X - Rad sa servisima i podacima

S A D R Ž A J

10.1 – Rad sa servisima

- Servis je komponenta aplikacije koja izvršava dugotrajne operacije u pozadini i nema korisnički interfejs.
- Druga komponenta aplikacije može startovati servis, a on će nastaviti da se izvršava u pozadini iako korisnik pređe na neku drugu aplikaciju.
- Komponenta se može povezati sa servisom i interagovati sa njim.
- Servis može upravljati transakcijama preko mreže, puštati muziku, realizovati I/O operacije ili komunicirati sa ***content*** provajderom, itd.
- Sve to servis radi u pozadini.
- Servis se pojavljuje u dve forme:

1. **Pokrenuti servis(Started)** – kada ga komponenta aplikacije pozivom ***metodestartService()*** pokrene. Tada servis može da se izvršava u pozadini neograničeno dugo, bez obzira na status komponente. Na primer, može ***download***-ovati ili ***upload***-ovati fajl preko mreže.
2. **Povezani (Bound)** – kada se komponenta aplikacije vezuje sa njim pomoću metode ***bindService()***. Tada se servis izvršava dokle god je aktivna komponenta ili komponente koje su sa njim povezane.

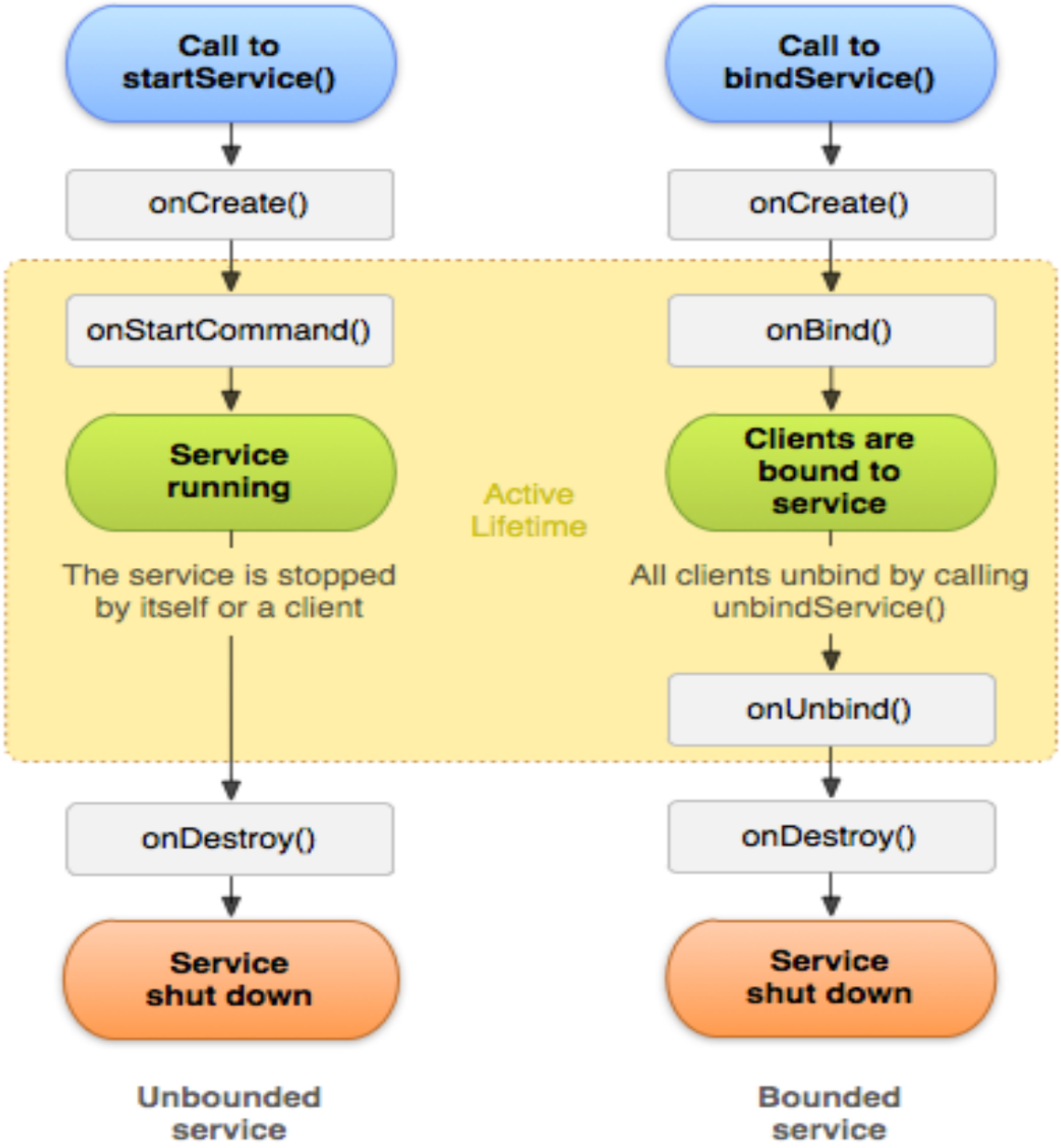
10.1 – Rad sa servisima

- Bez obzira na tip servisa, komponente aplikacije mogu koristiti servis isto kao i aktivnost – startujući ga pomoću **intent**-a.
- Servis u okviru Android Software Development Kit (SDK) može da znači **dve stvari**:
  1. Servis može da znači proces u pozadini, koji obavlja neke korisne informacije u redovnim intervalima.
  2. Servis može biti interfejs za udaljeni objekat koji se zove u okviru vaše aplikacije.
- U oba slučaja servis objekat proširuje klasu ***Service*** u okviru Andoird SDK, i to može da bude samostalna komponenta ili deo aplikacije sa kompletnim korisničkim interfejsom.
- Ovaj servis Andoird-a može se koristiti u sledećim situacijama:
  - ✓ Kada aplikacija obavlja neke duge i zahtevne procese, a da ne postoji potreba da korisnik pokrene pomenuti proces,
  - ✓ Aplikacija mora da obavlja neke rutinske operacije ili operacije u određenim redovnim vremenskim intervalima,
  - ✓ Aplikacija treba da aktivira unapred poznate procese.

10.2 – Formiranje servisa

- Pre nego što se krene u detaljno objašnjenje kako kreirati servis, mora se prvo razumeti na koji način servis ima interakciju sa Andorid OS.
