PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET METROPOLITAN BEOGRAD

KI205 - JAVA 8: PROGRAMIRANJE U JAVI NA ANDROID PLATFORMI

Razvoj Android servisa

Lekcija 10

PRIRUČNIK ZA STUDENTE

KI205 - JAVA 8: PROGRAMIRANJE U JAVI NA ANDROID PLATFORMI

Lekcija 10

RAZVOJ ANDROID SERVISA

UVOD

Aplikacija, koju Android izvršava u pozadini i bez potrebe za interakcijom sa korisnikom, naziva se servis.

Veoma često, u programerskoj stručnoj literaturi, sreće se termin servis. U Android kontekstu, pod servisom se podrazumeva aplikacija koja ne zahteva interakciju sa korisnikom i koju operativni sistem izvršava u pozadini. Kao primer moguće je navesti sledeći scenario: dok programer radi na razvoju nekog softverskog rešenja može u pozadini slušati neku muziku. Slušanje muzike omogućeno je programskim kodom koji omogućava reprodukovanje muzike u pozadini i ne zahteva interakciju sa korisnikom. Na ovaj način realizovan je specifičan Android servis.

Servisi su posebno značajni jer omogućavaju razvoj softverskih rešenja kod koji nije potrebno da se korisniku pokazuje korisnički interfejs. Na primer, razvoj aplikacije koja će stalno biti aktivna u pozadini i pratiti geografsku lokaciju mobilnog uređaja na kojem će biti instalirana.

Cilj ove lekcije jeste učenje kreiranja vlastitih Android servisa i načina njihovog korišćenja prilikom asinhronog izvršavanja u pozadini.

Lekcija će staviti poseban akcenat na sledeće teze:

- Kreiranje servisa koji se izvršava u pozadini;
- Izvršavanje dugotrajnog zadatka u posebnoj niti;
- Izvršavanje zadataka koji se u servisu ponavljaju;
- Ostvarivanje komunikacije između aktivnosti i servisa.

Savladavanjem ove lekcije, studenti će biti u potpunosti obučeni da kreiraju Android servise i da primenjuju asinhrone načine njihovog izvršavanja u pozadinskoj aktivnosti aplikacije.

Poglavlje 1

Kreiranje vlastitih servisa

PRIMER 1 - PRIMER SERVISA

U ovom delu lekcije biće prikazani koraci za kreiranje jednostavnog servisa.

Najbolji način za razumevanje servisa jeste njihovo kreiranje. Poseban zadatak, ovog dela lekcije, biće prikazivanje koraka za kreiranje jednostavnog Android servisa koji će u nastavku lekcije biti unapređivan dodatnim funkcionalnostima. Za početak biće dovoljno da se nauči kao se servis pokreće i zaustavlja.

U tu svrhu, koristeći Android Studio IDE, biće kreiran primer pod nazivom MojServisDemo za demonstraciju implementacije jednostavnog Android servisa. Za početak, biće kreirana jednostavna klasa servisa, koja će dobiti naziv MojServis, koja nasleđuje osnovnu klasu Service.

Za navedenu klasu kreiran je inicijalno kod prikazan narednim listingom. Kako se budu javljali novi zahtevi, tako će i ovaj kod biti proširivan.

```
package com.metropolitan.mojservisdemo;

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

/**
 * Created by Vladimir Milicevic on 27.11.2016..
 */

public class MojServis extends Service {
    @Override
    public IBinder onBind(Intent arg0) {
        return null;

    }
    @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
        // Želimo da servis bude aktivan dok se eksplicitno ne zaustavi
       // zato vraća vrednost sticky.
       Toast.makeText(this, "Servis je pokrenut", Toast.LENGTH_LONG).show();
       return START_STICKY;
    }
```

```
    @Override
    public void onDestroy() {
        super.onDestroy();

        Toast.makeText(this, "Servis je zaustavljen",
                Toast.LENGTH_LONG).show();
    }
}
```

PRIMER SERVISA – XML DATOTEKE

Biće implementirane kontrole za pokretanje i zaustavljanje servisa.

Prilikom implementacije Android servisa, veliku ulogu ima datoteka AndroidManifest.xml. Za implementaciju jednostavnog Android servisa biće neophodno uključiti kod u AndroidManifest.xml datoteku koji je prikazan sledećim listingom kao sadržaj XML elementa *<service .../>.*

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

    <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>

        <service android:name=".MojServis">
            <intent-filter>
                <action android:name="com.metropolitan.mojservisdemo.MojServis" />
            </intent-filter>
        </service>

    </application>

</manifest>
```

Nezamislivo je pokretanje servisa iz glavne aktivnosti bez prethodne interakcije korisnika sa nekom kontrolom iz korisničkog interfejsa. Tako će biti i u ovom konkretnom slučaju. Pokretanje i zaustavljanje Android servisa biće prepušteno kontrolama tipa dugmad. Kontrole

su kreiranje u activity_main.xml datoteci. Kod osnovne GUI datoteke, priložen je sledećim listingom.

```
<?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/btnStartService"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Pokreni servis"
    android:onClick="startService"/>

<Button android:id="@+id/btnStopService"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Zaustavi servis"
    android:onClick="stopService" />

</LinearLayout>
```

PRIMER SERVISA – KLASA AKTIVNOSTI

Pokretanje i zaustavljanje servisa, u glavnoj aktivnosti aplikacije, prepušteno je klasi aktivnosti.

Da bi servis bio pokrenut, neophodno je imati glavnu aktivnost koja će biti učitana u ekran mobilnog uređaja i iz koje će servis biti pokrenut da se izvršava u pozadini.

U folderu *src* projekta biće kreirana klasa aktivnosti pod podrazumevanim nazivom *MainActivity.java*. Ovom klasom biće omogućeno pokretanje i zaustavljanje servisa iz glavne aktivnosti. Za početak klasa će biti veoma jednostavna, a potom kako rastu zahtevi, njen kod će biti proširivan i dopunjavan. Inicijalni kod klase aktivnosti, priložen je sledećim listingom.

