Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Preddiplomski studij matematike

Ivan Škorić

Osnove izrade aplikacija za operativni sustav Android

Završni rad

Osijek, 2012.

Sveučilište J.J. Strossmayera u Osijeku
Odjel za matematiku
Preddiplomski studij matematike

Ivan Škorić

Osnove izrade aplikacija za operativni sustav Android

Završni rad

Mentor: doc.dr.sc. Domagoj Matijević

Osijek, 2012.

Sadržaj

Sažetak.

U završnom radu ćemo predstaviti Android operativni sustav i osnove programiranja aplikacija za isti u programskom jeziku Javi. Pokazat ćemo početke i nastanak Androida, te njegov napredak do danas. Objasnit ćemo osnovnu strukturu aplikacija, te proći kroz rad pojedinih komponenti, njihovu uporabu te ciklus rada. Proći ćemo kroz osnove upravljanja i povezivanje sa SQLite bazom podataka, te upravljanje izgledom aplikacije kroz XML jezik za označavanje podataka i programski, koristeći Javu . Sve što budemo obrađivali pokazat ćemo na primjeru aplikacije koja prati promjene na službenim stranicama Odjela za matematiku u Osijeku. Sučelje koje ćemo koristiti pri izradi aplikacija je Eclipse.

Ključne riječi:

Android, programiranje, Java, XML, mobilne aplikacije, Eclipse.

Abstract.

In this final paper we are going to present Android operating system and basics of programming Android applications in programming language Java. We will show the beginning and development of Android, and its progress to date. We will explain the basic structure of applications and go through the work of individual components, their use, and their lifecycle. We will go through the basics of managing and connecting with SQLite database, managing application layout through XML markup language and programatically, using Java. Everything we go through will be shown on the example of application that follows changes on the official websites of Department of Mathematics in Osijek. An interface we are going to use to build applications is Eclipse.

Keywords:

Android, programming, Java, XML, mobile applications, Eclipse.

Uvod

Android je operativni sustav dizajniran prvenstveno za uređaje sa zaslonom na dodir kao što su smartphone-ovi i tableti, baziran na Linuxovoj jezgri. Razvoj Androida preuzeo je Google 2005. godine od tvrtke Android Inc. Od tada je ova platforma doživjela enorman uspjeh, a broj uređaja koji koriste Android svakim danom je sve veći.

Google je predstavio Android 2007. kao open-source platformu. Od tada, Android ima veliku zajednicu developera koji prave Android aplikacije, pisane prvenstveno u Javi, a aplikacije mogu biti preuzete preko online trgovina kao što su Google Play (bivši Android Market), Amazon, GetJar, itd.

Neke prednosti programiranja za Android: platforma otvorenog koda (open-source), programiranje u Javi, svaka aplikacija je prihvaćena u kratkom roku od objave, mnoštvo dokumentacije, velika zajednica programera voljnih pomoći, broj korisnika raste svaki dan, itd. U prvom poglavlju objašnjavamo osnovnu arhitekturu aplikacije, Android Manifest, te detaljnije proučavamo aktivnosti i servise.

Drugo poglavlje se bavi instalacijom razvojnog okruženja odnosno pripremanjem potrebnih stvari za razvoj Android aplikacije na računalu. Također objašnjeni su dijelovi projektne datoteke.

U trećoj cijelini bavimo se izradom korisničkog sučelja, odnosno komunikacijskog kanala između korisnika i određenih instanci klasa `View` i `ViewGroup`.

Završna cijelina daje nam uvid u neke dijelove izrade aplikacije koja prati promijene na stranicama Odjela za matematiku, Osijek. Ti primjeri su napredniji i cijeloviti djelovi koda za rješavanje postavljenog problema.

1 Arhitektura aplikacije

Android je građen od više ugrađenih slojeva. Aplikacije se nalaze na vrhu, u sredini su okviri aplikacija (framework), biblioteke, Android runtime i sl. Na dnu se nalazi Linux jezgra za raznoraznim driverima. Android pokreću virtualni uređaj Dalvik (DVM) i biblioteke jezgre (Core Libraries). Svaka Android aplikacija dobiva svoj vlastiti Linux proces kao instancu Dalvik virtualnog uređaja. Dalvik pretvara Java kod u DEX format koji je zapakiran u aplikacijski paket, sa nastavkom .apk koji Android koristi kako bi instalirao aplikaciju na uređaj.

Aplikacija je skup komponenti: *aktivnosti, servisi, content provideri, broadcast receiveri*. Oni se vrte u Linux procesima i njima upravlja Android.

Aktivnost (activity) je komponenta koja prikazuje korisničko sučelje tako da korisnik može bit u interakciji s aplikacijom. Npr. aplikacija kalukulator prikazuje korisničko sučelje za upisivanje brojeva i ispis rezultata.Iako aplikacije mogu sadržavat samo jednu aktivnost, najčešće to nije slučaj. Primjerice, kalkulator može imati sučelje sa postavkama želimo li znanstveni ili programerski kalkulator i slično.

Servis (service) je komponenta koja se pokreće u pozadini u neograničenom vremenu i nema vlastito sučelje (npr. sviranje glazbe). Servise dijelimo na *lokalne (local service)*, koji se vrte u istom procesu kao i ostatak aplikacije i na *udaljene (remote service)*, koji se vrte u zasebnom procesu.

Broadcast Receiver je komponenta koja prima reakcije na broadcastove (emitirajuće signale), npr. promjena vremenske zone, slaba baterija i sl. Također aplikacija može i sama odaslati broadcast.

Content Provider služi za razmjenu podataka između različitih aplikacija. Ti podaci mogu biti spremljeni u Android filesystem, SQLite database, itd.

Intent je poruka koja opisuje koji postupak odnosno koja operacija će se izvršiti. Npr. slanje e-maila, prelazak na drugu aktivnost, odabir slika, poziv i sl. Gotovo sve u Androidu koristi Intent pa je moguće zamjeniti postojeće komponente vlastitima. Npr. ukoliko neka aplikacija želi poslati mail ona zahtjeva slanje maila preko Intenta. Kada se to pozove, otvaraju se svi mogući načini slanja e-maila, odnosno sve aplikacija preko kojih se može slati mail. Tako i mi možemo napraviti našu aplikaciju za slanje maila koja će primati „send mail“ Intent i na njega odgovarati.

1.1 Android Manifest

Android saznaje za aktivnosti i ostale dijelove aplikacije preko **Android Manifesta** odnosno datoteke *AndroidManifest.xml*, gdje su definirane i navedene sve komponente koje se koriste u aplikaciji. Ukoliko neki dio nije naveden u manifestu, Android ga neće prepoznati.

Primjer 1. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
    package="com.primjer.tutorial "
    android:versionCode="1"
    android:versionName="1.0" >

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".MojaAktivnost1"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.
                    MAIN" />
                <category android:name="android.intent.category.
                    LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Na početku imamo `<?xml version="1.0" encoding="utf-8"?>` što nam je obavezni početni dio, koji označava koju verziju i encoding koristimo.