- Prilikom kreiranja servisa, taj servis mora prvo da se registruje u **manifest** fajlu koji aplikacija koristi u okviru taga **<service>**.
- U okviru implementacije servisa moraju se definisati dozvole potrebne za **pokretanje**, **zaustavljanje** i **povezivanje** na servis, kao i neophodni poziv servisa.
- Nakon što je servis implementaran, servis se može koristiti korišćenjem metode ***Context.startService()***.
- Ako je usluga servisa već pokrenuta, kasnijem korišćenjem metode ***startService()***, usluga se ne pokreće ponovo.
- Servis radi sve dok se ne pozove metoda ***Context.stopService()*** ili ne završi sa radom i sam pozove funkciju za prestanak rada ***stopSelf()***.
- Aplikacije koje žele da koriste servis moraju da pozovu funkciju ***Context.bindService()*** za uspostavljanje veze sa tim servisom.
- Ako servis nije pokrenut, on se pokreće u tom trenutku
- Ako je servis već pokrenut, aplikacije mogu da šalju zahteve za izvršavanje određenih usluga ako za to imaju dozvolu.

10.2 – Životni ciklus servisa



10.2 – Formiranje servisa

- Kreiranje servisa i njegov rad biće prikazan na primeru servisa koji reaguje na pomeranje korisnika, tj. promenu njegovih koordinata (***latitude*** i ***longitude***).
- Kreirajmo običnu aplikaciju sa jednom aktivnošću - **ServisKontroler**.
- Ova aktivnost će se koristiti da pokrene/ugasi naš servis, pa ćemo u skladu sa time aktivnosti dodati dva dugmeta u ***layout*** fajlu (**main.xml**) a zatim pokupiti reference na njih u kodu.

```
<?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:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:id="@+id/dugmeStart" android:text="START"/>
   <Button android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:id="@+id/dugmeStop" android:text="STOP"/>
</LinearLayout>
```

10.2 – Formiranje servisa

- Kod aktivnosti (*ServisKontroler.java*):

```
public class ServisKontroler extends Activity {
Button dugmeStart, dugmeStop;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  dugmeStart = (Button)findViewById(R.id.dugmeStart);
  dugmeStop = (Button)findViewById(R.id.dugmeStop);
}
```

- Ovim dugmićima ćemo kasnije dodati listenere za pokretanje i zaustavljanje aktivnosti.

