RAZVOJNE MOBILNE APLIKACIJE ZA OPERACIJSKI SUSTAV ANDROID

Granc, Alen

VELEUČILIŠTE U ŠIBENIKU

ODJEL MENADŽMENTA

PREDDIPLOMSKI STRUČNI STUDIJ MENADŽMENT

Alen Granc

RAZVOJ MOBILNE APLIKACIJE ZA OPERACIJSKI SUSTAV ANDROID

Završni rad

Šibenik, 2015.

VELEUČILIŠTE U ŠIBENIKU

ODJEL MENADŽMENTA

PREDDIPLOMSKI STRUČNI STUDIJ MENADŽMENT

RAZVOJ MOBILNE APLIKACIJE ZA OPERACIJSKI SUSTAV ANDROID

Završni rad

Kolegij: Osnove programiranja

Mentor: Želimir Mikulić, dipl.ing.

Student: Alen Granc

Matični broj studenta: 13555121

Šibenik, rujan 2015.

SADRŽAJ:

1. UVOD 1
2. LAYOUT I VIEW 2
   - 2.1. Definiranje view-a 4
   - 2.2. View grupe 6
3. JAVA 12
   - 3.1. Imena i varijable 13
   - 3.2. Klase i metode 16
   - 3.3. Interakcija pomoću metoda 18
4. OBJEKTNO ORIJENTIRANO PROGRAMIRANJE 20
   - 4.1. Konstruktori 22
5. MOBILNA APLIKACIJA 24
6. ZAKLJUČAK 31
7. PRILOG A Java izvorni kod aplikacije 32
8. PRILOG B XML kod 34
9. LITERATURA 36

TEMELJNA DOKUMENTACIJSKA KARTICA

Veleučilište u Šibeniku Završni rad

Odjel Menadžmenta

Preddiplomski stručni studij Menadžment

RAZVOJ MOBILNE APLIKACIJE ZA OPERATIVNI SUSTAV ANDROID

ALEN GRANC

Ul. Svetog Andrije 36a, 43240 Čazma, agranc94@gmail.com

Sažetak rada

U ovom radu objašnjen je postupak izrade mobilne aplikacije za operativni sustav Android. U uvodnom dijelu rada objašnjeni su ključni pojmovi vezani za principe objektno orijentiranog programiranja, te programske jezike Java i XML. Dan je prikaz korištenih razvojnih alata Java Development Kit i Android Studio koji su svima dostupni za besplatno preuzimanje na službenim stranicama Jave i Androida. Početna točka u razvoju svake aplikacije je ideja što će aplikacija raditi nakon čega se stvara vizija kako bi aplikacija trebala izgledati na zaslonu uređaja. Taj dio razvoja se odvija u XML jeziku u kojem programer definira koje objekte želi na zaslonu te njihovu poziciju i izgled. Ključan dio razvoja uspješne aplikacije, a posebno uspješne mobilne aplikacije, je izraditi efikasno i intuitivno korisničko sučelje. Programer odlučuje kako i na koji način će omogućiti interakciju pri čemu koristi programski jezik, u ovom slučaju Java-u. Rad opisuje cjelokupan proces razvoja na primjeru konkretne aplikacije.

(36 stranice / 8 slika / 8 literaturnih navoda / jezik izvornika: hrvatski)

Rad je pohranjen u: Knjižnici Veleučilišta u Šibeniku

Ključne riječi: mobilne aplikacije, Android Studio, Java, XML

Mentor: Želimir Mikulić, dipl.ing.

Rad je prihvaćen za obranu:

BASIC DOCUMENTATION CARD

Polytechnic of Šibenik Final paper

Department of Management

Professional Undergraduate Studies of Management

MOBILE APPLICATION DEVELOPMENT FOR ANDROID

ALEN GRANC

Ul. Svetog Andrije 36a, 43240 Čazma, agranc94@gmail.com

Abstract

This paper explains the process of creating mobile applications for the Android operating system. The principles of object-oriented programming, and programming languages Java and XML are explained in the introductory part of the paper. An overview of the development tools used: Java Development Kit and Android Studio, which are all available for free download on the official website of Java and Android. The starting point in the development of any application is the idea what the application will do after which programmer creates a vision of how an application should look like on screen. That part of the development takes place in the XML language in which programmers define objects he wants to display and their position and appearance. A key part of the development of successful applications, particularly successful mobile applications, is to develop an efficient and intuitive user interface. A programmer decides how and in what way will he allow the interaction while using a programming language, in this case Java. This paper describes the entire process of development in the case of specific applications.

(36 pages / 8 figures / 8 references / original in Croatian language)

Paper deposited in:Library of Polytechnic of Šibenik

Keywords: mobile applications, Android Studio, Java, XML

Supervisor: Želimir Mikulić, dipl. ing.

Paper accepted:

1. UVOD

Razvoj mobilnih aplikacija je izraz koji se koristi za označavanje procesa ili akta razvoja aplikativnog softwarea za mobilne uređaje, tablete i slične uređaje. Takve aplikacije mogu biti instalirane od strane proizvođača prije nego što uređaj uopće dođe u ruke potrošača, a mogu biti i tako zvane web aplikacije. Programeri aplikacija moraju uzeti u obzir i različite veličine samih uređaja kao i hardware koji pojedini uređaju koriste. Razvoj mobilnih aplikacija konstantno raste, u smislu prihoda i poslova koji se pojavljuju samim razvojem i povećanjem popularnosti smartphonova i sličnih uređaja.

Kao dio razvojnog procesa korisničko sučelje je esencijalno u kreiranju mobilnih aplikacija. Mobilno korisničko sučelje ili skraćeno UI(User Interface) ne podrazumijeva samo izgled aplikacije na ekranu, već i mogućnost da korisnik ostvari interakciju s aplikacijom. Kada govorimo o interakciji s aplikacijom ono što korisnik napravi(npr. pritisne gumb, upiše tekst) nazivamo „input“, a za rezultat se očekuje neka reakcija aplikacije koju nazivamo „output“. U konačnici cilj korisničkog sučelja je da bude jednostavno i razumljivo krajnjim korisnicima.

Razvoj mobilnih aplikacija možemo podijeliti u dvije glavne cjeline. Prva cjelina se odnosi na samu ideju kojom se vodimo pri izradi aplikacije. Trebamo znati što želimo od aplikacije kako bi znali kako dizajnirati „layout“ za našu aplikaciju. Osim same ideje što želimo postići aplikacijom potrebno je imati i predodžbu kako želimo da aplikacija izgleda na ekranu, od kojih se objekata sastoji te kako su oni raspoređeni na zaslonu. Druga cjelina se odnosi na interakciju korisnika sa aplikacijom, to jest omogućavanje da aplikacija vrati neku informaciju korisniku koji je koristi.

U ovom seminarskom radu cilj nam je razviti mobilnu aplikaciju za mobilni uređaj koji koristi operacijski sustav Android. Alat koji ćemo pri tome koristiti se zove Android Studio koji radi sa XML i Java jezikom. Jezik koji se odnosi na layout, to jest sam izgled aplikacije je XML, dok je Java usmjerena na interakciju sa samom aplikacijom.

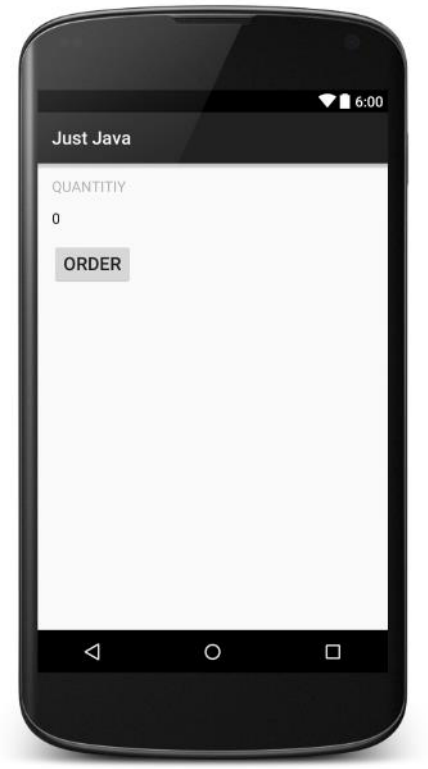
2. LAYOUT I VIEW

Prije početka same izrade layout-a kojeg želimo moramo imati bar donekle ideju kako bi on trebao izgledati i čemu će on služiti na kraju. Kako bi mogli razumjeti što je to točno layout prvo je potrebno objasniti što je „View“.

Svaka aplikacija se sastoji od više view-a, možemo reći da je layout grupa view-a i njegova je uloga pozicioniranje pojedinih view-a na samom zaslonu uređaja.

