Čuvanje podataka u Android aplikacijama

Doc. dr Vladimir Milićević

PRISTUPANJE PREFERENCIJAMA U POKRENUTOJ AKTIVNOSTI

Android obezbeđuje SharedPreferences objekat za čuvanje jednostavnih podataka u aplikacijama.

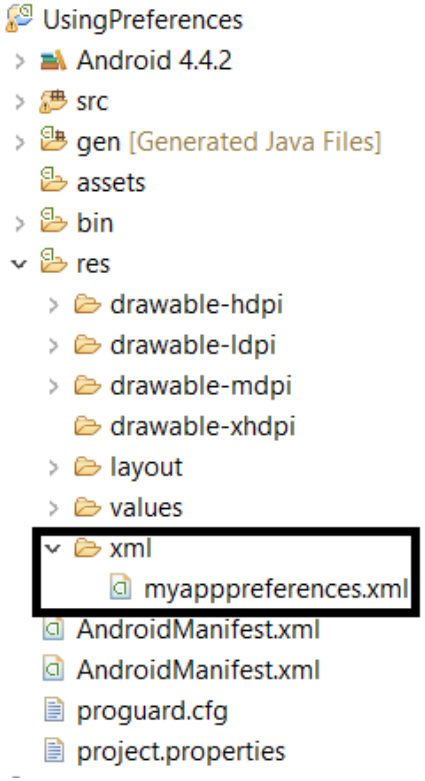
Android operativni sistem obezbeđuje *SharedPreferences* objekat za čuvanje jednostavnih podataka u aplikacijama. Od posebnog značaja su podaci koje je definisao sam korisnik, tako da kada ponovo pokrene aplikaciju, ona bude podešena upravo na način kako to korisnik želi.

Korišćenjem objekta *SharedPreferences* dobija se mogućnost snimanja podataka od značaja u formi para (naziv, vrednost) koji se, zatim, automatski snimaju u odgovarajuću XML datoteku.

Sledećim primerom biće demonstrirano čuvanje podataka upotrebom objekta *SharedPreferences* i kako korisnik može snimljene podatke da direktno modifikuje upotrebom posebnog tipa aktivnosti podržane Android operativnim sistemom.

U *Eclipse IDE* razvojnom okruženju biće kreiran projekat za demonstraciju rada sa preferencijama. Prvi zadatak je kreiranje podfoldera *xml* u okviru kojeg će biti kodirana XML datoteka sa nazivom *myapppreferences.xml*.

Lokacija kreirane datoteke prikazana je sledećom slikom.



Slika-1 Lokacija XML datoteke preferencija

PRIMENA PREFERENCIJA – XML DATOTEKE

XML datotekom preferencija čuvaju se podešavanja aplikacije.

Nakon kreiranja datoteke na lokaciji *res/xml* neophodno je kreirati XML kod koji će definisati izvesna podešavanja aplikacije. Kod datoteke je prikazan na sledeći način.

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <PreferenceCategory android:title="Kategorija 1">
        <CheckBoxPreference
            android:title="Izaberite opciju"
            android:defaultValue="false"
            android:summary="Čekiran ili nečekiran"
            android:key="checkboxPref" />
        </PreferenceCategory>
    <PreferenceCategory android:title="Kategorija 2">
        <EditTextPreference
            android:summary="Učitajte string"
            android:defaultValue="[Učitajte string ovde]"
            android:title="Izmenite tekst"
            android:key="editTextPref"
            />
        <RingtonePreference
            android:summary="Izaberite melodiju zvona"
            android:title="Melodije zvona"
            android:key="ringtonePref"
            />
        <PreferenceScreen
            android:title="Drugi ekran preferencija"
            android:summary=
                "Kliknite za prelazak na drugi ekran"
            android:key="secondPrefScreenPref" >
            <EditTextPreference
                android:summary="Učitajte string"
                android:title="Učitajte string (drugi ekran)"
                android:key="secondEditTextPref"
                />
        </PreferenceScreen>
    </PreferenceCategory>
</PreferenceScreen>
```

Slika-2 XML datoteka preferencija

Takođe, podešavanja korisničkog interfejsa definisana su XML datotekom pod nazivom main.xml.

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

<Button
    android:id="@+id/btnPreferences"
    android:text="Učitajte ekran preferencija"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickLoad"/>

<Button
    android:id="@+id/btnDisplayValues"
    android:text="Prikažite vrednosti preferencija"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickDisplay"/>

<EditText
    android:id="@+id/txtString"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<Button
    android:id="@+id/btnModifyValues"
    android:text="Modifikujte vrednosti preferencija"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickModify"/>
</LinearLayout>
```

Slika-3 XML datoteka main.xml

KLASA TIPA PREFERENCEACTIVITY

U nastavku projekta neophodno je implementirati naslednicu klase PreferenceActivity.

Za prvu verziju softverskog rešenja, i njegovu demonstraciju i testiranje, neophodno je kreirati JAVA klasu aktivnosti koja nasleđuje baznu klasu *PrefereceActivity*. Kod klase dat je sledećom slikom.

```
package net.learn2develop.UsingPreferences;

import android.os.Bundle;
import android.preference.PreferenceActivity;
import android.preference.PreferenceManager;

public class AppPreferenceActivity extends PreferenceActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        PreferenceManager prefMgr = getPreferenceManager();
        prefMgr.setSharedPreferencesName("appPreferences");

        //---učitavanje preferencija iz XML datoteke---
        addPreferencesFromResource(R.xml.myapppreferences);
    }
}
```

Slika-4 JAVA klasa aktivnosti

Kreiranjem XML datoteka preferencija i interfejsa, kreiran je korisnički interfejs koji sadrži sledeće elemente:

- Dve kategorije za obuhvatanje različitih tipova preferencija;
- Dva polja za potvrdu sa ključevima *checkBoxPref* i *secondPrefScreenPref*;
- Opciju za podešavanje melodije zvona sa ključem *ringtonePref*;
- Ekran sa opcijama za dodatne preferencije.

Specijalno, atribut *android:key* definiše ključ koji je moguće programski referencirati u kodu sa ciljem definisanja ili preuzimanja vrednosti određene preferencije.

Da bi Android mogao da prikaže sve opcije koje su dostupne za izmenu, bilo je neophodno kreirati klasu naslednicu osnovne klase *PreferenceActivity*. Izvršavanjem njene metode *addPreferenceFromResource()* učitava se xml datoteka sa odgovarajućim podešavanjima.

UČITAVANJE I MODIFIKACIJA PREFERENCIJA

ZA korišćenje podešavanja neophodno je koristiti klasu SharedPreferences.