U samom manifest tagu dajemo opis aplikacije. Prva naredba je obavezna, a u njoj definiramo Android namespace. Package daje ime paketa u kojem se nalazi aplikacija. Ime samoga paketa prati standardnu konvenciju za označavanje paketa u Javi. Zatim versionCode je broj verzije – uvijek cjelobrojna vrijednost, a versionName je naziv verzije, proizvoljno.

U tagu `<action>` definiramo sve komponente aplikacije. Icon i label su reference na mjesto gdje smo snimili ikonu odnosno opis aplikacije zapisan u `String`.

Nadalje imamo aktivnost naziva MojaAktivnost1, ali zapisujemo ju s točkom ispred imena zato jer se ona nastavlja na paket znaci npr.

com.primjer.tutorial.MojaAktivnost1.

Sljedeće što imamo je <intent-filter> koji služi kako bi se identificirala aktivnost ukoliko je pozvan određeni Intent:

- Prvi tag <action> identificira koju ćemo akciju izvest. U ovom slučaju android.intent.action.MAIN označava da je ovo početna aktivnost u aplikaciji.
- Kategorija komponente određena je u <category> tagu. Atribut android.intent.category.LAUNCHER znači da se ova aktivnost pokreće iz application launchera.

1.2 Aktivnosti

- opisane podklasama klase `android.app.Activity`
- `android.app.Activity` je indirektna podklasa apstraktne klase `android.content.Context`

Napomena: `Context` je apstraktna klasa koja pristupa globalnim informacijama, te omogućava aplikacijama izvršavanje kontekstualnih operacija, kao pokretanje aktivnosti i servisa, odašiljanje intenta, otvaranje datoteka itd.

Svaka aktivnost ima svoj životni ciklus. No, prije nego ga objasnimo napišimo primjer koda za neku proizvoljnu aktivnost:

Primjer 2. MojaAktivnost.java

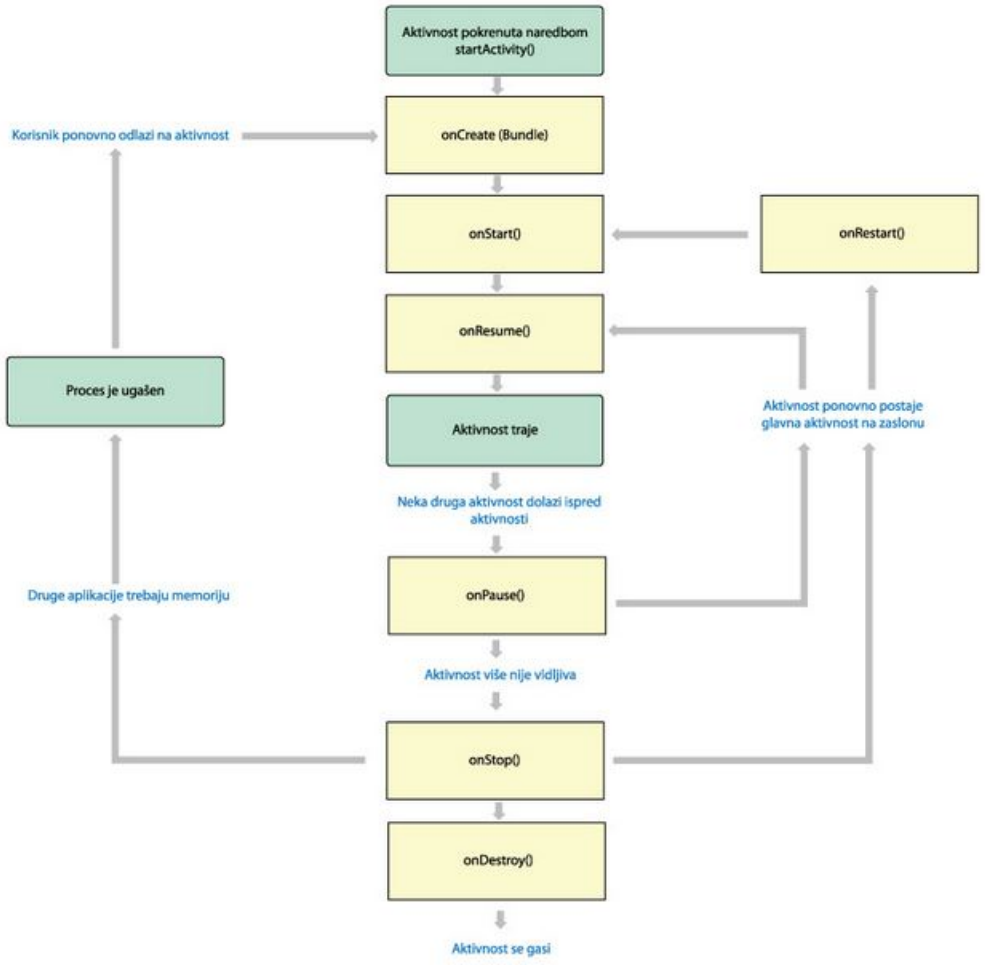
```
1  import android.app.Activity;
2  import android.os.Bundle;
3
4  public class MojaAktivnost extends Activity{
5      @Override
6      public void onCreate(Bundle savedInstanceState){
7          super.onCreate(savedInstanceState); //prvo pozivamo
               funkciju nadklase
8          System.out.println("pozvana je metoda onCreate()");
9      }
10
11     @Override
12     public void onDestroy(){
13         super.onDestroy(); //prvo pozivamo funkciju nadklase
14         System.out.println("pozvana je metoda onDestroy()");
15     }
16 }
```

Napravili smo override funkcija `onCreate(Bundle)` i `onDestroy()`, te prije bilo kakvog koda prvo smo pozvali `super.onCreate(Bundle)` i `super.onDestroy()`, što su metode nadklase `Activity` za koje pravimo override. Njih uvijek pozivamo da se prvo izvrši određena metoda u samoj klasi `Activity`, a nakon toga pišemo vlastiti kod koji će se izvršiti pri pokretanju određene metode.

Funkcija `onCreate(Bundle)` će se pokrenuti pri stvaranju aktivnosti, a `onDestroy()` pri uništavanju, no ovo je osnovni primjer dok je životni ciklus podrobnije objašnjen u sljedećem odjeljku.

1.2.1 Životni ciklus aktivnosti

Životni ciklus aktivnosti koristi 7 funkcija: `onCreate(Bundle)`, `onStart()`, `onPause()`, `onResume()`, `onStop()`, `onRestart()`, `onDestroy()`.



Slika 1: Životni ciklus aktivnosti

Aktivnost nastaje naredbom `onCreate(Bundle)` koja prima jednu `Bundle` varijablu. Ta `Bundle` varijabla sadrži podatke o prethodnom stanju aktivnosti ukoliko ga je bilo. Npr, ako smo na nekoj aktivnosti nešto mjenjali i poslje se na nju vratimo, najčešće želimo da nastavimo u onom stanju gdje smo stali. Za to će se pobrinuti `Bundle` varijabla. U `onCreate()` želimo staviti sve što nam je potrebno pri inicijaliziranju aktivnosti (pozadinu, kućice, tipke, itd.).

