Izrada i primjena razvojnih android aplikacija

Plasaj, Antonio

VELEUČILIŠTE U KARLOVCU
STROJARSKI ODJEL
MEHATRONIKA

ANTONIO PLASAJ

IZRADA I PRIMJENA RAZVOJNIH ANDROID APLIKACIJA

ZAVRŠNI RAD

Karlovac, 2016.

KARLOVAC UNIVERSITY OF APPLIED SCIENCES
MECHANICAL ENGINEERING DEPARTMENT

PROFESSIONAL UNDERGRADUATE STUDY OF MECHATRONICS

ANTONIO PLASAJ

USES AND DEVELOPMENT OF ANDROID APPLICATIONS

FINAL PAPER

Karlovac, 2016

VELEUČILIŠTE U KARLOVCU
STROJARSKI ODJEL
MEHATRONIKA

ANTONIO PLASAJ

IZRADA I PRIMJENA RAZVOJNIH ANDROID APLIKACIJA

ZAVRŠNI RAD

Mentor:
dr.sc. Adam Stančić, predavač

Karlovac, 2016

VELEUČILIŠTE U KARLOVCU
KARLOVAC UNIVERSITY OF APPLIED SCIENCES
Trg J.J. Strossmayera 9
HR-47000, Karlovac, Croatia
Tel. +385 - (0)47 - 843 – 510
Fax. +385 - (0)47 - 843 – 579

HRN EN ISO 9001:2009

VELEUČILIŠTE U KARLOVCU

Stručni studij: STROJARSTVO

Usmjerenje: MEHATRONIKA

Karlovac, 15.06.2016.

ZADATAK ZAVRŠNOG RADA

Student: ANTONIO PLASAJ

Matični broj: 0112612006

Naslov: IZRADA I PRIMJENA RAZVOJNIH ANDROID APLIKACIJA

Opis zadatka:

Nakon uvoda, u teorijskom dijelu rada pojasniti i opisati teorijsku podlogu sa definicijama sintakse, grupa, i funkcija programskih jezika Java i XML. Također opisati i prikazati način rada softvera Android Studio. Nakon pokrivenih teoretskih osnova, u eksperimentalnom dijelu treba pokazati razvoj android aplikacija za android operativne sustave, kroz niz primjera funkcionalnih i edukacijskih aplikacija sa postupkom i kompletnim programskim kodom. Na kraju rada napisati zaključak. Koristiti stručnu literaturu, i službene stranice kao referentnu dokumentaciju, uz redovite konzultacije s mentorom te rad uskladiti s Pravilnikom o pisanju završnih i diplomskih radova Veleučilišta u Karlovcu.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
15.06.2016.	20.09.2016.	28.09.2016.

Mentor:
dr.sc. Adam Stančić, predavač

Predsjednik Ispitnog povjerenstva:
Marijan Brozović, v. pred.

SAŽETAK

Tema ovog završnog rada je izrada aplikacija za android operativne sustave. S obzirom da je za razvoj android aplikacija potrebno znanje java programskog jezika, ovaj rad će se fokusirati i na javu kao programski jezik. Osim jave, koristi se i XML jezik. Softver za razvoj aplikacija koji je odabran je Android Studio. Rad pokriva sintaksu programskog jezika java, kao i funkcije, i grupe, a zatim će biti pojašnjeni konkretni primjeri koji se nadovezuju na teorijsku osnovu. Rad je namijenjen da se koristi kao priručnik kod razvoja android aplikacija.

Da se izbjegnu zabune i nepotrebne pogreške, za neke termine i funkcije će se koristiti engleski nazivi, zbog toga jer sintaksa koda također sadrži engleske termine, pa je bolje od početka koristiti engleske nazive. Kod prvog susreta sa engleskim terminom, njegovo značenje će biti pojašnjeno.

SUMMARY

The topic of this final paper is android applications development. Considering that android application development requires a sufficient level of knowledge of Java programming language, this final paper will cover its basis as well. Besides Java, XML programming language will also be used. Software used for development is Android Studio. Coverage of Java synthax, as well as functions and groups will be explained in the theoretical part of this final paper. After covering theoretical basis, there will be a set of educational and fully funcional applications which will show how to apply theoretical knowledge. This final paper is meant to be used as a manual for developers while developing applications for Android platforms.

To avoid any unnecessary confusion and eventual mistakes, most of the terms and functions will be written in its original english form, because coding synthax uses english as a base language for the programming languages. When a specific term is encountered for the first time, there will be an explanation of it.

SADRŽAJ

1. Uvod

Android je najraširenij otvoreni operacijski sustav za mobilne uređaje. Koristi se i u drugim uređajima poput TV-a, smart satova, tableta, i sličnog. Njegova zastupljenost i pristupačnost znači da android razvojni inženjeri imaju velik spektar tehnološki i softverski ostvarivih mogućnosti.

Namijenjen je za uređaje koji koriste grafičku knjižnicu (library) koja se temelji na OpenGL-u. Upravljanje bazama podataka koristi knjižnicu SQLite-a.

Danas postoji mnogo verzija androida, stoga biranje verzije androida za koju će aplikacija biti dostupna utječe na API (broj funkcija i procesa koje dopušta verzija). Što je API veći, ima više mogućnosti, ali stariji modeli nisu kompatibilni. To će imati ulogu kod odabira verzije androida.

U ovom radu će se raditi u Android studiu za Windows operativne sustave, ali uz sitne preinake se može jednako raditi i na Mac operativnom sustavu OSX, kao i na većini distribucija Linuxa.

Android studio je IDE kojeg je razvio intelliJ. IDE je skraćenica za Integrated Software Application, odnosno to je sučelje za razvoj koje se sastoji od compilera, debuggera i editora za izvorni kod (source code).

2. Teoretski dio

2.10. Programski jezik Java

Java je objektno orijentirani programski jezik, razvijen ranih 1990-ih. Razvijen je od strane tvrtke Sun Microsystems. Java je bazirana na C++-u, ali je sintaksa znatno jednostavnija. Jedan je od najkorištenijih i najpopularnijih programskih jezika u svijetu (slika 1).

Prednost jave nad C++-om je ta što je daleko prilagodljivija, što znači da kod mijenjanja operativnog sustava nije potrebno prilagođavati kod. To omogućuje JVM (Java virtual machine). Također, java ima veliku prednost nad C++-om što se tiče pouzdanosti. C++ je WOCA tip programskog jezika (Write once, compile anywhere), dok je Java WORA (Write once, run anywhere), što znači da nije potreban compiler na korisničkim uređajima.

Slika 1. Predodžba loga Jave (izvor http://fontslogo.com/java-logo-font/)

Prilikom kreiranja jezika Java određeno je 5 ciljeva koje jezik mora ostvarivati. Tih 5 ciljeva su:
-Mora biti jednostavan, objektno orijentiran, i prepoznatljiv
-Mora biti siguran i robustan
-Mora biti arhitekturalno neutralan i prenosiv
-Mora imati dobre performanse
-Mora biti lako čitljiv

2.11. Osnovna sintaksa programskog jezika Java

Zbog sažetosti ovog rada, ključne riječi i sintaksa će se pojašnjavati u koracima, kroz edukacijske i konkretne primjere. Osnovni kod napisan u javi izgleda ovako:

```
Public Class NazivKlase
{
public static void GlavnaFunkcija()
        {
        ...
        ...
        }
}
```

Public Class NazivKlase definira klasu naziva *NazivKlase* (proizvoljan naziv) i određuje njenu dostupnost. U ovom slučaju je javno dostupna, odnosno *public*. Tijelo klase, odnosno kod, se piše unutar vitičastih zagrada. Unutar tih zagrada je u ovom slučaju definirana glavnu funkcija za ovaj primjer, koja je nazvana *GlavnaFunkcija*. Njeno tijelo se također piše unutar vitičastih zagrada. Javne klase koje ne vraćaju nikakvu vrijednost u glavnu funkciju se deklariraju kao *Public Void*.

Tipovi podataka u Java jeziku

Postoji 8 osnovnih tipova podataka u Javi koji se razlikuju po veličini i po vrsti. To su :

Byte – Veličine od 1 byte, odnosno 8 bitova, sadrži numeričku vrijednost. Najniža vrijednost je -128 a najveća 127 (255 različitih vrijednosti)
Short – Veličine od 2 byte-a, odnosno 16 bitova, najniža mu je vrijednost -32768, a najveća 32767
Int – Int je skraćen naziv za Integer. Veličine je 32 bita.
Long – 64 bitni integer
Double – 64 bitni tip podatka, koristi se za decimalne vrijednosti. Nije preporučljiv za zapis valuta ili sličnih tipova podataka.
Float – 32 bitni tip podatka, koji koristimo kada nam su 32 bita dovoljna, radi štednje prostora (ako se koristi Double a dovoljan je Float, bespotrebno se troši dvostruko više memorije)

Boolean – Boolean tip podatka može imati samo dvije vrijednosti, 1 ili 0, odnosno, točno ili netočno. Može se interpretirati kao signalna lampica koja definira je li nešto upaljeno ili ugašeno.
Char – 16 bitni tip za Unicode podatke (primjerice, slovo A ima Unicode vrijednost od U+0041).

Java oznake (Tokeni)

U Javi, token (hrv. oznaka) je najmanji element u kodu koji ima logičku vrijednost. Koristeći oznake (tokene) se strukturira program. Postoji 5 vrsta oznaka:

1. Identifikatori – Identifikatori služe za dodjeljivanje proizvoljnih naziva. Ti se nazivi dodjeljuju funkcijama, metodama, klasama itd. sa svrhom identifikacije od strane računala i razvojnog inženjera.
2. Ključne riječi – To su riječi koje se ne mogu dodijeliti. One su već „rezervirane“ za predefinirane varijable i operacije. Neke od ključnih riječi su: *case,catch,char,class,const,continue,default,final,for,goto,if...*
3. Konstante – Varijable čije se vrijednosti ne mogu mijenjati nazivamo konstante.
4. Separatori - U njih se ubrajaju simboli, poput točke, dvotočke, zagrada, i sl.
5. Operatori – Oznake varijabla koje izvršavaju operacije na operandu (ili operandima) da dobave rezultat.

Deklariranje varijable

Kod deklariranja varijable, potrebno je navesti koji tip podatka je varijabla, te njen naziv. Naziv ne smije sadržavati prazan prostor (space), i potrebno je obraćati pozornost na velika i mala slova. Primjer definiranja varijable tipa Integer, naziva AtomskaMasaPlutonija kojoj je dodijeljena vrijednost 244 je:

Int AtomskaMasaPlutonija = 244;

2.12. XML Jezik

XML (Extensible Markup Language) je standardizirani programski jezik namijenjen za označavanje, odnosno *outsource* podataka. Prva verzija je objavljena 1998. godine. Sintaksa mu je vrlo čitljiva u usporedbi sa drugim programskim jezicima, što znači da je relativno lagan za savladati. XML je vrlo sličan jeziku HTML.

Pri kreiranju XML-a World Wide Web Consortium je odredio 10 ciljeva koje je XML trebao ostvarivati:
-XML mora biti izravno primjenjiv preko interneta
-XML mora podržavati širok spektar primjena
-XML mora biti kompatibilan s SGML-om (Standard Generalized Markup Language)
-Mora biti lako pisati programe koji procesiraju XML dokumente
-Broj opcionalnih svojstava u XML-u mora biti minimalan, ili jednak nuli

-XML dokumenti moraju biti čitljivi (ljudima), i što jednostavniji
-Standard mora biti specificiran što prije
-Dizajn XML-a mora biti formalan i precizan
-Kreiranje XML dokumenata mora biti jednostavno
-Sažetost kod označavanja dokumenta XML-om je od minimalnog značenja

2.13. Osnovna sintaksa XML jezika

XML dokument mora imati jedan root element koji je parent (hrv. roditelj) svim drugim elementima. To znači da svi drugi elementi dolaze od njega kao pod-elementi (child elementi).

```
<root>
...
...
        <child>
        ...
        ...
        </child>
...
...
</root>
```

Drugo pravilo XML-a je da svi elementi moraju imati početnu oznaku i završnu oznaku

```
        <root>  *početak
        ...
        ...
        </root> *završetak
```

XML obraća pažnju na veliko i malo slovo. To znači da <button1> i <Button1> ne označavaju istu komponentu.

Vrijednosti atributa se pišu pod navodnicima.

```
        <TextView android:text=„Dobro došao korisniče!“ />
```

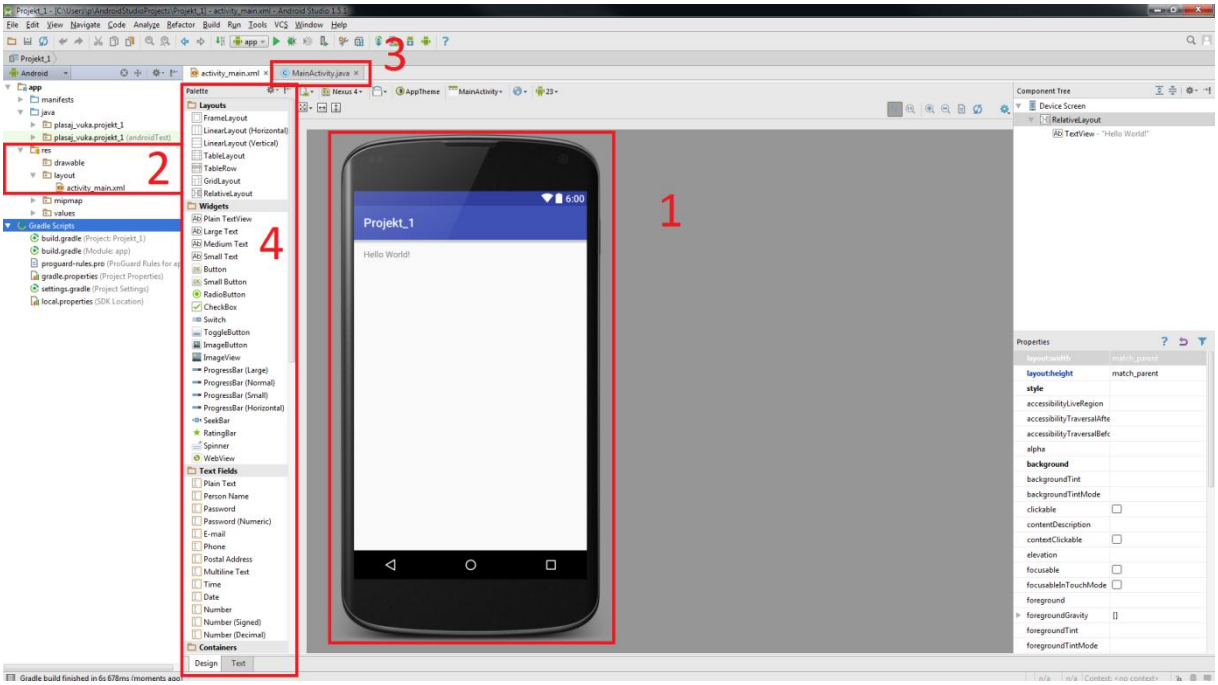
Navedena linija koda označava element textView koji je tekstualni prozor,a njegov sadržaj je naveden u navodnicima.

Nazivi elemenata mogu sadržavati slova, brojeve i sve posebne znakove,ali moraju počinjati slovom. Ne smiju počinjati tekstom xml ili XML. Ne smiju sadržavati praznine, niti sadržavati znakove povlake ni točke.

2.14. Pregled prve aplikacije (takozvana Hello World aplikacija)

Kao što je uobičajeno kod svakog programskog jezika, prva aplikacija će imati zadaću da na zaslonu uređaja prikaže poruku „Hello World!“. Ovaj primjer je namijenjen kao teoretska osnova.

Za početak, kratko upoznavanje sa sučeljem Android Studia (slika 2).



Slika 2. Predodžba sučelja softvera Android Studio

Dijelovi sučelja označeni rednim brojevima su:

1. Prikaz aktivnosti (Activity). Aktivnost je dio programa s kojim korisnik ima interakciju. Program nije ograničen sa brojem aktivnosti koje može imati.
2. Dizajn aktivnosti. Datoteka u kojoj se nalazi XML kod dizajna aktivnosti (activity_main.xml za glavnu aktivnost) nalazi se u padajućem izborniku pod *Android > app > res > layout.*
3. MainActivity.java sadrži logiku rada aplikacije. Tu se nalazi kod pisan u Javi koji određuje kako će dijelovi sučelja reagirati jedan s drugim, ili sami za sebe.
4. Paleta za dizajn. Sistemom drag-and-drop, android studio dopušta iznimno jednostavan način slaganja aplikacije sa dizajnerske strane. XML kod za dizajn se sam ažurira prilikom stavljanja, primjerice,gumba u aktivnost. Isto vrijedi iza obrnut slučaj; pri pisanju XML koda koji dodaje gumb na sučelje aktivnosti, na sučelju će se taj gumb automatski postaviti sa zadanim parametrima. Ali, taj gumb sam za sebe neće ništa raditi ako mu razvojni inženjer to ne zada u MainActivity.java, i taj kod poveže sa komponentom.

Kod koji se nalazi u activity_main.xml za ovaj slučaj je:

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="plasaj_vuka.projekt_1.MainActivity">

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello World!" />
</RelativeLayout>
```

Ovo je kod pisan u XML jeziku. To nije kod koji određuje kako će se aplikacija ponašati, već samo sažetak aplikacije sa dizajnerske strane. Moguće je i mijenjati vrijednosti parametara u ovom kodu, što će uistinu promijeniti izgled aktivnosti.

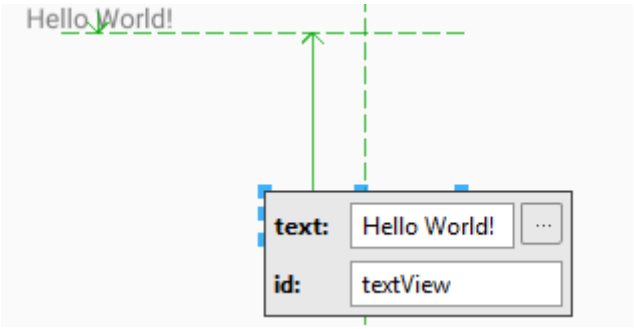
Kod koji se nalazi u MainActivity.java je

```
package plasaj_vuka.projekt_1;                    //naziv paketa

import android.support.v7.app.AppCompatActivity;  //biblioteka
import android.os.Bundlle;                        //biblioteka

public class MainActivity extends AppCompatActivity {

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

Mainactivity je definirana javna (public) klasa koja nasljeđuje karakteristike klase *AppCompatActivity* unutar koje se piše tijelo koda.

setContentView(R.layout.***activity_main***); -Ova linija koda omogućuje da korisnik vidi layout na zaslonu, odnosno prikazuje UI sučelje (user interface) aktivnosti activity_main na zaslonu simuliranog uređaja.

U ovom primjeru, java kod ne radi ništa osim toga što kreira glavnu aktivnost i dobavlja potrebne biblioteke za izvršavanje i prikaz layouta. S obzirom da je priroda ovog programa samo da ispiše tekstualnu poruku na ekranu, nije potrebna logika rada. Jednostavno se sa dizajnerske palete dovlači element „textView“ i unosi „Hello World!“ (slika 2.11).

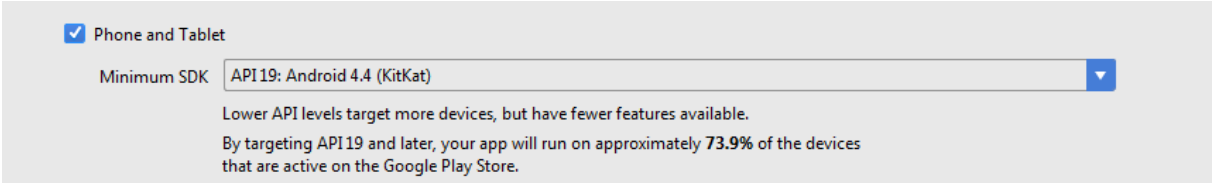


Slika 2.11. Predodžba unosa teksta u element textView

To ne znači da je ovaj kod suvišan. Ovaj kod je temelj za daljnji rad, jer bez definiranja aktivnosti i bez biblioteka, nije moguće raditi funkcije i povezivati komponente sučelja (naprimjer gumba) na logiku koja će tom gumbu dozvoliti da otvori novu aktivnost, internet preglednik, da izvrši operaciju zbrajanja,da spremi unesene parametre u bazu podataka ili nešto drugo. Kombinacija je beskonačno, zato je najbitnije shvatiti kako kod funkcionira, a sve ostalo zavisi o kreativnosti razvojnog inženjera.

2.15. API (Application programming interface)

Kod biranja verzije androida, bira se razina API-a koju će aplikacija koristiti (slika 2.12). Zastarjele verzije androida imaju manju razinu API-a, odnosno manji „API level“.



Slika 2.12. Predodžba odabiranja API razine kod kreiranja Android Studio projekta

Kod odluke koja će biti najstarija verzija androida koja će moći pokretati spomenutu aplikaciju, mora se napraviti kompromis između dostupnosti aplikacije, i modernizaciji iste. Na slici 2.12 se može vidjeti da se u ovom radu koristi API razina 19, koja je korištena na verziji androida 4.4. To je relativno moderna verzija, koja ima većinu značajki najnovije verzije, ali ima i solidnu zastupljenost (približno 73.9% korisnika Androida će moći pokrenuti tu aplikaciju).

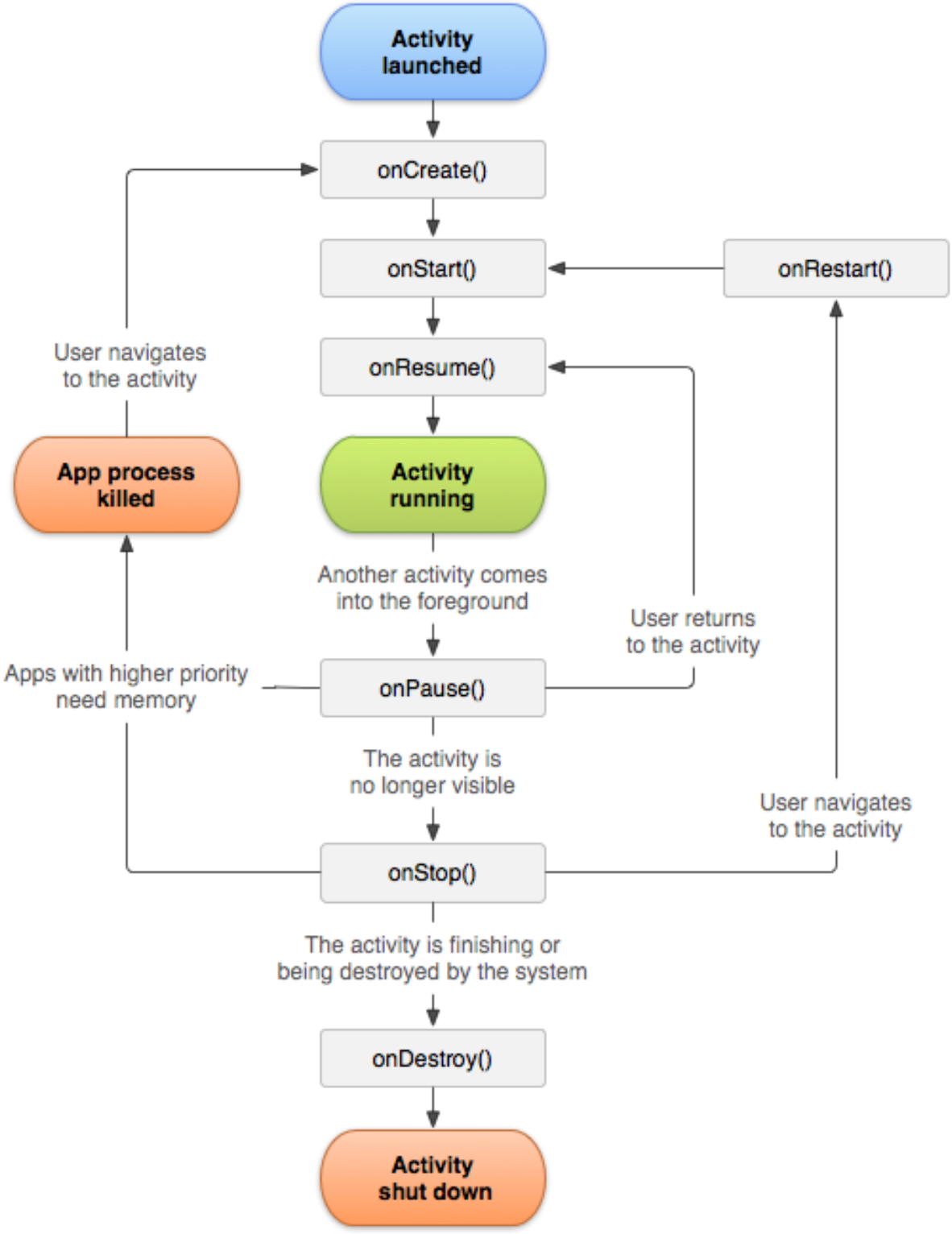
2.16. Životni ciklus aktivnosti

Iako su aktivnosti tipično prikazane preko cijelog zaslona aplikacije, mogu se koristiti i na druge načine; kao lebdeći prozori, ili implementirane unutar drugih aktivnosti koristeći ActivityGroup.

Postoje dvije metode koje gotovo sve klase i potklase aktivnosti implementiraju. To su:
onCreate(Bundle) - pokreće se u trenutku inicijalizacije aktivnosti. Ovdje se najčešće poziva *setContentView(int)* ,definira se UI (User interface), i logički se povezuju dijelovi sučelja sa kodom.
onPause() – implementira se u situacijama kada korisnik napusti aktivnost. Vrlo je bitno spremati promjene koje korisnik napravi prije implementacije ove metode u ContentProvider koji čuva podatke.

Na dijagramu (slika 2.13) je evidentan logički slijed životnog ciklusa aktivnosti. Naznačeno je da nakon završetka metode *onPause()* aktivnost se vraća u *onResume(),* a ne u *onCreate()*. Također, očito je da pauziranje (onPause) i zaustavljanje (onStop) aplikacije nije isto. Pri završetku spomenutih metoda, proces se vraća se u različita stanja aktivnosti. Pauziranje aktivnosti neće ponovno pokretati aplikaciju, nego će je samo nastaviti.

-Ako je aktivnost prednja na sučelju tada je aktivna (odnosno pokreće se).
-Ako je aktivnost izgubila svoje prednje mjesto na sučelju, ali je i dalje vidljiva, onda je pauzirana. Pauzirana aktivnost je kvazi-aktivna, što znači da njezino stanje i podatci ostaju u windows manageru.
-Ako je aktivnost potpuno zaklonjena drugom aktivnošću, ona je zaustavljena. Čim zatreba memorija koja je bila korištena za tu zaustavljenu aplikaciju, sustav će je ugasiti.

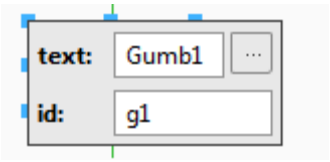


Slika 2.13. Grafička predodžba životnog ciklusa aktivnosti (izvor https://developer.android.com/reference/android/app/Activity.html)

2.17. Povezivanje predmeta sa sučelja sa kodom

Ako je cilj, kao što u je u ovom primjeru, da pritiskom na gumb aplikacija zbroji 2 broja i ispiše rješenje, nije dovoljno samo staviti gumb na sučelje. Taj gumb mora imati kod povezan na njega koji će mu reći da obavi matematičku operaciju zbrajanja i da ispiše rezultat na zaslon.

Svaki predmet na sučelju, bio to gumb,tekst,prozor,tablica,slika ili nešto drugo, mora imati svoj ID, odnosno svoje identifikacijsko ime. Gumb može imati naziv „Gumb1“ i dodjeljenjen ID „g1“ (slika 2.14).



slika 2.14. Predodžba sadržaja i ID elementa

Kada gumb ima definiran ID, može ga se referencirati u kodu. Sintaksa koda za definiranje predmeta je iduća :

```
public void onButtonClick(View v) { kod }
```

Funkcija gdje će se prilikom pritiska na gumb izvršavati kod smješten između vitičastih zagrada:

```
EditText e1= (EditText)findViewById(R.id.broj1);
```

Definirana je varijabla *e1* tako da je njezina vrijednost ono što je napisano pod id-em broj1 u formatu podatka *EditText*.
Ako je namjera manipulirati vrijednostima kako bi ih se moglo koristiti, pretvorba teksta u broj i obratno je kao sljedeće:

```
int num1 = Integer.parseInt(e1.getText().toString());
```

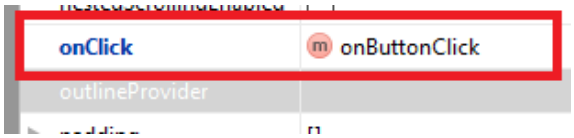
Uzima se tekst iz varijable *e1* i pretvara u string kako bi se pomoću *parseInt* mogao dobiti integer (broj). String je niz charova (charovi su prethodno objašnjeni). Time se ne mijenja vrijednost koja je unesena, već se jednostavno mijenja tip podatka da ga se može podvrgnuti matematičkim operacijama.

```
t1.setText(Integer.toString(zbroj));
```

Evidentno je da se Integer (broj) pretvara u String (string je niz znakova ograničene duljine), i zatim setText pretvara taj string u tekstualni tip podatka.

Gumb se povezuje sa funkcijom tako da se navigira u postavke gumba i pod opcijom „onClick“ odabere „onButtonClick“ (slika 2.15) što znači da će pokretati tu funkciju prilikom pritiska na gumb. Također, onButtonClick se može dodati gumbu kroz XML kod:

```
<Button
android:onClick="onButtonClick"/>
```

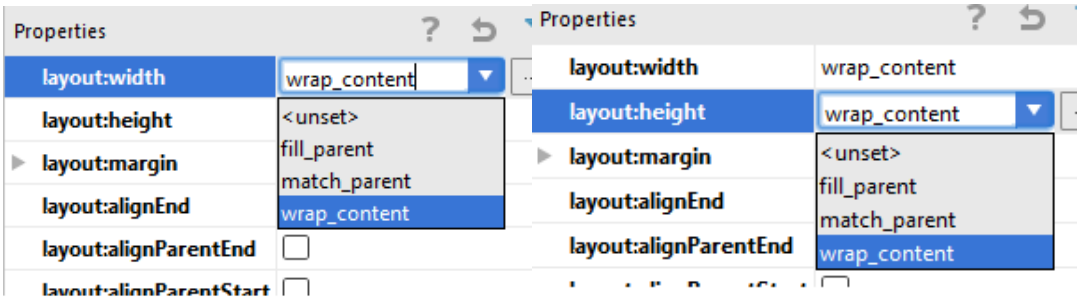


Slika 2.15. Predodžba podešavanja svojstva gumba za izvršenje funkcije prilikom pritiska

2.18. Wrap content i fill parent manipulacija komponentama korisničkog sučelja (UI-a)

Kada se aplikacija razvija, potrebno je imati na umu da različiti android uređaji imaju drugačije veličine i rezolucije zaslona. Stoga, dizajn sučelja mora biti prilagodljiv veličini zaslona. To znači da se komponente mogu razmještati po sučelju koristeći princip omjera udaljenosti. To pomaže pri ostvarivanju kontinuiteta dizajna na različitim uređajima.

Iznimka tog pravila su komponente koje su poravnate po aktivnosti u kojoj se nalaze. Za to se koristi *fill_parent*. To omogućuje da odabrana komponenta popuni aktivnosti u kojoj se nalazi po širini ili po duljini. *Wrap_content* mijenja veličinu komponente na najmanju moguću veličinu,a da i dalje sadržaj komponente bude vidljiv (slika 2.16).



Slika 2.16. Predodžba podešavanja svojstva širine i visine komponenata

Mjerna jedinica veličine komponenata je u DP (Density-independent pixels). DP mjerna jedinica je fleksibilna vrijednost koja skalira tako da se prilagodi dimenzijama zaslona.

