Aktivnosti (engl. *Activities*)

Konceptualno, svaka aplikacija je:

- skup od jedne ili više aktivnosti (Activity)
- svaka aktivnost je nezavisna
- npr. E-mail program može imati
  - aktivnost koja prikazuje popis svih e-mailova
  - aktivnost koja omogućava pisanje i slanje e-maila
  - aktivnost koja omogućava čitanje jednog konkretnog e-maila
- ovo omogućava da drugi programi zatraže obavljenje neke od aktivnosti koje program nudi (primjerice, program koji hvata sliku s kamere može zatražiti pokretanje aktivnosti stvaranja i slanja e-maila koji kao privitak sadrži tu sliku)

Podrazredi od `android.app.Activity`.

Aktivnost omogućava:

- specificiranje sadržaja koji korisnik trenutno vidi na ekranu
- praćenje gdje je korisnik bio prije (koncept stoga!)
- pohranu podataka prije uništavanja aktivnosti (kada nedostaje resursa u sustavu pa se aktivnost zbog toga uništava) i restauriranje sadržaja pri povratku na aktivnosti
- definiranje prijelaza iz jedne aktivnosti u drugu i razmjene podataka od aktivnosti do aktivnosti

Usluge (engl. *Services*)

Konceptualno, aplikacija bez korisničkog sučelja koja se izvodi u pozadini te obavlja poslove za druge aplikacije (npr. sviranje muzike u pozadini dok korisnik radi druge stvari, download podataka u pozadini, ...).

Podrazredi od `android.app.Service`.

Nećemo ih obrađivati na predavanju.

Primatelji obavijesti (engl. *Broadcast Receivers*)

Konceptualno, komponente koje omogućavaju aplikaciji primitak obavijest mimo onih koje bi aplikacija dobivala isključivo zbog aktivne aktivnosti (primjerice: dojave na razini operacijskog sustava poput: stanje baterije je nisko, ekran se ugasio, i slično). Aplikacije mogu potom odraditi neku akciju. Aplikacije i same mogu slati obavijesti – poput, završen je download neke datoteke i slično.

Ove komponente ne mgu stvarati korisničko sučelje, ali mogu generirati obavijesti koje će se prikazati u statusnoj traci (engl. *status bar*).

Podrazredi od `android.content.BroadcastReceiver`. Svaka se obavijest dojavljuje u obliku jednog objekta koji je tipa `android.content.Intent`.

Upravitelji podatcima (engl. *Content Providers*)

Omogućavaju aplikacijama pohranu, dohvat, izmjene i pretraživanje imenovanih podataka. Podatci se mogu čuvati u različitim spremnicima (datoteke u datotečnom sustavu, bazi podataka SQLite i slično).

Nisu zamišljeni kao općenita apstrakcija nad bazama podataka, već za rad s imenovanim sadržajima. Primjerice: zamislimo popis kontakata korisnika.

- Sastoji se od skupa osoba
- Svaka osoba ima jedinstveni URI
- Upravitelj podatcima omogućava dohvat podataka za konkretnu osobu, pohranu podataka za novu osobu te izmjenu podataka o osobi.
-

Aktiviranje komponenata

Aktivnosti, usluge te primatelji obavijesti aktiviraju se slanjem asinkronih poruka koje su tipa `android.content.Intent`. Ti objekti povezuju komponentu koja je stvorila i poslala poruku s komponentom koja je aktivirana u svrhu obrade te poruke (te komponente mogu biti čak iz različitih aplikacija).

`Intent` može biti:

- *eksplicitni* (kada direktno navodi koju komponentu treba aktivirati) ili
- *implicitni* (specificira vrstu poruke, pa operacijski sustav bira ili čak nudi korisniku da odabere čime želi obraditi tu poruku).

