem aplikacije pritiskom na ikonu. Za razliku od klasičnih desktop aplikacija, Android aplikacije nemaju *main* metodu koja služi kao ulazna točka programa, već se životni ciklus aplikacije, odnosno pojedinih aktivnosti oslanja na određene *callback* metode. Ove se metode pozivaju primjerice kod stvaranja *Activity*a ili kod njegova gašenja, a metode životnog ciklusa su: *onCreate()*, *onStart()*, *onResume()*, *onPause()*, *onStop()*, *onRestart()* i *onDestroy()*. Životni je ciklus *Activity*a prikazan slikom 10. Budući da je sučelje odvojeno od koda, nužno ga je „napuhati“, odnosno nužno je iz XML opisa kreirati objekte koji predstavljaju elemente korisničkog sučelja. Ovo obavlja metoda *setContentView()* kojoj se kao argument daje resurs koji predstavlja sučelje za *Activity*.



Slika 1.11. Resursi unutar Android aplikacije (developer.android.com)



Primjer 13. – Activity

Kako je vidljivo iz prikaza koda, klasa koja je kreirana zove se *MainActivity*, a pripada paketu *brunozoric.ferit.hr.myfirstawesomeapp*. Ona nasljeđuje klasu *AppCompatActivity* kako bi se osigurala kompatibilnost sa starijim inačicama Androida (u osnovi, danas vezano samo uz teme), a sadrži samo jednu metodu, *onCreate()*. Ova se metoda poziva prilikom stvaranja *Activitya*, odnosno prilikom pokretanja aplikacije. *Bundle* objekt služi za pohranu prethodnog stanja sučelja, dok *setContentView()* metoda uzima kao argument resurs koji definira sučelje *Activitya* i iz XML-a stvara sve objekte sa zadanim svojstvima te omogućuje njihov prikaz na zaslonu. Layout je jednak ranijem.

Activity:

```

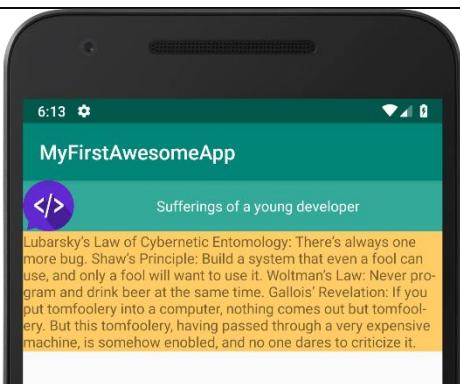
package brunozoric.ferit.hr.myfirstawesomeapp

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

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}

```



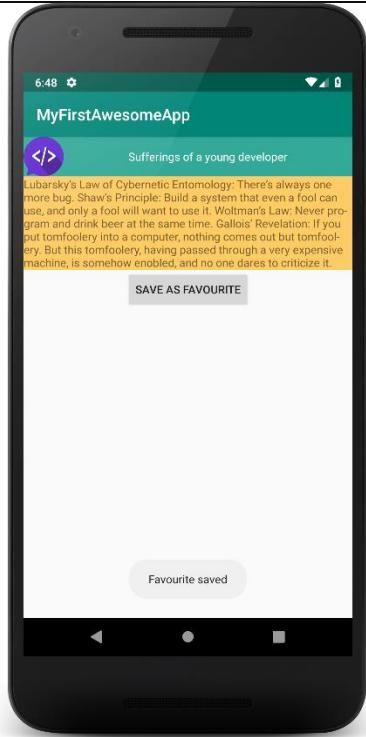
1.1.1. Obrada događaja

Jedan od ranijih primjera pokazuje kako je gumb koji je smješten na sučelje nefunkcionalan. Iako je njegov izgled odgovarajući, a animacija klika očekivana – kod klika se aplikacija ruši jer ne postoji metoda koja bi se pozvala da obradi događaj do kojeg je upravo došlo. Razlog tomu jest što je gumb nužno povezati s metodom koja će biti odgovorna za obavljanje posla, odnosno potrebno je osluškivati događaj i kada do njega dođe izvršiti određeni dio koda. Ovo se postiže implementacijom `onClickListener()` sučelja i postavljanjem objekta klase koja implementira navedeno sučelje kao osluškivača i to uporabom `setOnClickListener()` metode na gumbu. Za navedenu je potrebu najprije potrebno dohvatiti referencu na gumb iz koda što se postiže uporabom `findViewById()` metode kojoj se predaje id gumba, definiran u XML-u. Kako navedena metoda vraća objekt općenitije `View` klase, isti je nužno *castati* u objekt one klase kojoj on uistinu pripada – `Button`. U novijim inačicama Android SDK ova metoda je generička i ne traži više eksplicitan cast.



Primjer 14. – Obrada događaja

Dohvaća se referenca na gumb i postavlja se osluškivač, odnosno objekt neke klase koja implementira *onClickListener()* sučelje. Za tu svrhu može poslužiti sam *Activity*, ali se također može rabiti neka druga klasa ili pak anonimna unutarnja klasa ili se isto može ostvariti lambda izrazom. U slučaju anonimne unutarnje klase, moguće je definirati i svojstvo *onClick* unutar XML-a u kojem se navodi konkretna metoda koja će se pozvati kod klika na gumb.



Activity:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    // Bez korištenja Kotlin Android ekstenzija:
    val saveFavourite : Button = findViewById(R.id.button_main_savefavourite)

    // Korištenje anonimne inner klase:
    saveFavourite.setOnClickListener(object : View.OnClickListener{
        override fun onClick(v: View?) {
            saveShowAsFavourite()
        }
    })

    // Postavljanje osluškivanja uporabom Kotlin Android
    // ekstenzija i lambda izraza:
    // v je ovdje objekt koji je generirao događaj, može biti i bez njega
    button_main_savefavourite.setOnClickListener { v -> saveShowAsFavourite() }
}
```

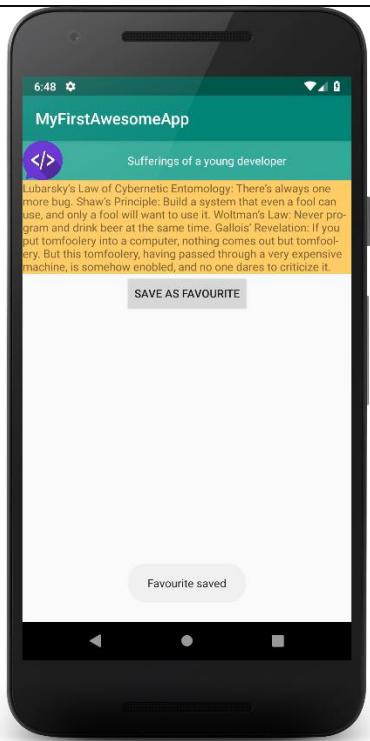
1.1.1. Toast poruke

Obavijesti se na Android platformi korisniku pružiti na različite načine, a jedan od najčešćih je korištenje kratkotrajnih tekstualnih poruka koje se nazivaju *Toast*. S ovim porukama ne postoji mogućnost interakcije, a kreiraju se korištenjem statičkih metoda *Toast* klase, kako je dano primjerom 15.



Primjer 15. – Toast poruke

Kako je uočljivo iz koda danog u klasi `MainActivity.java`, kreirana je posebna metoda za prikaz `Toast` poruka kojoj se predaje poruka u obliku objekta `String` klase ili identifikatora `String` resursa. Metoda se poziva prilikom klika na gumb, a unutar iste se poziva `makeToast()` statička metoda. Kao argumenti daju joj se kontekst (`Activity` jest kontekst, pa je moguće dati `this` referencu), poruka te trajanje. Kao trajanje se koristi jedna od preddefiniranih konstantnih vrijednosti unutar `Toast` klase. U konačnici je nužno pozvati i metodu `show()` kako bi poruka uistinu i bila prikazana.



Activity:

```
fun saveShowAsFavourite() {
    Toast.makeText(this, R.string.main_favouritesaved, Toast.LENGTH_SHORT).show()
}
```

1.1.1. Log poruke

Poruke `Toast` koriste se kada je nužno korisnika obavijestiti o nečemu. Kada je nužno programera obavijestiti o nečemu, one i nisu najbolji izbor. Pri razvoju Android aplikacija moguće je rabiti `Log` klasu te njene statičke metode koje omogućuju ispis prikaz poruka različite razine važnosti. Poruke se prikazuju u `logcatu`, a podržane su razine verbose - `Log.v()`, debug - `Log.d()`, info - `Log.i()`, warning - `Log.w()`, error - and `Log.e()` i what a terrible failure – `Log.wtf()`. Dobra je praksa definirati vlastite oznake (engl. *tag*) kako bi se mogli stvoriti filteri poruka.



Primjer 16. – Log poruke

U bilo kojem trenutku moguće je iskoristiti neku od ranije navedenih metoda kako bi se zabilježila poruka. Kreirana je i vlastita oznaka kako bi se poruka razlikovala od ogromnog broja poruka koje sustav stalno generira.

Activity:

```

class MainActivity : AppCompatActivity() {

    private val TAG = "Bruno"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        Log.d(TAG, "Before setting content view.");
        setContentView(R.layout.activity_main);
        Log.d(TAG, "After setting content view.");
    }
}

```

Logcat

Emulator Nexus_5X_Af | brunozoric.ferit.hr.myfirst | Verbose |

 2019-01-02 18:53:52.236 26116-26116/brunozoric.ferit.hr.myfirstawesomeapp D/Bruno: Before setting content view.
 2019-01-02 18:53:53.152 26116-26116/brunozoric.ferit.hr.myfirstawesomeapp D/Bruno: After setting content view.

1.2. Primjeri



Zadatak 1.

Potrebno je kreirati aplikaciju za računanje indeksa težine. Ovaj se indeks računa prema izrazu:

$$\text{BMI} = \text{masa[kg]} / (\text{visina[m]})^2$$

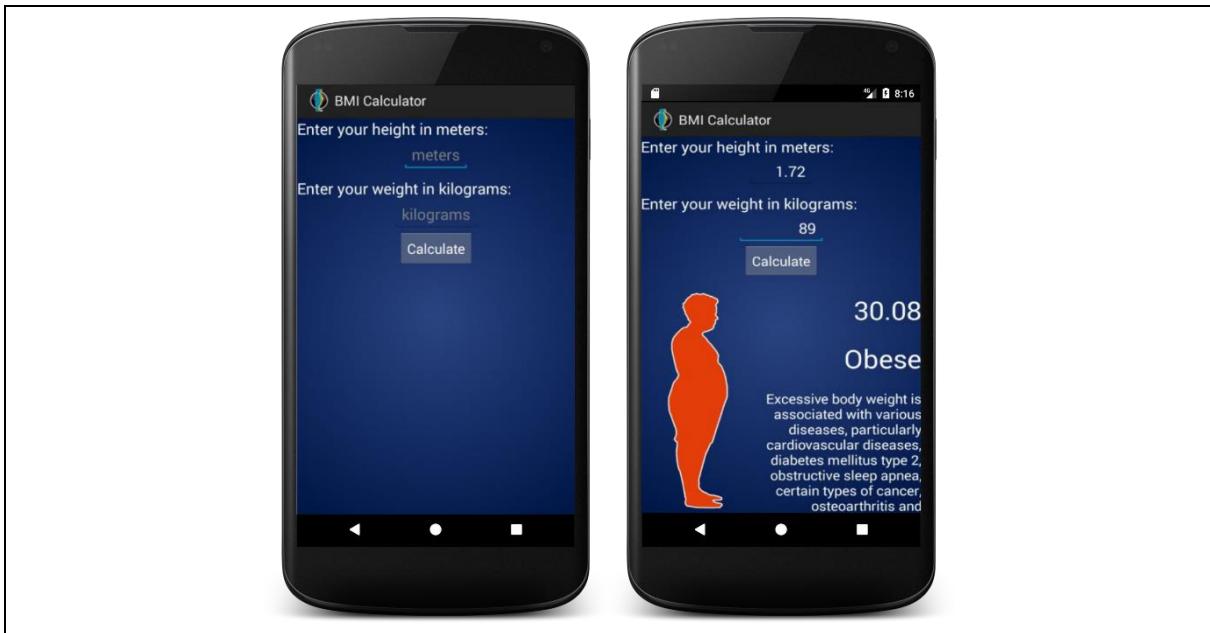
Korisničko sučelje moguće je kreirati prema slikama danim unutar ovog zadatka. Nužno je iskoristiti dva polja za unos visine i težine, dok je rezultat potrebno prikazati u obliku slike, indeksa, naziva i kratkog opisa. Potrebno je dodati sve resurse koji se koriste u aplikaciji poput boja, dimenzija, stringova za svaki string u korisničkom sučelju, kao i za sve mogućnosti rezultata (Pothranjen, Zdrav, Debeo, Pretil). Potrebno je provjeravati korisnički unos, onemogućiti unos negativnih brojeva ili nula, slova, težine veće od 350kg i visine veće od 2.5m.

Sučelje se sastoji od:

- Layouta za smještanje Viewova
- 2 TextViewa za prikaz detalja korisniku (labels)
- 2 EditTexta za unos visine i težine
- Buttona za računanje rezultata
- TextViewa za numerički prikaz BMI-ja
- TextViewa za tekstualni prikaz BMI-ja
- TextViewa za opis BMI-ja
- ImageViewa sa grafičkim prikazom rezultata

Dodatno:

- Prevesti aplikaciju na engleski jezik
- Definirati sučelje za horizontalnu orijentaciju zaslona



1.3. Zadaća

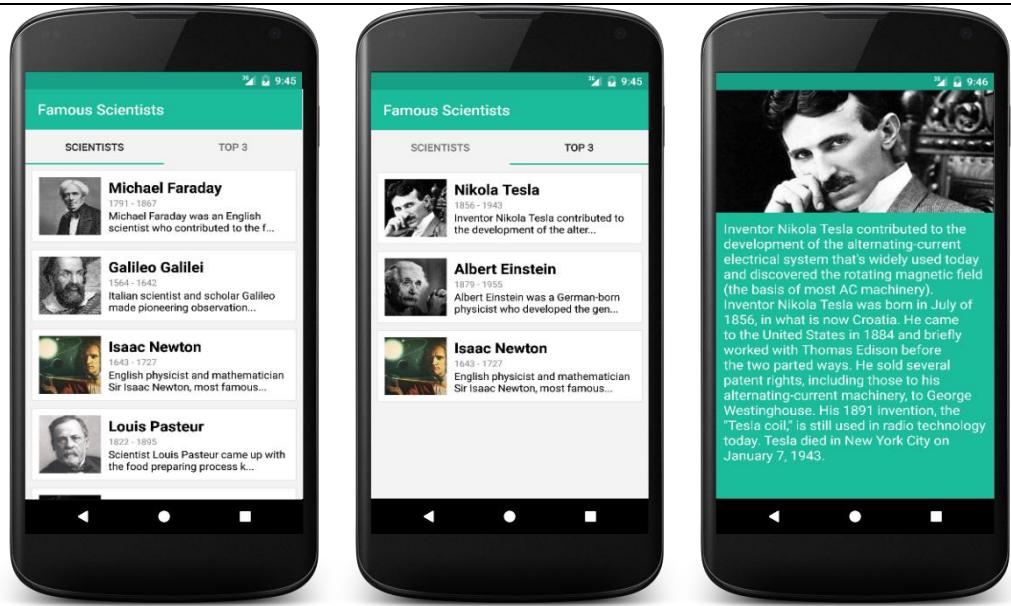


Zadaća 1.

Inspired - Aplikacija o inspirirajućim ljudima

Korištenjem naučenoga na ovoj vježbi, kreirajte jednostavnu aplikaciju koja nudi informacije o tri osobe iz povijesti računarstva koje Vas inspiriraju i za koje smatrate da su imale važan utjecaj. Za svaku osobu potrebno je staviti sliku, datum rođenja/smrti, i kratak životopis. Definirati klik na sliku koji ispisuje nasumičan citat te osobe u obliku *Toast* poruke, citate je moguće spremiti u XML datoteku u obliku polja stringova. Aplikaciju je potrebno testirati na barem dva uređaja (stvarna ili virtualna), te osmisliti ikonu. Cjelokupni projekt potrebno je podići na git repozitorij na nekom od popularnijih servisa (gitlab, bitbucket, github) kao privatni projekt i podijeliti s nastavnikom.

- ❶ Kreirajte novi Android projekt s jednim *Activityem*
- ❷ Kreirajte vlastitu klasu *InspiringPerson* koja će sadržavati podatke o osobi koji se traže (referencirati resurse, ne držati *stringove* unutar osobe)
- ❸ Sučelje definirati u XML-u, tako da sadrži jedan *root layout* u kojem su složeni ostali elementi, rabiti *ConstraintLayout*.
- ❹ Napisati metodu koja barata klikom na sliku daje citat osobe koju se kliknulo u obliku *toast* poruke ili na nekom elementu UI-ja.



Napomena:

Svaka je nadogradnja poželjna i dobrodošla. Korištenje znanja koja nadilaze opseg vježbi se potiče.



"Object-oriented programming offers a sustainable way to write spaghetti code. It lets you accrete programs as a series of patches."

- Paul Graham



LV2:

Intent, RecyclerView

2. Laboratorijska vježba 2

2.1. Uvod

Ova vježba daje uvid u povezivanje Activitya u smislenu cjelinu korištenjem Intenta te pokazuje na koji je način moguće koristiti Intent za pristup ostalim dijelovima Android sustava. U ovoj će laboratorijskoj vježbi također biti prikazano korištenje kontrole RecyclerView te adaptera za prikazivanje sadržaja u istima. Riječ je o najkorištenijim kontrolama u razvoju mobilnih aplikacija za Android platformu, radi prikaza sadržaja prilagođenog (relativno) malim ekranima mobilnih uređaja te mogućnosti dinamičkog popunjavanja sadržajem.

2.1.1. Potrebna predznanja

- Osnove programiranja
 - Kolegiji Programiranje I, Programiranje II, Algoritmi i strukture podataka
- Osnove objektno orijentiranog programiranja
 - Kolegij Objektno orijentirano programiranje
- Osnove Java programskog jezika
 - H. Schildt, Java, A Beginner's Guide, 5th Edition
 - B. Eckel, Thinking in Java, 4th edition
 - <http://docs.oracle.com/javase/tutorial/>
- Osnove izrade Android aplikacija (activity, osnove UI-ja)
 - Predložak LV1, predavanja

2.1.2. Korisna predznanja

- Intent <http://developer.android.com/reference/android/content/Intent.html>
- Fragment <http://developer.android.com/guide/components/fragments.html>
- Action bar (app bar) <https://developer.android.com/training/appbar/index.html>

2.2. Intent

Svaka Android aplikacija može se sastojati od jednog ili više *Activitya* (ne mora sadržavati niti jedan, no riječ je o specifičnim potrebama). Ako se sastoji od više njih, potrebno je na neki način navigirati od jednog do drugog. Mehanizam koji to omogućuje naziva se *Intent*. Štoviše, određeni *Activity* može pokrenuti bilo koja aplikacija na uređaju korištenjem odgovarajućeg *Intenta*, što predstavlja jedan od osnovnih koncepta Android okoline. Sam *Intent* je objekt koji enkapsulira zahtjev, odnosno skup informacija, čiji su osnovni elementi akcija (opća radnja koju je potrebno izvršiti, engl. *action*) i podaci nad kojima se akcija izvršava (engl. *data*). Razlikuju se pri tome implicitni i eksplicitni *Intent*. Eksplicitni

definiraju ciljanu komponentu u samom *Intentu*, dok implicitni traže od sustava da procjeni komponente temeljeno na podacima koje *Intent* sadrži. Uz to što omogućuje kretanje među *Activityima*, *Intenti* omogućuju i komunikaciju među drugim komponentama Android aplikacije (primjerice servisi, primatelji odašiljanja i sl.).



- Alt + Enter – Android Studio kratica za brzi popravak
- Ctrl + Space – Android Studio kratica za automatsko dovršavanje
- Ctrl + Click – Android Studio kratica za skok na definiciju

2.2.1. Eksplisitni intent

Eksplisitni *Intent* definira konkretnu komponentu koju sustav treba pozvati korištenjem klase kao identifikatora i uobičajeno se koristi za komponente unutar aplikacije. Ako se navodi komponenta koju se pokreće, to se radi u parametru „*component name*” *Intent* objekta – ovo čini *Intent* eksplisitnim jer se pokreće isključivo ciljana komponenta. Primjer 2.1, pokazuje kreiranje novog objekta klase *Intent* koji je korišten u metodi *StartActivity()*. Kreiranim *Intentom* pokreće se drugi *Activity*. Ukoliko *Activity* nije dio istog paketa, moguće ga je pokrenuti korištenjem punog identifikatora, odnosno identifikatora koji se sastoji od imena klase i imena paketa. Novi *Activity* moguće je pokrenuti korištenjem *StartActivity* metode koja prima *Intent* objekt kao parametar. *Intent* objekti omogućuju i prenošenje podataka, no ovo će biti pokazano u kasnijem primjeru.



Primjer 2.1. – Eksplisitni Intent

Potrebno je kreirati novi projekt uz dodavanje praznog *Activitya*, a zatim u projekt preko izbornika File→New→Other... dodati novi prazan Android *Activity*. Ovim načinom bit će generirana klasa, *layout* zapis u *Manifest* datoteci. Sučelja i definicije klase dani su niže. Moguće je i samostalno kreirati novu klasu koja će predstavljati *Activity*, no tada je potrebno povesti računa o njegovoj prijavi u *Manifestu* kao i o tome da treba definirati *layout* u odgovarajućim prepisanim metodama ga postaviti na sučelje. U dalnjim primjerima izostavljeni su *importi*, radi manje potrebe za prostorom. Nakon kopiranja koda, moguće je provesti brzi ispravak (engl. *quick fix*) uporabom prečaca Alt+Enter.

Layout: layout_activity_user.xml

```

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".DisplayActivity">
    <EditText
        android:id="@+id/userNameInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_margin="@dimen/marginLarge"/>
    <Button
        android:id="@+id/detailsAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/userNameInput"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/detailsActionText"
        android:layout_margin="@dimen/marginLarge"/>
</androidx.constraintlayout.widget.ConstraintLayout>

```

Activity: UserActivity.kt

```

class UserActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_user)
        setUpUi()
    }

    private fun setUpUi() {
        detailsNavigationAction.setOnClickListener{ navigateToDetails() }
    }

    private fun navigateToDetails() {
        val detailsIntent: Intent = Intent(this, DetailsActivity::class.java)
        startActivity(detailsIntent)
    }
}

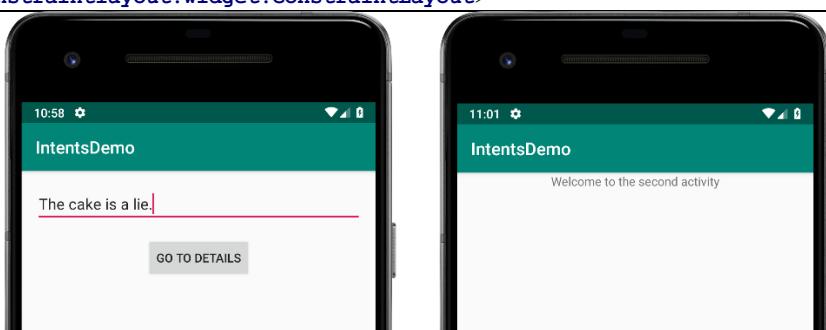
```

Layout: layout_activity_details.xml

```

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".DetailsActivity">
    <TextView
        android:id="@+id/detailsDisplay"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="top|center_horizontal"
        android:text="@string/detailsDisplayDefault"/>
</androidx.constraintlayout.widget.ConstraintLayout>

```

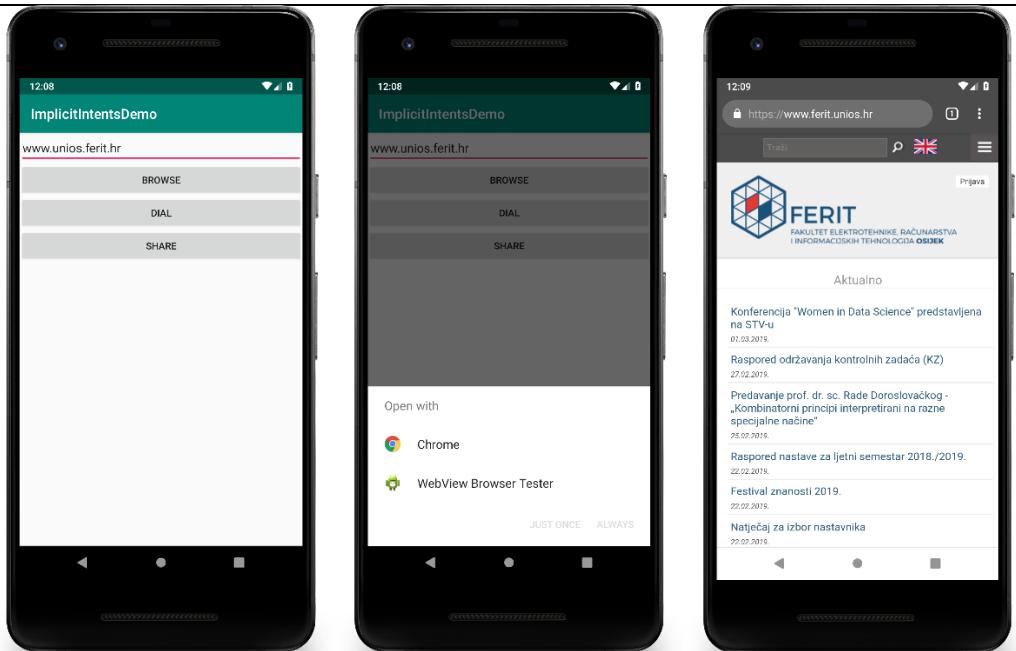


2.2.2. Implicitni intent

Implicitni *Intent* ne definira konkretnu komponentu koju sustav treba pozvati, već opisuje komponentu kakva je potrebna za obradu nekog zadatka nad određenom vrstom podataka i prepušta odluku Android sustavu, odnosno korisniku u krajnjoj liniji, ako postoji više komponenti registriranih u sustavu za obavljanje određene vrste posla (primjerice, instalirana su dva preglednika). Korištenjem implicitnog *Intenta* uobičajeno se pokreću komponente van same aplikacije koja kreira takav *Intent*. Kod kreiranja implicitnog *Intenta* uobičajeno se navodi akcija koju je potrebno izvesti te po potrebi podaci na kojima ju je potrebno izvršiti. Prema ovome će sustav tražiti komponentu koja u svom manifestu ima navedeno da obavlja takvu vrst posla. Ako se sustavu pošalje implicitni *Intent*, onda on pretražuje sve komponente koje su registrirane za navedenu radnju i navedene podatke prema *Intent* filterima u manifest datoteci.

 Primjer 2.2. – Implicitni intent

Potrebito je kreirati novi projekt uz dodavanje praznog *Activitya*. Sučelje i definicije klase dati su niže. Klik na pojedini gumb kreirat će novi *Intent* kojem se postavlja akcija i eventualno podaci nad kojima se ista obavlja. Važno je primijetiti da je kod unosa prije samih podataka potrebno unijeti i prefiks (zato se dodaje „http://“, „sms:“ i sl.) Ukoliko postoji više različitih komponenti koje mogu obaviti traženu radnju, sustav će korisniku ponuditi izbor između njih i dopustiti mu da neku od njih postavi kao podrazumijevanu za tu vrstu zadatka. Ukoliko ne postoji niti jedna komponenta koja zna kako obraditi traženi zadatak, potrebno je obavijestiti korisnika, a moguće ga je uputiti i na sustav za distribuciju aplikacija kako bi preuzeo adekvatnu aplikaciju.



Layout: activity_implicit.xml

```

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ImplicitActivity">

    <EditText
        android:id="@+id/etWebAddressInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        android:hint="@string/webAddressHint"/>

    <Button
        android:id="@+id/browseAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/etWebAddressInput"
        android:text="@string/browseActionText"/>

    <Button
        android:id="@+id/dialAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/browseAction"
        android:text="@string/dialActionText"/>