```
package com.metropolitan.mojservisdemo;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
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) {
        startService(new Intent(getBaseContext(), MojServis.class));
       }

    public void stopService(View view) {
        stopService(new Intent(getBaseContext(), MojServis.class));
    }

}
```

POKRETANJE I ZAUSTAVLJANJE SERVISA - FUNKCIONISANJE

Cilj je prikazivanje korak-po-korak kako jednostavan servis funckioniše.

Ovaj primer prikazuje razvoj jednog jednostavnog servisa. Ovaj servis, iako ne radi ništa korisno, služi kao dobra osnova za razumevanje procesa kreiranja servisa.

Prvi korak koji se preduzima jeste kreiranje klase koja nasleđuje osnovnu klasu Services. Navedena klasa je dobila naziv MojServis i ona implementira tri metode (videti priloženi kod):

- onBind() – omogućava povezivanje aktivnosti sa servisom. Obezbeđeno je da aktivnost direktno pristupa članovima i metodama servisa. Za početak, metoda vraća ***null***, a u daljem izlaganju biće prikazano više o povezivanju aktivnosti i servisa.
- onStartCommand() – izvršava se prilikom eksplicitnog pokretanja servisa pomoću startService() metode klase aktivnosti (videti priloženi kod). Metoda označava početak servisa, a programer će napisati kod kojim je definisano šta se dešava tokom aktivnosti servisa. U ovom slučaju metoda vraća konstantu ***START_STICKY*** koja ukazuje da će servis biti aktivan sve dok se eksplicitno ne zaustavi.
- onDestroy() – izvršava se kada je servis zaustavljen pomoću metode stopService() klase aktivnosti (videti priloženi kod).

Izvršavanjem metode onDestroy() slobađaju se svi resursi koje je servis zauzeo.

Svi kreirani servisi moraju jasno biti deklarisani u AndridManifest datoteci na sledeći način:

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

Ukoliko se želi da servis bude dostupan drugim aplikacijama, neophodno je definisati *Intent* filter sa nazivom akcije (videti AndroidManifest priloženi kod).

Da bi servis bio pokrenut, neophodno je iz klase aktivnosti pokrenuti metodu startService(), i to:

1. kada se metoda poziva iz matične klase:

startService(new Intent(getBaseContext(), MyService.class));

2. kada se metoda poziva iz eksterne aplikacije:

startService(new Intent("net.learn2develop.MyService"));

Na kraju, servis se zaustavlja pozivom metode stopService():

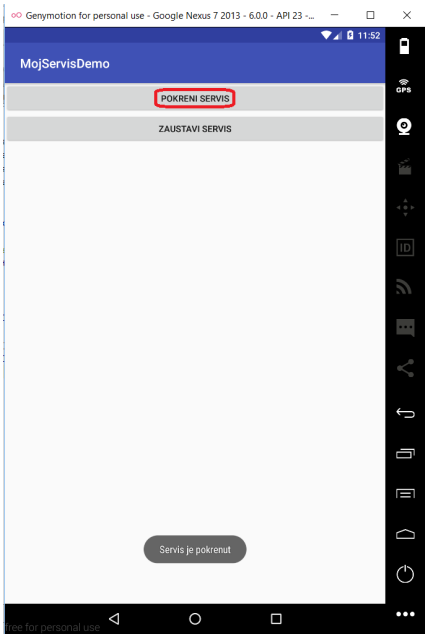
stopService(new Intent(getBaseContext(), MyService.class));

POKRETANJE I ZAUSTAVLJANJE SERVISA - DEMONSTRACIJA

Klikom na prvo dugme, servis se pokreće, a klikom na drugo zaustavlja.

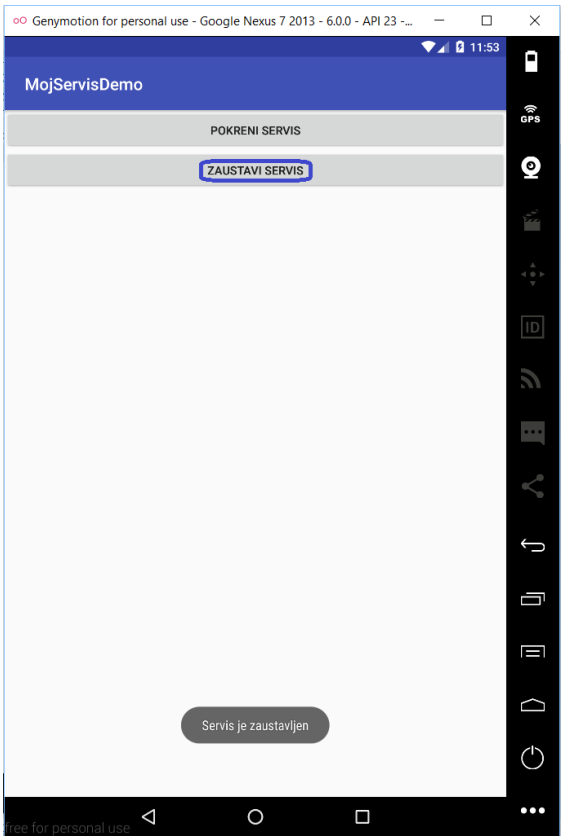
Pošto je kreiran najjednostavniji mogući servis, moguće je izvršiti testiranje aplikacije. U Android Studio IDE razvojnom okruženju, izborom opcije Run App ili Shift + F10, aplikacija se prevodi i testira u izabranom AVD ili realnom mobilnom uređaju.

Ukoliko je program uspešno preveden, sledećom slikom je prikazano pokretanje servisa.



Slika 1.1 Servis je pokrenut

Sledećom slikom je prikazano zaustavljanje servisa, nakon klika na odgovarajuće dugme u korisničkom interfejsu glavne aktivnosti aplikacije.



Slika 1.2 Servis je zaustavljen

Sledećim video materijalom je prikazano kreiranje jednostavnog Android servisa.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK 1

Pokušajte sami!!!

Vodeći se instrukcijama, iz ovog dela lekcije, pokušajte da kreirate samostalno vaš prvi Android servis.

Poglavlje 2

Izvršavanje dugotrajnih zadataka

PRIMER 2 - DUGOTRAJNI ZADACI U SERVISU

Cilj je simuliranje servisa za preuzimanja podataka sa Interneta.

U prethodnom izlaganju prikazan je jednostavan servis u svrsi opisivanja načina kreiranja, pokretanja i zaustavljanja servisa. Pošto taj servis ne radi ništa konkretno neophodno je proširiti njegovi definiciju dajući mu konkretan zadatak. Zadatak može biti simulacija ili konkretno realno zaduženje. U ovom slučaju zadatak servisa jeste da omogući preuzimanje sadržaja sa Interneta i prikazivanje vrednosti za preuzetu količinu podataka u bajtovima. U kreiranom projektu MojServisDemo biće izvršena modifikacija klase MojServis.java kodom koji je istaknut na sledećoj slici.