„View“ je ono što vidimo na korisničkom sučelju (User Interface), to jest na samom zaslonu mobilnog uređaja. Neki od najosnovnijih view-a su „TextView“, „ImageView“ i „Button“. Kao što im i samo ime govori TextView se odnosi na tekst, ImageView na slike, dok se Button odnosi na gumbove koje je moguće pritisnuti.

Slika 1.Prikaz view-a na zaslonu



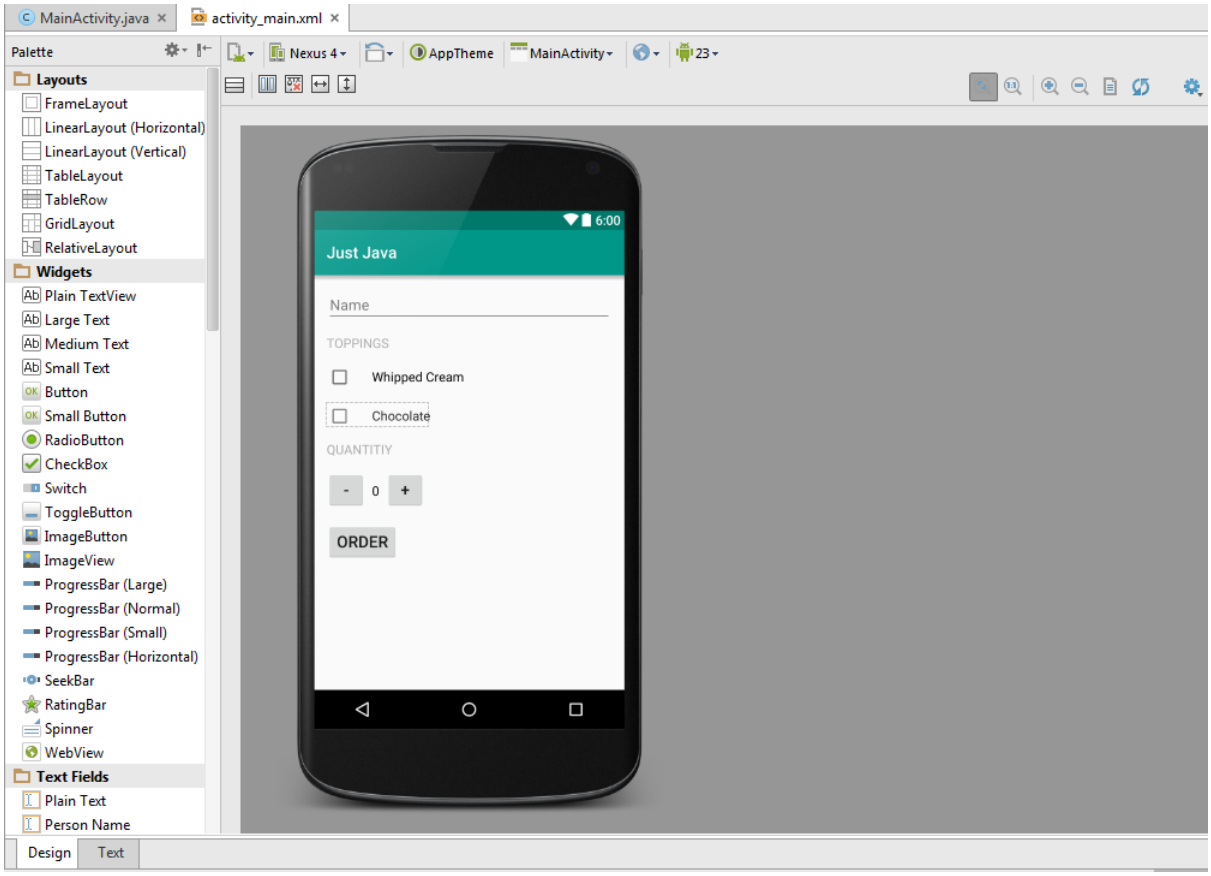
Izvor: vlastiti

Primjer služi kako bi mogli prepoznati koliko view-a se nalazi na zaslonu i o kojim se view-ima radi. Na slici možemo vidjeti TextView čiji tekst glasi “QUANTITY”, još jedan TextView čiji tekst glasi “0” te jedan gumb na kojem piše “ORDER”.

Kada znamo raspoznati pojedine view-e te odabrati one koji su nam potrebni da bi realizirali svoju aplikaciju, potrebno ih je znati i definirati u samom kodu, to jest u XML jeziku u Android Studio.

Dobra stvar kod Android Studio je ta što zapravo ne moramo ni znati sami definirati view da bi ga koristili. Postoji tab "Design" kojeg ako otvorimo dobijemo nešto slično ovome:

Slika 2. Prikaz taba Design u Android Studio



Izvor: vlastiti

Kao što vidimo lijevo od slike mobilnog uređaja imamo velik izbor view-a, kako bi ih koristili možemo jednostavno mišem odvući željeni view na zaslon mobilnog uređaja gdje želimo i u samom tekstu koda će taj view biti automatski dodan s raznim atributima.

Iako ovo izgleda puno jednostavnije od pisanja samom koda za svaki pojedini view, jako je važno poznavati kod view-a kojeg koristimo kako bi mu mogli mijenjati atribute i slično.

2.1. Definiranje View-a

Svaki view je definiran raznim atributima. Kada napravimo novi view svaki atribut ima neku početnu vrijednost koja je pre definirana, međutim svaki atribut je moguće jednostavno izmijeniti po želji programera.

Najbitniji dio definiranja i pisanja koda view-a te njegovih atributa je poštivanje pravila pisanja u XML jeziku. Računalo ne razmišlja kao čovjek pa je potrebno da je sintaksa u potpunosti točna, na primjer pri definiranju viewa za tekst moramo napisati TextView tako da su slovo T i V velikim slovom, u suprotnom će nam se javiti error i program neće raditi.

Primjer TextView-a

```
<TextView
android:text="Dobar dan"
android:background="@android:color/darker_gray"
android:layout_width="150dp"
android:layout_height="45dp"/>
```

Prvom linijom koda "<TextView" se otvara novi view za pisanje teksta, nadalje ide blok naredbi koji su zapravo atributi ovog view-a, te na kraju vidimo "/>" koji označava zatvaranje bloka naredbi i kraj TextView-a. Umjesto korištenog teksta za zatvaranje, moguće je još koristiti i " </TextView>" koji ima istu ulogu.

Unutar TextView-a su attributi koji su označeni plavom bojom njihova vrijednost se postavlja znakom =, a vrijednost koja se pridodaje atributu mora biti unutar navodnih znakova kako bi radilo.

Atributi redoslijedom odozgo prema dolje imaju sljedeća značenja:

1. Sam tekst koji želimo prikazati na ekranu (android:text)
2. Boja pozadine iza teksta (android:background)
3. Širina koju želimo dozvoliti da tekst zauzima na ekranu (android:layout_width)
4. Visina koju želimo dozvoliti da tekst zauzima na ekranu (android:layout_height)

Boja sa može mijenjati i na način da se u navodne znakove stavi šifra određene boje npr. #FFF000.

Kako ne bi bilo problema pri korištenju aplikacija na uređajima različitih veličina i rezolucija, za postavljanje visine i širine se koristi vrijednost dp (Density Independent Pixels). Ona služi tome da aplikacija i sam tekst i ostale komponente aplikacije izgledaju isto na različitim uređajima, to jest da su na svakom uređaju komponente jednake veličine neovisno o rezoluciji zaslona uređaja i veličini samog uređaja.

Isti koncept otvaranja i zatvaranja koristi i svaki drugi view. Prikazan je još i primjer za ImageView kako bi objasnili različite atribute i kako funkcioniraju.

Primjer ImageView

```
<ImageView
    android:src="@drawable/androidparty"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:scaleType="centerCrop"/>
```

Novi atribut koji se koristi je „android:src“ pomoću kojeg se odabire sliku naziva „cake“ iz mape „drawable“, na mapu se referenciramo putem simbola „@“.

Umjesto definiranja visine i širine slike, daje joj se vrijednost “wrap_content” kojom se slika prikazuje u njenom izvornom obliku. To znači da ako je slika dimenzija 200x200 pixela u toliko će pixela biti prikazana i na samom zaslonu, međutim slika može biti i veče rezolucije nego zaslon pa ona nije u potpunosti prikazana. Sljedeći atribut „android:scaleType“, čija je vrijednost postavljena na “centerCrop” pomaže pri otklanjanju tog problema. Pomoću te naredbe smanjila se veličina slike ali zadržala kvalitetu, te je sada slika prikazana preko cijelog zaslona.