    <Button
        android:id="@+id/shareAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/dialAction"
        android:text="@string/shareActionText"/>
</androidx.constraintlayout.widget.ConstraintLayout>

```

Activity:

```

class ImplicitActivity : AppCompatActivity() {

    private val TAG : String = "Bruno"
    private val COMPONENT_NOT_FOUND : String = "No component can handle the request"
    private val EMERGENCY_NUMBER: String = "911"
    private val HTTP_PREFIX: String = "http://"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_implicit)
        setUpUi()
    }

    private fun setUpUi() {
        browseAction.setOnClickListener{v -> onClick(v.id)}
        dialAction.setOnClickListener{v -> onClick(v.id)}
        shareAction.setOnClickListener{v -> onClick(v.id)}
    }

    private fun onClick(viewId: Int) {
        when(viewId){
            R.id.browseAction -> { browse(etWebAddressInput.text.toString()) }
            // Dodati dial
            // Dodati share
        }
    }
}

```

```

private fun browse(text: String) {
    var url: String = text.toLowerCase()
    if(url.startsWith(HTTP_PREFIX) == false) url = HTTP_PREFIX + url
    val browseIntent = Intent()
    val uri = Uri.parse(url)
    browseIntent.setData(uri)
    browseIntent.setAction(Intent.ACTION_VIEW)
    startIfPossible(browseIntent)
}

private fun startIfPossible(browseIntent: Intent) {
    if(canBeCalled(browseIntent)) startActivity(browseIntent)
    else Log.d(TAG, COMPONENT_NOT_FOUND)
}

private fun canBeCalled(intent: Intent): Boolean {
    return intent.resolveActivity(packageManager) != null
}
}

```

Odabir komponente koja je zadužena za obradu određene akcije zahtijevane *Intentom* izvršava se korištenjem *Intent* filtera. *Intent* filteri za svaku komponentu definiraju se u manifest datoteci. Ako se sustavu pošalje *Intent*, on provodi određivanje komponente koja odgovara pri čemu koristi podatke iz *Intenta*. Ovim načinom određuje se jedino koja komponenta će obraditi implicitni *Intent*, dok se za eksplisitni ona određuje u objektu *Intenta*. Ako postoji više komponenti koje mogu odraditi posao, onda sustav nudi izbor između njih (npr. klik na link iz FB aplikacije će ponuditi otvaranje poveznice u svim browserima instaliranim na uređaju). U tu svrhu dodaje se *Intent* filter u *Manifest* datoteci. Tri ključna podatka su akcija (npr. ACTION_SEND koja omogućuje dijeljenje sadržaja), podaci koji su podržani (npr. „text/plain“) te kategorija (npr. DEFAULT koja uključuje sve zahtjeve poslane zahtjeve gdje kategorija nije eksplisitno zadana).

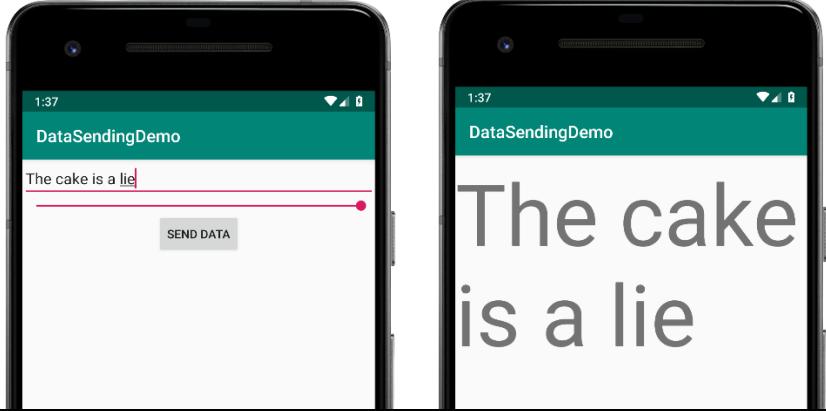
2.2.1. Prosljeđivanje podataka

Unutar objekta klase *Intent* moguće je smjestiti podatke i tim ih načinom prosljeđivati drugim komponentama unutar Android sustava. Vrlo često se ovaj princip rabi za komunikaciju između *Activitya*. S ciljem postizanja navedenog cilja koristi se preopterećena metoda *putExtra()*. Podaci koji se dodaju definirani su kao parovi Ključ/Vrijednost gdje je ključ uvijek objekt *String* klase, a vrijednost može biti jedan od ugrađenih tipova podataka (*int*, *float*,...) te objekt tipa *String* ili *Bundle*. Ukoliko se želi slati objekte vlastitih klasa, tada je potrebno uložiti dodatan napor. Moguće je vlastite objekte zapakirati u objekt klase *Bundle* pa ih s na strani primatelja raspakirati, no ovo je često neprikladno. Stoga se pristupa implementaciji jednog od sučelja *Serializable* ili *Parcelable*. Ako klasa implementira prvo sučelje, tada nije dužna implementirati nikakve dodatne metode već ju je automatski moguće serijalizirati uporabom refleksije. S obzirom da se rabi refleksija, riječ je o relativno sporoj serijalizaciji što može postati usko grlo unutar

aplikacije. Definirano je stoga sučelje *Parcelable* koje u početku traži više truda jer je programer sam odgovoran za serijalizaciju, no pri tome se ostvaruju značajna ubrzanja. Ipak, slanje velikih količina podataka bilo bi dobro izbjegći te pribjeći drugačijem načinu razmjene. Primjerice, umjesto slanja čitavih objekata pohraniti objekt u bazu podataka te poslati samo identifikator i slično.

 Primjer 2.3. – Proslijedivanje podataka putem Intenta

Kreirana su dva *Activitya*, prvi u kojem su smješteni *EditText*, *SeekBar* i *Button*. Od korisnika se zahtjeva unos teksta i odabir veličine tog teksta korištenjem klizača. Korištenjem gumba prikupljeni podaci se šalju u naredni *Activity* i ondje se prikazuju na labeli koja je jedini UI element. Vrijednosti se šalju korištenjem *putExtra()* preopterećene metode na objektu klase *Intent*, a šalju se u obliku parova ključ-vrijednost. S druge strane, kod primanja, potrebno je provjeriti postoji li određeni ključ te pristupati vrijednostima samo ukoliko postoji.



Layout: activity_data_entry.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".DataEntryActivity">
    <EditText
        android:id="@+id/messageInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        android:hint="@string/messageInputHint"/>
    <SeekBar
        android:id="@+id/messageFontSize"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/messageInput"/>
    <Button
        android:id="@+id/messageAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/messageFontSize"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/messageActionText"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Activity: DataEntryActivtiy.kt

```

class DataEntryActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_data_entry)
        setUpUi()
    }

    private fun setUpUi() = messageAction.setOnClickListener{ sendData() }

    private fun sendData() {
        val message = messageInput.text.toString()
        val size = messageFontSize.progress
        val displayIntent = Intent(this, DataDisplayActivity::class.java)
        displayIntent.putExtra(DataDisplayActivity.KEY_MESSAGE, message)
        displayIntent.putExtra(DataDisplayActivity.KEY_SIZE, size.toFloat())
        startActivity(displayIntent)
    }
}

```

Layout: activity_data_display.xml

```

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".DataDisplayActivity">
    <TextView
        android:id="@+id/dataDisplay"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>

```

Activity: DataDisplayActivtiy.kt

```

class DataDisplayActivity : AppCompatActivity() {

    companion object {
        const val KEY_MESSAGE: String = "message"
        const val KEY_SIZE: String = "size"
        const val DEFAULT_FONT_SIZE: Float = 12.0F
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_data_display)

        dataDisplay.text = intent?.getStringExtra(KEY_MESSAGE) ?: getString(R.string.unknownMessage)
        dataDisplay.setTextSize = intent?.getFloatExtra(KEY_SIZE, DEFAULT_FONT_SIZE) ?: DEFAULT_FONT_SIZE
    }
}

```

2.2.2. Pokretanje Activitya radi rezultata

Osim jednosmjernog pokretanja novog *Activitya*, moguće ga je pokrenuti radi dohvaćanja određenog rezultata. To se postiže metodom *startActivityForResult()*, korištenjem kodova za zahtjev i rezultat te *setResult()* metode. Moguće je pokrenuti i druge aplikacije radi dohvaćanja rezultata (ako one to dopuštaju).



Primjer 2.4. – Pokretanje activitya radi rezultata

Kreiraju se dva *Activitya*, jedan iz kojega se šalje zahtjev i u kojemu se hvata rezultat, te drugi koji je zadužen za obradu zahtjeva i dostavljanje rezultata. Svaki zahtjev šalje se s odgovarajućim kodom zahtjeva koji se kasnije rabi pri odgovoru. Svaki rezultat dolazi s kodom rezultata koji kazuje je li zahtjev uspješno obrađen, je li došlo do odustajanja i sl.

Layout: activity_data_request.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".DataRequestActivity">
    <Button
        android:id="@+id/emailAcquisitionAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/dataAcquisitionActionText"/>
    <TextView
        android:id="@+id/emailDisplay"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/emailAcquisitionAction"
        android:textSize="@dimen/textSizeLarge"
        android:gravity="center_horizontal"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Activity: DataRequestActivtiy.kt

```
class DataRequestActivity : AppCompatActivity() {

    companion object {
        const val REQUEST_EMAIL: Int = 10
        const val KEY_EXTRA_EMAIL: String = "email"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_data_request)
        setUpUi()
    }

    private fun setUpUi() = emailAcquisitionAction.setOnClickListener{ startActivityForEmail() }

    private fun startActivityForEmail() {
        val acquireEmailIntent = Intent(this, EmailAcquisitionActivity::class.java)
        startActivityForResult(acquireEmailIntent, REQUEST_EMAIL)
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        when(requestCode){
            REQUEST_EMAIL->{ emailDisplay.text = data?.getStringExtra(KEY_EXTRA_EMAIL) ?: "" }
        }
    }
}
```

Layout: activity_email_acquisition.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".EmailAcquisitionActivity">
    <EditText
        android:id="@+id/emailInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"/>
    <Button
        android:id="@+id/emailSaveAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/emailInput"
        android:text="@string/emailSaveActionText"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Activity: EmailAcquisitionActivtiy.kt

```

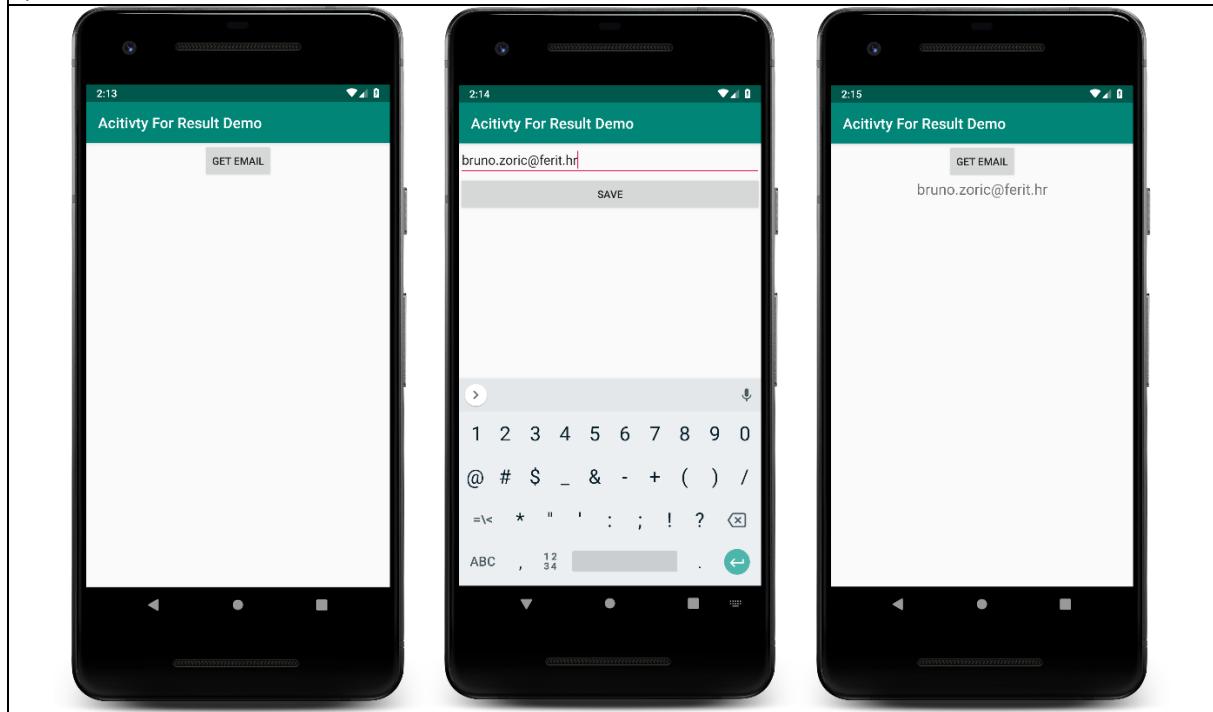
class EmailAcquisitionActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_email_acquisition)
        setUpUi()
    }

    private fun setUpUi() = emailSaveAction.setOnClickListener{ returnEmail() }

    private fun returnEmail() {
        val email = emailInput.text.toString()
        val resultIntent = Intent()
        resultIntent.putExtra(DataRequestActivity.KEY_EXTRA_EMAIL, email)
        this.setResult(Activity.RESULT_OK, resultIntent)
        this.finish()
    }
}

```



2.3. RecyclerView

RecyclerView je kontrola koja je nastala kao rješenje za ograničenja *ListView* kontrole koja omogućuje prikaz liste sadržaja. *RecyclerView* zahtjeva nešto više posla oko inicijalnog postavljanja nego kontrola koju zamjenjuje, no taj posao se isplati s obzirom da konačni rezultat nudi značajno veće mogućnosti prilagodbe. Naime, *RecyclerView* delegira posao rasporeda elemenata (engl. *itema*) *LayoutManageru*, pa je tako moguće postići primjerice pomicanje sadržaja i horizontalno i vertikalno. Nadalje, on, kao i *ListView*, omogućuje recikliranje pojedinih *View* objekata koji su izašli iz prikaza te njihovo ponovno korištenje i postavljanje u prikaz. Ovo značajno umanjuje zahtjeve prema računalnim resursima. Uz navedeno, on zahtjeva korištenje *ViewHolder* oblikovnog obrasca koji omogućuje manje poziva *findViewById* metodi na *View* objektima što dodatno ubrzava rad. Navedeno je bilo moguće, čak štoviše i poticano, i kod korištenja *ListViewa*, no moglo se ignorirati. Dodatno,

RecyclerView omogućuje bolju interakciju s elementima liste, animaciju pojedinih elemenata, prilagodbu razmaka među elementima i dr. Kod rada s *RecyclerView* kontrolom nužno je napisati vlastitu *Adapter* adapter klasu koja će omogućiti prilagodbu iz objekata modela u *View* objekte koje *RecyclerView* zna prikazati (riječ je o klasičnom oblikovnom obrascu (engl. *design pattern*) Adapter). Njegovo postavljanje nešto je složeniji u odnosu na *ListView*, a korištenje je prikazano primjerom 3.3. S obzirom da je potrebna značajnija količina koda u odnosu na prethodne primjere, svaki je korak objašnjen unutar zadatka.



Primjer 2.5. – Korištenje RecyclerView kontrole

Najprije se kreira vlastita klasa koja predstavlja ono što se želi prikazati. Riječ je o model klasi koja će držati podatke. U ovom slučaju odabrat ćemo knjigu kao primjer. Za ove potrebe koristi se Kotlinova *dana* klasa jer će prevoditelj automatski generirati često korištene funkcionalnosti poput *equals()*, *hashCode()*, *toString()* itd.

Book.kt:

```
data class Book (
    val id: Int = 0,
    val title: String,
    val author: String,
    val cover: String)
```

S obzirom da se korištenjem *RecyclerView* kontrole uobičajeno prikazuje veća količina sadržaja, potrebno je dobiti i sadržaj koji će biti prikazan. Često je riječ o podacima koji dolaze iz baze podataka ili s nekog udaljenog poslužitelja, no za potrebe ovog primjera kreirat ćemo jednostavan repozitorij knjiga. Da smo u Javi, implementirali bismo ga prema *Singleton* obrascu, no Kotlin pojednostavljuje priču pa je moguće rabiti *object*.

BookRepository.kt:

```
object BookRepository{
    val books: MutableList<Book>

    init {
        books = retrieveBooks()
    }

    private fun retrieveBooks(): MutableList<Book> {
        return mutableListOf(
            Book(1, "The Great Gatsby", "F. Scott Fitzgerald",
                "https://upload.wikimedia.org/wikipedia/en/f/f7/TheGreatGatsby_1925jacket.jpeg"),
            Book(2, "The Grapes of Wrath", "John Steinbeck",
                "https://upload.wikimedia.org/wikipedia/en/1/1f/JohnSteinbeck_TheGrapesOfWrath.jpg"),
            Book(3, "Nineteen Eighty-Four", "George Orwell",
                "https://upload.wikimedia.org/wikipedia/hr/7/7e/Orwell1984hrv.jpg"),
            Book(4, "It", "Stephen King",
                "https://upload.wikimedia.org/wikipedia/en/5/5a/It_cover.jpg"),
            Book(5, "The lord of the rings", "John R.R. Tolkien",
                "https://upload.wikimedia.org/wikipedia/en/e/e9/First_Single_Volume_Edition_of_The_Lord_of_the_Rings.gif")
        )
    }

    fun remove(id: Int) = books.removeAll{ book -> book.id == id }
    fun get(id: Int): Book? = books.find { book -> book.id == id }
    fun add(book: Book) = books.add(book)
}
```

Idući korak je definiranje izgleda (*layouta*) za pojedinu knjigu. Ovo je nužno napraviti jer *AdapterView* klase kakav je i *RecyclerView* kada prikazuju kolekciju sadržaja imaju mogućnost

kreiranja sučelja iz XML-a i popunjavanje takvog sučelja podacima iz objekta. Za potpuno jednostavan prikaz moguće je rabiti i neke od gotovih *layouta* poput *android.R.layout.simple_list_item_1* ili nekog sličnog njemu. Kako bi radio kod za *layout* u ovom primjeru, potrebno je pripremiti resurse koji su referencirani, a riječ je o stringovima, dimenzijama te jednoj slici naziva *ic_book.png* smještenoj u *drawable* direktorij aplikacije. Važna napomena, kako bi se izgled elementa ispravno prikazivao, koristi se širina *@dimen/match_constraint* što je samo zgodno ime definirano za 0dp. Radi se o tome da će se unutar *ConstraintLayouta* elementi prilagoditi zadanim ograničenjima slično kao *weight* svojstvo kod linearног *layouta*.

Item_book.xml:



RecyclerViewDemo
RecyclerViewDemo

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:padding="@dimen/spacingDefault">
    <ImageView
        android:id="@+id/itemBookCover"
        android:layout_width="@dimen/coverWidth"
        android:layout_height="@dimen/coverHeight"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        android:src="@drawable/ic_book"
        android:scaleType="centerCrop"/>
    <TextView
        android:id="@+id/itemBookTitle"
        android:layout_width="@dimen/match_constraint"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toRightOf="@+id/itemBookCover"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="@+id/itemBookCover"
        android:layout_marginStart="@dimen/spacingDefault"
        android:layout_marginBottom="@dimen/spacingDefault"
        android:textSize="@dimen/textSizeLarge"
        android:text="@string/app_name"/>
    <TextView
        android:id="@+id/itemBookAuthor"
        android:layout_width="@dimen/match_constraint"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toRightOf="@+id/itemBookCover"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/itemBookTitle"
        android:layout_marginLeft="@dimen/spacingDefault"
        android:textSize="@dimen/textSizeNormal"
        android:text="@string/app_name"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Nakon pripreme izgleda za jedan element kolekcije, potrebno je pripremiti izgled za sučelje aplikacije. Za potrebe primjera sučelje može biti vrlo jednostavno, samo jedan RecyclerView. Kako bi se uopće mogao rabiti potrebno je u *gradle.build* datoteku dodati ovisnost.

```
implementation 'androidx.recyclerview:recyclerview:1.0.0'
```

Layout: activity_book_case.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/booksDisplay"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/spacingDefault"
    xmlns:android="http://schemas.android.com/apk/res/android"/>
```

Kada je dodan kao element sučelja, moguće ga je referencirati i rabiti unutar *Activitya*. Nužno je napomenuti kako nakon dodavanja koda u *Activity* i pokretanja projekta neće na ekranu biti prikazano ništa. Još uvijek nismo spojili podatke i *RecyclerView*. Za ove potrebe pripremljena je i pozvana metoda *displayData()* koja za sada ne radi ništa, ali će posljednji korak ovog primjera biti njena implementacija.

Activity: BookCaseActivity

```
class BookCaseActivity : AppCompatActivity() {

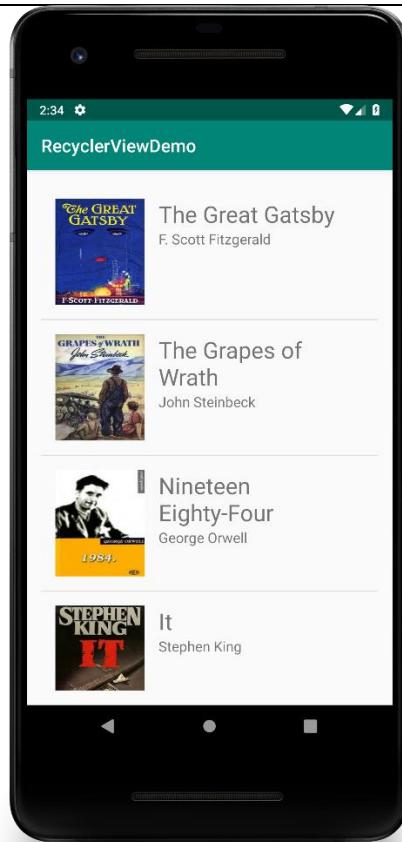
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_book_case)
        setUpUi()
    }

    private fun setUpUi() {
        booksDisplay.layoutManager = LinearLayoutManager(this, RecyclerView.VERTICAL, false)
        booksDisplay.itemAnimator = DefaultItemAnimator()
        booksDisplay.addItemDecoration(DividerItemDecoration(this, RecyclerView.VERTICAL))
        displayData()
    }

    private fun displayData() { }
}
```

Idući korak priče jest kreiranje vlastitog adaptera. Kreira se nova klasa koja nasljeđuje *RecyclerView.Adapter*<> i implementira određene metode osnovne klase. Također, korištenje *RecyclerView* kontrole uvjetovano je korištenjem holder oblikovnog obrasca pa je potrebno napraviti i klasu koja predstavlja holder. S obzirom da objekti klase *Book* sadrže poveznicu na sliku naslovnice, radi lakšeg baratanja slikama koristi se *Picasso* biblioteka, a učitavanje se obavlja unutar adaptera.

implementation 'com.squareup.picasso:picasso:2.71828'



Activity: BookAdapter

```
class BookAdapter(
    books: MutableList<Book>
) : RecyclerView.Adapter<BookHolder>() {

    private val books: MutableList<Book>

    init {
        this.books = mutableListOf()
        this.books.addAll(books)
    }

    fun refreshData(books: MutableList<Book>) {
        this.books.clear()
        this.books.addAll(books)
        this.notifyDataSetChanged()
    }

    override fun getItemCount(): Int = books.size

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): BookHolder {
        val bookView = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_book, parent, false)
        val bookHolder = BookHolder(bookView)
        return bookHolder
    }

    override fun onBindViewHolder(holder: BookHolder, position: Int) {
        val book = books[position]
        holder.bind(book)
    }
}

class BookHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    fun bind(book: Book) {
        itemView.itemBookTitle.text = book.title
        itemView.itemBookAuthor.text = book.author
        Picasso.get()
            .load(book.cover)
            .fit()
            .placeholder(R.drawable.ic_book)
            .error(android.R.drawable.stat_notify_error)
            .into(itemView.itemBookCover)
    }
}
```

U konačnici je preostalo implementirati samo metodu za postavljanje podataka u *RecyclerView* koja se nalazi u *Activityu*.

```
private fun displayData() {
    booksDisplay.adapter = BookAdapter(BookRepository.books)
}
```

2.3.1. Interakcija s RecyclerViewom

Za razliku od *ListView* kontrole koja je na vrlo jednostavan način izlaže klik i dugi klik na element liste, kod *RecyclerView* kontrole potreban je dodatan rad. Razlog leži u tome da *RecyclerView* brine isključivo o recikliranju i prikazu, a sav ostali posao delegira. Jedna od zamjerki *ListViewu* je bila problematična interakcija s elementima pojedinog *Viewa*. Kod *Recyclera* moguće je na različite načine ostvariti interakciju s pojedinim elementima liste. Moguće je jednostavno dodati osluškivanje unutar *ViewHoldera* ili je moguće poslati objekt koji implementira željeno sučelje kroz konstruktor *Adaptera*.



Primjer 2.6. – Korištenje RecyclerView kontrole

Dodaje se novo sučelje naziva *BookInteractionListener*. Sučelje sadrži dvije metode, *onShowDetails()* i *onRemove()*

BookInteractionListener.kt:

```
interface BookInteractionListener{
    fun onRemove(id: Int)
    fun onShowDetails(id: Int)
}
```

Dodaje se novo sučelje naziva *BookInteractionListener*. Sučelje sadrži dvije metode, *onShowDetails()* i *onRemove()*. Implementacija sučelja napravljena je kao object unutar *Activitya* u metodi *displayData()* gdje se ono i koristi, odnosno daje se adapteru kroz konstruktor.

```
private fun displayData() {
    val bookListener = object: BookInteractionListener{
        override fun onRemove(id: Int) {
            BookRepository.remove(id)
            (booksDisplay.adapter as BookAdapter).refreshData(BookRepository.books)
        }

        override fun onShowDetails(id: Int) {
            val book = BookRepository.get(id)
            Toast.makeText(applicationContext, book.toString(), Toast.LENGTH_SHORT).show()
        }
    }
    booksDisplay.adapter = BookAdapter(BookRepository.books, bookListener)
}
```

Iz prethodnog izlistanja koda uočljivo je kako se adapteru kroz konstruktor predaje i objekt koji implementira dano sučelje, tako da je potrebno proširiti konstruktor odgovarajućim parametrom. Također, proširuje se i *bind()* metoda holdera s obzirom da je potrebno delegirati događaje poput klikla ili dugog klikla na element unutar *RecyclerViewa* na objekt koji je poslan adapteru. Dan je ponovno kod za cijeli adapter, jednostavnosti radi (e.g. student može samo napraviti c/p i miran je do usmenog kada će trebati objasniti što ovo radi). Važno je još reći kako se u primjeru osluškuju klikovi na cijeli *View*. Ovo nije uvijek željena procedura, no na isti se način događaji mogu pridružiti bilo kojoj od kontrola na svakom pojedinom elementu kojeg *RecyclerView* prikazuje, umjesto da se postavlja na *itemView*.

```
class BookAdapter(
    books: MutableList<Book>,
    bookListener: BookInteractionListener
): RecyclerView.Adapter<BookHolder>() {

    private val books: MutableList<Book>
    private val bookListener: BookInteractionListener

    init {
        this.books = mutableListOf()
        this.books.addAll(books)
        this.bookListener = bookListener
    }

    fun refreshData(books: MutableList<Book>) {
        this.books.clear()
        this.books.addAll(books)
        this.notifyDataSetChanged()
    }

    override fun getItemCount(): Int = books.size

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): BookHolder {
        val bookView = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_book, parent, false)
        val bookHolder = BookHolder(bookView)
        return bookHolder
    }

    override fun onBindViewHolder(holder: BookHolder, position: Int) {
        val book = books[position]
        holder.bind(book, bookListener)
    }
}
```

```

class BookHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    fun bind(book: Book, bookListener: BookInteractionListener) {
        itemView.itemBookTitle.text = book.title
        itemView.itemBookAuthor.text = book.author
        Picasso.get()
            .load(book.cover)
            .fit()
            .placeholder(R.drawable.ic_book)
            .error(android.R.drawable.stat_notify_error)
            .into(itemView.itemBookCover)
        itemView.setOnClickListener { bookListener.onShowDetails(book.id) }
        itemView.setOnLongClickListener{ bookListener.onRemove(book.id); true; }
    }
}

```

2.4. Primjer



Zadatak 1.