Naredbe `onStart()` i `onResume()` prvenstveno služe za povratak iz `onStop()` i `onPause()` stanja. U nju možemo staviti dio koda ukoliko želimo da se nešto u aktivnosti promjeni pri povratku na nju.

Kada pređemo sa jedne aktivnosti na drugu, prva aktivnost se zaustavlja odnosno izvršava naredbu `onStop()` kratko prolazeći kroz `onPause()` i čeka. Ili će biti ponovno pokrenuta naredbom `onRestart()` ili, ukoliko sustav odluči da ju više ne treba, biti će uklonjena.

Aktivnost je u `onPause()` stanju kada se vidi u pozadini ali nije aktivna. Npr. kada nam se ponudi upitnik pomoću koje aplikacije želimo podjeliti neki sadržaj, u pozadini vidimo našu aktivnost koja je neaktivna. Kada iz tog stanja ponovo postaje aktivna izvršava se naredba `onResume()`.

Naredba `onDestroy()` se izvršava pri gašenju aplikacije. Pošto se ta naredba ne mora nužno izvršiti ne preporuča se u njoj izvoditi bitne funkcije, već u `onPause()` ako npr. želimo izmjeniti podatke od content providera i sl.

1.2.2 Pozivanje jedne aktivnosti iz druge

Primjetite sa *slike 1.* da aktivnost pozivamo naredbom `startActivity()`. Funkcija `startActvitiy()` je tipa `void` iz klase `Context` i prima `Intent`.

Neka imamo aplikaciju sa dvije aktivnosti: `MojaAktivnost1` i `MojaAktivnost2`. Definiramo ih prvo u manifestu na sljedeći način:

Primjer 3. AndroidManifest.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/
    android"
3     package="com.mathos.zavrsni"
4     android:versionCode="1"
5     android:versionName="1.0" >
6
7     <uses-sdk
8         android:minSdkVersion="7"
9         android:targetSdkVersion="15" />
10
11    <application
12        android:icon="@drawable/ic_launcher"
13        android:label="@string/app_name" >
14        <activity
15            android:name=".MojaAktivnost1"
16            android:label="@string/
                title_activity_moja_aktivnost1" >
17            <intent-filter>
```

```
18                  <action android:name="android.intent.action.MAIN
                        " />
19
20                  <category android:name="android.intent.category.
                        LAUNCHER" />
21              </intent-filter>
22          </activity>
23          <activity
24              android:name=".MojaAktivnost2" >
25              <intent-filter>
26                  <action android:name="android.intent.action.VIEW
                        " />
27
28                  <category android:name="android.intent.category.
                        DEFAULT" />
29              </intent-filter>
30          </activity>
31      </application>
32
33  </manifest>
```

Sada da bismo iz prve aktivnosti `MojaAktivnost1` (koja je postavljena kao `MAIN` što znači da je to prva, odnosno glavna aktivnost koja se pokreće) pozvali aktivnost `MojaAktivnost2` prvo kreiramo `Intent`, te ga pokrenemo naredbom `startActivity()`:

Primjer 4. MojaAktivnost1.java

```
1 Intent mojIntent = new Intent(MojaAktivnost1.this,
      MojaAktivnost2.class);
2 MojaAktivnost1.this.startActivity(mojIntent);
```

Dakle, prvo smo napravili `mojIntent` koji nam govori na koju aktivnost ćemo prijeći. U drugoj liniji koda taj `Intent` izvršavamo naredbom `startActivity(Intent)`.

Aktivnost možemo pozvati i implicitno, zadavanjem uvjeta odnosno atributa koje ispunjava aktivnost koju želimo pokrenuti. Ovakvo pozivanje aktivnosti se koristi kada npr. želimo poslati mail i onda pozovemo sve one aktivnosti koje podržavaju slanje maila. Time dozvoljavamo korisniku da sam odabere koju će aplikaciju odnosno aktivnost koristiti. Također preko Intenta prenosimo i potrebne podatke. Glavna aktivnost naime, ne prima nikakve ulazne podatke.

Primjer implicitnog pozivanja aktivnosti:

Primjer 5. MojaAktivnost1.java

```
1 Intent mojIntent = new Intent();
2 mojIntent.setAction("android.intent.action.VIEW");
3 mojIntent.addCategory("android.intent.category.DEFAULT");
4 startActivity(mojIntent);
```

Ovakvim pozivanjem aktivnosti Android će nam ponuditi sve aktivnosti koje imaju ove postavke. Odabiranjem imena naše aplikacije, pokrenut će se `MojaAktivnost2`.

1.3 Servisi

- opisani podklasama klase `android.app.Service`
- `android.app.Service` je indirektna podklasa apstraktne klase `android.content.Context`

Servis je komponenta koja se može vrtiti u pozadini čak i kada korisnik nije u interakciji sa našom aplikacijom. Servis bismo trebali napraviti samo kada nam je zaista potreban. Ako trebamo napraviti nekakav pozadinski posao ali samo za vrijeme korištenja aplikacije, to se treba napraviti u zasebnom threadu, bez stvaranja posebnog servisa.

Da bismo napravili servis, moramo stvoriti podklasu klase `Service`, tj. moramo napravit override metoda koje upravljaju osnovnim aspektima servisa. Najbitnije metode za koje trebamo napraviti override su:

- `onStartCommand()` - Sistem poziva ovu metodu kada neka komponenta, uglavnom aktivnost, zatraži pokretanje servisa tj. pozove naredbu `startService()`. Jednom kada se ova metoda izvrši, servis je započet i može se vrtiti u pozadini beskonačno dugo. Naša je dužnost i zaustaviti servis, metodom `stopSelf()` ili `stopService()`.
- `onBind()` - Ova metoda je pozvana kada se neka druga komponenta želi povezati na servis koristeći naredbu `bindService()`. Ovu metodu moramo uvijek implementirati, ali ukoliko ne želimo dozvoliti povezivanje (binding) moramo vratiti `null`.
- `onCreate()` - Poziva se kada je servis prvi puta stvoren, da se provedu jednokratne procedure i inicijalizacije.
- `onDestroy()` - Poziva se pri gašenju servisa kako bismo mogli npr. obrisatii stvorene threadove, registre i sl.

Svaki servis moramo navesti u manifestu na sljedeći način:

Primjer 6. Android Manifest.xml

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".MojServis" />
      ...
  </application>
</manifest>
```

Osnovni izgled servisa sa funkcijama koje smo naveli:

Primjer 7. MojServis.java

```
1  import android.app.Service;
2  import android.content.Intent;
3  import android.os.IBinder;
4
5  public class MojServis extends Service{
6    @Override
7    public void onCreate() {
8      super.onCreate();
9    }
10
11   @Override
12   public void onDestroy() {
13     super.onDestroy();
14   }
15
16   @Override
17   public IBinder onBind(Intent intent) {
18     return null;
19   }
20
21   @Override
22   public int onStartCommand(Intent intent, int flags, int
       startId) {
23     return super.onStartCommand(intent, flags, startId);
24   }
25 }
```

Već smo rekli da servisi dolaze u dva oblika: **lokalni(započeti)** i **udaljeni(vezani)**.