Do sada je pokazano kako klasa *PreferenceActivity* omogućava jednostavno kreiranje preferencija i modifikovanje određenih opcija. Da bi u toku izvršavanja aplikacije bilo moguće koristiti navedena podešavanja neophodno je koristiti klasu *SharedPreferences*. U nastavku biće kreirana klasa pod nazivom *UsingPreferenceActivity* koja nasleđuje baznu klasu *Activity* i koristi klasu *SharedPreferences*.

```
package net.learn2develop.UsingPreferences;
import android.app.Activity;
public class UsingPreferencesActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void onClickLoad(View view) {
        Intent i = new Intent("net.learn2develop.AppPreferenceActivity");
        startActivity(i);
    }
    public void onClickDisplay(View view) {
        SharedPreferences appPrefs =
                getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                        MODE_PRIVATE);
        DisplayText(appPrefs.getString("editTextPref", ""));
    }
    public void onClickModify(View view) {
        SharedPreferences appPrefs =
                getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                        MODE_PRIVATE);

        SharedPreferences.Editor prefsEditor = appPrefs.edit();
        prefsEditor.putString("editTextPref",
                ((EditText) findViewById(R.id.txtString)).getText().toString());
        prefsEditor.commit();
    }
    private void DisplayText(String str) {
        Toast.makeText(getBaseContext(), str, Toast.LENGTH_LONG).show();
    }
}
```

```
import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
```

Slika-5 Upotreba SharedPreferences

UČITAVANJE I MODIFIKACIJA PREFERENCIJA - FUNKCIONISANJE

Objekat klase SharedPreferences dobijen je pozivom metode getSharedPreferences() .

Prva značajna metoda (sledeća slika) koja je implementirana u klasi *UsingPreferenceActivity* je *onClickDisplay()*. U okviru ove metode dešava se poziv metode *getSharedPreferences()* pomoću koje je dobijen objekat klase *SharedPreferences.* Navedeno je omogućeno kroz specificiranje naziva XML datoteke (u konkretnom slučaju *net.learn2develop.UsingPreferences_preferences*) formatirano kao *nazivPaketa.NazivKlase_preferences.* Učitavanje podataka tipa string, realizovano je pozivom metode *getString()* kojoj je predat ključ preferencije koja se učitava. Konstanta *MODE_PRIVATE* ukazuje da datoteka sa preferencijama može da se koristi samo u aplikaciji projekta u kojem je kreirana.

```
public void onClickDisplay(View view) {

    SharedPreferences appPrefs =
            getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                    MODE_PRIVATE);
    DisplayText(appPrefs.getString("editTextPref", ""));
}
```

Slika-6 Prikazivanje i preuzimanje preferencija

U sledećoj metodi, *onClickModify()*, kreiran je objekat *SharedPreferences.Editor* upotrebom metode *edit()* pozvane objektom klase *SharedPreferences*. Za primenu vrednosti opcije, tipa string, iskorišćena je metoda *putString()*, a snimanje promene, u datoteku preferencija, obavljeno je metodom *commit()* (sledeća slika).

```
public void onClickModify(View view) {

    SharedPreferences appPrefs =
            getSharedPreferences("net.learn2develop.UsingPreferences_preferences",
                    MODE_PRIVATE);
    SharedPreferences.Editor prefsEditor = appPrefs.edit();
    prefsEditor.putString("editTextPref",
            ((EditText) findViewById(R.id.txtString)).getText().toString());
    prefsEditor.commit();
}
private void DisplayText(String str) {
    Toast.makeText(getBaseContext(), str, Toast.LENGTH_LONG).show();
}
```

Slika-6 Snimanje preferencija

PROMENA PODRAZUMEVANOG NAZIVA DATOTEKE SA PREFERENCIJAMA

Ponekad je korisno dati specifičan naziv datoteci sa preferencijama.

U konkretnom slučaju, datoteka sa prefernecijama je snimljena na mobilnom uređaju pod nazivom *net.learn2develop.UsingPreferences_preferences.* Međutim, programeri često daju izvesne specifične nazive datotekama sa preferencijama. To je moguće učiniti ako u klasi *AppPreferencesActivity.java* napravimo obeležene korekcije.

```
PreferenceManager prefMgr = getPreferenceManager();
prefMgr.setSharedPreferencesName("appPreferences");

//---učitavanje preferencija iz XML datoteke---
addPreferencesFromResource(R.xml.myapppreferences);
```

Slika-8 PreferenceManager

Klasa *UsingPreferenceActivity.java* definiše naziv datoteke za čuvanje preferencija, a to je *appPreferences.xml*, obeleženom modifikacijom prvobitnog koda.

```
package net.learn2develop.UsingPreferences;
import android.app.Activity;
public class UsingPreferencesActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    public void onClickLoad(View view) {
        Intent i = new Intent("net.learn2develop.AppPreferenceActivity");
        startActivity(i);
    }
    public void onClickDisplay(View view) {

        SharedPreferences appPrefs =
                getSharedPreferences("appPreferences", MODE_PRIVATE);

        DisplayText(appPrefs.getString("editTextPref", ""));
    }
    public void onClickModify(View view) {

        SharedPreferences appPrefs =
                getSharedPreferences("appPreferences", MODE_PRIVATE);

        SharedPreferences.Editor prefsEditor = appPrefs.edit();
        prefsEditor.putString("editTextPref",
                ((EditText) findViewById(R.id.txtString)).getText().toString());
        prefsEditor.commit();
    }
    private void DisplayText(String str) {
        Toast.makeText(getBaseContext(), str, Toast.LENGTH_LONG).show();
    }
}
```

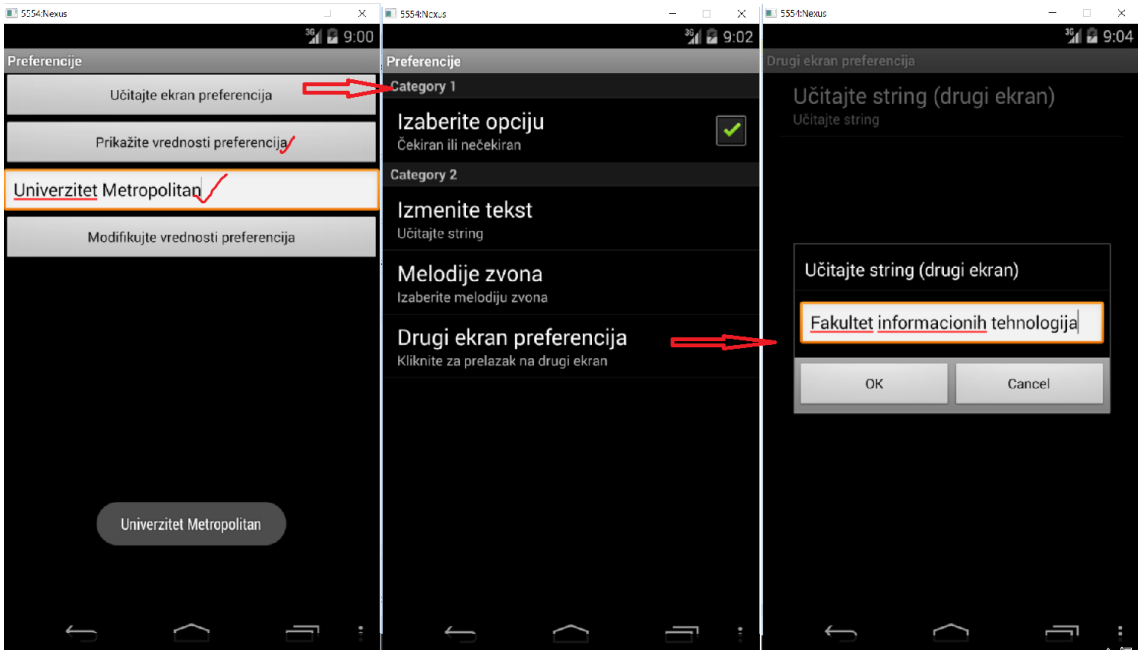
```
import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
```

Slika-9 Definisanje datoteke preferencija

PRIMER SA PREFERENCIJAMA - DEMONSTRACIJA

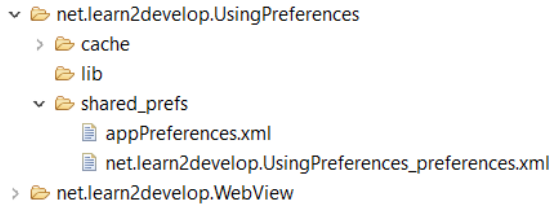
Nakon promene naziva podrazumevane datoteke sa preferencijama, obe datoteke su sačuvane u odgovarajućem folderu.