10.2 – Formiranje servisa

- Da bi kreirali novi servis, treba da nasledimo klasu ***Service*** i implementiramo njene metode – ***onCreate*** (kod kreiranja servisa), ***onStart/onStartCommand*** (kod poziva) i ***onDestroy*** (kod završetka).
- ***onStartCommand*** je novija verzija metode ***onStart*** koju treba izbegavati ako želimo da nam aplikacija bude kompatibilna i sa starijim verzijama Android sistema.
- Za početak napravićemo samo kostur servisa **MojServis.java**:

```
public class MojServis extends Service {
    private LocationManager lokacija = null;
    private NotificationManager notifikator = null;
    @Override
    public void onCreate() {
        super.onCreate();
    }
    @Override
    public void onStartCommand(Intent intent, int flags, int startId) {
        super.onStartCommand(intent, startId);
    }
     @Override
    public void onDestroy() {
        super.onDestroy();
    }
}
```

10.2 – Formiranje servisa

- Potrebno je u manifestu naglasiti da će naša aplikacija koristiti servis (**MojServis**) i da će trebati dozvola za pristupanje lokaciji korisnika

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

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

 <application>

...

<service android:name="MojServis" ></service>

...

</application>
```

- Da bismo povezali aktivnost i servis, tj. da bismo startovali/gasili aktivnost iz servisa, treba samo da pozovemo metode ***startService*** ili ***stopService***, što se radi kada korisnik pritisne odgovajuće dugme
- Na sledećem slajdu prikazan je odgovarajući kod

10.2 - Formiranje servisa

```
dugmeStart.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
        Intent namera = new Intent(ServisKontroler.this, MojServis.class);
        startService(namera);
        }
});
 dugmeStop.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
        Intent namera = new Intent(ServisKontroler.this,MojServis.class);
        stopService(namera);
        }
});
```

10.2 – Formiranje servisa

- Sada ćemo preći na pisanje konkretnog koda u servisu.
- Prvo ćemo implementiramo metodu ***onCreate***:

```
public void onCreate() {
    super.onCreate();
    lokacija = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
    notifikator = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
}
```

- Pri kreiranju servisa treba se referencirati na **LocationManager** koji ćemo koristiti za određivanje lokacije i **NotificationManager**-a koji ćemo koristiti da prikažemo notifikacije kada se korisnik pomeri.
- Prilikom razvoja servisa obavezno se mora testirati aplikacija da li je kompatabilna sa raznim verzijama Andoird SDK platforme.
- Da bi sve bilo kompatabilno aplikacija mora biti urađena u skladu sa SDK 5 platforme.
- Ponašanje servisa definišu ***onStart()*** i/ili ***onStartCommand()*** metode.
- Implementiraćemo metodu ***onStartCommand*** koja se poziva po startovanju servisa:

10.2 – Formiranje servisa

```
@Override
public int onStartCommand(Intent intent, int flags, int startId ) {
   Criteria kriterijum = new Criteria();
   kriterijum.setAccuracy(Criteria.NO_REQUIREMENT);
   kriterijum.setPowerRequirement(Criteria.POWER_LOW);
 // dobili provajder lokacije koji najviše odgovara traženim kriterijumima
   String best = lokacija.getBestProvider(kriterijum, true);
// zahtevamo periodično informisanje o promeni lokacije
   lokacija.requestLocationUpdates(best, 60000, 5, this);
// šaljemo notifikaciju da je započeto praćenje lokacije
   Notification obavestenje = new
   Notification(android.R.drawable.stat_notify_more,
                        "GPS pracenje", System.currentTimeMillis());
   obavestenje.flags |= Notification.FLAG_AUTO_CANCEL;
   Intent toLaunch = new Intent(getApplicationContext(),ServisKontroler.class);
   toLaunch.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
   PendingIntent intentBack =
   PendingIntent.getActivity(getApplicationContext(), 0, toLaunch, 0);
   obavestenje.setLatestEventInfo(getApplicationContext(),
   "GPS pracenje", "Pracenje je zapocelo ", intentBack);
   notifikacije.notify(1, obavestenje);                    // 1 - ID notifikacije
}
```

10.2 – Formiranje servisa

- Prvo kreiramo objekat klase **Criteria**.
- Telefon može imati više načina da odredi svoju lokaciju (pomoću mobilne mreže, GPS-a, Interneta i dr.), koristimo objekte ove klase da specificiramo kakav mehanizam za određivanje lokacije želimo.
- Ovde je specificiran mehanizam sa malim utroškom baterije, a bez ikakvih posebnih zahteva po pitanju preciznosti.
- Nakon što smo to podesili u **Criteria** objektu, predajemo objekat **LocationManager** metodi **getBestProvider** a kao rezultat dobijamo ime mehanizma koji najbolje odgovara onome što smo tražili.
- Da bi bili informisani o promeni lokacije, pozivamo metodu ***requestLocationUpdates()*** **LocationManager**-a i kao parametre joj dajemo ime željenog provajdera, minimalno vreme između dva obaveštenja, minimalnu razdaljenost koja treba da bude pređena da bi bila registrovana i referencu na klasu koja će biti periodično pozivana.
- Pošto smo tu prosledili "**this**" kao parametar, moraćemo kasnije da implementiramo ***LocationListener*** interfejs u našem servisu i njegovu metodu ***onLocationChanged***.

10.2 - Formiranje servisa

- Preostale linije koda se odnose na sastavljanje i slanje notifikacije
- Dalje ćemo pogledati metodu ***onLocationChanged*** koju treba implementirati:

```
@Override
public void onLocationChanged(Location location) {
  Notification obavestenje = new
Notification(android.R.drawable.stat_notify_more,
    "GPS pracenje", System.currentTimeMillis());
   obavestenje.flags |= Notification.FLAG_AUTO_CANCEL;
   Intent toLaunch = new Intent(getApplicationContext(),
ServisKontroler.class);
  toLaunch.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
   PendingIntent intentBack =
PendingIntent.getActivity(getApplicationContext(),  0, toLaunch, 0);
   // Saljemo notifikaciju sa vrednostima latitude i longitude
  obavestenje.setLatestEventInfo(getApplicationContext(), "GPS
pracenje", "Lokacija: " +
    location.getLatitude() + "," + location.getLongitude(), intentBack);
    notifikacije.notify(1, obavestenje);
}
```

10.2 – Formiranje servisa

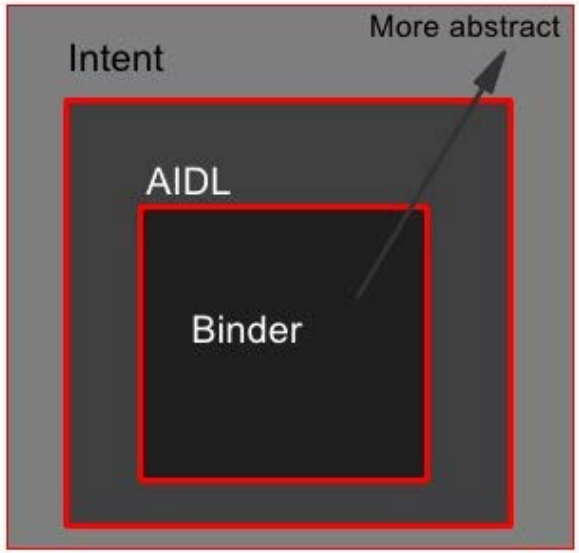
- Ovde ponavljamo proceduru sa slanjem notifikacije, gde stavljamo vrednosti ***latitude*** i ***longitude*** (parametri ***onLocationChanged*** metode)
- Potrebno je još očistiti servis nakon korišććenja i to na sledeći način:

```
@Override
public void onDestroy() {
        // odjavljujemo se od prijema obavestenja o promeni lokacije
        if (lokacija != null) {
                lokacija.removeUpdates(this);
                lokacija = null;
        }
        // saljemo notifikaciju da je servis ugasen
        Notification obavestenje = new
        Notification(android.R.drawable.stat_notify_more,
        "GPS pracenje", System.currentTimeMillis());
        obavestenje.flags |= Notification.FLAG_AUTO_CANCEL;
        Intent toLaunch = new Intent(getApplicationContext(),
        ServisKontroler.class);
        PendingIntent intentBack =
        PendingIntent.getActivity(getApplicationContext(), 0, toLaunch, 0);
        obavestenje.setLatestEventInfo(getApplicationContext(),
              "GPS pracenje", "Pracenje je stopirano!", intentBack);
        notifikacije.notify(1, obavestenje);
        super.onDestroy();
}
```

- Na kraju odjavljujemo servis od primanja obaveštenja o promeni lokacije i šaljemo notifikaciju o tome.

10.3 – Implementacija udaljenog servisa

- U većini slučajeva postoji potreba da se ima veća kontrola nad sistemom i servisima, koji servis je pokrenut ili stopiran.
- Andoird sadrži alate i formate za kreiranje ovog korisničkog interfejsa
- Da bi se daljinski interfejs definisao, potrebno je da se interfejs objavi u **AIDL** (***Android Interface Definition Language***) fajlu
- Nakon toga treba implementirati interfejs, a zatim vratiti instancu interfejsa kada se poziva ***onBind()*** metoda.
- Inter-procesna komunikacija u ***Binder*** okviru je implementirana kao odnos klijent-servis i postoje **tri mehanizma** implementacije:
  1. ***Intent*** – nezavistan od aplikacije u kojoj se primenjuje,
  2. ***Messenger*** - između Intenta i ***AIDL***
  3. ***AIDL -*** baziran na jeziku za definisanje Andorid interfejsa.
- ***Messenger*** i ***AIDL*** implementacije rade sa vezanim (*bound*) servisima, dok ***Intent*** objekti rade sa pokrenutim (*started*) servisima.



10.3 – Implementacija udaljenog servisa

- ***Messenger*** mehanizam IPC predstavlja srednji nivo apstrakcije u okviru ***Binder*** okvira.
- On predstavlja kompromis između ***Intent*** i ***AIDL*** mehanizma.
- **Dobre strane** su da ne traži toliko detaljnu implementaciju kao što definisanje interfejsa kod ***AIDL*** mehanizma i malo je efikasniji od ***Intent*** mehanizma kad se traži brži odziv.
- **Loše strane** su da nije tako jednostavan kao IPC mehanizam baziran na ***Intent*** objekatima i nije toliko brz kao ***AIDL*** implementacija.
- Service koristi ***AIDL*** interfejs da bi izložio svoje mogućnosti (metode) klijentu, koje on kasnije koristi kako bi komunicirao direktno sa njim.
- Međuprocesna komunikacija bazirna na ***AIDL*** daje daleko bolje performanse od prethodno dva gore navedena mehanizma za IPC.
- ***AIDL*** se koristi onda kada je učestala razmena poruka između procesa
- ***AIDL*** sintaksa je veoma slična sintaksi Java interfejsa.
- ***AIDL*** definiše metode koje je implementirao servis i koje klijent može jednostavno da koristi direktnim pozivanjem kao i kod drugih objekta
- Na ovaj način se podiže nivo apstrakcije IPC koji je u duhu objektno-orjentisanog programiranja.

10.3 – Implementacija udaljenog servisa

- U nastavku sledi kod koji je smešten u ***AIDL*** fajlu za interfejs:

```
interface IRemoteInterface {
    Location getLastLocation();
}
```

- AIDL fajl se veoma lako i jednostavno definiše na sledeći način:

```
private final IRemoteInterface.Stub
    mRemoteInterfaceBinder = new IRemoteInterface.Stub() {
        public Location getLastLocation() {
            Log.v("interface", "getLastLocation() je pozvana");
            return lastLocation;
        }
};
@Override
public IBinder onBind(Intent intent) {
//svi servisi imaju sam jedan interfjs, konkretno u ovom slucaju tako da
//nije potrebno nikakvo dodatno navođenje i proveravanje, vec samo
//vratimo instancu
    return mRemoteInterfaceBinder;
}
```

10.3 – Formiranje servisa

- U okviru **Android Manifest.xml** fajlu mora se dodati sledeći kod kako bi se omogućio rad sa interfejsima:

```
<action android:name =“com.androidbook.services.IRemoteInterface” />
```

- Nakon podešavanja u manifest fajlu servis se može koristiti, potrebno je još samo definisati glavne metode za konekciju i diskonekcija sa servisa.

```
public void onServiceConnected(ComponentName name, IBinder service)
{
   mRemoteInterface = IRemoteInterface.Stub.asInterface(service);
   Log.v(“ServiceControl”, “Interfejs se povezao.”);
}
public void onServiceDisconnected(ComponentName name) {
   mRemoteInterface = null;
   Log.v(“ServiceControl”, “Udaljeni interfejs se diskonektovao!”);
}
```

- Nakon poziva metode ***onServiceConnected()*** interfjs je spreman za korišćenje i potom se može koristiti. Kod za korišćenje izgleda ovako:

```
Location lokacija = mRemoteInterface.getLastLocation();
```

10.4 - Skladištenje podataka

- Android nudi nekoliko opcija za skladištenje podataka.
- Koji način će se koristiti zavisi od specifičnih potreba, da li su podaci dostupni drugim aplikacijama ili su privatni, kao i koliko prostora zauzimaju ti podaci.
- Andoird nudi sledeće opcije za skladištenje podataka:

1. **Zajednička podešavanja (*SharedPreferences*)** - objekat za čuvanje jednostavnih, primitivnih podataka u obliku para **ključ-vrednost**
2. **Inerne memorije** – čuvanje podataka u memoriji uređaja
3. **Eksterne memorije** – spoljne memorije (CD, USB, MicroSD)
4. **SQLite baze podataka** – lokalno čuvanje podataka u privatnoj bazi podataka
5. ***Network Connection*** – čuvanje podataka na nekom od web servera

- Andoird nudi mogućnost deljenja i privatnih podataka sa drugim aplikacijama uz korišćenje preko provajdera sadržaja.

10.4 – Zajednička podešavanja

- Klasa **SharedPreferences** daje opšti okvir za čuvanje primitivnih tipova podataka koji su upareni ključ-vrednost.
- Ova klasa se može koristiti za čuvanje podataka koji su tipa: boolean, floats, int, long, i string.
- Ovi podaci se pre svega koriste i prožimaju kroz različite sesije.
- Da bi kreirali objekat **SharedPreferences** koristi se jedan od dole dva navedena načina:

1. ***getSharedPreferences()*** – ova opcija se koristi ako je potrebno izvršiti više preferencija za datoteku koje su određene prvim parametrom koji predstavlja identifikator za sva preferenciranja
2. ***getPreferences()*** – opcija se koristi ako je potrebno dobiti samo jednu preferenciju datoteke za željenu aktivnost

- U nastavku sledi primer koda kako se čuvaju podaci na ovaj način.
- Prikazan je primer kako se čuva vrednost koja je uneta prilikom izračunavanja kod kalkulatora.

10.4 - Memorisanje podataka

```
public class Kalkulator extends Activity {
  public static final String PREFS_NAME = "mojPref";
  @Override
  protected void onCreate(Bundle stanje){
    super.onCreate(stanje);
    . . .
    // vrati preference
    SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       boolean silent = settings.getBoolean("silentMode", false);
       setSilent(silent);
  }
  @Override
  protected void onStop(){
  super.onStop();
  // svi objekti su sa android.context.Context
  SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
  SharedPreferences.Editor editor = settings.edit();
  editor.putBoolean("silentMode", mSilentMode);
  // potvrdimo promene
  editor.commit();
  }
}
```

10.4 – Primer

```
// Uzimanje vrednosti deljene preference
SharedPreferences app_preferences =
PreferenceManager.getDefaultSharedPreferences(this);

// Čitanje vrednost tekućeg brojača
int counter = app_preferences.getInt("counter", 0);

// Ažurirane TextView
TextView text = (TextView) findViewByld(R.id.text);
text.setText("This app has been started " + counter + " times.");

// Povećajte vrednost brojača

SharedPreferences.Editor editor = app_preferences.edit();
editor.putInt("counter", ++counter);
editor.commit(); // Very important
```

10.5 - Interne memorije

- Interne memorije omogućavaju da se skladište podaci direktno u memoriji uređaja.
- Podrazumevano je da su podaci koji se koriste u memoriji uređaja **privatni** tako da ostale aplikacije tim podacima ne mogu da pristupe.
- Kada se aplikacija ukloni sa uređaja, nestaju i podaci.
- Da bi upisali podatke u internu memoriju koriste se sledeće metode:
  - **openFileOutput()**
  - **write()**
  - **close()**
  - **String NAZIVFAJLA = "moj_prvi_fajl";**
  - **String string = "Dobar dan svima!";**
  - **FileOutputStream fos = openFileOutput(NAZIVFAJLA, Context.MODE_PRIVATE);**
  - **fos.write(string.getBytes());**
  - **fos.close();**
- Argument **MODE_PRIVATE** kreiraće novi fajl ili pak zameniti postojeću datoteku novim ako je isto ime.

10.5 – Interne memorije

- Ostali modovi koji su još dostupni prilikom upisivanja u fajl su:
  - ***MODE_APPEND***
  - ***MODE_WORLD_READABLE***
  - ***MODE_WORLD_WRITEABLE***.
- Podatke iz interne memorije čitamo sa sledećim metodama:
  - ***openFileInput()*** – ovde se definiše ime datoteke koja se čita
  - ***read()*** – čita bajtove iz datoteke
  - ***close()*** – zatvara tok za čitanje.
- Metode koje se još koriste prilikom čitanja i pisanja u datoteku su:
  - ***getFilesDir()*** – vraća absolutnu putanju direktorijuma gde se datoteka čuva u memoriji
  - ***getDir()*** – kreira ili otvara već postojeći direktorijum u okviru unutrašnjeg prostora
  - ***deleteFile()*** – briše datoteke koje se čuvaju u internoj memoriji
  - ***fileList()*** – vraća listu datoteka koje su sačuvane u memoriji

10.6-Skladištenje podataka na spoljašnju memoriju

- Svaki android uređaj podržava deljeno eksterno skladište, koje se može koristiti za čuvanje podataka.
- Ovde spadaju prenosivi mediji za čuvanje podataka (CD,DVD,USB)
- Ovi podaci su kompatabilni i čitljivi, što daje mogućnost da se oni, prilikom povezivnja medija na USB, mogu preneti na neki drugi uređaj
- Prvo je potrebno da se proveri da li je spoljašni mediji dostupan uz pomoć metode ***getExternalStorageState():***