Napravite igru pogađanja korištenjem *RecyclerView* kontrole. Kreirati repozitorij riječi i popuniti ga rijećima (koristiti on line rječnik i sl.). Sustav na početku svake igre nasumično odabire riječ iz rječnika i to je pojam koji korisnik mora pogoditi. Istovremeno sustav iz rječnika odabire 9 dodatnih pojmove koji predstavljaju smetalice. U *RecyclerView* kontroli prikazuju se svi pojmovi nasumičnim redoslijedom. Korisnik bira među njima i svakim promašajem gubi bodove (kreće sa 100 bodova, svaki promašaj uzima 10). Kod odabira smetalice, ona se uklanja sa sučelja. Kod odabira traženog pojma korisnik je pobijedio u igri i korištenjem eksplizitnog intinta usmjerava se na sučelje za prikaz rezultata. Ostvariti i pamćenje rezultata igre (dovoljno samo za trajanje aplikacije).

2.5. Zadaća

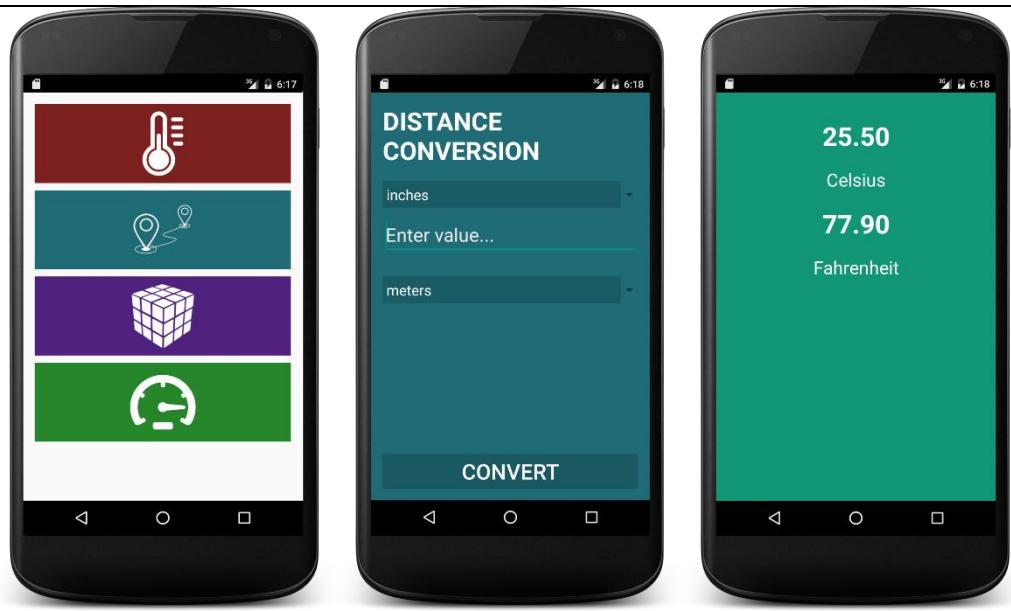


Zadaća 2.

UniCon – Konverter jedinica

Potrebno je kreirati aplikaciju za konverziju mjerne jedinice. Aplikacija se sastoji iz četiri *Activitya*. Prvi sadrži *RecyclerView* sa slikama i opisom koji govore o kakvom pretvaranju riječ (npr. temperatura, udaljenost...). Klik na pojedini element eksplizitnim *Intentom* pokreće Activity za željenu konverziju. U Activityu za konverziju potrebno je prikupiti korisnički unos vrijednosti, tip jedinice unosa (koristiti *Spinner*) i izračunati rezultat koji se zatim šalje u Activity namjenjen prikazu rezultata korištenjem extra podataka.

- ➊ Kreirati novi projekt
- ➋ Kreirati osnovni izbornik s gumbima
- ➌ Implementirati barem 4 pretvaranja po izboru (temp., težina, udaljenost, zapremina, podaci ...). Razmisliti o implementaciji! Tona *boilerplate* koda znači manje bodova
- ➍ Kreirati Activity za prikaz rezultata
- ➎ Povezati Activitye korištenjem Intenta
- ➏ Koristiti stilove za uređivanje prikaza rezultata.



Napomena: Svaka je nadogradnja poželjna i dobrodošla. Korištenje znanja koja nadilaze opseg Vježbe 1. i 2. se potiče.



"There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies.

The first method is far more difficult."

- C.A.R. Hoare



LV3:

Očuvanje stanja, Fragmenti

3. Laboratorijska vježba 3

3.1. Uvod

Jedan od čestih slučajeva korištenja *RecyclerView* kontrole obrađene prošlom laboratorijskom vježbom jest dohvaćanje sadržaja iz nekog oblika trajne pohrane za što se mogu koristiti lokalne ili udaljene baze podataka, dijeljene postavke ili datoteke. Ovi će koncepti biti prikazani i korišteni u ovoj laboratorijskoj vježbi. Uz njih, bit će pokriveni i fragmenti, posebne klase koje su se pojavile u verziji Androida 3.0, a namijenjene su lakšem prikazu dinamičkog sadržaja. Osim korištenja fragmenata unutar *ViewPager* klase, koriste se i u brojnim drugim slučajevima poput prikazivanja dijaloških okvira, navigacije koja se pojavljuje klizanjem s krajnjeg ruba ekrana i slično.

3.1.1. Potrebna predznanja

- Osnove programiranja
 - Kolegiji Programiranje I, Programiranje II, Algoritmi i strukture podataka
- Osnove objektno orijentiranog programiranja
 - Kolegij Objektno orijentirano programiranje
- Osnove izrade Android aplikacija (activity, osnove UI-ja)
 - Predložak LV1, LV2, predavanja
- Osnove Java programskog jezika
 - Java, A Beginner's Guide, 5th Edition - Herbert Schildt
 - Thinking in Java, 4th edition - Bruce Eckel
 - <http://docs.oracle.com/javase/tutorial/>
- Osnove relacijskih baza podataka
 - Kolegij Baze podataka
 - <https://www.sqlite.org/about.html>

3.1.2. Korisna predznanja

- Intent
 - <http://developer.android.com/reference/android/content/Intent.html>
- Adapter design pattern
 - http://sourcemaking.com/design_patterns/adapter
- RecyclerView
 - <https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html>

3.2. Dijeljene postavke

Dijeljene postavke (engl. *Shared Preferences*) omogućuju spremanje manjih količina podataka u obliku para ključ/vrijednost u XML datoteku, nakon čega je tim vrijednostima moguće pristupati na jednostavan način. Kao što i ime kaže, najčešće se koriste za pohranu nekih konfiguracijskih detalja, kao npr. preferirani poredak elemenata liste i sl. Postavke mogu biti namijenjene samo jednoj aktivnosti, ali i dijeljene među aktivnostima. Nekoliko je načina za pristup postavkama:

- `getPreferences()` - za pojedinu aktivnost
- `getSharedPreferences()` - za dijeljenje među aktivnostima
- `getDefaultSharedPreferences()` - za rad s kompletnim razvojnim okvirom postavki

Sve navedene metode vraćaju *SharedPreferences* objekt koji može rabiti pristupne metode za postavljanje i dohvaćanje podataka. Za njihovu izmjenu koristi se klasa *Editor* čija se instanca dobiva metodom *edit()* pozvanom na objektu dijeljenih postavki. Promjene se pohranjuju pozivom *commit()* metode za ranije inačice sustava ili *apply()* metode kod novijih (>2.3). Standardni *preference screen* je preferirani način spremanja ovih postavki, radi uniformnog prikaza korisnicima i zadržavanja stila okoline.



Primjer 3.1. – Dijeljene postavke

Kreirana je posebna klasa koja će se rabiti za zapisivanje i čitanje dijeljenih postavki. Zapisan je samo jedan *string*, ali svi drugi podaci bili bi zapisani na jednak način. Važno je uočiti da se podaci zapisuju u obliku parova ključ/vrijednost, a prilikom dohvaćanja je nužno navesti podrazumijevanu vrijednost.

PreferenceManager.kt:

```
class PreferenceManager {  
  
    companion object {  
        const val PREFS_FILE = "MyPreferences"  
        const val PREFS_KEY_USERNAME = "Bruno"  
    }  
  
    fun saveUsername(username: String) {  
        val sharedpreferences = MyApplication.ApplicationContext.getSharedPreferences(  
            PREFS_FILE, Context.MODE_PRIVATE  
        )  
        val editor = sharedpreferences.edit()  
        editor.putString(PREFS_KEY_USERNAME, username)  
        editor.apply()  
    }  
  
    fun retrieveUsername(): String {  
        val sharedpreferences = MyApplication.ApplicationContext.getSharedPreferences(  
            PREFS_FILE, Context.MODE_PRIVATE  
        )  
        return sharedpreferences.getString(PREFS_KEY_USERNAME, "unknown")  
    }  
}
```

Kako je vidljivo, da bi se pristupilo objektu koji kontrolira pristup dijeljenim postavkama, potreban je kontekst. Jedna od opcija je poslati kontekst kao argument pojedine metode ili čuvati kontekst

kao atribut ove pomoćne klase. Međutim, često je zgodnije koristiti aplikacijski kontekst kada je on nužan u različitim dijelovima aplikacije. Kako je spomenuto na ranijim vježbama, svaka aplikacija ima „*preleakani*“ kontekst, no da bi mu se moglo pristupiti bilo gdje u kodu, kreira se vlastita klasa koja predstavlja aplikaciju. U slučaju ovog primjera, to je klasa MyApplication (u stvarnom korištenju dajte joj pametnije ime).

MyApplication.kt:

```
class MyApplication : Application() {

    companion object {
        lateinit var ApplicationContext: Context
        private set
    }

    override fun onCreate() {
        super.onCreate()
        ApplicationContext = this
    }
}
```

Međutim, ovo nije dovoljno. Da bi se ova klasa zbilja smatrala aplikacijskom klasom, potrebna je i malena izmjena manifest datoteke.

Manifest.xml:

```
<application
    android:name=".MyApplication"
```

Sada je potrebno još izmijeniti sučelje glavnog *Activitya* kao i postaviti funkcionalnost unutar *Activitya*.

MainActivity.kt:

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        usernameSaveAction.setOnClickListener{saveUsername()}
    }

    override fun onResume() {
        super.onResume()
        displayWelcomeMessage()
    }

    private fun displayWelcomeMessage() {
        val username = PreferenceManager().retrieveUsername();
        usernameDisplay.text = getString(R.string.helloMessage, username)
    }

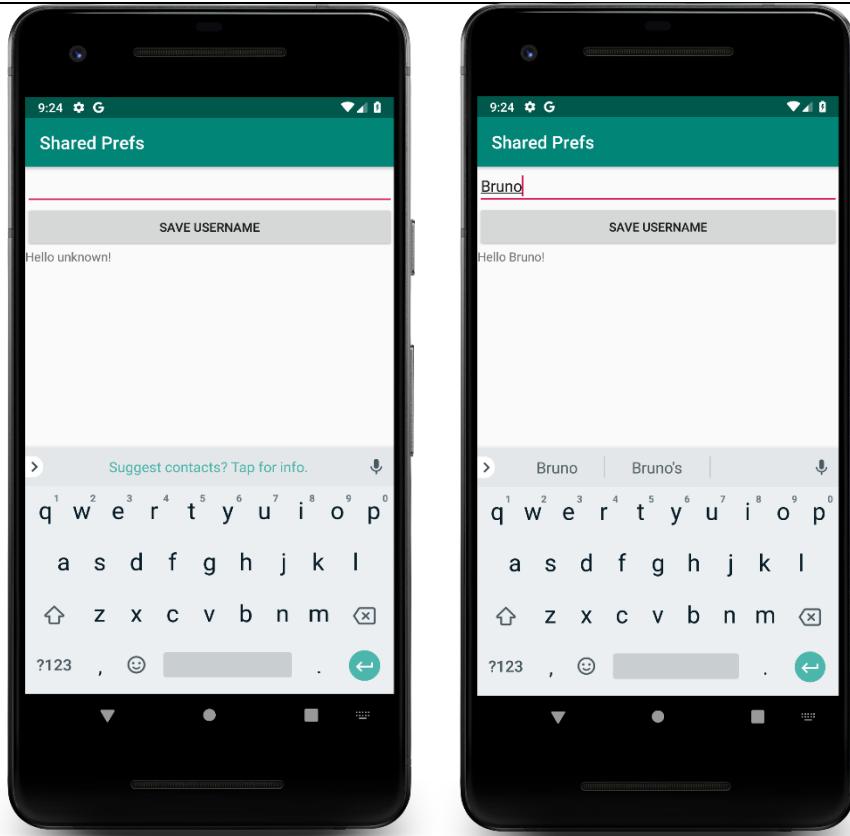
    private fun saveUsername() {
        val username: String = usernameInput.text.toString();
        val preferenceManager = PreferenceManager()
        preferenceManager.saveUsername(username)
        displayWelcomeMessage()
    }
}
```

activity_main.xml:

```

<LinearLayout
    android:orientation="vertical"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
<EditText
    android:id="@+id/usernameInput"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
<Button
    android:id="@+id/usernameSaveAction"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/usernameSaveActionText"/>
<TextView
    android:id="@+id/usernameDisplay"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
</LinearLayout>

```



3.1. Korištenje baze podataka

Osim dijeljenih postavki i uobičajenih datoteka, Android nudi mogućnost pohrane u bazu podataka. Riječ je o SQLite bazi, prilagođenoj za rad u okruženju s ograničenim resursima. Podaci koji se pohranjuju u bazu su dostupni samo aplikaciji, ne i ostalim komponentama sustava. Ukoliko je podatke nužno učiniti javno dostupnima, moguće je to postići putem *ContentProvidera*. Kada se baza kreira, nalazi se u `/data/data/<package_name>/databases` mapi uređaja i moguće joj je pristupiti na uređajima s *root* pravom pristupa.

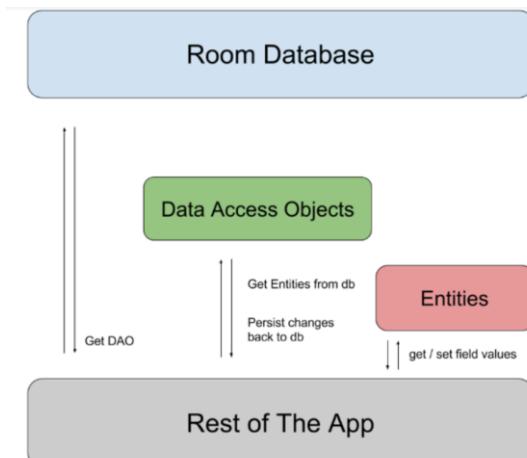
U pravilu se u prošlosti kod rada s bazom koristila klasa *SQLiteDatabase* koju se nasljeđuje, a koja omogućuje lakše upravljanje bazom (stvaranje baze, otvaranje,

zatvaranje konekcije itd.). Kako bi se osigurao siguran pristup iz različitih niti (s obzirom da bi se rukovanje bazom i zapisi i čitanja trebali odvijati mimo UI niti), preporučuje se ovu klasu implementirati putem obrasca *Singleton*. Uz ovu klasu, uobičajeno se kreira i klasa koja će predstavljati shemu (engl. *scheme*) baze, odnosno ugovor koji sadrži imena tablica, stupaca itd. Rad s bazom (pisanje upita) omogućen je na nekoliko različitih načina. Moguće je izvršavati legalne SQL upite u obliku stringova, a moguće je rabiti i pomoćne metode i slagati upite iz predanih argumenata. Kod umetanja sadržaja u bazu potrebno je koristiti *ContentValues* klasu i njene objekte u koje se umeću vrijednosti koje će biti zapisane metodom *put()*.

Međutim, izravan primjer kakav je upravo opisan donosi brojne potencijalne probleme pa se danas rijetko koristi. U pravilu se danas oslanja na različite biblioteke koje će ne samo sakriti većinu posla od programera, već i osigurati da, primjerice, SQL upiti budu podložni provjeri tijekom pisanja koda i sl. Dva najčešće korištena alata za rad s bazama podataka danas su Realm i Room. Svakako bi bilo dobro iskušati oba i donijeti informiran izbor, no na ovim će se vježbama rabiti Room ORM alat za objektno relacijsko preslikavanje. Riječ je o Googleovoj biblioteci koja omogućuje lako stvaranje baze i potrebnih tablica, kao i vrlo jednostavno baratanje CRUD operacijama.

3.1.1. ROOM biblioteka

Room biblioteka za trajnu pohranu podataka predstavlja sloj apstrakcije smješten iznad SQLitea kako bi olakšao njegovo korištenje. Olakšana je pohранa strukturiranih podataka i kasnije njihovo učitavanje. Tri su glavne komponente unutar Room biblioteke:



- *Database* – Služi kao pristupna točka bazi, riječ je o apstraktnoj klasi koja nasljeđuje klasu *RoomDatabase* i označena je anotacijom *@Database*. Unutar ove anotacije navedene su sve klase koje predstavljaju entitete. Također, ova klasa

sadrži apstraktne metode koje nemaju argumente i kao povratni tip imaju klase označene s anotacijom `@Dao`.

- *Entity* – predstavlja tablicu u bazi
- *DAO* – Sadrži metode za pristup bazi



Primjer 3.2. – Baza podataka

Ovaj primjer nastaviti će se na primjer obrađen na prošloj vježbi. Radit će se baza za pohranu knjiga. Da bi se mogao koristiti Room, potrebno je najprije dodati ovisnost u *gradle* datoteku. Primjena *plugina* radi se na samom vrhu app *build* datoteke, ovisnosti se dodaju kao i do sada, jedino je verzije ovisnosti moguće izdvajati u posebnu varijablu pa je ovdje to i učinjeno. Ova varijabla naziva *ext* nalazi se u *build* datoteci čitavog projekta. Ovo se smatra dobrom praksom te se preporučuje rabiti i s drugim vanjskim ovisnostima.

build.gradle (module:app):

```
apply plugin: 'kotlin-kapt'  
// ...  
  
implementation "androidx.room:room-runtime:$roomVersion"  
kapt "androidx.room:room-compiler:$roomVersion"
```

build.gradle (Project):

```
ext{  
    roomVersion = '2.1.0-alpha06'  
}
```

Kreirajmo prvo klasu čije objekte želimo pohranjivati. Takvu klasu označavamo anotacijom `@Entity` i pri tome se kreira tablica u bazi s danim imenom (ime klase je ime tablice, imena atributa su imena stupaca u tablici). Iako u ovom slučaju nije potrebno, moguće je anotacijom `@ColumnInfo` i dodavanjem svojstava postaviti imena stupaca u tablici, dok je korištenjem `@Ignore` anotacije moguće ignorirati pojedini atribut unutar klase entiteta.

Book.kt:

```
@Entity(tableName = "books")  
data class Book (  
    @PrimaryKey(autoGenerate = true) val id: Int,  
    @ColumnInfo(name = "title") val title: String,  
    @ColumnInfo(name = "author") val author: String  
)
```

Kada je entitet spreman, pristupimo kreiranju objekta za pristup, takozvanog DAO (engl. *Database Access Object*) objekta. Ovi se objekti definiraju u obliku Kotlin (Java) sučelja označenih anotacijom `@Dao`. U sučeljima se navode potpisi metoda koje želimo podržati u kontekstu pristupa podacima. Uobičajeno je definirati Dao objekte prema funkcionalnosti, primjerice, ako želimo pristupati tablici s knjigama, kreirat ćemo BookDao sučelje. Room će sam, na temelju propisanog sučelja, generirati konkretne implementacije ove klase, odnosno omogućiti pristup željenim podacima. Četiri su anotacije koje se rabe unutar ovih sučelja, a riječ je o poznatim CRUD operacijama - `@Query`, `@Insert`, `@Update`, `@Delete`. Kod anotacije `@Query` moguće je pisati i vlastite SQL upite. Osim provjere SQL upita i automatskog završavanja, moguće je parametrizirati upite. Jedan takav primjer dan je kod dohvatanja svih knjiga određenog autora.

BookDao.kt:

```

@Dao
interface BookDao{

    @Insert
    fun insert(book: Book);

    @Delete
    fun delete(book: Book);

    @Query("SELECT * FROM books")
    fun getAll(): List<Book>;

    @Query("SELECT * FROM books WHERE author = :author")
    fun getByAuthor(author: String): List<Book>;
}

```

U konačnici, kreira se klasa koja predstavlja bazu podataka, a nasljeđuje RoomDatabase klasu. Ovdje se navode i neki nužni podaci kao što su svi entiteti, verzija baze i slično. Dobra je praksa rabiti singleton obrazac u ovom slučaju. Kako bismo mogli pristupiti kontekstu, a ne ga morali svaki put slati, opet ćemo se poslužiti idejom iz prethodnog primjera.

MyApplication.kt:

```

class MyApplication : Application() {

    companion object {
        lateinit var ApplicationContext: Context
            private set
    }

    override fun onCreate() {
        super.onCreate()
        ApplicationContext = this
    }
}

```

BookDatabase.kt:

```

@Database(version = 1, entities = arrayOf(Book::class))
abstract class BookDatabase : RoomDatabase() {

    abstract fun bookDao(): BookDao

    companion object {
        private const val NAME = "book_database"
        private var INSTANCE: BookDatabase? = null

        fun getInstance(): BookDatabase {
            if(INSTANCE == null) {

                INSTANCE = Room.databaseBuilder(
                    MyApplication.ApplicationContext,
                    BookDatabase::class.java,
                    NAME)
                    .allowMainThreadQueries() // Not for real apps!
                    .build()
            }
            return INSTANCE as BookDatabase
        }
    }
}

```

Preostalo je napraviti nekakav oblik korisničkog sučelja koji bi rabio ovu bazu te prikazati knjige na sučelju.

activity_main.xml:

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <ListView
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/booksDisplay"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
    <Button
        android:id="@+id/addBookAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/addBookActionText"/>
</androidx.constraintlayout.widget.ConstraintLayout>

```

MainActivity.kt:

```

class MainActivity : AppCompatActivity() {

    val booksDao = BookDatabase.getInstance().bookDao()

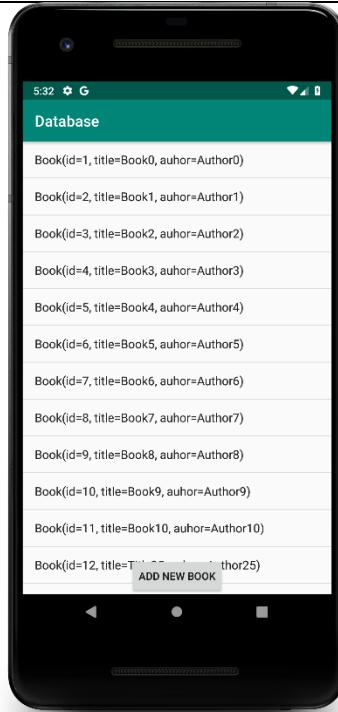
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        addBookAction.setOnClickListener{addBook() }
        displayBooks()
    }

    private fun displayBooks() {
        booksDisplay.adapter = ArrayAdapter<Book>(
            this,
            android.R.layout.simple_list_item_1,
            booksDao.getAll())
    }

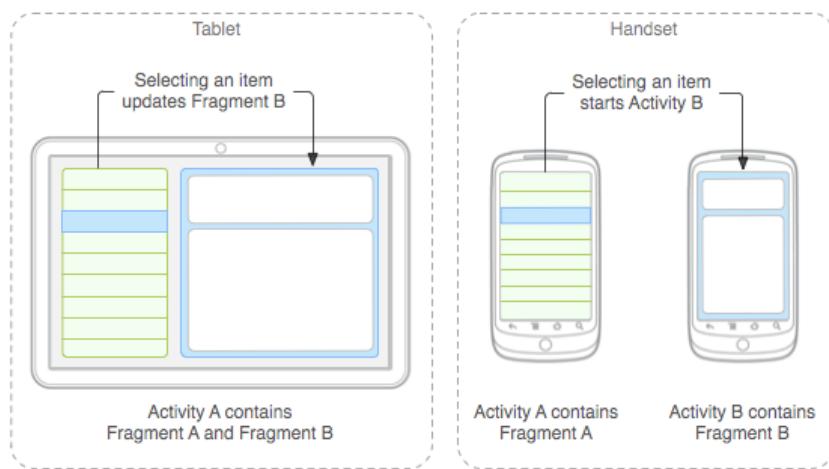
    private fun addBook() {
        val index = Random().nextInt(100)
        val book = Book(0, "Title$index", "Author$index")
        booksDao.insert(book)
        displayBooks()
    }
}

```



3.2. Fragmenti

Fragmenti predstavljaju objekte klase koji omogućuju modularan dizajn *Activitya*. Riječ je o dodatnom sloju smještenom između sučelja i *Activitya*. Moguće ih je koristiti više unutar jednog *Activitya*, ali i koristiti ih kroz više *Activitya*. Imaju vlastiti životni ciklus koji je usko vezan uz *Activityu* kojem se nalaze pa im metode životnog ciklusa odražavaju one *Activitya*. Tijekom životnog ciklusa *Activitya* svi fragmenti prolaze iste tranzicije, a dok je *Activity* aktivan moguće je manipulirati pojedinim fragmentima. Rad s fragmentima nije trivijalan, a prilikom njihova korištenja potrebno je biti oprezan.



Slika 3.1. Korištenje fragmenata (developer.android.com)



Ključnih pet pitanja za fragmente (Mark L. Murphy):

What? / Where?? / Who?!? / When?!? / Why?!?/? / OMGOMGOMG, HOW?!?!!??



Google I/O 2016: What The Fragment

<https://www.youtube.com/watch?v=k3IT-IJ0J98>

3.2.1. Kreiranje vlastitog fragmenta

Kako bi se kreirao fragment, potrebno je naslijediti klasu *Fragment* koja dolazi kao dio Android SDK. Riječ je o klasi koja je podržana od API razine 11, a u slučaju da aplikacija podržava i starije uređaje moguće je koristiti i alternativnu klasu danu kroz biblioteku podrške. Definira se sučelje za fragment u XML datoteci, slično kao i kod *Activitya*, stvara se objekt vlastite klase koja je naslijedila *Fragment* te se uporabom *LayoutInflater* iz XML-a napuhuje sučelje. Komunikacija između fragmenata obavlja se preko *Activitya* ili korištenjem događaja. Pojedinim je fragmentima moguće pristupiti preko *FragmentManagera* čija se instanca dobiva *getFragmentManager()* metodom. Fragmente je u sučelje moguće dodati kao fiksne elemente koristeći `<fragment>` oznaku. Ovo i nije strašno korisno, punu snagu fragmenti dobivaju dinamičkom uporabom korištenjem klase

FragmentManager koja omogućuje njihovo dodavanje, uklanjanje ili izmjenu tijekom izvođenja aplikacije. Svaka izmjena obavlja se pokretanjem takozvane transakcije metodom *beginTransaction()*, a svaka se transakcija potvrđuje i izvršava metodom *commit()*. Važna napomena kod stvaranja instanci klase fragment jest da se izbjegava kreiranje konstruktora. Naime, sustav će u ovisnosti o stanju aplikacije imati potrebu ponovno stvarati fragmente, a pri tom će rabiti podrazumijevani konstruktor. Ako se klasa koja predstavlja vlastiti fragment oslanja na parametre konstruktora, to može biti recept za katastrofu. Za ove potrebe rabi se stoga staticka metoda stvaranja, a ista će biti prikazana i primjerom.

Jedna od čestih situacija unutar koje se koriste fragmenti je niz ekrana između kojih se moguće kretati klizanjem slijeva na desno i obrnuto. Riječ je o vrlo uvriježenom UI obrascu na Android platformi, a kako bi se ostvario koristi se posebna klasa naziva *ViewPager*. Ova klasa može, uz korištenje odgovarajućeg adaptera, prikazati sadržaj na ovaj način, a jedan ekran u ovom slučaju predstavljen je jednim fragmentom.