Klikom na F11, prevodi se program i emulatorom se prikazuje mobilni uređaj koji koristi program za upravljanje preferencijama. Navigacija kroz program prikazana je sledećom slikom.



Slika-10 Program za rad sa preferencijama

U primeru je prikazan način kako je moguće programski promeniti podrazumevani naziv datoteke sa preferencijama. Obe datoteke, izvorna i nova, čuvaju se u uređaju u folderu *data/data/net.learn2develop.UsingPreferences/shared_prefs* (sledeća slika).



Slika-11 Datoteke preferencija

ČUVANJE PODATAKA U INTERNOJ MEMORIJI

Klase koje omogućavaju tradicionalno čuvanje podataka nalaze se u paketu java.io.

U prethodnom slučaju je pokazano da objekat klase *SharedPreferences* omogućava snimanje podataka u formi para (naziv, vrednost). Međutim, ponekad je poželjno čuvati podatke na tradicionalan način, u datotekama, poput raznih tekstualnih dokumenata koji bi trebalo da budu prikazani u nekoj Android aplikaciji. Kompletna podrška, klasa za rad sa datotekama, nalazi je u JAVA paketu *java.io*.

Mobilni uređaji dozvoljavaju dva načina čuvanja podataka u datotekama:

- Snimanje datoteka sa podacima na internu memoriju uređaja;
- Snimanje datoteka sa podacima na eksternu memoriju uređaja.

Oba načina čuvanja podataka biće prikazana, a sada će se krenuti od čuvanja podataka u internoj memoriji Android uređaja. U tu svrhu biće demonstrirano kako se string, kojeg unosi korisnik kroz interakciju sa korisničkim interfejsom, snima u datoteku na unutrašnjem memorijskom prostoru mobilnog uređaja.

Prvi korak će biti kreiranje projekta sa nazivom *Files* za koji se kreira main.xml datoteka sa sledećim XML kodom.

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

<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Unesite neki tekstualni podatak!" />

<EditText
    android:id="@+id/txtText1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />

<Button
    android:id="@+id/btnSave"
    android:text="Snimite"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickSave" />

<Button
    android:id="@+id/btnLoad"
    android:text="Učitajte"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:onClick="onClickLoad" />
</LinearLayout>
```

Slika-1 main.xml datoteka projekta

ČUVANJE PODATAKA U INTERNOJ MEMORIJI – JAVA KLASA

Metode klase za rad sa ulazno/izlaznim podacima datoteka moraju da obrađuju izuzetke.

Sledećim kodom biće prikazana JAVA klasa aktivnosti projekta. Moguće je primetiti da su sve programske manipulacije sa učitavanjem, prikazivanjem i snimanjem podataka praćene upravljanjem izuzecima i realizacijom *try – catch* blokova koda.

```
package net.learn2develop.Files;
import java.io.BufferedReader;
public class FilesActivity extends Activity {
    EditText textBox;
    static final int READ_BLOCK_SIZE = 100;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        textBox = (EditText) findViewById(R.id.txtText1);

        InputStream is = this.getResources().openRawResource(R.raw.textfile);
        BufferedReader br = new BufferedReader(new InputStreamReader(is));
        String str = null;
        try {
            while ((str = br.readLine()) != null) {
                Toast.makeText(getBaseContext(),
                    str, Toast.LENGTH_SHORT).show();
            }
            is.close();
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
```

```
public void onClickSave(View view) {
    String str = textBox.getText().toString();
    try
    {    //---Izbor skladišta datoteke---
        FileOutputStream fOut =
                openFileOutput("textfile.txt",
                        MODE_WORLD_READABLE);

        OutputStreamWriter osw = new
                OutputStreamWriter(fOut);

        //---Upisivanje stringa u datoteku---
        osw.write(str);
        osw.flush();
        osw.close();

        //---Poruka o snimljenom podatku---
        Toast.makeText(getBaseContext(),
                "Datoteka je uspešno snimljena!
                Toast.LENGTH_SHORT).show();

        //---Čisti EditText pogled---
        textBox.setText("");
    }
    catch (IOException ioe)
    {
        ioe.printStackTrace();
    }
}
```

```
public void onClickLoad(View view) {
    try
    {   //---Izbor skladišta datoteke---
        FileInputStream fIn =
                openFileInput("textfile.txt");
        InputStreamReader isr = new
                InputStreamReader(fIn);

        char[] inputBuffer = new char[READ_BLOCK_SIZE];
        String s = "";

        int charRead;
        while ((charRead = isr.read(inputBuffer))>0)
        {
            //---Konverzija niza karaktera u string---
            String readString =
                    String.copyValueOf(inputBuffer, 0,
                            charRead);
            s += readString;

            inputBuffer = new char[READ_BLOCK_SIZE];
        }
        //---Podešavanje EditText na učitani tekst
        textBox.setText(s);

        Toast.makeText(getBaseContext(),
                "Datoteka je uspešno učitana!",
                Toast.LENGTH_SHORT).show();
    }
    catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

Slika-2 JAVA klasa projekta

ČUVANJE PODATAKA U INTERNOJ MEMORIJI - FUNKCIONISANJE

Podaci se čuvaju kao niz karaktera.

Da bi neki sadržaj bio sačuvan u datoteci, neophodno je koristiti klasu *FileOutputStream*. Korišćenjem metode *openFileOutput()* otvara se datoteka za snimanje podataka u odgovarajućem režimu. Režimi mogu biti:

- U konkretnom slučaju, režim je *MODE_WORD_READABLE*, a to znači da datoteku mogu čitati i druge apliakcije;
- *MODE_PRIVATE* – datoteku može da čita jedino aplikacija koja ju je kreirala;
- *MODE_APPEND* – podaci se dodaju na kraj datoteke;
- *MODE_WORD_WRITABLE* – datoteci mogu da pristupaju i druge aplikacije sa mogućnošću snimanja podataka u datoteku.

U datoteci, podaci se čuvaju kao niz karaktera i, da bi bili učitani, neophodno ih je konvertovati u string. Takođe, pre upisivanja podataka u datoteku, podaci tipa string se prevode u niz karaktera.

Da bi niz karaktera bio konvertovan u string, koristi se objekat klase *OutputStreamWriter* kojem se prosleđuje objekat tipa *FileOutputStream*. Navedeno pokazuje priloženi kod klase aktivnosti.

Nakon konverzije, primenom metode *write()* vrši se snimanje stringa u datoteku. Da bi bilo osigurano da će svi bajtovi (karakteri) biti snimljeni u datoteci, koristi se metoda *flush()*. Na kraju, metodom *close()* zatvara se datoteka za upisivanje (videti kod klase ativnosti projekta).