Primjer implicitnog intenta je "otvaranje" PDF-datoteke: ako Vaš operacijski sustav ima instalirano više aplikacija koje omogućavaju čitanje PDF-dokumenata, dobit ćete popis aplikacija i moći odabrati s kojom od njih želite otvoriti dokument. Slično: potrebno je prikazati određenu stranicu za zadani URL: ako imate više instaliranih web-preglednika, OS će Vas pitati kojim od njih želite otvoriti specificiranu web-stranicu. Slično: treba stvoriti e-mail, treba ...

Kako je `Intent` poruka, isti sa sobom može prenijeti i niz podataka na temelju kojih će aktivirana komponenta odraditi svoj posao.

Aktivnost može biti aktivirana i kako bi pozivatelju vratila rezultat: u tom slučaju rezultat će također biti oblikovan kao novi objekt tipa `Intent`. (ovo ćemo trebati!)

Različite se komponente aktiviraju na različite načine.

- Aktivnosti se aktiviraju slanjem `Intent`a kroz metode `startActivity()` ili `startActivityForResult()` - ako natrag očekujemo rezultat.
- Od Androida 5.0 (API razine 21) možemo koristiti `app.job.JobScheduler`.
- S uslugama se može raditi slanjem `Intent`a kroz `startService()` i `bindService()`.
- Obavijesti se mogu slati predajom `Intent`a kroz metode `sendBroadcast()`, `sendOrderedBroadcast()`, `sendStickyBroadcast()`.
- Upravitelje podatcima možemo kontaktirati metodom `query()` nad `android.content.ContentResolver`.

Aplikacija

Aplikacija za android strukturira se analogno web-aplikaciji: gradi se APK-arhiva koja sadrži prevedeni programski kod, relevantne resurse te opisnik aplikacije (manifest; xml-datoteka imena `AndroidManifest.xml`).

Manifest:

- deklarira koje sve komponente postoje u aplikaciji (slično kao što web.xml u web-aplikaciji definira servlete, filtere i promatrače),
- deklarira dozvole koje su potrebne za izvođenje aplikacije (primjerice, aplikacija treba pristup za čitanje nad korisnikovim kontaktima, treba pristup internetu, ...)
- deklarira što aplikacija treba (kamera, bluetooth, ...)
- deklarira dodatne biblioteke koje su potrebne za izvođenje aplikacije (npr. Google Maps Library).

Deklariranje aktivnosti u manifestu:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png" ... >
        <activity android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label" ... >
        </activity>
        ...
    </application>
</manifest>
```

Primijetite odmah: argument od `android:label` predstavlja naziv aktivnosti namijenjem čovjeku – međutim, umjesto izravno upisanog naziva ovdje se nalazi string specifičnog formata: `"@string/"` + nekakav ključ. Operacijski sustav Android nativno podržava internacionalizaciju, te je uobičajena praksa na sva ovakva mjesta pisati string-ključeve i potom u lokalizacijskim datotekama pisati njihove prijevode.

Komponente se, ovisno o vrsti, trebaju deklarirati tagovima:

- <activity>
- <service>
- <receiver>
- <provider> (ili se mogu programski stvoriti i registrirati pozivom `registerReceiver()`)

Ako neka komponenta može obraditi neki Intent, to je potrebno specificirati kao `<intent-filter>` za tu komponentu. Primjerice, ako konkretnu aktivnost možemo koristiti za slanje e-mail poruke:

```
<manifest ... >
    ...
    <application ... >
        <activity android:name="com.example.project.ComposeEmailActivity">
            <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <data android:type="*/*" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Ako sada neka druga aplikacija stvori `Intent` s akcijom `ACTION_SEND` (vidi `Intent.html#ACTION_SEND`) i taj `Intent` preda metodi `startActivity()`, moguće je da kao rezultat bude pokrenuta prethodno definirana aktivnost.

Ako aplikacija ima zahtjeve (primjerice, minimalnu razinu API-ja s kojom radi), to također treba definirati u manifestu:

```
<manifest ... >
    <uses-feature android:name="android.hardware.camera.any"
                  android:required="true" />
    <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="19" />
    ...
</manifest>
```

Kako je za kameru postavljeno `android:required="true"`, ova se aplikacija neće moći niti instalirati na telefone koji nemaju kameru. Ako bi bilo `android:required="false"`, aplikacija bi se mogla instalirati no morala bi programski tijekom izvođenja provjeriti postoji li kamera pa prilagoditi svoje ponašanje.