2.2. View grupe

Do sada je na zaslonu bio samo jedan View, bilo da je to TextView ili ImageView. Kada bi se kopirao kod za jedan view i zalijepio ispod tog koda moglo bi se očekivati da se na zaslonu pokažu dva ista view-a. Međutim tome nije tako, kada bi to napravili javila bi se greška. Za prikazivanje više view-a na jednom zaslonu koriste se tako zvane „View groups“.

View grupe funkcioniraju slično kao i sam view, isto kao i oni imaju visinu, širini, boju pozadine i slično. Razlika je u tome što je view grupa širi pojam i može se reći da se grupa sastoji od više manjih view-a to jest ImageView, TextView i slično.

Primjer View Grupe

```
<LinearLayout
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:orientation="vertical">

<TextView
android:text="Dobar dan"
android:background="@android:color/darker_gray"
android:layout_width="150dp"
android:layout_height="45dp"/>

<TextView
android:text="Dobar dan"
android:background="@android:color/darker_gray"
android:layout_width="150dp"
android:layout_height="45dp"/>

</LinearLayout>
```

Unutar grupe (LinearLayout) su postavljena dva TextView-a. Grupa je otvorena na početku koda sa “<LinearLayout”, a zatvorena je tek na kraju drugog TextView-a sa “</LinearLayout>”. Sam layout još nazivamo i “parent” kao roditelj, a view “child” kao dijete.

Dvije osnovne vrste view grupa :

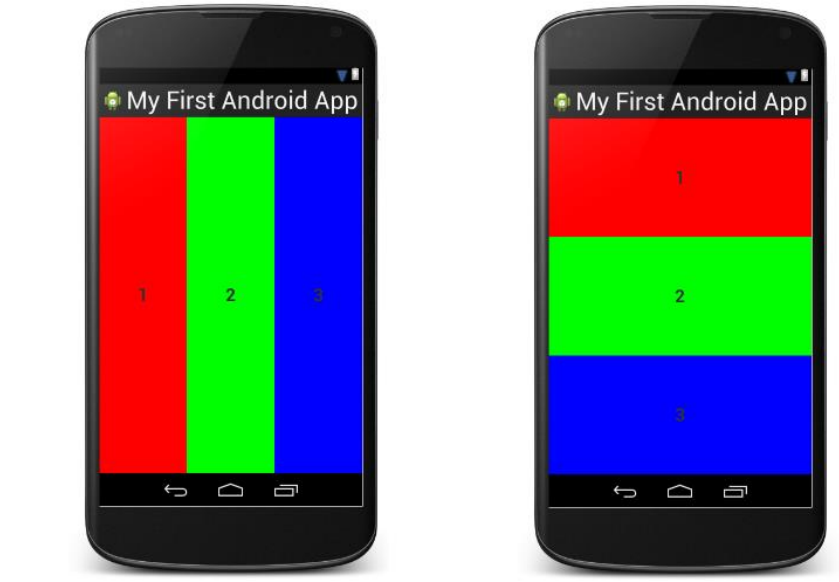
1. Linear Layout
2. Realtive Layout

Razlika između ove dvije vrste grupa je u tome kako se view-i unutar grupe pozicioniraju na samom ekranu to jest u samoj grupi.

Linearna grupa ima opciju vertikalne i horizontalne orijentacije. Što znači da će view-i unutar grupe biti poredani ili jedan do drugoga, ili jedan ispod drugoga. U primjeru je prikazan kod za vertikalnu orijentaciju, koji možemo iščitati iz sljedeće linije koda „android:orientation=”vertical”“.

Relative layout daje malo veće mogućnosti, pa se na primjer view može pozicionirati u odnosu na grupu (npr. pozicionirati se na dno grupe, ili odmaknuti od kojeg ruba grupe želimo i koliko), također view-i se mogu pozicionirati i u odnosu na druge view-e u grupi.

Slika 3. Linearni layout



Izvor: http://android4beginners.com/wp-content/uploads/2013/07/android_studio_linear_layout_example-600x393.png

Da bi se view-i unutar grupe prilagodili visini i širini samog pogleda možemo koristi atribut “match_parent” za atribut visine(height) ako programer želi da pogled bude jednake visine kao i grupa (lijeva slika) ili za atribut širine(width) ako želi da pogled bude jednake visine kao i grupa (desna slika). Također postoji mogućnost da se rasporede view-e unutar grupe

tako da svaki view zauzima jednako mjesta kao i drugi. To se radi uvođenjem novog atributa layout_weight. Za vertikalnu orijentaciju atributu širine ili „layout_width“ pridodaje se vrijednost “match parent”, atribut visine ili layout_height postavlja se na vrijednost “0dp” te se uvodi vrijednost „layout_weight“ i njegova se vrijednost postavlje na “1” i tako za svaki pogled ako se želim da svaki zauzima jednako mjesta na zaslonu, kao na desnoj slici.

Raspodjela prostora za view-e radi na principu razlomaka. Atribut weight može se postaviti na bilo koji broj.

Primjer: Imamo 3 view-a, jednom je definiram weight 1, drugom 2 i trećem 3. Raspodjela ekrana radi na način da se uzme atribut weight svakog view-a posebno i podijeli ga sa zbrojem svih weight atributa. U ovom primjeru bi prvi view dobio 1/6 zaslona, drugi 2/6 zaslona, a treći 3/6.

Slika 4. Relativni layout

Izvor: vlastiti

Na ovom primjeru se nalazi jedna slika to jest ImageView kao pozadina, te dva TextView-a. View-ovi su smješteni relativno na grupu, na primjer atributi za TextView čiji tekst glasi ”From Alen” što se tiče pozicioniranja bi izgledao ovako:

```
android:layout_alignParentBottom="true"
android:layout_alignParentRight="true"
```

Postoji još mnogo atributa za pozicioniranje koji se mogu koristit za relativni layout, ali važno je znati da svi atributi mogu imati samo dvije vrijednosti a to su “true” ili “false”, te da je vrijednost svakog atributa pre definirana na “False”. Što se tiče Android-a svaki novi view-a koji si napravi biti će automatski pozicionirati u gornji lijevi kut.

Za razliku od linearnog layouta u relativnom se view-i mogu pozicionirati i u odnosu na druge view-e. Kako bi se to moglo napraviti view-u je prvo potrebno dati ime ili ID preko kojeg se računalu kaže na koji točno view se misli kada se piše kod. To se radi dodavanjem novog atributa čiji kod bi izgledao otprilike ovako:“ android.id=”@+id/Jedan” „. U ovom slučaju ime view-a će biti Jedan, ime se može sastojati od brojeva i slova, nesmije imati razmak i nesmije sadržavati ne tipične znakove.

Ukoliko se želi pozicionirati jedan view u odnosu na view koji je prethodno definiran to bi izgledalo ovako:“ android:toLeftOf=”@id/Jedan” „.Ovdje nema znaka „+“ između „@“ i „id“, razlog tome je taj što se znak „+“ koristi samo kada po prvi put definiramo ID nekog view-a.

Primjer koda za relativni layout prikazan na prethodnoj slici:

```
<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"
    tools:context=".MainActivity">

    <ImageView android:src="@drawable/androidparty"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"/>

    <TextView android:text="Happy Birthday Patrik!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:textSize="36sp"
        android:fontFamily="sans-serif-light"
        android:textColor="@android:color/white"
        android:padding="16dp"/>

    <TextView android:text="From Alen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:textSize="36sp"
        android:fontFamily="sans-serif-light"
        android:textColor="@android:color/white"
        android:padding="16dp"/>

</RelativeLayout>
```

Korišten je relativni layout, kako bi se tekst “Happy Birthday Patrik” mogao staviti u gornji lijevi ugao, te teks “From Alen” u donji desni ugao. Međutim za tekst “Happy Birthday Patrik” nisu se morali dodavati atributi jer kao što je ranije rečeno po default-u se view postavlja u gornji lijevi ugao. U kodu se vide i neki novi atributi, na primjer „padding“ koji služi za odmicanje pogleda od svakog ugla, u ovom slučaju za 16dp. Također važno je napomenuti da je u kodu važan i sam redoslijed view-a. ImageView je u kodu prvi definiran, razlog tome je što to znači da će se svaki sljedeći pogled prikazati preko prethodnog. Na primjer kada bi TextView u kojem piše “From Alen” u kodu bio ispred ImageView-a koji sadrži sliku, taj teks se ne bi mogao vidjeti jer bi se nalazio ispod slike.