Da bi sadržaj neke datoteke mogao biti učitan, koristi se klasa *FileInputStream* u kombinaciji sa klasom *InputStreamReader*. Nakon konverzije niza karaktera u string, vrši se njegovo učitavanje. Često nije poznata veličina datoteke koja se učitava. Zato taj sadržaj može da se prosleđuje u bafer koji predstavlja, u konkretnom slučaju, blok od 100 karaktera. Sledećim kodom je prikazano učitavanje stringa iz datoteke uz napomenu da metoda *read()* proverava broj karaktera koji se učitavaju i vraća vrednost -1 ukoliko je dotignut kraj datoteke.

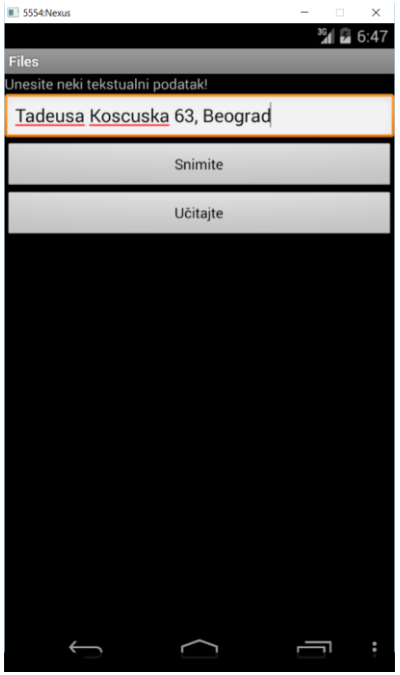
```
char[] inputBuffer = new char[READ_BLOCK_SIZE];
String s = "";
int charRead;
while ((charRead = isr.read(inputBuffer))>0)
{//---Konverzija niza karaktera u string---
    String readString =
            String.copyValueOf(inputBuffer, 0,
                    charRead);
    s += readString;
    inputBuffer = new char[READ_BLOCK_SIZE];
}
```

Slika-3 Učitavanje stringa iz datoteke

ČUVANJE PODATAKA U INTERNOJ MEMORIJI - DEMONSTRACIJA

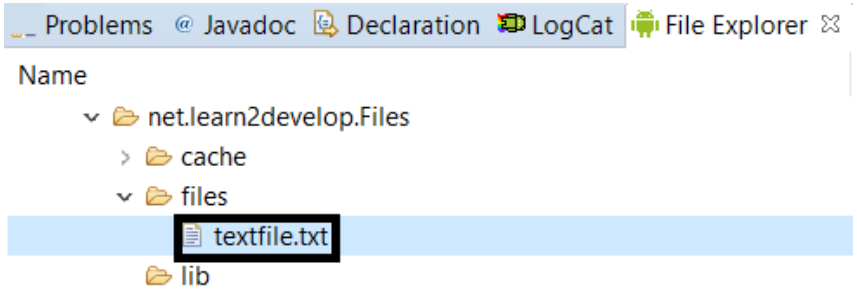
Moguće je u DDMS prikazu proveriti da li je aplikacija zaista snimila datoteku.

Klikom na F11, program se prevodi i pokreće se AVD uređaj. Korisnički interfejs nudi opcije čuvanja i čitanja podataka iz datoteka i polje za unos teksta koji može biti upisan u datoteku. Navedeno je prikazano sledećom slikom.



Slika-4 UI aplikacije za rad sa datotekama

Moguće je u DDMS prikazu, veoma lako korišćenjem *File Explorer-a*, proveriti da li je aplikacija snimila ciljanu datoteku u folder sa fajlovima aplikacije. Provera je prikazana sledećom slikom.



Slika-5 Snimljena datoteka

ČUVANJE PODATAKA U EKSTERNOJ MEMORIJI

Mobilnost SD kartice omogućava lakšu razmenu podataka između korisnika.

U prethodnom izlaganju je prikazano kako se čuvaju podaci na unutrašnjem memorijskom skladištu mobilnog uređaja. Ponekad je korisno snimiti određene datoteke na SD karticu uređaja, zbog njenog većeg kapaciteta ili olakšane distribucije podataka - vađenjem kartice iz uređaja i prosleđivanjem drugom korisniku. Na prethodnom primeru biće napravljena korekcija u formi zamene određenih linija koda metoda *onClick()* i *onClickLoad()*.

onClick()

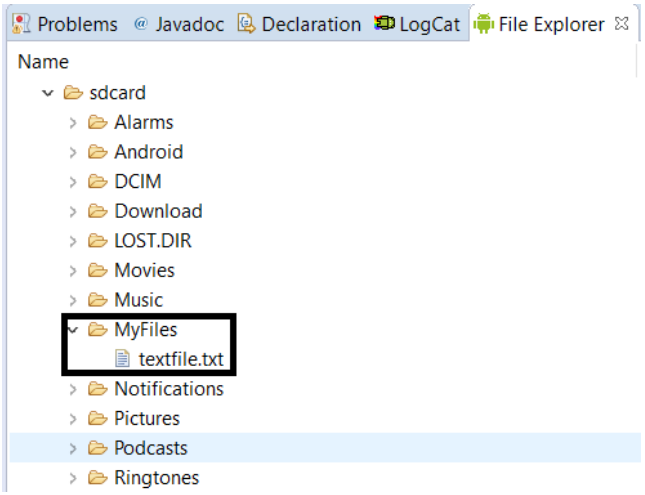
```
//---Stari kod---
/*FileOutputStream fOut =
        openFileOutput("textfile.txt",
                MODE_WORLD_READABLE);*/
//---Novi kod---
File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() +
    "/MyFiles");
directory.mkdirs();
File file = new File(directory, "textfile.txt");
FileOutputStream fOut = new FileOutputStream(file);
```

onClickLoad()

```
//---Stari kod---
/*FileInputStream fIn =
        openFileInput("textfile.txt");
InputStreamReader isr = new
        InputStreamReader(fIn);*/
//---Novi kod--
File sdCard = Environment.getExternalStorageDirectory();
File directory = new File (sdCard.getAbsolutePath() +
"/MyFiles");
File file = new File(directory, "textfile.txt");
FileInputStream fIn = new FileInputStream(file);
InputStreamReader isr = new InputStreamReader(fIn);
```

Slika-6 Kod za rad sa SD karticom

U prikazanom kodu, metoda *getExternalStorageDirectory()* vraća apsolutnu putanju do eksterne memorije uređaja. Putanja je najčešće predstavljena kao */sdcard* na realnom uređaju ili */mnt/sdcard* ukoliko se aplikacija izvršava na emulatoru. Budući da naziv putanje može da bude različit, u zavisnosti od proizvođača kartice, ne preporučuje se ovaj način obeležavanja kartice u samom kodu. Upravo iz navedenog razloga, primena date metode je od izuzetnog značaja. Datoteka će biti snimljena u folder *MyFiles* na SD kartici (videti kod i sliku).



Slika-7 Datoteka na SD kartici

ČUVANJE PODATAKA U EKSTERNOJ MEMORIJI – DODAVANJE PRIVILEGIJA

Bez dodavanja privilegije SD kartici, snimanje podataka na nju ne bi bilo moguće.