Kratko o resursima

Android-aplikacija može koristiti niz različitih vrsta resursa (slike, stilove, boje, tekstove, ...). Alat za izgradnju aplikacije svakom pojedinom resursu dodjeljuje jedinstveni cijeli broj tipa integer. Primjerice, ako imamo sliku `logo.png` spremljenu u direktorij `res/drawable`, automatski će u razred `R` biti dodan statički ugniježđeni razred `drawable` i u njega statička `int`-konstanta `logo`, pa ćemo u Javi u izvornom kodu moći pisati `R.drawable.logo`. Preko ovog cijelog broja moći ćemo referencirati navedenu sliku te je dodati primjerice u komponentu za prikaz slike kako bi je prikazali korisniku).

Internacionalizacija se postiže specificiranjem resursa za pojedine jezike tako da se na direktorij doda sufiks koji odgovara jezičnom kodu; primjerice:

```
res/values-hr/
res/values-en/
res/values-de/
```

Ovisno o karakteristikama ekrana (npr. *portret* vs. *landscape*) možemo definirati različite razmještaje (engl. *layout*) koje će pojedina aktivnost koristiti za slaganje korisničkog sučelja.

Za više informacija o prethodno napisanome:
https://developer.android.com/guide/components/fundamentals

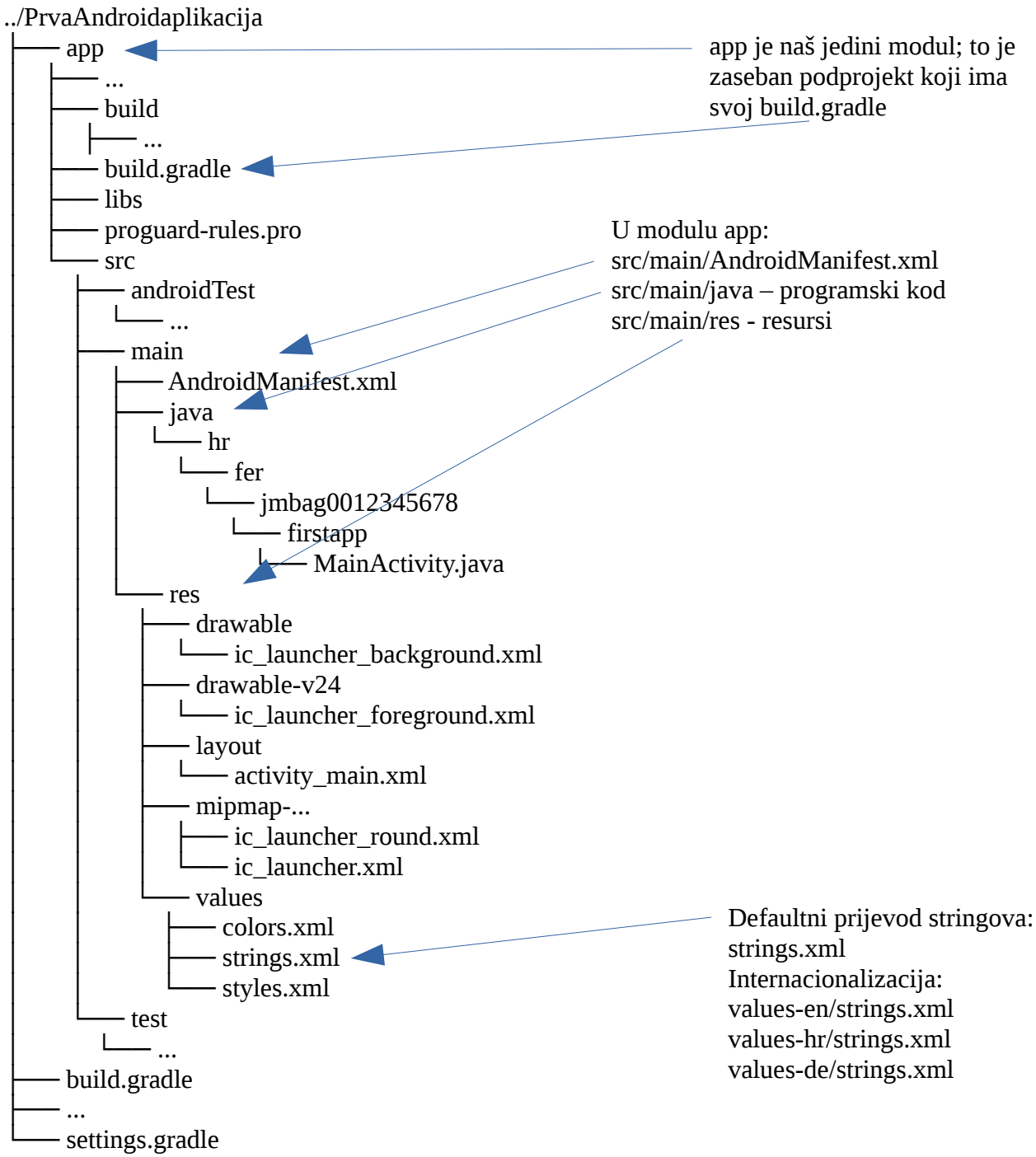
Idemo napraviti novi projekt

Android Studio -> New Project

- Phone and Tablet
  - Empty Activity
  - Next
- Postavite:
  - Name: `Prva Android aplikacija`
  - Package name: `hr.fer.jmbag0012345678.firstapp`
  - Save location: gdje želite snimiti projekt, npr. `/home/marcupic/AndroidStudioProjects/MyApplication`
  - Language: `Java` (po defaultu je Kotlin, pazite!)
  - Minimum API level: `23`
  - This project will support instant apps: ne
  - Use androidx.* artifacts: ne
