1. Android

1.1. Uvod

Android nam je poznat kao operacijski sustav (OS) za mobilne uređaje baziran na modificiranoj verziji Linuxa, od 2005. u vlasništvu Googlea. Ali Android je zapravo mnogo više od toga: skup softvera otvorenog koda (engl. *open source*) koji uključuje OS, međusoftver (engl. *middleware*) i ključne aplikacije, zajedno sa API bibliotekama za pisanje mobilnih aplikacija koje mogu oblikovati izgled i funkciju mobilnih uređaja. Google definira Android kao **prvu potpuno otvorenu platformu za mobilne uređaje**, sav softver potreban za mobilni uređaj, ali bez vlasničkih (engl. *proprietary*) zapreka koje bi kočile inovacije. U skladu s time, Android je otvoren i besplatan, objavljen uz *open source Apache licence* i svi ga mogu slobodno preuzimati (engl. *free download*) i modificirati.[1]

Ta otvorenost je velika prednost Androida; prihvatile su ga velike kompanije takozvanog *Open Handset Alliancea* (udruženje od preko 30 tehnoloških kompanija uključujući Motorolu, HTC, TMobile i ostale koje razvijaju uređaje koji koriste Android).

Druga velika prednost Androida je da se **sve aplikacije tretiraju jednako**, dok Windows Mobile i Appleov iPhone, koji su bazirani na vlasničkom (engl *proprietary*) softveru, daju veći prioritet svojima ugrađenim (engl. *native*) aplikacijama i ograničavaju komunikaciju između ugrađenih i aplikacija instaliranih iz drugih izvora. Kod Androida su sve aplikacije ravnopravne i sve se mogu izbrisati, zamijeniti i nadograditi. Također, srušene su granice među aplikacijama – aplikacije lako, bez ograničenja, komuniciraju međusobno i razmjenjuju podatke.

Iduća velika prednost je što se jednom napisana aplikacija za Android može pokrenuti na svim uređajima koji koriste Android (pametni telefoni, tableti, čitači *e-book*, MP4 playeri, Internet TV...) bez razmišljanja o hardveru. To je postignuto korištenjem **Dalvik** i **ART virtualnih mašina**.

Sve to doprinijelo je **brzom i laganom** razvoju aplikacija (prilično složena aplikacija može se napisati za 30-ak minuta), razvoju velike zajednice razvojnih programera širom svijeta (što omogućava pronalaženje odgovora na neka pitanja, ali i olakšava pronalaženje suradnika). Uz to, Android ima i **odličnu dokumentaciju i nema troškova za razvoj i distribuciju aplikacija** (sve potrebno se besplatno preuz-

1 Bitno je napomenuti da aplikacije razvijene za Android ne moraju biti otvorenog koda, ali bilo bi lijepo da jesu.

me s Interneta). Android tržište aplikacijama (Google Play Store) je *online* trgovina aplikacijama koja je instalirana na svim Android uređajima i omogućuje jednostavno preuzimanje aplikacija izravno na uređaj (tu se nalaze i besplatne aplikacije kao i one koje se plaćaju iako je pristup plaćenim aplikacijama u nekim zemljama zabranjen ili restringiran[2]).

Android podržava i koristi:

- standarde za komunikaciju i interakciju Bluetooth, SMS, MMS, dodirni ekran (engl. *multitouch*) koji može prepoznati više točki dodira odjednom, Flash...
- digitalne formate za multimedijske sadržaje MP3, MP4, WAV, MIDI, JPEG, PNG, GIF, BMP...
- fotoaparat/kamera, GPS, digitalni kompas...
- WebKit preglednik
- SQLite relacijsku bazu podataka (brza i efikasna, što je esencijalno za uređaje čiji su kapaciteti limitirani njihovom kompaktnom prirodom).

1.2. Dalvik virtualni stroj

Dalvik virtualni stroj (engl. Dalvik *virtual machine*) je specijalizirani virtualni stroj dizajniran za Android i optimiziran je za mobilne uređaje s limitiranom memorijom i procesorskim kapacitetom (CPU-om). Aplikacije za android pišu se u programskom jeziku Java, ali se ne izvršavaju u tradicionalnom Java virtualnom stroju (engl. *stack based virtual machine*) već u Dalvik virtualnom stroju (engl. *register based virtual machine*). Prevedene (engl. *compiled*) Java klase se transformiraju u Dalvik izvršne daoteke (engl. *executables)* .dex i izvršavaju kao posebni proces u vlastitoj instanci Dalvika (to omogućava i da ako jedna aplikacija prestane raditi zbog neke greške, to ne utječe na druge aplikacije). Sav pristup hardveru i nižim (engl. *low level*) sistemskim funkcionalnostima (memorija, višedretvenost, sigurnost...) ide preko Dalvika i osigurava da programeri ne moraju brinuti o tome za koji uređaj razvijaju aplikaciju.

Tvorac Dalvik virtualnog stroja je Dan Bornstein koji mu je dao ime po ribarskom selu Dalvik na Islandu odakle su mu predci.

1.3. ART

ART (engl. *Android Run Time*) je nasljednik Dalvik virtualnog stroja za Android. ART, poput Dalvika, izvršava .dex datoteke, stoga kôd koji je napisan za Dalvik može raditi pod ART-om, ali neke tehnologije koje su radile za Dalvik ne rade na ART-u. Prednosti ART-a nad Dalvikom su: Ahead Of Time compilation (AOT) što ubrzava performanse neke aplikacije, poboljšani Garbage Collector (GC), te jasnije iskazivanje grešaka u aplikaciji.

2 Ima zemalja gdje se smije prodavati, ali ne i preuzimati i obratno.

ART polako zamjenjuje Dalvik, ali prijelaz još nije načinjen u potpunosti. Android inačice 5.0 još uvijek koriste Dalvik kao početnu postavku, ali omogućuju prelazak na ART. Inačica 6.0 također koristi Dalvik kao primarni virtualni stroj. Počevši s inačicom 7.0 ART dolazi u prvi plan.

1.4. Optimizirano upravljanje memorijom i procesima

Vlastiti *run-time* i virtualni stroj, kao i Android, koriste i Java i .NET. Za razliku od njih, Android Run Time također upravlja i životnim vijekom procesa. Android osigurava dobar rad aplikacije zaustavljanjem i terminiranjem procesa ako je potrebno, da bi se oslobodili resursi za aplikacije višeg prioriteta. Prioritet se određuje ovisno o aplikaciji s kojom je korisnik u interakciji. Sve aplikacije dakle trebaju biti pripremljene za naglo gašenje, ali i za lako ponovno pokretanje u istom stanju u kojem su bile prilikom gašenja.

1.5. Arhitektura Androida

Arhitektura Android sustava sastoji se od pet dijelova koji su smješteni u četiri sloja (slika 1.1[3]):

- Linux jezgra (Linux Kernel)
- biblioteke (Libraries), Android Run Time
- aplikacijski okvir (Application Framework)
- aplikacije (Applications).

Dakle, Dalvik i Android Run Time nalaze se na Linux jezgri koja upravlja nižim hardverskim interakcijama (upravljački programi (engl. *drivers*), upravljanje memorijom), a skup API-ja (Application Programming Interface, to su razne vrste biblioteka) daje pristup svim hardverskim komponentama. Aplikacije se izvršavaju u Android Run Time koristeći klase iz aplikacijskog okvira.

Napomena. Moguće je pisati i C/C++ aplikacije direktno u Linux OS-u, ali za to nema potrebe, osim ako želite npr. modificirati Dalvik. To neće biti naša tema, za više informacija zainteresirani se upućuju na: `https://groups.google.com/forum/#!forum/android-internals`.

Napomena. Android je unificiran operacijski sustav za pametne telefone i tablete. Od inačice 3.0 ugrađene su neke specifične stvari za tablete, u verziji 4.0 to je preneseno i na pametne telefone, te su dodane nove opcije za obje vrste uređaja. Mi ćemo se baviti Androidom 4.0 i više (sve što se napravi za starije verzije možemo prenijeti na novije, ali ne i obratno).

3 Za izradu slika u ovoj knjizi korišten je besplatni softver Dia [15].

APPLICATIONS
- sve aplikacije koje dolaze s uređajem
- sve aplikacije koje preuzimamo
- sve aplikacije koje napišemo (npr. preglednik, *music player*, SMS *manager*...)