Eksterna kartica predstavlja dopunski element priključen na hardver mobilnog uređaja. Kao takvom, moraju mu se obezbediti mehanizmi pristupa i manipulacije njegovim sadržajem. U tu svrhu, u AndroidManifest.xml datoteci, neophodno je dodati *WRITE_EXTERNAL_STORAGE* privilegiju za snimanje podataka u eksternu memoriju. Sledećom slikom je priložena AndroidManifest.xml datoteka projekta *Files*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.Files"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

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

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

Slika-8 Privilegija snimanja na SD karticu

IZBOR OPTIMALNE OPCIJE ZA SNIMANJE

Za izbor načina čuvanja podataka u Android aplikacijama neophodno je pratiti neke prihvaćene preporuke.

U prethodnim izlaganjima obrađeno je čuvanje podataka u Android aplikacijama na tri načina. Korišćene su deljene preferencije, unutrašnja i spoljašnja memorijska skladišta Android uređaja. Da bi bio izabran pravi način čuvanja podataka, u Android aplikacijama, neophodno je poštovati određene preporuke izbora:

- Ukoliko se manipuliše podacima koji mogu da se prikažu u obliku para (naziv, vrednost) biće korišćene deljene preferencije. Na primer, deljenjim preferencijama moguće je sačuvati sledeće parove vrednosti: (korisnik, datum rođenja), (korisnik, vreme prijavljivanja na sistem), (pozadina, boja pozadine), (zvono, melodija zvona) itd.
- Ukoliko je neophodno brzo snimanje podataka poput preuzimanja slika sa web stranice, da bi ih neka ugrađena aplikacija naknadno prikazala, interna memorija mobilnog uređaja je dobar izbor za čuvanje podataka.
- U situacijama kada je neophodno olakšano razmenjivanje podataka sa drugim korisnicima, i kada je neophodno rasteretiti resurse unutrašnje memorije, trebalo bi koristiti skladište kao što je SD kartica za čuvanje podataka.

UPOTREBA STATIČKIH RESURSA

Moguće je koristiti i datoteke koje nisu kreirane dinamički, u aplikaciji, za čuvanje podataka.

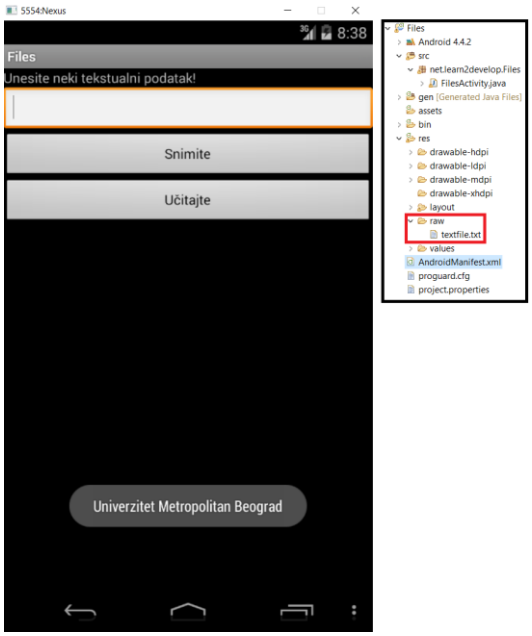
Pored datoteka koje se dinamički generišu aplikacijom, podaci, u Android aplikacijama, mogu biti čuvani i preuzimani iz datoteka koje su dodate manuelno u paket aplikacije. U konkretnom primeru, u folderu *res/raw* projekta, ubačena je datoteka sa nazivom *textfile.txt* u kojoj je sačuvan string koji odgovara nazivu našeg Univerziteta. Da bi ova datoteka bila iskorišćena, neophodno je uključiti *getResources()* metodu, klase *Acivity,* koja vraća objekat tipa *Resources()*. Nakon toga, primenjuje se metoda *openRawResource()* sa ciljem otvaranja željene datoteke. Sledećom slikom su prikazani neophodni paketi sa klasama i kod koji je neophodno implementirati u *onCreate()* metodu.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
```

```
InputStream is = this.getResources().openRawResource(R.raw.textfile);
BufferedReader br = new BufferedReader(new InputStreamReader(is));

String str = null;
try {
    while ((str = br.readLine()) != null) {
        Toast.makeText(getBaseContext(),
            str, Toast.LENGTH_SHORT).show();
    }
    is.close();
    br.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

Slika-9 Kod za rad sa statičkom datotekom

Identifikator resursa, koji je smešten u *res/raw* folder, dobija naziv na osnovu naziva datoteke bez odgovarajuće ekstenzije. U konkretnom slučaju to je *R.raw.textfile*. Sledećom slikom je pokazano preuzimanje podataka aplikacijom iz statičke datoteke i njena lokacija u hijerarhiji projekta.



Slika-10 Lokacija i podatak iz datoteke

UVOD U ANDROID BAZE PODATAKA

Za snimanje relacionih podataka u Android aplikacijama, biće korišćene baze podataka.

Tehnike prikazane u prethodnim delovima lekcije bavile su se čuvanjem skupova podataka. Ukoliko se žele sačuvati podaci, koji su različitog tipa, mogu biti tabelarno prikazani i povezani odgovarajućim relacijama, koristiće se baze podataka. Na primer, potrebno je kreirati Android aplikaciju koja će obraditi i sačuvati rezultate nekog ispita. Mnogo je efikasnije koristiti bazu podataka za čuvanje i prikazivanje podataka iz razloga što je nad njom moguće kreirati brojne upite, a sa ciljem dobijanja konkretnih podataka u vezi sa studentima koji su polagali ispit. Takođe, baze podataka obezbeđuju integritet podataka kroz specificiranje veza između različitih tabela.

Android operativni sistem podržava *SQLite* sistem za upravljanje bazom podataka. Ovde je potrebno napomenuti da baza podataka koja je kreirana za određenu Android aplikaciju, može da se koristi isključivo u toj aplikaciji i druge Android aplikacije nemaju pristup navedenoj bazi podataka.

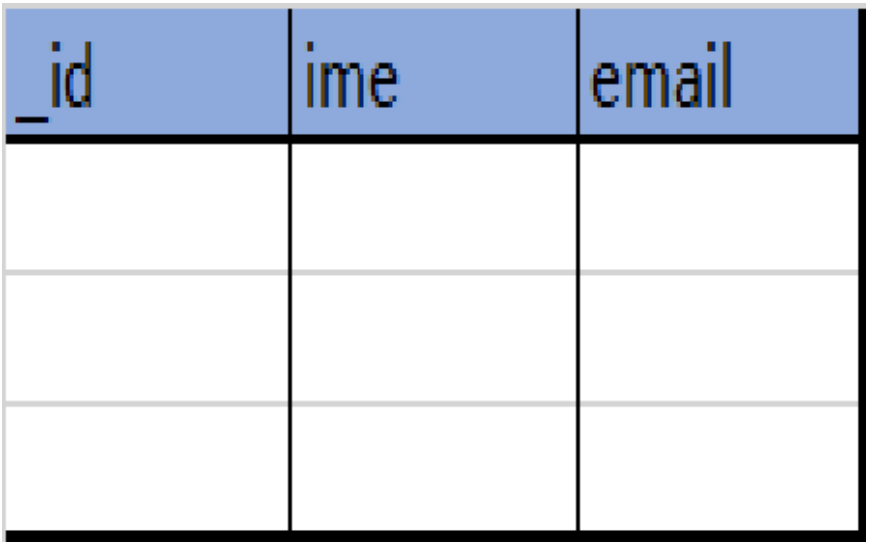
KREIRANJE KLASE DBADAPTER

Prilikom rada sa bazama podataka, praksa je kreiranje pomoćne klase koja enkapsulira postupak pristupa podacima.