Lokalne servise započinjemo naredbom `startService()`. Jednom kada se pokrenu, traju dok ih eksplicitno ne zatvorimo ili dok se ne zatvore sami, čak i nakon što aktivnost koja ih je stvorila više ne postoji.

Servis je udaljen ako se na njega spoji neka komponenta naredbom `bindService()`. Udaljeni servis pruža klijent-server sučelje, koje omogućava komponentama da šalju zahtjeve, primaju rezultate, vrše interprocesorsku komunikaciju itd. Udaljeni servis traje dok je neka komponenta povezana na njega (njih može biti i više). Kada više ni jedna komponenta nije spojena na taj servis, on se uništava.

2 Instalacija razvojnog okruženja

2.1 Java Development Kit

Za razvoj aplikacija prvo ćemo trebati Javu odnosno JDK (Java Development Kit). Preuzmite posljednju verziju koja je dostupna: `http://www.oracle.com/technetwork/java/javase/downloads/`. U ovome radu korišten je Eclipse 4.2.0.

2.2 Eclipse

Nakon instalacije JDK alata instalirat ćemo Eclipse, najčešće korišteno i preporučeno razvojno sučelje za Android. Posljednju verziju možete pronaći ovdje: `http://www.eclipse.org/downloads/`.

Preporučeno je preuzeti Eclipse Classic ili Eclipse IDE for Java Developers. Eclipse nema executable instalaciju već skinemo glavnu datoteku u kojoj se nalazi. Tu datoteku treba prekopirati tamo gdje želimo da Eclipse bude.

Kada pokrenete Eclipse pitat će vas koji workspace odnosno radni prostor želite koristiti. To je proizvoljna datoteka u kojoj će biti svi vaši projekti spremljeni. Napravite svoju mapu proizvoljno ili koristite ponuđenu mapu.

2.3 Android SDK

Sljedeće što nam je potrebno je Android SDK (Software Development Kit). To je softver za razvoj Android aplikacija, nepovezan s Eclipseom. Njega ćemo povezati u sljedećem koraku preko ADT plugina, i pokretat iz samog Eclipsea.

Koraci za instalaciju:

- Preuzmite odgovarajući SDK (posljednja verzija): `http://developer.android.com/sdk/index.html`.
- Ukoliko koristite Windowse možete automatski instalirati SDK, ili alternativno napraviti extract u bilo koju mapu, bitno je da zapamtite gdje ste stavili tu mapu.

2.4 Android Development Tools Plugin

ADT Plugin nam je potreban da povežemo Eclipse i Android SDK. Koraci za instalaciju su sljedeći:

- Otvorite Eclipse i odaberite Help → Install New Software...

- Kliknite Add u gornjem desnom kutu.
- U ćelije koje su se otvorile upišite proizvoljno ime, npr. "ADT Plugin" i sljedeći url: `https://dl-ssl.google.com/android/eclipse/`.
- Pritisnite OK.
- Nadalje, označite Developer Tools → Next. Sada će te vidjeti popis svih alata koji će se skinuti. Prihvatite uvjete korištenja i stisnite OK
- Kada instalacija završi resetirajte Eclipse.

2.5 Podešavanje ADT Plugina

Moramo dodati lokaciju na koju smo spremili Android SDK. Koraci su sljedeći:

- U Eclipseu: Window → Preferences (Eclipse → Preferences za Mac OS X).
- Iz lijevog izbornika izaberite Android.
- Pod SDK Location kliknite Browse... i pronađite mapu gdje se nalazi SDK (moguće je i da će Eclipse sam već pronaći mapu).
- Apply → Ok.

2.6 Android SDK Manager

Moramo instalirati verzije koje želimo koristiti pri razvoju Android aplikacija. Trenutne Android verzije:

Distribucija	API razina	Zastupljenost
4.0.x *Ice Cream Sandwich*	14-15	4.9%
3.x.x *Honeycomb*	11-13	3.3%
2.3.x *Gingerbread*	9-10	64.4%
2.2 *Froyo*	8	20.9%
2.0, 2.1 *Eclair*	7	5.5%
1.6 *Donut*	4	0.7%
1.5 *Cupcake*	3	0.3%

Slika 2: Android verzije

U Eclipseu odaberite: Window → Android SDK Manager. Sada će nam se otvoriti prozor u kojem možemo upravljati instaliranim verzijama. Za potrebe ovog rada korišten je Android 2.1 odnosno API 7.

Pod mapom Android 2.1 (API 7) označite SDK Platform te instalirajte. Kasnije ukoliko ćete raditi na većim ili manjim verzijama uvijek možete ući u SDK Manager i odabrat željene pakete za instalaciju.

Veće verzije uvijek podržavaju manje. Npr na Android 4.0 sustavu moći ćete pokrenuti aplikaciju namjenjenu sustavu Android 2.1, ali ne nužno i obrnuto.

2.7 Postavljanje emulatora

Android SDK dolazi sa AVD(Android Virtual Device) managerom preko kojega možemo stvoriti virtualni Android uređaj na računalu odnosno emulirati stvarni uređaj. Emulator će se ponašati identično kao i pravi uređaj samo što će biti nešto sporiji i neke funkcije će biti ograničene, npr. pozivi, kamera i sl.

Postavljanje emulatora:

- U Eclipseu: Window → AVD Manager → New...
- Name: proizvoljno, Target: Android 2.1 – API 7, Size: 100 – 500Mib (nije obavezno).
- Ostalo ostaviti kako je namješteno te stisnuti Create AVD.
- Kada smo ga napravili odaberemo ga i kliknemo Start.
- Scale display to real size označimo i promjenimo veličinu pod Screen Size (in) na recimo 6, jer će nam biti prevelik ekran ako ostavimo normalno, ali to je po vašem izboru (pod Scale vidimo koliki je odnos veličine emulatora i pravog uređaja), kliknemo Launch.
- Nakon određenog vremena emulator je pokrenut i možete s njime raditi.

Sada je emulator spreman i kada pokrenemo Android projekt on će se automatski instalirat na stvaran uređaj i pokrenuti.

2.8 Izrada projekta (Eclipse)

Nakon što smo instalirali razvojno okruženje otvorimo Eclipse i odaberemo workspace odnosno radnu datoteku. Ovdje ćemo izrađivati aplikacije. Jedan projekt predstavlja jednu aplikaciju.

Postupak izrade projekta:

- File → New → Project te zatim odaberemo Android Project.
- Sada odaberemo ime aplikacije, ime projekta i ime paketa. Npr. "Odjel za matematiku", "Mathos", `com.mathos.app`.
- Pod Build SDK biramo ciljanu verziju, a pod Minimum Required SDK najmanju verziju koju ćemo podržavati.
- Zatim pritiskamo Next do zadnjeg prozora koji nam nudi ime prve aktivnosti koju želimo stvoriti i ime layouta koji će ona imati.
- Klikom na Finish stvorili smo projekt.