APPLICATION FRAMEWORK
- različiti aspekti Androida koje razvojni inženjeri (engl. *developer*) vide i mogu koristiti za svoje aplikacije (klase za stvaranje Android aplikacija)
- generička apstrakcija za hardverski pristup (npr. *window manager*, *package manager*, *activity manager*)

LIBRARIES
- biblioteke za ugrađene aplikacije (SQLite biblioteka, WebKit biblioteka, grafičke i media biblioteke)

ANDROID RUN TIME
- Android Libraries: biblioteke koje omogućuju developerima pisanje aplikacija
- Dalvik virtualna mašina

LINUX KERNEL
- svi upravljački programi za uređaje (*display*, kamera, audio...)
- upravljanje energijom, upravljanje procesima i memorijom

Slika 1.1. Arhitektura Android sustava prikazana po slojevima

2. Integrirana razvojna okolina (IDE)

Za razvoj Android aplikacija danas se koristi Android Studio koji u sebi već sadrži osnovne pakete za razvoj, a lako se može nadopuniti drugim paketima. Ranije je preporučeni IDE (Integrated Development Environment, okruženje za razvoj softvera) za razvoj bio Eclipse koji se i dalje može koristiti, ali je izgubio razvojnu podršku (engl. *update*) za Android. Projekti izrađeni u Eclipseu se lako mogu prenijeti u Android Studio.

Razvojna okolina instalira se na razvojnu platformu (korisnikovo računalo), a ne na ciljnu platformu (Android uređaj). Android Studio je podržan za Linux, Windows i Mac OS X operacijske sustave. Ovdje će biti opisan postupak instalacije na Linux sustavima.

2.1. *Build* sustav – Gradle

Prevođenje (engl. *compilation*) programa je proces prevođenja programskog koda u strojni kod (takozvani objektni kôd). Danas kompajleri uz prevođenje vrše i povezivanje (engl. *linking*), koje strojni kod pretvara u datoteku koja se može izvršiti na računalu. Gradnja (engl. *build*) projekta, uz prevođenje i povezivanje, uključuje i neke dodatne radnje, na primjer izradu programa za instalaciju (engl. *installer*). Gradnja još omogućuje da se provode testovi prevedenog koda, moguće je definirati korake koje treba učiniti prije ili poslije prevođenja nekog dijela projekta, odrediti gdje će se izvršivi program nalaziti i još mnogo toga. Procesom gradnje moguće je stvarati statičke i dinamičke pomoćne biblioteke koje se kasnije mogu samostalno koristiti. Razlika među njima je u tome što kad statičku biblioteku unesemo u neki kôd, ona se u cijelosti povezuje prilikom prevođenja, dok se kod dinamičke datoteke povezuje samo dio koji će se koristiti u izvršnom programu. U slučaju razvoja Android aplikacija, *build* sustav služi kako bi uzeo izvorne programske datoteke (.java ili .xml), na njih primijenio neke alate prevođenja i povezivanja (na primjer .java datoteku pretvorio u .dex), te grupirao sve te datoteke u jednu komprimiranu .apk datoteku s kojom Android sustav zna raditi.

Build sustav koji se koristi u Android Studiju naziva se Gradle. Ovaj sustav je razvijen promatrajući druge *build* sustave, te integriranjem njihovih najboljih karakteristika. Gradle je baziran na JVM (Java Virtual Machine) build sustavima što znači da se mogu pisati skripte za gradnju u programskom jeziku Java.

2.2. Usporedba okruženja Android Studio i Eclipse

Najveće razlike između okruženja Android Studio i Eclipse su u sljedećim segmentima:

- ***Build* alati:** Android Studio koristi sve popularniji Gradle sustav gradnje. Gradle gradi projekte pomoću Apache Ant i Apache Maven, ali uvodi i Groovy DSL (Domain-Specific Language) koji omogućuje skriptiranje gradnji što na kraju omogućuje automatizaciju postavljanja beta .apk datoteka na TestFlight u svrhu testiranja. S druge strane Eclipse koristi Apache Ant kao osnovni *build* sustav koji koristi vrlo robusni XML koji je poznat razvojnim programerima u Javi.
- **Napredno autozavršavanje koda i refaktorizacija:** Oba IDE-a koriste standardno autozavršavanje koda u Javi[1], ali u slučaju Android Studia, Google je unio dodatne stvari vezane za Android i njegovu refaktorizaciju[2].
- **Organizacija projekta:** Eclipse je projekte organizirao u radni prostor (engl. *workspace*) koji se odredio pri pokretanju. Takav način rada je ograničavajući jer dopušta da se koriste samo projekti iz određenog radnog prostora. U slučaju potrebe za radom s projektima iz drugog radnog prostora, moramo ugasiti Eclipse i prilikom ponovnog pokretanja prijeći u drugi radni prostor. Android Studio taj problem rješava korištenjem modula. Moduli mogu biti svaki dio nekog projekta, na primjer jedan modul je aplikacija na kojoj se upravo radi, drugi modul je neka biblioteka koja je preuzeta s Interneta i želi se uključiti u trenutačni projekt, treći modul je integrirani SDK... Svaki od modula može imati svoje Gradle *build* datoteke i zahtijevati svoje pakete za ovisnost.
- **Performanse i stabilnost IDE:** Eclipse je IDE koji je razvijen u svrhu razvoja aplikacija u Javi. Zahtijeva puno RAM-a i CPU-a kako bi radio bez greške. Kada se u Eclipseu razvijaju aplikacije za Android, često se dogodi da se IDE sruši jer nije predviđen za tu vrstu razvoja. Zna se dogoditi da nakon nekoliko sati neprekidnog rada u Eclipseu, IDE prestane raditi pa ga se treba ponovno pokrenuti. S druge strane, Android Studio je, kada se počeo koristiti, bio još u beta-verziji razvoja, a to sa sobom donosi probleme sa stabilnosti razvojne okoline, koji su u međuvremenu bitno popravljeni, ali još nisu i sasvim uklonjeni.

2.3. Android Studio

Android Studio je integrirana okolina (Integrated Development Environment – IDE) za razvoj Android aplikacija. Preuzimanje je besplatno i dozvoljeno je Apache 2.0 licencom. Za razvoj aplikacija Android Studio zahtijeva instalaciju Java Development Kit (JDK) 7 ili više. Od Android Studija 2.3.0 nadalje, Android Studio dolazi s ugrađenom Javom, to jest nije više potrebno posebno instalirati Javu.

---

1 Autozavršavanje (engl. *autocompletion*) koda je funkcionalnost IDE okruženja koja, prilikom pisanja koda, od strane sučelja sugerira kako tu riječ nadopuniti.