Kada se konačno odluči kakav layout se koristi i koji će se view-e sve koristiti za aplikaciju najbolje je prvo definirati same view-e, postaviti ih na željena mjesta, a na kraju odabrati stil i izgled pojedinih pogleda.

3. JAVA

Java je objektno orijentirani programski jezik koji su razvili James Gosling, Patrick Naughton i drugi inženjeri u tvrtci Sun Microsystems. Razvoj je počeo 1991, kao dio projekta Green, a objavljen je u studenom 1995.

Tvrtka Sun posjeduje trademark na ime Java, ali samo okruženje je moguće bez plaćanja skinuti sa Sunovih internet poslužitelja.

Velika prednost u odnosu na većinu dotadašnjih programskih jezika je to što se programi pisani u Javi mogu izvoditi bez preinaka na svim operativnim sustavima za koje postoji JVM (Java Virtual Machine), dok je klasične programe pisane primjerice u C-u potrebno prilagođavati platformi(Operacijskom sustavu) na kojem se izvode.

Time i bogatim skupom klasa za rad s mrežnim komunikacijama u jednom trenutku je Java bila najbolji izbor za široku lepezu mogućih aplikacija. Microsoft je stoga razvio svoj C# i .NET platformu kao odgovor na open source alternative.

Java je jedan od najkorištenijih programskih jezika. Procjene i izvješća o broju korisnika kreću se od gotovo 7 do preko 10 milijuna.[1][2][3]

Na današnjem tržištu, Java se koristi široko i dosljedno tamo gdje preteže brzina razvoja programskog sustava nad zahtjevima do brzine rada programa. Iako inspirirana jezikom C, Java pruža bolji stupanj sigurnosti i pouzdanosti zahvaljujući VM-u i hermetički zatvorenom okolišu u kome svaki program operira: na Javi se brže razvija program s manje pogrešaka.

Upravo zbog toga je popularna za razvoj programa na mobilnim telefonima i kod financijskih kompanija. Javlja se kao osnovni jezik za programiranje Googleovog sustava Android.

Javu koristimo kako bi naša aplikacija bila nešto više od samih slika i teksta, pomoću Jave aplikaciju ćemo učiniti interaktivnom. To znači da ćemo po prvi puta od svoje aplikacije dobiti neku output. No prije nego što krenemo sa modificiranjem koda potrebno je znati neke osnove Java programskog jezika.[1]

[1] https://hr.wikipedia.org/wiki/Java_(programski_jezik)

3.1. Imena i varijable

Imena su ključna za programiranje. U programima, imena se koriste za pozivanje različitih stvari. Da bi mogao koristiti te stvari, programer mora razumjeti pravila davanja imena i pravila korištenja imena. Zapravo, programer mora znati sintaksu i semantiku imena.

Premas intaksnim pravilima Jave, ime je niz od jednog ili više karaktera. Mora počinjati slovom i mora biti u potpunosti sastavljeno od slova, brojeva i donje crte "_". Nekoliko primjera ispravnih imena: N, n, rate, x15, quite_a_long_name, HelloWorld.

Velika i mala slova smatraju se različitima pa su: HelloWorld, helloworld, HELLOWORLD i HElloWorLD sasvim različita imena.

Neka imena rezervirana su za posebne namjene u Javi i programer ih ne može koristiti za druge namjene. Rezervirane riječi uključuju: class, public, static, if, else, while i nekoliko desetaka drugih riječi.

Java koristi Unicode skup karaktera koji obuhvaća tisuće znakova različitih jezika i pisama.

Varijable

Programi upravljaju podacima spremljenim u memoriji. U strojnom jeziku podaci se mogu pozivati samo pozivom numeričke adrese memorijske lokacije na kojoj je podatak spremljen. U Javi se za poziv podataka koriste imena umjesto brojeva. Programer treba zapamtiti samo ime, a računalo vodi računa o tome gdje je stvarno podatak smješten. Ovakvo ime koje služi za pozivanje podataka spremljenih u memoriji se zove varijabla.

Točno gledano, varijabla nije ime samog podatka nego mjesta u memoriji koje čuva podatke. O varijabli treba razmišljati kao o kutiji u koju se spremaju podaci za kasnije korištenje. Varijabla se odnosi izravno na kutiju i neizravno na podatak. Budući se podaci u kutiji mogu mijenjati za vrijeme izvršavanja programa, varijabla će ukazivati na različite vrijednosti ali uvijek na istu kutiju. Zablude mogu nastati, pogotovo kod neiskusnih programera, jer

varijabla kad se koristi na jedan način ukazuje na kutiju, a korištena na drugi način odnosi se na podatak.

Jedini način na koji se u Javi može spremiti podatak u varijablu je korištenjem naredbe pridjeljivanja vrijednosti (assignment statement) u obliku:
varijabla = izraz;

Gdje izraz predstavlja bilo što što se odnosi na neku vrijednost ili računa neku vrijednost. Kad računalo za vrijeme izvođenje naiđe na naredbu pridjeljivanja, procjenjuje izraz i rezultat sprema u varijablu.

Vidimo da kad se varijable koriste u izrazu, koristimo zapravo vrijednosti spremljene u njima. U ovom izrazu računalo radi sa vrijednostima.

Varijable u Javi su takve da spremaju samo jednu određenu vrstu podataka i nijednu drugu. Svako kršenje ovog pravila kompiler će smatrati sintaksnom greškom. Kaže se da je Java "strogo pisani" jezik jer nameće ovo pravilo. Postoji osam tzv. primitivnih tipova ugrađenih u Javu. Primitivni tipovi su: byte, short, int, long, float, double, char, Boolean.

Prve četiri vrste spremaju cijele brojeve, a razlikuju se po rasponu brojeva koje mogu spremiti. Float i double spremaju realne brojeve i razlikuju se po rasponu i preciznosti. Char sprema jedan znak iz Unicode skupa, a boolean jednu od dvije logičke vrijednosti (true ili false).

Slika 5. Primitivne varijable

označava da slijedi hexdecimalni kod određenog slova

	tip	sadrži	osnovna vrijednost	veličina	min. vrijednost	maks. vrijednost
logički	boolean	true ili false	false	1 bit	N.A	N.A.
cjelo-brojni	char	Unicode character	\u0000	16 bits	\u0000	\uFFFF
	byte	signed integer	0	8 bits	-128	127
	short	signed integer	0	16 bits	-32768	32767
	int	signed integer	0	32 bits	-2147,483,648	2147,483,647
	long	signed integer	0	64 bits	-9,223,372,03 6,854,775,808	9,223,372,036,85 4,775,807
broj s pomičnim zarezom	float	IEEE 754 floating point	0.0	32 bits	-3.40292347E+38	3.40292347E+38
	double	IEEE 754 floating point	0.0	64 bits	-1.7976931E+308	1.7976831E+308

Izvor: http://laris.fesb.hr/java/varijable.htm

Varijabla se može koristit u programu samo ako je prethodno deklarirana. Naredba deklariranja varijable (variable declaration statement) se koristit da bi se varijabla deklarirala i pridijelilo joj se ime. Kada računalo izvršava naredbu deklaracije varijable, rezervira dio memorije za tu varijablu i povezuje ime varijable s tim dijelom memorije. Jednostavni oblik naredbe deklaracije je: tip_varijable ime_varijable; npr. int brojac;[2]

3.2. Klase i metode

U Javi, sve metode i varijable postoje unutar klasa ili objekata (instance klase). Java ne podržava globalne metode i varijable. Stoga kostur svakog Java programa čine definicije klasa.

[2] http://laris.fesb.hr/java/varijable.htm

Početna točka svake Java aplikacije je njena main metoda. Pri pokretanju aplikacije s Java interpreterom definira se ime klase koju se želi izvršiti. Interpreter poziva main metodu iz te klase. Main metoda upravlja tokom programa, pridjeljuje potrebne resurse i pokreće druge metode potrebne za ispravno izvršavanje programa.

Definiranje klase

Crvena linija u sljedećem listingu započinje blok koji definira klasu:

```
/**
 * HelloWorldApp klasa implementira aplikaciju koja
 * ispisuje "Hello World!" na standardni izlaz.
 */
public class HelloWorldApp {
    public static void main(String[] args) {
        // Ispiši "Hello World!"
        System.out.println("Hello World!"); }}
```

Class - osnovni tvorbeni blok objektno orijentiranog jezika kao što je Java, je uzorak koji opisuje podatke i ponašanje povezano s instancama (instances) te klase. Kad stvorite instancu klase, stvorili ste objekt koji izgleda i ponaša se kao i druge instance te iste klase.