Primjer 3.2. – ViewPager

Kako bi se na što lakši način približili fragmenti, kreirat će se jednostavan primjer u kojem se aplikacija sastoji od dva ekrana, jedan lijepi i jedan ružni. Svaki od ekrana bit će predstavljen fragmentom, a kretanje će osigurati *ViewPager*. Prvi korak prema implementaciji predstavlja kreiranje dvaju fragmenata, lijepog i ružnog. Za svaki je fragment potrebno definirati sučelje unutar odgovarajuće XML datoteke.

fragment_pretty.xml:

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/colorGold">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:src="@mipmap/ic_launcher"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

fragment_ugly.xml:

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/loremIpsum"
        android:gravity="center"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Nakon što je sučelje spremno, kreira se i klasa koja je dužna naslijediti klasu Fragment. Kako je ranije i navedeno, postoje dvije inačice ove klase, a u pravilu se rabi klasa iz biblioteke podrške.

PrettyFragment.kt

```
class PrettyFragment : Fragment() {

    companion object {
        fun newInstance(): PrettyFragment{
            return PrettyFragment()
        }
    }

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        val view = inflater.inflate(R.layout.fragment_pretty, container, false);
        return view;
    }
}
```

UglyFragment.kt

```
class UglyFragment : Fragment() {

    companion object {
        fun newInstance(): UglyFragment{
            return UglyFragment()
        }
    }

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {
        val view = inflater.inflate(R.layout.fragment_ugly, container, false);
        return view;
    }
}
```

Iako naizgled jednostavne, ove klase mogu biti od velike pomoći. Mogu upravljati prikazom na sličan način kao i Activity klase, moguće je na sučelje dodati kontrole, dodati osluškivanje događaja i sl. S obzirom da će se u primjeru objekti ovih klasa koristiti unutar ViewPager kontrole, nužno je kreirati i vlastiti Adapter.

HandsomeAdapter.kt

```
class HandsomeAdapter(fragmentManager: FragmentManager) : FragmentPagerAdapter(fragmentManager) {

    val fragments = arrayOf(
        PrettyFragment.newInstance(),
        UglyFragment.newInstance()
    )

    val titles = arrayOf("Pretty", "Ugly")

    override fun getItem(position: Int): Fragment {
        return fragments[position]
    }

    override fun getPageTitle(position: Int): CharSequence? {
        return titles[position]
    }

    override fun getCount(): Int {
        return fragments.size;
    }
}
```

Adapter klasa koja je kreirana nasljeđuje FragmentPagerAdapter. Kako bi ga mogla naslijediti, potrebno je osnovnoj klasi u konstruktoru poslati referencu na objekt klase FragmentManager kako bi se ispravno mogle obaviti transakcije (zamjene fragmenata). Osim te vrste adaptera, postoji još i FragmentStatePagerAdapter koji se rabi kada je potrebno rukovati većim brojem fragmenata. Razlika je u tome što točno čuvaju u memoriji dok fragmenti nisu vidljivi, FragmentPagerAdapter čuva sve fragmente, što u slučaju veće količine sadržaja nije baš poželjno.

Ono što je primjetno je da se za kreiranje objekta klase Pretty i Ugly fragment ne rabe konstruktori. Rabi se newInstance() metoda, a riječ je o ranije spomenutoj statičkoj metodi stvaranja. S obzirom da Kotlin ne poznaje statičke elemente, rabi se companion objekt u ove svrhe.

Nadalje, kako bi se podržalo korištenje ViewPagera, koji će zatražiti fragment od adaptera kada mu bude potreban, a kasnije i TabLayouta koji će dodati naslove nad svaki fragment, potrebno je implementirati tri metode – getCount(), getItem() i getPageTitle().

Ono što je preostalo sada jest spojiti Activity, njegovo sučelje i do sad napisan kod.

MainActivity.kt

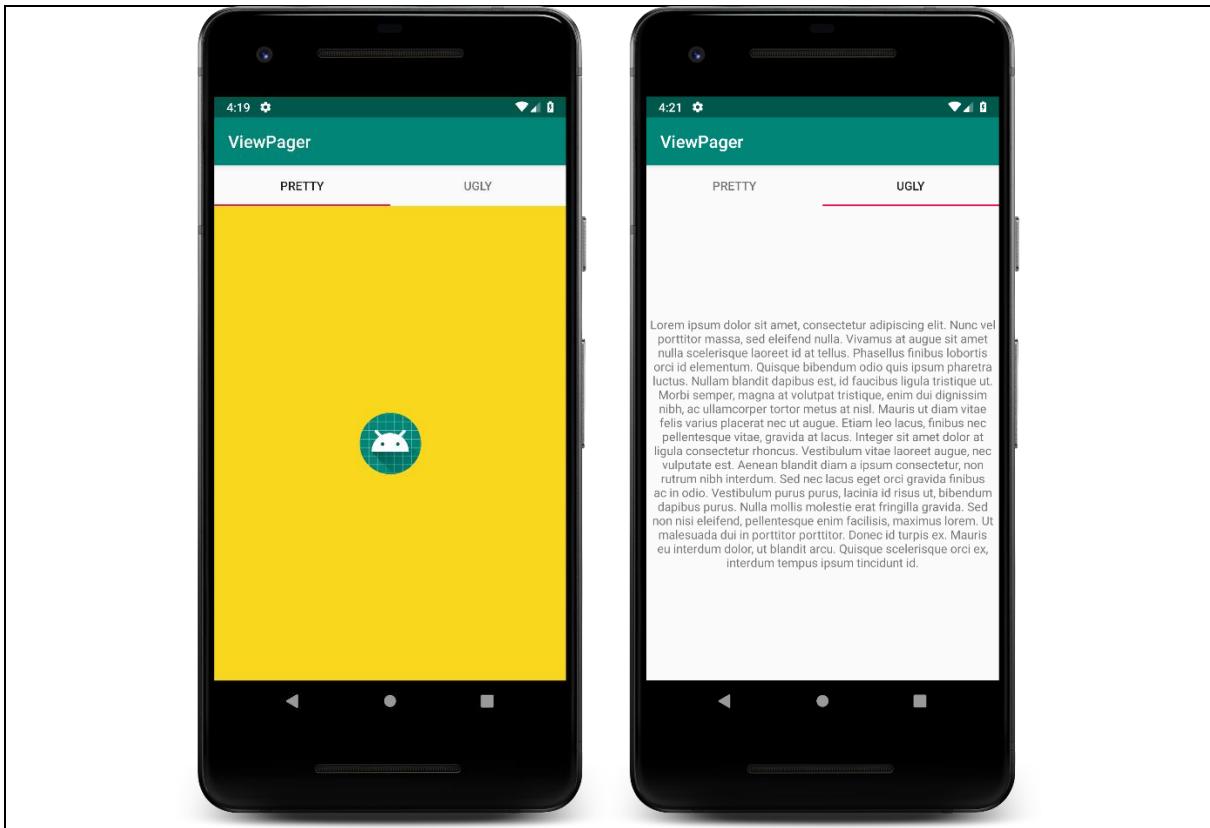
```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        viewPager.adapter = HandsomeAdapter(supportFragmentManager)
        tabLayout.setupWithViewPager(viewPager)
    }
}
```

activity_main.kt

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tabLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"/>
    <androidx.viewpager.widget.ViewPager
        android:id="@+id/viewPager"
        app:layout_constraintTop_toBottomOf="@+id/tabLayout"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```



3.2.2. Dijalozi

Dijalog je mali prozor koji dopušta interakciju s korisnikom i od njega uobičajeno zahtijeva nekakav unos, potvrdu ili odabir. Osnovna klasa za sve dijaloge je *Dialog*, ali uobičajeno se instancira jedna od izvedenih klasa. Također je uobičajeno da se koristi *DialogFragment* kao kontejner za dijaloge. Oni su uvijek dio *Activitya* i imaju fokus sve dok ih korisnik ne zatvori odabirom jedne od mogućnosti. Moguće je kreirati i vlastite dijaloge i izgled im definirati u XML-u. *AlertDialog* (And. 3.0.) jedna je od najraznovrsnijih implementacija dijaloga. Nudi brojne opcije koje omogućuju kreiranje dijaloga za najčešće slučajeve. Za kreiranje objekta koristi *Builder* oblikovni obrazac (vežu se metode i postupno dodaju parametri kod kreiranja objekta). Podržano je dodavanje gumba, liste s jednostrukim ili višestrukim izborom, vlastitog izgleda i sl.



Primjer 3.2. – Dijalozi

Potrebno je kreirati novi projekt s jednim Activityem čije sučelje sadržava samo jedan gumb. Postavlja se osluškivanje na klik gdje se kreira objekt vlastite *ExitDialog* klase te se na tom objektu poziva *show()* metoda kako bi se dijalog prikazao. Vlastita klasa nasljeđuje osnovnu klasu *DialogFragment* i u prepisanoj *onCreateDialog()* metodi postavlja osnovne podatke za dijalog koristeći *Builder* oblikovni obrazac (primjerice, *setTitle()*).

**Pri izlasku iz aplikacije nikada nemojte davati dijalog za izlaz, ovo je samo primjer.*

activity_main.xml:

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/dialogAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        android:text="@string/dialogActionText"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.kt

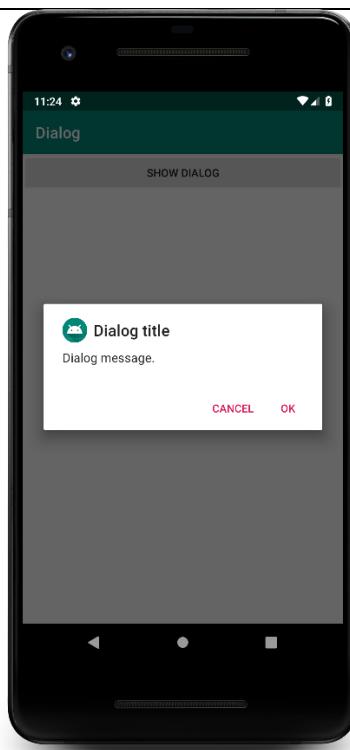
```
const val TAG = "RMA_Dialogs"

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        dialogAction.setOnClickListener(displayDialog())
    }

    private fun displayDialog() {
        AlertDialog.Builder(this)
            .setTitle(R.string.dialogTitle)
            .setMessage(R.string.dialogMessage)
            .setIcon(R.mipmap.ic_launcher)
            .setPositiveButton(android.R.string.yes){ dialog, which ->
                Log.d(TAG, "POSITIVE")
            }
            .setNegativeButton(android.R.string.no) { dialog, which ->
                Log.d(TAG, "NEGATIVE")
            }
            .create()
            .show()
    }
}
```

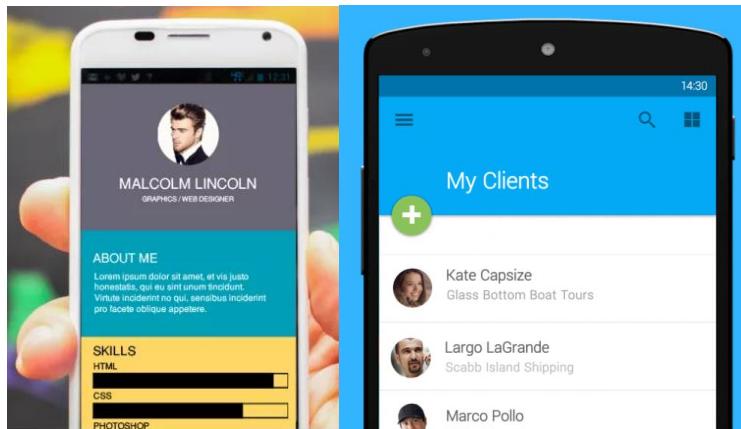


3.3. Primjeri



Zadatak 1.

Kreirajte aplikaciju koja predstavlja Vaš životopis i za te potrebe koristi ViewPager, TabLayout i fragmente. Potrebna su tri fragmenta, jedan za kratku biografiju sa slikom, jedan za obrazovanje i jedan radno iskustvo. Obrazovanje i radno iskustvo treba biti prikazano u obliku liste sadržaja poredane po starosti, od najnovijih do najstarijih (koristiti RecyclerView klasu).



3.4. Zadaća

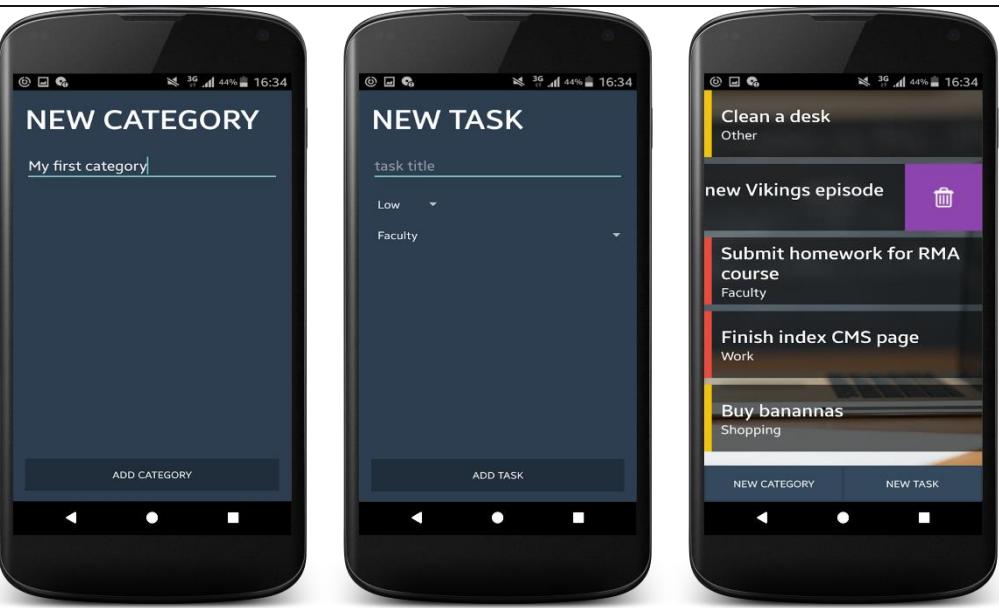


Zadaća 3.

Tasky - Aplikacija za pohranu obaveza

Potrebno je kreirati aplikaciju koja koristi jednostavnu bazu podataka za kreiranje korisničkih zabilješki(neka vrsta To do liste). Nužno je omogućiti korisniku kreiranje zabilješki, spremanje u bazu, pregled zabilješki u obliku liste te njihovo brisanje. Koristiti primjere u ovoj vježbi te naučeno na proteklim vježbama.

- ➊ Kreirati ListActivity prema slici koji će služiti kao početni zaslon, Listu popuniti iz baze.
- ➋ Kreirati vlastitu klasu Task koja će predstavljati zadatak, potrebno najmanje imati atribute za naslov, tekst zadatka, i sliku koja predstavlja prioritet (crveno, žuto zeleno)
- ➌ Definirati izgled elementa liste u XML-u
- ➍ Kreirati bazu po uzoru na LV i kreirati bazu i u njoj tablicu za pohranu
- ➎ Kreirati dodatni Activity ili Fragment za unos zadataka koji se pokreće klikom na gumb iz prvog Activitya ili Fragmenta
- ➏ Implementirati brisanje na dugi klik na element liste



Napomena:

Svaka je nadogradnja poželjna i dobrodošla. Korištenje znanja koja nadilaze opseg dosadašnjih vježbi se potiče.



“Without requirements or design, programming is the art of adding bugs to an empty text file.”

- Louis Srygley



LV4:

Rad s udaljenim resursima

4. Laboratorijska vježba 4

4.1. Uvod

Ova vježba opisuje metode rada s udaljenim resursima, u prvom redu s podacima i uslugama dostupnim u obliku web usluga (konkretnije REST usluge). U tu svrhu moguće je rabiti brojne klase koje su dio standardne Java ili dolaze kao dio Android SDK. Kako se radi o vrlo čestoj potrebi unutar aplikacije (ne samo kod mobilnih aplikacija), a s obzirom da je sve zahtjevnije zadatke nužno ukloniti s glavne niti, odnosno niti koja obrađuje UI događaje, često se rabe gotove biblioteke koje ovaj posao značajno olakšavaju. Prikazano je tako u sklopu ove vježbe dohvaćanje i konzumiranje resursa s Interneta te obrada informacija u XML i JSON formatima. Uz sve navedeno, dani su primjeri korištenja primatelja emitiranja (engl. *Broadcast Receiver*) radi izvršavanja određenih dijelova koda prilikom nekog događaja (npr. pristizanje SMS poruke), opisane su notifikacije te rad s dozvolama.

4.1.1. Potrebna predznanja

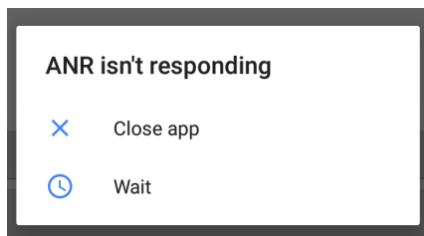
- Osnove programiranja
 - Kolegiji Programiranje I, Programiranje II, Algoritmi i strukture podataka
- Osnove objektno orijentiranog programiranja
 - Kolegij Objektno orijentirano programiranje
- Osnove izrade Android aplikacija (activity, osnove UI-ja)
 - Predložak LV1-4, predavanja
- Osnove Java programskog jezika
 - Java, A Beginner's Guide, 5th Edition - Herbert Schildt
 - Thinking in Java, 4th edition – Bruce Eckel
 - <http://docs.oracle.com/javase/tutorial/>
- Intent - <http://developer.android.com/reference/android/content/Intent.html>
- Adapter design pattern - http://sourcemaking.com/design_patterns/adapter

4.1.2. Korisna predznanja

- XML - <http://www.w3.org/TR/REC-xml/>
- JSON - <http://www.json.org/>
- RSS - <http://www.w3schools.com/rss/>
- Niti - <http://developer.android.com/guide/components/processes-and-threads.html>

4.2. Pozadinska obrada zadataka

Svi dosadašnji primjeri posao su obavljali unutar jedne niti, a bila je riječ o glavnoj, odnosno UI niti. Problem s ovim pristupom leži u činjenici da će doći do situacija u kojima će posao koji radi glavna niti biti previše zahtjevan, posljedica čega će biti sporija obrada UI događaja (glava će nit biti blokirana drugim poslom). Korisnici aplikacije ovo će osjetiti kroz trzanje, usporavanje i gubitak odziva (engl. *lag, unresponsiveness*). Sam Android sustav ima ugrađeno rješenje za ovakve prilike, a riječ je o ANR (engl. *application not responding*) dijalogu koji će se javiti nakon što glavna nit bude blokirana određeno vrijeme. Korisnik će kroz ovaj dijalog dobiti priliku „ubiti“ proces, odnosno prisilno izaći iz aplikacije. Odziv unutar ovog vremena (5 sekundi) ne samo da ne treba biti cilj, nego je nužno izbjegavati bilo kakvu ne-interaktivnost. Korisnik primjećuje i na njegovo iskustvo negativno utječe svako zastajkivanje dulje od približno 200 milisekundi. Kako bi se ovaj problem riješio, zahtjevni zadaci se prebacuju na izdvojene, pozadinske niti ili dretve (engl. *thread*) kako bi se glavna nit ostavila dostupnom korisničkoj interakciji. U nekim slučajevima (npr. pristup resursima preko mreže u Android verzijama 3.0+), zahtjevne operacije prema mrežnim resursima nije niti moguće implementirati na glavnoj niti, već se javlja greška.



Slika 4.1. Primjer ANR dijaloga

4.2.1. Niti

Android podržava brojne načine pozadinske obrade podataka, a niti jedan od njih nije dobar za apsolutno sve potrebe već se biraju prema vrsti zadatka koji je potrebno obaviti. Jedan od osnovnih su niti koje su poznate u brojnim programskim jezicima. Koristi se klasa *Thread*, poznata iz Java, koja implementira sučelje *Runnable*. Time je ova klasa dužna implementirati *run()* metodu koja zapravo odrađuje posao van glavne niti. Nit koja odrađuje nekakav posao nije sinkronizirana s UI niti, tako da nije moguće mijenjati niti pristupati korisničkom sučelju iz drugih niti. Pokušaj takve izmjene dovodi do podizanja iznimki. Da bi se navedeno ipak omogućilo, koristi se koncept nazvan *Handler* i metoda *postRunnable()* ili metoda *runOnUiThread()* kojoj se predaje *Runnable* objekt kao argument. Iako je uporaba niti vrlo korisna, može dovesti do brojnih problema ukoliko se ne obraća pažnja na detalje. Ako se sav posao odrađuje „pješke“, onda je potrebno voditi

računa o sinkronizaciji, poništavanju niti, promjenama konfiguracije (npr. orijentacije uređaja) i sl. Da bi se donekle olakšao rad s nitima, u Androidu postoji posebna klasa koja rješava dio ovih problema, nazvana *AsyncTask*, a opisana je u narednom poglavlju. Uz nju, često se koristi *IntentService*, za obavljanje dugotrajnijih zadataka za koje se ne zna kada će završiti.



Primjer 4.1. – Korištenje niti za obradu jednostavnih zadataka

Kreira se osnovna klasa *Activity* koja sadrži dva gumba i jedan *checkbox*. Jedan gumb pokrenut će simulirani posao na glavnoj niti, dok će drugi gumb isti posao odraditi na pozadinskoj niti. U ovom primjeru za simuliranje posla rabi se statička blokirajuća metoda *sleep()* prisutna unutar *Thread* klase. Ona je pogodna samo za primjere i ni za što drugo, a stvarni posao koji se obavlja može biti interakcija s bazom podataka, mrežni zahtjev, obrada slike ili nešto posve drugo.

activity_main.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <CheckBox
        android:id="@+id/testBox"
        android:layout_width="match_parent"
        android:layout_height="@dimen/uiTesterHeight"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_alignParentTop="true"
        android:text="@string/testBoxText"/>
    <Button
        android:id="@+id/mainThreadRunAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/testBox"
        android:layout_centerInParent="true"
        android:text="@string/mainThreadRunActionText"/>
    <Button
        android:id="@+id/backThreadRunAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/mainThreadRunAction"
        android:layout_centerInParent="true"
        android:layout_below="@+id/mainThreadRunAction"
        android:text="@string/backThreadRunActionText"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

MyThread.kt

```
class MyThread(
    val waitTime: Long,
    val repetitions: Int): Thread() {

    val TAG = "Thread"
    val message = "A sneaky background thread done."
    val startedMessage = "Started working in background."

    override fun run() {
        Log.d(TAG, startedMessage)
        try{
            for (i in 1..repetitions){
                Thread.sleep(waitTime)
            }
            Log.d(TAG, message)
        } catch (e: InterruptedException){
            Log.e(TAG, e.message)
        }
    }
}
```

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    val TAG = "Thread"
    val finishedMessage = "Glorious main thread. Work done."
    val startedMessage = "Started working on main thread."

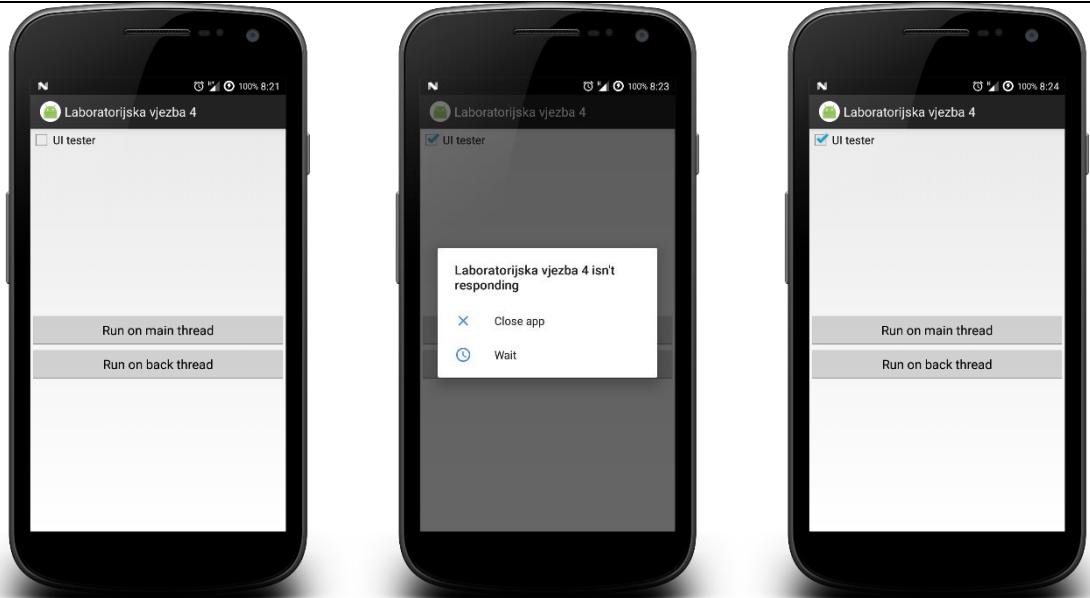
    val waitTime = 1000L;
    val repetitions = 10;

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        mainThreadRunAction.setOnClickListener { runHardWorkOnUi() }
        backThreadRunAction.setOnClickListener { runHardWorkInBackground() }
    }

    private fun runHardWorkOnUi() {
        Log.d(TAG, startedMessage)
        try {
            for (i in 1..repetitions) {
                Thread.sleep(waitTime)
            }
            Log.d(TAG, finishedMessage)
        } catch (e: InterruptedException) {
            Log.e(TAG, e.message)
        }
    }

    private fun runHardWorkInBackground() {
        MyThread(waitTime, repetitions).start()
    }
}
```



4.2.2. Asinkroni zadaci (engl. *Asynchronous task*)

Rad s nitima relativno je kompleksan, a jedan od razloga je i nužnost sinkronizacije s glavnom niti radi osvježavanja sučelja. Android stoga nudi pomoćnu klasu koja se naziva *AsyncTask* i čija je namjera izdvajanje **kratkotrajnog** zahtjevnog posla na zasebnu nit, ali i izlaganje metoda koje se izvode na UI niti i omogućuju lak prikaz podataka korisniku. Uobičajeno se implementira kao statička ugniježđena klasa klase koja ga rabi, primjerice

Activitya. Kako bi se koristio, potrebno je definirati vlastitu klasu koja nasljeđuje klasu *AsyncTask* i prepisati njene metode. Zatim je nužno instancirati klasu i pozvati metodu *execute()* na objektu. Svaki se asinkroni zadatak može izvršiti jednom, a u slučaju da je potrebno više puta koristiti zadatak, potrebne su njegove nove instance. Iako koristan, *AsyncTask* je nužno rabiti s oprezom. Ako se implementira kao unutarnja klasa, s obzirom da (kao i svaka) unutarnja klasa drži referencu na vanjsku klasu može doći do curenja memorije. Treba voditi računa i otkazivanju zadatka u slučaju uništavanja *Activitya* čije se sučelje osvježava, primjerice u slučaju rotacije ekrana. U ovakvim se slučajevima bolje osloniti na servise ili neku drugu alternativu (primjerice RxJava). Ako se rabi *AsyncTask*, tada se prepisuju četiri njegove metode

- *onPreExecute()*
- *onProgressUpdate()*
- *onPostExecute(Result)*
- *doInBackground(Params...)*

Sve od navedenih metoda, izuzev *doInBackground(Params...)*, izvršavaju se na glavnoj niti. Služe redom za priređivanje postavki za zadatak, obavještavanje korisnika o napretku te prikazu rezultata korisniku. Sva obrada na pozadinskoj niti obavlja se unutar *doInBackground(Params...)* metode. Kako *AsyncTask* koristi *genericse*, nužno je odrediti tri tipa podatka prilikom nasljeđivanja ove klase, a to je tip podataka s kojima će se raditi, tip podataka kojim će se obavještavati o napretku te tip podataka koji će biti korišten za rezultat (prva dva su argumenti varijabilne duljine, pa se stoga radi s poljima, kako je prikazano u primjeru).



Primjer 4.2. – Korištenje asinkronih zadataka

Kreira se jednostavno korisničko sučelje koje sadrži tek gumb za dohvaćanje podataka, listu za prikaz podataka i jedan skriveni *ProgressBar* koji će poslužiti za prikaz napretka. On je skriven postavljanjem *visibility* atributa na GONE vrijednost, što znači da ne samo da nije vidljiv na ekranu nego ne zauzima prostor. Ovo je nešto drugačije od postavljanja na INVISIBLE gdje također neće biti iscrtan, ali će zauzeti prostor na ekranu koji će tada djelovati prazno.

activity_main.xml