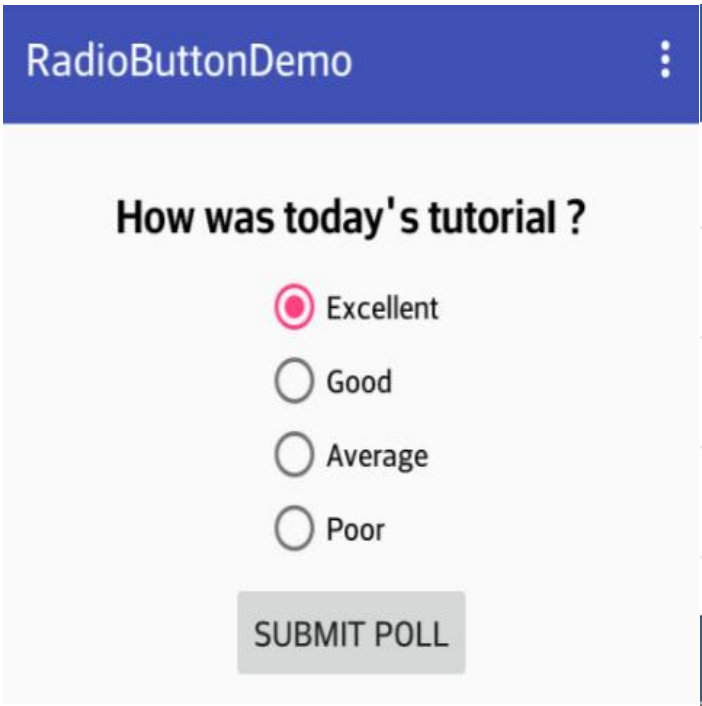
Formula za izračun DP-a je sljedeća:
dp = (širina u pikselima * 160) / gustoća zaslona
npr. 32x32 pixel ikona sa gustoćom 320 ima 16x16dp

To znači da ako postoji više ekrana iste veličine, ali različite gustoće zaslona, za sve zaslone će širina zaslona imati jednak broj dp-a.

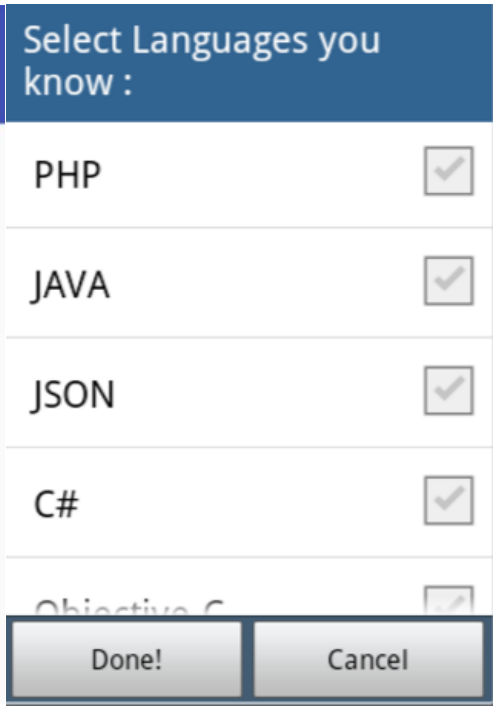
2.19. Checkbox i Radio button elementi sučelja

U Android Studiu, kod odabira između više ponuđenih opcija u sučelju aktivnosti koriste se dvije komponente koje imaju različite primjene. To su Checkbox i Radio button.
Razlika između navedena dva elementa je :

Kvadratići za odabir **i** tipa – moguće je odabrati više opcija (eng. Checkbox) (slika 2.17)
Kružići za odabir **ili** tipa – moguće je odabrati samo jednu opciju (eng. Radio button) (slika 2.18)



Slika 2.17. Predodžba Radio gumba

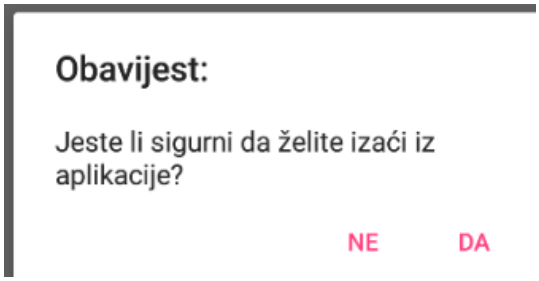


Slika 2.18. Predodžba checkboxa

Oba elementa se nalaze u dizajnerskom dijelu Android Studia. Primjeri koda za oba elementa se nalaze u eksperimentalnom dijelu ovog rada.

2.20. Alert Dialog

Alert dialog prozor se koristi da upozori ili obavijesti korisnika o nekoj aktivnosti, i ako postoji ta mogućnost, da odluči o tome kakva će se akcija izvršiti. Tipični primjer je potvrda o zatvaranju aplikacije ili želi li korisnik spremiti promjene prilikom izlaska (slika 2.19).



Slika 2.19 Predodžba alert dialog prozora

Da bi se implementirao alert dialog u aplikaciju, potrebna je biblioteka *android.app.AlertDialog*

Princip slaganja koda za funkcionalni alert dialog je sljedeći:
Potreban je listener (slušatelj) na aktivnost koja će biti uzrok stvaranju alert dialoga. Primjerice pritisak gumba (onButtonClick()).

AlertDialog.Builder abuilder = **new** AlertDialog.Builder(MainActivity.**this**);
- Označuje za koju aktivnost se gradi alert dialog.

abuilder.setMessage(**"Ovdje ide poruka alert dialoga"**)

*.setCancelable(**false ili true**)* – Bira se hoće li biti moguće ugasiti alert dialog bez odabira jedne od opcija.

*.setPositiveButton(**"Poruka za odabir opcije 1"**, **new** DialogInterface.OnClickListener()*
{ „*Ovdje se unosi kod za ono što se treba dogoditi prilikom pritiska na opciju 1*“}

*.setNegativeButton(**"Poruka za odabir opcije 2"**, **new** DialogInterface.OnClickListener()*
{„*Ovdje se unosi kod za ono što se treba dogoditi prilikom pritiska na opciju 2*“}

Sljedeći blok koda pokreće funkciju koja će izgraditi alert dialog ako se dogodi uzrok :

```
AlertDialog alert = abuilder.create();
alert.setTitle("Naslov prozora ide ovdje");
alert.show();
```

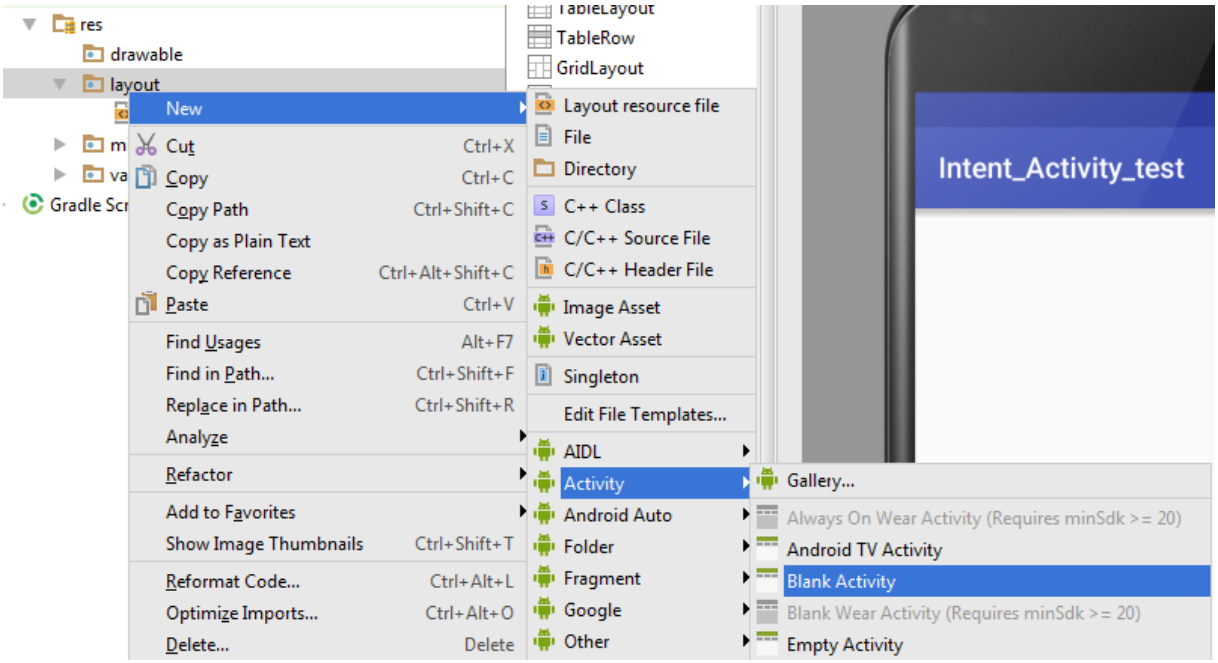
2.21. Dodatne aktivnosti i Intent

Svaka aplikacija ima barem jednu aktivnost.Tipično se naziva Main Activity, odnosno glavna aktivnost. Dok je jedna aktivnost dovoljna za jednostavnije aplikacije, često će biti potrebno imati više aktivnosti.

Da bi se dodala aktivnost u aplikaciju potreban je *intent*. Intent se može shvatiti kao most koji povezuje dvije aktivnosti. To je pasivna struktura podataka koja sadrži apstraktni opis operacije koja se mora izvršiti ili se izvršava. Postoje eksplicitne i implicitne Intent operacije, ali više o tome kasnije.

Intent se koristi sa **startActivity** da bi se pokrenula aktivnost, **broadcastIntent** da bi se proslijedila **broadcastReciever** komponentama koje ju zahtijevaju, i **startService(intent) / bindService(intent,ServiceConnection,int)** za komunikaciju sa pozadinom programa.

Proces dodavanja nove aktivnosti:
-Potrebno je dodati aktivnost u mapu sa resursima (res) pod datoteku layout (slika 2.20)



Slika 2.20. Dodavanje nove aktivnosti u layout datoteku aplikacije

Kada je druga aktivnost dodana, njen dizajn je vidljiv u mapi layout, kao i u karticama iznad sučelja, a kod u java folderu i u karticama iznad sučelja (slika 2.21)

Slika 2.21. Predodžba tabova (kartica) u sučelju Android Studia

Slika 2.22. Predodžba lokacije AndroidManifest.xml datoteke

U manifest datoteci (slika 2.22), se nalazi AndroidManifest.xml u koji je dodan dio koda za novu aktivnost:

```
<activity
   android:name="vuka.aplasaj.activity.Second"
   android:label="@string/title_activity_second"
   android:theme="@style/AppTheme.NoActionBar" />
<activity android:name=".Secondary"></activity>
```

Evidentno je da druga aktivnost nema intent filter kao što ga ima glavna aktivnost. *Intent filter* deklarira sposobnosti parent komponente, odnosno što aktivnost/usluga može napraviti i koje tipove podataka prima. Zato će se u kod za drugu aktivnost dodati sljedeće linije koda:

```
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

S obzirom da je .LAUNCHER dio glavne aktivnosti,umjesto njega se unosi .DEFAULT, a umjesto „name“ se unosi naziv druge aktivnosti. U ovom slučaju je to „**plasaj_vuka.intent_activity_test2.Secondary“.**

Druga aktivnost je spremna za korištenje. U eksperimentalnom dijelu ovog rada se nalazi primjer koji implementira više aktivnosti, kao i intent filter.

2.22. Implicitni i eksplicitni intent

-Eksplicitni intent se koristi da pokrene određenu komponentu aplikacije, poput aktivnosti.
-Implicitni intent navodi radnju koja može pozvati bilo koju aplikaciju na uređaju da je izvrši.
Drugačije rečeno, eksplicitni intent imenuje komponentu, dok implicitni intent ne imenuje komponentu, već samo operaciju.

2.23. Umetanje slika pomoću elementa ImageView

ImageView se koristi kada je potrebno unutar aplikacije (aktivnosti) umetnuti sliku. Kod umetanja slika u sučelje aktivnosti aplikacije, u obzir se uzima kvaliteta i rezolucija slike. Postoje četiri tipa veličina slika po kojima android studio svrstava slike (slika 2.23). To su:

Slika 2.23.

-hdpi (high dots per inch – odnosno visoka vrijednost točaka po jedinici incha)
-mdpi (medium dots per inch)
-xhdpi (extra high dots per inch)
-xxhdpi (extra extra high dots per inch)

Podjelu kvalitete slika se može vidjeti unutar android datoteke „drawable“

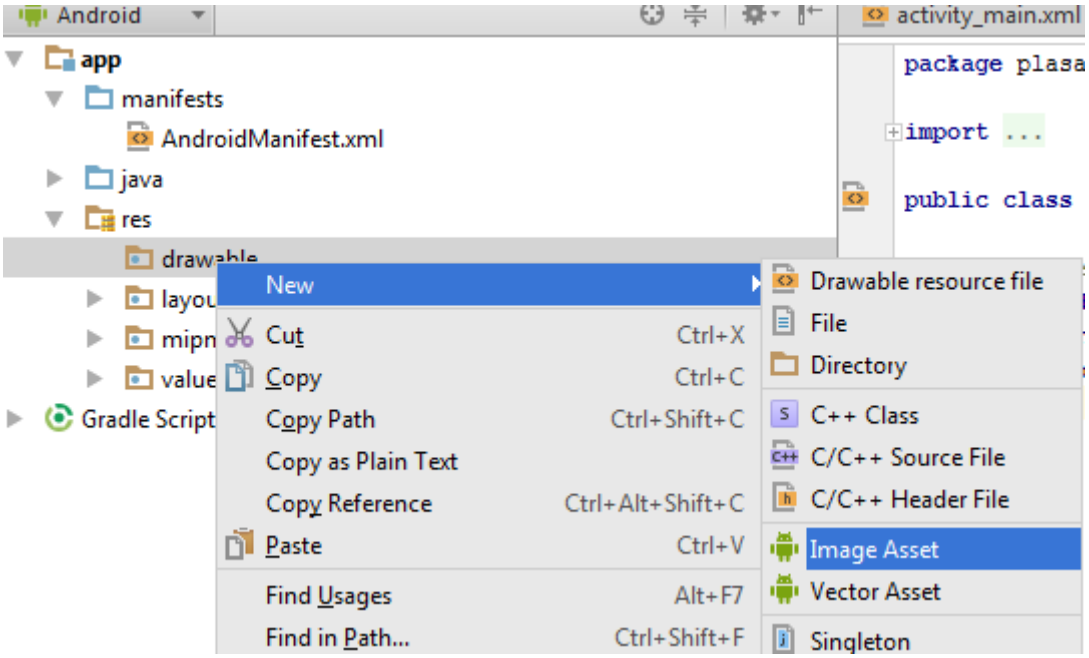
Često se događa da se poistovjećuju DPI i PPI. PPI predstavlja pixele po inchu, a to naravno, nije isto.

Softver (Android studio) automatski odabire u koju grupu će svrstati sliku s obzirom na njenu kvalitetu i veličinu, ili može konvertirati sliku u svaku od navedenih tipova. Ako je potrebno dodati novu sliku, prvo se dodaje u drawable datoteku. To se čini pomoću Asset studia (desni klik na drawable, add new image asset). U asset studio izborniku, bira se tip slike (launcher ikona, action bar ikona, ili ikona za obavijesti).

Te slike se ne mogu jednostavno „dovući“ na dizajnersko sučelje, već je potrebno odrediti prostor za umetanje slike. Alat za to se naziva *ImageView*.

ImageView funkcionira kao okvir za slike unutar aktivnosti aplikacije. Potrebno je odrediti njegovo ponašanje, kao i ponašanje slika unutar njega. Implementacija je prikazana u primjeru u eksperimentalnom dijelu rada.

Kod dodavanja slika u datoteku drawable, prvo slovo naziva slike mora biti ili malo slovo,ili broj. Isto vrijedi za bilo koje resurse koji se dodaju u poddatoteke glavne datoteke res.

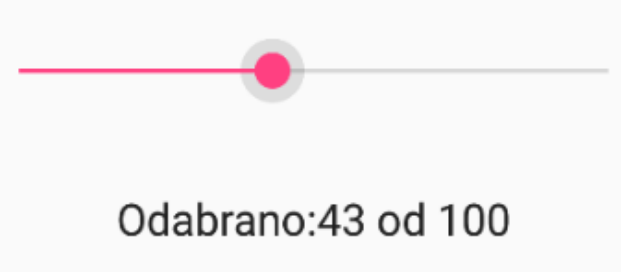


Slika 2.24. Predodžba dodavanja slika u „drawable“ datoteku aplikacije

ListView se koristi kada je na zaslonu (unutar aktivnosti) potrebno prikazati neku listu ili popis. Slično kao i kod ImageView-a, potrebni su resursi iz kojih će se dobavljati podatci.

2.24. Seek bar

Element za biranje vrijednosti između dvije granične vrijednosti u stilu klizećeg gumba se naziva Seek bar. (slika)



Slika 2.24. Predodžba SeekBar komponente

Kod povezivanja elementa sa kodom u slučaju seekbara postoji nekoliko specifičnih dijelova sintakse. To su:

Import android.widget.Seekbar – dobavljanje biblioteke za rad seekbara

Sbar.getmax() – dobavlja najveću vrijednost seek bara
Sbar.getmin() – dobavlja najmanju vrijednost seek bara
OnSeekBarChangeListener() – prilikom promjene vrijednosti klizećeg gumba pokreće iduće tri funkcije:

onProgressChanged- obavlja se nakon što je gumb namješten na novu vrijednost
onStartTrackingTouch- obavlja se tijekom namještanja gumba
onStopTrackingTouch-obavlja se nakon prestanka namještanja

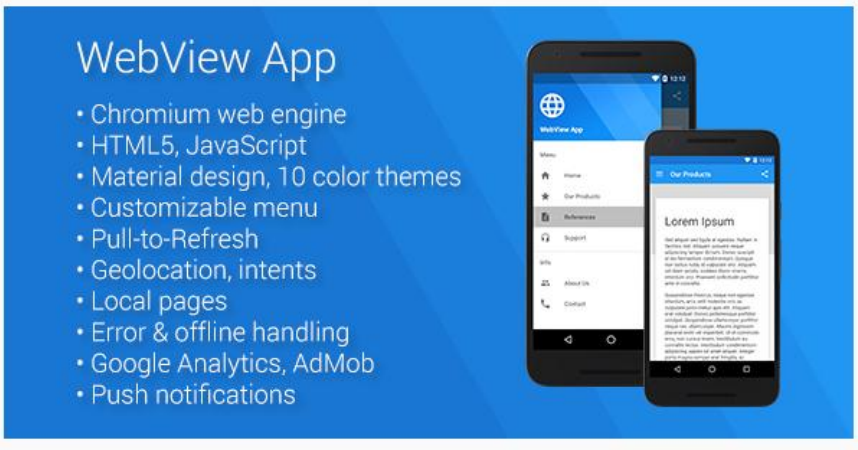
2.25. WebView

Android *WebView* komponenta je sistemska komponenta koja dozvoljava android aplikacijama da prikazuju internet sadržaj. Drugim riječima, *WebView* dopušta neautoriziranim aplikacijama da prikazuju sadržaj internet preglednika unutar aplikacije, koristeći podatkovnu mrežu.

Koristeći WebView, android razvojni programeri imaju mogućnost da izrađuju aplikacije koristeći tehnologije za web dizajn, primjerice HTML, javascript, CSS i slično, zadržavajući strukturu klasične aplikacije.

Javascript je po početnim postavkama ugašen. Moguće ga je omogućiti koristeći naredbu *setJavaScriptEnabled(true)*. Također, dodavanje Java objekata u *WebView* je moguće naredbom *addJavascriptInterface(Object, String)*, gdje su Object i String parametri koje naredba uzima. Da bi se pristupilo java objektima u WebView prikazu, javascript mora biti omogućen.

Primjer aplikacije koja implementira WebView je univerzalna android webview aplikacija tvrtke Codecanyon (slika 2.25). Jedna je od najprodavanijih aplikacija koja implementira WebView, i također je razvijena u android studiu, za verziju androida 4.0.3 (Ice Cream Sandwich). Njena glavna namjena je da konvertira web stranice u aplikacije.



Slika 2.25. Predodžba reklamnog oglasa Codecanyon WebView aplikacije (izvor: https://codecanyon.net/item/universal-android-webview-app/8431507)

2.26. Geste

Kod androida, pod geste se ubrajaju svi različiti pokreti koje korisnik može napraviti pokazivačem. U to se ubrajaju jednostruki, dvostruki, dugi , brzi klikovi (pritisci), listanje,

potezi u određenim smijerovima itd. Aplikacije dobivaju na funkcionalnosti ako je moguće implementirati više gesta da obavljaju različite funkcije. Za to je potrebno koristiti *GestureDetector*. On služi za prepoznavanje gesta, koje se nazivaju *MotionEvent*. *GestureDetector*. Vezano za notifikacije, *OnGestureListener* će obavijestiti korisnika kada se izvede određena gesta.

Ako je za određenu aktivnost željeno da implementira prepoznavanje gesta, poslije definiranja aktivnosti je potrebno napisati:
Implements GestureDetector.OnGestureListener

Biblioteke bitne za rad s gestama su:
Android.view.GestureDetector
Android.view.MotionEvent
Android.gesture.Gesture
*Android.view.GestureDetector.**

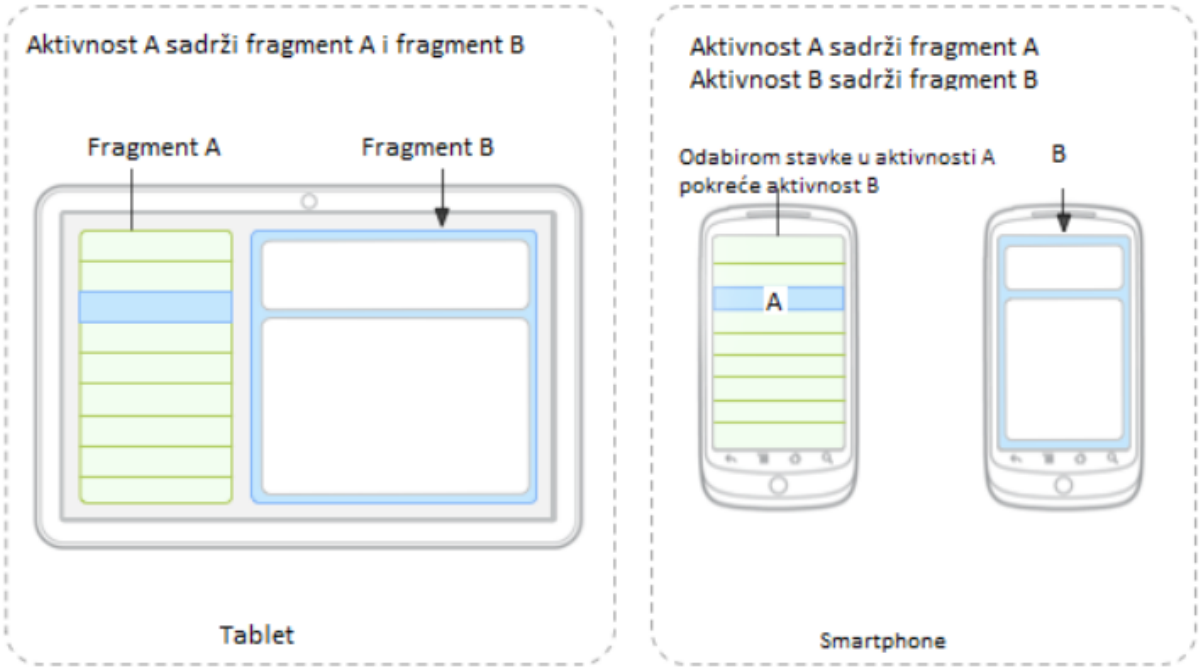
Za kreiranje prepoznavatelja gesti sa svojim osobnim listenerom se koristi sljedeća linija koda:
GestureDetector (GestureDetector.OnGestureListener listener, Handler handler)

Dva su parametra; listener (*GestureDetector.OnGestureListener*) i handler (*Handler*). Listener je onaj koji reagira na akciju korisnika, a handler je zadužen za dostavljanje sistemskih poruka za tu funkciju. Poruke koje handler dostavlja imaju svoj red i obavljaju se jedna po jedna.

2.27. Android fragmenti

Fragment predstavlja način ponašanja dijela sučelja (UI-a) u aktivnosti. Fragmenti se mogu kombinirati, tako da u jednoj aktivnosti može biti veći broj fragmenata. Fragmenti su građevni elementi od kojih se sastavlja aktivnost. Svaki od fragmenata ima svoj životni ciklus, svoje parametre, prima svoje inpute i ostalo. Jedan fragment se može koristiti u više aktivnosti. Fragmenti se mogu definirati kao modularni dijelovi aktivnosti.

Fragment ne može opstati bez aktivnosti. Mora biti utkan u aktivnost. Također, životni ciklus aktivnosti ima utjecaj na ciklus fragmenta. Ako bi se aktivnost pauzirala, svi njeni fragmenti su također pauzirani. Ali, ako pauziramo fragment, aktivnost nije pauzirana. Najveća prednost fragmenata je ta što pojednostavljuje proces izrade sučelja za zaslone različitih veličina, jer se svaki fragment prilagođava prostoru na zaslonu zasebno. Za korištenje fragmenata je potrebno koristiti verziju androida 3.0 (Honeycomb) ili veću. To je ekvivalentno API razini od 11 ili više.



Slika 2.26. Predodžba prikaza fragmenata kod različitih veličina zaslona (tablet/smartphone).

Na slici 2.26 je prikazano kako fragmenti u aktivnosti mogu izgledati. Na primjeru tableta su unutar jedne aktivnosti smještena dva fragmenta. Odabirom stavke u fragmentu A, mijenja se sadržaj fragmenta B. Na primjeru smartphonea su prikazane dvije aktivnosti i dva fragmenta. Kada se odabere stavka u aktivnosti A, pokreće se aktivnost B sa fragmentom B.

Oba primjera ostvaruju istu funkciju sa istim fragmentima, samo kod tableta, zbog većeg zaslona, je drugačiji dizajn sučelja.

Za transakcije između dva ili više fragmenata se koristi *FragmentManager*. *FragmentManager* se poziva naredbom *getFragmentManager()* unutar aktivnosti. Neke od operacija koje FragmentManager može odraditi su:
FindFragmentById() – nalazi fragmente unutar aktivnosti koji imaju sučelje
FindFragmentByTag() –nalazi fragmente unutar aktivnosti koji nemaju sučelje
popBackStack() – obavlja funkciju gumba „natrag“ koju aktivira korisnik
addOnBackStackChangedListener() – angažira listenera koju primjećuje promjene u pauziranim fragmentima (onima koji nisu aktivni)
FragmentTransaction – dozvoljava micanje i dodavanje fragmenata

2.28. Stvaranje fragmenata

Kod kreiranja fragmenata, koristi se potklasa Fragment. Njena sintaksa je slična sintaksi aktivnosti. Ima iste pozivne metode poput *onCreate()*, *onPause()*, *onStop()* itd.

Preporučeno je koristiti barem sljedeće tri ciklusne metode:

onCreate() – poziva se prilikom kreiranja fragmenta. Poželjno je odrediti osnovne komponente fragmenta koje će ostati nakon pauziranja/zaustavljanja, te ponovnog paljenja.

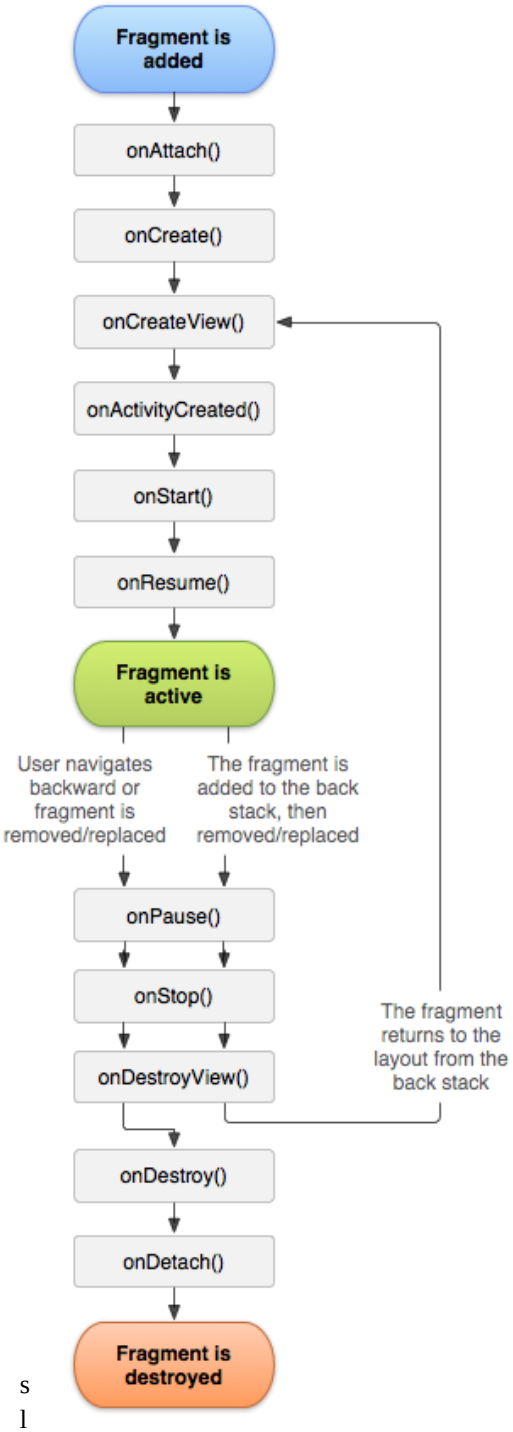
onCreateView() – poziva se kada je vrijeme da se fragment prvi puta „podigne“, to jest, inicijalizira svoje sučelje.

onPause() – poziva promjene koje će ostati i nakon što korisnik izađe.

Za stvaranje sučelja fragmenta, mora se implementirati *onCreateView()* metoda, i njen povrat mora biti *View*. Fragmenti ne moraju imati sučelje. Oni koji su bez sučelja se koriste za pozadinske procese. Sintaksa za kreiranje fragmenta bez sučelja je *Add(Fragment, String)*

2.29. Koordinacija sa životnim ciklusom aktivnosti

Fragmenti imaju nekoliko dodatnih faza u životnom ciklusu, koji rukuju specifičnim interakcijama sa aktivnošću poput građenja i rušenja sučelja fragmenta. Te dodatne faze su:
onAttach() – poziva se kada se fragment poveže sa aktivnošću, aktivnost se poziva u zagradama
onCreateView() – poziva se da kreira view povezan sa fragmentom
onActivityCreated() – poziva se kada onCreate() metoda vrati podatke
onDestroyView() – poziva se kada se view povezan sa fragmentom uništi
onDetach() – poziva se kada se fragment odspoji od aktivnosti



s
l
Slika 2.27. Grafička predodžba dijagrama životnog ciklusa fragmenta. (izvor: https://developer.android.com/guide/components/fragments.html)

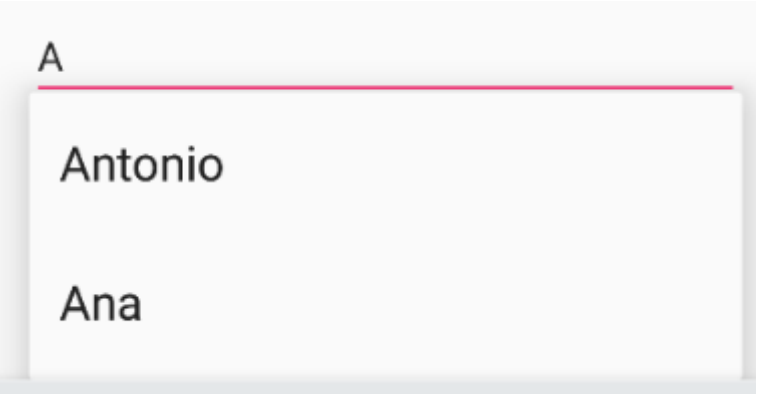
2.30. AutoCompleteTextView

Komponenta AutoCompleteTextView je tekstualni prozor koji omogućuje automatsku dopunu teksta koji korisnik unosi. Prilikom upisivanja, pojavljuje se padajući izbornik sa prijedlozima nadopune i korisnik ima mogućnost odabrati jednu od opcija (slika 2.28).

.*setThreshold(n)* dozvoljava da se odredi broj znakova (n-broj) nakon kojega će se pojavljivati prijedlozi. Ako je postavljeno *AutoCompleteTextView.setThreshold(3)*, tek nakon što je unesen treći znak će početi prijedlozi sa dopunama. Baza podataka se sprema u obliku polja stringova (interno), ili u obliku vanjskih baza podataka u obliku biblioteka (eksterno).

AutocompleteTextView (Context context AttributeSet attrs) – izrađuje novi auto complete text view. Prima 2 argumenta, context govori gdje se pokreće text view, kroz kojega dobiva pristup resursima, i drugi argument je attribute set, a to su atributi .xml tipa koji izgrađuju text view.

Da bi se podatci iz polja stringova mogli koristiti kao opcije za dopunu, potreban je adapter za podatke. Njega se angažira operacijom getAdapter().



Slika 2.28 Predodžba AutoComplete komponente

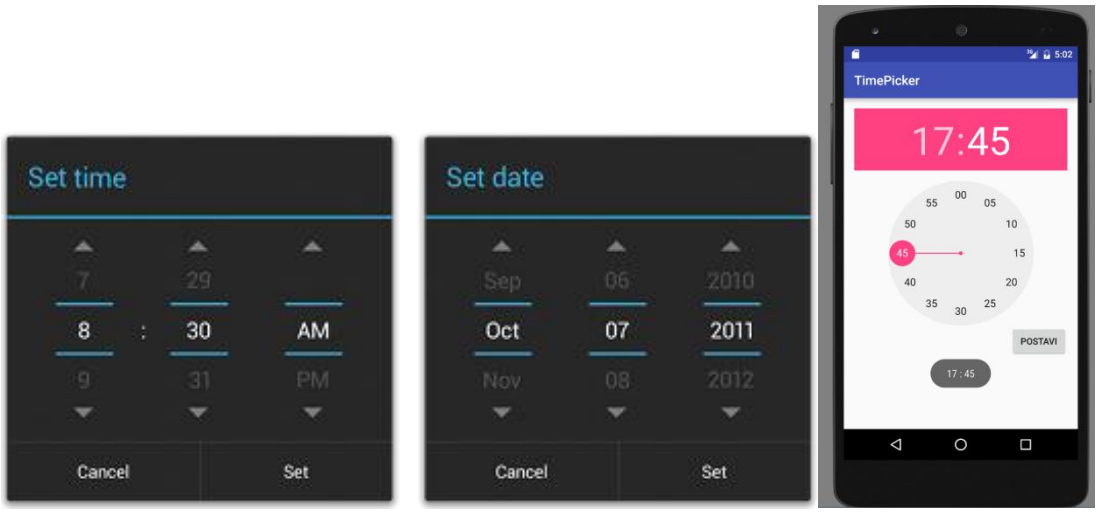
2.31. TimePicker i DatePicker

TimePicker je komponenta koja dopušta korisniku da namješta vrijeme. Moguć je prikaz u 24 satnom prikazu ili u AM/PM načinu. To se podešava naredbom .*setIs24hourView(True/false)*. Preporučljivo je koristiti TimePickerDialog za podešavanje vremena, zbog njegove prilagodljivosti.TimePicker uzima previše prostora za funkcionalno korištenje (slika 2.29).

Da bi se TimePickerDialog implementirao koristeći DialogFragment, mora se definirati fragment koji vraća vrijednost TimePickerDialog iz funkcije za kreiranje TimePickerDialoga.

Za definiranje DialogFragmenta, definirana je onCreateDialog() metoda koja ima povratnu varijablu iz TimePickerDialoga. Ako su potrebne povratne informacije, treba implementirati .OnTimeSetListener sučelje.

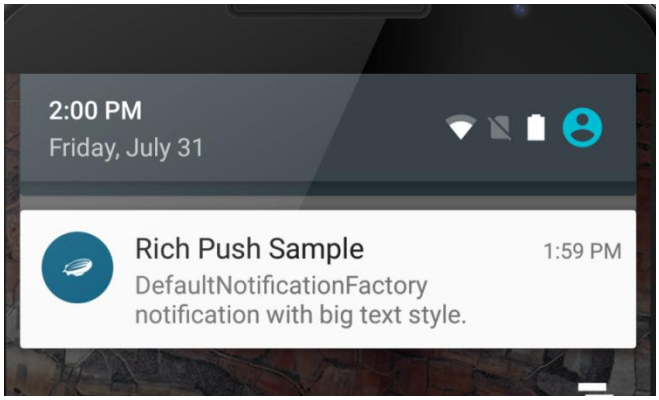
Između TimePickera(Dialog) i DatePickera(Dialog) nema razlike. Logika je ista, samo se upotrebljava druga komponenta. Cijela priča sa fragmentima i dalje vrijedi.



Slika 2.29. Predodžba TimePickerDialog, DatePickerDialog i TimePicker komponenata

2.32. Notifikacije, NotificationManager, PendingIntent i flagovi

Notifikacije su obavijesti koje se prikazuju korisniku izvan sučelja aplikacije. Notifikacija se pojavljuje u traci za obavijesti (gornji dio ekrana) koji se može povući prema dolje i time otvara „ladicu“ notifikacija. Traka i ladica za notifikacije je dostupna i izvan i unutar aplikacija (slika 2.30) .



Slika 2.30. Predodžba notifikacija na android mobilnom uređaju

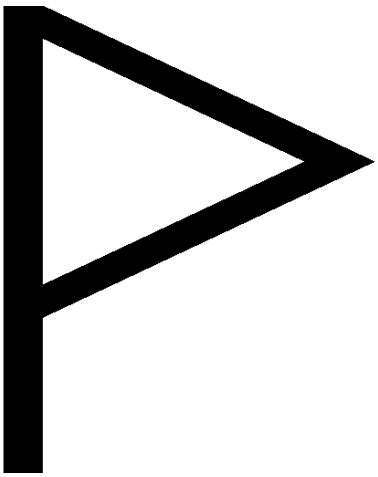
Da bi se kreirala notifikacija, koristi se *NotificationCompat.Builder.build()*. Povratna informacija te funkcije je definirana notifikacija. Tu definiranu notifikaciju je nužno lansirati koristeći *NotificationManager* (ili *NotificationManager.notify()* za direktno pokretanje notifikacije). Da bi notifikacija bila definirana, potrebno je definirati minimalno tri stavke: ikonu *(.setSmallIcon())* , naslov (*.setContentTitle())* i tekst obavijesti (*.setContentText()*).

Pokretanje notifikacije i Intent (PendingIntent) – Čin lansiranja notifikacije se definira koristeći *PendingIntent* koji sadrži *Intent* (Što je intent je objašnjeno u prethodnom poglavlju). PendingIntent je oznaka koju razvojni inženjer može dati drugoj aplikaciji, naprimjer notification manageru, alarm manageru, ili nekoj drugoj aplikaciji koja ne mora biti službena android aplikacija (može biti 3rd party aplikacija). To dopušta drugim aplikacijama da koriste dozvole aplikacije koja se razvija u ovom slučaju i izvrši određeni kod. Ako se želi pokrenuti neka aktivnost pri kliku na notifikaciju, dodaje se PendingIntent pozivajući funkciju *setContentIntent()*. Notifikaciju je moguće maknuti koristeći *cancel()*, gdje je argument

notifikacija koja se želi deaktivirati, ili *cancelAll()*, koja deaktivira sve notifikacije koje su pozvane.

2.33. Flag

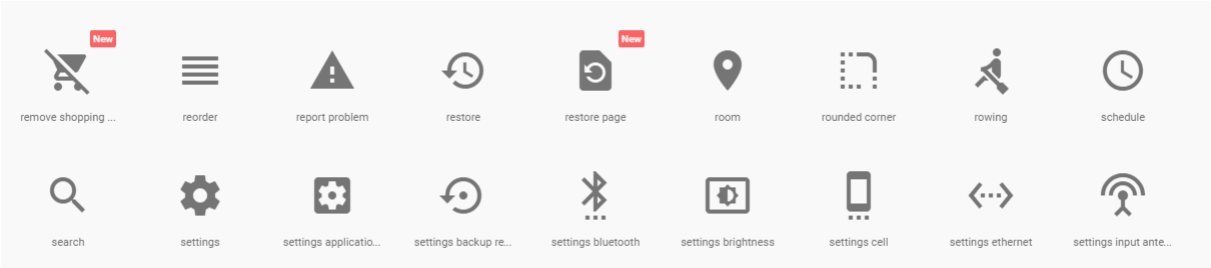
Flagovi (ili zastavice) su predefinirane sekvence koji sadrže binarne vrijednosti. Program koristi flagove da ostavi znak na nečemu za drugi program, ili sam za sebe. Flagovi se koriste od strane emitera digitalne televizija za zaštitu autorskih prava. Flag se enkodira u signal te se može prepoznati ako se sadržaj piratizira. Flagovi u programerskom smislu imaju široke primjene, te ih je najbolje shvaćati kao signalizatore (je li nešto točno ili netočno, jedinica ili nula, gore ili dolje, i slično).



Slika 2.31. Predodžba simbola flaga (zastavica)

2.34. Korištenje prilagođenih ikona za dizajn

Nakon što su ikone dizajnirane, ili skinute iz biblioteke (postoje besplatne biblioteke ikona, naprimjer *https://design.google.com/icons/index.html* (slika 2.32)) i stavljene u mapu drawable od android studia, na raspolaganju su za korištenje prilikom dizajniranja aplikacije.



Slika 2.32. Predodžba nekih od ikona u besplatnom paketu sa google-ove stranice za android developere.

U XML kodu ikone se implementiraju sljedećom linijom koda:

```
android:icon="@drawable/ic_naziv_XYdp"
```

A u java kodu na sljedeći način:

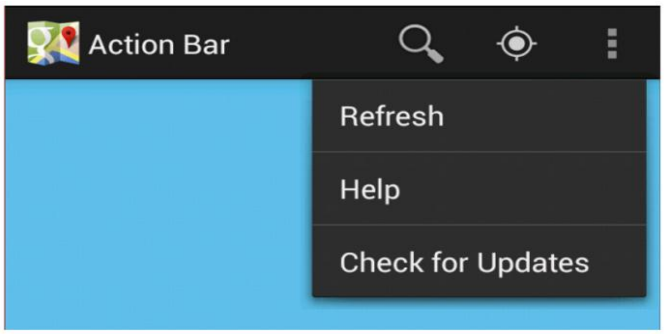
```
abar.setLogo(R.drawable.ic_label_black_24dp);
```

2.35. Action bar i overflow menu

Action bar je dio sučelja aplikacije. Nalazi se na vrhu sučelja, ispod trake za notifikacije (slika). Action bar ne nestaje prilikom prijelaza između aktivnosti. Na action bar se tipično stavljaju najbitnije funkcije, primjerice, home, natrag, tražilica i slično. Time su najbitnije funkcije uvijek na sučelju i pristupačne u svakom trenutku. Naročito je bitna za navigaciju

kroz aktivnosti aplikacije. Također, ako postoji naslov za aktivnost, action bar je idealno mjesto za njega.

Klasična tema (odnosno, izgled) action bara je AppCompat tema. Moguće je koristiti druge teme koristeći naredbu *requestFeature* (FEATURE_SUPPORT_ACTION_BAR) ili prilagođenu temu sa svojstvom windowActionBar. Ukoliko se unutar određene aktivnosti poziva action bar, poziva se koristeći *getSupportActionBar()*. Razlog zašto se koristi *getSupportActionBar*, a ne *getActionBar* je taj što se time dobavljaju odgovarajuće biblioteke. Za definiranje action bara se mogu koristiti flagovi. Primjerice, ako se koristi logo, flag koji to daje na znanje je DISPLAY_USE_LOGO.



Slika 2.33. Predodžba action bara sa padajućim overflow izbornikom

2.36. Service

Service je komponenta aplikacije koja se odvija bez obzira na sučelje. Odvija se u pozadini, i predviđena je za dugotrajne periode trajanja operacija. Service može pokrenuti neka komponenta sa sučelja, naprimjer gumb.

Service će se nastaviti odvijati i nakon što je aplikacija koja ga je pokrenula zaustavljena. Primjerice, service se može koristiti da reproducira glazbu iz pozadine sučelja.

Oblici service-a

Service može biti:

1) Pokrenut (started) – Service je pokrenut nakon što se izvrši naredba *startService()*. Nakon što je pokrenut, može se odvijati beskonačno, dok se ne ugasi (termin je „destroy“). Može imati određeno trajanje, primjerice dok izvrši svoju funkciju, a zatim se sam ugasi, najčešće bez obavijesti korisniku.

2) Vezan (bound) - Service je vezan kada se komponenta aplikacije poveže s njime koristeći naredbu *bindService()*. Vezani service dozvoljava komponenti da komunicira sa service-om, ili da koristi service da komunicira interprocesno (IPC- interprocess communication). Takav service traje dok je komponenta vezana na njega. Moguće je imati više komponenti vezanih na jedan service.

Kod dizajniranja aplikacije, bitno je znati da service ne stvara nove veze sa procesorom, već koristi nit (thread) koju je kreirala njena matična aplikacija. To znači da ako service odvija kompliciranе operacije, optimizacija aplikacije će biti ugrožena i može doći do pogoršanja fluidnosti i nedostatka memorije za aplikaciju, a naposljetku i rušenje (crashanje) aplikacije. Moguće je service-u dodijeliti zasebni thread (nit) i time će se znatno poboljšati stabilnost aplikacije.

Service je potrebno deklarirati u AndroidManifest.xml datoteci sljedećom linijom koda:
<service android:name)“.nazivServicea“ />

Najbitnije funkcije vezane za service su:

onStartCommand() – Koristi se prilikom zahtjeva za pokretanje servisa, pozivajući startService() naredbu.

onBind() – Koristi se kada sustav želi vezati komponentu sa servisom, pozivajući bindService() naredbu.

onCreate() – Koristi se prvi puta kada je servis kreiran. Izvodi inicijalizaciju. Ako je service već pokrenut, ova funkcija se ne poziva.

onDestroy() – Koristi se kada se service više ne koristi. Zadaća ove funkcije je da ugasi service i oslobodi radnu memoriju i CPU resurse koje je service koristio.

2.37. Pohrana podataka

Postoji više načina pohrane podataka preko aplikacije. Svaka od opcija ima svoje prednosti i nedostatke. Primjerice, dostupnost podataka i količina prostora koju podatak zauzima.

Opcije pohrane su:

1) Preference – pohranjuju iznimno malu količinu podataka; sadrže osnovne, primitivne vrste podataka (boolean, integer, string,float i long). Da bi se koristio Shared Preference način pohrane, poziva se metoda *getSharedPreferences()* za više preference datoteka, gdje je argument naziv datoteke. Za samo jedan preference podatak koristimo *getPreferences()*, za kojega nije potrebno navesti naziv, jer je jedini preference podatak u aktivnosti.

   Vrijednosti se unose tako da se pozove *edit()* i time dobije SharedPreferences.Editor, a zatim se dodaje vrijednost sa *putString/putInt* ili koji je već tip varijable, te ih se primjenjuje sa *commit()*. Vrijednosti se čitaju sa *getString/getInt*.

2) Interna pohrana – pohranjuje privatne podatke na memoriji uređaja. Privatni podatci se mogu dijeliti sa drugim aplikacijama koristeći content provider.

   Da bi unijeli vrijednost poziva se openFileOutput() sa nazivom datoteke (vraća *FileOutputStream* iz funkcije koristeći *return*). Podatak se upisuje sa *write()*, a unos podataka (stream) se završava sa *close()*. Analogno tome, podatci se učitavaju sa pohrane koristeći *openFileInput()*, koji vraća *FileInputStream*. Čitanje podataka se izvodi sa *read()* a zatvaranje sa *close()*.

3) Eksterna pohrana – pohranjuje javne podatke.

4) SQLite baze podataka – pohranjuje podatke na web-u u privatnim bazama podataka.

5) Mrežna veza – pohranjuje podatke na web-u u mrežnim serverima.

2.38. Baze podataka, SQL jezik, mySQL sustav, i SQLite

S obzirom da se ovaj rad fokusira na programske jezike Java i XML, koristiti će se SQLite baze podataka koje ne zahtijevaju PHP skripte poput MySQL mrežnih servera.

SQL je programski jezik za rukovanje bazama podataka. Pod rukovanje tu u obzir dolazi traženje, brisanje, ažuriranje i dodavanje podataka.

Ukoliko će aplikacija rukovati bazama podataka, Android dolazi sa ugrađenim implementacijama SQLite bazama podataka. SQLite je relacijska, open source (besplatna) baza podataka. Prednosti SQLite baza podataka su :

-Zahtijeva vrlo malo administracije, što znači da ima poprilično visok stupanj automatizacije
-Podatci ostaju nepokvareni (uncorrupted) ukoliko dođe do pada sustava.
-Baza podataka je locirana u jednoj datoteci
-Jednostavno sučelje, brza izvedba

2.39. Priprema i instalacija biblioteka i softvera za rad

Prije nego je razvojni inženjer spreman za rad, potrebno je instalirati JDK (Java development kit). JDK je softversko okruženje potrebno za razvijanje i pokretanje JAVA aplikacija. Uključuje JRE (Java Runtime Environment), javac compiler, jar arhiver, javadoc generator dokumentacije i druge alate potrebne za JAVA development.

Java JDK se može skinuti sa službene Oracle stranice (slika 2.34).
(http://www.oracle.com/technetwork/java/javase/downloads/index.html)

Java SE Development Kit 8u101

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.77 MB	jdk-8u101-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.72 MB	jdk-8u101-linux-arm64-vfp-hflt.tar.gz
Linux x86	160.28 MB	jdk-8u101-linux-i586.rpm
Linux x86	174.96 MB	jdk-8u101-linux-i586.tar.gz
Linux x64	158.27 MB	jdk-8u101-linux-x64.rpm
Linux x64	172.95 MB	jdk-8u101-linux-x64.tar.gz
Mac OS X	227.36 MB	jdk-8u101-macosx-x64.dmg
Solaris SPARC 64-bit	139.66 MB	jdk-8u101-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.96 MB	jdk-8u101-solaris-sparcv9.tar.gz
Solaris x64	140.33 MB	jdk-8u101-solaris-x64.tar.Z
Solaris x64	96.78 MB	jdk-8u101-solaris-x64.tar.gz
Windows x86	188.32 MB	jdk-8u101-windows-i586.exe
Windows x64	193.68 MB	jdk-8u101-windows-x64.exe

Java SE Development Kit 8u102

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

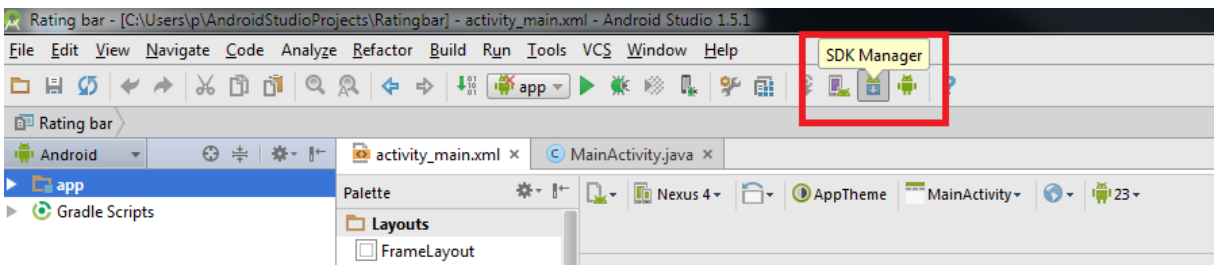
○ Accept License Agreement ◉ Decline License Agreement

Product / File Description	File Size	Download
Linux x86	160.35 MB	jdk-8u102-linux-i586.rpm
Linux x86	175.03 MB	jdk-8u102-linux-i586.tar.gz
Linux x64	158.35 MB	jdk-8u102-linux-x64.rpm
Linux x64	173.03 MB	jdk-8u102-linux-x64.tar.gz
Mac OS X	227.35 MB	jdk-8u102-macosx-x64.dmg
Solaris SPARC 64-bit	139.59 MB	jdk-8u102-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	98.98 MB	jdk-8u102-solaris-sparcv9.tar.gz
Solaris x64	140.02 MB	jdk-8u102-solaris-x64.tar.Z
Solaris x64	96.24 MB	jdk-8u102-solaris-x64.tar.gz
Windows x86	189.2 MB	jdk-8u102-windows-i586.exe
Windows x64	194.68 MB	jdk-8u102-windows-x64.exe

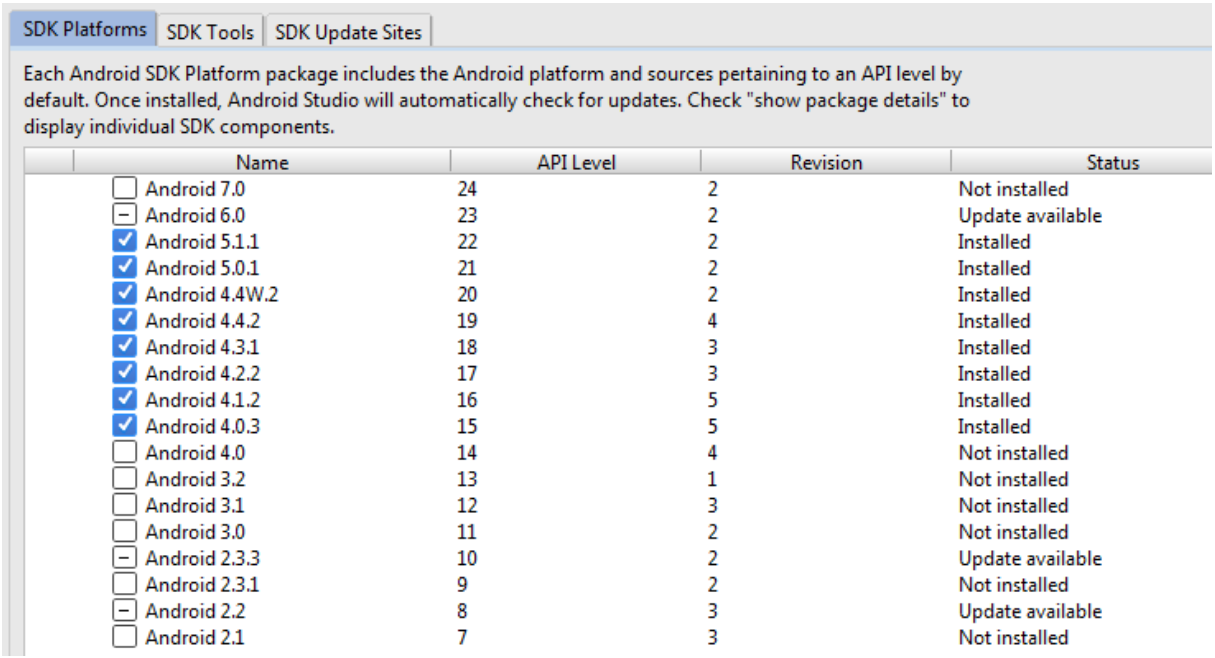
Slika 2.34. Predodžba odabir verzije i platforme JDK-a na službenoj Oracle stranici (izvor: https://www.oracle.com/downloads/index.html)

Nakon što je Java JDK uspješno instaliran na računalu, potrebno je instalirati Android studio. *https://developer.android.com/studio/index.html*.

Nakon instalacije, još je nužno konfigurirati SDK (software development kit), to jest skup alata koji omogućuju stvaranje aplikacija za odabranu platformu. U postavkama se nalazi SDK manager, unutar kojega je moguć odabir paketa (na slici su paketi koji su korišteni u ovom radu).

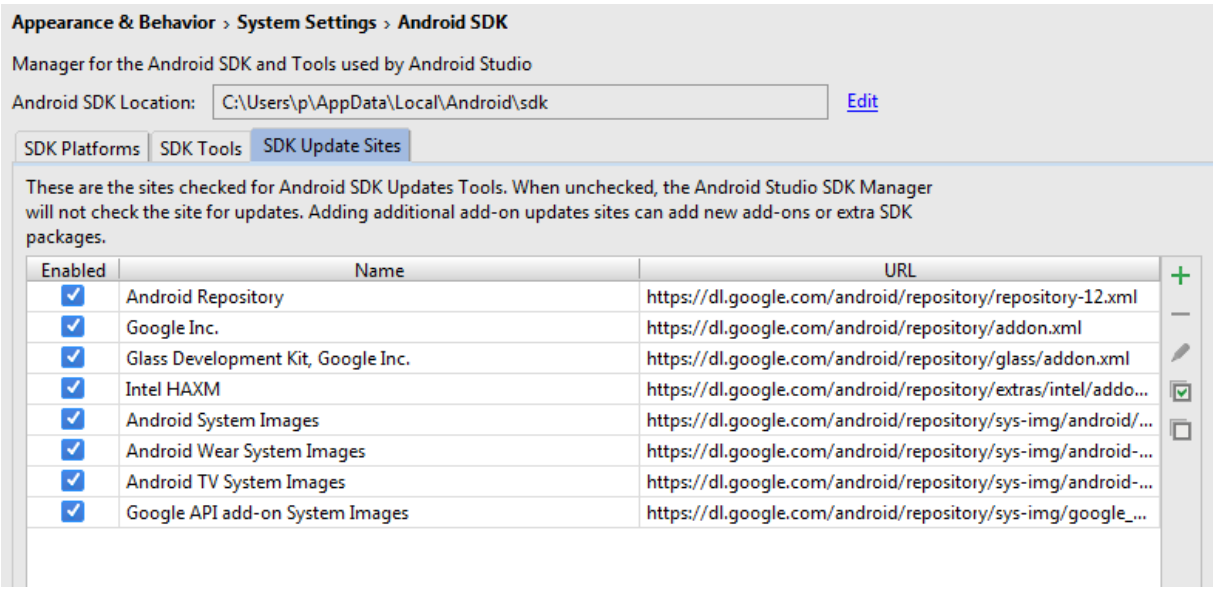


Slika 2.35. Predodžba SDK Managera u Android Studiu



Slika 2.36. Predodžba Odabira API razine Android sustava za koji se razvija aplikacija

Na slici 2.37 se mogu vidjeti biblioteke koje su ažurirane u Android Studiu za ovaj slučaj. Napomena za Intel HAXM (Intel Hardware Accelerated Execution Manager) ; to je emulator na kojem se simulira android uređaj koji pokreće aplikacije, podešavanje emulatora ovisi o verziji androida na kojoj se želi testirati aplikacija. Nalazi se u kategoriji „extras“.



Slika 2.37 Predodžba biblioteka u SDK manageru

Alternativa za Intel HAXM emulator je koristiti neki od eksternih emulatora, naprimjer Genymotion emulator, ali u ovom radu će se koristiti HAXM. Na nekim računalima je potrebno dopustiti vršenje procesnih simulacija u BIOS-u. Nakon navedenih koraka, računalo je spremno za razvoj android aplikacija te emulaciju istih.

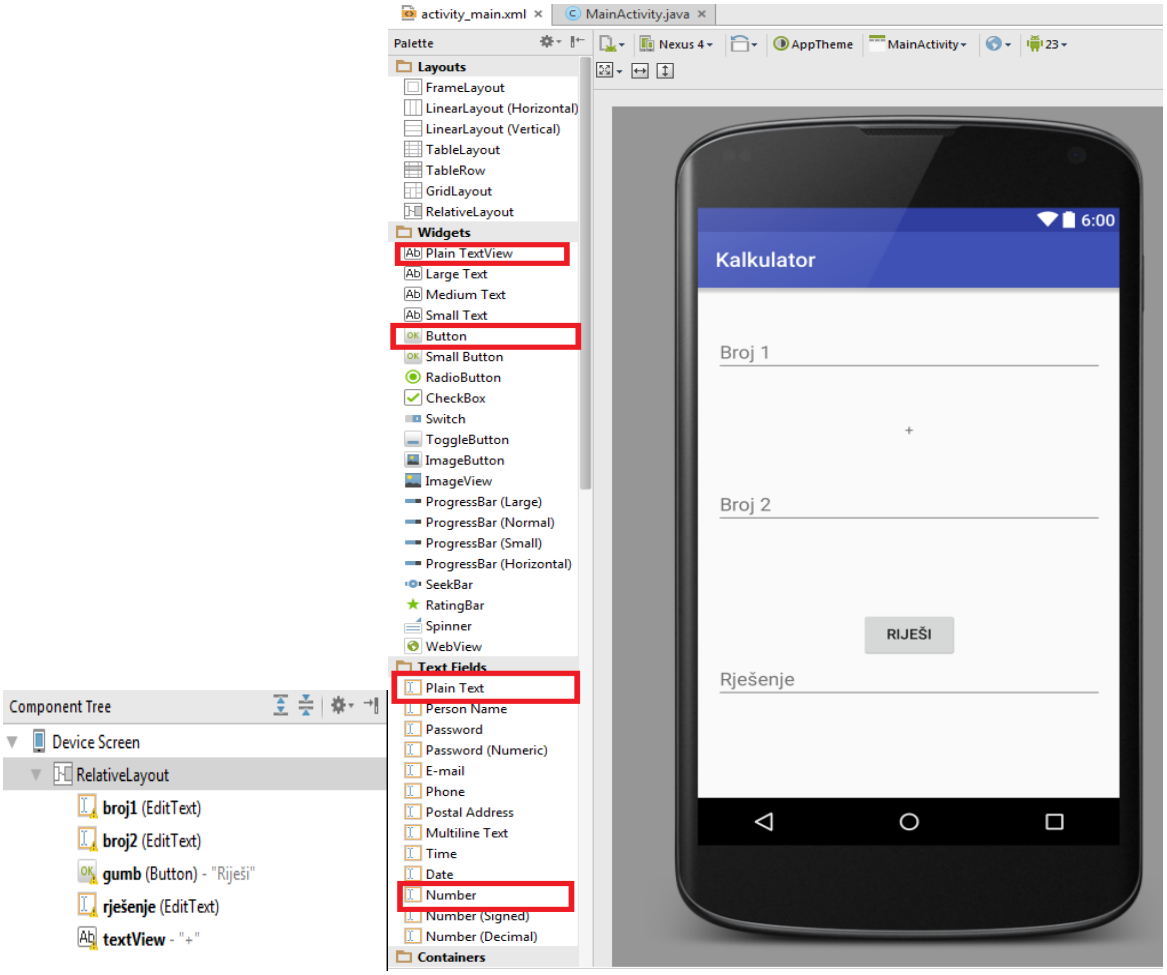
3. Eksperimentalni dio

Ovo poglavlje se temelji na praktičnim i edukacijskim primjerima koji će biti u redoslijedu inkrementalne složenosti, što znači da će svaki primjer implementirati znanja naučena iz prethodnih primjera.

Primjer 1: Jednostavni kalkulator

Ovaj primjer se bazira na povezivanju tekstualnih prozora i gumba sa kodom, te pisanja koda za elementarni kalkulator. Zbog izbjegavanja redundantnog koda, kalkulator će obavljati samo funkciju zbrajanja (za ostale funkcije je potrebno pisati gotovo isti kod uz sitne,jednostavne preinake).

Postupak:
Prvo se postavljaju 2 number text polja, 1 plain text view (za rješenje) i 1 gumb i stavljaju im nazivi kao na popisu komponenata (slike 3.10 i 3.11).



Slika 3.10. Predodžba komponenata koje se koriste

Slika 3.11. Predodžba dizajna sučelja

Dizajn je spreman, sada se piše kod (lokacija java datoteke: app > java > MainActivity.java)

Java kod:

```
package plasaj_vuka.kalkulator;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

    public void onButtonClick(View v) {
        EditText e1= (EditText)findViewById(R.id.broj1);
        EditText e2= (EditText)findViewById(R.id.broj2);
        TextView t1= (TextView)findViewById(R.id.rješenje);
        int num1 = Integer.parseInt(e1.getText().toString());
        int num2 = Integer.parseInt(e2.getText().toString());
        int zbroj = num1 + num2 ;
        t1.setText(Integer.toString(zbroj));
    }
}
```

Biblioteke koje su nužne za normalan rad aplikacije, stoga se moraju dodati su:

```
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
```

Kratko pojašnjavanje logike koda : Kreirana je funkcija *onButtonClick* koja će se izvršiti svaki puta kada se klikne na gumb koji je ubačen u aktivnost. U toj funkciji se spremaju zapisane vrijednosti iz tekstualnih prozora „broj1“ i „broj2“ i spremaju se u varijable e1 i e2 u obliku *EditText* tipa podatka.*TextView t1* je predmet gdje će se ispisati rješenje.S obzirom da nije moguće zbrajati *EditText* tipove podataka jer oni nisu nužno samo brojevi već i tekst, potrebno je pretvoriti *EditText* tip podataka u integer, koji se može podvrgnuti matematičkim operacijama. Za to se koristi *parseInt*, u kojem slučaju se prvo pretvara text u string, te napokon string u int (integer).

Tada se mogu zbrojiti vrijednosti unesene pod *Broj1* i *Broj2*. Taj zbroj je integer, koji će se pretvoriti u tekst za ispis inverznom logikom; pretvara se integer u string i zatim string u text. Gumb se povezuje sa funkcijom tako da se u postavkama gumba i pod opcijom „*onClick*“ odabere „*onButtonClick*“ što znači da će pokretati tu funkciju kada gumb bude pritisnut.

Slika 3.12 Predodžba testiranja aplikacije pomoću emulatora

Primjer 2: Polja za lozinku i toast poruke

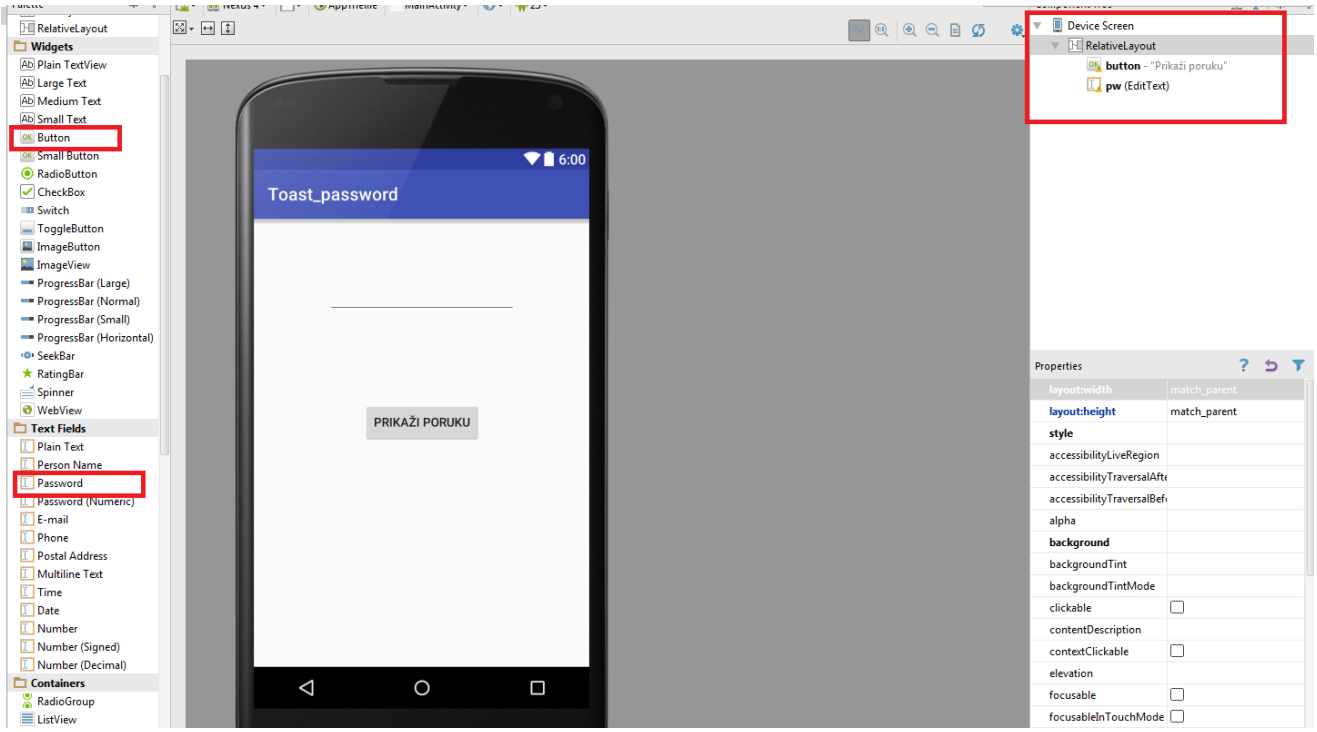
Ovaj primjer će implementirati biblioteku android.widget.Toast.

Toast biblioteka dozvoljava ispis privremene poruke na zaslonu. To se zove toast poruka. Cilj ovog primjera je da se upiše poruka u polje za lozinku i da se pritiskom na gumb ta poruka ispiše na zaslonu.

Dizajn UI-a

Za funkcionalnost ove aplikacije potrebne su dvije komponente: tekst prozor za lozinke (razlika između tog prozora i običnog je ta što se u prozoru za lozinku ne vidi upisana poruka) i običan gumb.

Id textboxa je „*pw*“ , a Id gumba je „*button*“.



Slika 3.13 Predodžba UI Dizajna

Kod:

```
package plasaj_vuka.toast_password;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   private EditText pass;
   private Button btn;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      addListenerOnButton();
   }
   public void addListenerOnButton() {
     pass = (EditText)findViewById(R.id.pw);
      btn =(Button)findViewById(R.id.button);
      btn.setOnClickListener(
            new View.OnClickListener() {
               @Override
```

```
            public void onClick(View v) {
                Toast.makeText(
                        MainActivity.this, pass.getText(),
                        Toast.LENGTH_SHORT
                ).show();
            }
        }
    );
  }
}
```

Logika koda:

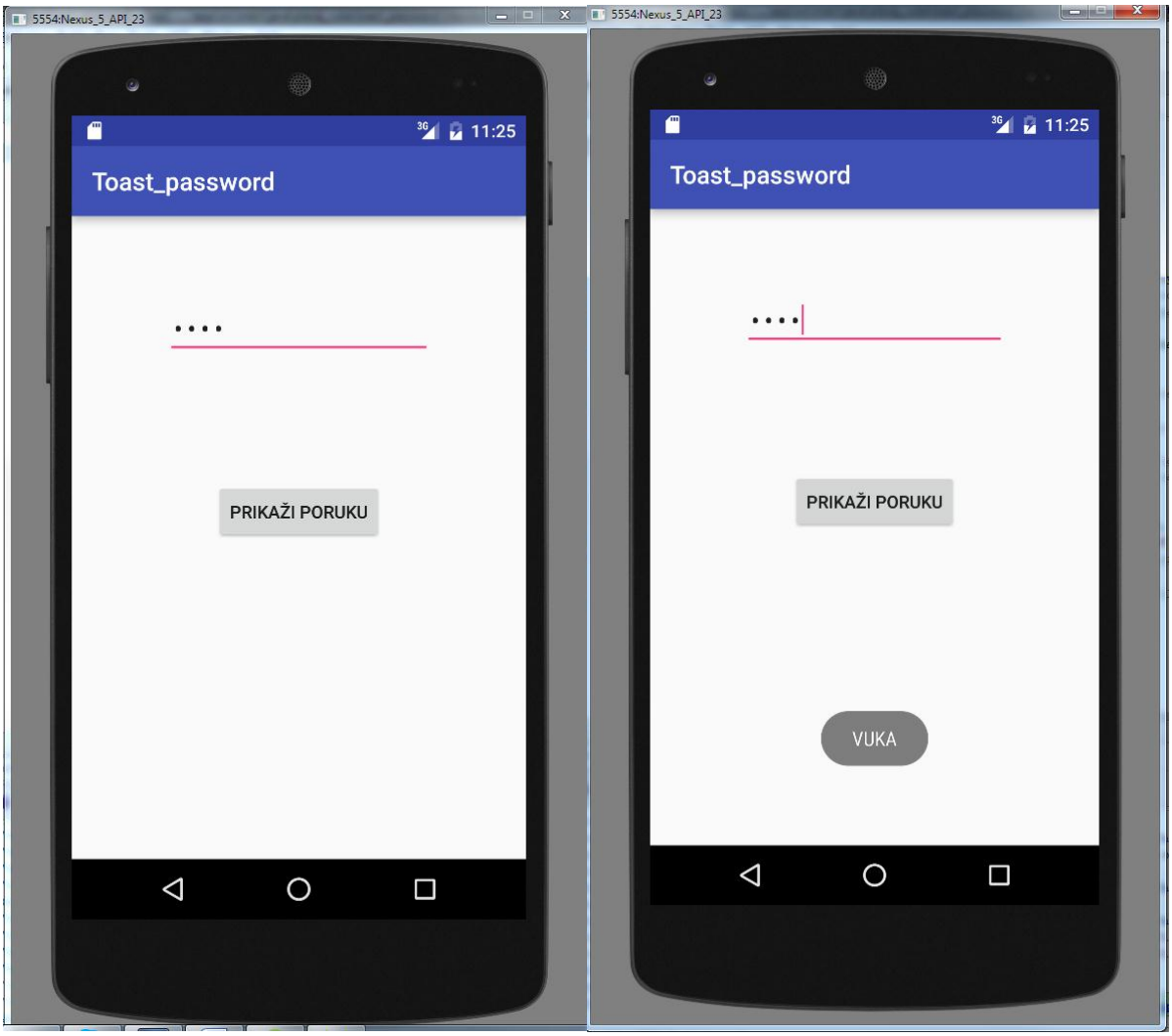
Cilj je da se poruka unesena u obliku lozinke pritiskom na gumb ispiše na zaslonu kao toast poruka. Da bi se to ostvarilo, potrebno je napraviti metodu (funkciju) koja će na klik gumba obaviti traženu operaciju. Tu se metodu (funkciju) pokreće unutar glavne funkcije.

Ta funkcija će se zvati *addListenerOnButton* (hrv. Stavi slušaoca na gumb). Prvo, unutar glavne funkcije (*public class MainActivity {}*) treba deklarirati varijablu pass koja je *editText* tip podatka, i varijablu *btn* koja je gumb. Tada se te varijable povezuju (eng. Castamo) unutar podfunkcije *addListenerOnButton* na komponente. Varijablu *pass* treba povezati sa komponentom textboxa (id:*pw*) a varijablu *btn* sa gumbom (id:*button*).

Sad kada su sve komponente povezane sa logičkim varijablama u kodu, moguće je upravljati komponentama. Unutar funkcije *addListenerOnButton()* se stavlja podfunkcija koja će prilikom klika na gumb „uhvatiti“ poruku iz textboxa i ispisati je. Ta je podfunkcija nazvana *btn.setOnClickListener ()*. Unutar nje je pozvan *Toast* iz dodane biblioteke.

Da bi *Toast* poruka mogla biti ispisana, potrebna su joj tri argumenta koji su navedeni: *(MainActivity.this, pass.getText()),length_SHORT*. Prvi argument je funkcija za koju toast ispisuje poruku, druga varijabla sadrži vrijednost unesene tajne poruke u polju za lozinke a treća određuje duljinu poruke (SHORT ili LONG).Naredba *.show()* prikazuje poruku.

Nakon što je cijela funkcija slušaoca na gumb sa pripadajućom podfunkcijom napisana, preostaje izvršiti tu funkciju (*addListenerOnButton*) u glavnoj aktivnosti (*MainActivity*) tako da je se pozove.

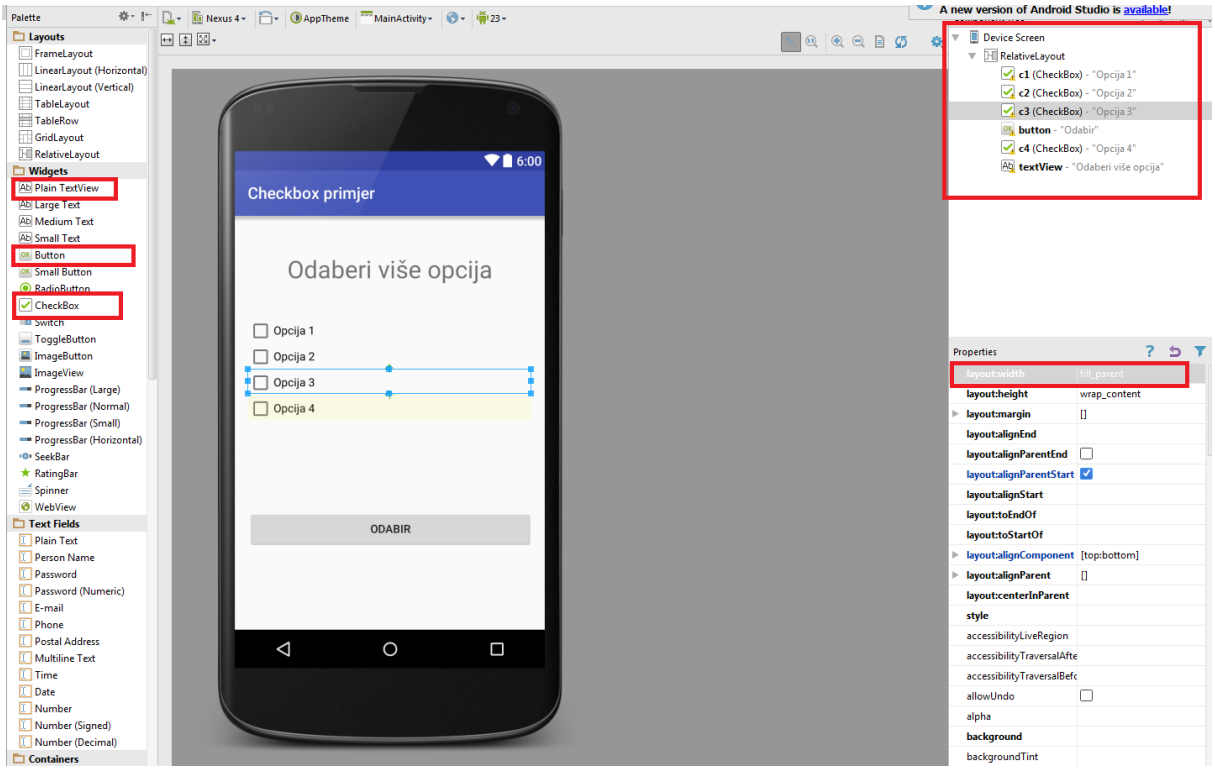


Slika 3.14 i 3.15. Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 3: Primjena komponenata Checkbox i Radio button

Checkbox – odabir **i** tipa

Za UI je potreban odabran broj checkboxova (u ovom slučaju 4) i gumb. Zbog toga jer android većinom koristi zaslone sa dodirnim senzorom, komponente koje se biraju su razvučene po širini (fill parent by width). Time je povećana površina koju korisnik može označiti pritiskom na zaslon.



Slika 3.16 Predodžba dizajna sučelja i popis komponenti

Kod :

```
package plasaj_vuka.checkboxprimjer;

import android.content.DialogInterface;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    private CheckBox ch1,ch2,ch3,ch4;
    private Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        addListenerOnButton();
        addListenerOnCheckBox();
    }
```

```
   public void addListenerOnCheckBox(){
      ch1 = (CheckBox)findViewById(R.id.c1);
      ch1.setOnClickListener(
            new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                  if(((CheckBox)v).isChecked()()){
                     Toast.makeText(MainActivity.this, "Opcija 1 je
odabrana",Toast.LENGTH_LONG).show();
               }
            }
            }
      )
   }
   public void addListenerOnButton() {
      ch1 = (CheckBox)findViewById(R.id.c1);
      ch2 = (CheckBox)findViewById(R.id.c2);
      ch3 = (CheckBox)findViewById(R.id.c3);
      ch4 = (CheckBox)findViewById(R.id.c4);
      btn = (Button)findViewById(R.id.button);

      btn.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                  StringBuffer result = StringBuffer();
                        result.append("Opcija 1 :").append(ch1.isChecked());
result.append("\n Opcija 2 :").append(ch2.isChecked());
                        result.append("\n Opcija 3 :").append(ch3.isChecked());
                        result.append("\n Opcija 4 :").append(ch4.isChecked());

                        Toast.makeText(MainActivity.this,result.toString(),
                        Toast.LENGTH_LONG).show();
                     }
                  }

      );
   }
}
```

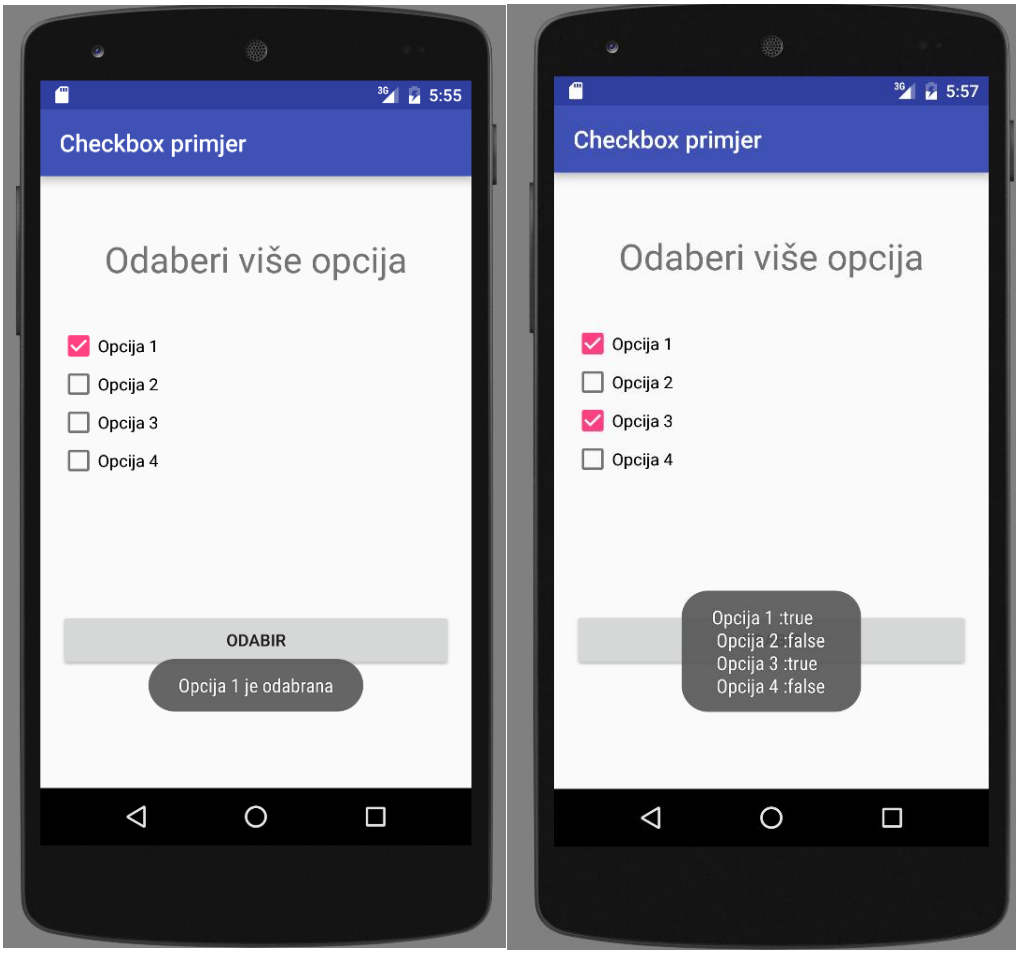
Logika koda:

Cilj ovog primjera je da pri odabiru jedne od opcija korisnik dobije Toast poruku da je odabrao spomenutu opciju, i da pritiskom na gumb „ODABIR“ dobije Toast poruku sa izvješćem koje opcije su odabrali, a koje nisu.

Za to će biti potrebne 2 funkcije; podfunkcija *addListenerOnCheckBox()*, koja će ispisati poruku prilikom označavanja opcija, i podfunkcija *addListenerOnButton()*, koja će ispisati poruku prilikom pritiska na gumb. Obje podfunkcije je potrebno izvršiti u *MainActivity()* ,to jest, glavnoj aktivnosti da bi se implementirale. S obzirom da te funkcije ne vraćaju nikakve podatke, deklarirane su kao public **void** funkcije.

Potrebno je deklarirati varijable *ch1, ch2, ch3, ch4* kao checkbox varijable i *btn* kao varijablu gumba. U podfunkcijama se castaju, to jest, pridružuju varijable njihovim odgovarajućim elementima na sučelju. Za ispis više dijelova teksta (stringa) se koristi **stringBuffer**, koji služi kao spremište za više dijelova stringa, pa će se tako ispisati za svaku zasebnu opciju je li označena ili nije. Pomoću operacije **append**, popisu opcija su dodane binarne vrijednosti koje indiciraju je li opcija označena ili nije (*.append(ch1.isChecked()*).

Za poruku prilikom označavanja, koristi se **if** funkcija, koja je kao uvjet također koristila .isChecked operaciju.



Slika 3.17 Predodžba Simulacije aplikacije u emulatoru na računalu

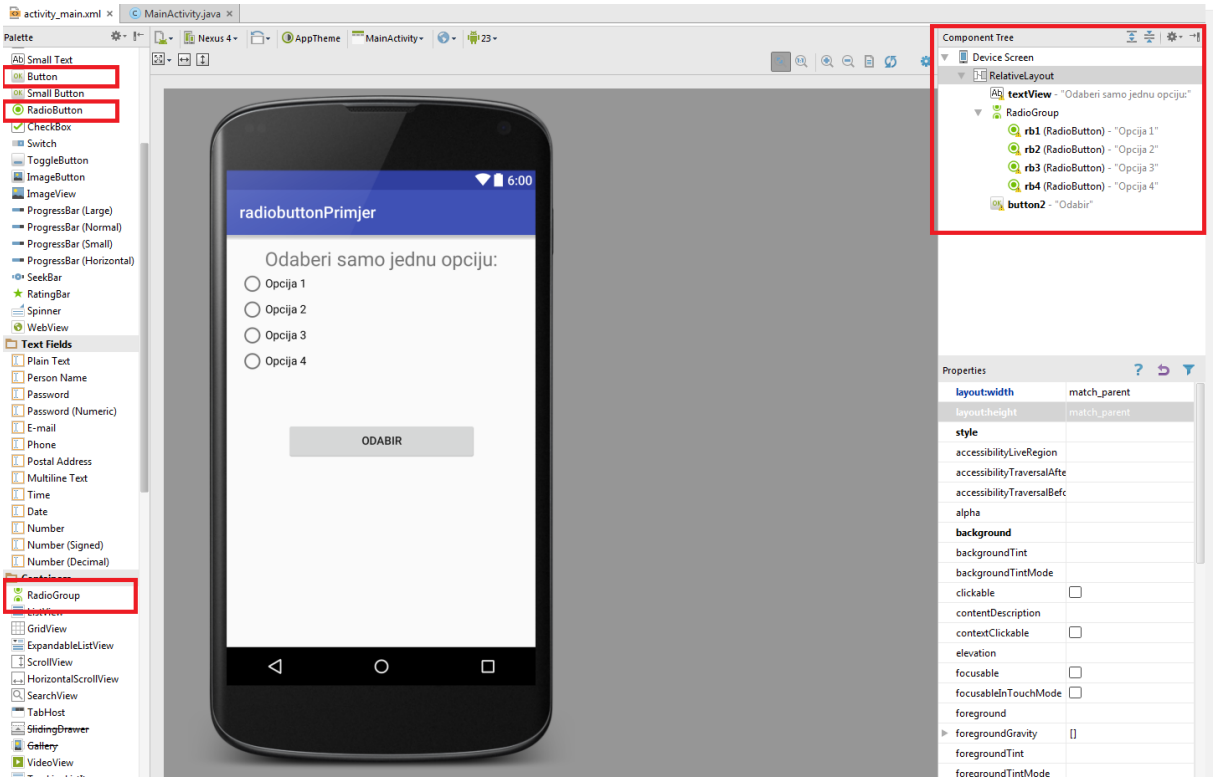
Pri odabiru opcije (funkcija *addListenerToCheckBox*) i pri pritisku gumba (funkcija *addListenerToButton*) korisnik dobiva Toast poruku sa obavijesti koja je opcija odabrana, odnosno koje su opcije odabrane te koje nisu.

Radio button – odabir **ili** tipa

Za ovaj primjer potrebne su sljedeće komponente: 4 radio gumba, 1 običan gumb, i radio grupa. Radio grupa je potrebna da se u nju smjeste radio gumbi. Ako nisu u grupi program ne zna da su ta 4 gumba povezana u odabiru ili tipa, što znači da samo jedan može biti odabran u određenom trenutku.

Radio grupa nema određen ID, ali s obzirom da je potreban za ovaj primjer, nužno mu je dodijeliti ID tako da odabirom komponente radio grupe u svojstvima (properties) u prozor ID treba unijeti proizvoljni ID

Slika 3.18. Predodžba RadioGroup ID-a



Slika 3.19. Predodžba UI dizajna

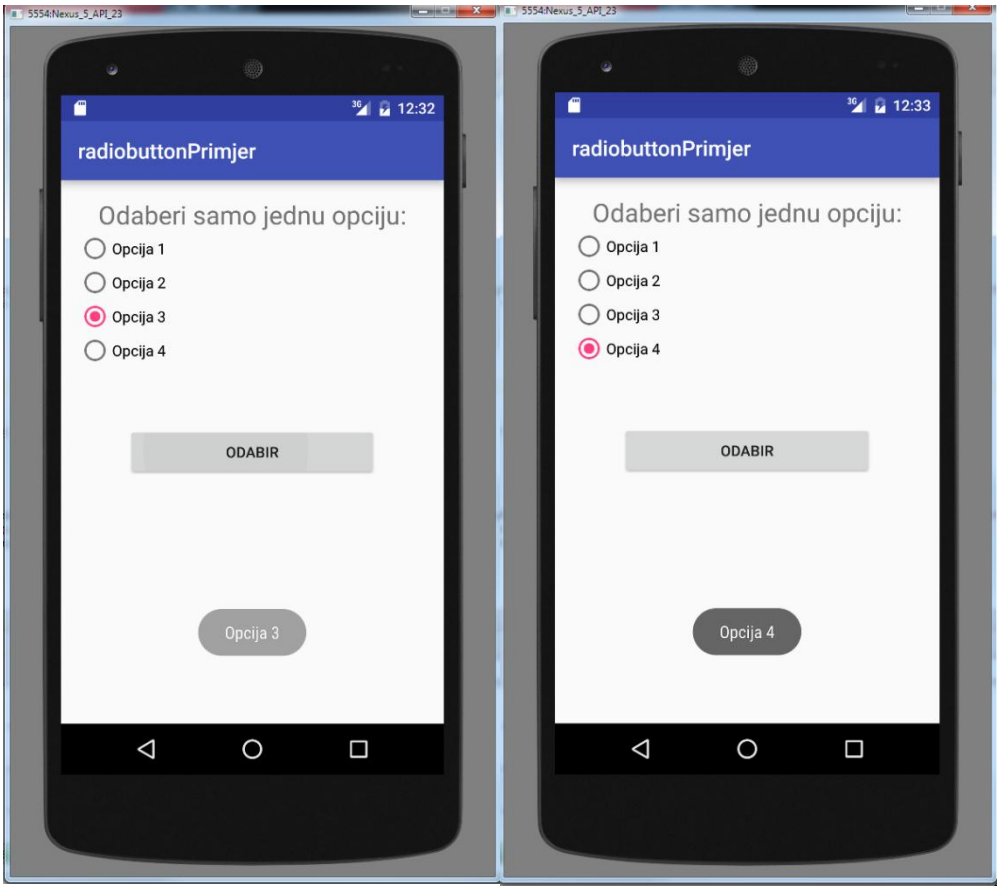
Kod:

```
package plasaj_vuka.radiobuttonprimjer;

import android.provider.MediaStore;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;
```

```
public class MainActivity extends AppCompatActivity {
   private static RadioGroup rgroup;
   private static RadioButton rbtn;
   private static Button btn;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      onClickListenerForButton();
   }

   public void onClickListenerForButton(){
      rgroup = (RadioGroup)findViewById(R.id.rgroupid);
 btn = (Button)findViewById(R.id.btn);
      btn.setOnClickListener(
            new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                  int odabrani_id =rgroup.getCheckedRadioButtonId();
                  rbtn = (RadioButton)findViewById(odabrani_id);
Toast.makeText(MainActivity.this,rbtn.getText().toString(),Toast.LENGTH_SHORT).show(
);
               }
            }
      );
   }
}
```

Logika koda:

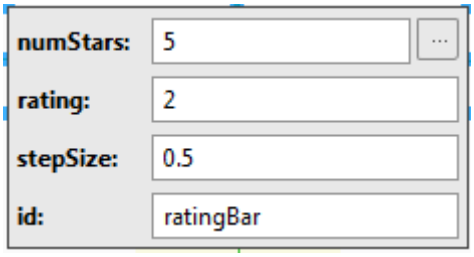
Princip je gotovo isti kao i kod checkbox varijante. Dovoljna je samo jedna deklarirana varijabla za radio gumb iako ih je četiri u dizajnu, zato jer samo jedan može biti označen u određenom trenutku. Varijabla rgroup se povezuje sa komponentom sa sučelja id-a *rgroupid*. Time je radio grupa definirana. Gumb deklariran kao btn je povezan sa gumbom za odabir iz sučelja. U podfunkciji je deklarirana integer varijabla koja će vraćati integer koji sadrži vrijednost id-a odabranog radio gumba (*.getCheckedRadioButtonId()*). Tu integer varijablu treba povezati (castati) na radio button varijablu. Na kraju je potrebno staviti ispis u obliku toast poruke i pokrenuti podfunkciju u glavnoj funkciji (MainActivity).



Slika 3.20. Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 4: Rating bar

Kod RatingBar komponente,namještanje svojstva je sljedeće:
numStars – broj zvjezdica koje predstavljaju ocjenu ili rating
rating - početna ocjena koja će biti postavljena
stepSize - najmanja vrijednost za koju se može povećavati/smanjivati rating



Slika 3.21 Predodžba liste svojstva koje se namještaju za komponentu Rating bar

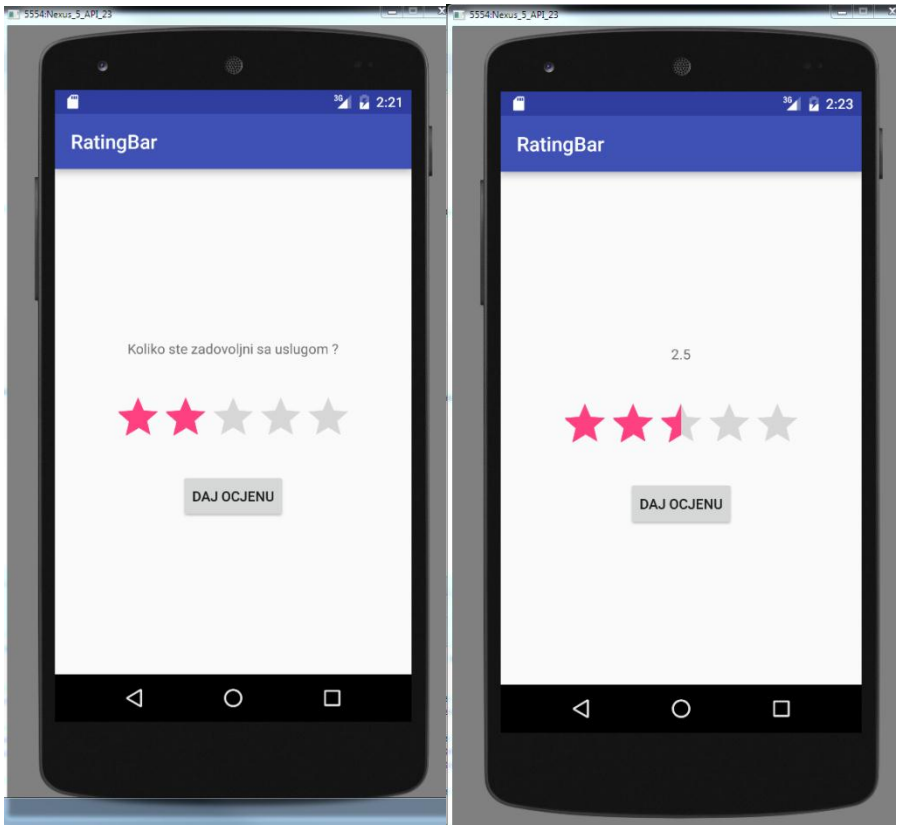
Kod:

```
package plasaj_vuka.ratingbar;

import android.media.Rating;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RatingBar;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   private static Button btn;
   private static TextView text;
   private static RatingBar rbar;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      listenerOnRatingBar();
      onButtonClickListener();
   }
   public void listenerOnRatingBar(){
      rbar = (RatingBar)findViewById(R.id.ratingBar);
      text = (TextView)findViewById(R.id.textView);
      rbar.setOnRatingBarChangeListener(
            new RatingBar.OnRatingBarChangeListener() {
               @Override
               public void onRatingChanged(RatingBar ratingBar, float rating, boolean
fromUser) {
                  text.setText(String.valueOf(rating));
               }
            }
      );

   }
   public void onButtonClickListener(){
      rbar = (RatingBar)findViewById(R.id.ratingBar);
      btn= (Button)findViewById(R.id.button);
      btn.setOnClickListener(
            new View.OnClickListener() {
```