```
boolean mSpoljnaMemorijaAvailable = false;
boolean mSpoljnaMemorijaWriteable = false;
String stanje = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(stanje)) {
    // sada se može čitati i upisivati u memoriju
    mSpoljnaMemorijaAvailable = mSpoljnaMemorijaWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(stanje)) {
    // u ovom slučaju moguće je samo čitati podatke
    mSpoljnaMemorijaAvailable = true;
    mSpoljnaMemorijaWriteable = false;
} else {
    // nastala je neka greška, možda uređaj nije dobro povezan ili nešto slično
    mSpoljnaMemorijaAvailable = mSpoljnaMemorijaWriteable = false;
}
```

10.6-Skladištenje podataka na spoljašnju memoriju

- Da bi se pristuplo datotekama na spoljašnom skladištu ako se koristi **API 8** ili noviji, koristi se metoda ***getExternalFilesDir()***.
- Ova metoda prihvata parametar koji određuje tip poddirektorijuma koji se želi otvoriti, kao što je **DIRECTORY_MUSIC** ili **DIRECTORY_RINGTONES**.
- Ako direktorijum ne postoji, ova metoda će automatski kreirati novi.
- Ako se koristi **API 7** ili stariji, koristi se metoda ***getExternalStorageDirectory()***, da bi se otvorila datoteka koja predstavlja root eksterne memorije.
- U tom slučaju podaci se čuvaju na sledećoj putanji:

  /Android/data/<package_name>/files/
- Ako se želi sačuvati neka datoteka, koja se neće obrisati sa deinstaliranjem aplikacije, potrebno je tu datoteku sačuvati na javnom delu skladišta.
- Ovi direktorijumi su postavljeni u root-u direktorijumu eksternog skladišta **Music/**, **Pictures/**, **Ringtones/**, i drugi.

10.7 – Upotreba statičkih resursa

- 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 prikazanom primeru, u folderu **res/raw** projekta, ubačena je datoteka sa nazivom ***textfile.txt*** u kojoj je sačuvan string koji odgovara nazivu naše škole.
- 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.
- Na sledećom slajdu je prikazan neophodni programski kod sa klasama koji je neophodno implementirati u ***onCreate()*** metodu.

10.7 - Upotreba statičkih resursa

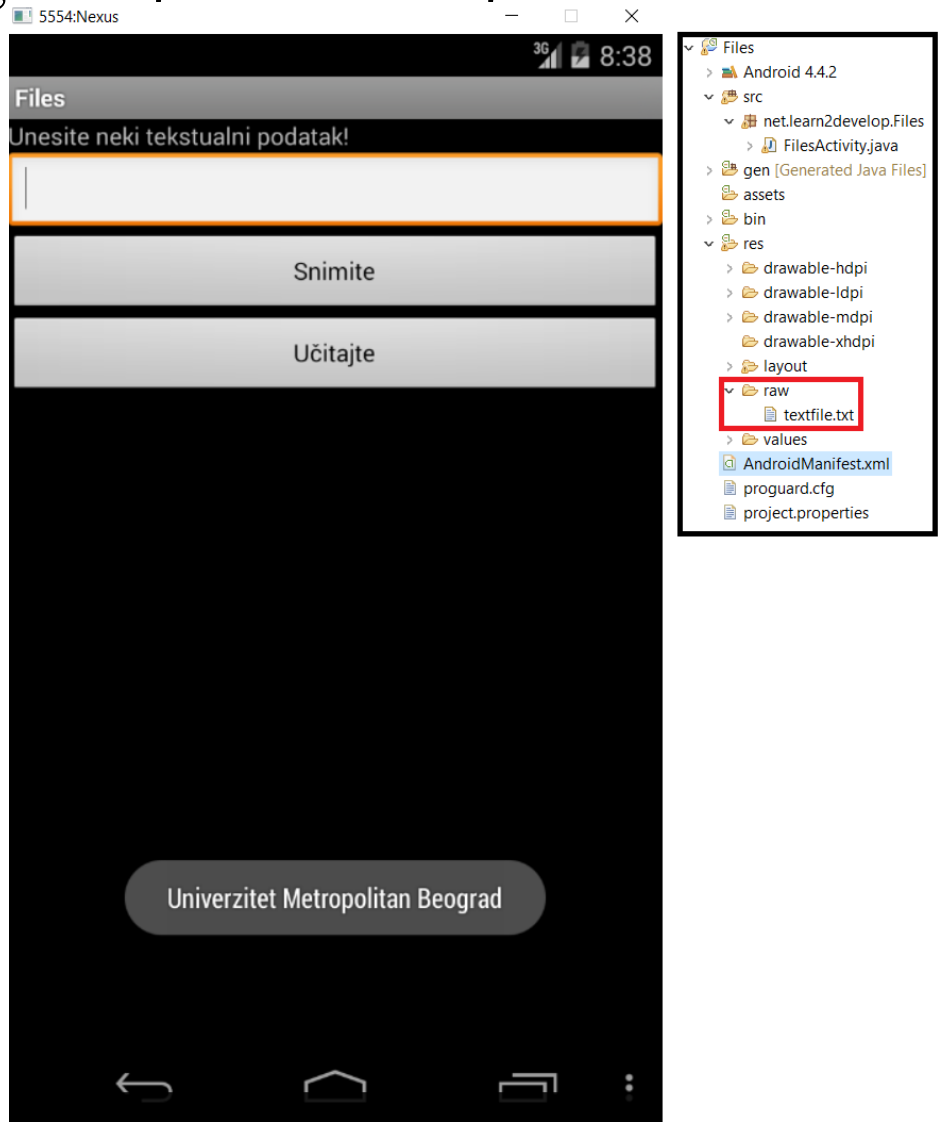
```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
```