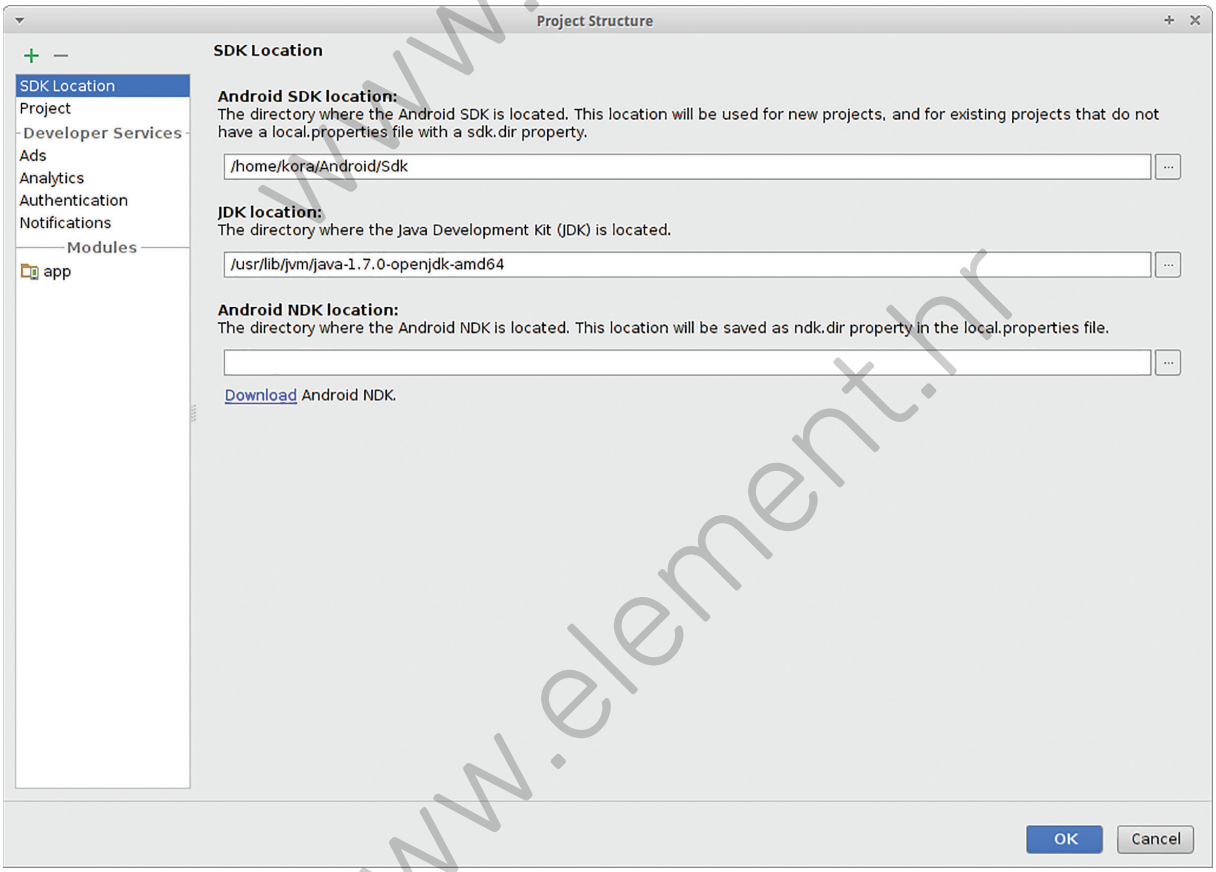
2 Refaktorizacija je proces kojim mijenjamo kôd nekog programa, a da ne mijenjamo način na koji se on prezentira, drugim riječima, uvodimo promjene u pozadini koje korisnik neće primijetiti. Takve promjene u kodu rade se radi lakšeg održavanja koda, smanjenja složenosti i slično.

Android Studio je dostupan za preuzimanje na adresi `https://developer.android.com/sdk/index.html`.

Instalacija

Nakon preuzimanja paketa za Linux operacijski sustav, potrebno je preuzeti paket raspakirati na željenu lokaciju, nazovimo je *ANDROIDHOME*. U preuzetoj mapi nalaze se upute za daljnju instalaciju.

Programiranje za Android oslanja se na programski jezik Java, koji se standardno koristi za objektno orijentirano programiranje. Stoga, prije nego što započnemo podešavanje Android Studija (u starijim verzijama, prije 2.3.0), potrebno je instalirati JDK (Java Development Kit) na sustav. JDK se može besplatno preuzeti s Interneta na `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. Treba je instalirati prema navedenim uputama (za Linux se upute mogu naći ovdje: `http://www.wikihow.com/Install-Oracle-Java-JDK-on-Ubuntu-Linux`).