```
            @Override
            public void onClick(View v) {
                Toast.makeText(MainActivity.this,
String.valueOf(rbar),Toast.LENGTH_SHORT).show();
            }
        }
    );
  }
}
```

Logika koda:

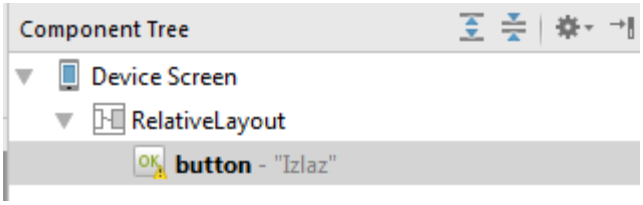
Kao i u prethodnim primjerima, kreiraju se dodatne metode koje će reagirati na unesenu ocjenu i na pritisak gumba. Unesena ocjena će se ispisati iznad rating bara a prilikom pritiska na gumb će se ispisati privremena toast poruka na ekran sa ocjenom koja je dodijeljena. Te metode su *listenerOnRatingBar()* i *onButtonClickListener()*. Obje treba izvršiti u MainActivity. U metodi *listenerOnRatingBar()* je podfunkcija *onRatingChanged*, koja ima definirane varijable *ratingBar*, *rating*, *fromUser*. Ovo su varijable koje su automatski napravljene jer su potrebne za funkciju *onRatingChanged*. Implementirane su iz biblioteke *android.widget.RatingBar*. Ostatak koda ne donosi nikakve novitete u odnosu na prethodne primjere.



Slika 3.22 Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 5: Alert Dialog

Cilj ovog primjera je da pritiskom na gumb „Izlaz“ aplikacija izgradi Alert dijalog prozor koji će tražiti korisnika da potvrdi izlaz iz aplikacije. Pritiskom na „Da“ korisnik izlazi iz aplikacije, dok pritiskom na „Ne“ će se vratiti u aplikaciju. Sučelje je jednostavno, sastoji se od jednog gumba.



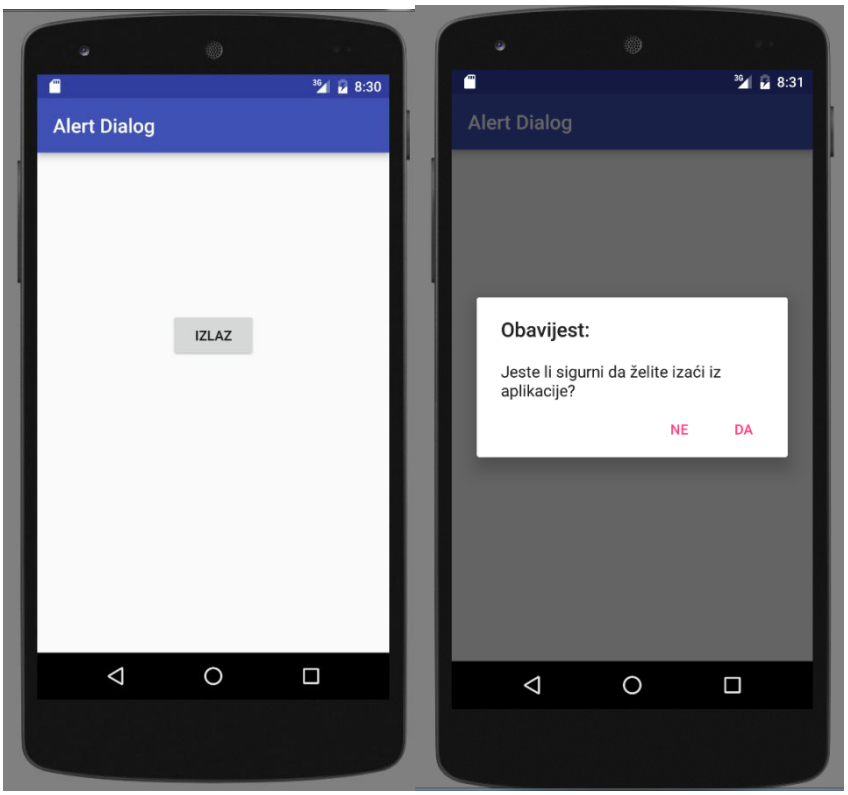
Slika 3.23 Predodžba popisa komponenti sučelja

Kod:

```
package plasaj_vuka.alertdialog;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
   private static Button button;

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

   public void onButtonClickListener () {
      button = (Button)findViewById(R.id.button);
      button.setOnClickListener(
            new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                  AlertDialog.Builder  abuilder = new AlertDialog.Builder(MainActivity.this);
                  abuilder.setMessage("Jeste li sigurni da želite izaći iz aplikacije?")
                        .setCancelable(false)
                        .setPositiveButton("Da", new DialogInterface.OnClickListener() {
                           @Override
```

```
                    public void onClick(DialogInterface dialog, int which) {
                        finish();
                    }
                })
                .setNegativeButton("Ne", new DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        dialog.cancel();
                    }
                });
        AlertDialog alert = abuilder.create();
        alert.setTitle("Obavijest:");
        alert.show();

    }
}
);
}
}
```

Logika koda:
Da bi se alert prozor pojavio, potrebno je da se dogodi neka uzročna akcija, a to može biti bilo koja radnja. U ovom primjeru je to pritisak na gumb. Stoga je angažiran slušaoc (listener) na gumb koji će se izvršiti prilikom pritiska (metoda *onButtonClickListener()*). Unutar nje je definirana varijablu gumba i povezana je sa odgovarajućom komponentom sučelja. Nakon toga slijedi kreiranje funkcije listenera unutar koje je definiran *AlertDialog.Builder*. On je zadužen za izgrađivanje alert prozora. Sintaksa za podešavanje tog prozora je navedena u teoretskom dijela. Ovdje je pod pozitivnu opciju stavljen izlaz iz aplikacije operacijom *finish()*, a pod negativnu opciju stavljen ostanak u aplikaciji operacijom *dialog.cancel()*. Još je preostalo staviti naredbu za izgradnju dijalog prozora, i dati naredbu da se pokrene. Time je podfunkcija gotova, te se izvršava u glavnoj aktivnosti MainActivity.



Slika 3.24 Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 6: Prijelaz između aktivnosti sa intent funkcijom

Pritiskom na gumb „Želim u drugu aktivnost“ korisnik se prebacuje iz prve u drugu aktivnost. Klikom na gumb „natrag“ na android sučelju koji je prikazan kao strelica se vraća natrag u prvu aktivnost.

Kod za AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="plasaj_vuka.intent_activity_test2">

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

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity
      android:name=".Secondary"
```

```
        android:label= "activity_secondary" >
      <intent-filter>
        <action android:name="plasaj_vuka.intent_activity_test2.Secondary" />

        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
      </activity>
   </application>

</manifest>
```

Logika koda:
.xml kod je pojašnjen u teoretskom dijelu rada. Napomena; bilo je potrebno dodati intent filter u drugu aktivnost tako da postoji veza između aktivnosti.

Kod za prvu aktivnost:

```
package plasaj_vuka.intent_activity_test2;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
   private static Button btn;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      OnClickButtonListener();
   }
   public void OnClickButtonListener(){
      btn = (Button)findViewById(R.id.button);
      btn.setOnClickListener(
            new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                  Intent intentvarijabla = new Intent
("plasaj_vuka.intent_activity_test2.Secondary");
                  startActivity(intentvarijabla);
               }
            }
      );
```

Logika koda:

U prvoj aktivnosti se nalazi gumb za prijelaz između aktivnosti, to znači da je potrebno angažirati slušaoca (listener) na gumb koji će izvršiti izmjenu aktivnosti. Za to je definirana

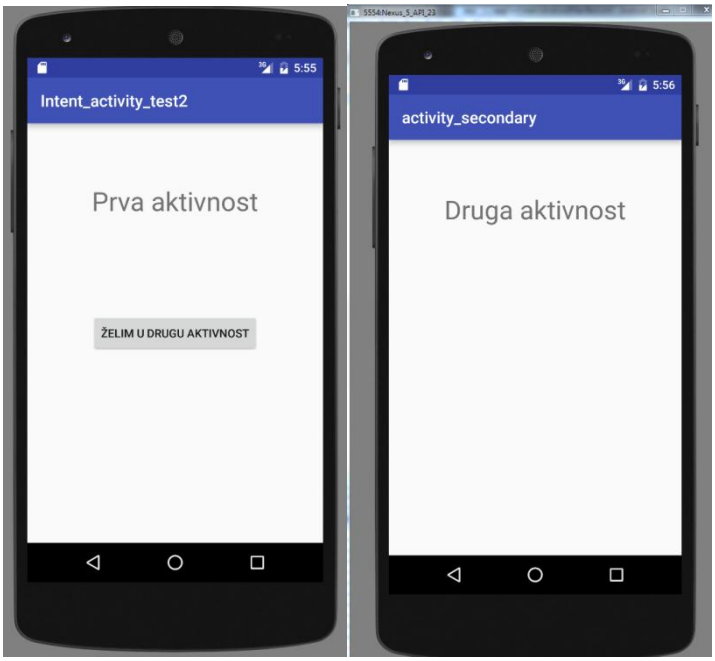
varijabla intent tipa i nazvana je *intentvarijabla*. Ta varijabla prima jedan parametar, i taj parametar je naziv aktivnosti u koju prebacuje. U ovom slučaju to je **plasaj_vuka.intent_activity_test2.Secondary.** Preostalo je pokrenuti tu aktivnost sa naredbom *startActivity*,čiji je parametar varijabla tipa intent, koju je već definirana.

Kod za drugu aktivnost:

```
package plasaj_vuka.intent_activity_test2;

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

public class Secondary extends AppCompatActivity {

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

Logika koda:

S obzirom da druga aktivnost ne izvršava nikakvu akciju, kod je tu jednostavno da izvrši onCreate funkciju.

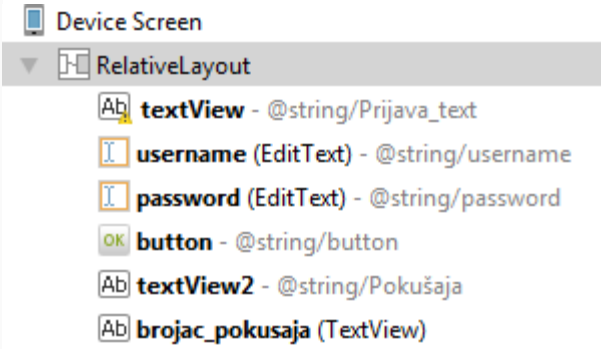


Slika 3.25 Predodžba simulacije aplikacije u emulatoru na računalu

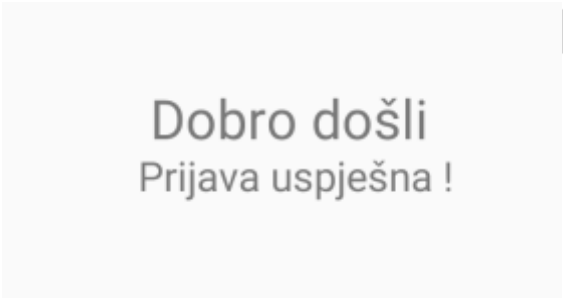
Primjer 7: Sustav za prijavu korisnika

Cilj ovog primjera je da prilikom pritiska gumba nakon točnog unosa korisničkog imena i lozinke, aplikacija dovede korisnika u drugu aktivnost. Ako korisnik pogriješi pri unosu podataka, aplikacija će ga upozoriti. Ako 3 puta pogriješi, aplikacija mu neće dozvoliti da nastavi sa prijavom. Implementira se intent za prelazak između aktivnosti. Bitno je napomenuti da je potrebno dodati intent filter u .xml kod druge aktivnosti.

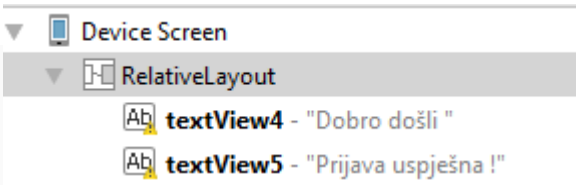
Slika 3.26. Predodžba sučelja prve aktivnosti



Slika 3.27. Predodžba komponenata prve aktivnosti



Slika 3.28. Predodžba sučelja druge aktivnosti



Slika 3.29. Predodžba komponenata druge aktivnosti

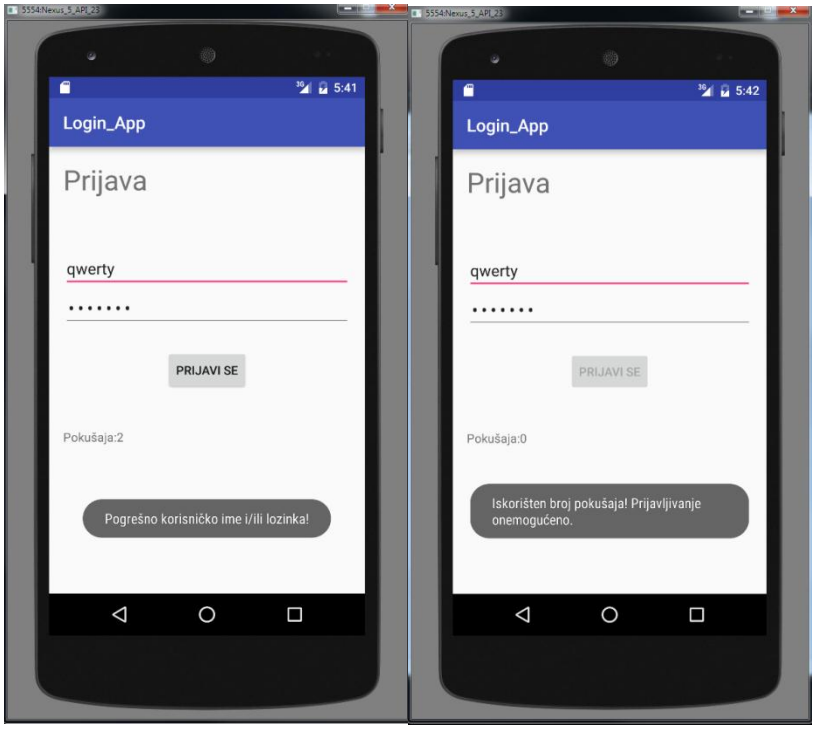
Kod glavne aktivnosti:

```
package plasaj_vuka.login_app;

import android.app.AlertDialog;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
```

```
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    private static EditText username;
    private static EditText password;
    private static TextView attempts;
    private static Button button;
    int brojač = 3;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Login();
    }
    public void Login(){
        username = (EditText) findViewById(R.id.username);
        password = (EditText) findViewById(R.id.password);
        attempts = (TextView) findViewById(R.id.brojac_pokusaja);
        button = (Button)findViewById(R.id.button);
        attempts.setText(Integer.toString(brojač));
        button.setOnClickListener(
                new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        if(username.getText().toString().equals("aplasaj")&&
                                password.getText().toString().equals("vuka"))

                        {
                            Toast.makeText(MainActivity.this,"Točno korisničko ime i
lozinka!",Toast.LENGTH_SHORT).show();
                        Intent prijavljivanje = new Intent("plasaj_vuka.login.second");
                            startActivity(prijavljivanje);
                        }
                        else
                        {
                            Toast.makeText(MainActivity.this,"Pogrešno korisničko ime i/ili
lozinka!",Toast.LENGTH_SHORT).show();
                        brojač--;
                            attempts.setText(Integer.toString(brojač));
                        }
                        if (brojač == 0)
                        {
                            button.setEnabled(false);
                            Toast.makeText(MainActivity.this,"Iskorišten broj pokušaja!
Prijavljivanje onemogućeno.",Toast.LENGTH_SHORT).show();
                        }

                    }
                }
```

```
        );
    }
}
```

Logika koda:
Kod deklariranja varijabli je deklarirana varijabla tipa Integer, imena brojač. Ta varijabla definira koliko će pokušaja korisnik imati da ispravno unese podatke. U podfunkciji koja je slušaoc na gumb (*onClickListener*) koristeći if/else logičke operacije, razvojni inženjer programira aplikaciju da u slučaju da je korisničko ime „*aplasaj*“, a lozinka „*vuka*“, aplikacija ispiše poruku da su podatci točni i koristeći intent, prebacujemo u sekundarnu aktivnost, do koje je jedini pristup prijavljivanjem kroz glavnu aktivnost. Kod za prijavljivanje je pisan isključivo u glavnoj aktivnosti. Ako podatci za prijavu nisu točni, ispisuje se poruka da prijava nije ispravna, i vrijednost integera brojač se smanjuje za 1 (-- znači da se dekrementira za 1,kao što ++ znači inkrementiranje za 1). Vrijednost brojača je vezana na varijablu *attempts*, tako da ju je moguće ispisati unutar glavne aktivnosti unutar tekstualnog prozora, da bi korisnik mogao pratiti koliko još pokušaja ima na raspolaganju. Operacija *if* je korištena još jednom, za slučaj kada je vrijednost pokušaja pala na nulu. Ako padne na nulu, gumb za prijavu se deaktivira (*button.setEnabled(false)*) i ispisuje se poruka koja korisnika obavještava o tome.



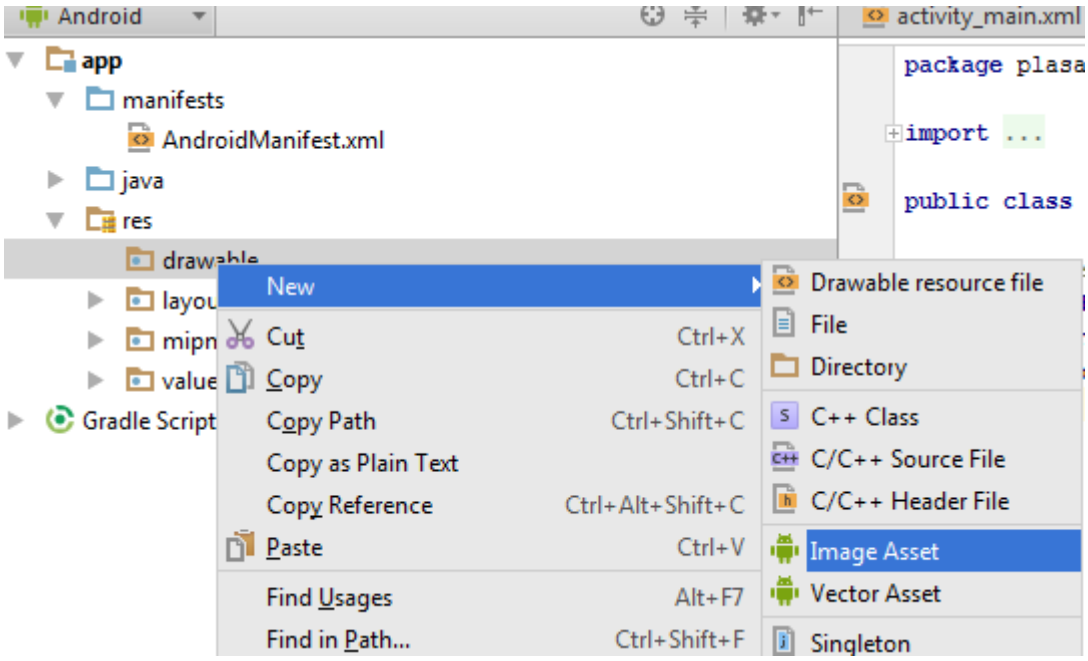
Slika 3.30 Predodžba simulacije prilikom pogrešnog unosa Slika 3.31 Predodžba simulacije prilikom 0 preostalih pokušaja

Slika 3.32 Predodžba simulacije prilikom točnog unosa podataka za prijavu

Primjer 8: Implementacija ImageView elementa, grupe

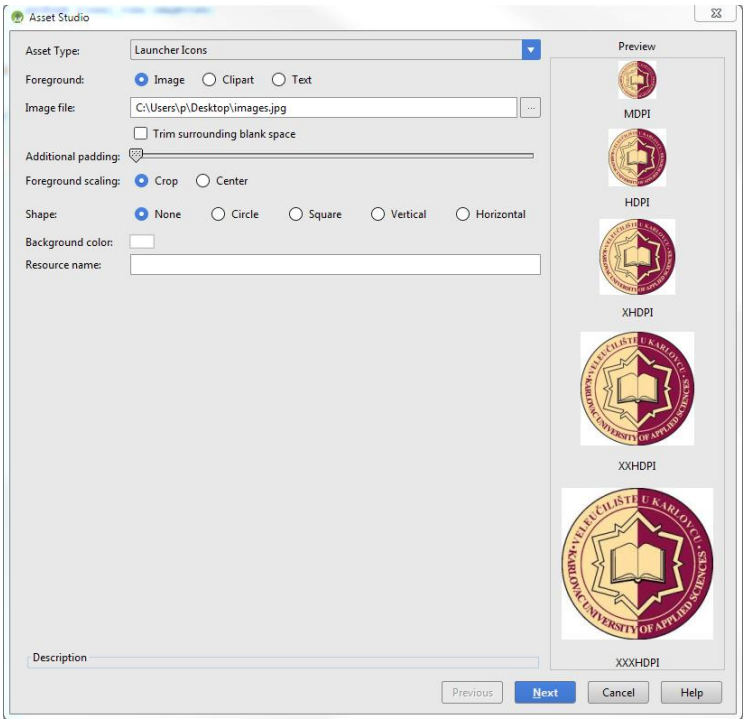
Cilj ove aplikacije će biti da pritiskom gumba korisnik mijenja sliku koja se prikazuje na zaslonu.

Kao prvo, potrebno je dodati nekoliko slika u datoteku sa resursima. To se radi dodavanjem image asseta, pomoću asset studia. (slika 3.33)



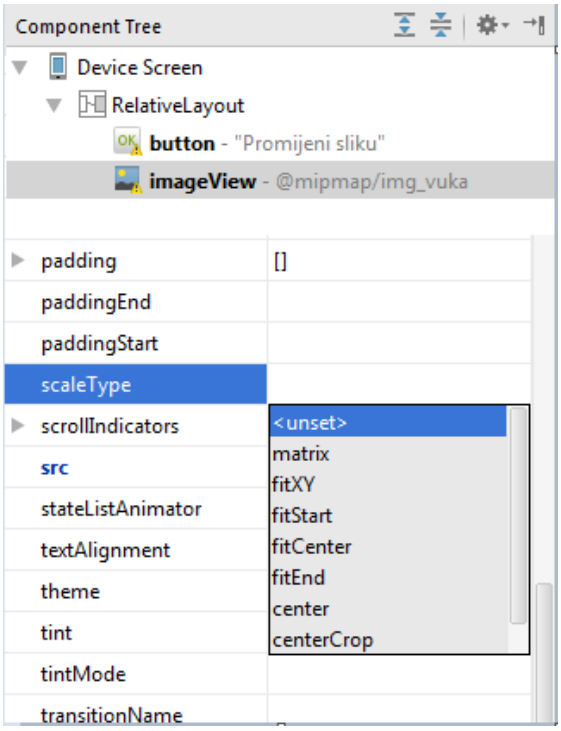
Slika 3.33. Predodžba dodavanja novog Image Asset-a.

Kod imenovanja slika, bitno je imati na umu da se svi resursi imenuju malim početnim slovom, ili brojkom. Nakon dodavanja slike, njene verzije u svim odabranim veličinama bi trebale biti prikazane unutar datoteke res, u odabranoj poddatoteci. U ovom slučaju se ta datoteka zove *mip_map*. Nije bitno koja je veličina odabrana za sliku, android studio će ga prilagoditi automatski. Nakon što su resursi spremni, prelazi se na dizajn sučelja.



Za dizajn sučelja potreban je gumb koji će davati naredbu za prebacivanje slika na zaslonu, te ImageView komponenta. Namještanje ImageViewa će namještati i sliku koja će se u njemu prikazivati. U svojstvima komponente ImageView nalazi se opcija scaleType koja će određivati kako će, i hoće li uopće, slika skalirati sa ImageView komponentom.

Slika 3.34 Predodžba podešavanja Image Asset-a



Slika 3.35 Predodžba popisa komponenti i scaleType svojstva ImageView-a

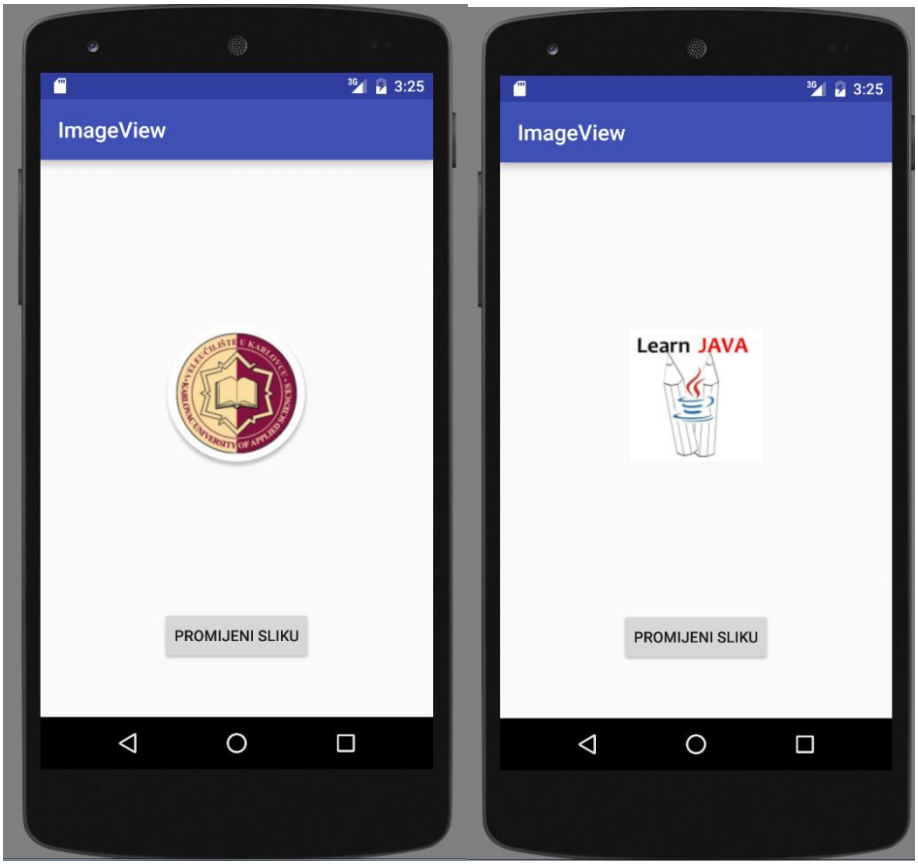
Kod:

```
package plasaj_vuka.imageview;

import android.media.Image;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {
   private static ImageView iv;
   private static Button btn;
   private int trenutnaSlika;
   int[] images = {R.mipmap.img_vuka,R.mipmap.slika2,R.mipmap.ic_launcher};
   public void onButtonClick() {
      iv = (ImageView)findViewById(R.id.imageView);
      btn = (Button)findViewById(R.id.button);
      btn.setOnClickListener(
            new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                  trenutnaSlika++;
                  trenutnaSlika=trenutnaSlika % images.length;
                  iv.setImageResource(images[trenutnaSlika]);
               }
            }
      );
   }
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      onButtonClick();
   }
}
```

Logika koda:

Zadaća koda je bila da poveže komponente sa varijablama, i da pomoću tih varijabli izvrši zamjenu slika unutar komponente ImageView. Deklarirane su varijable *iv* (ImageView tip varijable), Gumb *btn*, i varijabla *trenutnaSlika* koja je varijabla za redni broj, odnosno, index trenutno aktivne slike (slike koja je prikazana na zaslonu). Dodatno je deklarirano polje integera naziva *images*. Polja su skupovi parametara istog tipa. U ovom slučaju, integera. U polje su unesene tri slike koje su u datoteci resursa mipmap. Iz sintakse koda je evidentno da se slike deklariraju kao R.[naziv_datoteke_u_kojoj_ je_slika].[naziv_slike]. Zatim treba napraviti podfunkciju čija je zadaća da obavi prebacivanje slika prilikom pritiska na gumb, naziva *onbuttonClick()*. Povezuju se varijable iv i btn sa komponentama ImageView i gumbom i stvara se podfunkcija *OnClickListener()*. Ono što je unutar nje će se odviti

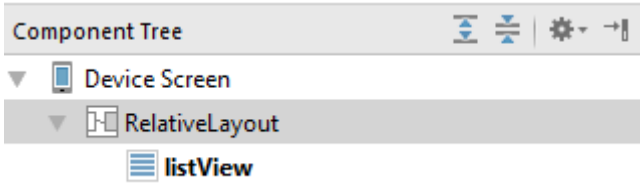
pritiskom na gumb. Index trenutno prikazane slike će se povećati za 1, i postaviti će se nova slika čiji je index novoodređeni integer unutar parametra *trenutnaSlika*. Preostalo je još pokrenuti podfunkciju unutar glavne aktivnosti.



Slika 3.36. Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 9: ListView

Svrha ovog primjera je da ispiše listu unutar glavne aktivnosti, te da pritiskom na bilo koju od vrijednosti navedenih unutar liste ispiše poruku koja govori koji je redni broj tog podatka i što je u njemu zapisano. Implementira se *ListView* komponenta kao sredstvo za ispis liste (slika 3.37).



Slika 3.37. Predodžba popisa komponenata

Potrebno je kreirati novu .xml datoteku (desni klik miša na layout datoteku unutar res mape i odabrati New .xml layout file. Ukoliko je uspješno kreiran, vidljiv je novi layout.xml unutar mape layout.

S obzirom da se novokreirana datoteka neće koristiti za dizajn sučelja već za popis (listu), potrebno je otvoriti spomenutu datoteku i promijeniti </LinearLayout> u </TextView>

Nova .xml datoteka:

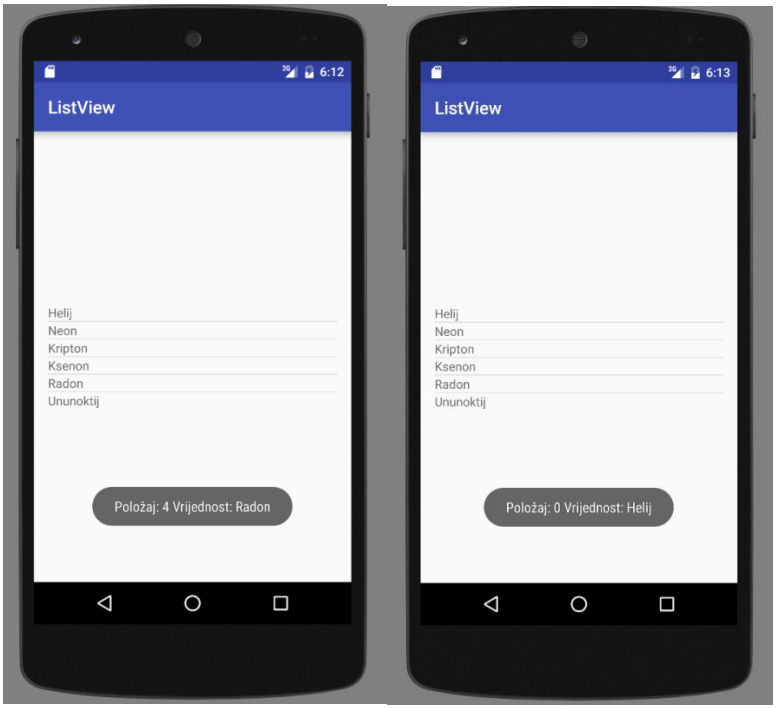
```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

</TextView>
```

Kod glavne aktivnosti:

```
package plasaj_vuka.listview;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.util.List;

public class MainActivity extends AppCompatActivity {
    private static ListView lv;
    private static String[] Plemeniti_plinovi= new String[]
{"Helij","Neon","Kripton","Ksenon","Radon","Ununoktij"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lista();
    }
    public void lista() {
        lv = (ListView)findViewById(R.id.listView);
        ArrayAdapter<String> adapter = new
ArrayAdapter<String>(this,R.layout.layout,Plemeniti_plinovi);
        lv.setAdapter(adapter);
        lv.setOnItemClickListener(
                new AdapterView.OnItemClickListener() {
                    @Override
                    public void onItemClick(AdapterView<?> parent, View view, int position, long
id) {
                        String vrijednost = (String)lv.getItemAtPosition(position);
                        Toast.makeText(MainActivity.this,"Položaj: "+ position + " Vrijednost: " +
vrijednost,Toast.LENGTH_LONG).show();
                    }
                }
```