Sada ako proširimo datoteku našeg projekta vidjet ćemo sljedeće datoteke koje sadržava:

- **src** - Sadrži izvorni kod programa pod imenom paketa. Tu će nam biti svi .java fileovi.
- **gen** - Sadrži automatski generiran Java kod i taj dio se ne dira ručno.
- **assets** - Ovdje možemo spremiti razne file-ove kojima pristupamo direktno u aplikaciji, npr. neke svoje tablice, dokumente, itd.
- **bin** - Sadrži binarni kod aplikacije, potreban za instalaciju na uređaj.
- **res** - Sadrži sve resurse koji su nam potrebni u aplikaciji: xml fileove, slike, stringove, zvukove, ikone, itd.

3 Grafičko korisničko sučelje

Grafičko sučelje na Android uređajima izgrađeno je od `View` i `ViewGroup` objekata. `View` je objekt koji nešto grafički prezentira na ekranu i sa kojim korisnik može doći u interakciju. `ViewGroup` je objekt koji sadrži `View` objekte, kako bi se definirao cjelokupan izgled i položaj na ekranu.
`ViewGroup` objekti koje ćemo detaljnije promatrati:

- `LinearLayout`
- `RelativeLayout`
- `ListView`

Još neki bitniji `ViewGroup` objekti:

- `WebView` - služi za prikaz web stranica u aktivnosti
- `ScrollView` - prikaz koji možemo pomicati(scrollati) ukoliko tekst izađe van okvira
- `GridView` - koristi se za prikaz npr. više slika u galeriji, sadrži više elemenata dijagonalno i vertikalno, kao u tablici

`View` objekti koje ćemo detaljnije promatrati:

- `Button`
- `CheckBox`
- `TextView`

Još neki bitniji `View` objekti:

- `EditText` - služi za unos teksta od strane korisnika
- `RadioButton` - nudi jedan mogući od više odabira neke opcije
- `Spinner` - drop-down meni sa opcijama

Sada ćemo proći kroz nekoliko osnovnih atributa koje možemo pridavati `ViewGroup` i `View` objektima. Svi atributi pišu se u samome tagu nakon imena taga, kao što ćemo vidjeti kasnije u kodu.

- `android:id` - ovime pridružujemo nekom elementu identifikacijsku oznaku kako bismo kasnije u kodu mogli referirati na taj objekt, npr. `android:id="@+id/mybutton"`, gdje `@+id` znači da dodajemo taj id, ako nebi bilo plusa onda bi referirali na neki postojeći id

- `android:layout_width` - širina određenog layouta, preporuča se upotreba `match_parent` - širina objekta u kojem je sadržana, `wrap_content` - širina samo oko sadržaja koji je sadržan u objektu, te ako se mora ručno pisati dužina preporuča se `dp`, odnosno density-pixel koji će se mjenjat ovisno o gustoći ekrana, tako da se održi postojanost na različitim veličinama ekrana
- `android:layout_height` - analogno prethodnoj točki
- `android:text` - ukoliko objekt može u sebi sadržavati tekst njega definiramo na ovaj način, ali pozivanjem na string unutar datoteke `\res\values\strings.xml` zbog lakog kasnijeg mjenjanja i prevođenja, npr. `android:text="@string/nekistring"`

Ostale atribute ćemo upoznati kroz daljnje primjere.

3.1 Layouti

Layout predstavlja vizualnu strukturu aktivnosti. Svi elementi se raspoređuju u layoute, koji određuju kako i gdje će koji objekt biti smješten.

Elementi layouta mogu biti deklarirani u XML datoteci samog layouta, ili programatski unutar java koda. U XML datoteci bi trebali biti definirani elementi koji se neće mjenjati u aktivnostii, ili nekakav početan izgled, a programatski oni djelovi koji će se mjenjat i koji ovise o nekim parametrima ili korisnikovoj interakciji.

XML kod layouta koji se zove npr. `mojlayout` nalazi se u: `\res\layout\mojlayout.xml`

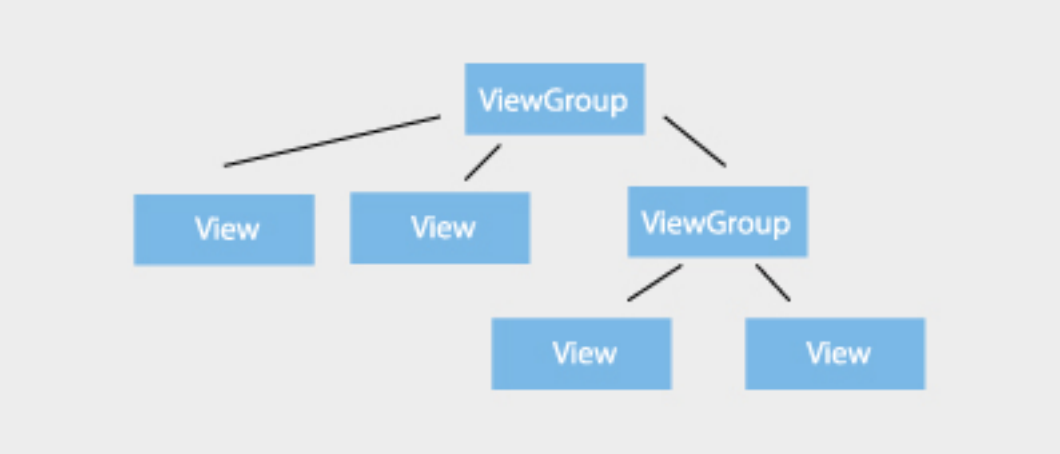
Kada aktivnosti pridružujemo određeni layout to radimo naredbom `setContentView()` koja prima `id` naseg layouta odnosno ime layouta sa `R.layout` ispred imena. Iniciranje layouta se odrađuje u metodi `onCreate()`, tj. prilikom nastanka aktivnosti, osim ako nam nije potrebno drukčije, npr. želimo kada se vratimo u aplikaciju postaviti nekakav drukčiji layout, onda ćemo pozvat `setContentView()` u npr. `onResume()` metodi.

Primjer 8. MojaAktivnost.java

```
1 public class MojaAktivnost extends Activity {
2   @Override
3   protected void onCreate(Bundle savedInstanceState) {
4     super.onCreate(savedInstanceState);
5     setContentView(R.layout.mojlayout); //postavljamo mojlayout
          kao layout aktivnosti
6 }}
```

3.2 `ViewGroup` objekti

Layout čini `ViewGroup` objekt unutar kojega su posloženi svi drugi `View` `ViewGroup` objekti. Primjer hijerarhije jednog layouta:



Slika 3: Hijerarhija layouta

3.2.1 Linear Layout

`LinearLayout` je `ViewGroup` objekt u kojem su `View` objekti posloženi u jednome smjeru, vertikalno ili horizontalno.