  - Finish

Za izgradnju projekta koristi se alat gradle, koji slično kao i Maven omogućava izvođenje svih faza u izgradnji projekta – od prevođenja, preko testiranja do pakiranja. Omogućava deklariranje ovisnosti prema bibliotekama koje su dostupne u Mavenovim repozitorijima. Glavni opisnik projekta je datoteka `build.gradle` koja se mora nalaziti u korijenskom direktoriju projekta.

Jedan projekt može imati više podprojekata koji dalje imaju vlastite opisnike. Tako je u našem slučaju napravljena struktura (izbacio sam dijelove):


../PrvaAndroidaplikacija
app
...
build
...
build.gradle
libs
proguard-rules.pro
src
androidTest
...
main
AndroidManifest.xml
java
hr
fer
jmbag0012345678
firstapp
MainActivity.java
res
drawable
ic_launcher_background.xml
drawable-v24
ic_launcher_foreground.xml
layout
activity_main.xml
mipmap-...
ic_launcher_round.xml
ic_launcher.xml
values
colors.xml
strings.xml
styles.xml
test
...
build.gradle
...
settings.gradle
app je naš jedini modul; to je zaseban podprojekt koji ima svoj build.gradle
U modulu app:
src/main/AndroidManifest.xml
src/main/java – programski kod
src/main/res - resursi
Defaultni prijevod stringova:
strings.xml
Internacionalizacija:
values-en/strings.xml
values-hr/strings.xml
values-de/strings.xml

Nakon stvaranja projekta dobili smo prvu aktivnost:
`hr.fer.jmbag0012345678.firstapp.MainActivity`

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

Aktivnost pri stvaranju stvara korisničko sučelje prema razmještaju `R.layout.activity_main`. Ovaj razmještaj definiran je XML-datotekom:
`app/src/main/res/layout/activity_main.xml`
i tamo možemo podešavati strukturu korisničkog sučelja.