Podaci vezani uz klasu ili objekt su spremljeni u varijablama, a ponašanje vezano uz klase ili objekte se opisuje pomoću metoda. Metode su slične funkcijama ili procedurama u proceduralnim jezicima kao što je C.

Uobičajeni primjer iz svijeta programiranja je klasa koja predstavlja pravokutnik. Klasa sadrži varijable s položajem, širinom i visinom pravokutnika. Klasa također može sadržavati i

metodu koja izračunava površinu pravokutnika. Jedna instanca klase pravokutnika sadržava sve podatke o tom određenom pravokutniku, konkretan položaj, surinu i visinu i predstavlja jedan objekt, a primjenom metode na njega možemo dobiti njegovu površinu što u konkretnom slučaju može predstavljati npr. površinu ureda, veličinu ploče i sl. ovisno o tome što konkretni objekt predstavlja. Najjednostavniji oblik definicije klase u Javi je:

```
class ime_klase {

...}
```

Ključna riječ class započinje definiciju klase za klasu koja se zove ime_klase. Varijable i metode jedne klase su obuhvaćene vitičastim zagradama koje označavaju početak i kraj bloka dafinicije klase. Aplikacija "HelloWorld" nema varijabli i ima samo metodu main.

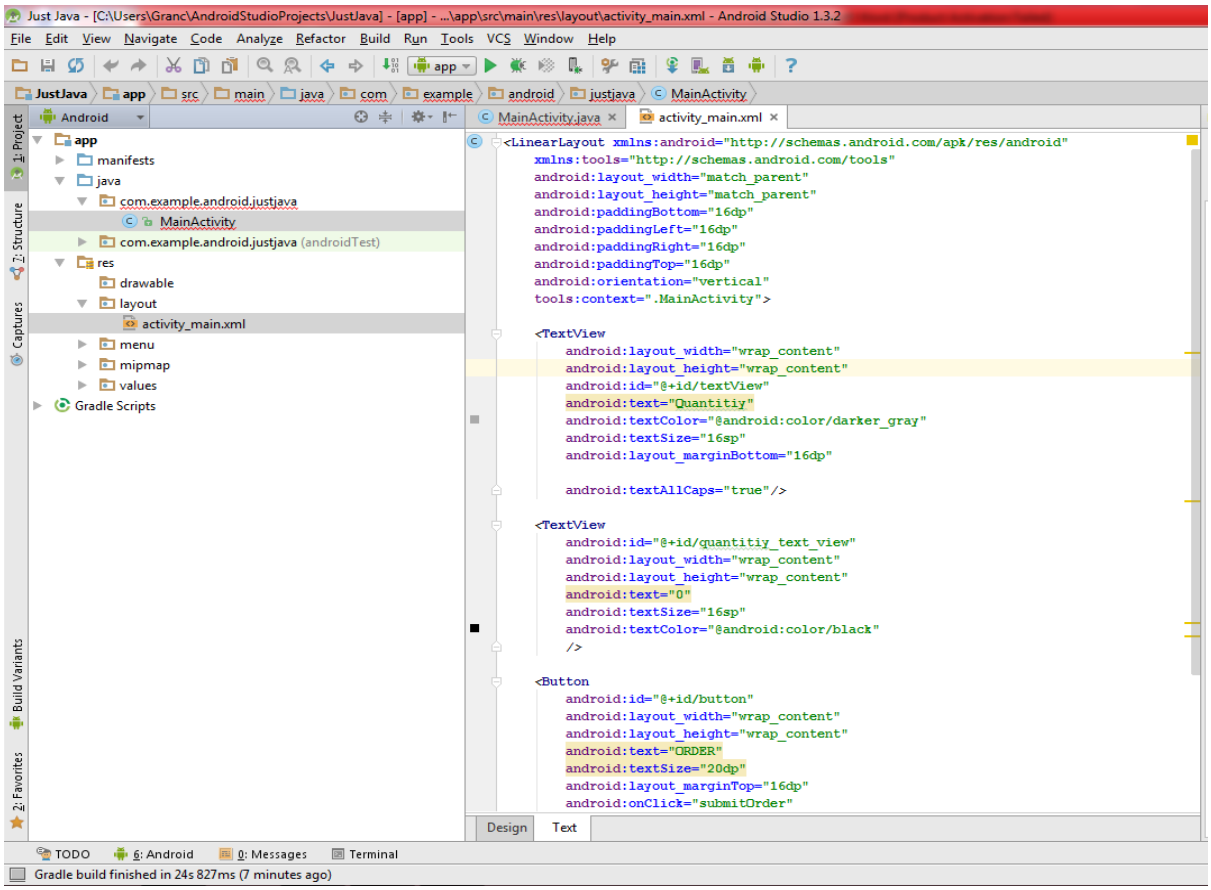
Ovakav princip možemo primjeniti i u stvarnom životu. Na primjer zamislimo da u aplikaciji moramo baratati s podacima o automobilima. Postoji mnogo vrsta automobila i nema smisla za svaki automobil definirati novu podatkovnu strukturu, jer svi oni imaju brojne zajedničke atribute (godina proizvodnje, boja, marka i sl.). Najjednostavnije je definirati klasu koja ima sve te atribute i automobile definirati kao objekte iz te klase. U slučaju da pojedini automobili imaju potrebu za dodatnim atributima možemo definirati novu klasu koja nasljeđuje ovu prvu i sadrži varijable za te dodatne atribute pa za takve slučajeve konstruiramo objekte nove klase koji naslijediti sve atribute i metode stare i uz njih imati i nove.[3]

3.3. Interakcija pomoću metoda

Do sada smo na zaslonu imali samo tekst i slike, sljedeći veliki korak je da aplikacija postane interaktivna. Kako bi to ostvarili moramo se odmaknuti od XML jezika u kojem smo gradili layout i početi pisati kod u Java jeziku.

[3] prema http://laris.fesb.hr/java/class.htm

Slika 6. Android Studio



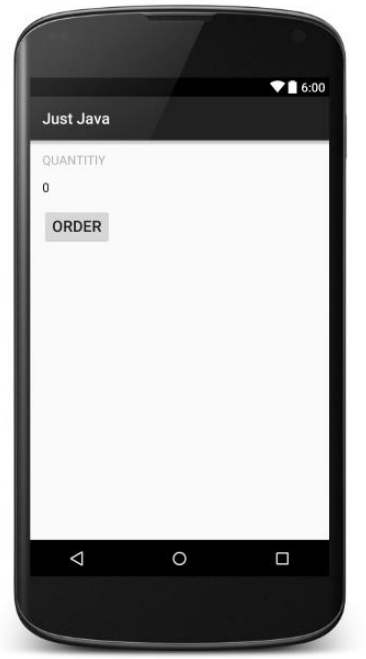
Izvor: vlastiti

Na slici je otvoren program Android Studio u kojem smo do sada pisali kod,kao što vidimo lijevo od samog koda su označene dvije linije u kojima piše „MainActivity“ i „activity_main.xml“. Ta imena nisu fiksna već ih sami definiramo pri otvaranju novog projekta. Do sada smo kod pisali samo u activity_main.xml, međutim kako bi aplikacija postala interaktivna potrebno je otvoriti MainActivity, koji ima ekstenziju „.java“.

Prije nego što krenemo pričati o MainActivity-u i kako da aplikacija postane interaktivna uočimo zadnju liniju koda koja glasi „**android:onClick="submitOrder"**“ koja će se kasnije koristiti za postizanje interakcije između uređaja i korisnika

Kako bismo lakše i bolje razumjeli kako funkcionira novi atribut i MainActivity.java potrebno je imati predodžbu što zapravo želimo i kako bi aplikacija trebala funkcionirati.

Slika 7. Izgled aplikacije



Izvor: vlastiti

Ovako bi na zaslonu trebao izgledati kod koji je prikazan na prethodnoj slici. Cilj je postići to da se broj „0“ promjeni u neki drugi željeni broj bez da mijenjamo sam tekst u tom view-u. To želimo postići klikom na gumb u kojem piše “ORDER”.

Novim atributom „**android:onClick="submitOrder"**“ ubiti pozivamo metodu koja radi upravo to. Metode smo definirali u MainActivity.java.

Ovako izgleda kod MainActivity:

```
public void submitOrder(View view) {
    display(1);}

private void display(int number) {
    TextView quantityTextView = (TextView) findViewById(
            R.id.quantity_text_view);
    quantityTextView.setText("" + number);  }}
```

Kao što vidimo opet nam se pojavljuje „submitOrder“ kojeg smo koristili za atribut na gumbu. submitOrder je naziv metode, također ispod vidimo još jednu metodu koja se zove

„display()“. U metodi display je rečeno da na mjestu gdje je trenutno „0“ prikaže broj koji mi upišemo. Metoda submitOrder poziva metodu display() i u zagradu pišemo broj koji želimo da se prikaže kada kliknemo na gumb “ORDER”. Znači pritiskom na gumb “ORDER” pozivamo metodu submitOrder koja izvršava kod metode display kako bi se promijenila vrjedost u TextView-u kojem smo dali id „quantity_text_view“. U ovom slučaju umjesto broja “0” bi se na zaslonu prikazao broj “1”.