Slika 4: Linear Layout

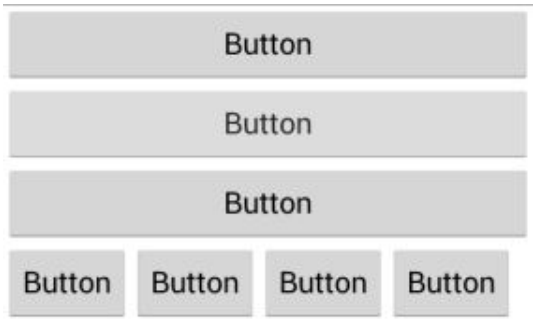
Linear Layout koristimo za jednostavan jednosmjerni izgled. Također `LinearLayout` objekt može sadržavati drugi `LinearLayout` objekt druge orjentacije, pa time možemo dobiti kompleksniji izgled.

Primjer 9. mojlayout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button" />

    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button" />

    <Button
        android:id="@+id/button3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button" />

    <LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/
            android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal" >

        <Button
            android:id="@+id/button4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/button" />

        <Button
            android:id="@+id/button5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/button" />

        <Button
```

```
45              android:id="@+id/button6"
46              android:layout_width="wrap_content"
47              android:layout_height="wrap_content"
48              android:text="@string/button" />
49
50          <Button
51              android:id="@+id/button7"
52              android:layout_width="wrap_content"
53              android:layout_height="wrap_content"
54              android:text="@string/button" />
55      </LinearLayout>
56
57  </LinearLayout>
```

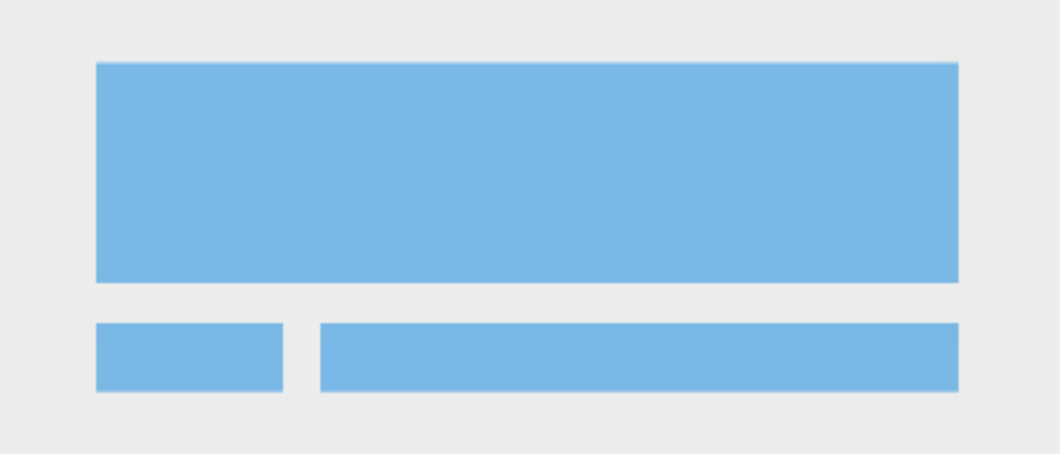
Znači imamo dva Linear Layouta s time da prvi sadrži tri `Button` objekta, i jedan `LinearLayout` objekt koji sadrži još četiri `Button` objekta i orjentiran je horizontalno za razliku od prvog koji je orjentiran vertikalno. Dobivamo ovakav izgled:



Slika 5: Složeniji Linear Layout

3.2.2 Relative Layout

Relative Layout je preporučen ispred Linear Layouta, zbog svoje mogućnosti adaptiranja na razne uređaje različitih veličina ekrana.



Slika 6: Relative Layout

Kod Relative Layout objekta, elemente možemo orjentirati prema drugim elementima, pa npr. jednu tipku možemo staviti desno od neke druge tipke a ispod neke slike, što kod Linear Layouta ne možemo, nego bismo morali uvoditi više Linear Layouta sa različitim orjentacijama što bi zakompliciralo stvari.

Pozicioniramo elemente sa sljedecim atributima: `android:layout_toRightOf`, `android:layout_toLeftOf`, `android:layout_below`, `android:layout_above`, koji primaju id elementa od kojeg je naš element "desno od", "lijevo od", "ispod", "iznad".

Napravimo sada isti izgled kao i u *primjeru 9*, ali koristeći `RelativeLayout`:

Primjer 10. mojlayout.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/
    res/android"
    xmlns:tools="http://schemas.android.com/tools">
    <Button
        android:id="@+id/button1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button" />

    <Button
        android:id="@+id/button2"
```

```
11          android:layout_width="match_parent"
12          android:layout_height="wrap_content"
13          android:layout_below="@id/button1"
14          android:text="@string/button" />
15
16      <Button
17          android:id="@+id/button3"
18          android:layout_width="match_parent"
19          android:layout_height="wrap_content"
20          android:layout_below="@id/button2"
21          android:text="@string/button" />
22
23      <Button
24          android:id="@+id/button4"
25          android:layout_width="wrap_content"
26          android:layout_height="wrap_content"
27          android:layout_below="@id/button3"
28          android:text="@string/button" />
29
30      <Button
31          android:id="@+id/button5"
32          android:layout_width="wrap_content"
33          android:layout_height="wrap_content"
34          android:layout_below="@id/button3"
35          android:layout_toRightOf="@id/button4"
36          android:text="@string/button" />
37
38      <Button
39          android:id="@+id/button6"
40          android:layout_width="wrap_content"
41          android:layout_height="wrap_content"
42          android:layout_below="@id/button3"
43          android:layout_toRightOf="@id/button5"
44          android:text="@string/button" />
45
46      <Button
47          android:id="@+id/button7"
48          android:layout_width="wrap_content"
49          android:layout_height="wrap_content"
50          android:layout_below="@id/button3"
51          android:layout_toRightOf="@id/button6"
52          android:text="@string/button" />
53  </RelativeLayout>
```

Rezultat je *Slika 5*, a kao što vidimo korišten je samo jedan `RelativeLayout`.

3.2.3 List View

`ListView` koristimo kada imamo nekakvu listu koju želimo popuniti nekakvim vrijednostima. Prednost `ListView` objekta je što kroz `ArrayAdapter` koji ćemo pokazati kroz primjer, možemo listu vrlo popuniti npr. `String` varijablama iz polja stringova.