```
        );
    }
}
```

Logika koda:
Kao i do sada, prvo se deklariraju varijable. Za pohranu svih podataka za listu je potrebno polje podataka tipa String. S obzirom da će se ispisati popis plemenitih plinova, polje je nazvano *Plemeniti_plinovi*, i unesen je popis svih plemenitih plinova. Zatim se u podfunkciji casta (povezuje) varijabla *lv* sa komponentom u sučelju *ListView*. Za pretvorbu liste varijabli u podatke spremne za ispis u obliku ListView-a, koristi se *ArrayAdapter*. On je veza između .xml datoteke i ListView-a. Za funkcionalan adapter su potrebna tri argumenta: Kontekst (*this*), lokacija .xml datoteke (*R.layout.layout*), te podatak (u ovom slučaju to je popis plemenitih plinova). Operacija *.setAdapter (adapter)* će unijeti sve odabrane podatke u ListView koristeći adapter koji je deklariran kao *adapter*. (*ArrayAdapter<String> adapter= new*). Sada se angažira slušaoc na ListView komponentu koja čeka da korisnik pritisne jedan od popisanih elemenata unutar glavne aktivnosti. Kod klika na listu (*onItemClick()*) ispisati će se redni broj, to jest pozicija stavke, i njena vrijednost (njen naziv) u obliku toast poruke. Pozicija i vrijednost se dobivaju pomoću automatski generiranih varijabli prilikom kreiranja *onItemClick()* funkcije, a to su *position,view,* i *id.*

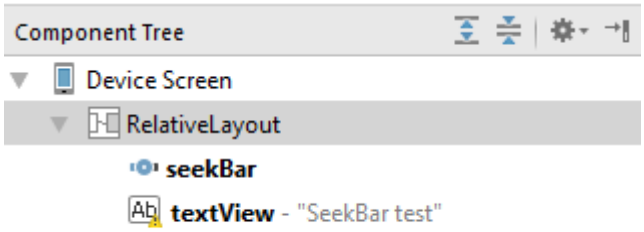


Slika 3.38. Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 10: Implementacija SeekBar komponente

Cilj ovog primjera je da se unutar sučelja glavne aktivnosti postavi seekBar element, da se prati njegova vrijednost (od 0 do 100), te da aplikacija obavještava korisnika o trenutnom

statusu (je li vrijednost promijenjena, mijenja li se trenutno, i koja je vrijednost odabrana). Potrebne komponente su tekstualni prozor i SeekBar (slika 3.39).



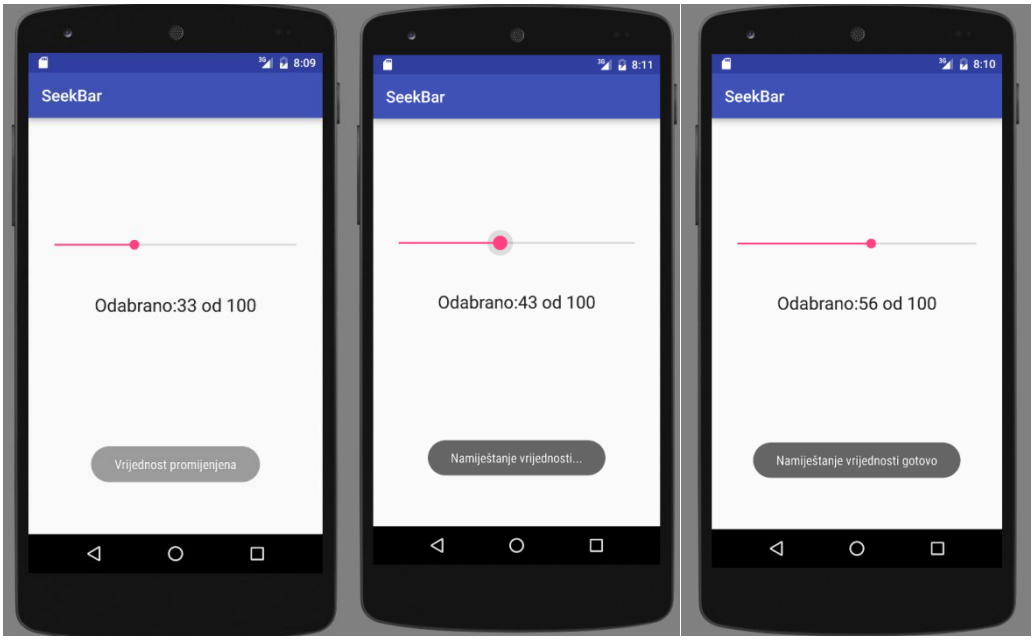
Slika 3.36. Predodžba popisa komponenata

Kod:

```
package plasaj_vuka.seekbar;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   private static SeekBar sbar;
   private static TextView tview;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      seekBar();
   }
   public void seekBar(){
      sbar = (SeekBar)findViewById(R.id.seekBar);
      tview= (TextView)findViewById(R.id.textView);
      tview.setText("Odabrano:" + sbar.getProgress() + " od " + sbar.getMax());
      sbar.setOnSeekBarChangeListener(
            new SeekBar.OnSeekBarChangeListener() {
               int vrijednost;
               @Override
               public void onProgressChanged(SeekBar seekBar, int progress, boolean
fromUser) {
                  vrijednost = progress;
                  tview.setText("Odabrano:" + progress + " od " + sbar.getMax());
                  Toast.makeText(MainActivity.this," Vrijednost promijenjena
",Toast.LENGTH_LONG).show();
               }

               @Override
               public void onStartTrackingTouch(SeekBar seekBar) {
                  Toast.makeText(MainActivity.this," Namiještanje vrijednosti...
",Toast.LENGTH_LONG).show();
```

```
            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
                tview.setText("Odabrano:" + vrijednost + " od " + sbar.getMax());
                Toast.makeText(MainActivity.this," Namiještanje vrijednosti gotovo
",Toast.LENGTH_LONG).show();
            }
        }
    );
  }
```

Logika koda:
Jedini novitet kod ovog programa je taj što se koristi set naredbi vezan za *SeekBar*. Oni su svi objašnjeni u teoretskom dijelu ovog rada. Kreira se podfunkcija koja prilikom promjene pozicije gumba na klizećem izborniku mijenja vrijednost u tekstualnom prozoru ekvivalentno poziciji klizećeg gumba, i obavještava korisnika o stanju seek bara u obliku toast poruke. Ta podfunkcija ima svoje automatski generirane parametre (*seekBar*,*progress*,*fromUser*).

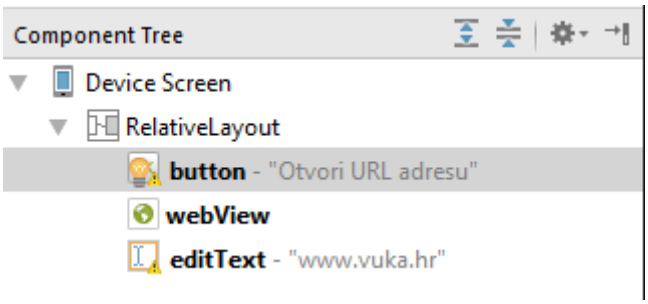


Slika 3.37. Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 11: WebView internet preglednik

Implementacijom WebView komponente, zadaća aplikacije koja se razvija u ovom primjeru je da pritiskom na gumb otvori URL (web adresu) koji smo naveli u tekstualnom prozoru.

Za sučelje su potrebne tri komponente; WebView prozor, gumb, i editText prozor. (slika 3.38)



Slika 3.37. Predodžba popisa komponenata

Prije nego razvojni inženjer počne razrađivati kod, potrebno je promijeniti dozvole koje android operativni sustav pruža aplikaciji. Dozvole se izmjenjuju u AndroidManifest.xml datoteci. Treba dodati sljedeću liniju koda za dozvolu za korištenje interneta:

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

Java kod:

```
package plasaj_vuka.webviewpreglednik;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.webkit.WebView;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
   private static Button btn;
   private static EditText url;
   private static WebView preglednik;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      otvoriURL();
   }
   public void otvoriURL() {
      btn = (Button)findViewById(R.id.button);
      url = (EditText)findViewById(R.id.editText);
      preglednik = (WebView)findViewById(R.id.webView);
      btn.setOnClickListener(
            new View.OnClickListener() {
               @Override
               public void onClick(View v) {
                  String s_url = url.getText().toString();
                  preglednik.getSettings().setLoadsImagesAutomatically(true);
                  preglednik.getSettings().setJavaScriptEnabled(true);
                  preglednik.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);
                  preglednik.loadUrl(s_url);
               }
            }
      );
   }
}
```

Logika koda:
Deklarirane su varijable tipa *Button*, *EditText*, i *WebView* tako da ih se može povezati sa njihovim odgovarajućim komponentama u sučelju aktivnosti. Kreira se podfunkcija koja otvara URL adresu prilikom pritiska na gumb. U *onClickListener()* podfunkciji je nužno podesiti postavke WebView komponente koja je sada povezana na varijablu *preglednik*. Sljedećim linijama koda si postavljene opcije automatskog učitavanja slika i JavaScripta:

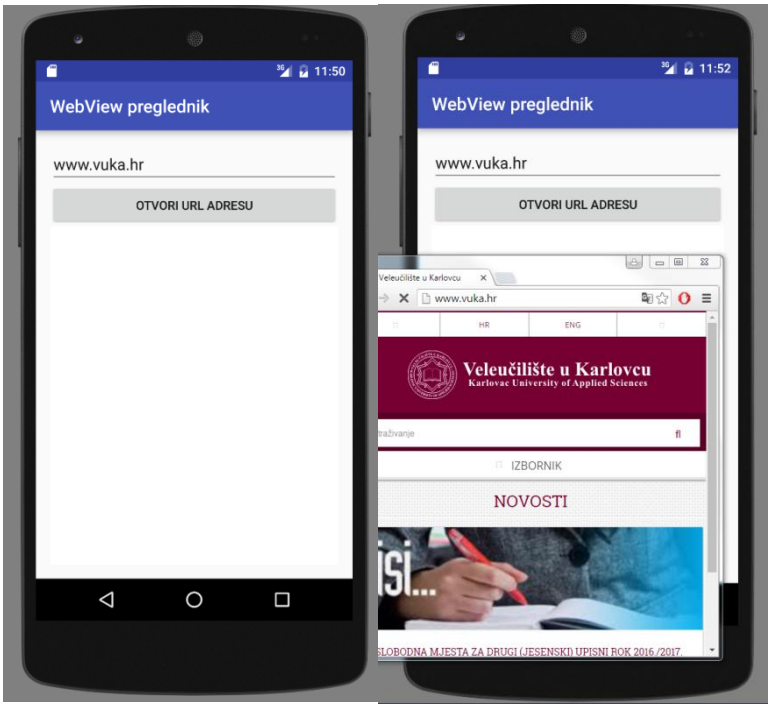
```
preglednik.getSettings().setLoadsImagesAutomatically(true);
preglednik.getSettings().setJavaScriptEnabled(true);
```

Ovom linijom koda je omogućeno prelistavanje web stranice unutar samog WebView prozora:

```
preglednik.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);
```

Preostalo je dati naredbu za otvaranje URL-a, koji mora biti varijabla tipa string (radi toga je vršena pretvorba teksta u string (*url.getText().toString()*), a ta naredba je *preglednik.loadUrl(s_url)*.

Napomena prilikom korištenja WebView komponente na emulatoru; zbog internet postavka za emulator, postavke po standardnim vrijednostima će otvarati URL u vanjskom pregledniku,kao i na slici.



Slika 3.38. Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 12: Geste

U ovoj vježbi je funkcija aplikacije da prepozna gestu koju korisnik izvodi u glavnoj aktivnosti te da ispiše koja je to gesta na zaslonu.

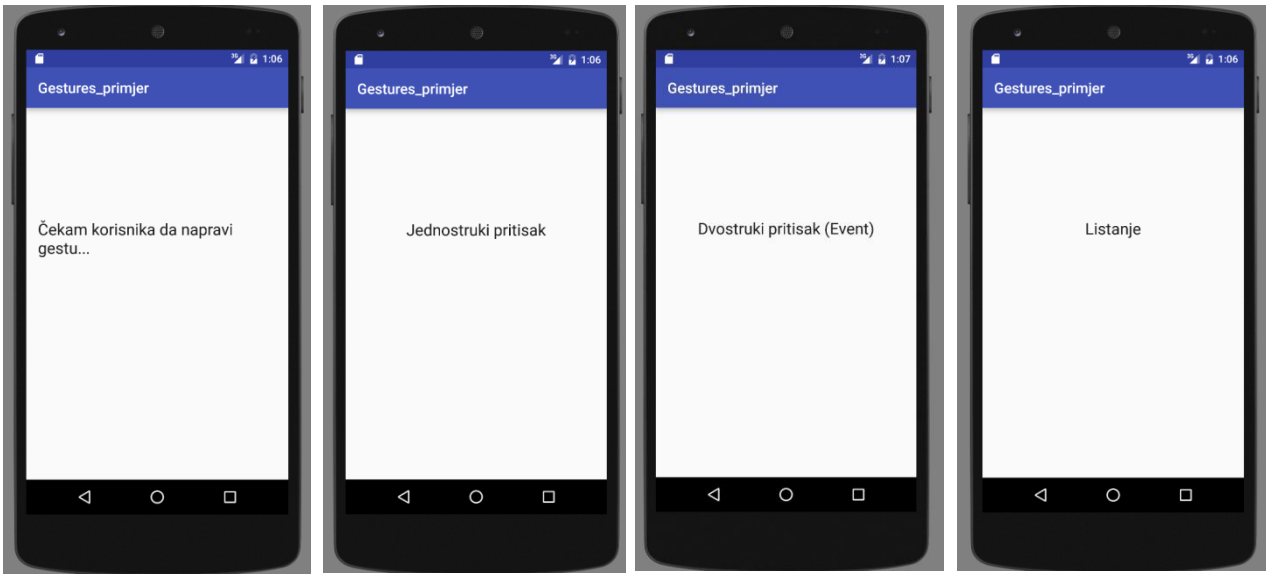
Od komponenti se koristi jedino tekstualni prozor za ispis naziva geste. Prepoznavanje gesti zahtijeva određene biblioteke:

Android.view.GestureDetector
Android.view.MotionEvent
Android.gesture.Gesture
*Android.view.GestureDetector.**
Kod:

```
package plasaj_vuka.gestures_primjer;

import android.support.v4.view.GestureDetectorCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

import android.view.GestureDetector;
import android.view.MotionEvent;
import android.gesture.Gesture;
import android.widget.TextView;

import static android.view.GestureDetector.*;


public class MainActivity extends AppCompatActivity implements
GestureDetector.OnGestureListener,GestureDetector.OnDoubleTapListener{

    private static TextView tview;
    private GestureDetectorCompat Prepoznaj_gestu;

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

        tview = (TextView)findViewById(R.id.textView);
        Prepoznaj_gestu = new GestureDetectorCompat(this,this);
        Prepoznaj_gestu.setOnDoubleTapListener(this);
    }
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        Prepoznaj_gestu.onTouchEvent(event);
        return super.onTouchEvent(event);
    }
    @Override
    public boolean onSingleTapConfirmed(MotionEvent e) {
        tview.setText(" Jednostruki pritisak ");
        return false;
    }
    @Override
    public boolean onDoubleTap(MotionEvent e) {
        tview.setText(" Dvostruki pritisak ");
        return false;
    }
```

```
    @Override
    public boolean onDoubleTapEvent(MotionEvent e) {
        tview.setText(" Dvostruki pritisak (Event) ");
        return false;
    }
    @Override
    public boolean onDown(MotionEvent e) {
        tview.setText(" Dolje ");
        return false;
    }

    @Override
    public void onShowPress(MotionEvent e) {
        tview.setText(" Prikaz ");
    }
   @Override
    public boolean onSingleTapUp(MotionEvent e) {
        tview.setText(" Jednostruko prema gore ");
        return false;
    }
   @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {
        tview.setText(" Listanje ");
        return false;
    }
    @Override
    public void onLongPress(MotionEvent e) {
        tview.setText(" Dugi pritisak ");
    }
    @Override
    public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {
        tview.setText(" Prelet ");
        return false;
    }
}
```

Logika koda:
Da bi glavna aktivnost podržavala metode vezane za geste, kod deklariranja glavne aktivnosti dopisali smo dodatne implementacije iz biblioteke *GestureDetector* koje su potrebne a to su: *OnGestureListener* i *OnDoubleTapListener*. Definirali smo varijablu *prepoznaj_gestu* kao *GestureDetector*. Ona se koristi u podfunkciji *onTouchEvent*, da bi prepoznala o kojoj se gesti radi. Zatim, u skupu podfunkcija, za svaku gestu posebno, govorimo tekstualnom prozoru što da ispiše ostvari li se ta gesta. Primjerice, podfunkcija *onDoubleTap(MotionEvent e)* prilikom dvostrukog pritiska na zaslon ispisuje poruku „*Dvostruki pritisak*“. Kompliciranije geste imaju svoje zasebne automatski generirane varijable koje definiraju tu gestu, primjerice kod scrollanja (listanja) se definira udaljenost scrollanja.

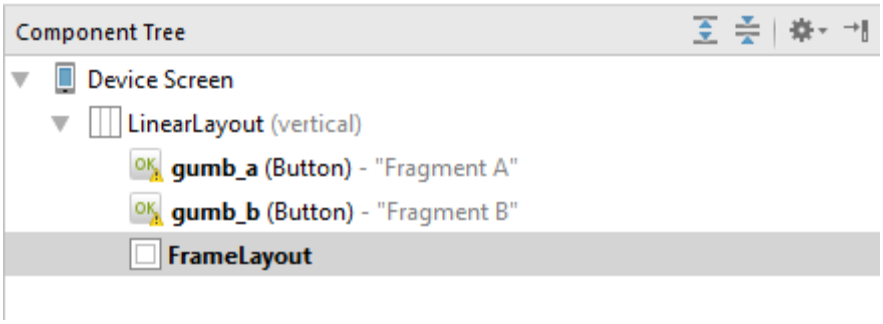


Slika 3.39. Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 13: Fragmenti u aktivnosti

U ovom primjeru zadatak je da pritiskom gumba korisnik mijenja fragmente na sučelju. Za to je potreban fragment manager, čija je funkcija da upravlja fragmentima. Također je potreban fragmentTransaction, on daje naredbe u koracima o izmjeni, dodavanju, i micanju fragmenata.

U glavnu aktivnost su dodani 2 gumba i FrameLayout. FrameLayout je u ovom slučaju okvir za fragment koji je aktivan.



Slika 3.40. Predodžba popisa komponenata

Activity_mail.xml datoteka izgleda ovako:

```
<?xml version="1.0" encoding="utf-8"?>
<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="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="plasaj_vuka.fragments_primjer.MainActivity"
    android:orientation="vertical">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment A"
        android:id="@+id/gumb_a"
        android:onClick="PromjenaFragmenata" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Fragment B"
        android:id="@+id/gumb_b"
        android:onClick="PromjenaFragmenata" />

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/FrameLayout"></FrameLayout>

</LinearLayout>
```

Evidentno je da su definirane sve komponente, te da je vrsta layouta promijenjena na LinearLayout.

Java kod fragmenta (isti je i za A i za B, osim naravno, naziva klase):

```
package plasaj_vuka.fragments_primjer;

import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;



public class fragmentB extends Fragment {


    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
            return inflater.inflate(R.layout.fragment_fragment_b,
container, false);
    }
}
```

Ovaj kod je bazičan, nema nikakve specifične funkcije jer se fragmenti sastoje samo od teksta koji navodi koji je fragment trenutno aktivan. U stvarnim primjenama će fragmenti imati više funkcija pa će samim time biti i više koda.

Java kod glavne aktivnosti:

```
package plasaj_vuka.fragments_primjer;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.app.Fragment;
import android.app.FragmentManager;
import android.app.FragmentTransaction;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void PromjenaFragmenata (View view) {
        Fragment frag;
        if( view==findViewById(R.id.gumb_a)) {
            frag = new fragmentA();
            FragmentManager fmanager = getFragmentManager();
            FragmentTransaction ftransaction = fmanager.beginTransaction();
            ftransaction.replace(R.id.FrameLayout, frag);
            ftransaction.commit();
        }
        if( view==findViewById(R.id.gumb_b)){
            frag= new fragmentB();
            FragmentManager fmanager = getFragmentManager();
            FragmentTransaction ftransaction = fmanager.beginTransaction();
            ftransaction.replace(R.id.FrameLayout, frag); //id iz .xml
            ftransaction.commit();
        }
    }
}
```

Logika koda:

Kod glavne aktivnosti ima zadaću da upravlja izmjenama fragmenata na znak pritiska odgovarajućeg gumba. Postoje dvije *if* petlje, jedna za svaki gumb i fragment. Ako je pritisnut gumb *A*, *fragmentTransaction* će započeti transakciju, gdje će zamijeniti ono što se nalazi u *FrameLayout* okviru sa onime što se nalazi u varijabli frag (koja je tip fragmenta), pa je u tu varijablu unesen fragment A (*frag=new fragmentA()*). Na kraju ide naredba *ftransaction.commit()* koja će izvršiti transakciju.

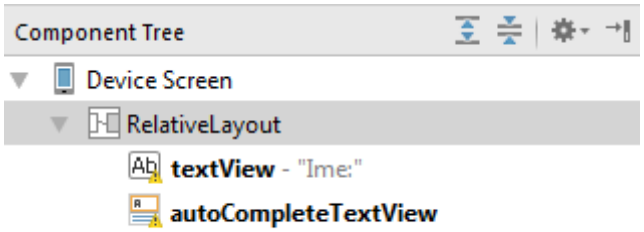
Kod kodiranja ove aplikacije vrlo je bitno paziti na biblioteke i na nazive fragmenata te na njihove lokacije.

Slika 3.41. Predodžba simulacije aplikacije u emulatoru na računalu. Prva slika je ono što se prikaže pritiskom na gumb A, a druga za gumb B. Tekst koji se pojavi je dio aktivnog fragmenta

Primjer 14: Automatska nadopuna teksta sa AutoCompleteTextView

Svrha ove aplikacije je da korisniku prilikom unošenja imena u tekstualni prozor dobije ponudu za automatskom nadopunom teksta. Za bazu podataka se koristi polje stringova u koje se unosi popis imena.

Komponenta u koju se upisuje tekst je autoCompleteTextView (slika 3.42).



Slika 3.42. Predodžba popisa komponenata.

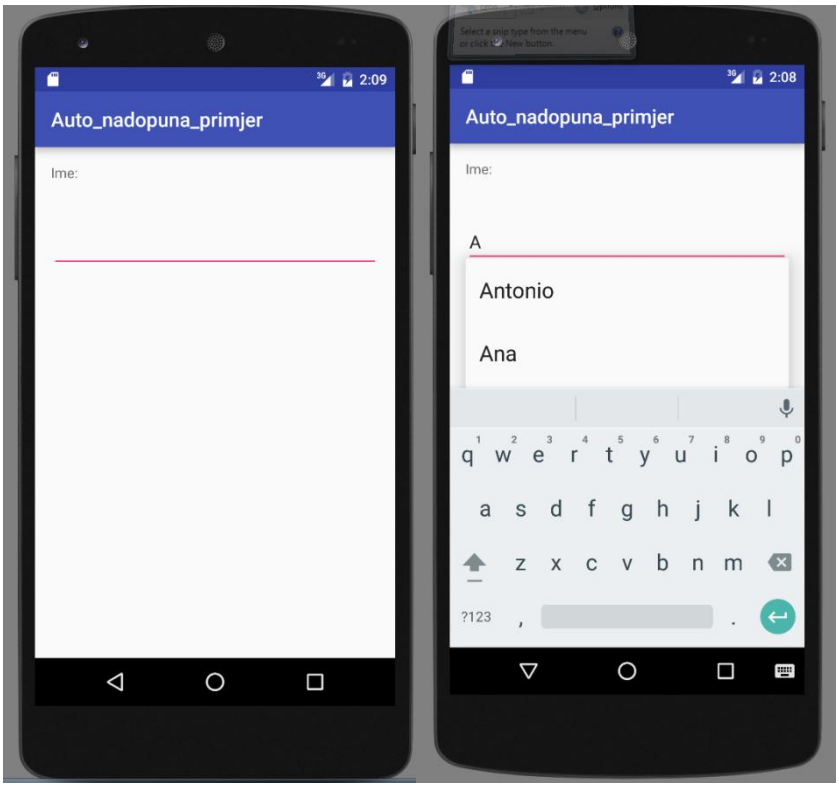
Kod:

```
package plasaj_vuka.auto_nadopuna_primjer;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;

public class MainActivity extends AppCompatActivity {
    AutoCompleteTextView autocomplete;
    String[] Imena = {"Antonio","Ana","Anamarija","Andrej"};
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        autocomplete =
(AutoCompleteTextView)findViewById(R.id.autoCompleteTextView);
        ArrayAdapter a1 = new
ArrayAdapter(this,android.R.layout.select_dialog_item,Imena);
        autocomplete.setThreshold(1); // na toliko znamenki se pojavljuje
preporuka
        autocomplete.setAdapter(a1);
    }
}
```

Logika koda :

Unutar polja varijabli tipa string je unesen proizvoljan broj imena. Ta imena će biti dostupna za automatsku nadopunu. Također je potreban *ArrayAdaper*, čija je namjena da omogući korištenje polja sa imenima za *AutoCompleteTextView* komponentu. *ArrayAdapter* prima 3 argumenta, od kojih je prvi kontekst (*this*), iduće je smisao korištenja polja (*select_dialog_item*), i zadnje je varijabla polja u kojem su spremljena imena. Sa *.setThreshold(n)* se određuje nakon koliko unesenih znakova će nam aplikacija ponuditi automatsku nadopunu teksta. Ovdje je odabrana ponuda dopune nakon jednog unesenog znaka. Za kraj se pokreće adapter *a1* za *AutoCompleteTextView* varijablu autocomplete.

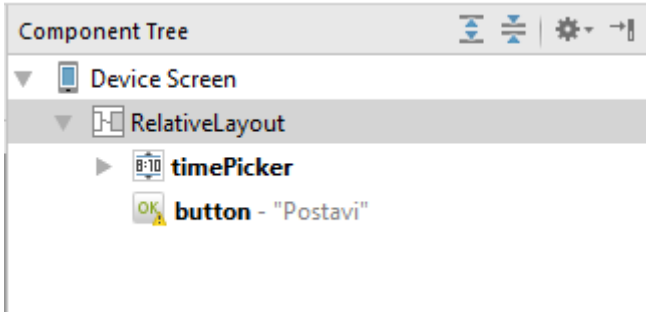
Sa simulacije (slika 3.43) je očito da korisnik nije dobio ponudu za nadopunu prije nego je unio minimalno jedan znak.



Slika 3.43. Predodžba simulacije aplikacije u emulatoru na računalu.

Primjer 15: TimePicker primjer

Zadatak ovog programa je da omogući korisniku namještanje vremena. To vrijeme se sprema u međuspremnik i ispisuje na zaslonu u obliku toast poruke.



Koristi se timePicker komponenta, te gumb koji daje odredbu za ispis vremena u toast poruci (slika 3.44).

Slika 3.44. Predodžba popisa komponenata

Kod:

```
package plasaj_vuka.timepicker;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.TimePicker;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    TimePicker birac;
    Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        prikaziVrijeme();
    }
    public void prikaziVrijeme() {
        birac = (TimePicker)findViewById(R.id.timePicker);
        btn= (Button)findViewById(R.id.button);
        birac.setIs24HourView(true);
        btn.setOnClickListener(
                new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {

Toast.makeText(getBaseContext(),birac.getCurrentHour() +" : " +
birac.getCurrentMinute(),Toast.LENGTH_SHORT).show();
                    }
                }
        );
    }
}
```

Logika koda:

Definira se TimePicker varijabla *birac*, i gumb *btn*. U funkciji *prikaziVrijeme()* se odabire 24-satni prikaz vremena naredbom *.setIs24hourView(true)*. Pri pritisku gumba ispisuje se toast poruka sa *.getCurrentMinute* i *.getCurrentHour* naredbama koje ispisuju vrijeme sa timePicker-a.

Kao što je evidentno na slici (slika 3.45), veoma je nepraktičan za primjenu u aplikacijama pa se koristi *TimePickerDialog* koji je praktičniji i kompaktniji.

Slika 3.45. Predodžba simulacije aplikacije u emulatoru na računalu

Primjer 16: TimePickerDialog

Radi uštede prostora, TimePicker se gotov uvijek prikazuje u prozoru koji se zatvara nakon što je obavljena interakcija za njime. Za to nam je potreban TimePickerDialog.

Kod:

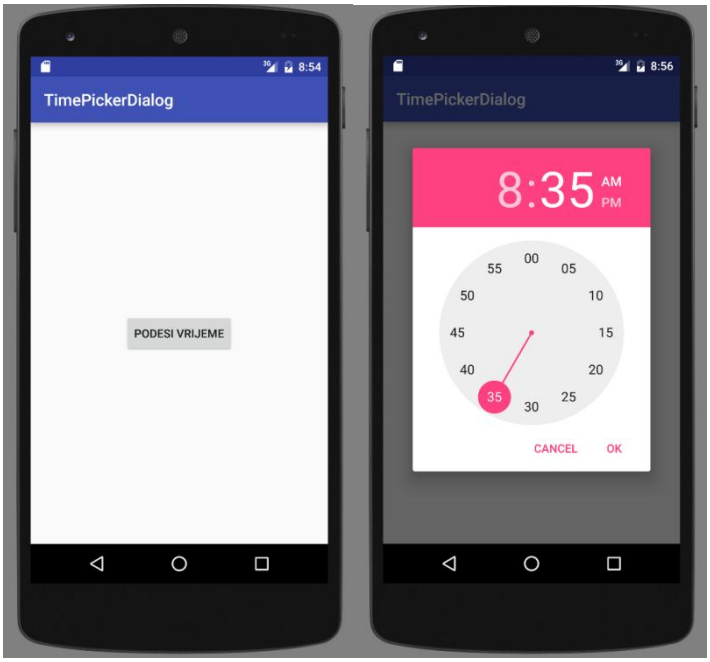
```
package plasaj_vuka.timepickerdialog;

import android.app.Dialog;
import android.app.TimePickerDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TimePicker;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    Button btn;
    static final int dialog = 0;
    int sat;
    int minuta;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        prikaziVrijemeDialog();
    }
    @Override
    protected Dialog onCreateDialog(int id) {
        if (id == dialog)
            return new TimePickerDialog(MainActivity.this,povrat_vrijeme,
sat,minuta,false );
        return null;
    }
    protected TimePickerDialog.OnTimeSetListener povrat_vrijeme = new
TimePickerDialog.OnTimeSetListener() {
        @Override
        public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
            sat=hourOfDay;
            minuta=minute;
            Toast.makeText(MainActivity.this,sat+ " + "
+minuta,Toast.LENGTH_SHORT).show();
        }
    };
    public void prikaziVrijemeDialog() {
        btn = (Button)findViewById(R.id.button);
        btn.setOnClickListener(
                new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        showDialog(dialog);
                    }
                }
        );
    }
}
```

Logika koda:
Definiraju se integeri *dialog*, *sat*, i *minuta*. Varijablu *dialog* se uspoređuje sa id-em funkcije Dialog. Varijabla ima početnu vrijednost nula. Ako su isti, gradi se novi TimePickerDialog, ako ne, funkcija vraća nulu. Kod izgrađivanja novog TimePickerDialoga, potrebno je unijeti 5 argumenata; prvi je kontekst MainActivity.this, drugi je naziv varijable time dialoga *povrat_vrijeme*, treći je varijabla za sat, četvrti je varijabla za minute, i peti je oblik prikaza vremena (*24h-TRUE ili PM/AM mod-FALSE*).

Prilikom postavljanja vremena, zadatak zahtijeva da korisnik dobije obavijest u obliku toast poruke o trenutnom vremenu. Za to se koristi metoda *TimePickerDialog.OnTimeSetListener*, koji čeka da se postavi vrijeme te onda ispisuje novo odabrano vrijeme. Ta metoda ima svoju podfunkciju *onTimeSet()* koja automatski generira lokalne varijable *hourOfDay* i *minute*. Te varijable se povezuju i poistovjećuju sa globalnim varijablama *sat*, i *minuta*.

Posljednja metoda koja je nužna je proizvoljno nazvana *prikaziVrijemeDialog()* metoda. U njoj je angažiran slušaoca na gumb, koji prilikom pritiska na gumb otvara dialog pomoću naredbe *showDialog(dialog)*, gdje je dialog varijabla koja je prethodno deklarirana.

Sa simulacije (slika 3.46) se vidi da je u glavnoj aktivnosti znatno ušteđeno na prostoru jer se TimePicker komponenta prikazuje jedino kada se želi promijeniti vrijeme, odnosno, prilikom pritiska na gumb „*Podesi vrijeme*“.



Slika 3.46. Predodžba simulacije aplikacije u emulatoru na računalu.

Primjer 17: DatePickerDialog

DatePickerDialog istom metodom kao i kod TimePickerDialoga, pritiskom na gumb otvara sučelje za namještanje datuma. Nakon što je novi datum namješten, aplikacija korisnika obavještava o novom datumu kroz toast poruku.

Kod:

```
package plasaj_vuka.datepickerdialog;

import android.app.DatePickerDialog;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.Toast;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity {
    Button btn;
    int godina;
    int mjesec;
    int dan;
    static final int dialog = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final Calendar datum = Calendar.getInstance();
        godina=datum.get(Calendar.YEAR);
        mjesec=datum.get(Calendar.MONTH);
        dan=datum.get(Calendar.DAY_OF_MONTH);
        prikaziDialog();
    }
    public void prikaziDialog() {
        btn = (Button)findViewById(R.id.button);
        btn.setOnClickListener(
                new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        showDialog(dialog);
                    }
                }
        );
    }
    @Override
    protected Dialog onCreateDialog(int id){
        if (id == dialog)
            return new DatePickerDialog(this, birac, godina,mjesec,dan);
        return null;
    }
    private DatePickerDialog.OnDateSetListener birac = new
DatePickerDialog.OnDateSetListener() {
        @Override
        public void onDateSet(DatePicker view, int year, int monthOfYear,
int dayOfMonth) {
```

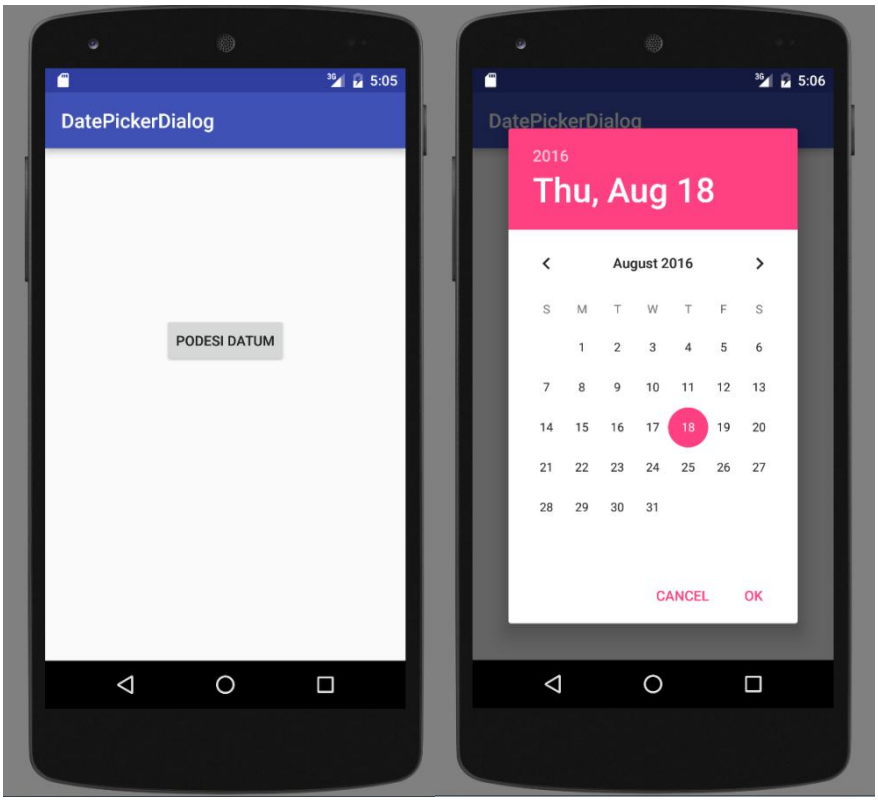
```
                godina=year;
                mjesec=monthOfYear +1;
                dan=dayOfMonth;
                Toast.makeText(MainActivity.this, +godina+ "/" +mjesec+
"/"+dan,Toast.LENGTH_SHORT).show();
            }
        };
}
```

Logika koda:

S obzirom da je kod gotovo identičan kodu u prošlom primjeru, nije potrebno pojašnjavati izradu dialog prozora. Razlika je ta što je zadan početni datum, tako da je u glavnoj aktivnosti deklarirana varijablu datum, u kojoj su pohranjeni podatci o trenutnom datumu sa uređaja.