```
package com.metropolitan.mojservisdemo;

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

import java.net.MalformedURLException;
import java.net.URL;

/**
 * Created by Vladimir Milicevic on 27.11.2016..
 */

public class MojServis extends Service {
    @Override
    public IBinder onBind(Intent arg0) {
        return null;

    }
    @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
        // Želimo da servis bude aktivan dok se eksplicitno ne zaustavi
       // zato vraća vrednost sticky.
       //Toast.makeText(this, "Servis je pokrenut", Toast.LENGTH_LONG).show();
        try {
            int result = DownloadFile (new URL(
                    "http://www.amazon.com/somefiles.pdf"));
            Toast.makeText(getBaseContext(),
                    "Preuzeto je " + result + " bajtova",
```

```
                    Toast.LENGTH_LONG).show();
        }catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return START_STICKY;

    }


    private int DownloadFile(URL url) {
        try {
            //---simulacija proticanja vremena dok se fajl preuzima---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //---vraćanje proizvoljnog broja koji
        // predstavlja veličinu preuzete datoteke---
        return 100;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();

        Toast.makeText(this, "Servis je zaustavljen",
                Toast.LENGTH_LONG).show();
    }
}
```

DUGOTRAJNI ZADACI U SERVISU - FUNKCIONISANJE

Metoda DownloadFile() vraća ukupan broj bajtova koji su preuzeti.

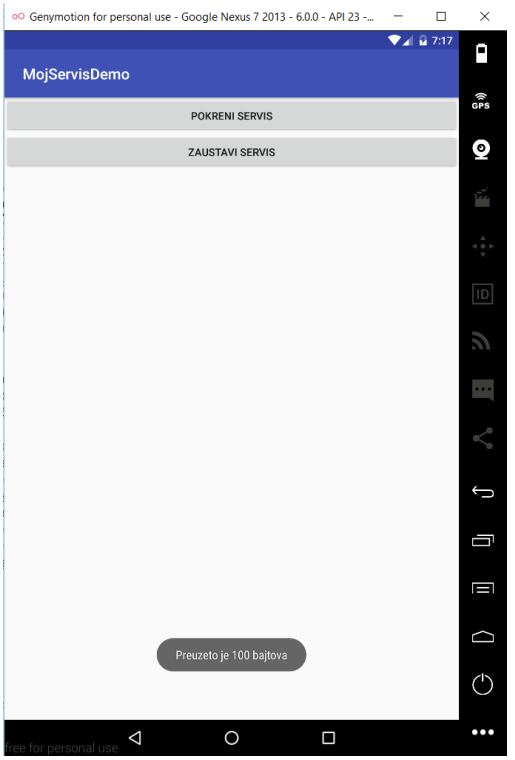
U ovom, proširenom primeru, servis izvršava metodu DownloadFile() za simuliranje preuzimanja datoteke sa zadate URL adrese. Metoda kao rezultat vraća ukupan broj preuzetih bajtova (što je fiksirano u kodu brojem 100 – videti priloženi kod).

Za simuliranje kašnjenja, nastalog zbog preuzimanja datoteke sa Interneta, korišćena je metoda Thread.Sleep() kojom je kašnjenje postavljeno na 5 sekundi (5000 ms).

Kada se pokrene servis, aktivnost je pauzirana 5 sekundi, a to je Interval u okviru kojeg se simulira preuzimanje sadržaja sa Interneta. U toku ovog intervala aktivnost nije responzivna, a to označava jednu bitnu stvar: servis se izvršava u istoj niti u kojoj se izvršava aktivnost.To, upravo, znači da će u ovom intervalu biti zaustavljena i glavna aktivnost u trajanju od 5s. Iz navedenog razloga, kada se koriste servisi koji izvršavaju dugotrajne operacije, važno je da te operacije budu smeštene u posebnu nit i da se na taj način spreči privremeno blokiranje aplikacije koja koristi servis. Upravo u tom kontekstu će ići buduće izlaganje ove teme.

Sada, pošto su izvršene modifikacije u aplikaciji, neophodno je izvršiti njeno ponovno prevođenje sa ciljem njenog testiranja.

Klasa MojServis.java je snimljena i klikom na Shift + F10, u Android Studio IDE razvojnom okruženju, pokrenuto je debagovanje aplikacije u izabranom emulatoru, ovde će to biti emulator visokih performansi Genymotion. Sledeća slika demonstrira pokretanje servisa.



Slika 2.1 Simuliranje servisa koji traje

ZADATAK 2

Pokušajte sami!!!

Pokušajte da nađete na Internetu neki primer sa dugotrajnim zadacima, na primer preuzimanje datoteke, i pokušajte da ga implementirate na način prikazan u ovom delu lekcije.

Poglavlje 3

Asinhrono izvršavanje zadataka u servisu

PRIMER 3 - PRIMENA ASYNCTASK KLASE U SERVISIMA

Da bi se servisi asinhrono izvršavali neophodno je primeniti klasu AsyncTask.

Kao što je napomenuto u prethodnom izlaganju, akcenat će biti prenet na izvršavanje servisa u pomoćnoj niti u odnosu na glavnu nit aplikacije.

U prethodnom primeru je pokazano da dugotrajno izvršavanje servisa može da izazove obustavu izvršavanja glavne aktivnosti za vreme trajanja servisa. Da bi se to sprečilo neophodno je pokrenuti servis u posebnoj niti i primeniti poznatu klasu AsyncTask. U tom svetlu će i prethodni primer biti modifikovan. Za početak, neophodno je u klasu MojServis.java uvesti sledeće pakete i klase.

```
package com.metropolitan.mojservisdemo;

import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;
import java.net.MalformedURLException;
import java.net.URL;
```

Nakon obavljene navedene akcije, pristupa se modifikaciji klase MojServis.java. Modifikovani kod ove klase je priložen sledećim listingom.

```
package com.metropolitan.mojservisdemo;

import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;
import java.net.MalformedURLException;
import java.net.URL;

```

```
/**
 * Created by Vladimir Milicevic on 27.11.2016..
 */

public class MojServis extends Service {
    @Override
    public IBinder onBind(Intent arg0) {
        return null;

    }
    @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
        // Želimo da servis bude aktivan dok se eksplicitno ne zaustavi
       // zato vraća vrednost sticky.
       //Toast.makeText(this, "Servis je pokrenut", Toast.LENGTH_LONG).show();
        try {
            new DoBackgroundTask().execute(
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf"));

        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return START_STICKY;

    }

    private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
        protected Long doInBackground(URL... urls) {
            int count = urls.length;
            long totalBytesDownloaded = 0;
            for (int i = 0; i < count; i++) {
                totalBytesDownloaded += DownloadFile(urls[i]);
                //---računa procenat preuzimanja i
                // izveštava o napretku---
                publishProgress((int) (((i+1) / (float) count) * 100));
            }
            return totalBytesDownloaded;
        }

        protected void onProgressUpdate(Integer... progress) {
            Log.d("Preuzimanje fajlova",
                    String.valueOf(progress[0]) + "% preuzeto");
            Toast.makeText(getBaseContext(),
                    String.valueOf(progress[0]) + "% preuzeto",
                    Toast.LENGTH_LONG).show();
        }

    protected void onPostExecute(Long result) {
        Toast.makeText(getBaseContext(),
```

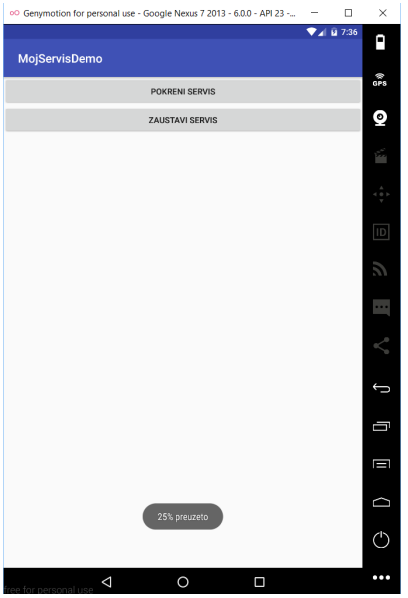
```
                "Preuzeto je " + result + " bajtova",
                Toast.LENGTH_LONG).show();
        stopSelf();
        }
    }

    private int DownloadFile(URL url) {
        try {
            //---simulacija proticanja vremena dok se fajl preuzima---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //---vraćanje proizvoljnog broja koji
        // predstavlja veličinu preuzete datoteke---
        return 100;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();