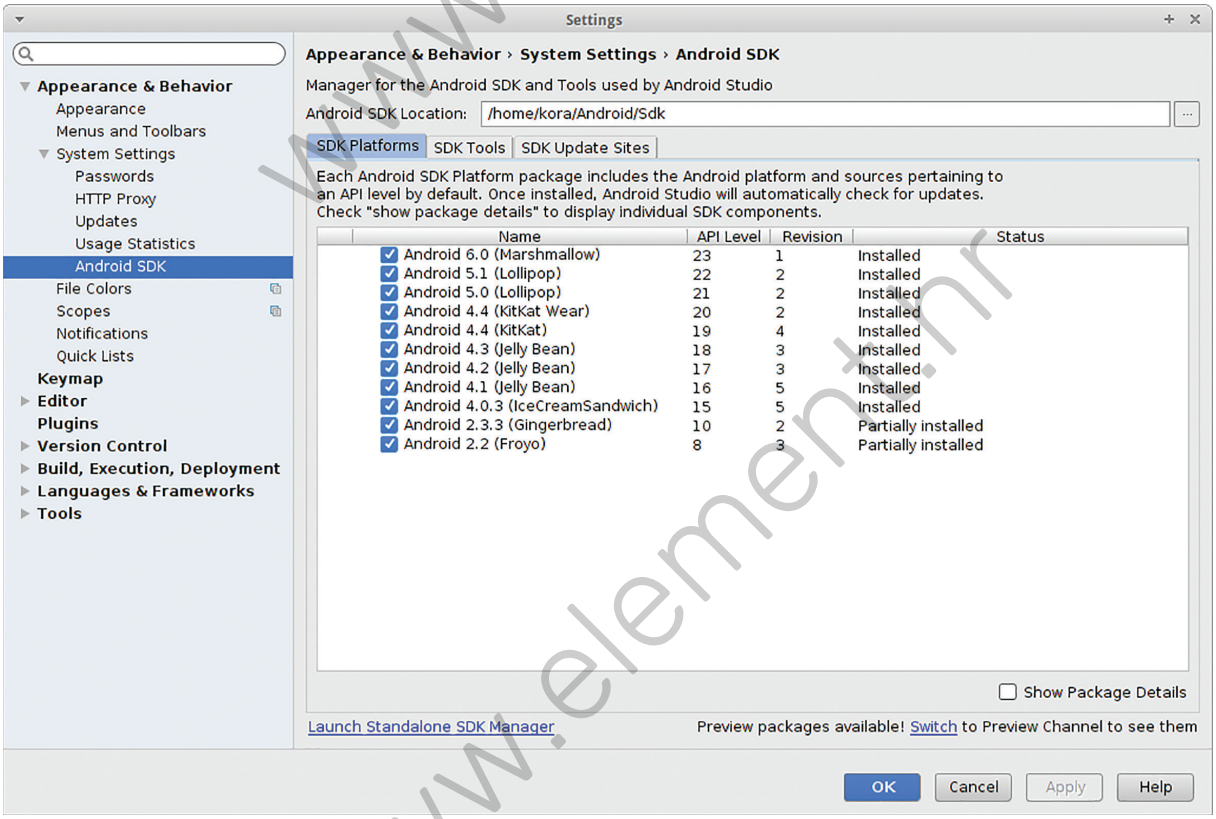


Slika 2.1. Podešavanje lokacije u Android Studiju

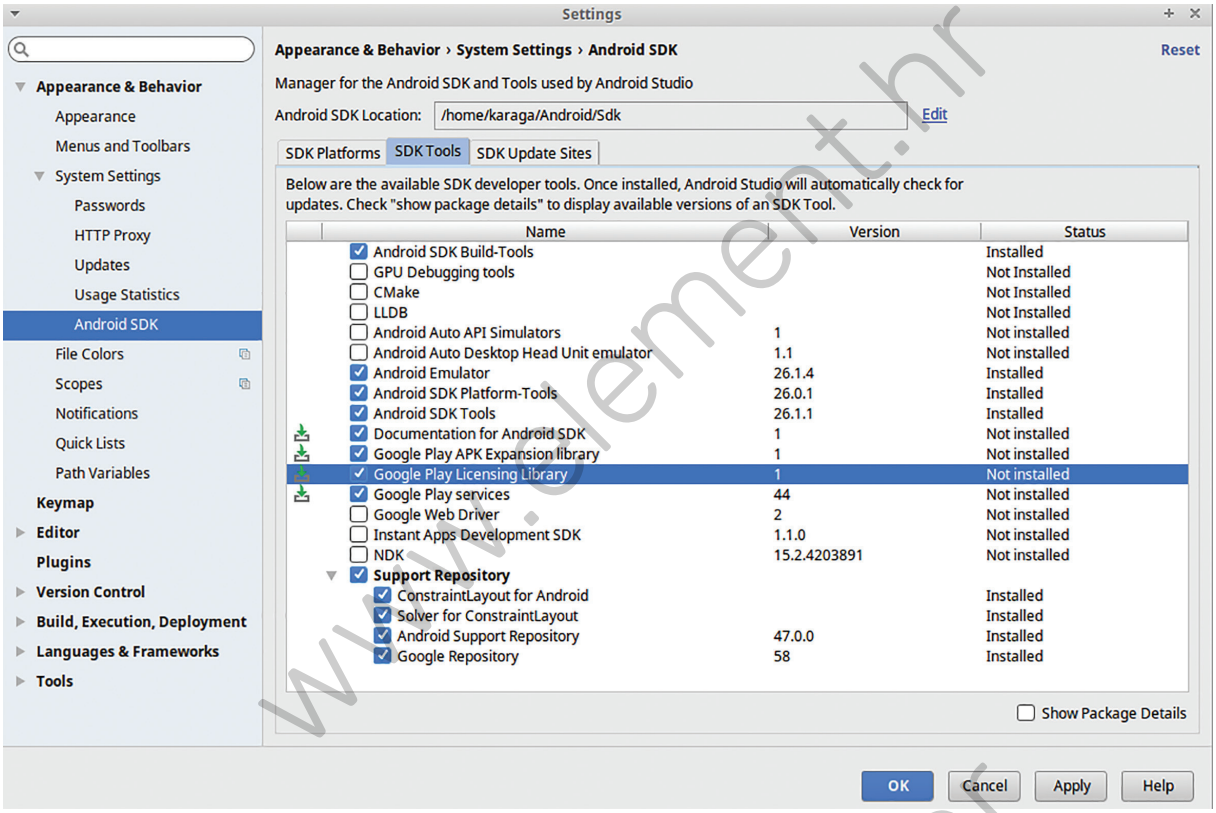
Podešavanje Android Studija

Pozicioniramo se u *ANDROIDHOME/bin* direktorij te pokrenemo *studio.sh* skriptu (ili radimo li putem terminala upišemo *./studio.sh*). Naravno, možemo postaviti i desktop ikonu za direktno pokretanje skripte. Ako smo instalirali vlastitu Javu i želimo je koristiti umjesto ugrađene, ili nemamo ugrađenu, tada trebamo podesiti da Studio zna koju Javu treba koristiti. U *File → Project structure* unesemo lokaciju Jave u za to predviđeno polje (slika 2.1), uglavnom je to /usr/bin/jvm/ i upišemo verziju koju smo instalirali, te pohranimo postavke.

Sljedeći korak je instalirati podršku za sve inačice Androida za koje mislimo razvijati aplikacije (u našem slučaju su to sve inačice). Navigiramo u *File → Settings → Appearance & Behavior → System Settings → Android SDK*, izaberemo karticu *SDK Platforms*, te označimo sve navedene inačice (slika 2.2, kako se pojavljuju nove inačice, i one će biti ponuđene na popisu). Zatim na kartici *SDK Tools* označimo kao što je prikazano na slici 2.3. Kliknemo gumb za instalaciju i čekamo da se sve instalira.



Slika 2.2. Instalacija *SDK Platforms*



Slika 2.3. Instalacija *SDK Tools*

Pokretanje Android Studija

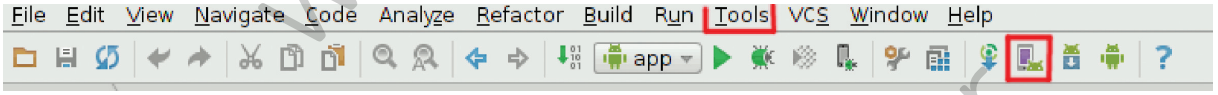
Prilikom prvog pokretanja Android Studija javlja se izbornik koji nam nudi da izaberemo stvaranje novog projekta (engl. *Start a new Android Studio project*), pokretanje postojećeg projekta (engl. *Open an existing Android Studio project*) ili pokretanje projekta stvorenog na drugom računalu ili u drugom IDE-u (engl. *Import project*). Nakon što izaberemo stvaranje novog projekta, treba popuniti ime projekta[3], *company domain* iz kojeg se generira ime paketa (engl. *package name*), te lokaciju projekta. Ime paketa je jedinstveni identifikator naše aplikacije. Obavezno mora sadržati barem jednu točku u imenu. Uobičajeno je koristiti ime domene naše organizacije (tvrtke, fakulteta i slično) plus još nekoliko dodatnih identifikatora aplikacije. Ime paketa piše se po takozvanom naopakom url principu (na primjer hr.math.primjeri.prvaaplikacija, gdje je ime aplikacije Prvaaplikacija, a *company domain* primjeri.math.hr). Lokacija projekta je datoteka u koju želimo spremati naše projekte. Nakon toga biramo uređaje za koje se razvija naša aplikacija (za naše primjere to će uvijek biti telefoni i tableti) i minimum SDK što

3 Uobičajena je konvencija da ime projekta počinje velikim slovom. Treba voditi računa da će to ime biti naziv naše aplikacije u Google Play trgovini, pa je poželjno da bude lako pamtljivo što će se svidjeti korisnicima i što dobro opisuje našu aplikaciju)

je najniža verzija Androida koju će naša aplikacija podržati (za početak možemo izabrati Android 8.0), to jest aplikacija će raditi za tu verziju i za sve verzije nakon nje. Prilikom izbora minimalnog SDK-a treba biti oprezan – izbor niže verzije ne uskraćuje nam pristup novijim funkcionalnostima, to se lako može postići, ali izbor previsoke verzije uskraćuje nam pristup svim korisnicima koji imaju uređaje s nižim verzijama Androida. S druge strane izbor preniske verzije iziskuje previše uzaludnog truda. I na kraju, većina aplikacija sadrži barem jednu aktivnost. Stoga izaberimo za početak praznu aktivnost (engl. *Empty Activity*) i neka zadrži svoje zadano (engl. *default*) ime (engl. *MainActivity*) i postavke. Android Studio sada stvara naš projekt, koji odmah sadrži čitav niz datoteka koje ćemo detaljno obraditi u idućem poglavlju.

2.4. Android emulator

Android Studio uključuje i Android emulator (Android Virtual Device – AVD), koji nam omogućava da modeliramo stvarni uređaj. Dakle, sve svoje aplikacije možemo testirati i bez Android uređaja (iako ih je svakako poželjno prije objave testirati i na što više pravih uređaja). Emulatora možemo stvoriti koliko želimo. Poželjno je imati više njih s raznim konfiguracijama kako bismo mogli dobro testirati svoju aplikaciju. Emulatori se stvaraju korištenjem Android Virtual Device Managera (slika 2.4).



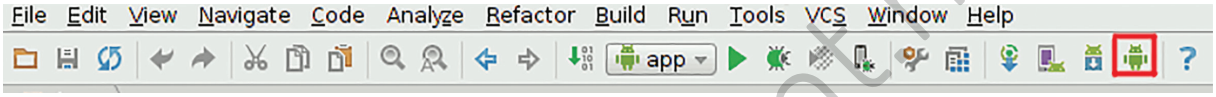
Slika 2.4. Android Virtual Device Manager

Za stvaranje novog emulatora izaberemo *Create virtual device*, te biramo neki iz velikog ponuđenog popisa stvarnih uređaja i konfiguracija, popunimo mu ime i karakteristike. Tako stvoreni uređaj ponašat će se kao stvarni uređaj. Nakon što ga stvorimo, njegovo ime se pojavljuje na popisu stvorenih uređaja u Android Virtual Device Manageru i pokrenemo ga pritiskom na zelenu strelicu kraj imena uređaja koji želimo pokrenuti. Prvo pokretanje uređaja, osobito na starijim i slabijim računalima ponekad traje dosta vremena (u starijim konfiguracijama i do 5 minuta), zbog toga je dobro jednom pokrenuti uređaj ne gasiti čitavo vrijeme rada u Android Studiju. Osim gumba za pokretanje kraj imena uređaja imamo i ikonu olovke koja služi za naknadnu promjenu postavki već stvorenog uređaja.