U sljedećem primjeru napraviti ćemo listu koju ćemo popuniti imenima nekih profesora Odjela za matematiku:

Primjer 11. MojaAktivnost.java

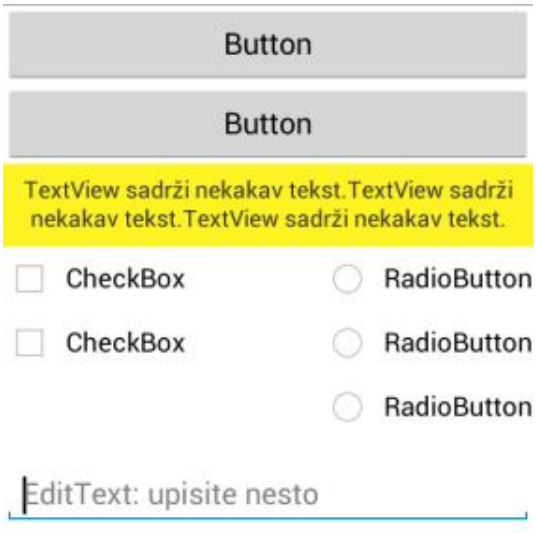
```
1  import android.os.Bundle;
2  import android.app.Activity;
3  import android.widget.ArrayAdapter;
4  import android.widget.ListView;
5
6  public class MojaAktivnost extends Activity {
7
8    public String[] nastavnici = {"prof.dr.sc. Mirta Bensic",
9        "doc.dr.sc. Kresimir Burazin",
10       "Josip Cvenic",
11       "prof.dr.sc. Dragan Jukic",
12       "prof.dr.sc. Antoaneta Klobcar",
13       "prof.dr.sc. Zdenka Kolar-Begovic",
14       "doc.dr.sc. Darija Markovic",
15       "doc.dr.sc. Tomislav Marosevic",
16       "doc.dr.sc. Ivan Matic",
17       "doc.dr.sc. Domagoj Matijevic",
18       "doc.dr.sc. Mihaela Ribicic Penava",
19       "prof.dr.sc. Kristian Sabo",
20       "prof.dr.sc. Rudolf Scitovski",
21       "doc.dr.sc. Nenad Suvak",
22       "doc.dr.sc. Zoran Tomljanovic",
23       "prof.dr.sc. Ninoslav Truhar",
24       "prof.dr.sc. Sime Ungar"};
25
26     @Override
27     public void onCreate(Bundle savedInstanceState) {
28         super.onCreate(savedInstanceState);
29         ArrayAdapter<String> mojAdapter = new ArrayAdapter<
               String>(this,
30                 android.R.layout.simple_list_item_1,nastavnici);
31         ListView mojListView = new ListView(this);
32         mojListView.setAdapter(mojAdapter);
33         setContentView(mojListView);
34     }}
```

Naš `ArrayAdapter` prvo je primio `Context` naše aplikacije, zatim layout koji će se primjeniti na svaki tekstualni red (mi smo ovdje koristili Androidov već unaprijed napravljeni layout), te `String` koji treba pretvoriti u listu. Dobiveni izgled:

Slika 7: List View

3.3 `View` objekti

`View` objekti su oni sa kojima korisnik dolazi u interakciju. Pokazat ćemo kako definirati objekte u XML kodu, te kako ih zatim očitati unutar Java koda i postaviti metode koje će reagirati na korisnikovu interakciju sa tim objektom. Sljedeća slika pokazuje primjer kako izgledaju neki `View` objekti unutar aktivnosti:



Slika 8: `View` objekti

3.3.1 Button

Button odnosno tipka sastoji se od teksta ili ikone, ili i teksta i ikone, a reagira na korisnikov pritisak. Primjer definiranja jednog Buttona u XML kodu:

Primjer 12. mojlayout.xml

```
<Button
    android:id="@+id/posalji"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text" ... />
```

Sada ga definiramo u Java kodu i postavimo `View.OnClickListener`, tj. metodu iz klase `View` koja će se izvršiti kada korisnik klikne na tipku za koju smo listener postavili:

Primjer 13.MojaAktivnost.java

```
// Prvo ga inicijaliziramo
Button button = (Button) findViewById(R.id.posalji);
// Postavimo listener
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Uciniti nesto nakon klika na taj button
    }
});
```

3.3.2 Check Box

Check box nam daje mogućnost označavanja određenih opcija. Primjer definiranja `CheckBox` objekta u XML kodu:

Primjer 14. mojlayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

     <CheckBox android:id="@+id/opcija_1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/prvaopcija"
         android:onClick="onCheckboxClicked" />

     <CheckBox android:id="@+id/opcija_2"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="@string/drugaopcija"
         android:onClick="onCheckboxClicked" />

</LinearLayout>
```

Atribut `android:onClick="nekoIme"` je atribut koji je uveden od Androida 2.0 koji nam dozvoljava da postavimo listener tako da samo napišemo funkciju `public void nekoIme(View v)` gdje v prestavlja `View` objekt na koji se listener odnosi. Isto ovo smo mogli napraviti i za `Button`.

Primjer 15. MojaAktivnost.java

```
public void onCheckboxClicked(View v) {
    // Provjerimo je li view v odnosno nas checkbox oznacen
    boolean checked = ((CheckBox) v).isChecked();

    //  Provjerimo koji sve checkboxovi su odabrani
    switch(v.getId()) {
        case R.id.opcija_1:
            if (checked)
                // Uciniti nesto ako je opcija_1 odabrana
            else
                // Uciniti nesto ako opcija_1 nije odabrana
            break;
        case R.id.opcija_2:
            if (checked)
                // Uciniti nesto ako je opcija_2 odabrana
            else
                // Uciniti nesto ako opcija_2 nije odabrana
            break;
    }
}
```

3.3.3 Text View

`TextView` objekt koristimo za postavljanje nekakvog teksta na ekran. Također možemo mu zadati font, boju pozadine, veličinu slova, itd.

Primjer 15. mojlayout.xml

```
<TextView
        android:id="@+id/textView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#DEDEDE"
        android:padding="10dp"
        android:gravity="center"
        android:text="@string/tekst" ../>
```

4 Napredniji primjeri

U ovom poglavlju pokazat ćemo neke naprednije primjere iz aplikacije koja prati promjene na službenim stranicama Odjela za matematiku u Osijeku.

4.1 Prvi primjer: download HTML koda i pretvorba u MD5 checksum

U ovom primjeru cilj nam je preuzeti HTML kod određene stranice fakulteta, te ga pretvoriti u jedinstveni *MD5 checksum* kod kako bismo ga lakše mogli spremiti u bazu podataka bez da gubimo jedinstvenost.

Primjer 16. DownloadActivity.java

```
1  import java.io.BufferedReader;
2  import java.io.IOException;
3  import java.io.InputStream;
4  import java.io.InputStreamReader;
5  import java.math.BigInteger;
6  import java.security.MessageDigest;
7  import java.security.NoSuchAlgorithmException;
8
9  import org.apache.http.HttpResponse;
10 import org.apache.http.client.ClientProtocolException;
11 import org.apache.http.client.HttpClient;
12 import org.apache.http.client.methods.HttpGet;
13 import org.apache.http.impl.client.DefaultHttpClient;
14
15 import android.os.Bundle;
16 import android.app.Activity;
17
18 public class DownloadActivity extends Activity {
19   @Override
20   public void onCreate(Bundle savedInstanceState) {
21     super.onCreate(savedInstanceState);
22
23     // Adresa stranice ciji html kod zelimo
24     String url = "http://www.mathos.unios.hr/uvis/obavijesti.
         html";
25
26     // Interface za http klijenta
27     HttpClient client = new DefaultHttpClient();
28
29     // GET metodom preuzimamo sve sa zadane adrese
30     HttpGet request = new HttpGet(url);
31
32     // Postavimo http odgovor za sada na null
```