        Toast.makeText(this, "Servis je zaustavljen",
                Toast.LENGTH_LONG).show();
    }
}
```

TESTIRANJE I PRAĆENJE

U emulatoru i LogCat prozoru biće praćeno izvršavanje servisa

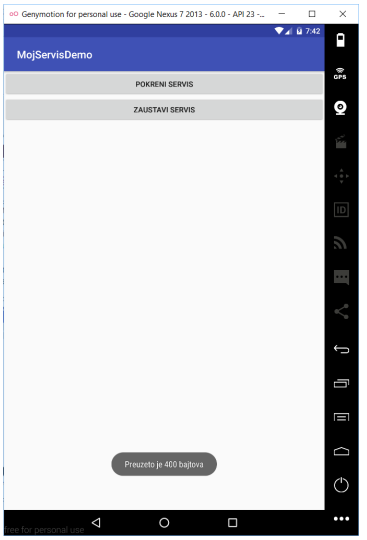
Budući da je izvršena modifikacija prethodnog koda, aplikaciju je neophodno ponovo prevesti u razvojnom okruženju Android Studio IDE sa ciljem dalje analize i testiranja.

Nakon pokretanja aplikacije, Toast klasa prikazuje koja količina podataka, u procentima, je preuzeta (sledeća slika).



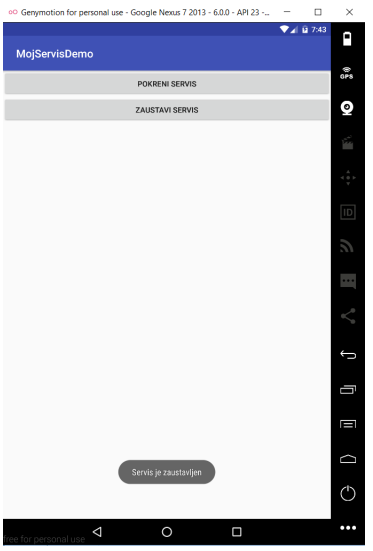
Slika 3.1 Početak servisa i procenat preuzetog sadržaja

Servis je započeo i u procentima se simulira progres u preuzimanju sadržaja sa zadate web lokacije. Nakon završenog preuzimanja, Toast klasa prikazuje ukupnu količinu preuzetih podataka (sledeća slika).



Slika 3.2 Količina preuzetih podataka

Na samom kraju, Toast klasa prikazuje da je servis zaustavljen.



Slika 3.3 Servis je zaustavljen

Takođe, proces preuzimanja je moguće pratiti kroz LogCat prozor (sledeća slika).

Tag	Text
Preuzimanje fajlova	50% preuzeto
Preuzimanje fajlova	75% preuzeto
Preuzimanje fajlova	100% preuzeto

Slika 3.4 Praćenje progresa u LogCat prozoru

ASINHRONI SERVISI - FUNKCIONISANJE

Ugnježdavanjem klase naslednice AsyncTask klase, omogućeno je pozadinsko izvršavanje zadataka u servisu.

Primer je pokazao kako je moguće asinhrono izvršiti zadatak u kreiranom servisu. Navedeno je moguće realizovati kreiranjem unutrašnje klase koja je naslednica AsyncTask klase. AsyncTask klasa omogućava da se izvršavaju zadaci u pozadini, bez potrebe za manuelnim upravljanjem nitima i rukovaocima.

Klasa DoBackgroundTask nasleđuje klasu AsyncTask kroz specifikaciju tri generička tipa:

private class DoBackgroundTask extends AsyncTask <URL, Integer, Long>

Navedeni tipovi podataka: *URL, Integer i Long* specificiraju tipove podataka koji se koriste u sledećim metodama implementiranim *AsyncTask* klasom:

- doInBackground() – Prihvata polje prvog generičkog tipa, a u ovom slučaju je *URL*. Metoda se izvršava u pozadinskoj niti i nosi programski kod koji se dugo izvršava. Za izveštavanje o procesu izvršavanja zadatka, koristi se metoda *publishProgress()* (videti priloženi kod). Ova metoda poziva metodu *onProgressUpdate()* koja je implementirana u *AsyncTask* klasi. Povratni tip ove metode je treći generički tip, *Long* u ovom slučaju.

- onProgressUpdate() – Poziva nit korisničkog interfejsa kada se izvrši metoda *publishProgress()*. Metoda prihvata kao argument drugi generički tip, *Integer* u ovom slučaju. Metoda se koristi za izveštavanje o napretku izvršavanja zadataka u pozadini.
- onPostExecute() – Poziva se u niti korisničkog interfejsa i izvršava se po okončanju metode *doInBackground()*. Metoda prihvata argument određen trećim generičkim tipom, u ovom slučaju *Long*. Sledeća slika prikazuje vezu između tipova podataka i metoda izvedene klase *DoBackgroundTask*.

```
private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalBytesDownloaded = 0;
        for (int i = 0; i < count; i++) {
            totalBytesDownloaded += DownloadFile(urls[i]);
            //---računa procenat preuzimanja i
            // izveštava o napretku---
            publishProgress((int) (((i+1) / (float) count) * 100));
        }
        return totalBytesDownloaded;
    }

    protected void onProgressUpdate(Integer... progress) {
        Log.d("Preuzimanje fajlova",
                String.valueOf(progress[0]) + "% preuzeto");
        Toast.makeText(getBaseContext(),
            String.valueOf(progress[0]) + "% preuzeto",
            Toast.LENGTH_LONG).show();
    }

      protected void onPostExecute(Long result) {
          Toast.makeText(getBaseContext(),
                  "Downloaded " + result + " bytes",
                  Toast.LENGTH_LONG).show();
          stopSelf();
        }
    }
```

Slika 3.5 Veza generičkih tipova i metoda

ASINHRONI SERVISI – FUNKCIONISANJE - NASTAVAK

Objektom klase DoBackgroundTask obezbeđeno je preuzimanje više datoteka u pozadini.

Za izvršavanje zadatka preuzimanja više datoteka u pozadini, neophodno je kreirati objekat klase DoBackgroundTask koja će izvršiti metodu execute() prosleđujući joj *URL* adrese. Ovim kodom servisu je omogućeno da preuzima datoteke u pozadini i vrši izveštavanje o napretku. Na ovaj način, aktivnost i dalje ostaje funkcionalna jer se preuzimanje odvija u posebnoj niti.

Sledećom listingom izdvojen je kod koji obavlja gore-navedene zadatke.