U daljem izlaganju, cilj je pokazivanje načina kreiranja *SQLite* baze podataka u Android aplikaciji. Kreirana baza podataka, u Android operativnom sistemu, uvek se nalazi, za datu aplikaciju, u folderu *data/data/nazivPaketa/databases*.

Dobra praksa, u radu sa Android bazama podataka, jeste kreiranje pomoćne klase koja enkapsulira veoma složen postupak pristupa podacima. Iz navedenog razloga, biće kreirana klasa *DBAdapter* koja će omogućiti kreiranje, otvaranje i zatvaranje baze podataka, kao i sve poznate načine manipulacije nad podacima pohranjenim u bazi.

Za potrebe demonstracije, biće uveden primer koji podrazumeva kreiranje baze podataka sa jednom tabelom pod nazivom *kontakti*. Tabela će biti izgrađena od tri kolone: *_id, ime* i *email* (sledeća slika). Bazu podataka će se zvati *MyDB*.



Slika-1 Tabela kreirane baze podataka

KREIRANJE KLASE DBADAPTER – JAVA KOD

U okviru JAVA koda, kao stringovi, biće implementirani i upiti za izvršavanje određenih akcija nad bazom podataka.

Sledeći zadatak predstavlja kreiranje JAVA koda pomoćne klase *DBAdapter.java*. Klasom će biti definisano: kreiranje, zatvaranje i otvaranje baze, učitavanje i upisivanje podataka, a biće implementirani i određeni upiti za izvršavanje određenih akcija nad bazom podataka. Lokacija klase biće standardi folder projekta *src*.

```
package net.learn2develop.Databases;
import android.content.ContentValues;
public class DBAdapter {
    static final String KEY_ROWID = "_id";
    static final String KEY_IME = "ime";
    static final String KEY_EMAIL = "email";
    static final String TAG = "DBAdapter";
    static final String DATABASE_NAME = "MyDB";
    static final String DATABASE_TABLE = "kontakti";
    static final int DATABASE_VERSION = 2;

    static final String DATABASE_CREATE =
        "create table kontakti (_id integer primary key autoincrement, "
        + "name text not null, email text not null);";

    final Context context;

    DatabaseHelper DBHelper;
    SQLiteDatabase db;

    public DBAdapter(Context ctx){
        this.context = ctx;
        DBHelper = new DatabaseHelper(context);
    }

    private static class DatabaseHelper extends SQLiteOpenHelper
    {
        DatabaseHelper(Context context){
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db){
            try {
                db.execSQL(DATABASE_CREATE);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Ažuriranje baze podataka iz verzije " +
                oldVersion + " u verziju "
                + newVersion + ", uništiće stare podatke.");
        db.execSQL("DROP TABLE IF EXISTS kontakti");
        onCreate(db);
    }
}
```

```
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
```

```
//---otvaranje baze podataka---
public DBAdapter open() throws SQLException {
    db = DBHelper.getWritableDatabase();
    return this;
}
//---zatvaranje baze podataka---
public void close(){
    DBHelper.close();
}
//---umetanje kontakta u bazu podataka---
public long insertContact(String ime, String email){
    ContentValues initialValues = new ContentValues();
    initialValues.put(KEY_IME, ime);
    initialValues.put(KEY_EMAIL, email);
    return db.insert(DATABASE_TABLE, null, initialValues);
}
//---brisanje konkretnog kontakta---
public boolean deleteContact(long rowId){
    return db.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}
//---preuzimanje svih kontakata---
public Cursor getAllContacts(){
    return db.query(DATABASE_TABLE, new String[] {KEY_ROWID, KEY_IME,
            KEY_EMAIL}, null, null, null, null, null);
}
//---preuzimanje konkretnog kontakta---
public Cursor getContact(long rowId) throws SQLException{
    Cursor mCursor =
            db.query(true, DATABASE_TABLE, new String[] {KEY_ROWID,
            KEY_IME, KEY_EMAIL}, KEY_ROWID + "=" + rowId, null,
            null, null, null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}

//---ažuriranje kontakta---
public boolean updateContact(long rowId, String ime, String email){
    ContentValues args = new ContentValues();
    args.put(KEY_IME, ime);
    args.put(KEY_EMAIL, email);
    return db.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
}
}
```

Slika-2 Klasa DBAdapter

KLASA DBADAPTER - FUNKCIONISANJE

Primenom klase Cursor omogućeno je efikasnije upravljanje vrstama i kolonama.

Na samom početku klase kreirano je nekoliko konstanti koje se odnose na polja tabele koja će biti kreirana u bazi podataka, npr. **static final String *KEY_ROWID = "_id".*** Posebno je konstantom *DATABASE_CREATE* predstavljena *SQL* naredba za kreiranje tabele *kontakti*.

U nastavku, u klasu *DBAdapter* ugrađena je privatna klasa *DataBaseHelper*, naslednica klase *SQLiteOpenHelper,* čiji je zadataka asistiranje Android operativnom sistemu prilikom kreiranja baze podataka i upravljanja njenim verzijama.

Metoda *onCreate()* kreiraće novu bazu podataka, ukoliko ona nije prisutna u trenutku poziva metode. Ukoliko je neophodno poboljšati bazu podataka, metoda *onUpgrade()* biće pozvana. Za proveru verzije baze podataka, koju je neophodno poboljšati, metoda proverava konstantu *DATABASE_VERSION*.

U daljem radu su kreirane metode za otvaranje i zatvaranje baze podataka, kao i za dodavanje, brisanje i ažuriranje vrsta u tabeli *kontakti* (videti priloženi kod klase *DBAdapter*).

Posebnu pažnju je neophodno obratiti na klasu *Cursor* koju Android operativni sistem koristi kao povratnu vrednost za upit. Kursor predstavlja tip pokazivača na skup dobijen izvršavanjem upita nad bazom podataka.

Primena ove klase, u Android operativnom sistemu, omogućava efikasnije upravljanje vrstama i kolonama tabele.

Za snimanje podataka u bazu, u formi para (naziv vrednost), korišćen je *ContentValue* objekat koji, primenom metode *put()*, vrši snimanje kao na primer, initialValues.put (***KEY_IME, ime).***

Konačno, da bi kreiranje baze podatak bilo moguće, u aplikaciji je neophodno kreirati objekat klase *DBAdapter* primenom konstuktora sa sledeće slike. Ovaj konstruktor kreira objekat klase *DatabaseHelper* da bi kreirao novi bazu podataka.

```
public DBAdapter(Context ctx)     //konstruktor
{
    this.context = ctx;
    DBHelper = new DatabaseHelper(context);
}

private static class DatabaseHelper extends SQLiteOpenHelper
{
    DatabaseHelper(Context context)
    {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
```

Slika-3 DBAdapter konstruktor

PROGRAMSKO MANIPULISANJE BAZOM PODATAKA - CRUD OPERACIJE

Nad SQLite bazom podataka izvršavaju se standardne CRUD operacije.