```
33      HttpResponse response = null;
34
35      // Provedemo request preko klijenta i spremimo u response
36      try {
37        response = client.execute(request);
38      } catch (ClientProtocolException e) {
39        // Hvatanje iznimki
40        e.printStackTrace();
41      } catch (IOException e) {
42        // Hvatanje iznimki
43        e.printStackTrace();
44      }
45
46      // InputStream koristimo za primanje podataka u obliku
            byteova
47      InputStream in = null;
48      try {
49        // Preuzimamo odgovor i upisujemo ga u InputStream
50        in = response.getEntity().getContent();
51      } catch (IllegalStateException e) {
52        // Hvatanje iznimki
53        e.printStackTrace();
54      } catch (IOException e) {
55        // Hvatanje iznimki
56        e.printStackTrace();
57      }
58
59      // Deklariramo citac u koji pretvara InputStream iz byte u
            char oblik
60      // Za prevodjenje bitova iz input streama koristimo
            InputStreamReader
61      BufferedReader reader = new BufferedReader(new
            InputStreamReader(in));
62
63      // Deklariramo StringBuilder koji ce spojiti pretvorene char
            varijable u
64      // Stringove
65      StringBuilder str = new StringBuilder();
66      String line = null;
67      try {
68        // Sve dok imamo teksta, appendamo (priljepljujemo) ga na
              String
69        // line
70        while ((line = reader.readLine()) != null) {
71          str.append(line);
72        }
73      } catch (IOException e) {
74        // Hvatanje iznimki
75        e.printStackTrace();
```

```
76      }
77      try {
78        in.close();
79      } catch (IOException e) {
80        // Hvatanje iznimki
81        e.printStackTrace();
82      }
83
84      // Deklariramo String html u kojeg sve prepisemo
85      String html = str.toString();
86
87      try {
88        // String pretvoren funkcijom stringToMd5
89        // koju smo deklarirali na kraju
90        String md5 = stringToMd5(html);
91      } catch (NoSuchAlgorithmException e) {
92        e.printStackTrace();
93      }
94
95    }
96
97    // Funkcija za pretvaranje Stringa u novi string
98    // koji sadrzava njegov MD5 Checksum
99    public String stringToMd5(String s) throws
          NoSuchAlgorithmException {
100     // MessageDigest - klasa koja koristi jednosmjernu hash
            funkciju
101     MessageDigest md5 = MessageDigest.getInstance("MD5");
102     // U MessageDigest instancu postavimo bajtove, pocetni bajt,
            te duzinu stringa
103     md5.update(s.getBytes(), 0, s.length());
104     // Izvrsimo pretvorbu
105     String md5String = new BigInteger(1, md5.digest()).toString
            (16);
106     // Vratimo pretvoreni string
107     return md5String;
108   }
109 }
```

4.2 Drugi primjer: učitavanje vanjske baze podataka

U ovom primjeru cilj nam je učitati vanjsku bazu podataka i pri instalaciji aplikacije prekopirati ju u memoriju i koristiti ju u aplikaciji. Bazu cemo snimiti u \assets folder.

Primjer 17. DatabaseHandler.java

```
1  import java.io.FileOutputStream;
2  import java.io.IOException;
3  import java.io.InputStream;
4  import java.io.OutputStream;
5
6  import android.content.ContentValues;
7  import android.content.Context;
8  import android.database.Cursor;
9  import android.database.SQLException;
10 import android.database.sqlite.SQLiteDatabase;
11 import android.database.sqlite.SQLiteException;
12 import android.database.sqlite.SQLiteOpenHelper;
13
14 public class DatabaseHandler extends SQLiteOpenHelper {
15
16   // Deklariramo ime baze i gdje ju zelimo snimiti
17   private static String DB_PATH = "/data/data/com.mathos.app/
         databases/";
18   private static String DB_NAME = "mathos";
19
20   // Ucitamo objekt klase SQLiteDatabase
21   private SQLiteDatabase myDataBase;
22   private final Context myContext;
23
24   // Konstruktor
25   public DatabaseHandler(Context context) {
26     super(context, DB_NAME, null, 1);
27     this.myContext = context;
28   }
29
30   // Metoda u kojoj pravimo bazu
31   public void createDataBase() throws IOException {
32     // Provjerimo da li baza vec postoji
33     boolean dbExists = checkDataBase();
34
35     // Ukoliko baza postoji ne radimo nista
36     if (dbExists) {
37
38     } else {
39       // Ako baza postoji uzimamo objekt SQLiteDatabase klase
40       // i pozivamo metodu copyDataBase() koja kopira nasu
```

```
41        // vanjsku bazu u novi objekt odnosno unutrasnju bazu
42        this.getReadableDatabase();
43        try {
44          copyDataBase();
45        } catch (IOException e) {
46          throw new Error("Error copying database");
47        }
48      }
49    }
50
51    // checkDataBase() metoda
52    private boolean checkDataBase() {
53      SQLiteDatabase checkDB = null;
54      try {
55        String myPath = DB_PATH + DB_NAME;
56        checkDB = SQLiteDatabase.openDatabase(myPath, null,
57            SQLiteDatabase.OPEN_READWRITE);
58      } catch (SQLiteException e) {
59
60      }
61
62      if (checkDB != null) {
63        checkDB.close();
64      }
65
66      return checkDB != null ? true : false;
67    }
68
69    // Metoda za vrsenje kopiranja baze
70    private void copyDataBase() throws IOException {
71      InputStream myInput = myContext.getAssets().open(DB_NAME);
72      String outFileName = DB_PATH + DB_NAME;
73      OutputStream myOutput = new FileOutputStream(outFileName);
74
75      byte[] buffer = new byte[1024];
76      int length;
77      while ((length = myInput.read(buffer)) > 0) {
78        myOutput.write(buffer, 0, length);
79      }
80      myOutput.flush();
81      myOutput.close();
82      myInput.close();
83    }
84
85    // Metoda kojom otvaramo bazu u koju mozemo pisati
86    // i iz koje mozemo citati
87    public void openDataBase() throws SQLException {
88
89      String myPath = DB_PATH + DB_NAME;
```

```
90      myDataBase = SQLiteDatabase.openDatabase(myPath, null,
91          SQLiteDatabase.OPEN_READWRITE);
92    }
93
94    @Override
95    public synchronized void close() {
96      if (myDataBase != null)
97        myDataBase.close();
98
99      super.close();
100   }
101
102   @Override
103   public void onCreate(SQLiteDatabase db) {
104
105   }
106
107   @Override
108   public void onUpgrade(SQLiteDatabase db, int oldVersion, int
          newVersion) {
109
110   }
111 }
```

Literatura

[1] D. Smith: Android Recipes: A Problem-Solution Approach, First Edition, Apress, 2011.

[2] M. Gragenta: Learning Android, First Edition, O' Reilly, 2011.

[3] http://http://developer.android.com/guide/components/index.html