```
try {
            new DoBackgroundTask().execute(
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf"));

        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

Kao dopunsku funkcionalnost u proširenoj aplikaciji moguće je navesti automatsko zaustavljanje pokrenutog servisa. Zadatak automatskog zaustavljanja servisa obavila je metoda stopSelf(). Ova metoda je ekvivalentna metodi *stopService()* koja se koristi za ručno zaustavljanje servisa. Sledećom listingom izdvojen je kod koji implementira navedenu metodu.

```
protected void onPostExecute(Long result) {
        Toast.makeText(getBaseContext(),
                "Preuzeto je " + result + " bajtova",
                Toast.LENGTH_LONG).show();
        stopSelf();
        }
```

ZADATAK 3

Pokušajte sami!!!

Pokušajte da primer sa vašim servisom nadogradite primenom AsyncTask klase.

Poglavlje 4

Ponavljanje zadataka u servisu

PONAVLJANJE ZADATAKA POMOĆU TIMER KLASE

Za izvršavanje određenog bloka koda u fiksnim vremenskim intervalima, koristiti se klasa Timer.

Veoma često, koristeći mobilne uređaje, moguće je sresti se sa zadacima koji se periodično ponavljaju, a inicirani su aplikacijama koje su ili ugrađene u uređaj, ili preuzete sa Android Marketa ili su ih korisnici kreirali i instalirali na mobilni uređaj.

Android aplikacija, pored dugotrajnih, može da obrađuje i zadatke koji se ponavljaju u servisu. Postoje brojni primeri za takve zadatke, a jedan od njih može biti servis za simuliranje alarma koji je stalno aktivan u pozadini. Servis može periodično da izvršava neki kod sa ciljem provere poklapanja aktivacije alarma sa unapred definisanim vremenom. U slučaju podudaranja, aktivira se alarm. Za izvršavanje određenog bloka koda u fiksnim vremenskim intervalima, moguće je koristiti klasu Timer. Tekući primer biće prilagođen na taj način da njegova funkcionalnost uključuje klasu Timer.

Za funkcionisanje klase MojServis biće neophodno koristiti sledeće import instrukcije sa kojim počinje modifikacija koda ove klase, a u skladu sa navedenim izlaganjem.

```
import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.Timer;
import java.util.TimerTask;
```

PRIMER 4 - MODIFIKACIJA KLASE MYSERVICES

Klasa MyServices mora da bude modifikovana novim metodama i funkcionalnostima.

U daljem radu, neophodno je konkretizovati modifikaciju klase MojServis.java. To podrazumeva dodavanje metoda koje će omogućiti da se izvestan servis, iniciran aplikacijom,

ponavlja po unapred zadatom intervalu. U ovom delu izlaganja biće priložen kod modifikovane klase MojServis.java, sledećim listingom, da bi taj listing mogao da bude analiziran i objašnjen u izlaganju koje sledi. Posebno je potrebno obratiti pažnju na novu metodu doSomethingRepeatedly(), modifikovanu metodu onDestroy() i polja koja su dodata u definiciju klase.

```
package com.metropolitan.mojservisdemo;


import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.Timer;
import java.util.TimerTask;

/**
 * Created by Vladimir Milicevic on 27.11.2016..
 */

public class MojServis extends Service {

    int counter = 0;
    static final int UPDATE_INTERVAL = 1000;
    private Timer timer = new Timer();

    @Override
    public IBinder onBind(Intent arg0) {
        return null;

    }
    @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
        // Želimo da servis bude aktivan dok se eksplicitno ne zaustavi
       // zato vraća vrednost sticky.
       //Toast.makeText(this, "Servis je pokrenut", Toast.LENGTH_LONG).show();

        doSomethingRepeatedly();

        try {
            new DoBackgroundTask().execute(
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf"));

        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
```

```
            e.printStackTrace();
        }
        return START_STICKY;

    }

    private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
        protected Long doInBackground(URL... urls) {
            int count = urls.length;
            long totalBytesDownloaded = 0;
            for (int i = 0; i < count; i++) {
                totalBytesDownloaded += DownloadFile(urls[i]);
                //---računa procenat preuzimanja i
                // izveštava o napretku---
                publishProgress((int) (((i+1) / (float) count) * 100));
            }
            return totalBytesDownloaded;
        }

        protected void onProgressUpdate(Integer... progress) {
            Log.d("Preuzimanje fajlova",
                    String.valueOf(progress[0]) + "% preuzeto");
            Toast.makeText(getBaseContext(),
                    String.valueOf(progress[0]) + "% preuzeto",
                    Toast.LENGTH_LONG).show();
        }

    protected void onPostExecute(Long result) {
        Toast.makeText(getBaseContext(),
                "Preuzeto je " + result + " bajtova",
                Toast.LENGTH_LONG).show();
        stopSelf();
        }
    }

    private void doSomethingRepeatedly() {
        timer.scheduleAtFixedRate( new TimerTask() {
            public void run() {
                Log.d("MyService", String.valueOf(++counter));
            }
        }, 0, UPDATE_INTERVAL);
    }

    private int DownloadFile(URL url) {
        try {
            //---simulacija proticanja vremena dok se fajl preuzima---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //---vraćanje proizvoljnog broja koji
        // predstavlja veličinu preuzete datoteke---
        return 100;
```

```
    }

    @Override
    public void onDestroy() {
        super.onDestroy();

        if (timer != null){
            timer.cancel();
        }
        /*Toast.makeText(this, "Servis je zaustavljen",
                Toast.LENGTH_LONG).show();*/
    }
}
```

NAČIN FUNKCIONISANJA

U metodi doSomethingRepeatedly() kreiran je objekat klase Timer.

Nakon prilaganja modifikovanog koda klase MojServis.java, neophodno je pristupiti njenoj analizi i objašnjenju načina funkcionisanja.

Nova metoda u primeru je doSomethingRepeatedly() u kojoj je kreiran objekat klase Timer. Takođe, prosleđen je objekat klaseTimerTask do metode scheduleAtFixedRate() čime je omogućeno ponavljanje koda koji se nalazi u metodi run(). Metoda doSomethingRepeatedly() poseduje još dva argumenta. Drugi predstavlja vremenski interval izražen u milisekundama do prvog izvršavanja, a treći predstavlja vremenski interval izražen u milisekundama između uzastopnih izvršavanja. Objašnjene funkcionalnosti odnosi se na kod koji pripada metodi doSomethingRepeatedly(), izolovan iz klase MojServis.java, i priložen je sledećim listingom.

```
private void doSomethingRepeatedly() {
        timer.scheduleAtFixedRate( new TimerTask() {
            public void run() {
                Log.d("MyService", String.valueOf(++counter));
            }
        }, 0, UPDATE_INTERVAL);
    }
```

Konstantom ***UPDATE_INTERVAL=1000*** prikazana je vrednost brojača u milisekundama. Servis neprekidno prikazuje vrednost brojača sve dok se ne zaustavi njegovo dalje izvršavanje metodom *onDestroy()*. Što se tiče metode doSomethingRepeatedly() kod se izvršava u fiksnim vremenskim intervalima, nezavisno od toga koliko traje svaki pojedinačan zadatak. Takođe, ova metoda je pozvana direktno od metode onStartCommand() i bez predefinisanja u klasi naslednici AsyncTask. Ovo je moguće zato što klasa Timer implementira Runnable interfejs koji omogućava izvršavanje u posebnoj niti.