```

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ProgressBar
        android:id="@+id/progressDisplay"
        android:layout_width="match_parent"
        android:layout_height="@dimen/progressViewHeight"
        android:layout_centerInParent="true"
        android:max="100"
        style="@style/Base.Widget.AppCompat.ProgressBar.Horizontal"
        android:visibility="gone"/>
    <Button
        android:id="@+id/getCountryInfoAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:text="@string/getCountryInfoText"/>
    <ListView
        android:id="@+id/countriesInfo"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/getCountryInfoAction"/>
</RelativeLayout>

```

Asinkroni zadatak rabit će se za dohvaćanje informacija o državama. Ovaj naš neće to zbilja raditi (iako nema nikakve zapreke) zbog jednostavnosti primjera, samo će za predana imena država pričekati po sekundu (čekanje glumi obradu) i onda dodati string „I <3 „ime_države“ u listu s rezultatima. S obzirom da se poslovi poput zahtjevnije obrade ili pristupa Internetu ne trebaju ili čak ne smiju obavljati na glavnoj niti, kada je potrebno obaviti takav posao, to se delegira na asinkroni zadatak i njegovu metodu *dolnBackground()*. Activity ovdje sadrži nekoliko metoda za prikaz informacija na sučelju i metodu koja se poziva kod klika na gumb, a koja okida asinkroni zadatak. Važno je zapamtiti kako asinkrone zadatke nije moguće pokrenuti ponovno jednom nakon što su ispaljeni već je potrebno uvijek kreirati novu instancu i pokrenuti nju. Zadatak se pokreće pozivom *execute()* metodi koja može primiti parametre.

MainActivity.kt

```

class MainActivity : AppCompatActivity() {

    private val countries = arrayOf("Croatia", "Uganda", "Italy", "Utopia")

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        getCountryInfoAction.setOnClickListener{
            CountriesTask(this).execute(*countries)
        }
    }

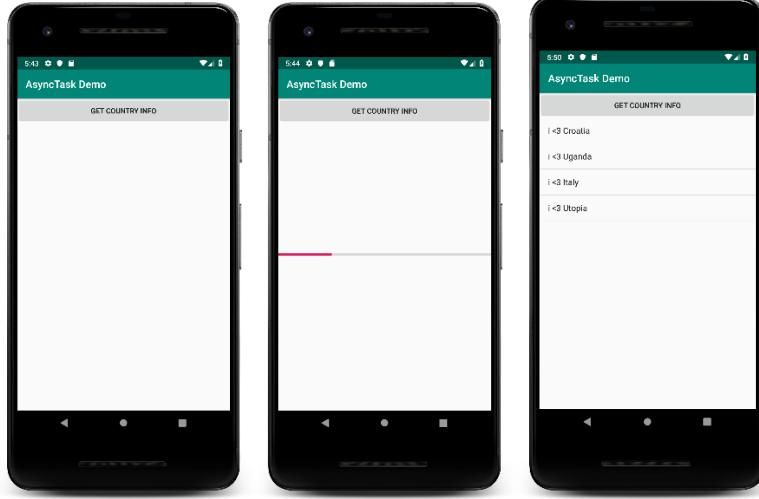
    fun displayProgressDialog(){
        progressDisplay.visibility = View.VISIBLE
    }

    fun hideDisplayProgressDialog(){
        progressDisplay.visibility = View.GONE
    }

    fun updateProgress(progress: Int){
        progressDisplay.setProgress(progress)
    }

    fun displayResults(results: List<String>){
        countriesInfo.adapter = ArrayAdapter(
            this,
            android.R.layout.simple_list_item_1,
            results
        )
    }
}

```



Klasa `CountriesTask` predstavlja asinkroni zadatak jer nasljeđuje `AsyncTask` klasu. S obzirom da je `AsyncTask` generička klasa, potrebno ju je parametrizirati navođenjem konkretnih tipova koji će se rabiti za ulazne podatke, podatke za obavještavanje o napretku te podatke koji sadrže rezultat. U našem slučaju, predajemo kolekciju stringova, ažuriranje sučelja obavlja se u postotcima (cjelobrojna vrijednost), a rezultat će biti lista stringova. Jedina metoda koja se uistinu odvija na drugoj niti je `doInBackground()`. Ondje se smješta sav težak posao. Bitna napomena jest da je nužno osigurati da `MainActivity` koji je poslan kao argument konstruktoru ove klase bude omotan `WeakReference` objektom. Razlog leži u činjenici da takav tip reference ne sprječava GC da pokupi objekt jednom kada su izgubljene sve čvrste reference na njega. Da ovo nije napravljeno, asinkroni bi zadatak sam imao čvrstu referencu na `MainActivity` zajedno sa svim objektima koje ovaj sadrži, što bi predstavljalo curenje memorije.

CountriesTask.kt

```
class CountriesTask(mainActivity: MainActivity) : AsyncTask<String, Int, List<String>>() {

    val activity: WeakReference<MainActivity> = WeakReference(mainActivity)

    override fun onPreExecute() {
        super.onPreExecute()
        (activity.get() as MainActivity).displayProgressDialog()
    }

    override fun onProgressUpdate(vararg values: Int?) {
        super.onProgressUpdate(*values)
        val progress = values.first() ?: 0
        (activity.get() as MainActivity)?.updateProgress(progress)
    }

    override fun doInBackground(vararg params: String?): List<String> {
        val countriesInfo = mutableListOf<String>()

        for(i in 0 until params.size){
            countriesInfo.add("i <3 ${params[i]}")
            val percent = i.toDouble() / params.size * 100
            publishProgress(percent.toInt())
            Thread.sleep(1000)
        }
        return countriesInfo
    }
}
```

```
    override fun onPostExecute(result: List<String>?) {
        super.onPostExecute(result)
        (activity.get() as MainActivity)?.let{
            it.hideDisplayProgressDialog()
            val results = result ?: listOf()
            it.displayResults(results)
        }
    }
}
```

4.2.1. Anko biblioteka i korutine

Kako je vidljivo iz prethodnih primjera, rad u pozadini je prilično kompleksna stvar i zahtjeva pažnju oko detalja. Također je dosta značajno kognitivno opterećenje, a u slučaju korištenja rješenja poput AsyncTaska traži pisanje dosta boilerplate koda. U Kotlinu postoji zgodno rješenje za rad s asinkronim poslovima, a naziva se korutine (engl. *coroutines*). Više o njima je moguće pročitati na <https://kotlinlang.org/docs/reference/coroutines-overview.html> te <https://kotlinlang.org/docs/reference/coroutines/coroutines-guide.html> i <https://antonioleiva.com/coroutines/>.

Međutim, tvorci Kotlin jezika ponudili su biblioteku koja olakšava posao s dosta standardnih Android radnji poput stvaranja dijaloga, toast poruka i sl. među čime je i asinkrona obrada zadataka te osvježavanje sučelja. Ta se biblioteka naziva Anko i moguće ju je pronaći na <https://github.com/Kotlin/anko>. Primjerom 4.2. pokazano je kako je na izuzetno jednostavan način moguće neki posao obaviti asinkrono uz osvježavanje korisničkog sučelja.



Primjer 4.3. – Korištenje Anko biblioteke

Kreirat ćemo jednostavan primjer nalik ranijim primjerima. Sučelje će sadržavati gumb, *TextView* i *CheckBox* za provjeru funkciranja li sučelje. Korištenjem Anko korutina pozivat ćemo zloglasnu *Thread.Sleep()* metodu.

build.gradle (app)

```
implementation "org.jetbrains.anko:anko-commons:0.10.8"
```

activity_main.xml

```

<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <CheckBox
        android:layout_width="match_parent"
        android:layout_height="@dimen/uiTesterHeight"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        android:text="@string/uiTesterText"/>

    <TextView
        android:id="@+id/messageDisplay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
    <Button
        android:id="@+id/startWorkAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/messageDisplay"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/startWorkText"/>
</android.support.constraint.ConstraintLayout>

```

MainActivity.kt

```

class MainActivity : AppCompatActivity() {

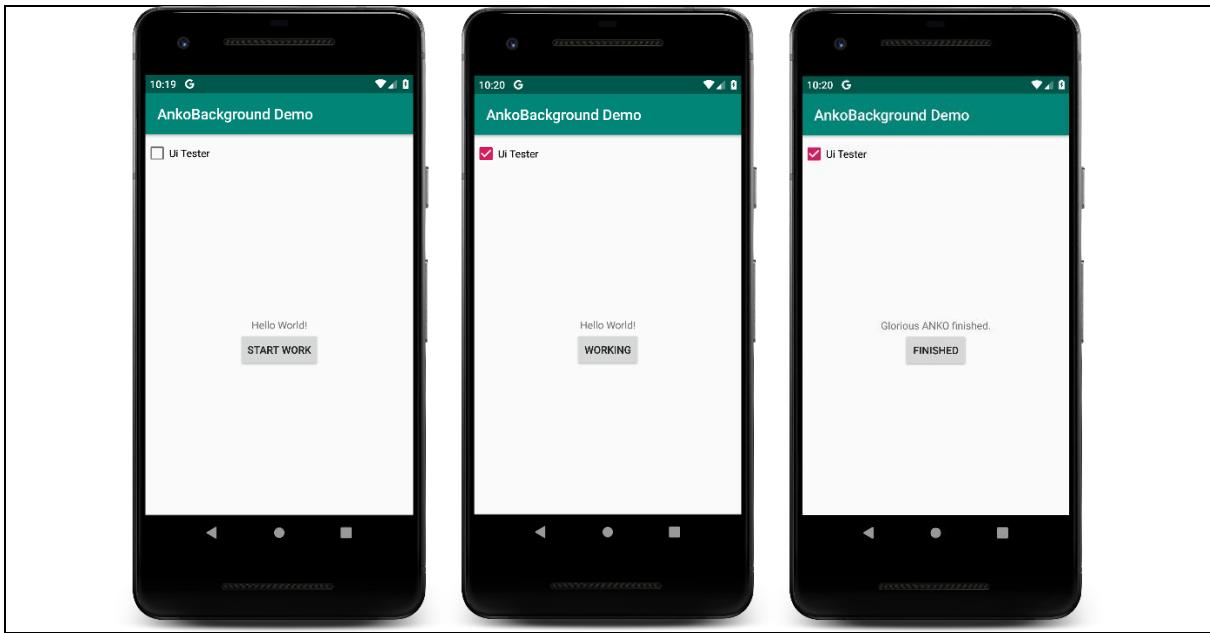
    private val repeatCount = 5
    private val finishedMessage = "Glorious ANKO finished."
    private val started = "Started"
    private val working = "Working"
    private val finished = "Finished"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        startWorkAction.setOnClickListener{ runHeavyWorkload() }
    }

    fun runHeavyWorkload(){
        startWorkAction.text = started
        doAsync {
            for (i in 1..repeatCount){
                Thread.sleep(1000)
                uiThread {
                    messageDisplay.text = i.toString()
                }
            }
            uiThread{
                messageDisplay.text = finishedMessage
                startWorkAction.text = finished
            }
        }
        startWorkAction.text = working
    }
}

```



4.2.2. RESTful usluge

REST označava *REpresentational State Transfer*. Riječ je o arhitekturi koja se oslanja na HTTP protokol i resurse. Svaka komponenta je resurs, a resursima se pristupa kroz zajedničko sučelje korištenjem standardnih HTTP metoda. Server ovdje samo daje klijentu pristup resursima, a klijent pristupa i mijenja resurse. Svaki resurs identificiran je URI-jem. Kako bi predstavio resurs, REST rabi različite reprezentacije (prijenosne formate), a najčešći su JSON i XML. Standardne metode koje se rabe su

- GET – pruža pristup čitanja
- POST – za kreiranje novog resursa
- DELETE – za uklanjanje resursa
- PUT – osvježavanje postojećeg ili stvaranje novog resursa

Web usluge koje rabe REST arhitekturu nazivaju se RESTful uslugama. RESTful usluga uobičajeno definira URI, pruža reprezentaciju resursa u JSON obliku te nudi skup HTTP metoda.



Više o RESTu

<https://www.restapitutorial.com/>

4.2.2.1. Rad s podacima u JSON-u

Efikasan način za predstavljanje informacija je JSON (JavaScript Object Notation). Radi se o laganom formatu za razmjenu podataka koji je lako čitljiv ljudima, a istovremeno jednostavan za generiranje, parsiranje i obradu na računalima. Primjer izgleda JSON-a je:

```
{
  "Search": [
    {
      "Title": "Arrival",
      "Year": "2016",
      "imdbID": "tt2543164",
      "Type": "movie",
      "Poster": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTExMzU0ODcxNDheQTJeQWpwZ15BbWU4MDE1OTI4MzAy._V1_SX300.jpg"
    },
    ...
  ]
}
```

Slika 4.2. Primjer JSON-a

Slikom 4.3 prikazan je JSON objekt koji je vraćen prilikom upita na isti poslužitelj. Unutar JSON objekta nalazi se JSON *array* pod nazivom *Search* koje sadržava nove JSON objekte. Objekti unutar elemenata polja sadržavaju parove ključ/vrijednost koji opisuju pojedini film. Već je na prvi pogled vidljivo kako je ovaj format znatno jednostavniji od XML-a.

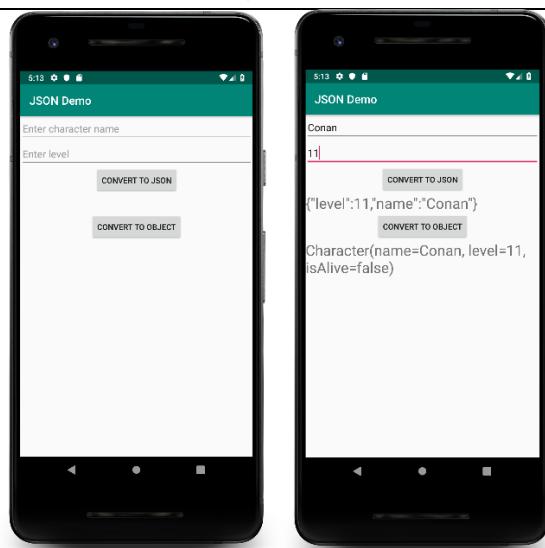


Slikom 4.3 prikazan je formatirani JSON, a moguće ga je formatirati na
<https://jsonformatter.curiousconcept.com/>



Primjer 4.4. – Jednostavan rad s JSON podacima

Prije nego započnemo raditi sa zahtjevima prema poslužitelju, razmotrimo kako se može rabiti JSON unutar vlastitog koda. Kreirat ćemo jednostavan program koji radi konverziju iz objekta vlastitih klasa u JSON reprezentaciju, a nakon toga ćemo iz JSON reprezentacije vratiti originalan objekt. Iako bi cijelu ovu konverziju mogli obaviti i ručno, dodajući pojedine elemente u JSON string, u pravilu se to ne isplati. Pamatniji bi pristup bio rabiti neku od biblioteka za rad s JSON formatom, a najčešće korištene su GSON (Google), Moshi (Square) i Jackson. Primijetite kako je kod pretvaranja u JSON i obrnuto riječ o obliku serijalizacije, odnosno deserijalizacije. To je još jedan od načina, uz Serializable i Parcelable kako je moguće serijalizirati objekte. Iste ideje koje se primjenjuju u ovom primjeru moguće je primijeniti kod rada s XML transportnim formatom. U primjeru će biti korištena GSON biblioteka koja će se rabiti i u ostatku ove vježbe.



build.gradle (app)

```
implementation 'com.google.code.gson:gson:2.8.5'
```

Kako bi Gson biblioteka znala raditi s našim klasama, potrebno je dodati određene anotacije koje govore biblioteci što točno unutar objekta preslikava u JSON string. Ako ne želimo da se unutar JSON reprezentacije rabe imena koja smo koristili unutar klase već neka druga, koristimo anotaciju @SerializedName. Ako je potrebno izostaviti neki od atributa, onda je moguće rabiti @Transient anotaciju (ili @Exposed koja će označiti ono što se želi serijalizirati).

Character.kt

```
data class Character (
    @SerializedName("name") val name: String,
    @SerializedName("level") val level: Int,
    @Transient val isAlive: Boolean
)
```

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    val gson: Gson = GsonBuilder().create()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        convertToObjectAction.setOnClickListener{ convertFromJsonAndDisplay() }
        convertToJsonAction.setOnClickListener{ convertToJsonAndDisplay() }
    }

    private fun convertToJsonAndDisplay() {
        val name = characterNameInput.text.toString()
        val level = characterLevelInput.text.toString().toInt()

        val character = Character(name, level, isAlive = true)
        val jsonCharacter = gson.toJson(character)

        jsonDisplay.text = jsonCharacter
    }

    private fun convertFromJsonAndDisplay(){
        val jsonCharacter = jsonDisplay.text.toString()
        val character = gson.fromJson(jsonCharacter, Character::class.java)

        objectDisplay.text = character.toString()
    }
}
```

activity_main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <EditText
        android:id="@+id/characterNameInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        android:hint="@string/characterNameInputHint"/>
    <EditText
        android:id="@+id/characteLevelInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/characterNameInput"
        android:hint="@string/characterLevelInputHint"
        android:inputType="numberSigned"/>
    <Button
        android:id="@+id/convertToJsonAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/characteLevelInput"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/convertToJsonText"/>
    <TextView
        android:id="@+id/jsonDisplay"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/convertToJsonAction"
        android:textSize="@dimen/textSizeNormal"/>
    <Button
        android:id="@+id/convertToObjectAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/jsonDisplay"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/convertToObjectText"/>
    <TextView
        android:id="@+id/objectDisplay"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/convertToObjectAction"
        android:textSize="@dimen/textSizeNormal"/>
</android.support.constraint.ConstraintLayout>

```

Osim jednostavnog korištenja JSON-a unutar vlastite aplikacije (primjerice, dodavanje JSON datoteke unutar *asset* mape u resursima i korištenje deserijalizacije kako bi se učitali podaci), odgovor koji se dobije od servera često je zapakiran u obliku JSON stringa. Kada je potrebno napraviti zahtjev prema serveru, potrebno je napraviti *http* zahtjev korištenjem neke od ranije opisanih *http* metoda prema određenoj krajnjoj točki određenoj URI-jem. Od servera se dobiva odgovor u *http* formatu sa statusnim kodom te sadržajem u tijelu odgovora, ako je na upit uspješno odgovoren. Iako je moguće kod koji se spaja na server (u pozadinskoj niti, komunikacija s mrežom nije dopuštena na glavnoj niti) napisati ručno, u pravilu ovo predstavlja značajnu količinu posla, traži veliku pažnju prema detaljima i nije isplativo. Bolja je varijanta rabiti neku od prokušanih biblioteka poput Retrofit-a, Volley-a i sl. U ostatku ove vježbe bit će korištena biblioteka Retrofit. Ona omogućuje kreiranje sučelja koja predstavljaju odgovarajuće upite prema željenim krajnjim točkama poslužitelja kojima se pristupa traženim resursima. Ova sučelja su anotirana, a

implementaciju generira sam Retrofit. Komunikacija prema poslužitelju obavlja se korištenjem OKHttp biblioteke, ali istu je moguće i zamijeniti. Dobiveni odgovor moguće je odmah deserijalizirati iz transportnog formata u kojeg je zapakiran uporabom prikladnog konvertera (npr. za JSON Moshi, Gson, Jackson, za XML SimpleXmlParser itd.).



Primjer 4.5. – Dohvaćanje podataka s poslužitelja u JSON formatu

Za potrebe izrade ovog primjera koristit će se besplatan servis naziva TvMaze. Riječ je o servisu koji omogućuje dobivanje informacija o TV serijama. Informacije o API-ju dostupne su na <https://www.tvmaze.com/api>. Ovo je samo jedan od mnogobrojnih besplatnih servisa dostupnih za pristup podacima, a zgodan katalog dostupan je na <https://www.programmableweb.com/>. Za korištenje *Retrofit-a* potrebno je izmijeniti *gradle* datoteku prema primjeru. Također odmah dodajemo i *RecyclerView*, Picasso te *LoggingInterceptor* za lakše *debuggiranje* pogrešaka vezanih uz http zahtjeve. Nužno je još u android *closure-u* dodati *compileOptions* jer se dio biblioteka koje rabimo oslanja na funkcionalnosti dostupne od Jave 8.

build.gradle (app)

```
// unutar android{ }
compileOptions {
    sourceCompatibility 1.8
    targetCompatibility 1.8
}

implementation 'com.squareup.retrofit2:retrofit:2.5.0'
implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
implementation 'com.android.support:recyclerview-v7:28.0.0'
implementation 'com.squareup.okhttp3:logging-interceptor:3.14.1'
implementation ('com.squareup.picasso:picasso:2.71828'){
    exclude group: 'com.android.support'
    exclude module: ['exifinterface', 'support-annotations']
}
```

S obzirom da želimo omogućiti korisnicima pretragu TV serija, pogledajmo najprije kako izgleda api na koji se spajamo:

<http://api.tvmaze.com/search/shows?q=sherlock>

Osnovni URL je <https://api.tvmaze.com/> gledamo endpoint search/shows na koji http GET metodom radimo upit i šaljemo parametar naziva q kojem smo dodijelili vrijednost sherlock. Kao rezultat, očekujemo (jer to propisuje dokumentacija API-ja) polje JSON objekata kako se može vidjeti ako se klikne na link. Iako postoje brojni alati na Internetu koji će na temelju ovog JSON-a generirati odgovarajuće klase, primjera radi sami ćemo napraviti odgovarajuće klase. Prva koja nam treba predstavlja jedan SearchResult objekt. Svaki rezultat sadrži seriju, a svaka serija ima poster. To se odražava i u našim podatkovnim klasama.

SearchResult.kt

```
data class SearchResult (
    @SerializedName("score") val score: Double,
    @SerializedName("show") val show: Show
)
```

Show.kt

```
data class Show (
    @SerializedName("id") val id: Long,
    @SerializedName("name") val name: String,
    @SerializedName("summary") val summary: String,
    @SerializedName("image") val thumbnailUrl: Thumbnail
)
```

Thumbnail.kt

```
data class Thumbnail(
    @SerializedName("medium") val smallThumbnailUrl: String,
    @SerializedName("original") val largeThumbnailUrl: String
)
```

Nadalje, kako ćemo koristiti Internet, potrebno je u manifest datoteci zatražiti dopuštenje. Ono što je još potrebno jest dopustiti komunikaciju preko http protokola, jer novije inačice Androida dopuštaju samo https. S obzirom da nemamo kontrolu nad poslužiteljem, ovo također radimo u manifestu unutar application taga.

Manifest.xml

```
<uses-permission android:name="android.permission.INTERNET"/>

<application
    android:usesCleartextTraffic="true"
```

Kreiramo zatim vlastito sučelje koje predstavlja web uslugu te kreiramo, pomoću Retrofit objekta, njegovu implementaciju.

TvMazeApi.kt

```
const val BASE_URL = "https://api.tvmaze.com/"

interface TvMazeApi {

    @GET("search/shows")
    fun findShow(@Query("q") name: String): Call<List<SearchResult>>
}
```

Networking.kt

```
object Networking{
    val showSearchService: TvMazeApi = Retrofit.Builder()
        .addConverterFactory(ConverterFactory.converterFactory)
        .client(HttpClient.client)
        .baseUrl(BASE_URL)
        .build()
        .create(TvMazeApi::class.java)
}

objectConverterFactory{
    val converterFactory = GsonConverterFactory.create()
}

object HttpClient{
    val client = OkHttpClient.Builder()
        .addInterceptor(HttpLoggingInterceptor().setLevel(HttpLoggingInterceptor.Level.BODY))
        .build()
}
```

Preostalo je još kreirati sučelje za Activity, sučelje za jedan item te u MainActivityu iskoristiti sve ranije napravljeno.

Item_search_result.xml

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <ImageView
        android:id="@+id/thumbnail"
        android:layout_width="@dimen/thumbnailWidth"
        android:layout_height="@dimen/thumbnailHeight"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"/>
    <TextView
        android:id="@+id/name"
        android:layout_width="@dimen/match_constraint"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toRightOf="@+id/thumbnail"
        app:layout_constraintRight_toRightOf="parent"
        android:textSize="@dimen/textSizeTitle"/>
    <TextView
        android:id="@+id/summary"
        android:layout_width="@dimen/match_constraint"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/name"
        app:layout_constraintLeft_toRightOf="@+id/thumbnail"
        app:layout_constraintRight_toRightOf="parent"/>
</android.support.constraint.ConstraintLayout>
```

activity_main.xml

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <ImageButton
        android:id="@+id/searchAction"
        android:layout_width="@dimen/searchActionWidth"
        android:layout_height="@dimen/searchActionHeight"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:src="@android:drawable/ic_menu_search"/>
    <EditText
        android:id="@+id/showNameInput"
        android:layout_width="@dimen/match_constraint"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toLeftOf="@+id/searchAction"/>
    <android.support.v7.widget.RecyclerView
        android:id="@+id/showsFound"
        android:layout_width="match_parent"
        android:layout_height="@dimen/match_constraint"
        app:layout_constraintTop_toBottomOf="@+id/searchAction"
        android:layout_marginTop="@dimen/marginSmall"/>
</android.support.constraint.ConstraintLayout>
```

MainActivity.kt

```

class MainActivity : AppCompatActivity(), Callback<List<SearchResult>> {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        searchAction.setOnClickListener{findShows()}
        showsFound.adapter = SearchResultAdapter()
        showsFound.layoutManager = LinearLayoutManager(
            this,
            RecyclerView.VERTICAL,
            false
        )
        showsFound.addItemDecoration(
            DividerItemDecoration(this, RecyclerView.VERTICAL)
        )
        showsFound.itemAnimator = DefaultItemAnimator()
    }

    private fun findShows() {
        val name = showNameInput.text.toString()
        Networking.showSearchService.findShow(name).enqueue(this)
    }

    override fun onFailure(call: Call<List<SearchResult>>, t: Throwable) {
        Toast.makeText(this, R.string.networkError, Toast.LENGTH_SHORT).show()
    }

    override fun onResponse(call: Call<List<SearchResult>>, response: Response<List<SearchResult>>) {
        val results = response.body() ?: listOf()
        Log.d("Results", results.toString())
        (showsFound.adapter as SearchResultAdapter).refreshData(results)
    }
}

```

SearchResultAdapter.kt

```

class SearchResultAdapter : RecyclerView.Adapter<SearchResultHolder>() {

    val results: MutableList<SearchResult> = mutableListOf()

    fun refreshData(newResults: List<SearchResult>) {
        results.clear()
        results.addAll(newResults)
        notifyDataSetChanged()
    }

    override fun getItemCount(): Int = results.size

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): SearchResultHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_search_result, parent, false)
        return SearchResultHolder(view)
    }

    override fun onBindViewHolder(holder: SearchResultHolder, position: Int) {
        holder.bind(results.get(position))
    }
}