Bitno je znati da se u zagrade može napisati bilo kakav matematički izraz i računalo će samo izračunati i prikazati rezultat.

Kao što na primjeru vidimo metode definiramo na način da na prvo mjesto pišemo tko će sve imati pristup toj metodi, ako je metoda definirana kao „private“ onda se može koristiti samo u okviru klase u kojoj je definirana za razliku od public metode koju mogu koristiti svi. Nakon odlučivanja tko ce imati pristup metodi definiramo kakav tip podatak metoda vrača. U slučaju da piše „void“ znači da metoda ne vrača nikakav tip podatka, umjesto void mogu još na primjer biti „int“ koji vrača broj ili „String“ koji vrača neke znakove, teks ili brojke. Nakon toga metodi dajemo ime u ovom slučaju display i submitOrder. Te na kraju u zagradama pišemo parametre koje želimo dati metodi.

Metode i varijable klase vezane su uz određenu klasu. Sistem na kojem se izvršava aplikacija postavlja varijablu klase jednom za klasu, bez obzira na broj instanci te klase. Varijablama i metodama klase se pristupa preko klase. Metode i varijable instance su vezane uz određeni objekt (instancu klase). Prilikom svakog stvaranja objekta, novi objekt dobiva kopiju svih varijabli instance definiranih u toj klasi. Varijablama i metodama primjera pristupa se preko objekata. [4]

[4] http://laris.fesb.hr/java/objekti.htm

4. OBJEKTNO ORIJENTIRANO PROGRAMIRANJE

Objektno orijentirano programiranje je pokušaj približavanja modela programa načinu ljudskog razmišljanja.

Kod starog načina programiranja, programer je morao pronaći računalnu zadaću koju treba izvršiti da bi se riješilo neki problem. Programiranje se zatim sastojalo od pronalaženja niza naredbi koje bi izvršile taj zadatak.

U osnovi objektno orijentiranog programiranja, umjesto zadaća nalazimo objekte - jedinice koje imaju svoje ponašanje, drže podatke i mogu međusobno djelovati. Programiranje se sastoji od oblikovanja skupa objekata koji na neki način opisuju problem koji treba riješiti. Softverski objekti u programu predstavljaju stvarne ili zamišljene jedinice u području problema. Na ovaj način razvoj programa trebao bi biti prirodniji i stoga lakši za postavljanje i razumijevanje.

Do neke granice, objektno orijentirano programiranje je samo promjena točke gledanja, u okvirima standardnog programiranja objekt se može promatrati kao skup varijabli i nekih potprograma za upravljanje tim varijablama. Zapravo, moguće je koristiti objektno orijentirano programiranje u bilo kojem programskom jeziku. Naravno postoji velika razlika između programskih jezika u kojima je objektno orijentirano programiranje moguće i onih koji ga uistinu aktivno podržavaju. Objektno orijentirani programski jezici poput Jave imaju niz mogućnosti koje ih čine različitim od standardnih programskih jezika. Ispravno razmišljanje je osnova za korištenje tih mogućnosti.

Objekti su usko vezani uz klase. Potrebno je najprije razjasniti razlike između objekata i klasa. Klase, zapravo njihovi nestatički dijelovi opisuju objekte. Uobičajeno je reći da objekti pripadaju klasama. Sa programerskog gledišta natočnije bi bilo reći da se klase koriste za opisivanje objekata. Nestatički dijelovi klasa određuju koje će varijable i potprogrami biti sadržani u objektima. Ovo je ujedno i dio objašnjenja po čemu se objekti razlikuju od klasa: objekti se stvaraju i uništavaju tijekom izvođenja programa, a korištenjem jedne klase može biti stvoren neograničen broj objekata. Primjer klase koja se može koristiti za spremanje podataka o korisniku programa:

```
class KorisnikoviPodaci {

        static String ime;
        static int godine;}
```

U programu koji koristi ovu klasu postoji samo jedan primjerak svih varijabli u klasi: KorisnikoviPodaci.ime i KorisnikoviPodaci.godine. Klasa KorisnikoviPodaci i podaci koje sadržava postoje samo dok se program izodi.

Sličan primjer, s nestatičkim varijablama:

```
class PodaciIgraca {
        String ime;
        int godine;}
```

U ovom slučaju nema varijabli PodaciIgraca.ime i PodaciIgraca.godine, jer ime i godine nisu statički članovi klase PodaciIgraca. Dakle, u klasi nema ništa osim mogućnosti stvaranja objekata. To su, zapravo, ogromne mogućnosti jer je broj objekata koji se mogu stvoriti neograničen, a svaki od tih objekata ima svoje varijable ime i godine. Program može pomoću ove klase spremati podatke o svim igračima. Kad novi igrač uđe u igru, kreira se novi objektPodaciIgraca koji predstavlja tog igrača i sadrži njegove podatke. Kad igrač napusti igru, objekt koji ga predstavlja može biti uništen. Ovakav sistem sa objektima se koristi za dinamičko opisivanje događanja u igri, što nije oguće uraditi pomoću statičkih varijabli.

Objekt koji pripada klasi naziva se instance te klase. Varijable koje taj objekt sadrži nazivaju se varijable instance. Potprogrami koje objekt sadrži nazivaju se metode instance. Na primjer, ako se gore definirana klasa PodaciIgraca koristi za kreiranje objekta, tada je taj objekt instanca klase PodaciIgraca, a ime i godine u tom su objektu varijable instance. Važno je zapamtiti da klasa objekta definira tip varijable instance; stvarni podaci sadržani su u pojedinom objektu, ne u klasi. Stoga, svaki objekt ima svoje podatke.

Aplet koji skrola poruku po web stranici može, npr., sadržavati metodu scroll(). Budući da je aplet objekt, ova metoda je metoda instance ovoga apleta. Izvorni kod metode je u klasi koja

je korištenja za kreiranje apleta. Ipak ispravnije je reći da metoda instance pripada objektu a ne klasi. Nestatičke metode u klasi samo određuju koje će metode instance sadržavati objekti kreirani iz klase. scroll() metode u dva različita apleta rade istu stvar, tj. obje skrolaju poruke po ekranu. Stvarna razlika tih dviju metoda je u porukama koje skroliraju, one se mogu razlikovati. Dakle, metode određuju vrstu ponašanja objekata, ali to ponašanje se razlikuje između objekata po vrijednostima varijabli instance.

Očito je da su statički i nestatički dijelovi klase vrlo različite stvari i sa vrlo različitim namjenama. Mnogo klasa sadrži isključivo statičke ili nestatičke članove, iako je moguće miješanje statičkih i nestatičkih članova u jednoj klasi. Statičke varijable i potprogrami u klasama se zovu varijable klase, odnosno metode klase, jer pripadaju klasi a ne instancama te klase.[5]

4.1. Konstruktori

Konstruktori su potprogrami posebne vrste. Nisu metode instance jer ne pripadaju objektima, budući da su odgovorni za stvaranje objekata postoje prije njih. Najsličniji su static potprogramima, ali nisu i ne mogu biti deklarirani kao static. Zapravo, po specifikaciji Java programskog jezika, oni i nisu članovi klasa.

Za razliku od ostalih potprograma, konstruktor se može pozvati jedino korištenjem new operatora u izrazu oblika: new ime-klase (popis-parametara) pri čemu popis-parametara može biti i prazan. Naziv izraz se koristi zato jer izračunava i vraća vrijednost, točnije poziv na objekt koji je stvoren. Najčešće se vraćeni poziv sprema u varijablu, iako je dozvoljeno koristiti poziv konstruktora i na druge načine, npr. kao parametar pri pozivu potprograma ili kao dio složenijeg izraza. Naravno, ako se poziv ne spremi u varijablu, neće postojati način na koji bi se upravo stvoreni objekt mogao pozivati.