LogCat prozor pokazuje detalje izvršavanja programa nakon klika na *Pokreni servis* (sledeća slika).

Slika 4.1 Demonstracija ponavljanja zadataka u servisu u LogCat prozoru

ZADATAK 4

Pokušajte sami!!!

Proširite vaš primer primenom zadatak koji se ponavljaju u servisu.

Poglavlje 5

Primena IntentService klase

PRIMER 5 - SAMOSTALNO ZAUSTAVLJANJE ZADATKA

IntentService je osnovna klasa za upravljanje asinhronim zahtevima.

Samostalno zaustavljanje zadataka u Android aplikaciji je nezaobilazan koncept koji mora biti razmotren u okviru ove lekcije iz razloga što se veoma često i neizostavno koristi.

Nakon obavljanja izvesnog zadatka, servis mora da bude zaustavljen da ne bi došlo do zauzimanja resursa. Iz ovog razloga je korišćena metoda stopSelf() kojom se obustavlja dalje izvršavanje servisa po obavljenom zadatku. U velikom broju slučajeva, dešava se da programer zaboravi da zaustavi servis nakon što je u njemu obavljen izvesni zadatak. Za jednostavno kreiranje servisa koji asinhrono izvršava zadatak i, potom, automatski se zaustavlja moguće je kreirati i koristiti klasu IntentService.

IntentService je osnovna klasa za upravljanje asinhronim zahtevima. U ovom slučaju, servis se pokreće kao i svaki drugi. Takođe, izvršava svoj zadatak u okviru radne niti i samostalno se završava kada se taj zadatak završi. U Narednom primeru biće prikazana primena klase koja nasleđuje osnovnu klasu IntentService.

U Android Studio IDE razvojnom okruženju, u folderu kreiranog projekta, MojServisDemo, u podfolderu src rezervisanom za JAVA klase aplikacije, biće kreiranja nova klasa pod nazivom MyIntentService.java. Klasa će biti realizovana kao naslednica klase IntentServise, na način priložen sledećim listingom. Ovaj listing će biti posebno analiziran u narednom izlaganju.

```
package com.metropolitan.mojservisdemo;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

import java.net.MalformedURLException;
import java.net.URL;

/**
 * Created by Vladimir Milicevic on 29.11.2016..
 */

public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentServiceName");
    }
```

```
    @Override
    protected void onHandleIntent(Intent intent) {
        try {
            int result =
                    DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
            Log.d("IntentService", "Preuzeo je " + result + " bajtova");


        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

    private int DownloadFile(URL url) {
        try {
            //---simulacija preuzimanja datoteke---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return 100;
    }
}
```

SAMOSTALNO ZAUSTAVLJANJE ZADATKA – ANDROIDMANIFEST.XML

AndroidManifest.xml je neophodno modifikovati dodavanjem servisa i intent filtera.

Posebnu pažnju bi sada trebalo obratiti na datoteku AndroidManifest.xml. U projekat je ugrađena nova klasa kojom je omogućeno manipulisanje servisima. Iz navedenog razloga, u AndroidManifest.xml datoteci, moraće da bude kreiran novi *<service ... />* element kojem će biti pridružena kreirana klasa. Modifikacija datoteke AndroidManifest.xml, neophodna za realizovanje novih zahteva, priložena je sledećim listingom.

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

    <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>

        <service android:name=".MojServis">
            <intent-filter>
                <action android:name="com.metropolitan.mojservisdemo.MojServis" />
            </intent-filter>
        </service>

        <service android:name=".MyIntentService" />

    </application>

</manifest>
```

SAMOSTALNO ZAUSTAVLJANJE ZADATKA – KLASA AKTIVNOSTI

Neophodno je modifikovati metodu startService() klase aktivnosti.

Upravljanje aplikacijom u celini, u domenu je glavne klase aktivnosti. Uvedena je nova servisna klasa i klasi aktivnosti je neophodno obezbediti Intent objekat pomoću kojeg će moći da se obrati novoj klasi. Ovaj objekat je funkcionalan u metodi kojom je omogućeno pokretanje servisa. Upravo iz navedenih razloga, u klasu aktivnosti neophodno je ugraditi sledeće modifikacije koje se odnose na njenu metodu *startService()*.

```
public void startService(View view) {
        startService(new Intent(getBaseContext(), MyIntentService.class));
       }
```

U program su ugrađene željene modifikacije koje je neophodno u daljem izlaganju pokazati i testirati. Ponovo je neophodno, u Android Studio IDE razvojnom okruženju prevesti aplikaciju. Ukoliko razvojno okruženje ne pokaže greške, aplikacija se pokreće u izabranom emulatoru i odgovarajući servis započinje izvršavanje. Nakon 5 sekundi (pogledati sledeći kod)u LogCat prozoru moguće je primetiti izveštaj priložen sledećom slikom.

```
private int DownloadFile(URL url) {
        try {
            //---simulacija preuzimanja datoteke---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

```
        return 100;
    }
```

Slika 5.1 Izveštaj u LogCat prozoru

Sledećim video materijalom je prikazano kreiranje servisa primenom IntentService.

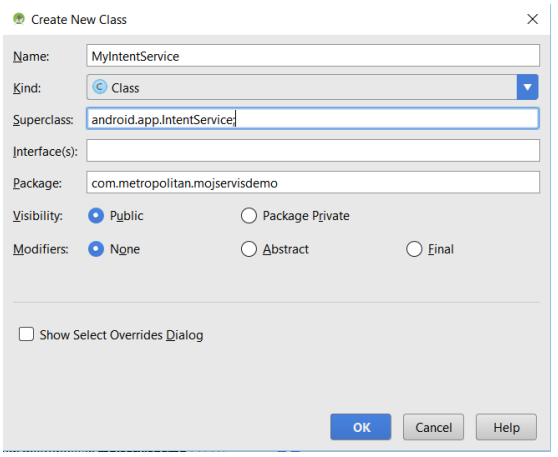
Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

NAČIN FUNKCIONISANJA

Metoda onHandleIntent() se izvršava u radnoj niti i predstavlja lokaciju na kojoj se postavlja kod koji treba da se izvrši u posebnoj niti.