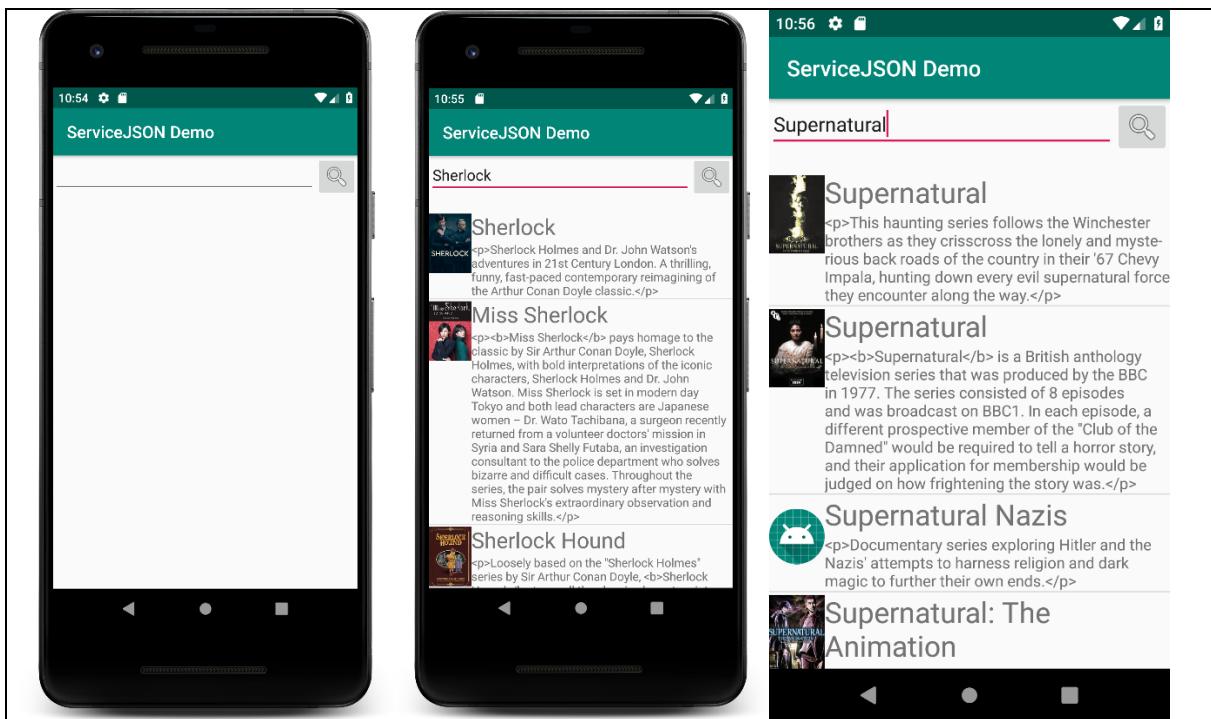
class SearchResultHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

    fun bind(result: SearchResult) {
        itemView.name.text = result.show.name
        itemView.summary.text = result.show.summary

        val url = result.show.thumbnailUrl?.smallThumbnailUrl ?: null

        Picasso.get()
            .load(url)
            .placeholder(R.mipmap.ic_launcher)
            .error(android.R.drawable.stat_notify_error)
            .into(itemView.thumbnail)
    }
}

```



4.2.3. Rad s podacima u XML-u

Android koristi XML prilikom definiranja sučelja, resursa i metapodataka o aplikaciji no isto tako omogućuje parsiranje XML datoteka dostupnih iz različitih izvora. Nekoliko je parsera kojima je ovo moguće postići. Moguće je rabiti *XMLPullParser* i napisati kod koji prolazi kroz oznake i iz XML-a izvlači podatke koji su potrebni. Međutim, različite biblioteke omogućuju kreiranje klase koja predstavlja model podataka, a onda se one pobrinu da se iz XML-a dobiju željeni objekti. Kako je u ranijim vježbama navedeno, XML predstavlja proširivi opisni jezik kojeg je W3C definirao 1998. XML dokument sastoji se od elemenata gdje svaki element ima otvarajuću oznaku, sadržaj i zatvarajuću oznaku. Mora sadržavati točno jedan korijenski element. Dobro oblikovana XML datoteka mora zadovoljavati:

- Mora započinjati prologom
- Svaki otvarajuća oznaka mora imati zatvarajuću
- Sve su oznake ugniježđene

```
<?xml version="1.0"?>
<!--This is a comment -->
<address>
    <name>Pero</name>
    <street>Kneza Trpimira 2b</street>
    <telephonenumber>"0123"/>
</address>
```

Slika 4.3. Pravilno formiran XML dokument



- “*XML is a classic political compromise: it balances the needs of man and machine by being equally unreadable to both.*” – Matthew Might

- “XML combines the efficiency of text files with the readability of binary files” – Unknown



Primjer 4.6. – Korištenje XML konvertera

Ako odgovor sa servera dolazi u obliku XML-a (primjerice RSS feed), sve ostaje slično kao u ranijem primjeru s JSON formatom. Jedina praktična razlika jest u korištenju drugog konvertera, odnosno, rabi se SimpleXml konverter kako je dano u build.gradle datoteci. Kako TvMaze daje podatke samo u JSONu, za ovaj primjer korišten je OMdb api dostupan na www.omdbapi.com/. Ovaj servis koristi api ključ koji se dobije prilikom registracije. Potrebno je i dalje imati Retrofit i RecyclerView, ako se žele prikazati podaci u listi. S obzirom da ova usluga koristi API ključ kako bi ograničila pristup resursima, njega je potrebno poslati uz svaki zahtjev. Kako će ga se poslijediti, ovisi o usluzi pa je dobro konzultirati dokumentaciju usluge. Moguće je rabiti @Header anotaciju ili @Query anotaciju gdje se ključ predaje unutar zahtjeva kao jedan od parametara. Ako se šalje više parametara, onda ih je moguće pohraniti u mapu ključ-vrijednost parova gdje ključ označava parametar zahtjeva, a vrijednost predstavlja vrijednost koju se želi poslati.

Pročitati i <https://developers.google.com/maps/api-key-best-practices> i <https://guides.codepath.com/android/Storing-Secret-Keys-in-Android> za rad s ključevima.

```
<root totalResults="287" response="True">
    <result title="Arrival" year="2016" imdbID="tt2543164" type="movie" poster="https://m.media-amazon.com/images/M/MV5BMTEzMzU0ODcxNDheQTJeQWpwZ15BbWU4MDE1OTI4MzAy._V1_SX300.jpg"/>
    <result title="The Arrival" year="1996" imdbID="tt0115571" type="movie" poster="https://m.media-amazon.com/images/M/MV5BMzhIN2M1NGUtOWVmNi00YWU1LWJ-1NGMtZDMzYjkwZTQ4MmY4XkEyXkFqcGdeQXVyNjc2NDI1ODA@._V1_SX300.jpg"/>
    <result title="The Arrival of a Train" year="1896" imdbID="tt0000012" type="movie" poster="https://m.media-amazon.com/images/M/MV5BZjE2MGVkJMTAtMWIwYy00YzQ5LWE2YTAtMTU2NGJ-mNGNjY2IyXkEyXkFqcGdeQXVyNjMzM3NDI@._V1_SX300.jpg"/>
    <result title="Arrival II" year="1998" imdbID="tt0122961" type="movie" poster="https://m.media-amazon.com/images/M/MV5BMjA2MDI3NjcyN15BMl5BanBnXkFtZTcwOTMxOTUyMQ@@._V1_SX300.jpg"/>
</root>
```

activity_main.xml

```
compile ('com.squareup.retrofit2:converter-simplexml:2.5.0') {
    exclude group: 'xpp3', module: 'xpp3'
    exclude group: 'stax', module: 'stax-api'
    exclude group: 'stax', module: 'stax'
```

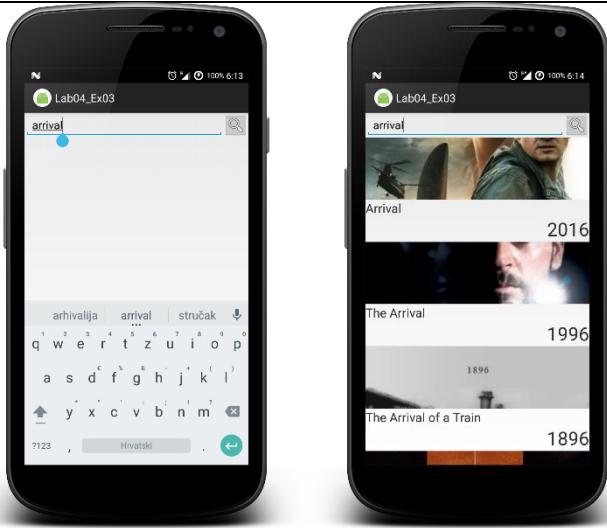
Movie.kt

```
@Root(name = "result", strict = false)
class Movie {
    @Attribute(name = "title") val title: String? = null
    @Attribute(name = "year") val year: Int = 0
    @Attribute(name = "imdbID") val imdbID: String? = null
    @Attribute(name = "type") val type: String? = null
    @Attribute(name = "poster", required = false) val posterUrl: String? = null
}
```

```

@Root(strict = false, name = "root")
class SearchResult {
    @Attribute(name = "totalResults") val total: Int = 0
    @Attribute(name = "response") val response: Boolean = false
    @ElementList(name = "result", inline = true) val movies: List<Movie>? = null
}

```



4.1. Primatelji odašiljanja (engl. *Broadcast Recievers*)

Primatelji odašiljanja odgovaraju na emitiranje poruka od strane drugih aplikacija ili samog sustava. Ove se poruke ponekad nazivaju događaji. Primjerice, *broadcast* se šalje kod završetka podizanja sustava, primitka SMS poruke, paljenja ili gašenja nekog od mrežnih uređaja i slično. Aplikacije se mogu podesiti tako da neka njihova komponenta reagira na takav događaj i izvrši određene radnje. Dva glavna koraka kod korištenja primatelja odašiljanja su njihovo stvaranje i njihova prijava. Prijaviti se mogu u *Manifest* datoteci, pri čemu se radi o statičkoj prijavi. Dinamičku prijavu moguće je obaviti bilo gdje u kodu (primjerice u nekoj od metoda životnog ciklusa *Activitya*), a uobičajeno ih je odjaviti kada prestane potreba za njihovim korištenjem. Važno je voditi računa o tome da *onReceive()* metoda primatelja ne obavlja složen posao i traje kratko.



Primjer 4.7. – Korištenje Broadcast receivera

U ovom primjeru kreirana je klasa koja predstavlja primatelj odašiljanja za vlastito odašiljanje (u smislu zadanog *IntentFilter-a*). Pritiskom gumba stvara se novi *Intent* koji se odašilje. Unutar *onResume()* metode kreiran je i registriran primatelj odašiljanja za ranije kreirani *Intent*. Važno je odjaviti primatelja jednom kada više nije potreban, što je i učinjeno u *onPause()* metodi.

MainActivity.kt

```

class MainActivity : AppCompatActivity() {

    val receiver = CustomReceiver()
    val intentFilter = IntentFilter(CustomReceiver.ACTION_CUSTOM)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setUpUi()
    }

    private fun setUpUi() {
        broadcastSendAction.setOnClickListener{ broadcastCustomIntent() }
    }

    private fun broadcastCustomIntent() {
        val message = messageInput.text.toString()
        val intent = Intent()
        intent.apply{
            putExtra(CustomReceiver.KEY_MESSAGE, message)
            action = CustomReceiver.ACTION_CUSTOM
        }
        sendBroadcast(intent)
    }

    override fun onResume() {
        super.onResume()
        registerReceiver(receiver, intentFilter)
    }

    override fun onPause() {
        super.onPause()
        unregisterReceiver(receiver)
    }
}

```

CustomReceiver.kt

```

class CustomReceiver: BroadcastReceiver() {

    companion object {
        const val KEY_MESSAGE = "message"
        const val ACTION_CUSTOM: String = "hr.ferit.broadcast.CUSTOM_INTENT"
    }

    override fun onReceive(context: Context?, intent: Intent?) {
        context?.let {
            val s = intent?.getStringExtra(KEY_MESSAGE) ?: it.getString(R.string.noMessage)
            Toast.makeText(it, s, Toast.LENGTH_SHORT).show()
        }
    }
}

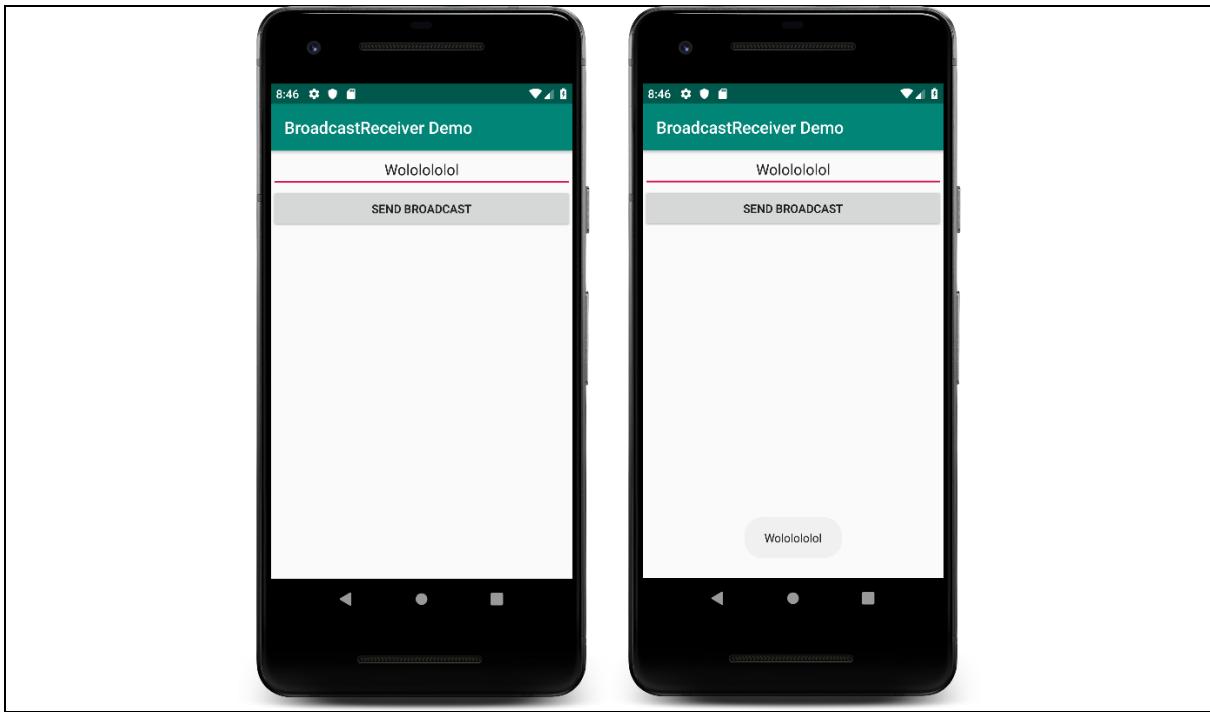
```

activity_main.xml

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <EditText
        android:id="@+id/messageInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:hint="@string/messageInputHint"/>
    <Button
        android:id="@+id/broadcastSendAction"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/broadcastSendText" />
</LinearLayout>

```

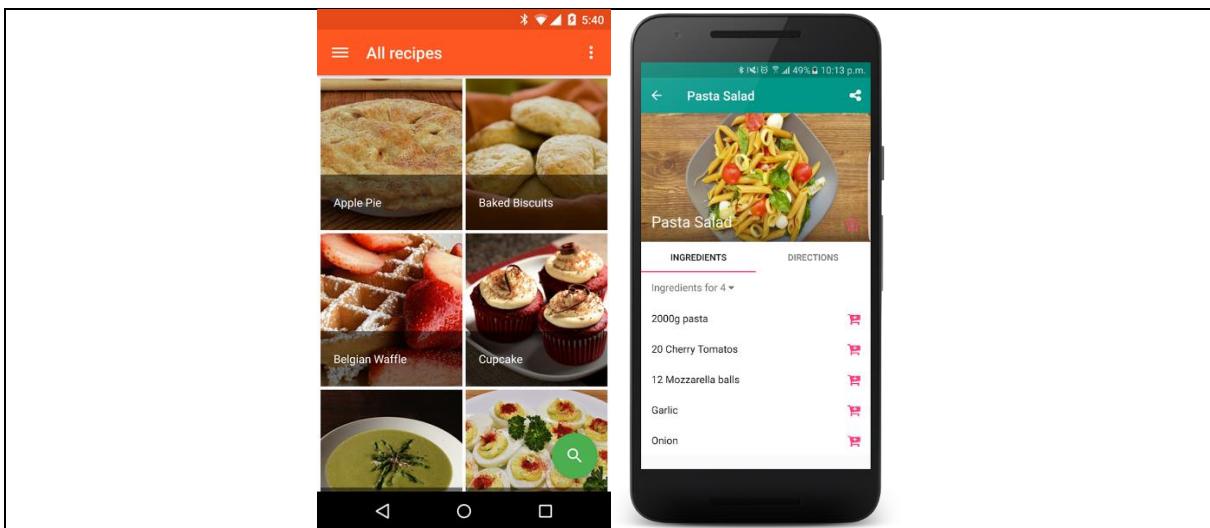


Za razliku od prethodnog primjera, gdje je *Broadcast receiver* registriran dinamički, moguće ih je registrirati i statički. Ovo se postiže navođenjem *receiver* komponente unutar manifest datoteke, slično kao što je bio slučaj kod *Activitya*. Ovim načinom vrlo se jednostavno može prijaviti na sustavske obavijesti poput završetka podizanja sustava, promjene konekcije prema Internetu i sl. S obzirom da će ovaj primatelj odašiljanja biti prijavljen u *Manifest* datoteci, nije ga nužno odjaviti. Ako se ipak za tim pojavi potreba, moguće je koristiti *PackageManager* i odjaviti statičke *receivere*. Bez obzira na to je li *Activity* pokrenut ili ne, *BroadcastReciever* će biti okinut na dolazak sustavske poruke te će biti izvršena njegova *onRecieve()* metoda. Više i detaljnije o *Broadcast Receiverima* moguće je pronaći na <http://codetheory.in/android-broadcast-receivers/>

4.1. Primjeri

 Primjer 4.
Foodster

Kreirajte aplikaciju koja se oslanja na <http://www.recipepuppy.com/about/api/> i omogućuje korisniku da na temelju unesenih sastojaka dobije informacije o jelima.



4.2. Zadaća



Zadatak 4.

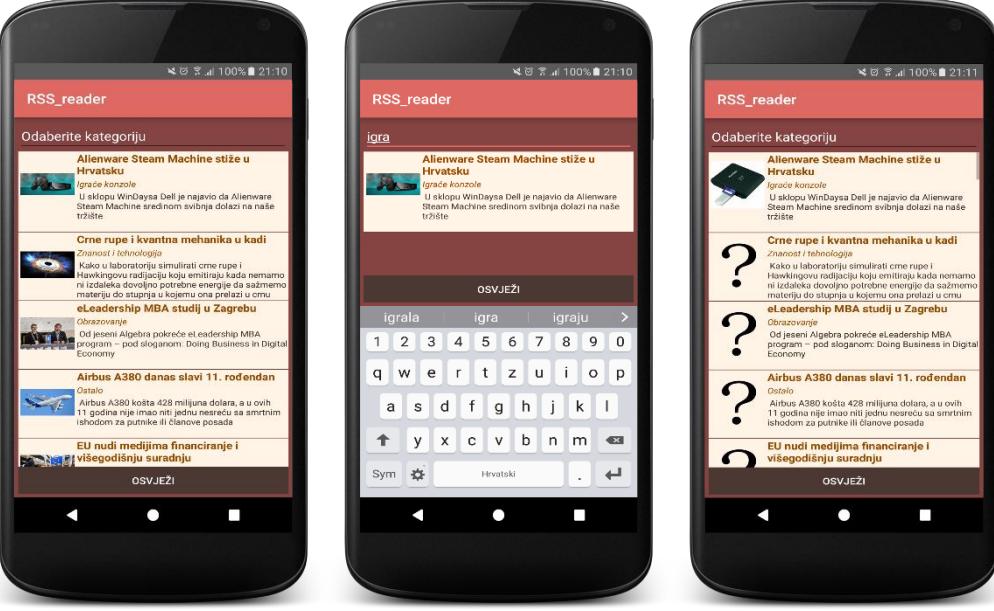
Bugsy – Bug RSS čitač

Kreirajte aplikaciju koja omogućuje prikaz vijesti iz RSS feed-a na <http://www.bug.hr/rss/vijesti/> te omogućuje prikaz po kategorijama. Potrebno je prikazati sliku, vijest, vrijeme objave i omogućiti čitanje vijesti klikom na element liste.

- ➊ Kreirati Activity (ili fragment) koji će sadržavati RecyclerView
- ➋ Implementirati vlastiti adapter
- ➌ Kreirati vlastiti layout za element liste
- ➍ Omogućiti osvježavanje sadržaja (pull to refresh/gumb)
- ➎ Omogućiti prikaz po kategorijama (ili kao više fragmenata, ili kroz Spinner)

Dodatno:

- ➏ Spremanje odabranih vijesti u bazu
- ➐ Vlastiti WebView koji otvara vijest unutar aplikacije



Napomena:

Svaka je nadogradnja poželjna i dobrodošla. Korištenje znanja koja nadilaze opseg dosadašnjih vježbi se potiče.



“ You can use an eraser on the drafting table or a sledgehammer on the construction site ”

- Frank Lloyd Wright



LV5: Geolokacijske usluge, senzori

5. Laboratorijska vježba 5

5.1. Uvod

Ova vježba najprije prikazuje rad sa senzorima dostupnim na uređaju. Nakon toga opisuju se lokacijske usluge (usluge temeljene na trenutnoj lokaciji uređaja). Iako se ove usluge u prvom redu povezuju s Googleovim kartama, one omogućuju i brojne druge stvari unutar aplikacije. Kako GPS koordinate ne predstavljaju oblik informacije koji je prilagođen korisniku, opisano je i korištenje Googleovih karata za bolji uvid u lokaciju. Dodatno, opisana je i jednostavna uporaba multimedijalnih datoteka za koje Android sustav pruža kvalitetnu podršku. Na kraju su pokazane notifikacije, koje predstavljaju nemetljiv način obavještavanja korisnika o različitim događajima. Jednom kada je aplikacija završena potrebno ju je distribuirati korisnicima što je moguće učiniti preko brojnih trgovina aplikacijama. Primjer korištenja Googleove trgovine PlayStore dan je u zadnjem poglavljtu.

5.1.1. Potrebna predznanja

- Osnove programiranja
 - Kolegiji Programiranje I, Programiranje II, Algoritmi i strukture podataka
- Osnove objektno orijentiranog programiranja
 - Kolegij Objektno orijentirano programiranje
- Osnove izrade Android aplikacija (*Activity*, osnove UI-ja)
 - Predložak LV1-4, predavanja
- Osnove Java programskog jezika
 - Java, A Beginner's Guide, 5th Edition - Herbert Schildt
 - Thinking in Java, 4th edition – Bruce Eckel
 - <http://docs.oracle.com/javase/tutorial/>
- Intent - <http://developer.android.com/reference/android/content/Intent.html>

5.1.2. Korisna predznanja

- Lokacijske usluge - <https://developer.android.com/training/location/index.html>
- Notifikacije - <https://developer.android.com/guide/topics/ui/notifiers/notifications.html>
- Multimedija - <https://developer.android.com/training/building-multimedia.html>

5.2. Senzori

Android uređaji sadrže brojne senzore koji se mogu rabiti u aplikacijama, poput akcelerometra, žiroskopa, barometra i slično. Općenito postoji tri kategorije senzora: senzori pokreta, senzori okoline i senzori smještaja. Za pristup senzorima i podacima koje prikupljaju koristi se senzorski okvir unutar Android sustava, odnosno klasa

SensorManager i klasa *Sensor*. Uz navedene, koriste se još i *SensorEvent* koji predstavlja neki događaj (primjerice promjena vrijednosti koju senzor očitava) te *SensorEventListener* koji omogućuje praćenje stanja senzora. Iako na emulatoru nisu podržani, moguće ih je koristiti uz *SensorSimulator* koji se da vezati uz emulator. Međutim, u svakom je slučaju aplikacije koje se oslanjaju na senzore lakše testirati korištenjem stvarnog, fizičkog uređaja. Neki od senzora koji su dostupni preko klase *SensorManager* uistinu su fizički senzori, dok su drugi virtualni senzori koji se oslanjaju na mjerjenja nekoliko fizičkih senzora (nazivaju se još i sintetičkim senzorima). Kako nisu svi senzori prisutni na svakom Android uređaju, moguće je preko *SensorManagera* dobiti listu svih senzora određenog tipa, a isto je tako moguće dobiti i pristup podrazumijevanom senzoru za neki tip. Tipovi su određeni konstantama unutar *Sensor* klase. Moguće je da uređaj nema sve senzore koje očekujete, a isto je tako moguće da za neki tip uređaj ima više različitih senzora.

</>Primjer 5.1. – Pristup senzorima

Dohvaćanje informacija o dostupnim senzorima. Kako bi se pristupilo senzorima, rabi se klasa *SensorManager*. Instanci ove klase pristupa se korištenjem konteksta i to pozivom *getSystemService* metodi s parametrom *Context.SENSOR_SERVICE*. Vraćeni je objekt potrebno pretvoriti u *SensorManager* korištenjem eksplicitnog pretvaranja tipa.

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    private val TAG = "Sensors"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        listSensorsInLog()
    }

    private fun listSensorsInLog() {
        val sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        val sensors = sensorManager.getSensorList(Sensor.TYPE_ALL)
        sensors.forEach{
            Log.d(TAG, "Sensor: ${it.name} type: ${it.type}")
        }
    }
}
```

Logcat

The screenshot shows the Android Studio Logcat window. The title bar says "Emulator Telefon Android 9, API". The dropdown shows "hr.ferit.brunozoric.sensorsdemo (1)". The filter bar has "Verbose" selected and contains the search term "Sensor". The log output lists various sensors: Accelerometer, Gyroscope, Magnetic field, Orientation, Ambient Temperature, Proximity, Light, Pressure, Humidity, 3-axis Magnetic field (uncalibrated), Game Rotation Vector, GeoMag Rotation Vector, and Gravity Sensors. Each entry includes the timestamp (2019-04-23 16:50:50.095), the package name (hr.ferit.brunozoric.sensorsdemo), the sensor type (e.g., D/Sensors: Sensor: Goldfish 3-axis Accelerometer type: 1), and the sensor name (e.g., Sensor: Goldfish 3-axis Accelerometer).

5.3. Korištenje akcelerometra

Akcelerometar je senzor koji omogućuje mjerjenje akceleracije po svakoj osi uređaja, a mjerjenje uključuje iznos gravitacije. Zbog potonje činjenice akceleracija uređaja koji miruje na površini stola iznosit će 9.81 m/s^2 . Riječ je o senzoru koji je vrlo koristan u situacijama u kojima je nužno mjeriti kretanje uređaja, postoji na većini Android uređaja te ima relativno niske zahtjeve na potrošnju.

Primjer 5.2. – Korištenje akcelerometra

U ovom primjeru bit će prikazano korištenje akcelerometra s ciljem određivanja akceleracije uređaja. Ovaj senzor dohvaća tri vrijednosti, akceleraciju po svakoj od osi uređaja. U mjerjenje je također uključena i gravitacija, tako da je potrebno biti pripaziti s konačnom vrijednosti ukoliko se ovaj senzor koristi za ozbiljnije zadatke. Ostali se senzori koriste na sličan način, a važno je obratiti pozornost na prijavu i odjavu osluškivanja na promjenu vrijednosti senzora.

activity_main.xml

```
<android.support.constraint.ConstraintLayout
    android:id="@+id/rootView"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/accelerationDisplay"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:gravity="center"
        android:textSize="@dimen/TextSize"
        android:text="@string/accelerationText" />
    <TextView
        android:id="@+id/accelerationComponents"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/accelerationDisplay"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:gravity="center"
        android:text="@string/accelerationComponentText" />
</android.support.constraint.ConstraintLayout>
```

The screenshot shows the 'Sensors Demo' application running on a virtual device. The main screen displays the value '9.81' in large font, with smaller values '0.00', '9.81', and '0.00' below it. To the right, the 'Extended controls' window is open, showing a preview of the phone and various control options. Under 'Virtual sensors', the 'Move' option is selected. Below the preview, three sliders show values for X (0.3), Y (0.4), and Z (0.0). On the right side of the window, 'Resulting values' are listed: Accelerometer (m/s²): 0.00 9.81 0.00, Gyroscope (rad/s): 0.00 0.00 0.00, Magnetometer (µT): 0.00 5.90 -48.40, and Rotation: ROTATION_0.

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    private val TAG = "Sensors"

    private val decimalFormat = DecimalFormat("0.00")
    private lateinit var sensorManager: SensorManager;
    private lateinit var accelerationSensor: Sensor

    private val sensorListener = object: SensorEventListener{
        override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
        }
        override fun onSensorChanged(event: SensorEvent?) {
            val values = event?.values ?: floatArrayOf(0.0f, 0.0f, 0.0f)
            updateUi(values)
        }
    }

    override fun onCreate(savedInstanceState: Bundle?) {

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        accelerationSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    }

    override fun onResume() {
        super.onResume()
        sensorManager.registerListener(sensorListener, accelerationSensor, SensorManager.SENSOR_DELAY_UI)
    }

    override fun onPause() {
        super.onPause()
        sensorManager.unregisterListener(sensorListener)
    }

    private fun updateUi(values: FloatArray) {

        val totalAcceleration = Math.sqrt(values.map { x -> x*x }.sum().toDouble())
        accelerationDisplay.text = decimalFormat.format(totalAcceleration)
        if(totalAcceleration > 12)
            rootView.setBackgroundResource(R.color.colorHigh)
        else
            rootView.setBackgroundResource(R.color.colorLow)

        accelerationComponents.text = ""
        values.forEach { accelerationComponents.append("${decimalFormat.format(it)}\n") }
    }
}
```

5.1. Notifikacije

Aplikacija može koristiti obavijesti (notifikacije) kako bi javila korisniku da se dogodio određeni događaj koji zahtijeva njegovu pažnju. Radi se o nemetljivom načinu obavještavanja koji se preferira u Android okruženju, s obzirom da korisnik na njih može reagirati u trenutku koji mu odgovara. Notifikacije korisnika mogu obavijestiti korištenjem ikone u statusnoj traci, bljeskanjem LED svjetla, vibracijom, puštanjem zvučnih signala, prikazom informacija u notifikacijskoj ladici itd. Notifikacije se kreiraju i njima se upravlja korištenjem *NotificationManager* objekta. Od Android 3.0 verzije notifikacije su podložne značajnim prilagodbama izgleda, moguće je čak koristiti *layout* za prikaz sadržaja unutar notifikacije, a od verzije 7.0 doživjele su daljnje izmjene i prilagodbe što je detaljno prikazano u službenoj dokumentaciji.