```
  final Calendar datum = Calendar.getInstance();
                godina=datum.get(Calendar.YEAR);
                 mjesec=datum.get(Calendar.MONTH);
                 dan=datum.get(Calendar.DAY_OF_MONTH);
```

Također, napomena kod pohrane datuma u androidu; mjeseci se broje od nule. To znači da je siječanj označen kao nulti mjesec u godini, a ne prvi. Stoga je varijabli *mjesec*, i lokalnoj varijabli *monthOfYear* dodan +1.



Slika 3.47. Predodžba simulacije aplikacije u emulatoru na računalu.

Primjer 18: Notifikacije

Ovaj primjer implementira notification manager kako bi na android uređaju dao notifikaciju koju definira razvojni inženjer. Notifikacija se šalje prilikom pritisak na gumb. Gumb je jedina komponenta koja se koristi na sučelju.

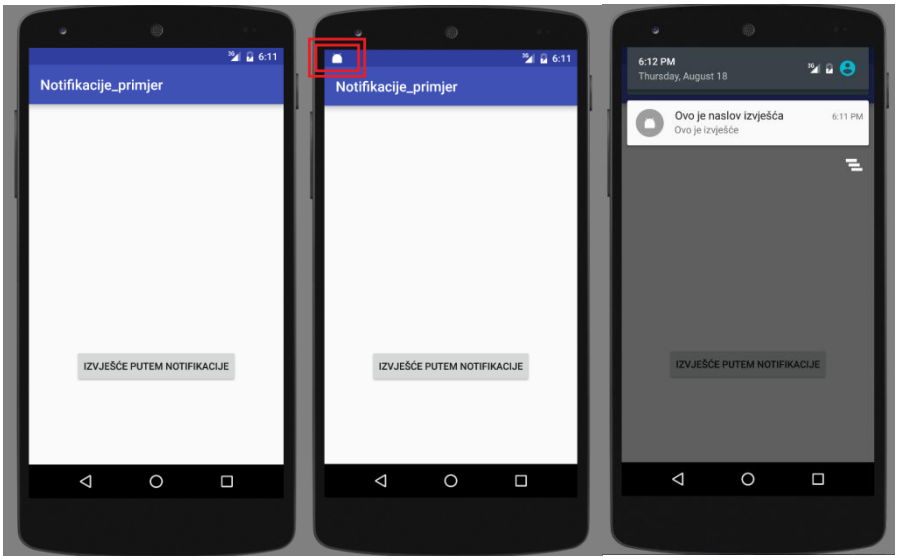
Kod:

```
package plasaj_vuka.notifikacije_primjer;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
    Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btn= (Button)findViewById(R.id.button);
        btn.setOnClickListener(
                new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        Intent intent = new Intent();y
                        PendingIntent pi =
PendingIntent.getActivity(MainActivity.this,0,intent,0;
Notification n = new Notification.Builder(MainActivity.this).setTicker("
Izvješće ").setContentTitle(" Ovo je naslov izvješća ")
                                    .setContentText(" Ovo je izvješće
").setSmallIcon(R.mipmap.ic_launcher).setContentIntent(pi).getNotification(
);
                        n.flags = Notification.FLAG_AUTO_CANCEL;
                        NotificationManager nm =
(NotificationManager)getSystemService(NOTIFICATION_SERVICE);
                        nm.notify(0,n);
                    }
                }
        );
    }
}
```

Logika koda:
Unutar *onClick* funkcije koja se pokreće prilikom pritiska na gumb se koristi *PendingIntent*. Pending intent je oznaka koja se može dati drugoj aplikaciji, naprimjer notification manageru, alarm manageru, ili nekoj drugoj aplikaciji koja ne mora biti službena android aplikacija (može biti 3rd party aplikacija). To dopušta drugim aplikacijama da koriste dozvole ove aplikacije i izvrše određeni kod. Najbolji primjer je davanje dozvole notification manageru da napravi notifikaciju koja je definirana u kodu. Definirana je varijabla tipa Notification i pomoću *.set()* funkcija su navedena sva njena svojstva, od naziva i teksta, do ikone. Moguće je i dodati akcije pomoću *.setAction()* ali u ovom primjeru nije bilo potrebno. Također se

koriste flagovi. Korišten je FLAG_AUTO_CANCEL, koji dozvoljava korisniku da ugasi notifikaciju pritiskom na nju.

Na slici 3.48 se vidi da pritiskom gumba korisnik dobije obavijest u traci za obavijesti,i prilikom povlačenja trake može vidjeti detalje obavijesti. Pritiskom na notifikaciju zbog dodanog flaga se obavijest gasi.



Slika 3.48. Predodžba simulacije aplikacije u emulatoru na računalu.

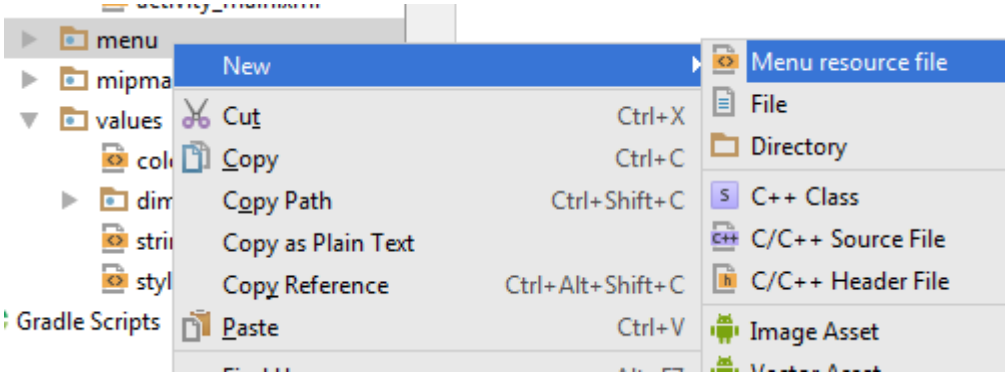
Primjer 19: ActionBar (1)

U ovom primjeru će zadatak biti u Action baru postaviti niz gumba koji će obavljati operacije. Gumbi će biti unutar overflow menu-a.

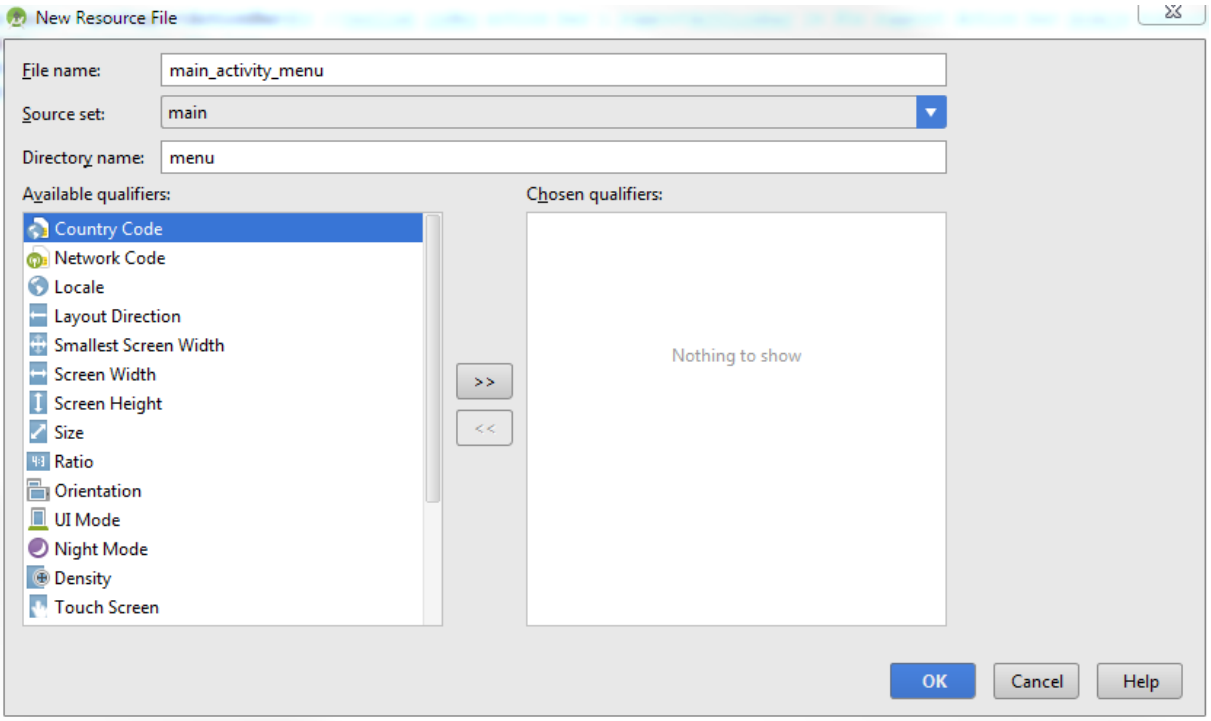
Za početak će ti gumbi ispisati toast poruke, a u sljedećem primjeru će ih biti potrebno povezati sa aktivnostima. Prvo treba izraditi menu unutar kojega će se XML kodom dizajnirati gumbi gumb1 i gumb2, a zatim isprogramirati.

Izrada menu-a.

Unutar drawables mape, nužno je imati mapu "*menu*", ukoliko je nema, potrebno ju je napraviti. Unutar nje se izrađuje Menu resource file koji je nazvan *main_activity_menu*. (slike 3.49 i 3.50). Nakon kreiranja menu resource datoteke, u XML kod menu-a se dodaju 2 gumba. Taj kod se može vidjeti u sljedećem koraku ovog primjera.



Slika 3.49. Predodžba izrade nove res datoteke.



Slika 3.50. Predodžba izrade nove res datoteke.

XML kod menu-a:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
xlmns:app="http://schemas.android.com/apk/res-auto">
<item
    android:id="@+id/gumb1_id"
    android:title="gumb1"
    android:icon="@drawable/ic_label_black_24dp"
    android:showAsAction="never"></item>
    <item
        android:id="@+id/gumb2_id"
        android:title="gumb2"
        android:icon="@drawable/ic_pets_black_24dp"
        android:showAsAction="never"></item>
</menu>
```

Logika XML koda:

Iz koda je lako zaključiti da su dodana 2 gumba, s imenima gumb1 i gumb2 i pripadajućim ID-evima. Pridodane su im i ikone, koje prikazuju gumbe kada ih se prepozna kao Action gumbe. To je podesivo tako da je *showAsAction="always"*, umjesto *"never"*. S obzirom da je potrebno imati overflow padajući izbornik, postavljeno je na *"never"* .

Da bi gumb prilikom pritiska izvršio funkciju, može mu se dati ime funkcije u *onClick* svojstvima komponente, ili u XML kodu glavne aktivnosti za komponentu gumba dodati sljedeću liniju koda: **android:onClick="Naziv_funkcije_koju_zelimo_pokrenuti"**

Java kod:

```
package plasaj_vuka.actionbar;

import android.app.ActionBar;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        android.support.v7.app.ActionBar abar = getSupportActionBar();
        abar.setLogo(R.drawable.ic_label_black_24dp;
        abar.setDisplayUseLogoEnabled(true);
        abar.setDisplayShowHomeEnabled(true);
    }

    @Override
    public boolean onCreateOptionsMenu (Menu menu) {
        MenuInflater minflater = getMenuInflater();
        minflater.inflate(R.menu.main_activity_menu, menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.gumb1_id:
                Toast.makeText(getApplicationContext(), " gumb 1 je
pritisnut ", Toast.LENGTH_SHORT).show();
            case R.id.gumb2_id:
                Toast.makeText(getApplicationContext(), " gumb 2 je
pritisnut ", Toast.LENGTH_SHORT).show();
             default:
             return super.onOptionsItemSelected(item);
         }
    }
}
```

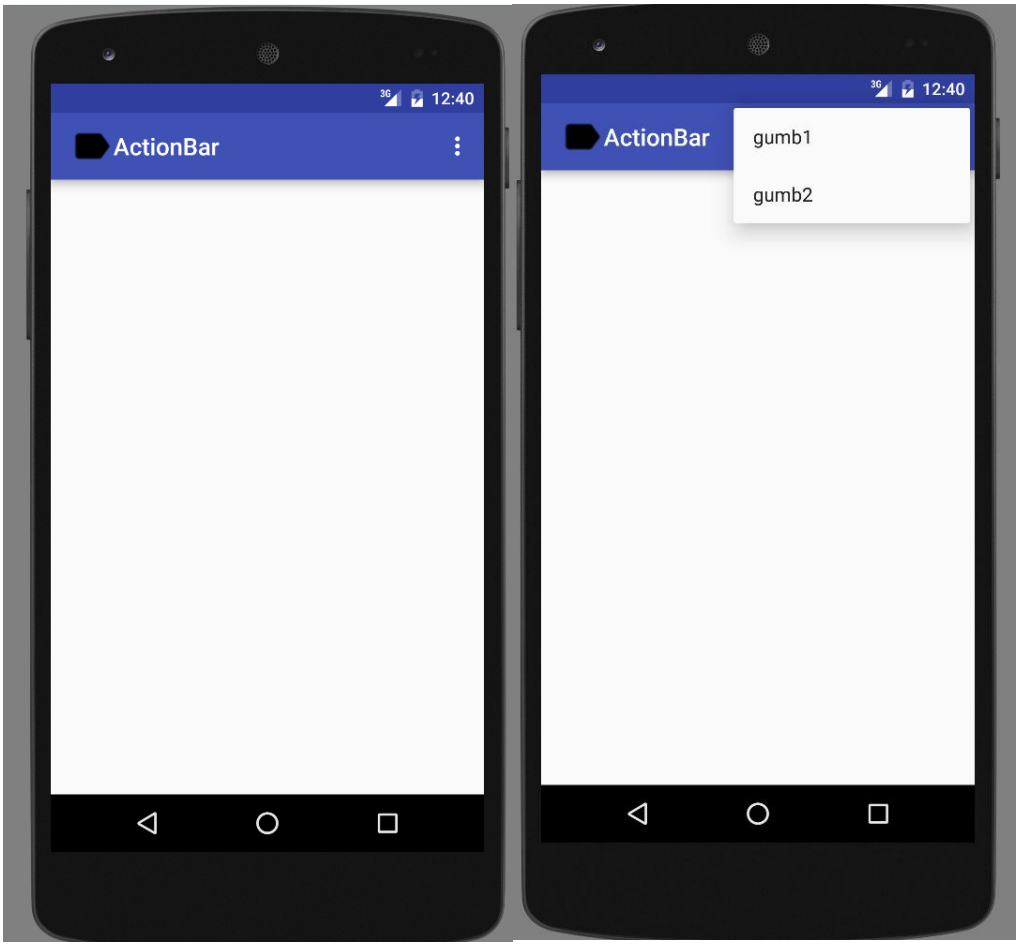
Logika koda:

Unutar *onCreate* funkcije se može vidjeti da je action bar definiran sa *getSupportActionBar()*. Razlog što je korišten *getSupportActionBar*, a ne *getActionBar* je taj što sa njime dolazi i niz biblioteka potrebnih za action bar implementaciju kod određenih API razina. Također je postavljen logo (sa *.setLogo*), omogućeno je action baru da prikazuje logo *(.setDisplayUseLogoEnables(True)*), te omogućeno prikazivanje home gumba.

Koristi se i funkcija *onCreateOptionsMenu*, koja koristi varijablu *menu* da bi razvila izbornik koristeći *Menuinflater.inflate(X,Y)* , gdje je X adresa kreiranog menua iz prethodnog koraka, a Y naziv varijable menu-a, koja je deklarirana upravo kao "*menu*".

Sljedeća funkcija je *onOptionsItemSelected*. Ona ima zadaću da obavlja funkcije koje se trebaju izvršiti prilikom pritiska na jedan od dva gumba. Ta funkcija će biti ispisivanje

toast poruke. Implementira se switch (*item.getItemId()*), jer je praktičnije da se sa jednom funkcijom izvršava kontrola nad oba gumba. Stoga, switch/case metoda je korisna za ovaj primjer. U primjeru su 2 case-a (slučaja), jedan je za gumb 1, i drugi za gumb 2.



Slika 3.51. Predodžba emulacije aplikacije: pritiskom na overflow menu ikonu se otvara padajući izbornik.

Primjer 20: ActionBar (2)

U ovome primjeru će se na action bar staviti gumb koji će korisnika vraćati natrag u glavnu aktivnost nakon što je napusti. Da bi se to ostvarilo, potrebno je imati dodatnu aktivnost. Dodatne aktivnosti su obrađene u prethodnim primjerima.

U prvoj aktivnosti je potreban gumb prilikom čijeg pritiska će korisnik otići u drugu aktivnost. Druga aktivnost ne treba imati taj gumb jer će se vraćati u glavnu aktivnost pomoću action bara. Da bi sustav znao kuda nas vratiti, druga aktivnost mora imati određenu "*parent*" aktivnost. U prijevodu, roditeljska aktivnost iz koje je druga aktivnost kreirana. Parent aktivnost se namješta u AndroidManifest.xml datoteci.

AndroidManifest.XML kod:

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

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

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <activity
            android:name=".prva"
            android:parentActivityName=".MainActivity">
            <meta-data android:name="android.support.PARENT_ACTIVITY"
                android:value=".MainActivity"></meta-data>
        </activity>

    </application>

</manifest>
```

Naznačene su sljedeće linije koda :

```
            <meta-data        android:name="android.support.PARENT_ACTIVITY"
                android:value=".MainActivity"></meta-data>
```

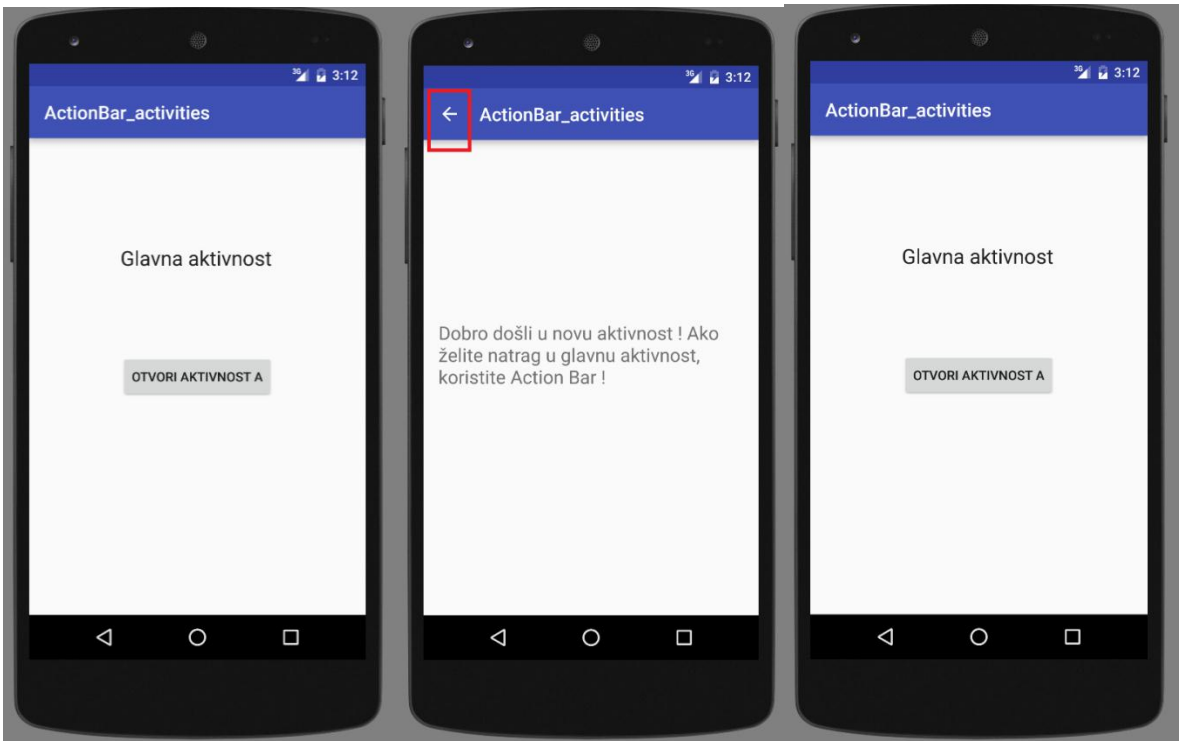
To je dio koda u kojem je namještena parent aktivnost.

Java kod glavne aktivnosti:

```
package plasaj_vuka.actionbar_activities;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getSupportActionBar().setDefaultDisplayHomeAsUpEnabled(true);
    }
    public void otvoriA (View view) {
        startActivity(new Intent(this,prva.class));
    }
}
```

Logika koda:

.setDefaultDisplayHomeAsUpEnabled(true) omogućuje da se koristi home gumb koji će korisnika vraćati u glavnu aktivnost. Druga funkcija *otvoriA(View view)* koja uzima argument *view*, pokreće novu aktivnost koristeći intent. Intent je također obrađen u prethodnim primjerima. Kod druge aktivnosti nije bitan jer druga aktivnost ne obavlja nikakvu funkciju.



Slika 3.52. Predodžba simulacije aplikacije u emulatoru na računalu. Pritiskom na gumb „Otvori aktivnost A“ korisnik dolazi u novu aktivnost, iz koje se vraća u glavnu aktivnost pritiskom strelice na action baru.

Primjer 21: Vježba sa implicitnim i eksplicitnim intentima

Svrha ovog primjera je da pokaže razliku između implicitnog i eksplicitnog intenta. Za početak, izraditi će se aplikacija koja koristi eksplicitni intent.

Eksplicitni intent – primjer

Ova aplikacija će pritiskom na gumb otvoriti novu aktivnost.

Java kod glavne aktivnosti:

```
package plasaj_vuka.intent_v2;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_main);
    }
    public void otvoriAktivnost(View view) {
        Intent i1 = new Intent(this,aktivnost.class);
        startActivity(i1);
    }
}
```

u XML kod glavne aktivnosti pod kategorijom gumba se dodaje linija koda :

```
android:onClick="otvoriAktivnost"
```

Slika 3.53. Predodžba simulacije aplikacije u emulatoru na računalu.

Sada kada je primjer aplikacije koja koristi eksplicitni intent spreman, može ga se koristiti za primjer implicitnog intenta.

Implicitni intent – primjer

Ova aplikacija će pritiskom na gumb otvoriti drugu aktivnost iz prethodnog primjera. Da bi to bilo moguće, koristi se prethodni primjer i u AndroidManifest.xml se dodaje Intent filter za drugu aktivnost. Dio koda za drugu aktivnost u AndroidManifest.xml prethodne aplikacije će izgledati ovako:

```
</activity>
<activity android:name=".aktivnost"></activity>
<intent-filter>
    <action android:name="plasaj_vuka.intent_v2.aktivnost" />

    <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

Kada je taj dio koda unesen, ta aktivnost je postala vidljiva svim aplikacijama na uređaju.

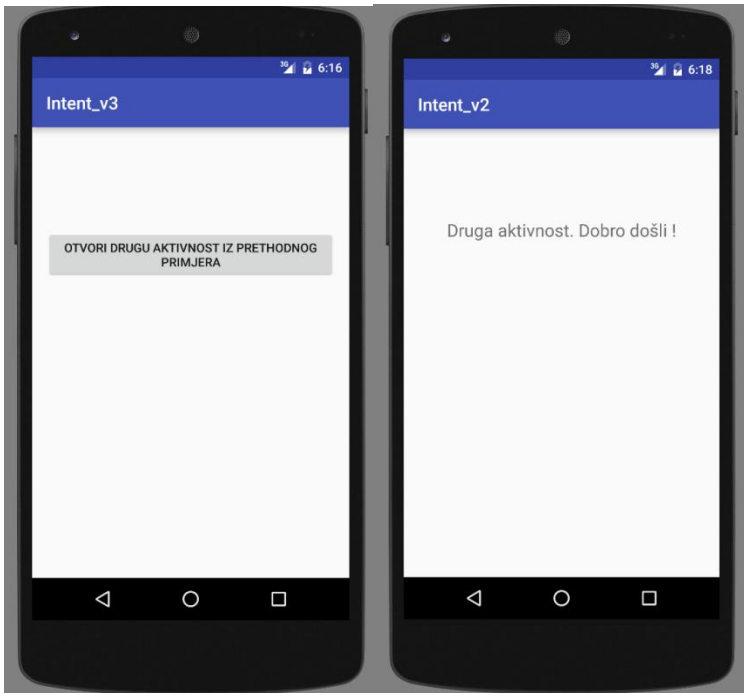
MainActivity.xml kod :

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="plasaj_vuka.intent_v3.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Otvori drugu aktivnost iz prethodnog primjera"
        android:id="@+id/button"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="82dp"
        android:onClick="otvoriDruguAktivnostIzPrveAplikacije"/>

</RelativeLayout>
```

Java kod :

```
package plasaj_vuka.intent_v3;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void otvoriDruguAktivnostIzPrveAplikacije (View view) {
       Intent i2 = new Intent("plasaj_vuka.intent_v2.aktivnost");
        startActivity(i2);
    }
}
```

Kod java koda glavne aktivnosti, za intent *i2* nije naveden naziv aktivnosti, već mu je data lokacija aktivnosti na uređaju. Upravo time je ostvarena implicitnost intenta.



Slika 3.54. Predodžba simulacije aplikacije u emulatoru na računalu.

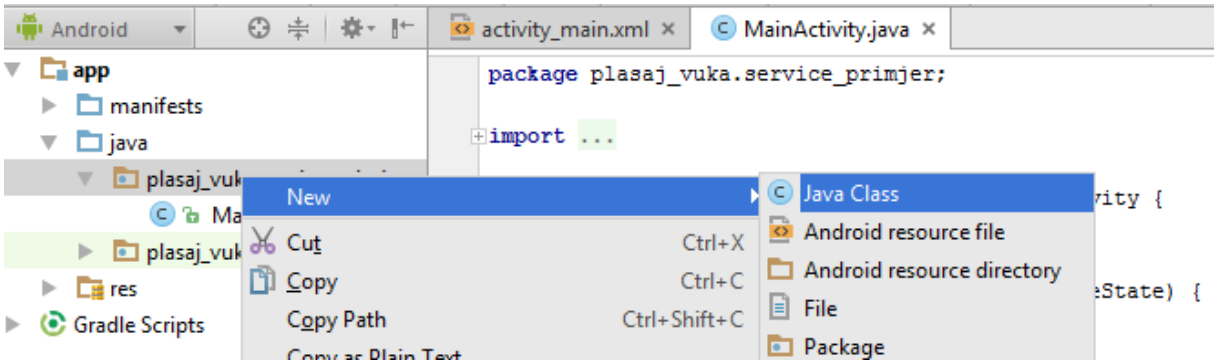
Primjer 22: Service, Kreiranje service-a

Svrha aplikacije je da unutar glavne aktivnosti korisnik pali i gasi service, te da ga o tome aplikacija obavještava u obliku toast poruke. Service će ostati aktivan i nakon gašenja aplikacije, sve dok korisnik ne stisne gumb „Gasi service“.

Nakon dodavanja gumba na sučelje, unutar activity_main.xml se dodaju dvije linije koda:

```
android:onClick="startService"        se dodaje unutar <Button1>
android:onClick="stopService"         se dodaje unutar <Button2>
```

Time se povezuju funkcije *startService* i *stopService* sa odgovarajućim gumbima.
Kreira se nova .java class datoteku koja će sadržavati kod service-a. Ta datoteka je proizvoljno nazvana *Service*_javaclass.java (slika 3.53).



Slika 3.55. Predodžba izrade nove Java klase

Java kod:

```
package plasaj_vuka.service_primjer;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.support.annotation.Nullable;
import android.widget.Toast;

public class Service_javaclass extends Service {
    @Override
    public void onCreate() {
        super.onCreate();
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Toast.makeText(Service_javaclass.this," Service pokrenut
",Toast.LENGTH_LONG).show();
        return START_STICKY;
    }
    @Override
    public void onDestroy() {
        Toast.makeText(Service_javaclass.this," Service ugašen
",Toast.LENGTH_LONG).show();
    }
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

Sa funkcijom *onStartCommand* se pokreće service. Funkcija vraća flag START_STICKY. Time se daje na znanje sustavu da će ovaj service imati trajanje od određenog perioda vremena. *onDestroy* funkcija gasi service. To su sve automatski generirane funkcije sa vlastitim parametrima koje nije potrebno definirati.