Prebacivanje datoteka

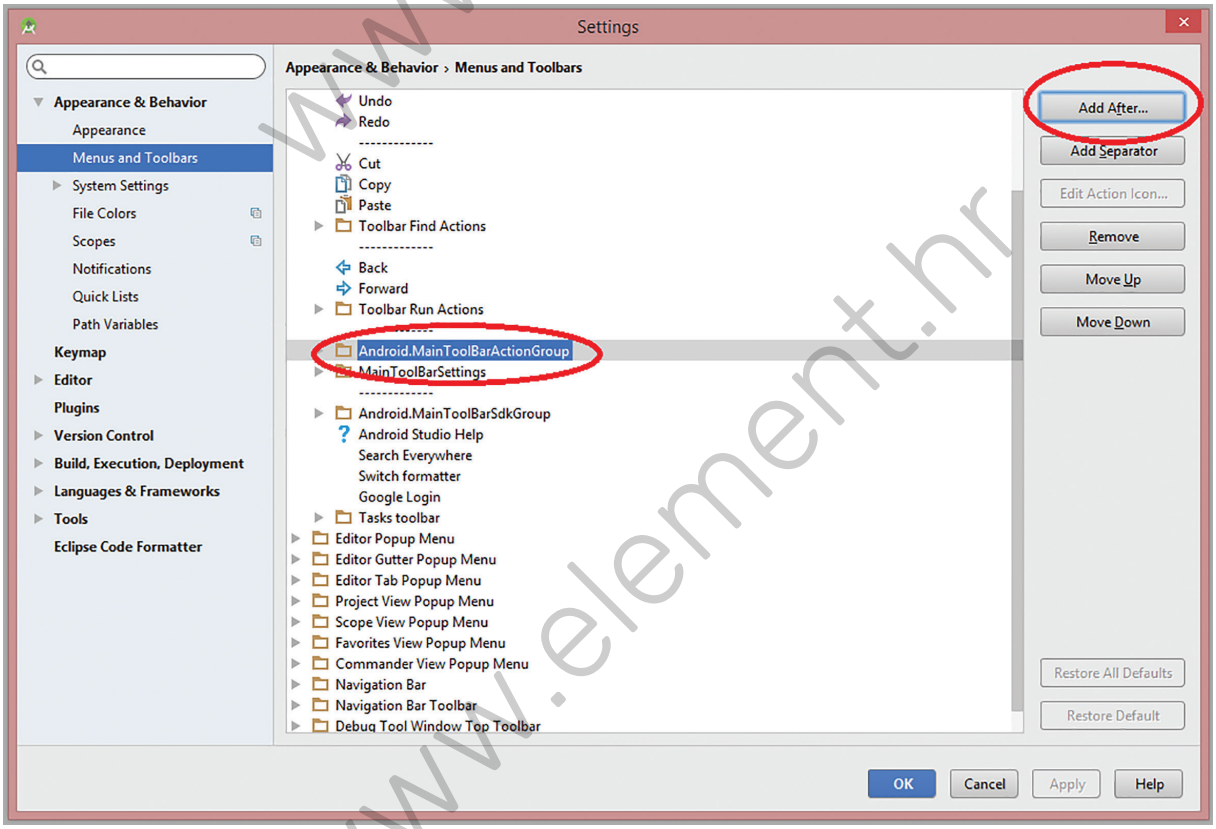
Prebacivanje datoteka na i s emulatora nam neće često trebati jer, kad napišemo vlastitu aplikaciju i pokrenemo je, ona automatski traži postojeći emulator s odgovarajućom inačicom Androida i prebacuje se na njega (u slučaju više takvih, nudi nam se da izaberemo jedan). Ipak, ponekad ćemo trebati prebacivati datoteke. To je najlakše korištenjem DDMS perspektive (Dalvik Debug Monitor Server) koja se otvara pritiskom na ikonu *Android Device Monitor* (slika 2.5) ili izborom *Tools* → *Android* → *Android*

Device Monitor. U DDMS perspektivi biramo *File Explorer* i s pomoću njega na uobičajen način možemo prebacivati datoteke. Tu ujedno vidimo i ovlaštenja pojedinih datoteka.



Slika 2.5. Ikona *Android device monitor*

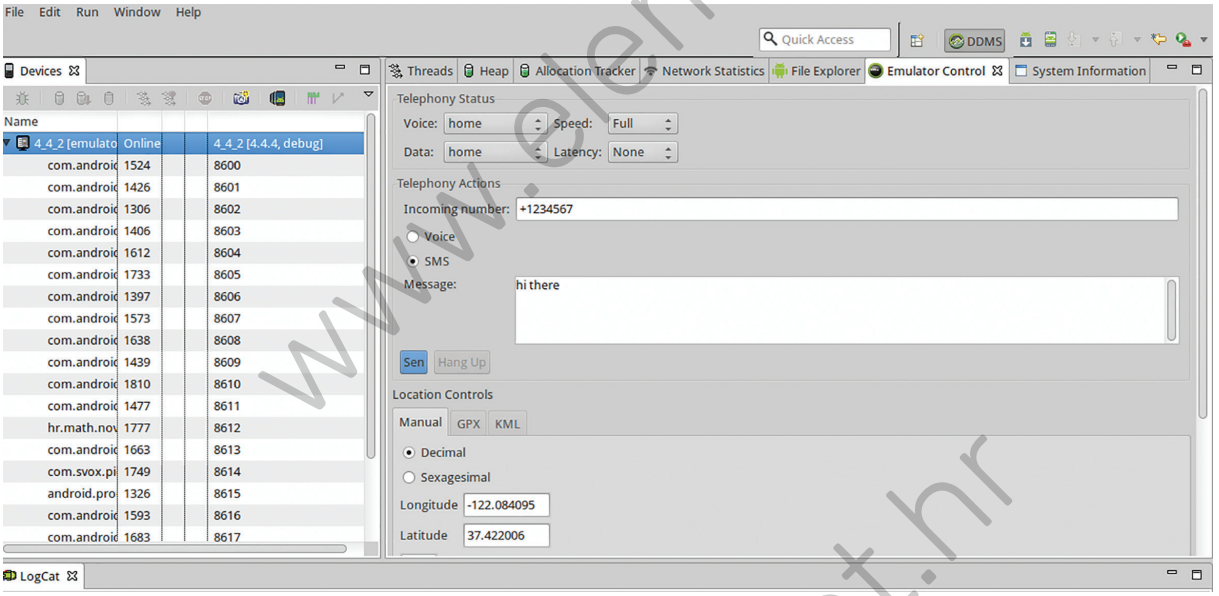
Napomena. Nove inačice Android Studija više ne sadrže ikonu *Android device monitor* po definiciji, već se za pristup *File exploreru* i ostalim funkcijama DDMS perspektive oslanjaju na izbornik koji se pojavljuje desno od emulatora, a omogućava dodatne funkcionalnosti emulatora. Kako dodati ovu (ili bilo koju drugu ikonu): biramo *File* → *Settings* → *Main Toolbar*, slijedimo postupak sa slike 2.6, te dodamo *Main Menu* → *Tools* → *Android* → *Android Device Monitor* i pritisnemo *OK*. Ikona je sada uspješno dodana na svoje uobičajeno mjesto u alatnoj traci.



Slika 2.6. Dodavanje ikone

SMS poruke i telefonski pozivi emulatoru

Naš emulator može primati telefonske pozive i sms poruke. To je ponekad potrebno radi testiranja aplikacija. Da bismo to mogli, najprije moramo znati port emulatora. Port emulatora piše u naslovnom okviru (engl. *title bar*) Android emulator prozora kada pokrenemo emulator. Po definiciji, prvi pokrenuti emulator ima port 5554, a svaki idući emulator +2, to jest 5556, 5558 i tako dalje. Za slanje poziva i poruka koristimo DDMS perspektivu. Biramo iz popisa emulatora onaj kojem želimo poslati poruku i u *Emulator control*, *Telephony actions* (slika 2.7) ispunimo fiktivni telefonski broj s kojeg želimo poslati poruku (na primjer +1234567) i tekst poruke te pritisnemo *Send*. Poruka će se pojaviti na uređaju.



Slika 2.7. Slanje SMS-ova

Napomena. Imamo li više emulatora pokrenutih odjednom, oni mogu slati poruke jedan drugome. U tom slučaju umjesto telefonskog broja pišemo port emulatora (na primjer 5556).