Korištenje notifikacija



Primjer 5.3. – Korištenje notifikacija

Na novijim inačicama Androida, notifikacije su doživjele najveće izmjene. Za korištenje je potrebno definirati kanale notifikacija i svaku notifikaciju se objavljuje u pojedinom kanalu.

activity_main.xml

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/likeAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/likeActionText"/>
    <Button
        android:id="@+id/commentAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/likeAction"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        android:text="@string/commentActionText"/>
</android.support.constraint.ConstraintLayout>
```

MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) createNotificationChannels()
        likeAction.setOnClickListener{displayLikeNotification() }
        commentAction.setOnClickListener{displayCommentNotification() }
    }

    private fun displayLikeNotification() {

        val intent = Intent(this, MainActivity::class.java)
        // We can put extra in the intent.

        val pendingIntent = PendingIntent.getActivity(
            this,
            0,
            intent,
            PendingIntent.FLAG_CANCEL_CURRENT
        )

        val notification = NotificationCompat.Builder(this, getChannelId(CHANNEL_LIKES))
            .setSmallIcon(R.mipmap.ic_launcher)
            .setContentTitle("Like!")
            .setContentText("Liked your post.")
            .setAutoCancel(true)
            .setContentIntent(pendingIntent)
            .build()

        NotificationManagerCompat.from(this)
            .notify(1001, notification)
    }

    private fun displayCommentNotification() {
        // Homework!
    }
}
```

NotificationHelper.kt

```
fun getChannelId(name: String): String = "${MyApp.instance.packageName}-$name"
const val CHANNEL_LIKES = "Likes_Channel"
const val COMMENTS_CHANNEL = "Comments_Channel"

@RequiresApi(api = Build.VERSION_CODES.O)
fun createNotificationChannel(name: String, description: String, importance: Int): NotificationChannel {
    val channel = NotificationChannel(getChannelId(name), name, importance)
    channel.description = description
    channel.setShowBadge(true)
    return channel
}

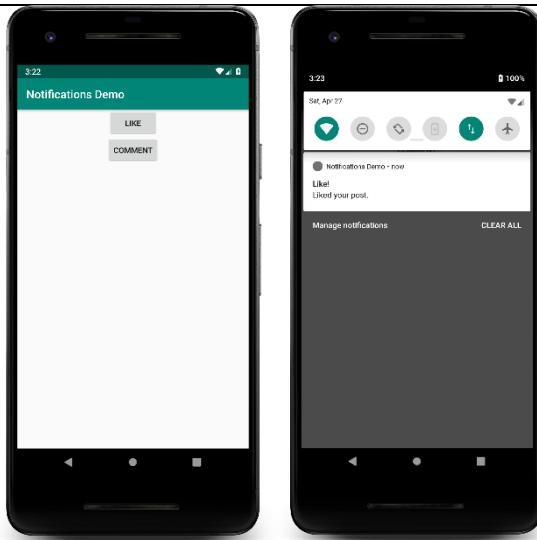
@RequiresApi(api = Build.VERSION_CODES.O)
fun createNotificationChannels() {

    val channels = mutableListOf<NotificationChannel>()

    channels.add(createNotificationChannel(
        CHANNEL_LIKES,
        "Likes on your posts",
        NotificationManagerCompat.IMPORTANCE_DEFAULT
    ))

    channels.add(createNotificationChannel(
        COMMENTS_CHANNEL,
        "Comments on your posts",
        NotificationManagerCompat.IMPORTANCE_HIGH
    ))

    val notificationManager = MyApp.instance.getSystemService(NotificationManager::class.java)
    notificationManager.createNotificationChannels(channels)
}
```



5.1. Soundpool

Za rad sa zvukom unutar Android aplikacija dostupno je nekoliko mogućnosti. Jedna od njih je *MediaPlayer* klasa koja omogućuje upravljanje izvođenjem zvučnih i video datoteka. Rad s ovom klasom relativno je složen zbog brojnih metoda životnog ciklusa na koje je potrebno обратити pažnju. Ovu se klasu u pravilu koristi prilikom izrade alata za reprodukciju glazbe, audio knjiga i sličnih, dugotrajnih zvučnih zapisa.



Upute za korištenje *MediaPlayer* klase

<https://developer.android.com/guide/topics/media/mediaplayer.html>

Alternativa koja se rabi za izvođenje relativno kratkih zvučnih zapisa zove se *SoundPool*. Ona omogućuje učitavanja i paralelno izvođenje nekoliko zvučnih zapisa što je česta potreba bilo u aplikacijama bilo u igrama za Android platformu.

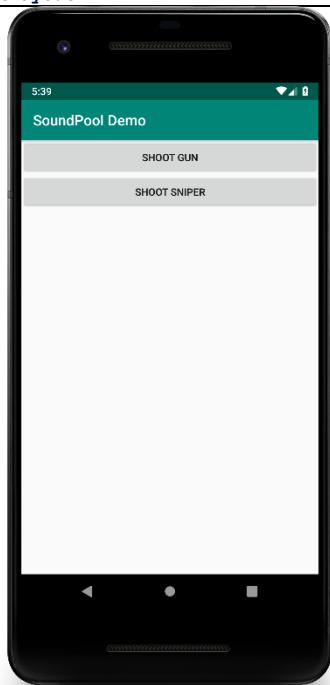


Primjer 5.4. – Korištenje notifikacija

Zvukovi su preuzeti sa soundbible.com i smješteni u *raw* mapu unutar *res* mape s resursima. Potrebno je zvučne zapise najprije učitati u *SoundPool* objekt, a prilikom učitavanja svakom od njih dodijeljen je ID. Preko ovog ID-a odabire se zvučni zapis koji se izvodi. U primjeru su zvučni zapisi pohranjeni u *HashMap* objekt kao parovi identifikatora pojedinog zapisa u *raw* mapi i identifikatora unutar *SoundPool*a. Preko ove veze se odabire zvuk na pritisak gumba. Zvukove je nužno učitati asinkrono, a završetak učitavanja signaliziran je metodom *onLoadComplete* unutar *OnLoadCompleteListener*era koji se postavlja na *SoundPool* objekt.

activity_main.xml

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/shootGun"
        android:layout_width="0dp"
        app:layout_constraintTop_toTopOf="parent"
        android:layout_height="wrap_content"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:text="@string/shootGunText"/>
    <Button
        android:id="@+id/shootSniper"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/shootGun"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:text="@string/shootSniperText"/>
</android.support.constraint.ConstraintLayout>
```



MainActivity.kt

```
class MainActivity : AppCompatActivity(), View.OnClickListener {

    private lateinit var mSoundPool: SoundPool
    private var mLoaded: Boolean = false
    var mSoundMap: HashMap<Int, Int> = HashMap()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        this.setUpUi()
        this.loadSounds()
    }

    private fun setUpUi() {
        this.shootGun.setOnClickListener(this)
        this.shootSniper.setOnClickListener(this)
    }

    private fun loadSounds() {

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            this.mSoundPool = SoundPool.Builder().setMaxStreams(10).build()
        } else {
            this.mSoundPool = SoundPool(10, AudioManager.STREAM_MUSIC, 0)
        }

        this.mSoundPool.setOnLoadCompleteListener { _, _, _ -> mLoaded = true }
        this.mSoundMap[R.raw.gun] = this.mSoundPool.load(this, R.raw.gun, 1)
        this.mSoundMap[R.raw.sniper] = this.mSoundPool.load(this, R.raw.sniper, 1)
    }

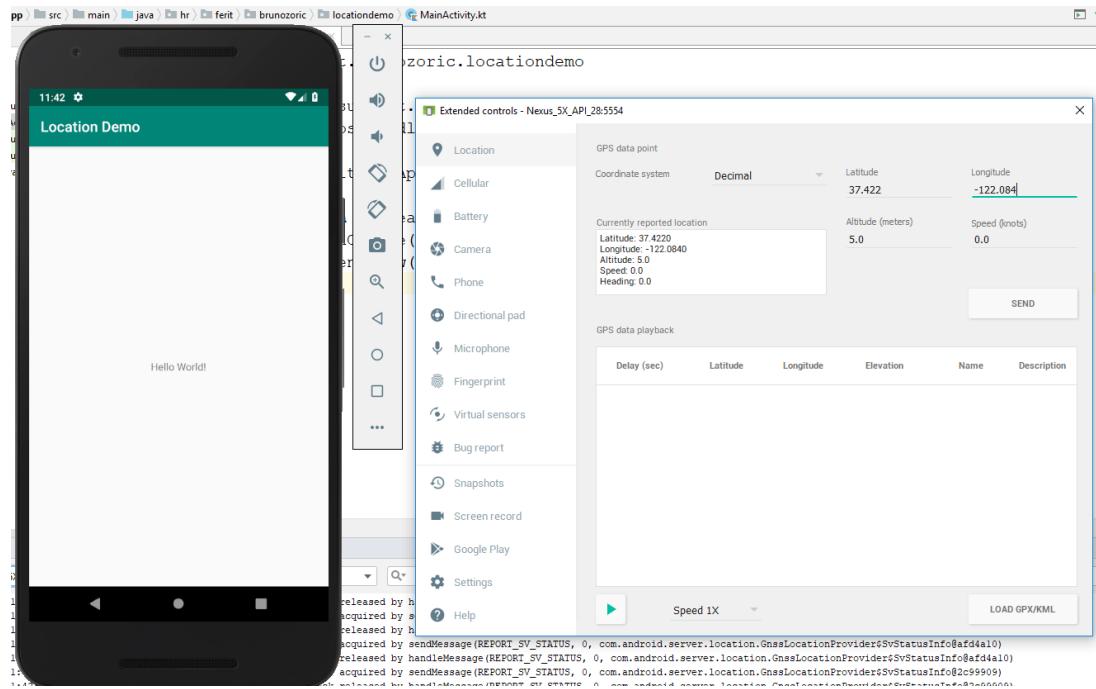
    override fun onClick(v: View) {
        if (this.mLoaded == false) return
        when (v.getId()) {
            R.id.shootGun -> playSound(R.raw.gun)
            R.id.shootSniper -> playSound(R.raw.sniper)
        }
    }

    fun playSound(selectedSound: Int) {
        val soundID = this.mSoundMap[selectedSound] ?: 0
        this.mSoundPool.play(soundID, 1f, 1f, 1, 0, 1f)
    }
}
```

5.2. Geolokacijske usluge

Usluge temeljene na lokaciji podrazumijevaju razne tehnologije koje je moguće koristiti za pronaalaženje trenutne lokacije uređaja. Glavna dva elementa koja se koriste su upravitelj lokacija (engl. *Location Manager*) i pružatelji lokacija (engl. *Location Providers*). Kako bi se mogle koristiti ove usluge potrebno je dodati oznake za dozvole, u ovisnosti koja se razina preciznosti traži. Uključivanje preciznije (FINE) lokacije uključuje automatski i nepreciznije (COARSE). Prve podrazumijevaju uporabu GPS koordinata, dok se druge oslanjaju na lokaciju dobivenu na temelju mreže i sl. Kako bi se lokacije testirale na virtualnim uređajima, Android omogućuje tzv. „*mock*“ lokacije. Za virtualne uređaje ovo znači da je moguće kroz kontrole emulatora ostvariti postavljanje lokacije uređaja iz „*Location*“ izbornika, što je i prikazano slikom 5.1. Moguće je osim direktnog umetanja koordinata koristiti i GPX ili KML datoteke s više različitih lokacija. Ako je riječ o fizičkom uređaju, tada je u opcijama razvoja potrebno dopustiti korištenje lažnih lokacija. Za

potrebe generiranja lokacija moguće je preuzeti aplikacije s *Googleovog Play Storea* koje isto omogućuju ili koristiti vlastiti pružatelj lažnih lokacija.



Slika 5.1. DDMS alat

5.2.1. GPS koordinate

Kako bi se dohvatile GPS koordinate uređaja, potrebno je zatražiti dopuštenje za pristup lokaciji unutar manifest datoteke. Nakon toga, moguće se osloniti na *LocationManager* i zatražiti lokaciju. Jedan od načina na koji je moguće ovo učiniti jest *getLastKnownLocation()* metoda. Problem je što je lokacija *null* sve dok neka aplikacija ne zatraži praćenje lokacije. Kako bi se ovome doskočilo, moguće je registrirati primatelj odašiljanja s akcijom "SINGLE_LOCATION_UPDATE_ACTION", koji će reagirati na jedno dohvaćanje lokacije i zatim sam sebe odjaviti. Ako se želi pratiti lokacija, potrebno je registrirati *LocationListener* i raditi s njim na sličan način kao u ranijem primjeru sa *SensorEventListenerom*. Nužno je voditi računa da je praćenje lokacije jako zahtjevan posao, koji ima značajan utjecaj na performanse uređaja te je osluškivanje na promjenu lokacije potrebno prekinuti u trenutku kada više nije nužno.

Strategije za baratanje lokacijom uređaja

<https://developer.android.com/guide/topics/location/strategies.html>

<https://android-developers.googleblog.com/2011/06/deep-dive-into-location.html>

<https://www.youtube.com/user/androiddevelopers/videos>



Primjer 5.5. – Korištenje lokacije

Kako bi se počelo i zaustavilo osluškivanje na promjenu lokacije postavlja se i uklanja *LocationListener*. Za vrijeme rada aplikacije (dok je aktivan *MainActivity*) unutar *TextViewa* se prikazuje lokacija u obliku GPS koordinata. Praćenje se prekida u *onPause* metodi *Activitya*. U ovom se primjeru obavljaju dodatne radnje koje se tiču *Permission* okvira. U novijim inačicama Androida korisnik dopuštenja koja su proglašena „opasnima“ odobrava tek kada *developer* od njega to zatraži. Potrebno je stoga zatražiti odobrenja u odgovarajućem trenutku, ali i poštovati mogućnost korisnika da odbije dati dopuštenje.

activity_main.xml

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/trackLocationAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_margin="@dimen/marginMedium"
        android:text="@string/trackLocationActionText"/>
    <TextView
        android:id="@+id/locationDisplay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/locationDisplayTextDefault"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
</android.support.constraint.ConstraintLayout>
```

Manifest

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

PermissionUtils.kt

```
fun AppCompatActivity.hasPermissionCompat(permission: String): Boolean{
    return ActivityCompat.checkSelfPermission(this, permission) == PackageManager.PERMISSION_GRANTED
}

fun AppCompatActivity.shouldShowRationaleCompat(permission: String): Boolean{
    return ActivityCompat.shouldShowRequestPermissionRationale(this, permission)
}

fun AppCompatActivity.requestPermissionCompat(permission: Array<String>, requestCode: Int){
    ActivityCompat.requestPermissions(this, permission, requestCode)
}
```

MainActivity.kt

```

class MainActivity : AppCompatActivity() {

    private val locationPermission = Manifest.permission.ACCESS_FINE_LOCATION
    private val locationRequestCode = 10
    private lateinit var locationManager: LocationManager

    private val locationListener = object: LocationListener{
        override fun onProviderEnabled(provider: String?) { }

        override fun onProviderDisabled(provider: String?) { }

        override fun onStatusChanged(provider: String?, status: Int, extras: Bundle?) { }

        override fun onLocationChanged(location: Location?) {
            updateLocationDisplay(location)
        }
    }

    private fun updateLocationDisplay(location: Location?) {
        val lat = location?.latitude ?: 0
        val lon = location?.longitude ?: 0

        locationDisplay.text = "Lat: $lat\nLon: $lon"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        trackLocationAction.setOnClickListener{ trackLocation() }
        locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager
    }

    private fun trackLocation() {
        if(hasPermissionCompat(locationPermission)){
            startTrackingLocation()
        } else {
            requestPermisionCompat(arrayOf(locationPermission), locationRequestCode)
        }
    }

    override fun onRequestPermissionsResult(
        requestCode: Int,
        permissions: Array<out String>,
        grantResults: IntArray) {

        when(requestCode){
            locationRequestCode -> {
                if(grantResults.size == 1 && grantResults[0] == PackageManager.PERMISSION_GRANTED)
                    trackLocation()
                else
                    Toast.makeText(this, R.string.permissionNotGranted, Toast.LENGTH_SHORT).show()
            }
            else -> super.onRequestPermissionsResult(requestCode, permissions, grantResults)
        }
    }

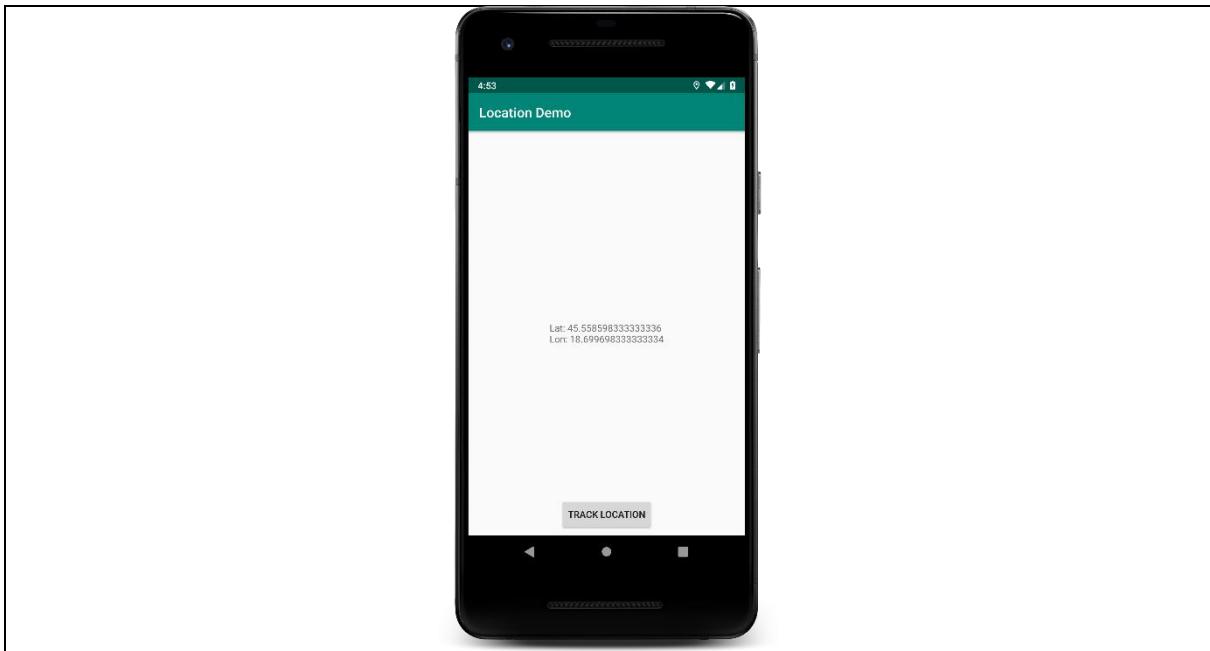
    private fun startTrackingLocation() {
        Log.d("TAG", "Tracking location")

        val criteria: Criteria = Criteria()
        criteria.accuracy = Criteria.ACCURACY_FINE

        val provider = locationManager.getBestProvider(criteria, true)
        val minTime = 1000L
        val minDistance = 10.0F
        try{
            locationManager.requestLocationUpdates(provider, minTime, minDistance, locationListener)
        } catch (e: SecurityException){
            Toast.makeText(this, R.string.permissionNotGranted, Toast.LENGTH_SHORT).show()
        }
    }

    override fun onPause() {
        super.onPause()
        locationManager.removeUpdates(locationListener)
    }
}

```



5.2.1. Geokoder

Geokodiranje predstavlja postupak pretvorbe iz adrese u koordinate i obrnuto. Ovim pristupom moguće je iz koordinata dobiti ljudima razumljiviju reprezentaciju lokacije. Pretvorba se obavlja na udaljenom poslužitelju, pa je potrebna dozvola i za pristup internetu, što je pokazano u ranijim LV. I kodiranje i dekodiranje kao rezultat vraćaju listu rezultata, koja će biti prazna ukoliko nije pronađena adresa ili koordinate, ovisno o smjeru pretvaranja. Kod dekodiranja, riječ je o listi objekata *Address* klase, a svaki objekt sadrži sve detalje do kojih je Geokoder uspio doći poput države, lokaliteta, fizičke adrese i drugih.



Primjer 5.6. – Korištenje geokodiranja

Kod korištenja Geocodera potrebno je provjeriti postoji li on na sustavu. Ako postoji, tada ga je moguće instancirati, predati mu lokaciju i zatražiti dohvaćanje adresa. Iako to u ovom primjeru nije učinjeno, zahtjev za Adresama i čekanje na odgovor potrebno je obaviti na drugoj niti, a za tu je svrhu moguće koristiti *IntentService*. Za korištenje *Geocoder* klase potrebno je na uređaju imati Google Play Services, s obzirom da je *Geocoder* dio Google API add-onsa koji nije dio AOSP-a. Moguće ih je instalirati na emulator ili na stvarni uređaj (koristeći drugi *recovery*), a dostupne su na opengapps.org U ovom primjeru rabi se cijelokupan kod iz prethodnog primjera, a izmijenjeni su samo manifest datoteka te metoda *updateLocationText()* unutar klase *MainActivity*.

activity_main.xml

```

<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/trackLocationAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_margin="@dimen/marginMedium"
        android:text="@string/trackLocationActionText"/>
    <TextView
        android:id="@+id/locationDisplay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/locationDisplayTextDefault"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
</android.support.constraint.ConstraintLayout>

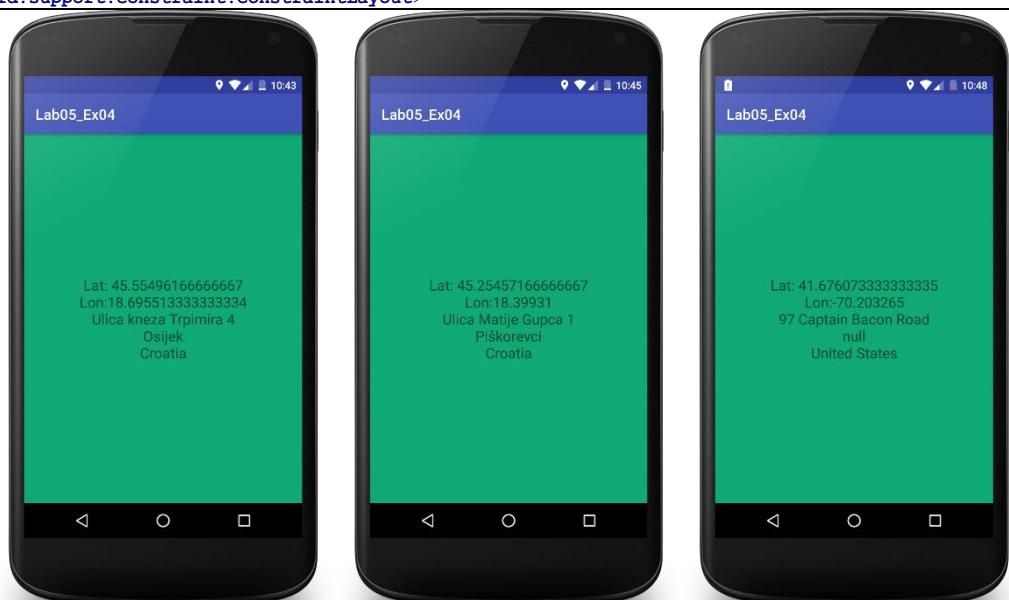
```

MainActivity.kt

```

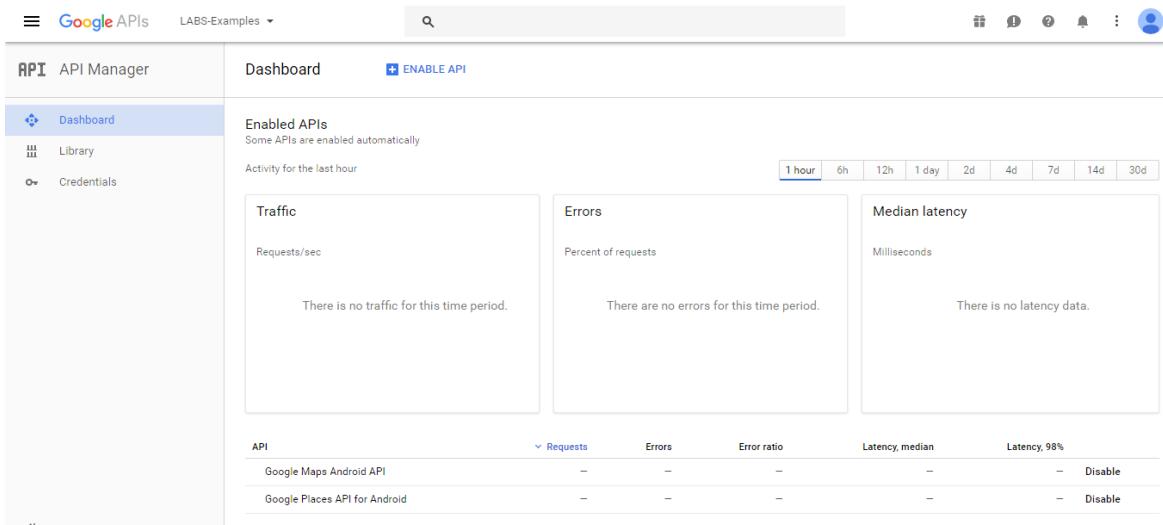
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/trackLocationAction"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_margin="@dimen/marginMedium"
        android:text="@string/trackLocationActionText"/>
    <TextView
        android:id="@+id/locationDisplay"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/locationDisplayTextDefault"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>
</android.support.constraint.ConstraintLayout>

```

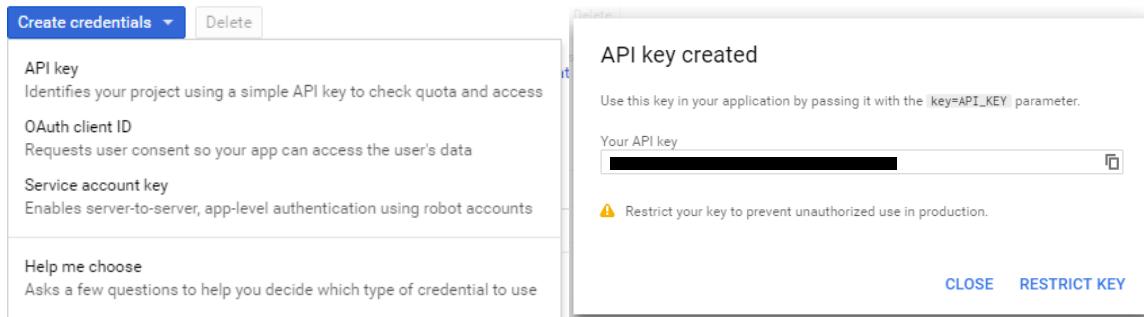


5.2.1. Karte (Google maps)

Kako koordinate baš i nisu podaci prikladni za izravan prikaz korisniku, pribjegava se njihovu primjerenijem prikazu. Jednu od mogućnosti sasvim sigurno predstavlja i Googleov servis Maps. Kako bi ga se moglo koristiti u aplikacijama, potrebno je registrirati se za korištenje Google API-ja na <https://console.developers.google.com/?hl=hr>. Ondje se Kako



Slika 5.4. Kontrolna ploča Google API-ja



Slika 5.5. Stvaranje novog API ključa za Google Mape

koordinate baš i nisu podaci prikladni za izravan prikaz korisniku, pribjegava se njihovu primjerenijem prikazu. Jednu od mogućnosti sasvim sigurno predstavlja i Googleov servis Maps. Kako bi ga se moglo koristiti u aplikacijama, potrebno je registrirati se za korištenje Google API-ja na <https://console.developers.google.com/?hl=hr>. Ondje se kreira novi projekt, omogućuje se uporaba Maps API-ja i generira se ključ. Uporabom ovog ključa, odnosno njegovim dodavanjem u manifest datoteku omogućit će se pristup kartama (bez ključa će biti prikazane samo sive pločice). Ključ je moguće generirati kroz izbornik *Credentials* s lijeve strane upravljačke ploče. Ključ je dobro ograničiti, primjerice prema imenu paketa ili prema kakvom drugom kriteriju.