Java kod glavne aktivnosti:

```
package plasaj_vuka.service_primjer;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {
   @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void startService(View view) {
        Intent i1 = new Intent(this,Service_javaclass.class);
        startService(i1);
    }
    public void stopService(View view) {
        Intent i1 = new Intent(this,Service_javaclass.class);
        stopService(i1);
```

```
    }
}
```

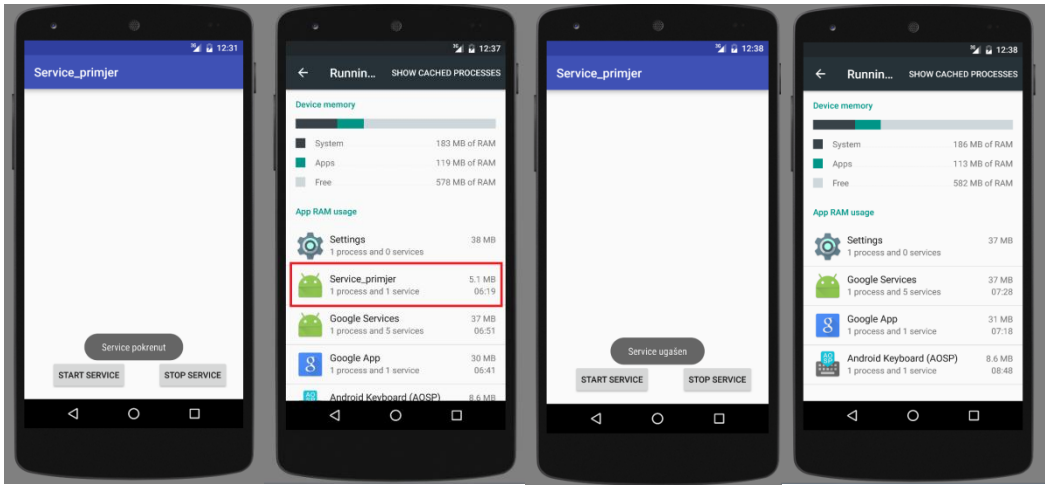
Koristeći intent se pokreće service. Ovdje su navedene dvije podfunkcije koje su prethodno navedene u .xml datoteci kao *onClick* funkcije; to su *startService()* i *stopService().*

U AndroidManifest.xml se dodaje sljedeći kod:

```
<service android:name=".Service_javaclass"
        android:exported="false"></service>
```

Prva linija koda definira service, a druga linija odlučuje hoće li druge aplikacije na uređaju moći koristiti ovaj service. Odabirom parametra *„false“* se zabranjuje drugim aplikacijama korištenje service-a.

Da se proces service-a odvija neovisno o aplikaciji, je evidentno u simulaciji aplikacije (slika 3.54). Prvo je pokrenuta aplikacija, i pritiskom na gumb pokrenut servis. Nakon toga je aplikacija potpuno ugašena, ali u procesima se vidi da se proces i dalje odvija. Nakon ponovnog ulaska u aplikaciju i gašenja service-a preko gumba, se vidi da se je proces ugasio.



Slika 3.56. Predodžba simulacije aplikacije u emulatoru na računalu.

Primjer 23: Service,kreiranje zasebnog threada

Da bi aplikacija bila čim stabilnija, za service se mogu koristiti zasebni threadovi (niti) koji koriste resurse CPU-a (Central Processor Unit), tako da ne uzimaju memoriju i resurse od aplikacije.

Kod unutar Service java class datoteke koji je potrebno dodati slijedi:
Kreira se nova funkciju *thread_class* koja implementira svojstva Runnable funkcije. U njoj je definiran integer varijabla *service_id* koja će sadržavati id od servicea za koji je potreban thread.

```
final class thread_class implements Runnable {
    int service_id;
    thread_class(int service_id){
        this.service_id = service_id;
    }
```

Funkcija *run()* obavlja funkciju koju ima service, s time da je dodana naredba *stopSelf(service_id)*, koja gasi service čiji je id jednak vrijednosti varijable *service_id*.

```
    @Override
    public void run() {
        Toast.makeText(Service_javaclass.this," Service pokrenut
    ",Toast.LENGTH_LONG).show();
        stopSelf(service_id);
    }
```

Unutar *onStartCommand* funkcije se dodaju sljedeće dvije linije koda:

```
public int onStartCommand(Intent intent, int flags, int startId) {
...
...
Thread thread = new Thread(new thread_class(startId));
thread.start();
...
...
}
```

startID je automatski generirana varijabla *onStartCommand* funkcije.

Primjer 24: IntentService

Ovaj primjer će kreirati service za zasebnim threadom, koristeći IntentService. Service se pokreće pritiskom na gumb. Ovakav način kreiranja service-a je dobar za aplikacije koje imaju više service-a.

Kod u Service_javaClass.java datoteci:

```
package plasaj_vuka.service_intentservice;

import android.app.IntentService;
import android.content.Intent;
import android.widget.Toast;

public class Service_javaClass extends IntentService{

    public Service_javaClass() {
        super(" IntentThread_name ");
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Toast.makeText(Service_javaClass.this," Service pokrenut
",Toast.LENGTH_LONG).show();
        return super.onStartCommand(intent, flags, startId);
    }

    @Override
    public void onDestroy() {
        Toast.makeText(Service_javaClass.this," Service zaustavljen
",Toast.LENGTH_LONG).show();
        super.onDestroy();
    }
```

```
    @Override
    protected void onHandleIntent(Intent intent) {
        Toast.makeText(Service_javaClass.this," onHandleIntent aktivan
",Toast.LENGTH_LONG).show();
    }
}
```

Za funkciju *Service_javaClass* su dodana svojstva funkcije *IntentService*. Unutar funkcije je definiran naziv threada, koji se naziva worker thread (*IntentThread_name*). Dalje se nalaze *onStartCommand* i *onDestroy* funkcije koje su prethodno spomenute.

Kod u MainActivity.xml

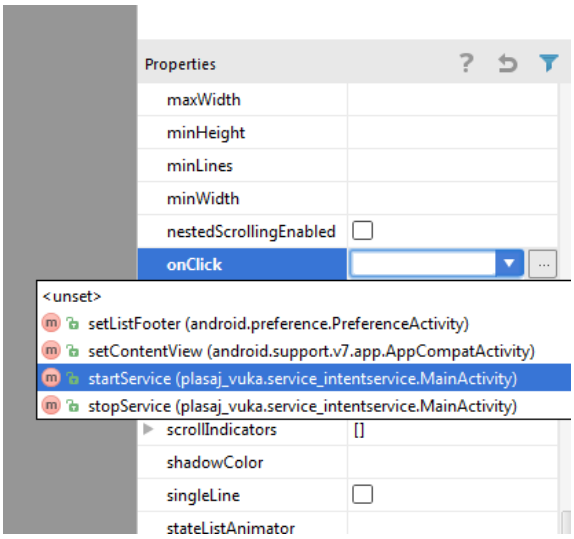
```
package plasaj_vuka.service_intentservice;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void startService(View view){
        Intent intent=new Intent(this,Service_javaClass.class);
        startService(intent);
    }
    public void stopService(View view) {
        Intent intent=new Intent(this,Service_javaClass.class);
        stopService(intent);
    }
}
```

Funkcije *startService* i *stopService* implementiraju intent za davanje naredbi za pokretanje i zaustavljanje service-a. Ostatak koda bi trebao biti poznat.

Preostalo je u AndroidManifest.xml datoteci definirati service i isključiti mogućnost dijeljenja service-a sa drugim aplikacijama:

```
<service android:name=".Service_javaClass"
    android:exported="false"></service>
```

Uz to, potrebno je dodati onClick svojstvo gumbima. U prošlih par primjera je korišten .XML kod za ovu svrhu, ali u ovom primjeru će se koristiti svojstva gumba u dizajnerskim postavkama (slika 3.53).



Slika 3.57. Predodžba dodavanja funkcije koja će se izvršiti prilikom pritiska na gumb (onClick)

Primjer 25: Service, kreiranje bound (vezanog) service-a

Kao što je pojašnjeno u teoretskom dijelu rada, bound service služi kao server-klijent veza, ili kao interprocesni komunikator. On se veže na neku komponentu aplikacije, naprimjer aktivnost.

Cilj ovog primjera je da primijeni bound service na aplikaciju koja će pritiskom na gumb dati jedan nasumičan broj od 0 do 100 na sučelju glavne aktivnosti. Kao i kod prijašnjih service primjera, prvo je potrebno kreirati class.java datoteku koja će sadržavati kod service-a.

Kod MyService.java datoteke:

```
package plasaj_vuka.service_bound_primjer;

import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;

import java.util.Random;

public class MyService extends Service {
    private final IBinder ib =new LocalBinder();
    private final Random gen = new Random();
    public class LocalBinder extends Binder {
        MyService getService () {
            return MyService.this;
        }
    }
    public MyService() {
    }

    @Override
    public IBinder onBind(Intent intent) {
        return ib;
    }
    public int getRandom() {
        return gen.nextInt(100);
```

```
    }
}
```

Logika koda:
U funkciju *MyService* su implementirana svojstva funkcije Service (pomoću extend Service). Također se koristi *onBind* funkcija koja je nužna za bound service, jer ona mora vraćati (return) sučelje bindera koje je definirano. U funkciji *MyService* se nalazi podfunkcija *LocalBinder* koja ima svojstva funkcije *Binder*. Unutar nje je potrebno kreirati metodu koja vraća *MyService (MyService.this*) i binder tip varijable Interface Binder. To znači da je on zadužen za definiranje sučelja bindera. Inicijalizira se sa operacijom new *LocalBinder()*. Vidljivo je da *onBind* funkcija vraća varijablu *ib*, koja je *Ibinder* varijabla. Time je ostvaren uvjet kreiranja bound service-a.

Funkcija service-a je da generira nasumični broj od 0 do 100. Za to se definira varijabla tipa Random i naziva se *gen*. U onBind funkciji se nalazi podfunkcija *getRandom()* koja vraća nasumični integer vrijednosti od 1 do 100 (argument je maksimalni broj, u ovom slučaju 100).

Unutar sučelja glavne aktivnosti, postavljen je gumb za pokretanje service-a i textView za prikazivanje rezultata. Gumb mora imati podešeno *onClick* svojstvo kao i u svim prethodnim primjerima.

```
android:onClick="Nasumicni_broj"
```
(Nasumični broj je naziv funkcije)

Kod glavne funkcije:

```
package plasaj_vuka.service_bound_primjer;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.IBinder;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    MyService myService;
    boolean isBound = false;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Intent intent = new  Intent(this,MyService.class);
        bindService(intent,serviceConnection, Context.BIND_AUTO_CREATE);
    }
    public void Nasumicni_broj (View view) {
        TextView textView = (TextView)findViewById(R.id.textView);
        textView.setText(Integer.toString(myService.getRandom()));
    }

    private ServiceConnection serviceConnection= new ServiceConnection() {
```

```
        @Override
        public void onServiceConnected(ComponentName name, IBinder service)
{
            MyService.LocalBinder binder = (MyService.LocalBinder) service;
            myService=binder.getService();
            isBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            isBound = false;
        }
    };
}
```

Logika koda:

Kreirana je funkcija *Nasumicni_broj(View view)*, koja će kao rezultat vraćati nasumični broj. Unutar MainActivity klase, definirana je *myService* varijabla, i *isBound* varijabla. *isBound* varijabla će djelovati kao flag, odnosno, davati će obavijest da li je service vezan ili nije (true/false). Na početku nije vezana, pa je stavljena početna vrijednost (false). Prilikom definiranja varijable *serviceConnection*, koja je tip varijable *ServiceConnection*, automatski će se izgraditi dvije funkcije. Te funkcije su *onServiceConnected()* i *onServiceDisconnected()*. Unutar *onServiceConnected()* funkcije je definiran *LocalBinder* naziva binder, i vezan je (castan) na MyService. Vrijednost varijable *isBound* je postavljena na true, jer je sada servis povezan. Unutar *onServiceDisconnected* se ta vrijednost mijenja natrag na false.

Unutar *onCreate* funkcije je kreiran objekt *intent* (tipa Intent) i inicijaliziran kao Intent (*new Intent(this,MyService.class*). Naredba *bindService* koristi sljedeće argumente:

1.) Intent koji je kreiran (u ovom slučaju njegovo ime je intent)
2.) ServiceConnection instanca (u ovom slučaju serviceConnection)
3.) Context.BIND_AUTO_CREATE (flag)

Preostala je funkcija *Nasumicni*_broj koja zapisuje vrijednost varijable *getRandom*() u tekstualni prozor (*textView*) koji je kreiran u sučelju glavne aktivnosti. Potrebno je promijeniti tip podatka iz integer u string.

Slika 3.58. Predodžba simulacije aplikacije u emulatoru na računalu.

Primjer 26: Pohrana .txt datoteke

Ovaj primjer će imati funkciju kreiranja datoteke formata .txt i pohrane teksta koji korisnik unese u tu datoteku. Pritiskom na gumb „*Pohrana*“ tekst će se pohraniti, a pritiskom na gumb „*Ispis*“ će se ispisati na sučelju glavne aktivnosti aplikacije.

OnClick svojstva oba gumba moraju biti namještena na odgovarajuću funkciju, kao i u prethodnim primjerima.

Kod MainActivity.java datoteke:

```
package plasaj_vuka.storage_pohrana;

import android.preference.TwoStatePreference;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;

public class MainActivity extends AppCompatActivity {
    EditText et;
    TextView tw;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et = (EditText) findViewById(R.id.editText);
        tw = (TextView) findViewById(R.id.textView);
    }

    public void pohrani(View view) {
        String poruka = et.getText().toString();
        try {
            FileOutputStream fileOutputStream =
openFileOutput("dokument.txt", MODE_PRIVATE);
            fileOutputStream.write(poruka.getBytes());
            fileOutputStream.close();
            Toast.makeText(getApplicationContext(), " Pohranjeno ",
Toast.LENGTH_LONG).show();
            et.setText("");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void ispisi(View view) {
        try {
```

```
            FileInputStream fileInputStream =
openFileInput("dokument.txt");
            InputStreamReader inputStreamReader = new
InputStreamReader(fileInputStream);
            BufferedReader bufferedReader = new
BufferedReader(inputStreamReader);
            StringBuffer stringBuffer = new StringBuffer();
            String s1;
            while ((s1=bufferedReader.readLine())!=null) {
            stringBuffer.append(s1+ "\n");}
            tw.setText(stringBuffer.toString());}
            catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

Logika koda:

Varijable *EditText* i *TextView* su definirane i povezane sa njihovim odgovarajućim komponentama u sučelju aktivnosti. Kreirane su dvije funkcije, *ispisi*() i *pohrani*(), koje su povezane sa gumbima koristeći onClick (unutar .xml datoteke). U funkciji pohrani(), se koristi *.getText().toString()* da bi se pohranile upisane podatke u varijablu poruka. Deklariran je fileOutputStream koji je potreban za pohranjivanje podataka. On otvara prijenos podataka (stream). *openFileOutput* uzima varijablu naziva datoteke, te način rada. Način rada je odabran kao privatan (MODE_PRIVATE), odnosno, samo ova aplikacija može koristiti pohranjene podatke.

Try/catch metoda je nužna, jer ukoliko neće biti moguće pohraniti podatak, aplikacija mora znati što da učini. Te dvije podfunkcije se automatski generiraju ukoliko su potrebne. *fileOutputStream.write* pohranjuje podatke, a parametri su naziv varijable koja sadrži podatke, i tip podatka koji se prenosi (*getBytes* prenosi bitove). Nakon toga se zatvara stream pohrane sa *.close*(). Dodana je i toast poruka koja potvrđuje uspješnu pohranu.

Za čitanje podataka iz pohranjene datoteke je postupak sličan. Koristi se *FileInputStream* za otvaranje stream-a prijenosa podataka, ali u drugome smjeru nego kod zapisa podataka. *InputStreamReader* je čitač podataka sa streama. S obzirom da je cilj da se zapisuje niz podataka, red po red, koristiti će se *BufferedReader* i *StringBuffer*. While petlja dozvoljava da se ispisuju linije koda, jedna po jedna, sve dok vrijednost linije nije 0 (!= null znači, različita vrijednost od ničega, odnosno, ako je bilo što u liniji koda, ispiši je). Try/catch metoda je i ovdje nužna. Na kraju, ispisuje se poruka iz StringBuffera u textView u obliku stringa.

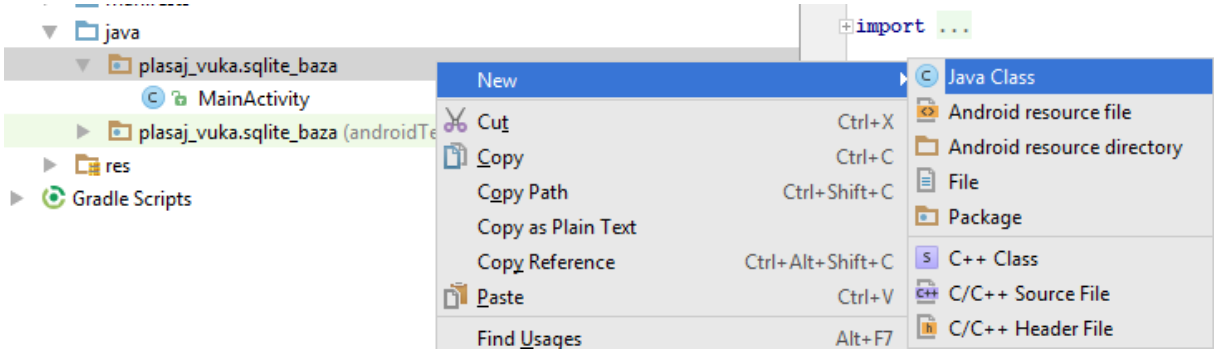
Slika 3.59. Predodžba simulacije aplikacije u emulatoru na računalu. Prva slika prikazuje zaslon prilikom pritiska gumba „POHRANA“, a druga slika prikazuje zaslon nakon pritiska gumba „ISPIS“. Pohranjen tekst je ispisan.

Primjer 27: SQLite baze podataka

Ovaj primjer će pojasniti kako kreirati bazu podataka, i umetati podatke u bazu podataka..

Kreiranje baze podataka

Baza podataka se kreira koristeći SQLhelper, stoga prije početka, potrebno je kreirati novu .java datoteku koju nazivamo DatabaseHelper (slika 3.60).



Slika 3.60. Predodžba kreiranja Java klase.

Kod DataBaseHelper.java datoteke:

```
package plasaj_vuka.sqlite_baza;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DataBaseHelper extends SQLiteOpenHelper {
    public static final String BazaPodataka_naziv = "plemeniti_plinovi.db";
    public static final String Tablica_naziv =
```

```
"plemeniti_plinovi_tablica.db";
    public static final String Stupac1 = "ID";
    public static final String Stupac2 = "SustavElemenata";
    public static final String Stupac3 = "Ime";
    public static final String Stupac4 = "Simbol";

    public DataBaseHelper(Context context) {
        super(context, BazaPodataka_naziv, null, 1);
        SQLiteDatabase db = this.getWritableDatabase(); // kreirati će bazu
podataka i tablicu, to koristimo samo sada za provjeru, kasnije ćemo micati
ovu liniju koda
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("create table " + Tablica_naziv +" (ID INTEGER PRIMARY
KEY AUTOINCREMENT,SustavElemenata TEXT,Ime TEXT,Simbol INTEGER)");
    }


    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " +Tablica_naziv);
        onCreate(db);
    }
}
```

Kod MainActivity.java datoteke:

```
package plasaj_vuka.sqlite_baza;

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

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

Logika koda:

Cilj je izraditi bazu podataka unutar koje je postavljena tablica spremna za unos podataka. Unutar DataBaseHelper.java koda se kreira DatabaseHelper. S obzirom da su za kreiranje baze podataka potrebna SQLiteOpenHelper svojstva koja funkcija nasljeđuje koristeći *extends*. Da se kreira baza podataka, potreban je naziv baze. Za ovaj primjer je kreirana baza koja sadrži tablicu sa podatcima o plemenitim plinovima (Naziv, simbol,redni broj u sustavu elemenata). Deklarirane su sljedeće varijable: naziv baze podataka (*string*), naziv tablice (*string*), ID, to jest, redni broj unutar tablice, naziv, redni broj u sustavu elemenata, te simbol.

Potrebno je pozvati funkciju koja će kreirati bazu podataka. To je *super(context, BazaPodataka_naziv, null,1).* Parametri koje funkcija uzima su context, naziv baze, factory i verzija. Ova funkcija kreira bazu podataka, zato se naziva Constructor.

Tablicu koja će biti unutar baze podataka je potrebno kreirati prilikom kreiranja baze. Za to služi funkcija onCreate(), u kojoj se koristi naredba *db.execSQL(...)* koja pokreće ono što joj je zadano kao argument. U argumentu je ID-u dodano svojstvo AUTOINCREMENT, što znači da će se ID povećavati za 1 svakoj dodanoj vrijednosti automatski, bez da korisnik mora davati redni broj svakoj unesenoj vrijednosti u tablici.

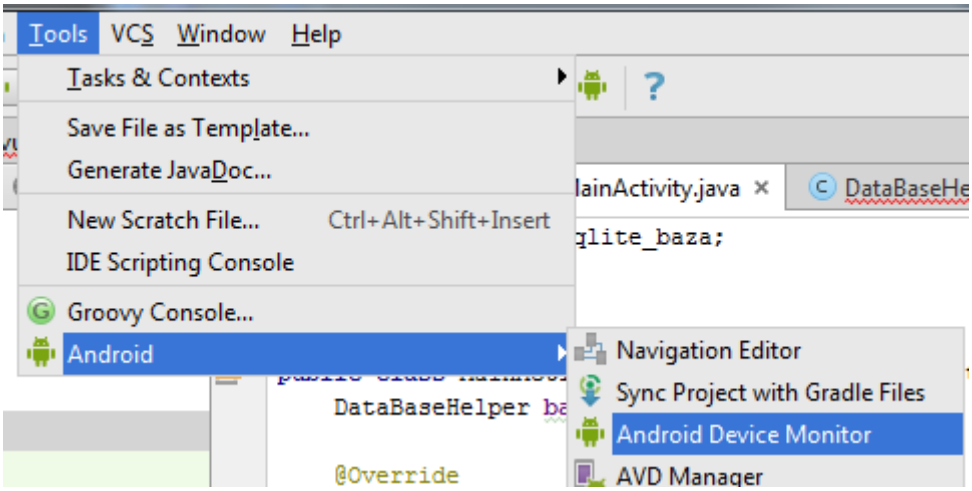
U *onUpgrade* funkciji, koja se koristi prilikom ažuriranja vrijednosti u bazi podataka poziva se .execSQL da makne tablicu ako je već kreirana, i poziva se onCreate() funkcija.

Unutar funkcije koja kreira bazu podataka (DataBaseHelper) sljedeća linija koda kreira bazu podataka i tablicu:

SQLiteDatabase db=this.getWritableDatabase();

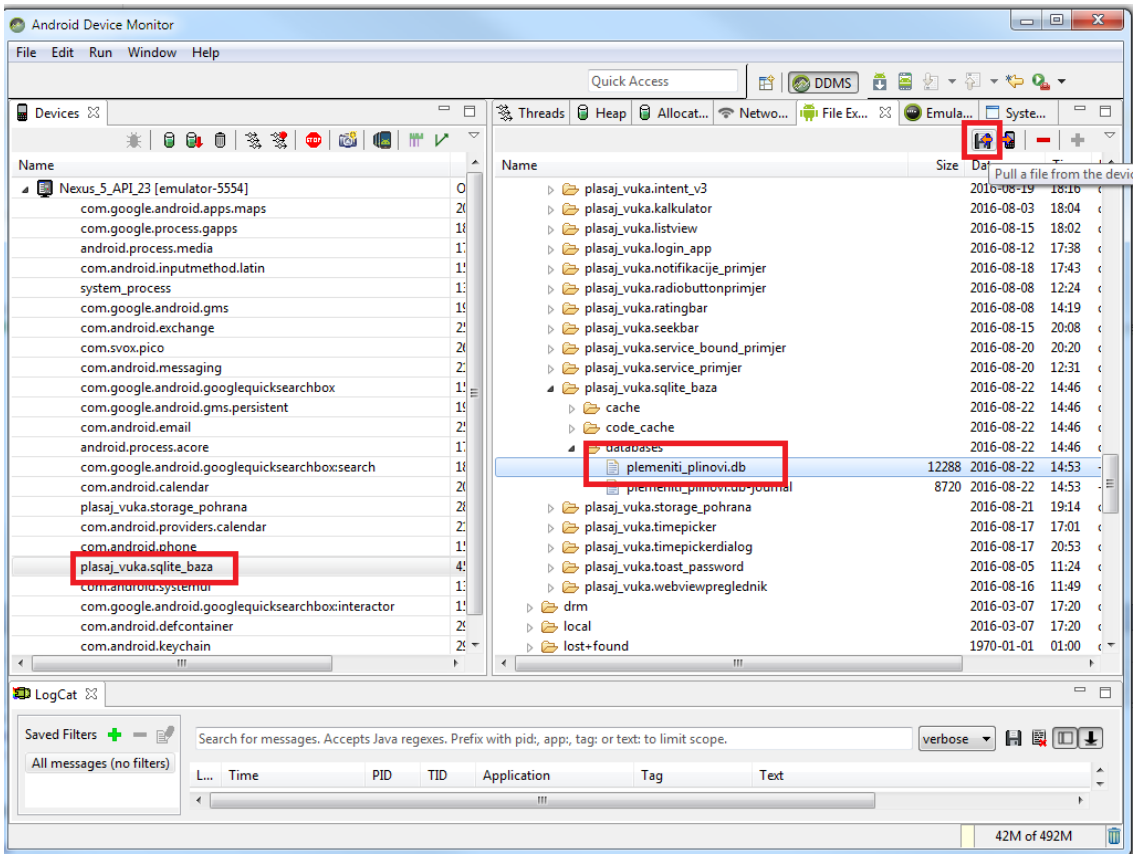
Ova liniju koda se koristi samo za ovaj primjer, kao provjera. Kako će se dodavati funkcije na ovaj kod, koristiti će se drugačije metode pokretanja. Još je potrebno u MainActivity.java datoteci unutar onCreate funkcije pridodati kod koji će pozvati konstruktor baze (*new DataBaseHelper*(*this*)).

Sada bi baza i tablica trebali biti kreirani prilikom pokretanja aplikacije. Provjeru da li je baza podataka napravljena vršimo u Android Device Monitor (slika).



Slika 3.61. Predodžba lokacije AVD-a (Android Device Monitor)

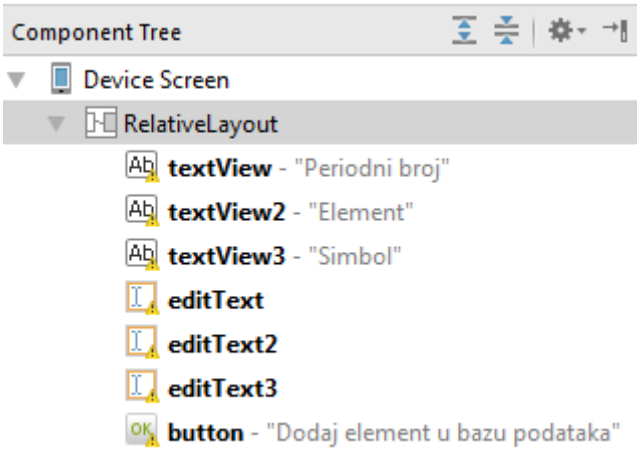
Na lokaciji *data>data>naziv_projekta>database* se nalazi kreirana baza podataka (slika). Nije ju moguće otvoriti ovim putem, ali ju je moguće prekopirati na računalo, gdje je se može otvoriti sa bilo kojim SQLite managerom.



Slika 3.62. Predodžba lokacije datoteke baze podataka.

Ubacivanje podataka u tablicu baze podataka

S obzirom da je potrebno unijeti podatke, na sučelje glavne aktivnosti su dodana tri tekstualna prozora za unos podataka, i tri indikatora koji pojašnjavaju unos podataka (običan tekst ispred prozora za unos), te gumb koji će imati onClick funkciju da izvrši ubacivanje podataka u bazu.



Slika 3.63. Predodžba popisa komponenata.

Kod se nadovezuje na prethodni primjer.

Kod DatabaseHelper.java datoteke:

```
package plasaj_vuka.sqlite_baza;

import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DataBaseHelper extends SQLiteOpenHelper {
    public static final String BazaPodataka_naziv = "plemeniti_plinovi.db";
    public static final String Tablica_naziv =
"plemeniti_plinovi_tablica.db";
    public static final String Stupac1 = "ID";
    public static final String Stupac2 = "periodnirednibroj";
    public static final String Stupac3 = "ime";
    public static final String Stupac4 = "simbol";

    public DataBaseHelper(Context context) {
        super(context, BazaPodataka_naziv, null, 1);

    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("create table " + Tablica_naziv + " (ID INTEGER PRIMARY
KEY AUTOINCREMENT,periodnirednibroj TEXT,ime TEXT,simbol INTEGER)");
    }


    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " +Tablica_naziv);
        onCreate(db);
    }

    public boolean insertData(String periodnirednibroj,String ime,String
simbol) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put(Stupac2,periodnirednibroj);
        contentValues.put(Stupac3,ime);
        contentValues.put(Stupac4,simbol);
        long result = db.insert(Tablica_naziv,null,contentValues);
        if (result == -1)
            return false;
        else
            return true;
    }
}
```

Kod Mainactivity.java datoteke:

```
package plasaj_vuka.sqlite_baza;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    DataBaseHelper baza_podataka;
    EditText periodnirednibroj,ime,simbol;
    Button dodajvrijednost_gumb;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        baza_podataka = new DataBaseHelper(this);
        periodnirednibroj= (EditText)findViewById(R.id.editText);
        ime= (EditText)findViewById(R.id.editText2);
        simbol= (EditText)findViewById(R.id.editText2);
        dodajvrijednost_gumb = (Button)findViewById(R.id.button);
        dodajVrijednosti_funkcija();
    }
    public void dodajVrijednosti_funkcija () {
        dodajvrijednost_gumb.setOnClickListener(
                new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                      boolean provjera =
baza_podataka.insertData(periodnirednibroj.getText().toString(),
                                ime.getText().toString(),
                                simbol.getText().toString());
                        if(provjera=true)
                            Toast.makeText(MainActivity.this, "Element je
ubačen",Toast.LENGTH_LONG).show();
                        else
                            Toast.makeText(MainActivity.this, "Element nije
ubačen",Toast.LENGTH_LONG).show();
                    }
                }
        );
    }
}
```

Logika koda:

U DatabaseHelper.java datoteci se kreira nova funkcija *public boolean insertData()* koja za argumente uzima određene tipove podataka (*String periodnirednibroj,String ime,String simbol*). Konstruktora baze, za kojeg je u prošlom primjeru rečeno da će biti maknut, se prebacuje iz funkcije DatabaseHelper(), u funkciju insertData(). Unutar insertData se koriste varijable *contentValues* za postavljanje podataka u odgovarajuće stupce unutar tablice (argumenti koje koristi su broj stupca, i naziv vrijednosti). *Db.insert()* ubacuje podatke (argumenti koje koristi su ime tablice, *null*, contentValues).

Da bi se ostvarila provjeru uspješnosti umetanja podataka u bazu, potrebno je znati kako naredba .insert funkcionira. Ukoliko se dogodi pogreška pri umetanju, ta funkcija vraća vrijednost od „-1“. Stoga, koristi se if/else metoda gdje se umetnuti podatci uspoređuju sa vrijednošću „-1“. Ako je umetnuta poruka jednaka „-1“, to znači da se je dogodila greška pri umetanju podataka. Na taj način je moguće obavijestiti korisnika da je podatak uspješno ili neuspješno umetnut u tablicu.

Unutar glavne aktivnosti, komponente sa sučelja su povezane sa definiranim varijablama. Također, napravljena je funkcija *dodajVrijednosti_funkcija* koja prilikom pritiska na gumb ubacuje podatke u tablicu koristeći varijablu tipa DataBaseHelper (baza_podataka). Varijabla provjera je tipa boolean tako da postoji true/false odaziv koji se u if/else metodi može koristiti za provjeru. Na kraju, funkciju dodajVrijednosti() se pokreće iz glavne aktivnosti.

Slika 3.64. Predodžba simulacije aplikacije u emulatoru na računalu.

4. Zaključak

S obzirom na rastuće tržište pametnih telefona, kontinuirano raste i tržište mobilnih aplikacija. Većinski udio tog tržišta se odnosi na Android aplikacije. Kako bi razvojni inženjer koji razvija aplikacije bio konkurentan, potrebno je stalno nadopunjavati znanja i prilagođavati se novim verzijama operativnih sustava koji se distribuiraju. Statistike za 2016. godinu, pokazuju da približno 86.2% pametnih telefona koristi Android operativni sustav. Drugi najkorišteniji sustav je iOS (Apple iPhone), a treći je Microsoft sa 0.6% raširenosti. Ovi podatci govore o konkurentnosti tržišta za Android, kao i o obećavajućem opstanku te platforme. Sa znanjem korištenja potrebnih alata, programskog jezika Java, markup jezika XML, PHP skriptiranjem, poznavanja web servera i baza podataka, razvojni inženjer je osposobljen samostalno staviti na tržište konkurentnu, funkcionalnu i komercijalnu aplikaciju.

Literatura

Referentni linkovi:
https://www.jetbrains.com/idea
https://hr.wikipedia.org/wiki/Android_(operacijski_sustav)
https://developer.android.com/studio/features.html
https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B
https://developer.android.com/reference/android/app/Activity.html
https://sqlite.org
https://en.wikipedia.org/wiki/Java_(programming_language)
http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html