Osnovna ideja Androida jest da se struktura korisničkog sučelja (elementi koji postoje prikazani na aktivnosti te njihov razmještaj) ne specificiraju programski kao što je slučaj kod Swinga, već da se opišu jednom XML-datotekom. Prilikom stvaranja aktivnosti, nad aktivnosti se pozivom metode `setContentView` da referenca na opisnik razmještaja i ta će metoda potom stvoriti sve relevantne komponente i postaviti ih u korisničko sučelje. Ako će nam trebati referenca na neku od komponenata (primjerice, na neki gumb ili kutiju za unos teksta), postoje metode kojima možemo dohvatiti reference na njih.

Datoteka s definicijom razmještaja:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        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" />
</android.support.constraint.ConstraintLayout>
```

Tag **`android.support.constraint.ConstraintLayout`** definira istovremeno kontejner koji možemo koristiti za razmještanje djece po njegovoj površini te upravljač razmještajem koji raspoređuje djecu. Za razliku od Swinga gdje su kontejneri (npr. `JPanel`) razdvojeni od upravljača razmještajem (npr. `BorderLayout`, `GridLayout`), na Androidu je svaka kombinacija kontejner+upravljač spojena u zaseban razred.

Tag **`TextView`** definira komponentu za prikaz teksta. Koji je tekst inicijalno prikazan definirano je atributom **`android:text`** i u ovom primjeru sadržaj je eksplicitno "hardkodiran" – bolje je to internacionalizirati: ovdje definirati neku konstantu poput `@string/zdravo_svijete` i zatim u `src/main/res/values/strings.xml` definirati prijevod pridružen toj konstanti.

Ako komponentu želimo programski dohvaćati iz aktivnosti, trebamo joj dati identifikator. Stoga bismo u prethodnom primjeru u tag **`TextView`** dodali još primjerice **`android:id="@+id/tvHelloWorld"`** što bi nam omogućilo da u kodu pišemo:

```
TextView tv =  (TextView) findViewById(R.id.tvHelloWorld);
```

Pogledajmo još i manifest aplikacije:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="hr.fer.jmbag0012345678.firstapp">
    <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>
```

Svaka se aplikacija razlikuje po paketu koji je deklariran kao atribut taga **`manifest`**.

Manifest potom navodi jednu aktivnost imena **`".MainActivity"`** i ta se može koristiti kada korisnik želi pokrenuti ovu aplikaciju (što je vidljivo iz sadržanog taga **`<intent-filter>`**).

Mjerne jedinice

Pri specifikaciji željene širine i visine komponenata, kao i pri specifikaciji veličine fonta, na Androidu se mogu koristiti različite mjerne jedinice.

Različiti telefoni/tableti danas dolaze s različitim gustoćama slikovnih elemenata koje variraju od `100` do `480` slikovnih elemenata po jednom inču (`dpi`: *dots per inch*). Ako dimenzije izražavamo u slikovnim elementima (npr. širina je `100px`), na ekranima s različitim gustoćama element će imati različitu "fizičku" širinu: na ekranu gustoće `100 dpi`, element će biti širok `1 inch`; na ekranu gustoće `480 dpi` element će biti širok `0.21 in` što je skoro 5 puta manje.

Stoga se na Androidu mogu koristiti mjerne jedinice `dp` (*density-independent pixels*) te `sp` (*scalable pixels*). Stvarne fizičke dimenzije prikazanog elementa čije su dimenzije zadane u `dp` na ekranima različitih gustoća bit će jednake: na temelju zadane dimenzije i gustoće ekrana preračunat će se koliko slikovnih elemenata dimenzija treba biti. Jedinica `sp` koristi se za specificiranje veličine fontova, jer osim što se ponaša kao `dp`, još se dodatno skalira s korisnikovim odabirom veličine fonta (npr. osobe sa slabijim vidom mogu u postavkama telefona zatražiti da se fontovi prikazuju veći).