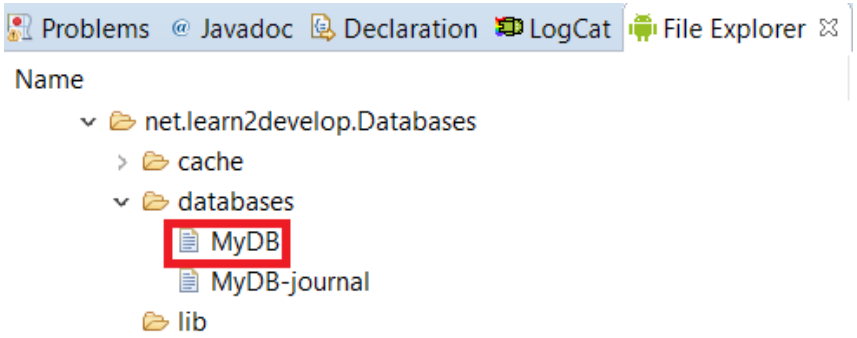
Nakon kreiranja *DBAdapter* pomoćne klase sve je spremno za rad sa bazom podataka. U nastavku lekcije biće pokazano kako se nad SQLite bazom podataka izvršavaju se standardne *CRUD* operacije (create, read, update i delete). U tu svrhu, neophodno je bilo kreirati klasu aktivnosti aplikacije pod nazivom *DatabasesActivity*.

Za dodavanje novih kontakata u bazu podataka, neophodno je u *onCreate()* metodu, klase aktivnosti, dodati kod prikazan sledećom slikom.

```
package net.learn2develop.Databases;

import java.io.File;

public class DatabasesActivity extends Activity {
    /** Poziva se kada se aktivnost kreira. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        DBAdapter db = new DBAdapter(this);


        //---dodavanje kontakata---
        db.open();
        long id = db.insertContact("vlada", "v@gmail.com");
        id = db.insertContact("pera", "p@gmail.com");
        id = db.insertContact("maja", "m@gmail.com");
        db.close();

    }
}
```

Slika-4 Dodavanje u bazu podataka

Iz koda se vidi da je prvo kreiran objekat klase *DBAdapter*: **DBAdapter** db = new **DBAdapter**(this). Objekat poziva metodu *insertContact()* koja vraća identifikator unete vrste. Ukoliko se, prilikom izvršavanja metode, javi greška, vraća se vrednost -1. Pokretanjem aplikacije, uređajem ili emulatorom, moguće je uočiti, primenom DDMS i File Explorer-a, da je baza podataka *MyDB* kreirana u *databases* folderu (sledeća slika).



Slika-5 Folder databases

CRUD OPERACIJE – UČITAVANJE KONTAKATA

Implementacijom metode getAllContacts(), DBAdapter klase, prikazuju se svi kontakti.

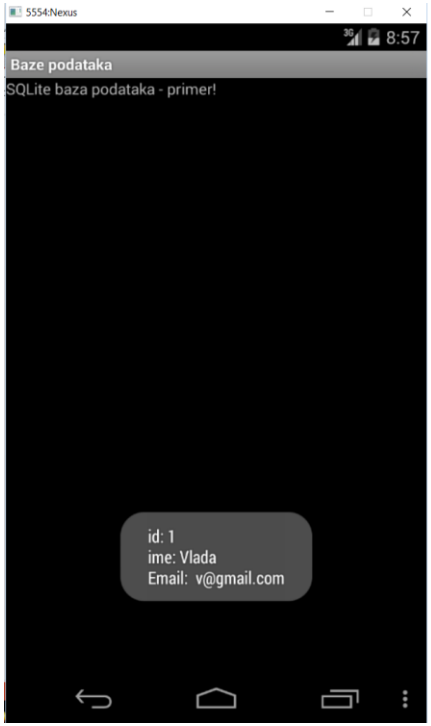
Da bi svi kontakti, iz kreirane *MyDB* baze podataka, bili učitani, neophodno je implementirati metodu *getAllContacts()* klase *DBAdapter*. Takođe, neophodno je implementirati metodu *DisplayContact()* (sledeća slika), koja preuzima objekat tipa *Cursor*, za prikazivanje pojedinačnih kontakata. Prikazivanje kontakata funkcioniše na sledeći način:

- Metoda *getAllContacts()* klase *DBAdapter* učitava sve kontakte koji se nalaze u bazi podataka;
- Rezultat učitavanja se vraća kao *Cursor* objekat;
- *Cursor* objekat izvršava metodu *moveToFirst()*;
- Ukoliko je izvršavanje metode uspešno, sledeća, *DisplayContact()*, metoda prikazaće konkretan kontakt.
- Metoda *moveToNext()* pomera pokazivač na sledeći kontakt.

```
public void DisplayContact(Cursor c)
{
    Toast.makeText(this,
            "id: " + c.getString(0) + "\n" +
            "ime: " + c.getString(1) + "\n" +
            "Email:  " + c.getString(2),
            Toast.LENGTH_LONG).show();
}
```

Slika-6 Metoda DisplayContact()

Sledećom slikom prikazan je kod koji je neophodno ugraditi u *onCreate()* metodu klase aktivnosti aplikacije i rezultati učitavanja baze na pokrenutom programu.




```
//--preuzimanje svih kontakata---
db.open();
Cursor c = db.getAllContacts();
if (c.moveToFirst())
{
    do {
        DisplayContact(c);
    } while (c.moveToNext());
}
db.close();


//---preuzimanje jednog kontakta---
db.open();
Cursor d = db.getContact(2);
if (d.moveToFirst())
    DisplayContact(d);
else
    Toast.makeText(this, "Nije pronađen kontakt",
            Toast.LENGTH_LONG).show();
db.close();
```

Slika-7 Učitavanje kontakata iz MyDB

CRUD OPERACIJE – AŽURIRANJE KONTAKATA

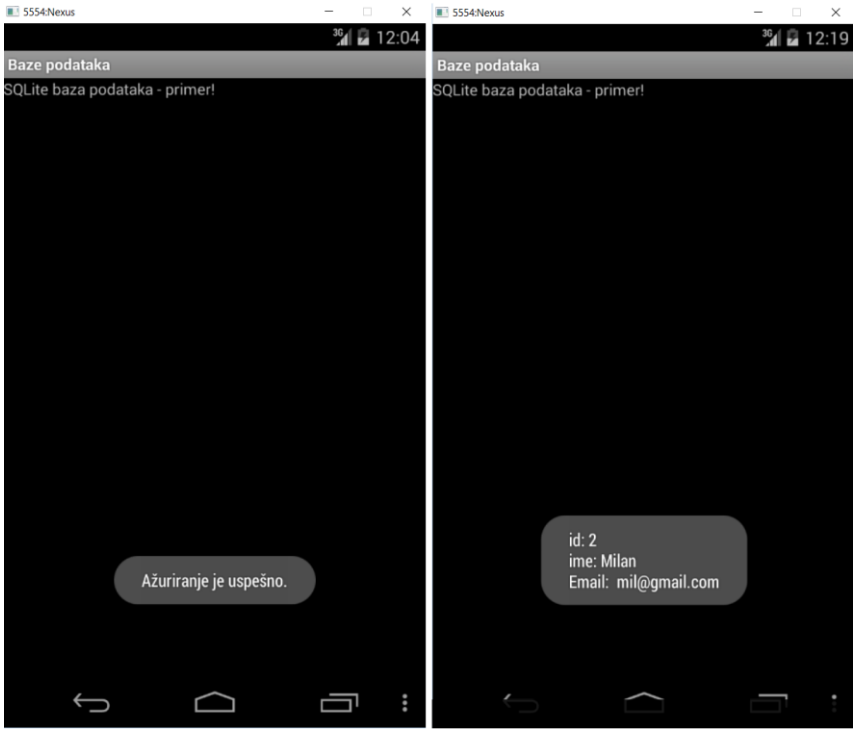
Ažuriranje kontakata se obavlja metodom updateContact() DBAdapter klase.