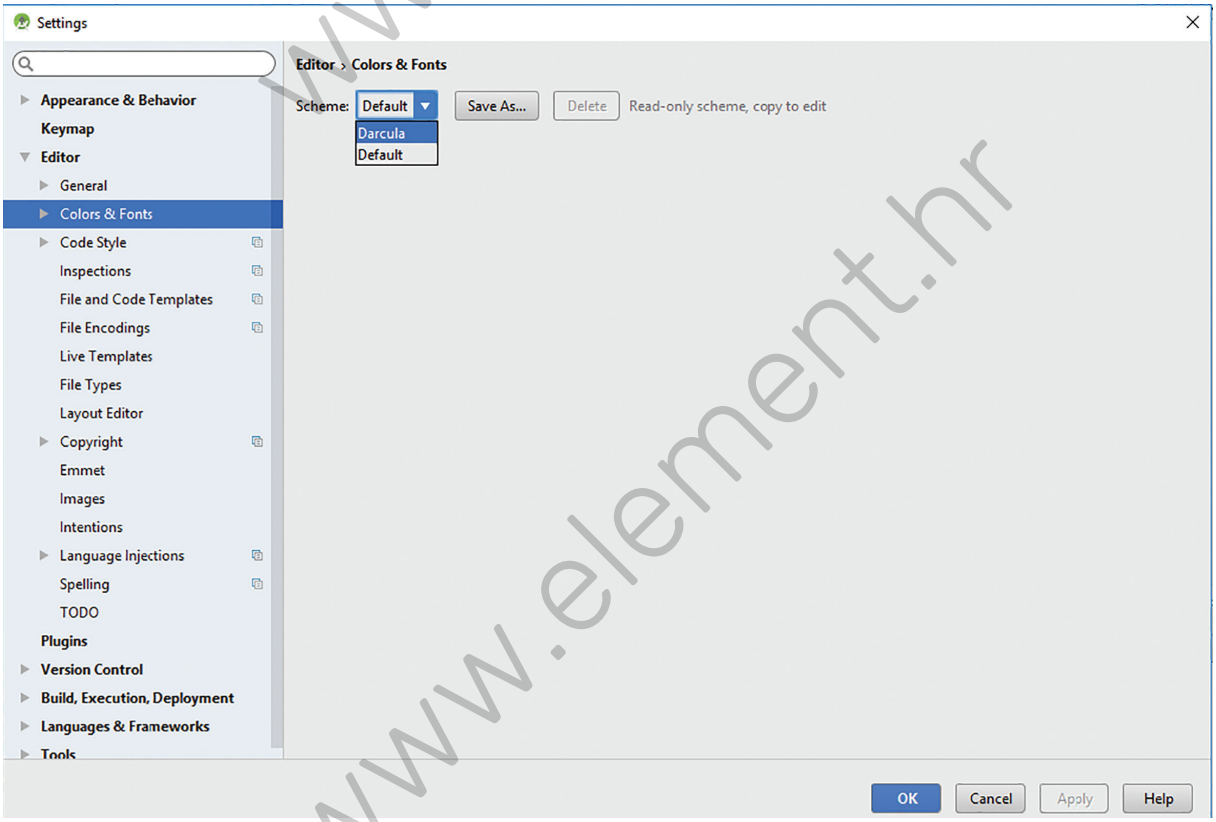
Resetiranje emulatora

Ponekad želimo pobrisati sve što smo na emulator stavili, a da ga ne moramo ponovno stvorati. To činimo tako da pronađemo na računalu direktorij *avd* (obično *users/username/.android/avd*) i u njemu ime emulatora koji želimo resetirati. Zatim u tom direktoriju obrišemo *userdata-quemu.img* datoteku.

Zadatak 1. Stvorite jedan emulator. Pokrenite ga i pogledajte koje aplikacije sadrži. Pošaljite svom pokrenutom emulatoru sms poruku sa fiktivnog telefonskog broja. Pogledajte primljeni sms na emulatoru.

2.5. Još o Android Studiju

Android Studio podešen je tako da nam njegovo korištenje umnogome olakšava programiranje za Android uređaje. Jedna od glavnih prednosti je njegova *autocomplete* opcija – kada počnemo pisati ime neke funkcije ili nekog drugog elementa, već nam se nude sve mogućnosti koje počinju na taj način. Također, kao što nam je poznato iz programiranja, postojeće klase su organizirane u pakete koje treba uvesti (engl. *import*) u naš program ako ih želimo koristiti. U Androidu ne moramo znati u kojem su paketu klase ili metode koje želimo. Dovoljno je u kodu kliknuti mišem na objekt koji javlja grešku i pritiskom tipki *alt+Enter* dobivamo popis paketa koje bi trebalo uvesti, te ih automatski uvezemo klikom miša. Još jedno korisno svojstvo je istovremeno mijenjanje imena objekta i varijabli – ako kliknemo mišem na neki objekt ili varijablu, izaberemo *Refactor* → *Rename*, mijenja se ime u čitavom našem projektu. I na kraju, ponekad želimo promijeniti temu Android Studija kako bi nam druga kombinacija boje pozadine i fonta olakšala višesatno gledanje u ekran. Android Studio dolazi s dvijema zadanim (engl. *default*) temama, *Default* i *Darcula*, a korisnik može dodavati teme na način da preuzme .icl datoteku teme [22], pohrani je u *user/.AndroidStudiobrojverzije/config/colors* direktorij (ako poddirektorij *colors* ne postoji, treba ga stvoriti) i ponovno pokrene Android Studio. Nova tema trebala bi se pojaviti na popisu tema zajedno sa zadanima, te je sada možemo izabrati (slika 2.8).



Slika 2.8. Teme Android Studija

3. Prva Android aplikacija

Kada smo pokrenuli Android Studio i stvorili novi projekt, dobili smo automatski generiran čitav niz datoteka. Naš novostvoreni projekt nije prazan već ispisuje dobro poznati *Hello World*. Pokretanjem programa (pritiskom tipke *Run* koja izgleda kao zeleni trokutić u alatnoj traci) on se automatski instalira na odabrani emulator i pokreće na njemu. Ako nismo stvorili ni jedan emulator koji konfiguracijom (prije svega inačicom Androida) odgovara našem programu, potrebno ga je stvoriti i pokrenuti da bi se program mogao izvršiti.

Zadatak 2. Pokrenite *Hello World* program na emulatoru.

Kada stvorimo projekt, automatski se otvaraju dvije datoteke – to su .java i .xml datoteke za glavnu aktivnost. Aktivnost je prozor s korisničkim sučeljem aplikacije. Aplikacija može sadržavati više aktivnosti (ovisno koliko će različitih korisničkih sučelja trebati za izvršenje svojeg zadatka), a tipično sadrži barem jednu aktivnost. To je glavna aktivnost u koju se ulazi odmah po pokretanju programa, to jest prvo korisničko sučelje koje korisnik vidi. Ime te glavne aktivnosti je tradicionalno *MainActivity*. U Androidu je, slično kao i u XHTML-u gdje imamo razdvajanje na XHTML kod i CSS, prisutno strogo razdvajanje koda na dio koji opisuje što se radi (.java datoteka) i dio koji opisuje kako to izgleda (.xml datoteka). Dakle, ako na primjer naša aplikacija sadrži jedno polje za unos teksta i jedan gumb čijim pritiskom se taj tekst negdje šalje, gumb i polje za unos teksta sa svojim oblicima, lokacijom, bojama i slično nalazit će se u .xml datoteci, a funkcije koje iščitavaju tekst i šalju ga dalje nakon pritiska gumba pisat će u .java datoteci. Da bismo mogli dohvatiti elemente korisničkog sučelja u kodu, dodajemo im identifikacijske atribute. Dakle, jasno je razdvojena logika programa od dizajna korisničkog sučelja. To razdvajanje ima mnoge prednosti, jer čini kôd preglednijim, a i lakše je raditi naknadne promjene.

Pogledajmo najprije .java datoteku. Ona u Androidu 8 izgleda kao na slici 3.1, a u ranijim inačicama kao na slici 3.2. Sva sintaksa preuzeta je od Jave (jedino će se koristiti neke specifične optimizacijske tehnike). U oba slučaja najprije imamo ime paketa, zatim biblioteke koje uključujemo (engl. *import*), te jednu klasu koja je proširenje klase `Activity` ili `AppCompatActivity` i u kojoj nadjačamo (engl. *override*) metodu `onCreate( )` iz nadklase. (Za objašnjenje proširenja klase, nasljeđivanja i sličnih koncepata objektno orijentiranog programiranja vidjeti Dodatak.) `onCreate( )` je metoda koja se izvršava u trenutku stvaranja aktivnosti, to jest ono što stavimo u nju izvršit će se odmah. Naša metoda samo poziva

`onCreate( )` metodu iz nadklase, te `setContentView( )` metodu. `Activity` je bazna klasa za vizualne, interaktivne komponente naše aplikacije. Sve aktivnosti će biti bazirane na klasi `Activity`. Ono što ne piše u .java datoteci je kako izgleda naše koriničko sučelje. U Androidu se vizualne komponente zovu *Views*. Zbog toga se poziva `setContentView( )` metoda koja postavlja korisničko sučelje. Dakle to je dio koda koji povezuje aktivnost s korisničkim sučeljem. Kod nas se korisničko sučelje nalazi u datoteci *activity_main.xml* stoga ga navodimo kao argument metode.

```
activity_main.xml ×   MainActivity.java ×

1   package hr.math.karaga.primjeri.prvaaplikacija;
2
3   import android.support.v7.app.AppCompatActivity;
4   import android.os.Bundle;
5
6   public class MainActivity extends AppCompatActivity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.activity_main);
12      }
13  }
14
```