Konačno, priložene poslednje modifikacije u projektu primera, neophodno je obrazložiti u narednom izlaganju.

U prvom koraku definisana je klasa MyIntentServices koja nasleđuje IntentServiceklasu, a to je prezentovano sledećom slikom.



Slika 5.2 Kreiranje nove klase u projektu

Klikom na OK, u ovom prozoru Android Studio IDE razvojnog okruženja, kreirana je prazna klasa MyIntentService:

```
public class MyIntentService extends IntentService {

}
```

Potom je implementiran konstruktor ove klase sa pozivom njene superklase i navođenjem naziva Intent servisa (pomoću istaknutog stringa):

```
.....
public MyIntentService() {

super("MyIntentServiceName");

}
.....
```

U nastavku, bilo je neophodno implementirati onHandleIntent() metodu, za rukovanje namerom,koja se izvršava u radnoj niti.

Metoda onHandleIntent() predstavlja lokaciju na kojoj se postavlja kod koji treba da se izvrši u posebnoj niti, u ovom slučaju simuliranje preuzimanja datoteke sa servera. Nakon izvršavanja koda, nit se prekida, servis se automatski zaustavlja. Upravo ovde su u potpunosti izvršeni poslednji zahtevi koji su postavljeni pred ovu aplikaciju.

ZADATAK 5

Pokušajte sami!!!

Omogućite u vašem primeru samostalno zaustavljanje zadataka.

Poglavlje 6

Komunikacija izmedju servisa i aktivnosti

PRIMER 6 - USPOSTAVLJANJE KOMUNIKACIJE IZMEĐU AKTIVNOSTI I SERVISA

Za demonstraciju komunikacije servisa i aktivnosti biće iskorišćena BroadcastReceiver klasa.

U prethodnim izlaganjima su postavljene dobre osnove za građenje složenijih scenarija primene Android aplikacija koje koriste servise. U nastavku, akcenat će biti na razmatranju komunikacije između aktivnosti i servisa

Kada servis identifikuje podatke, koji su u fokusu, i kada postoji potreba da se ti podaci proslede do aktivnosti, neophodno je pronaći adekvatne mehanizme kojima će navedeno biti omogućeno. Drugim rečima, programer će morati da osmisli odredjeni način kako da servis komunicira sa konkretnom aktivnošću.

U tom svetlu će biti neophodna pomoć novih klasa. Za demonstraciju komunikacije servisa i aktivnosti biće iskorišćena BroadcastReceiver klasa. Definicija klase MyIntentService.java, za početak, biće promenjena, a njene ***import*** instrukcije biće identične kao u prethodnom primeru. Korekcije su istaknute sa tri nove linije koda, umetnute u metodu onHandleIntent(), u bloku try, ispod instrukcije: *Log.d("IntentService", "Preuzeo je " + result + " bajtova");*

Kod klase MyIntentService.java prikazan je sledećim listingom.

```
package com.metropolitan.mojservisdemo;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

import java.net.MalformedURLException;
import java.net.URL;

/**
 * Created by Vladimir Milicevic on 29.11.2016..
 */

public class MyIntentService extends IntentService {

    public MyIntentService() {
        super("MyIntentServiceName");
```

```
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        try {
            int result =
                    DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
            Log.d("IntentService", "Preuzeo je " + result + " bajtova");

            //---slanje poruke kojom se aktivnost obaveštava
            // da je datoteka preuzeta---
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("FILE_DOWNLOADED_ACTION");
            getBaseContext().sendBroadcast(broadcastIntent);

        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

    private int DownloadFile(URL url) {
        try {
            //---simulacija preuzimanja datoteke---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return 100;
    }
}
```

NOVA DEFINICIJA KLASE MAINACTIVITY.JAVA

Klasa aktivnosti mora da bude dopunjena funkcionalnostima za komunikaciju servis-aktivnost.

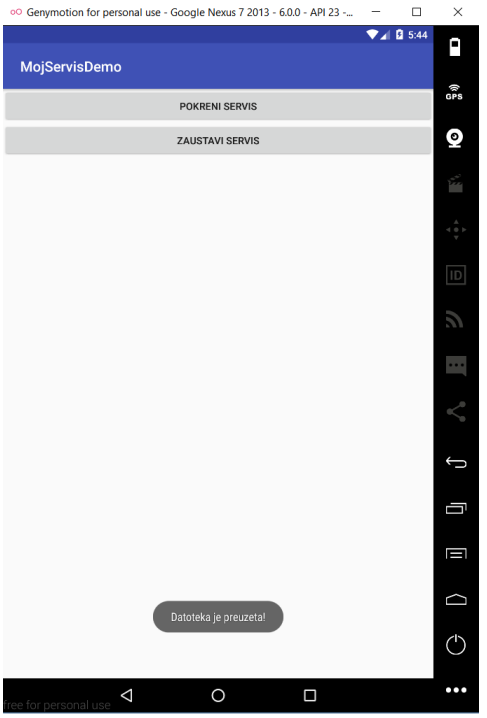
Za omogućavanje komunikacije između servisa i aktivnosti, posebne modifikacije mora da pretrpi glavna klasa aktivnosti MainActivity.java. Klasa će biti obogaćena novim import instrukcijama kojima će biti omogućeno korišćenje objekata tipa BroadcastReceiver i IntentFilter. Takođe, metode ove klase će biti značajno izmenjene i klasa aktivnosti će da dobije novu formu. Kod modifikovane klase aktivnosti prikazan je sledećim listingom koji će u narednom izlaganju biti predmet analize i demonstracije.

```
package com.metropolitan.mojservisdemo;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
```

```
import android.content.IntentFilter;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Toast;



public class MainActivity extends AppCompatActivity {

    IntentFilter intentFilter;

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

    @Override
    public void onResume() {
        super.onResume();

        //---filtriranje sadržaja za preuzimanje datoteke---
        intentFilter = new IntentFilter();
        intentFilter.addAction("FILE_DOWNLOADED_ACTION");

        //---registrovanje prijemnika---
        registerReceiver(intentReceiver, intentFilter);
    }

    @Override
    public void onPause() {
        super.onPause();

        //---uklanjanje prijemnika---
        unregisterReceiver(intentReceiver);
    }

    public void startService(View view) {
        startService(new Intent(getBaseContext(), MyIntentService.class));
       }

    public void stopService(View view) {
        stopService(new Intent(getBaseContext(), MojServis.class));
    }

    private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getBaseContext(), "Datoteka je preuzeta!",
                    Toast.LENGTH_LONG).show();
        }
    };
```

```
}
```

Da bi modifikacije programa bile vidljive i mogle da budu testirane, neophodno je u Android Studio IDE razvojnom okruženju ponovo prevesti aplikaciju, izabrati AVD uređaj za testiranje i, ukoliko ne postoje greške prevođenja, pokrenuti aplikaciju. Modifikovani kod projekta imaće izlaz prikazan sledećom slikom.