[5] http://laris.fesb.hr/java/blokovi_instance.htm

Poziv konstruktora je složeniji od običnog poziva potprograma ili funkcije. Korisno je poznavati sve korake kroz koje računalo prolazi pri pozivu konstruktora:

1. Računalo osigurava dio neiskorištene memorije u heapu, dovoljno velik za spremanje objekta određenog tipa.
2. Inicijalizacija varijabli instance objekta. Ako su zadane početne vrijednosti, te se vrijednosti izračunavaju i spremaju u varijable, u suprotnom se varijablama pridjeljuju default vrijednosti.
3. Procjenjuju se vrijednosti stvarnih parametara u konstruktoru, te se pridjeljuju formalnim parametrima konstruktora.
4. Izvršavaju se naredbe iz tijela konstruktora, ako ih uopće ima.
5. Vraća se poziv na objekt kao vrijednost poziva konstruktora.

Krajnji rezultat svega ovoga je poziv na novi objekt, koji se može koristiti za pristup varijablama instance ili poziv metoda instance tog objekta.[6]

[6] http://laris.fesb.hr/java/konstruktor.htm

5. MOBILNA APLIKACIJA

Aplikacija koja se razvija temelji se na primjeru aplikacije koja je razvijena u sklopu e-learning tečaja „Android development for Begginers“ na stranici „www.udacity.com“.Radi se o narudžbi preko internet, to jest preciznije preko e-maila. Radi se o jednostavnoj aplikaciji koja korisniku nudi opciju da naruči određen broj kava, te ima izbor da li uz kavu želi šlag, čokoladu, oboje ili ni jedno. Cijena konačne narudžbe se mijenja ovisno o odabranim dodacima i samom broju kava koju korisnik naručuje. Korisnik bi prvo trebao upisati svoje ime, zatim odabrati dodatke i količinu kave. Po završetku naručuje kavu pritiskom na gumb. Kada se gumb pritisne aplikacija korisnika automatski šalje na email u kojem pišu detalji narudžbe te ima opciju narudžbu i poslati.

Slika 8. Konačan izgled aplikacije

Izvor: vlastiti

Ovako aplikacija izgleda na mobilnom uređaju. Aplikacija se zove “Just Java”.

Redom je objašnjeno za što je koji view zadužen i na koji način je postignuto da aplikacija bude interaktivna.

U prvom redu je horizontalno polje preko cijele širine ekrana u kojem piše “Ime i Prezime”. Ovdje se od korisnika očekuje da upiše svoje ime. View koji se koristio je „EditText“ view.

```
<EditText
   android:id="@+id/nameField"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:hint="Ime i Prezime"
   android:inputType="textCapWords"
   android:layout_marginBottom="16dp"/>
```

Kako bi se kasnije moglo referencirati na ovaj EditText view potrebno je postaviti ID.
Atribut „hint“ prikazuje tekst “Ime i Prezime” na ekranu, taj tekst nestane pritiskom na polje.
Zanimljiv je još atribut „inputType“ čija je vrijednost “textCapWords” kojim postižemo da se prvo slovo imena i prezimena uvijek velikim slovom.
Kako bi kasnije u mail mogli prosljediti upisano ime u Mainactivity.java imamo sljedeći kod:

```
EditText fieldName = (EditText) findViewById(R.id.nameField);
String name = fieldName.getText().toString();
```

U prvoj liniji se pronalazi željeni view,a u drugoj liniji se taj tekst sprema u varijablu pod imenom“name”, koja je tipa String.
Valja napomenuti da se te dvije linije koda nalaze u metodi createOrderSummary koja se poziva iz metode submitOrder pritiskom na gumb u kojem piše “ORDER”.

TextView u kojem se nalazi tekst “TOPPINGS” i TextView u kojem se nalazi tekst “QUANTITY” nemaju nikakvih interakcija već su prisutni više radi boljeg opisa i lakšeg razumjevanja aplikacije.
Primjer TextView

```
<TextView
   android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:id="@+id/toppingsView"
    android:text="Dodaci"
    android:textColor="@android:color/darker_gray"
    android:textSize="16sp"
    android:layout_marginBottom="16dp"
    android:textAllCaps="true"/>
```

Nadalje imamo dva „ChekBox“ jedan ispod drugoga. Označavanjem svakog od njih ukupna cijena jedne kave bi se trebala povećati za 1 ukoliko je označen “Šlag” odnosno za 2 ukoliko je označen “Čokolada”. Treba naglasiti da je početna cijena jedne kave postavljena na 5 dolara.

Primjer CheckBox

```
<CheckBox
    android:id="@+id/whippedCreamCheckbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Šlag "
    android:textSize="16sp"
    android:paddingLeft="24dp"
    android:layout_marginBottom="16dp"/>
```

Radi referenciranja također se postavlja ID. U Java kodu je potrebno postaviti da se cijena mijenja ukoliko je polje označeno, te je potrebno dodati informaciju dali je korisnik naručio šlag ili čokoladu u narudžbu.
Za promjenu cijene mora se napraviti novu metodu te po prvi puta koristiti petlju, u ovom slučaju koristit će se petlja „IF“.

Primjer

```
private int calculatePrice(boolean addWhippedCream, boolean addChocolate)
{
   int basePrice=5;
   if (addWhippedCream){
      basePrice=basePrice+1; }
   if (addChocolate){
      basePrice=basePrice+2;}
```

Metodu je nazvana „calculatePrice“ i dana su joj dva parametra tipa „Boolean“. Boolean ima samo dvije vrijednosti, a to su “true” i “false”što je dovoljno za ove potrebe. Kod ubiti kaže da ako je polje „Šlag“ označeno to jest “true” neka cijeni doda 1 i ukoliko je polje „Čokolada“ označeno to jest “true” doda 2. Ukoliko polje nije označeno program samo preskače taj dio koda.

Također u metodu submitOrder se upisuje seljedeći kod:

```
CheckBox whippedCreamCheckbox = (CheckBox)
findViewById(R.id.whippedCreamCheckbox);
boolean hasWhippedCream = whippedCreamCheckbox.isChecked();
```

Kojim se provjerava dali je polje označeno te upisujemo u detalje narudžbe. Kod je isti i za polje “Čokolada” samo što se referencira na drugi ID.

Ispod TextView-a u kojem piše “Količina” nalaze se dva gumba te TextView između njih. To se može postići tako da se napravi nova linearna View grupu.

Primjer

```
<LinearLayout
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:orientation="horizontal">

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

```
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:text="-"
        android:textSize="20dp"
        android:onClick="decrement"
        />

    <TextView
        android:id="@+id/quantity_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:text="0"
        android:textSize="16sp"
        android:textColor="@android:color/black"
        />

    <Button
        android:id="@+id/button1"
        android:layout_width="48dp"
        android:layout_height="48dp"
        android:text="+"
        android:textSize="20dp"
        android:onClick="increment"
        />

</LinearLayout>
```

Pritiskom na gumb se povečava odnosno smanjuje količina koja se pokazuje u TextView-u sredini.

Količina se povečava pritiskom na gumb sa tekstom „+“, pozivanjem metode “increment”.

```
public void increment(View view)
{
    if (quantity == 100) {
        Toast.makeText(this, "Ne možete imati više od 100 šalica kave!",
Toast.LENGTH_SHORT).show();
        return;
    }
    quantity=quantity+1;
    display(quantity);
```

Također postavljen je „Toast“ ukoliko je količina 100, što znači da će se na ekranu izbaciti kratka poruka u kojoj piše “Ne možete imati više od 100 šalica kave!”. I naravno ukoliko se pritisne gumb sa znakom “-” količina koja se pokazuje će se smanjiti za 1. Metoda decrement je vezana za gumb u kojem piše „-“, slična je kao increment, samo što se Toast izbaci ukoliko korisnik pokuša naručiti manje od 1 kave.

Na kraju se poziva metoda „display“ koja je također definirana, te koristi za prikazivanje broja u TextView-u.

```
private void display(int number)
{
   TextView quantityTextView = (TextView) findViewById(
         R.id.quantity_text_view);
   quantityTextView.setText("" + number);
}
```

Prije samog gumba u kojem piše “NARUČITI” potrebno je pokazati metodu u koju su sabrani svi dosadašnji parametri, kako bi se ispisala poruka o narudžbi. Metoda je nazvana „createOrderSummary“.

```
private String createOrderSummary(String name, int price, boolean addWhippedCream,
boolean addChocolate)
{
   String priceMessage ="Ime i Prezime:" + name;
   priceMessage +="\nDodati šlag: " + addWhippedCream;
   priceMessage +="\nDodati čokoladu: " + addChocolate;
   priceMessage +="\nKoličina: " + quantity;
   priceMessage +="\nCijena $: " + price;
   priceMessage +="\nHvala vam!";
   return priceMessage;
}
```

Metoda očekuje String kao povratnu informaciju, a u zagradi se nalaze svi parametre koji su proslijeđeni ovoj metodi. Svi su parametri već korišteni osim „int price“ koji je definiran u metodi submitOrder.