```
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();
}
```

10.7 - Upotreba statičkih resursa

- Identifikator resursa, koji je smešten u **res/raw** folderu, dobija naziv na osnovu naziva datoteke bez odgovarajuće ekstenzije.
- U konkretnom slučaju to je ***R.raw.textfile***.
- Na sledećoj slici pokazana je aplikacija koja preuzima podatke iz statičke datoteke kao i njena lokacija u hijerarhiji projekta.



10.8 – Kontrola baza podataka

- Do sada smo se upoznali sa tehnikama za čuvanje skupova podataka.
- Kada čuvamo podatke, koji su različitog tipa, koji se tabelarno prikazuju i povezani su relacijama, koriste 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 jer nam pruža mnogo veće mogućnosti
- 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.

10.8 – Kontrola baza podataka

- Prikazaćemo način kreiranja ***SQLite*** baze podataka u Android aplikac.
- Kreirana baza podataka, u Android-u, **uvek se nalazi**, za datu aplikaciju, u folderu ***data/data/nazivPaketa/databases***.
- Dobra praksa, u radu sa Android bazama, je kreiranje pomoćne klase koja enkapsulira veoma složen postupak pristupa podacima.
- Iz navedenog razloga, biće kreirana klasa ***DBAdapter*** koja će omogućiti kreiranje, zatvaranje i otvaranje baze, učitavanje i upisivanje podataka, a biće implementirani i određeni upiti za izvršavanje određenih akcija nad bazom podataka.
- Dat je primer koji podrazumeva kreiranje baze podataka sa jednom tabelom pod nazivom ***kontakti***.
- Tabela će biti izgrađena od tri kolone: ***_id***, ***ime*** i ***email*** (na slici).
- Nazvaćemo bazu podataka ***MyDB***.

_id	ime	email

- Prvi zadatak je kreiranje JAVA koda pomoćne klase ***DBAdapter.java***.
- Lokacija klase biće standardi folder u okviru projekta ***src***.

10.8 - Kontrola baza podataka

```
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;
}
}
```

10.7-Izbor optimalnog načina skladištenja podataka

- Obrađeno je čuvanje podataka u Android aplikacijama na četri načina.
- Korišćene su deljene preferencije, unutrašnja i spoljašnja memorijska skladišta Android uređaja kao i baze podataka.
- Da bi bio izabran pravi način čuvanja podataka, u Android aplikacijama, neophodno je poštovati određene preporuke izbora:

1. Ukoliko se manipuliše podacima koji mogu da se prikažu u obliku para (**naziv, vrednost**) biće korišćene **deljene preferencije**.
   Primer: *deljenjim preferencijama moguće je sačuvati sledeće parove vrednosti: (korisnik, datum rođenja), (korisnik, vreme prijavljivanja na sistem), (pozadina, boja pozadine), (zvono, melodija zvona) itd.*
2. 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
3. Kada je neophodno 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.
4. U slučaju veće količine različitih podataka **baze podataka** su rešenje

Hvala na pažnji !!!

Pitanja

? ? ?