Slika 3.1. Java datoteka glavne aktivnosti Android 8

```
MainActivity.java   activity_main.xml   strings.xml   P

package hr.math.Prva;

import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Slika 3.2. Java datoteka glavne aktivnosti ranije inačice

Zbog čega se u Androidu 8 pojavljuje `AppCompatActivity`? Android se brzo i često mijenja, uvode se nove mogućnosti i poboljšanja. U isto vrijeme nastoji se osigurati i kompatibilnost unatrag, to jest da kôd napisan za starije inačice i dalje radi besprijekorno.[1] Klasa `Activity` je u početku uključivala `ActionBar` element po definiciji. Kasnije je `ActionBar` proglašen zastarjelim i izbačen iz klase. Stoga, ako želimo koristiti `ActionBar` u novim verzijama, treba nam klasa `AppCompatActivity`. Klasa `Activity` još uvijek postoji i radi.

1 U Android programiranju opasno se koncentrirati samo na najnovije inačice, jer tada gubimo pristup širokom tržištu korisnika starijih uređaja. Ujedno, promjene su stalne i česte, pa se važno naučiti nositi s njima na ispravan način. Zbog tih razloga ćemo često pokazivati i starije verzije koda.

Napomena. *ActionBar* je traka koja se pojavljuje na vrhu aplikacije i sadrži ime aplikacije s lijeve strane i izbornik (⋮) s desne strane. Pruža Android aplikacijama konzistentan izgled. Naravno, za neke aplikacije *ActionBar* nije primjeren i izgledaju bolje bez njega.

U .xml datoteci imamo XHTML oznake (engl. *tag*), koji svi trebaju biti pravilno ugniježđeni i zatvoreni (za više o XHTML-u vidjeti Dodatak). Svaki tag predstavlja po jedan vizualni element. Zadani pogled na .xml datoteku je na njen grafički izgled. Želimo li bolje vidjeti njegovu strukturu ili nešto mijenjati, potrebno se prebaciti na tekstualni pogled (*Text* kartica na dnu prozora).[2] XML datoteka u Androidu 8 izgleda kao na slici 3.3, a u starijim inačicama kao na slici 3.4.

activity_main.xml × MainActivity.java ×

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context="hr.math.karaga.primjeri.prvaaplikacija.MainActivity">
8
9      <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18 </android.support.constraint.ConstraintLayout>
19
```

Slika 3.3. XML datoteka glavne aktivnosti Android 8

MainActivity.java × strings.xml × activity_main.xml ×

```
<RelativeLayout 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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

    <TextView android:text="@string/hello_world" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Slika 3.4. XML datoteka glavne aktivnosti starije inačice

2 I takozvani dizajnerski pogled (na grafički izgled) dopušta izmjene i dodavanje elemenata na korisničko sučelje, ali takvo grafički generirano korisničko sučelje neće nikada biti dobro i stabilno, ni izgledati profesionalno kao ono napisano u tekstualnom pogledu. Zbog toga se takva praksa treba izbjegavati.

Obje datoteke sadrže dva glavna taga, *Layout* tag i *TextView* tag, s pripadnim atributima. Svi tagovi i atributi bit će detaljno objašnjeni u kasnijim poglavljima. *Layout* tag služi za određivanje rasporeda svih ostalih grafičkih elemenata u grafičkom sučelju. Ima mnogo mogućnosti za *layout* i biramo onaj koji nam najbolje odgovara ovisno o situaciji. Do Androida 8 je za aktivnost zadani *layout* bio *RelativeLayout* koji određuje položaj elemenata relativno jedan prema drugome (dakle element 1 je ispod elementa 2 i desno od elementa 3 i slično), a od Androida 8 uvodi se novi *layout* koji je ujedno i predložen kao zadani za aktivnost, *ConstraintLayout*. On je proširenje za *RelativeLayout*, to jest ima mogućnosti kao *RelativeLayout* i još neke dodatne. Najvažniji *layout* atributi su svakako `width` i `height` (širina i visina), koji očito definiraju širinu i visinu elementa. Njihove najčešće vrijednosti su `match_parent` (određuje da element bude širok odnosno visok kao i element roditelj, dakle u slučaju *layouta* kao čitav ekran, a u slučaju ugniježđenog elementa, kao element u kojem se nalazi) i `wrap_content` (određuje da element bude širok ili visok kao što je to potrebno za sadržaj – na primjer, ako na gumbu piše Android, on će biti širok i visok točno koliko je potrebno da obuhvati ta slova). *TextView* tag je element koji služi za prikaz nekog teksta, i kod nas je u njemu smješten *Hello World*. On sadrži `android:text` atribut u kojem se nalazi tekst koji se prikazuje. Uočite da se sadržaj tog atributa barem naizgled razlikuje u naše dvije verzije. Vrijednost *text* atributa može se "hardkodirati" to jest napisati izravno u kodu, međutim to se smatra jako lošom praksom i treba je izbjegavati. U Androidu je važno razdvajanje koda od resursa (a tekst koji se ispisuje ili piše na raznim elementima, na primjer gumbima i slično smatra se tekstualnim resursom). Zbog toga sav tekst treba umjesto direktno u kodu pisati u odgovarajuću datoteku, a to je u ovom slučaju *strings.xml* datoteka koja se nalazi u *res/values* direktoriju projekta i koja je jedna od automatski stvorenih datoteka u svakoj novoj aplikaciji. Sintaksa je

```
android:text="@string/ime_tekstualne_konstante"
```

za vrijednost atributa, a u *strings.xml* datoteci imamo tag

```
<string name="ime_tekstualne_konstante">Tekst koji se ispisuje</string>
```

`@string` dio atributa govori da je riječ o tekstualnom resursu koji se nalazi u *strings.xml* datoteci, a `ime_tekstualne_konstante` je naravno ime resursa i dajemo ga proizvoljno (uz uobičajenu konvenciju u programiranju: poželjno je da imena budu deskriptivna i jednostavna). Sam tekst se nakon toga piše u odgovarajući tag.

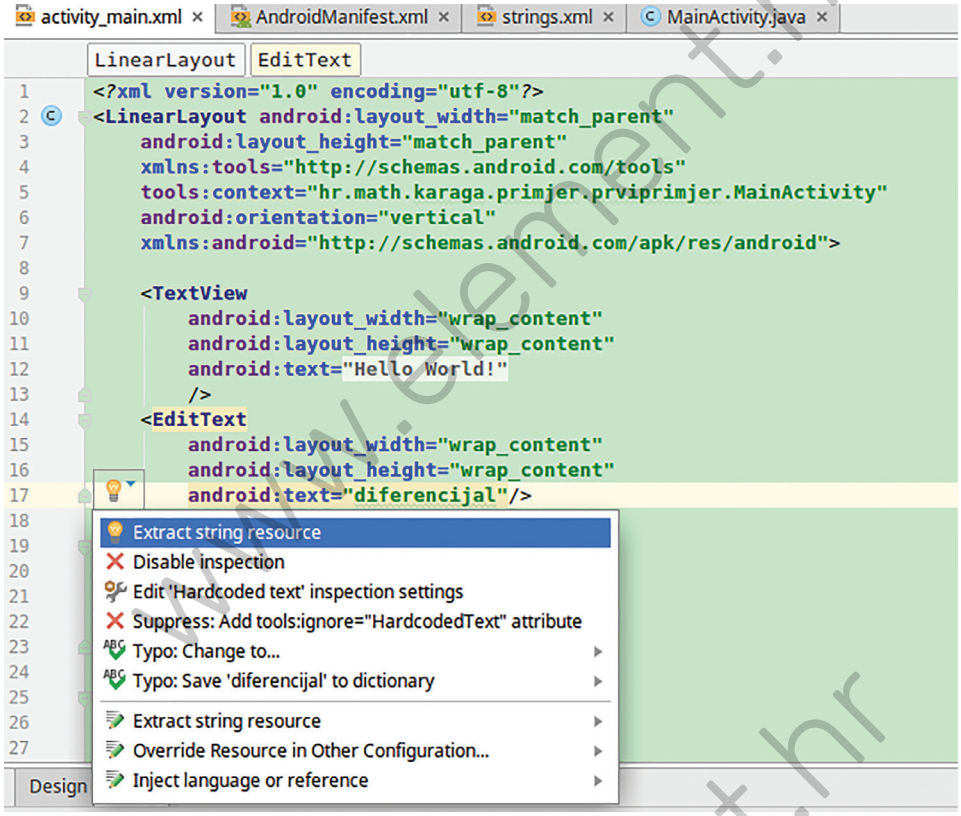
Zadatak 3. Otvorite i pogledajte *strings.xml* datoteka projekta. (Napomena: direktoriji *res* i *values* nalaze se u *Project* prozoru, lijevo od prozora s kodom.)

U Androidu 8 na početku se u toj datoteci nalazi samo ime projekta, dok su ranije inačice imale i ispravno zapisan *Hello world*:

```
<string name="hello_world">Hello_world!</string>
```

Unos mnogobrojnih tekstualnih konstanti na taj način može biti naporan. Zbog toga novije inačice Android Studija sadrže opciju *Extract string resource*, to jest, unesemo tekst direktno kao vrijednost

atributa u kodu, pozicioniramo se mišem na njega i izaberemo opciju koja će zatim automatski unijeti vrijednost u *strings.xml* datoteku (slika 3.5).



Slika 3.5. Unos string atributa

Koja je svrha ovog razdvajanja tekstualnih resursa? Da bi naša aplikacija bila što bolja i zanimljivija na širokom tržištu, želimo je prevesti na što više jezika. Kada se svi tekstovi nalaze u istoj datoteci, prevođenje je očito puno lakše i jednostavnije. Osim samog procesa prevođenja, nakon kojeg imamo više datoteka s prevedenim tekstovima, i izbor koja će se datoteka koristiti je bitno olakšan – Android uređaj sam bira datoteku s jezikom koji mu je podešen kao primarni u postavkama.

Sada smo gledali *res* direktorij u *Project* prozoru Android Studija. Osim direktorija *res*, tamo se nalaze i direktoriji *java* i *manifests*. U direktorij *java* smještamo sve Java datoteke. U *res* smještamo sve resurse. Osim tekstualnih resursa koji se nalaze u *strings.xml* datoteci (u direktoriju *values*), tu nalazimo i našu XML datoteku glavne aktivnosti (unutar direktorija *layouts*). Direktorij *res* sadrži sve resurse korištene u aplikaciji. Resursi su vanjske datoteke koje naš projekt koristi i prevodi (engl. *compile*) u našu aplikaciju – zapravo, sve što aplikacija koristi, a nije kôd (tekstove, slike, *layoute*...). Direktorij *manifests* sadrži samo jednu datoteku *AndroidManifest.xml* i to je manifest datoteka za našu aplikaciju.

Android Manifest

Android Manifest je datoteka koju mora imati svaka aplikacija (i svaka ima točno jedan). On se automatski generira, sadrži detaljne informacije o našoj aplikaciji, i uglavnom ga nećemo mijenjati (dapače, mijenjanje manifesta može dovesti do brojnih i teško ispravljivih grešaka).

Zadatak 4. Otvorite manifest datoteku svoje aplikacije.

Unutar manifesta (slika 3.6) najprije imamo `<manifest>` tag i u njemu definirano ime paketa.[3] Zatim imamo `<application>` tag koji u svojim atributima sadrži ime aplikacije, ime slike koja je ikona za pokretanje aplikacije (engl. *launcher*), temu koju smo prilikom stvaranja izabrali za našu aplikaciju i slične podatke. Manifest smije imati samo jedan `<application>` tag. Unutar `<application>` taga nalaze se tagovi za sve aktivnosti koje aplikacija sadrži, kao i podatci o njima (s kakvim podatcima ta aktivnost radi, kako može biti pokrenuta i slično). Naš projekt na početku sadrži samo jednu, glavnu aktivnost. Da se radi o glavnoj aktivnosti i aktivnosti koja se može pokrenuti s uređaja s pomoću ikone (dakle o prvoj aktivnosti koju vidimo kada pokrenemo aplikaciju) vidimo iz vrijednosti atributa akcija i kategorija (`MAIN` i `LAUNCHER`).

activity_main.xml × | AndroidManifest.xml × | settings.gradle × | Prvaaplikacija × | app × | strings.xml ×

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="hr.math.karaga.primjeri.prvaaplikacija">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/ic_launcher"
8          android:label="Prvaaplikacija"
9          android:roundIcon="@mipmap/ic_launcher_round"
10         android:supportsRtl="true"
11         android:theme="@style/AppTheme">
12         <activity android:name=".MainActivity">
13             <intent-filter>
14                 <action android:name="android.intent.action.MAIN" />
15
16                 <category android:name="android.intent.category.LAUNCHER" />
17             </intent-filter>
18         </activity>
19     </application>
20
21 </manifest>
```

Slika 3.6. Android Manifest

3 Ranije inačice Androida imale su definiranu i verziju aplikacije, te minimalnu inačicu Androida na kojoj će aplikacija raditi. To je sada prebačeno u Gradle (*Gradle Scripts* direktorij, *build.gradle (app)* skripta).

3.1. Dodavanje koda u aplikaciju

Dodajmo sada neke izmjene u našu prvu aplikaciju[4]. Najprije promijenimo *layout* iz zadanog *Constraint Layouta* u *Linear Layout*: otvorimo .xml datoteku glavne aktivnosti i zamijenimo dio koda

`android.support.constraint.ConstraintLayout`

sa

`LinearLayout`

i na početku i u zatvaraču taga. *Linear Layout* stavlja objekte u jedan redak ili jedan stupac, ovisno o vrijednosti atributa `android:orientation` koja može biti `horizontal` za redak ili `vertical` za stupac. Dodajmo

`android:orientation="vertical"`

u *LinearLayout* tag. Unutar *LinearLayout* taga dodajmo najprije jedno polje za unos teksta (`EditText` tag) s pripadnim osnovnim atributima:

```
<EditText
      android:id="@+id/edit_text_polje"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:hint="@string/hint_poruka"/>
```

Atribut `android:hint` odnosi se na tekst koji će se nalaziti u polju za unos teksta dok je još prazno, i u koji se obično piše kratka uputa (engl. *hint*) korisniku što bi trebao unijeti u polje. Vrijednost tog teksta treba sad upisati u datoteku *strings.xml* unutar `<resourcesA>` taga, pod imenom koje smo odabrali, u ovom slučaju `hint_poruka`:

```
<string name="hint_poruka">Unesite login ime</string>
```

Da bismo elemente XML datoteka mogli dohvatiti u kodu, potrebno im je definirati i *id* (`android:id`) atribut. Ovdje ga definiramo kao `edit_text_polje`. Znak `@` se odnosi na resurs (engl. *resource*), a znak + nam govori da definiramo *id* resursa prvi put (daljnje reference na *id* neće koristiti +). Kasnije dohvaćanje resursa u kodu je s pomoću `findViewById( )` metode. Dodajmo sada još jedan gumb (`Button`) element u našu .xml datoteku:

```
<Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/button_send"/>
```

i pripadni tekst koji želimo da se ispiše na gumbu u *strings.xml* datoteku:

4 Nakon ovog poglavlja sav kôd koji trebate unijeti možete pronaći na *webu*. Poveznica se nalazi u popisu *web*-izvora na kraju knjige, međutim za ovaj zadatak ga još nema kako bismo ručnim unosom stekli uvid u moćno automatsko završavanje (engl. *autocomplete*) funkcionalnost Android Studija.

```
<string name="button_send">Send</string>
```

Gumb nećemo dohvaćati u kodu, stoga mu ne treba *id* atribut. Naša aplikacija za sada može primati upisano prijavljeno ime, i možemo pritiskati gumb *Send*, ali ne radi ništa prilikom pritiska na gumb. Kako bismo joj dali pravu funkcionalnost, potrebni su nam komponente Aktivnosti i Intenti i prikladni kôd u .java datoteci.

Zadatak 5. Dodajte u svoju aplikaciju polje za unos teksta i gumb, promijenite *layout* na linearan te pokrenite aplikaciju na emulatoru. Probajte unijeti neki tekst u polje za unos teksta.