Da bi podatak, u konkretnom slučaju kontakt, bio ažuriran, neophodno je izvršiti metodu *updateContact()* klase *DBAdapter*. Ova metoda ažurira detalje koji se odnose na kontakt upotrebom identifikacionog broja koji odgovara kontaktu koji se ažurira. Sledećom slikom prikazan je detalj metode *onCreate()* kojim je omogućeno ažuriranje baze kontakata.

```
//---ažuriranje kontakta---
db.open();
if (db.updateContact(4, "Milan", "mil@gmail.com"))
    Toast.makeText(this, "Ažuriranje je uspešno.",
            Toast.LENGTH_LONG).show();
else
    Toast.makeText(this, "Greška u ažuriranju.",
            Toast.LENGTH_LONG).show();
db.close();
```

Slika-8 Kod primera za ažuriranje baze podataka

Pokretanjem programa, sa ugrađenim priloženim kodom, dobija se informacija o uspešnosti ažuriranja konkretnog kontakta, a potom se izlistava ažurirana lista (sledeća slika).



Slika-9 Ažuriranje baze kontakata

CRUD OPERACIJE – BRISANJE KONTAKATA IZ BAZE PODATAKA

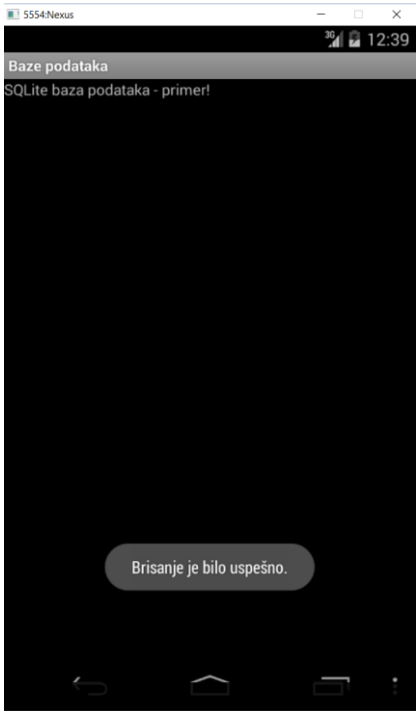
Metoda deleteContact() koristi identifikacioni broj kontakta za njegovo brisanje iz baze podataka.

Za brisanje kontakata, iz kreirane baze podataka, neophodno je koristiti metodu klase *DBAdapter*. Metoda ima naziv *deleteContact()* i koristi identifikacioni broj kontakta za njegovo lociranje i brisanje iz baze podataka. Metoda je logičkog tipa i vraća vrednost *true*, ukoliko je brisanje bilo uspešno, ili *false* ukoliko je iz nekog razloga brisanje ciljanog kontakta izostalo. U metodi *onCreate()*, neophodno je obezbediti sledeći kod kojim se omogućava brisanje kontakata iz baze podataka.

```
//---brisanje kontakta---
db.open();
if (db.deleteContact(1))
    Toast.makeText(this, "Brisanje je bilo uspešno.",
            Toast.LENGTH_LONG).show();
else
    Toast.makeText(this, "Brisanje je bilo neuspešno.",
            Toast.LENGTH_LONG).show();
db.close();
```

Slika-10 Kod za brisanje kontakata iz baze

Ako se pokrene program, sa navedenim modifikacijama u klasi aktivnosti, dobija se informacija o uspešnosti brisanja iz baze (sledeća slika) i prikazuje se nova, ažurirana, lista kontakata.



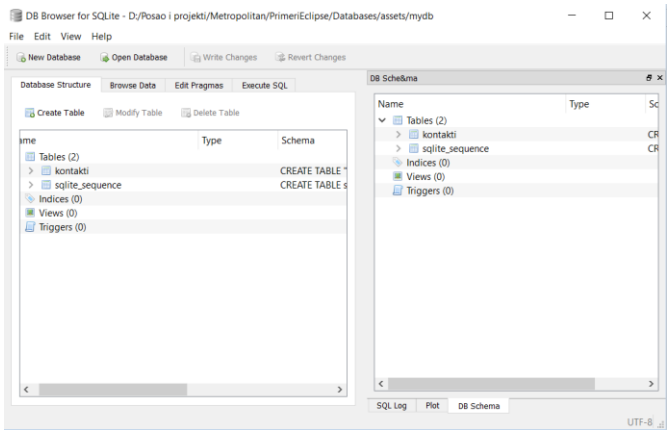
Slika-11 Ishod brisanja kontakta iz baze

SQLITE ALATI

Za rad sa SQLite bazom podataka moguće je koristiti i brojne alate.

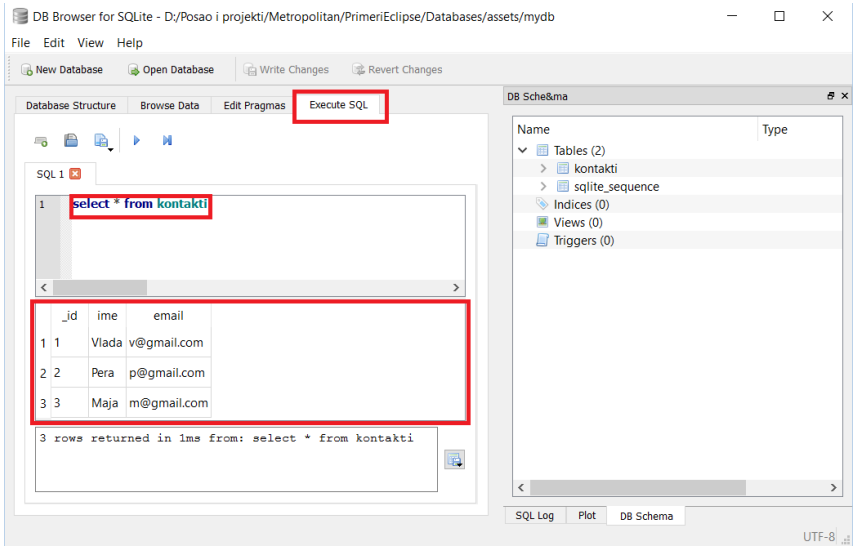
Za kreiranje SQLite baze podataka i za sve poznate operacije nad bazom i njenim tabelama, moguće je koristiti neki od brojnih alata koji su dostupni za preuzimanje preko Interneta. Jedan od njih je *SQLite Database Browser* koji je potpuno besplatan i veoma jednostavan za korišćenje (sledeća slika). Ovaj alat je moguće preuzeti sa lokacije *http://sqlitebrowser.org/*.

SQLite Database Browser omogućava potpuno vizuelno kreiranje baze podataka, upita nad bazom i tabelama za dodavanje novih podataka, ažuriranje starih, brisanje polja, tabela i baze itd.



Slika-14 SQLite Database Browser

Na veoma jednostavan način moguće je kreirati i izvršiti željeni upit primenom ovog alata, kao što je prikazano sledećom slikom.



Slika-15 SQLite Database Browser - primena