```
public void submitOrder(View view) {
   CheckBox whippedCreamCheckbox = (CheckBox)
findViewById(R.id.whippedCreamCheckbox);
   boolean hasWhippedCream = whippedCreamCheckbox.isChecked();
```

```
    CheckBox chocolateCheckbox = (CheckBox) findViewById(R.id.chocolateCheckbox);
    boolean hasChocolate = chocolateCheckbox.isChecked();

    EditText fieldName = (EditText) findViewById(R.id.nameField);
    String name = fieldName.getText().toString();

    int price = calculatePrice(hasWhippedCream, hasChocolate);

    String priceMessage = createOrderSummary(name, price, hasWhippedCream,
hasChocolate);

    Intent intent = new Intent(Intent.ACTION_SENDTO);
    intent.setData(Uri.parse("mailto:"));
    intent.putExtra(Intent.EXTRA_SUBJECT, "Just Java narudžba za  " + name);
    intent.putExtra(Intent.EXTRA_TEXT, priceMessage);

    if (intent.resolveActivity(getPackageManager()) != null) {
       startActivity(intent);  }
```

U varijablu “price” se pohranjuje konačna cijena nakon eventualnog dodavanja dodataka (Šlag ili Čokolada). Također je dodan String “priceMessage” u koji se pohranjuje poruka o cjelokupnoj narudžbi. U nastavku koda se pokreče slanje email-a te se u email učitava čitav tekst narudžbe i korisnik ima mogučnost slanja email-a.

6. ZAKLJUČAK

Svaka nova aplikacija počinje sa idejom i nekom vizijom kako bi ta aplikacija trebala izgledati kada je gotova. Najbolje je početi tako da se na papiru ili u nekom programu napravi skica kako bi imali bolju ideju kako će aplikacija zapravo izgledati. Kada već otprilike znamo što i kako želimo potrebno je odrediti koji nam view-i trebaju kako bi svoju ideju prebacili na zaslon mobilnog uređaja. Kada krenemo sa programiranjem najlakše je prvo napraviti sve što smo zamislili da imamo na zaslonu i posložiti sve kako želimo. Nakon toga je potrebno shvatit kako želimo omogučiti interakciju između korisnika i same aplikacije. Najbolje je odmah postaviti atribute za ID svim objektima koje želimo da budu interaktivni ili da se mijenjaju. Nakon toga možemo dodavati metode koje su nam potrebne ili čak definirati klase ako za to ima potrebe. Poslije svakog većeg koraka bilo bi dobro provjeriti dali aplikacija funkcionira kako želimo, na taj način možemo lakše ispraviti eventualne greške. Naravno važno je naučiti i čitati poruku koja nam govori gdje se greška nalazi. Opcija da se kopira poruka sa greškom i potraži rješenje na google-u uvjek postoji i jako često se na taj način lako riješi većina problema. Kada je aplikacija konačno gotova i spremna za korištenje možemo lako doraditi njen izgled dodavanjem boja, stilova i razno raznih atributa, pa čak i mijenjati temu Android-a.

7. PRILOG A

Java izvorni kod

```
package com.example.android.justjava;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.view.View;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.text.NumberFormat;


public class MainActivity extends ActionBarActivity {
    int quantity=0;

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

    /**
     *Ova metoda se poziva kada je pritisnuto na gumb "ORDER".
     */
    public void submitOrder(View view) {
        CheckBox whippedCreamCheckbox = (CheckBox)
findViewById(R.id.whippedCreamCheckbox);
        boolean hasWhippedCream = whippedCreamCheckbox.isChecked();

        CheckBox chocolateCheckbox = (CheckBox)
findViewById(R.id.chocolateCheckbox);
        boolean hasChocolate = chocolateCheckbox.isChecked();

        EditText fieldName = (EditText) findViewById(R.id.nameField);
        String name = fieldName.getText().toString();

        int price = calculatePrice(hasWhippedCream, hasChocolate);
        String priceMessage = createOrderSummary(name, price, hasWhippedCream,
hasChocolate);

        Intent intent = new Intent(Intent.ACTION_SENDTO);
        intent.setData(Uri.parse("mailto:"));
intent.putExtra(Intent.EXTRA_SUBJECT, "Just Java narudžba za " + name);
        intent.putExtra(Intent.EXTRA_TEXT, priceMessage);

        if (intent.resolveActivity(getPackageManager()) != null) {
            startActivity(intent);


        }
    }
    private String createOrderSummary(String name, int price, boolean
addWhippedCream, boolean addChocolate)
    {
        String priceMessage ="Ime i Prezime:" + name;
        priceMessage +="\nDadati šlag: " + addWhippedCream;
        priceMessage +="\nDodati čokoladu: " + addChocolate;
```

```
        priceMessage +="\nKoličina: " + quantity;
        priceMessage +="\nCijena $: " + price;
        priceMessage +="\nHvala vam!";
        return priceMessage;


    }
    private int calculatePrice(boolean addWhippedCream, boolean addChocolate)
    {
        int basePrice=5;
        if (addWhippedCream){
            basePrice=basePrice+1;
        }
        if (addChocolate){
            basePrice=basePrice+2;
        }

        return quantity*basePrice;
    }

    /**
     *Ova metoda prikazuje broj kava koju je korisnik naručio.
     */
    private void display(int number)
    {
        TextView quantityTextView = (TextView) findViewById(
                R.id.quantity_text_view);
        quantityTextView.setText("" + number);
    }



    public void increment(View view)
    {
        if (quantity == 100) {
            // Pokazuje error poruku kao toast
            Toast.makeText(this, "Ne možete imati više od 100 šalica kave!",
Toast.LENGTH_SHORT).show();
            // Izlazi iz metode ranije jer se više ništa ne može napraviti
            return;
        }
        quantity=quantity+1;
        display(quantity);
    }
    public void decrement(View view)
    {
        if (quantity == 1) {
            // Pokazuje error poruku kao toast
            Toast.makeText(this, "Ne možete imati manje od 1 šalice kave",
Toast.LENGTH_SHORT).show();
            // Izlazi iz metode ranije jer se više ništa ne može napraviti
            return;
        }
        quantity=quantity-1;
        display(quantity);
    }

}
```

8. PRILOG B

XML kod

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

    <EditText
        android:id="@+id/nameField"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Ime i Prezime"
        android:inputType="textCapWords"
        android:layout_marginBottom="16dp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/toppingsView"
        android:text="Dodaci"
        android:textColor="@android:color/darker_gray"
        android:textSize="16sp"
        android:layout_marginBottom="16dp"
        android:textAllCaps="true"/>

    <CheckBox
        android:id="@+id/whippedCreamCheckbox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Šlag"
        android:textSize="16sp"
        android:paddingLeft="24dp"
        android:layout_marginBottom="16dp"/>

    <CheckBox
        android:id="@+id/chocolateCheckbox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Čokolada"
        android:textSize="16sp"
        android:paddingLeft="24dp"
        android:layout_marginBottom="16dp"/>


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView"
        android:text="Količina"
        android:textColor="@android:color/darker_gray"
        android:textSize="16sp"
        android:layout_marginBottom="16dp"

        android:textAllCaps="true"/>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:orientation="horizontal">

        <Button
            android:id="@+id/button2"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:text="-"
            android:textSize="20dp"
            android:onClick="decrement"
            />

        <TextView
            android:id="@+id/quantity_text_view"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="8dp"
            android:layout_marginRight="8dp"
            android:text="0"
            android:textSize="16sp"
            android:textColor="@android:color/black"
            />

        <Button
            android:id="@+id/button1"
            android:layout_width="48dp"
            android:layout_height="48dp"
            android:text="+"
            android:textSize="20dp"
            android:onClick="increment"
            />

    </LinearLayout>


    <Button
        android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="NARUČI"
        android:textSize="20dp"
        android:layout_marginTop="16dp"
        android:onClick="submitOrder"
        />


</LinearLayout>
```

9. LITERATURA

1. https://www.udacity.com, e-Learning tečaj „Android development for Begginers“
2. https://hr.wikipedia.org/wiki/Java_(programski_jezik)
3. http://laris.fesb.hr/java/varijable.htm
4. http://laris.fesb.hr/java/class.htm
5. http://laris.fesb.hr/java/blokovi_instance.htm
6. http://laris.fesb.hr/java/konstruktor.htm
7. R. Sedgewick, K. Wayne “Introduction to Programming in Java”, Addison Wesley, 2008
8. W. Savitch, F.Carrano “Java, An Introduction to Problem Solving & Programming”, Prentice Hall, 2009