Primjer 5.7. – Korištenje Google maps servisa

Kako je opisano ranije u tekstu, prvi korak u korištenju Google mapa jest generiranje API ključa. Kada je on spreman, izgrađuje se jednostavna aplikacija s jednim Activityem. Potrebno je dodati vanjsku biblioteku u *gradle build* skriptu, a riječ je *play-services* biblioteci. Također je potrebno dodati određene informacije u manifest datoteku. Riječ je upravo o ranije generiranom API ključu, ali i verziji korištene *play services* biblioteke. Moguće je ovdje dodati još informacija, tražiti dozvolu za pisanje po vanjskoj pohrani, no kako je ovo minimalni radni primjer, zadržat će se na osnovama. U *layout* datoteku dodaje se fragment koji će prikazivati kartu, a riječ je o posebnom fragmentu definiranom u gms.maps imeniku. U glavnom *Activityu* dohvata se taj fragment i preko njega sam *GoogleMap* objekt na koji se postavlja osluškivanje na klik. Važno je napomenuti kako se dohvatanje karte obavlja asinkrono i koristi se *callback* metoda *onMapReady* (izložena kroz sučelje koje *Activity* implementira). Klik na kartu stvara novi marker sa slikom jednakom ikoni aplikacije i porukom koja se prikazuje prilikom klika na marker.

Iako je sve navedeno moguće napraviti samostalno, lakše je isto postići dodavanjem predloška *Activitya* kroz izbornik Android studija. Ako se odabere taj predložak, onda će se pripremiti sve za rad s Google Kartama. Generirat će se čak i xml datoteka u kojoj će se navesti API ključ.

build.gradle (module:app)

```
implementation 'com.google.android.gms:play-services-maps:16.1.0'
```

Manifest

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.INTERNET"/>

<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="@string/google_maps_key"/>
```

activity_main.xml

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/map"
    tools:context=".MapsActivity"
    android:name="com.google.android.gms.maps.SupportMapFragment"/>
```

MainActivity.kt

```
import com.google.android.gms.maps.model.MarkerOptions

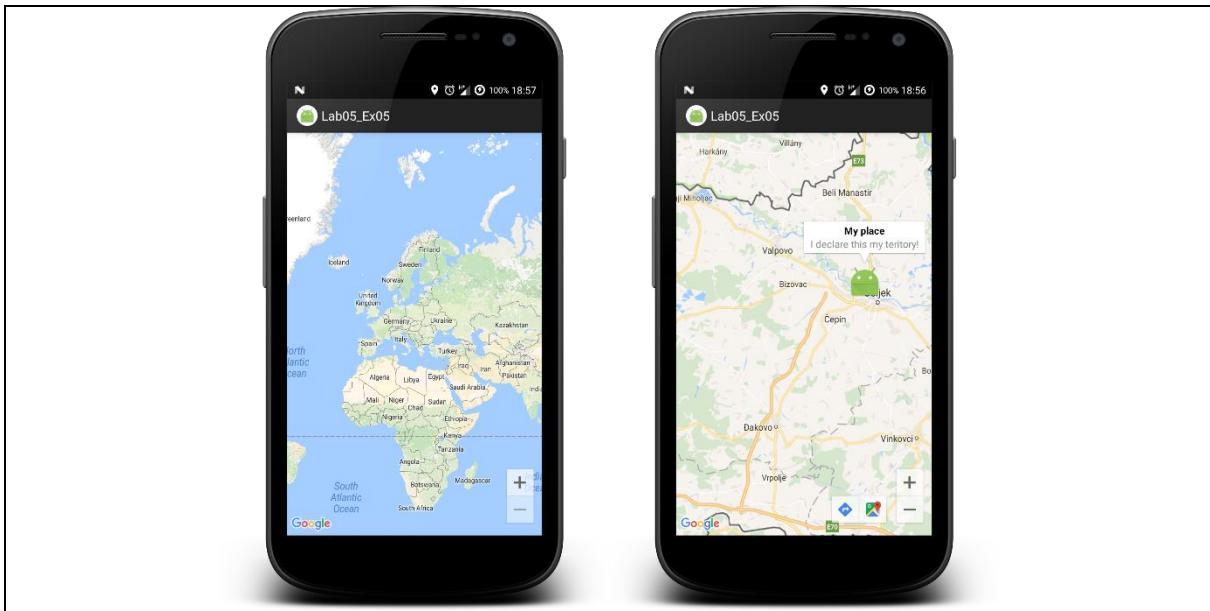
class MapsActivity : AppCompatActivity(), OnMapReadyCallback {

    private lateinit var map: GoogleMap

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_maps)
        val mapFragment = supportFragmentManager.findFragmentById(R.id.map) as SupportMapFragment
        mapFragment.getMapAsync(this)
    }

    override fun onMapReady(googleMap: GoogleMap) {
        map = googleMap

        val osijek = LatLng(45.55111, 18.69389)
        map.addMarker(MarkerOptions().position(osijek).title("Marker in Osijek"))
        map.mapType = GoogleMap.MAP_TYPE_SATELLITE
        map.uiSettings.isZoomControlsEnabled = true
        map.moveCamera(CameraUpdateFactory.newLatLng(osijek))
    }
}
```

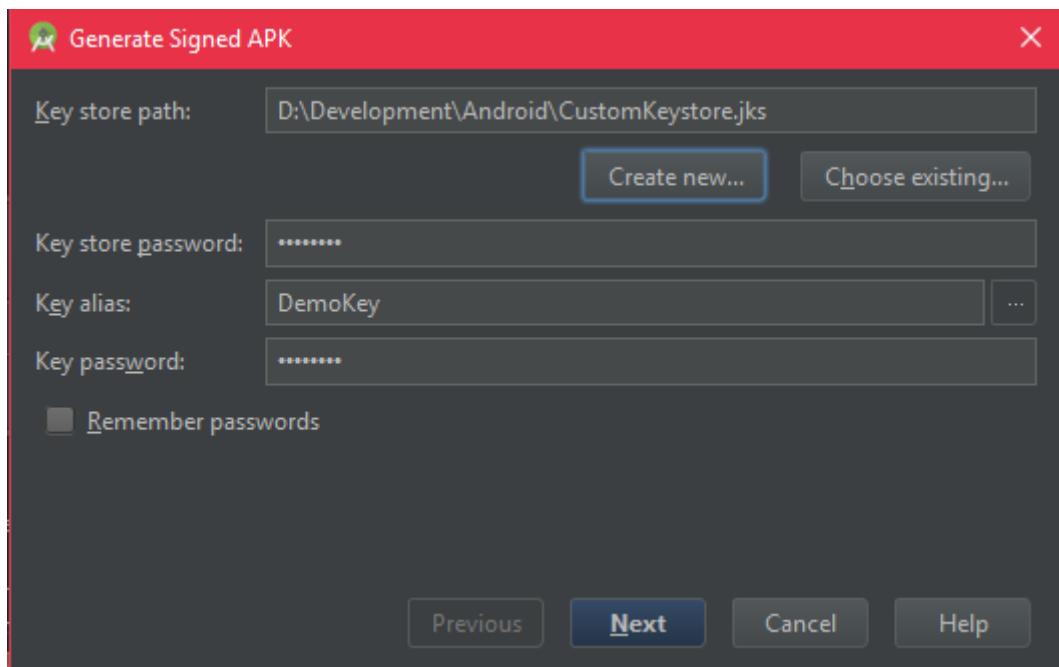


5.3. Objava aplikacija

Jednom kada je aplikacija završena, moguće ju je ponuditi korisnicima na razne načine. Postoje brojne trgovine aplikacijama, a najraširenija je zasigurno Googleov Play Store. Aplikaciju nije nužno nuditi preko bilo kakve trgovine, no njihovo korištenje uvelike olakšava postupak distribucije, ali i monetizaciju aplikacije.

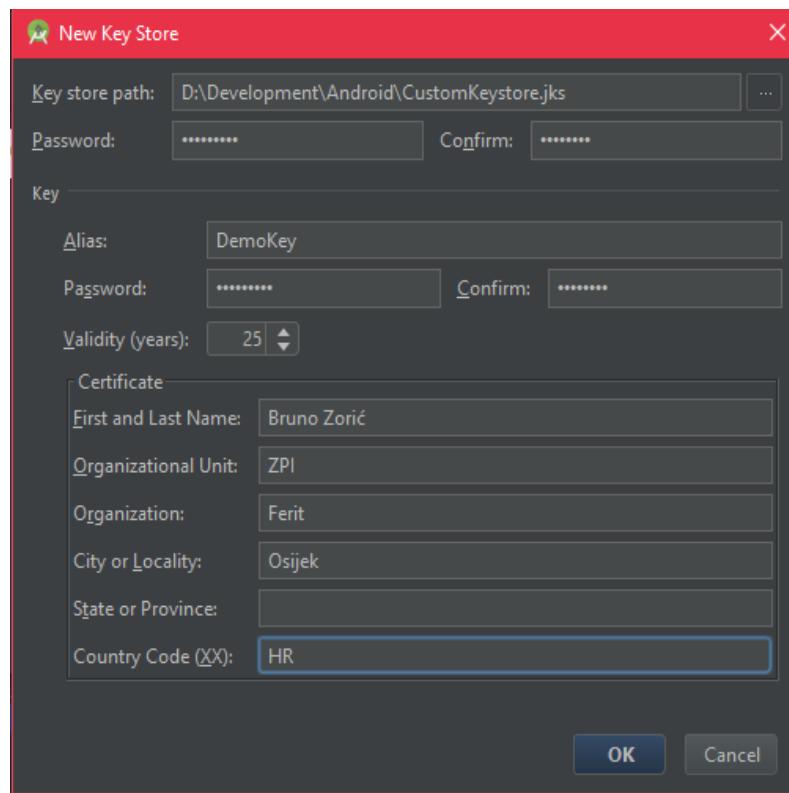
5.4. Potpisivanje

Android aplikacije distribuiraju se u obliku .apk datoteka, a kako bi se mogle instalirati na emulator ili uređaj, moraju biti potpisane. Tijekom razvoja one se također potpisuju, ali *debug* ključem koji se generira automatski. Kako bi se mogle distribuirati, nužno ih je nakon završetka potpisati tzv. *release* ključem. Ovo je moguće učiniti na više načina, a jedan je korištenje čarobnjaka za izvoz Android aplikacija. Sigurnost certifikata kojim se obavlja potpisivanje je vrlo bitna, jer se na temelju njega vrše nadogradnje aplikacije, osigurava autentičnost i slično. U slučaju krađe, postoji mogućnost zamjene Vaše aplikacije malicioznom, a u slučaju gubitka bilo bi nužno potpisati aplikaciju novim ključem pri čemu bi se izgubili svi podaci, korisnici, recenzije itd. na Google Play storeu jer bi svi bili prisiljeni skinuti i instalirati novu verziju.



Slika 5.7. Odabir ili stvaranje novog ključa za potpisivanje aplikacije

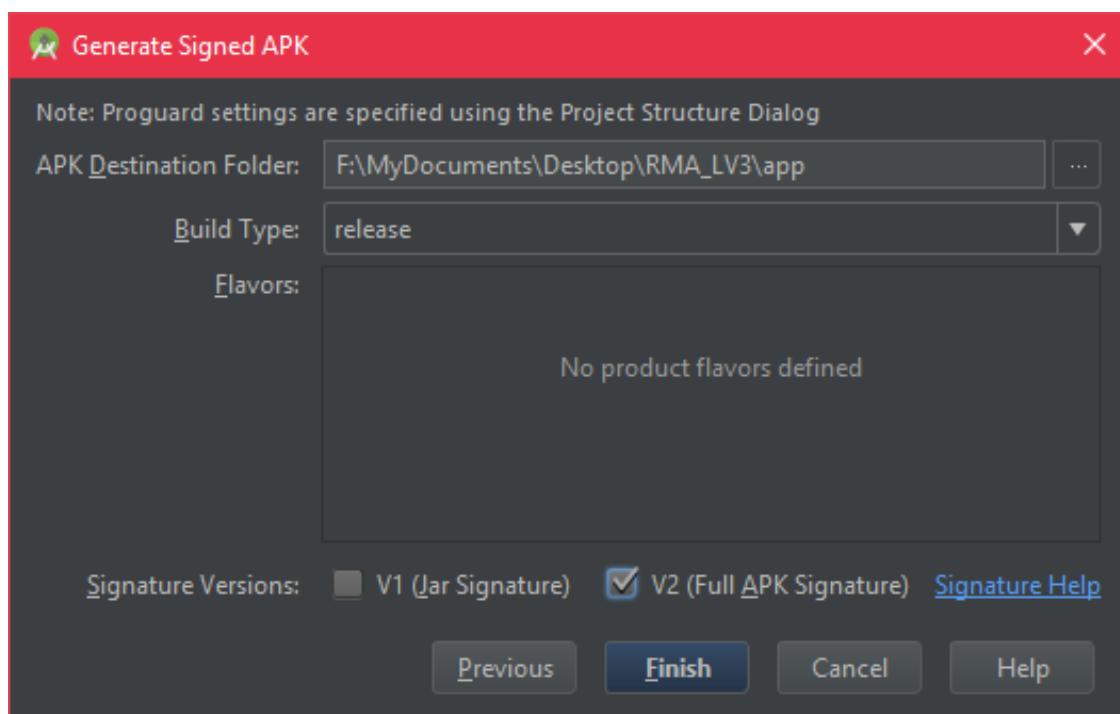
Postupak objave započinje odabirom *Build→Generate signed apk...*. Otvara se pri tom izbornik u kojem je potrebno unijeti detalje vezane uz ključ. Prvenstveno je riječ o putanji do pohrane za ključeve, takozvane *keystore* datoteke. Ukoliko ona ne postoji, tada je potrebno kreirati novu i zaštititi ju lozinkom. U ovu se datoteku pohranjuju ključevi za Vaše aplikacije. Ako ne postoji niti jedan *keystore*, moguće ga je kreirati u novom dijalogu, kako



Slika 5.6. Stvaranje novog keystorea

je prikazano slikom. Kroz taj je izbornik moguće odmah dodati i ključ. Kada se dodaje ključ, bitno je obratiti pozornost na nekoliko stvari. Trajanje ključa ne može biti kraće od 25 godina. Ključ izdajete sami, pa uneseni podaci to i odražavaju. Ako ključ izdaje tvrtka, ovdje mogu biti njeni podaci (iako ovi podaci nisu nužni).

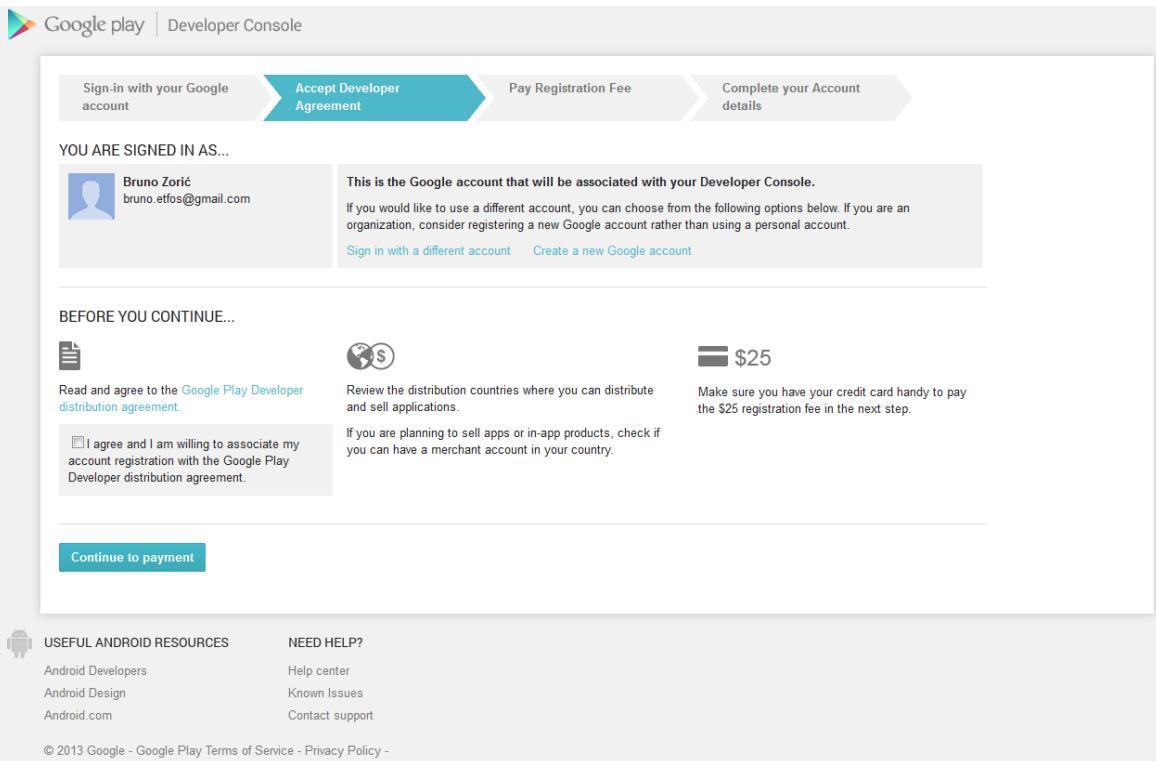
Kada je generiran i odabran ključ, odabirom gumba *Next* odlazi se na sljedeći dijalog koji zahtjeva informacije o tome gdje se želi kreirati .apk datoteka, koji je oblik *builda (debug ili release)*, koja je vrsta potpisa (preferirani je v2, novi način koji omogućuje bržu instalaciju aplikacija) te dodatne postavke vezane uz *build*. Klikom na *Finish* završava se proces pakiranja aplikacije i rezultat je datoteka koja se može postaviti na mobilni uređaj putem nekog kanala za distribuciju. Da bi se aplikacija nadogradila, potrebno je koristiti isti ključ za njeno potpisivanje. Bilo bi dobro, prema Android smjernicama, sve aplikacije potpisati istim certifikatom kako bi se omogućilo dijeljenje funkcionalnosti među njima. Backup *keystore-a* uvijek je dobra ideja, ali ga je bez obzira na backup nužno čuvati sigurnim iz ranije opisanih razloga. Ime paketa (kako je rečeno na prvim LV) mora biti jedinstveno, dok ime .apk datoteke ne mora.



Slika 5.8. Izbor načina potpisivanja i generiranje .apk datoteke

5.5. Google Play Store

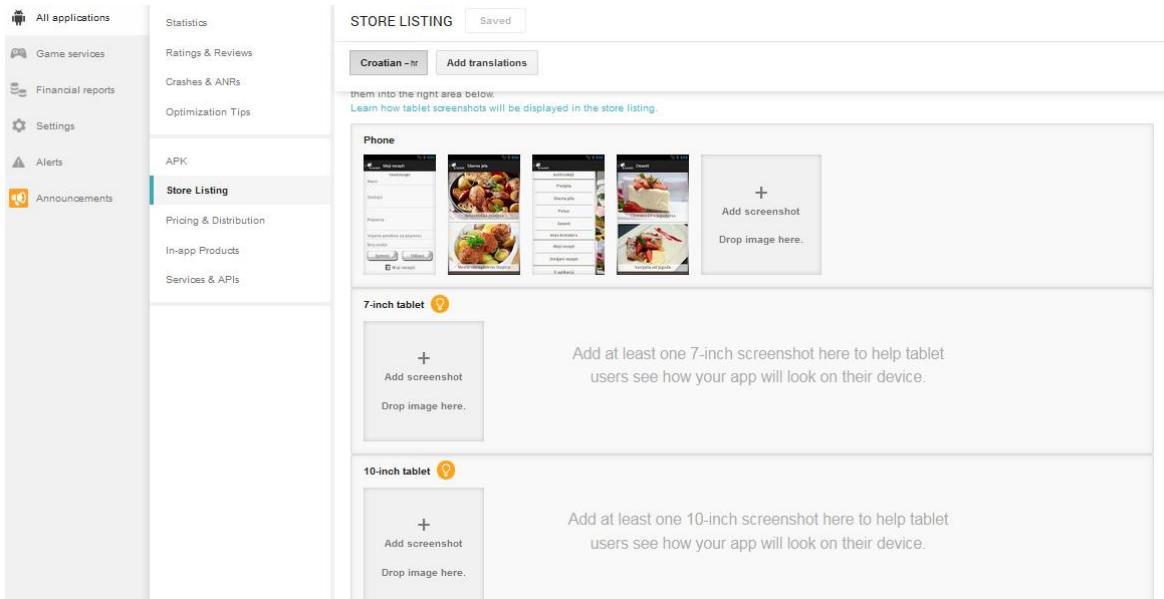
Google Play Store najpopularnija je tržnica aplikacija za Android sustav. Osim njega postoje brojne druge tržnice (Amazon, Samsung itd.), no Googleova je najraširenija i nudi



Slika 5.9. Prijava za Google Developer račun

najveći broj aplikacija. Kako bi se aplikacije mogle objavljivati na Play Storeu, potrebno je registrirati se kao *developer* i platiti 25\$ članstva. Adresa na kojoj je to moguće učiniti je <https://play.google.com/apps/publish/signup/>. Jednom kada je napravljena registracija i povezan Gmail račun, moguće je podići aplikaciju na servis. Potrebno je uz svaku aplikaciju unijeti opisne podatke, definirati dobne skupine za koje je namijenjena, dodati slike i eventualno video koji će predstaviti aplikaciju te obavezno ikonu kojom će aplikacija biti prikazana na Trgovini (ne mora biti identična ikoni aplikacije na uređaju, ali zbog vizualne Google Play Store najpopularnija je tržnica aplikacija za Android sustav. Osim njega postoje brojne druge tržnice (Amazon, Samsung itd.), no Googleova je najraširenija i nudi najveći broj aplikacija. Kako bi se aplikacije mogле objavljivati na Play Storeu, potrebno je registrirati se kao *developer* i platiti 25\$ članstva. Adresa na kojoj je to moguće učiniti je <https://play.google.com/apps/publish/signup/>. Jednom kada je napravljena registracija i povezan Gmail račun, moguće je podići aplikaciju na servis. Potrebno je uz svaku aplikaciju unijeti opisne podatke, definirati dobne skupine za koje je namijenjena, dodati slike i eventualno video koji će predstaviti aplikaciju te obavezno ikonu kojom će aplikacija biti

prikazana na Trgovini (ne mora biti identična ikoni aplikacije na uređaju, ali zbog vizualne konzistentnosti taj odabir bi bio dobar). Izgled sučelja za registraciju, kao i za *upload* aplikacije prikazan je slikom.



Slika 5.10. Upload nove aplikacije

5.6. Primjeri

Zadatak 1.

Kreirajte vlastitu *soundboard* aplikaciju koja će pustiti određeni zvuk klikom na neku sliku na sučelju. Zapakirajte i potpišite aplikaciju ključem u trajanju od 50 godina.

Osnovni zahtjevi:

- Kreirati osnovni Activity s ImageButton kontrolama
- Povezati gume s različitim zvukovima

A smartphone is shown displaying a mobile application. The screen shows three portrait photographs of men stacked vertically. The top photo is of a man with dark hair, the middle is of a smiling man with glasses, and the bottom is of a man with a beard. This is likely a prototype of a soundboard application where each photo is a button that plays a specific sound when tapped.

5.7. Zadaća

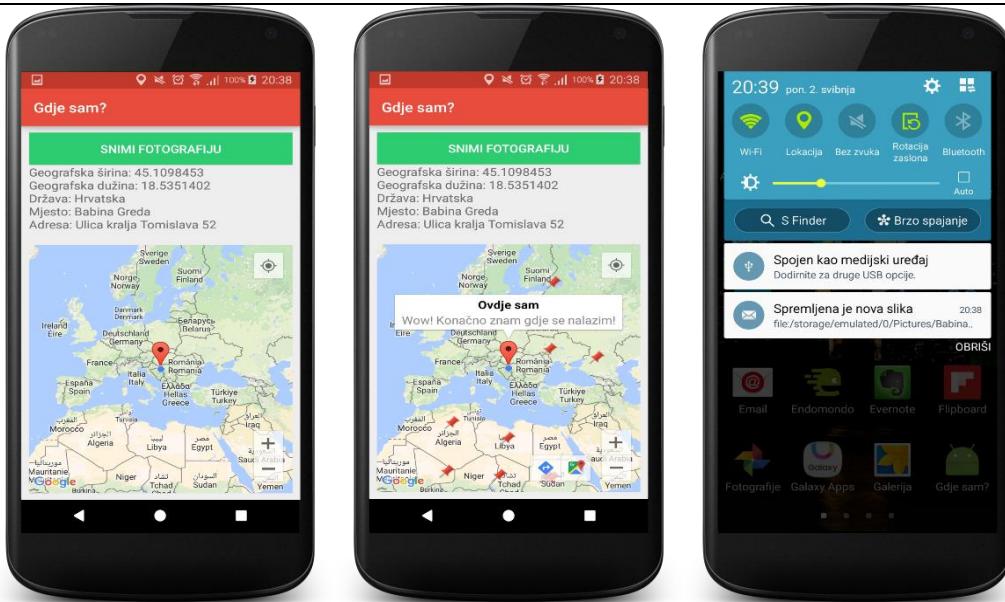


Zadaća 5.

Gdje je ____?

Kreirajte aplikaciju pod nazivom „Gdje je {vaše_ime}?“. Aplikacija treba korištenjem lokacije na karti prikazati trenutni položaj korisnika markerom. Osim prikaza karte, aplikacija na sučelju treba ispisati lokaciju, adresu, grad i zemlju u kojoj se korisnik nalazi. Omogućiti postavljanje markera drugačije ikone na kartu, prilikom dodavanja markera pustiti zvuk preko *SoundPool* klase. Omogućiti slikanje klikom na gumb pri čemu se slika spremna na uređaj, ime se zadaje pomoću trenutne lokacije, a korisniku se šalje notifikacija klikom na koju se otvara ta slika u galeriji. Zapakirajte i potpišite aplikaciju ključem u trajanju od 50 godina.

- Osnovni Activity s map fragmentom
- Dodavanje markera
- Puštanje zvuka po dodavanju markera
- Slikanje uporabom kamere
- Potpisivanje i izvoz aplikacije



Napomena:

Svaka je nadogradnja poželjna i dobrodošla. Korištenje znanja koja nadilaze opseg dosadašnjih vježbi se potiče.