Slika 6.1 Informacija o preuzimanju datoteke

TESTIRANJE I NAČIN FUNKCIONISANJA

Aktivnost dobija informaciju o završenom zadatku servisa primenom metode sendBroadcast().

Navedene modifikacije u programskom kodu datoteka projekta primera, zahtevaju posebno objašnjenje u ovom delu izlaganja. Prvo, analiza započinje metodom onHandleIntet() klase MyServiseIntent.java. Kod ove klase je izolovan sledećim listingom i sledi njegovo objašnjenje.

```
@Override
    protected void onHandleIntent(Intent intent) {
        try {
            int result =
                    DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
            Log.d("IntentService", "Preuzeo je " + result + " bajtova");

            //---slanje poruke kojom se aktivnost obaveštava
            // da je datoteka preuzeta---
```

```
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("FILE_DOWNLOADED_ACTION");
            getBaseContext().sendBroadcast(broadcastIntent);

        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }
```

Za informisanje aktivnosti o završenom zadatku servisa iskorišćena je metoda sendBroadcast(). Poruka koja se emituje definisana je konstantom ***FILE_DOWNLOADED:ACTION*** , a to znači da se inicira svaka aktivnost koja prati nameru da se odredjeni zadatak izvrši. Iz navedenog razloga u klasi MainActivity.java definisan je kod koji omogućava praćenje ove namere metodom registerReceiver() klaseIntentFilter.

Navedeno je prikazano sledećim izdvojenim kodom.

```
@Override
    public void onResume() {
        super.onResume();

        //---filtriranje sadržaja za preuzimanje datoteke---
        intentFilter = new IntentFilter();
        intentFilter.addAction("FILE_DOWNLOADED_ACTION");

        //---registrovanje prijemnika---
        registerReceiver(intentReceiver, intentFilter);
    }
```

Nakon identifikovanja namere o izvršavanju određene akcije, poziva se objekat tipa BoradcastReceiver(sledeći izdvojeni listing). Izvršavanjem programa dobija se poruka Datoteka je preuzeta!U narednom izlaganju biće više reči o prosledjivanju odredjenih podataka iz servisa u aktivnost primenom Intentobjekta.

```
private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getBaseContext(), "Datoteka je preuzeta!",
                    Toast.LENGTH_LONG).show();
        }
    };
```

ZADATAK 6

Pokušajte sami!!!

Proširite vaš primer primenom BroadcastReceiver klase.

Poglavlje 7

Povezivanje aktivnosti sa servisima

PRIMER 7 - PROSLEĐIVANJE PODATAKA IZ SERVISA U AKTIVNOST INTENT OBJEKTOM.

Direktnim povezivanjem, aktivnost može da koristi sve javne članove i metode u komunikaciji sa servisom.

Primer nastavlja sa sve složenijim scenarijima primene servisa u Android aplikacijama.

U dosadašnjem izlaganju bilo je govora o kreiranju i izvršavanju servisa, kao i prekidanju njihovog daljeg izvršavanja kada se završi zadatak koji je u servisu definisan. Servisi su bili jednostavni i koristili su neki brojač ili su simulirali preuzimanje datoteka. U praksi, javljaju se servisi koji izvršavaju mnogo složenije operacije sa prosleđivanjem podataka.

Ako se uzme servis koji je do sad korišćen, u ovoj lekciji, i da se omogući utvrđivanje koje datoteke treba da se preuzmu u aktivnosti, najbolji način je primena direktnog povezivanja aktivnosti sa servisom. Tada aktivnost može direktno da koristi sve javne članove i metode u komunikaciji sa servisom.
U nastavku tekući primer biće modifikovan u navedenom svetlu, a to podrazumeva modifikaciju MojServis.java klase sa posebnim akcentom na metodu onStartCommand () koja je zadužena za obradu koda koji se odnosi na pokretanje servisa klikom na dugme.
Sledećim kodom, koji će u narednom izlaganju biti analiziran i demonstriran, data je modifikovana MojServis.java klasa.

```
package com.metropolitan.mojservisdemo;

import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Binder;
import android.os.IBinder;
import android.widget.Toast;

import java.net.URL;
import java.util.Timer;


/**
 * Created by Vladimir Milicevic on 27.11.2016..
 */

public class MojServis extends Service {
```

```
int counter = 0;
URL[] urls;

static final int UPDATE_INTERVAL = 1000;
private Timer timer = new Timer();

private final IBinder binder = new MyBinder();

public class MyBinder extends Binder {
    MojServis getService() {
        return MojServis.this;
    }
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // Želimo da servis bude aktivan dok se eksplicitno ne zaustavi
    // zato vraća vrednost sticky.
    Toast.makeText(this, "Servis je pokrenut", Toast.LENGTH_LONG).show();
    new DoBackgroundTask().execute(urls);
    return START_STICKY;
}

private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalBytesDownloaded = 0;
        for (int i = 0; i < count; i++) {
            totalBytesDownloaded += DownloadFile(urls[i]);
            //---računa procenat preuzimanja i
            // izveštava o napretku---
            publishProgress((int) (((i+1) / (float) count) * 100));
        }
        return totalBytesDownloaded;
    }
}



private int DownloadFile(URL url) {
    try {
        //---simulacija proticanja vremena dok se fajl preuzima---
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    //---vraćanje proizvoljnog broja koji
    // predstavlja veličinu preuzete datoteke---
    return 100;
}

@Override
```

```
    public void onDestroy() {
        super.onDestroy();

        if (timer != null){
            timer.cancel();
        }
        /*Toast.makeText(this, "Servis je zaustavljen",
                Toast.LENGTH_LONG).show();*/
    }


    @Override
    public IBinder onBind(Intent arg0) {
        //return null;
        return binder;
    }

}
```

MODIFIKACIJA DATOTEKE MAINACTIVITY.JAVA

U datoteku aktivnosti neophodno je uvesti novi kod kojim se menja postojeća startService() metoda.

Upravljanje kompletnim izvršavanjem Android aplikacije je u domenu glavne klase aktivnosti. U svetlu navedenih novih zahteva, glavna klasa aktivnosti projekta će morati da bude značajno modifikovana. Posebno će biti uvedena nova metoda onServiceConected() i modifikovana startService() metoda. Modifikovani kod klase aktivnosti prikazan je sledećim listingom i detaljno će biti analiziran i demonstriran u narednom izlaganju.

```
package com.metropolitan.mojservisdemo;

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

import java.net.MalformedURLException;
import java.net.URL;
```

```
public class MainActivity extends AppCompatActivity {

    IntentFilter intentFilter;
    MojServis serviceBinder;
    Intent i;

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

    private ServiceConnection connection = new ServiceConnection() {


    public void onServiceConnected(ComponentName className, IBinder service) {
        //---poziva se kada je konekcija napravljena---
        serviceBinder = ((MojServis.MyBinder)service).getService();
        try {
            URL[] urls = new URL[] {
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf")};
            //---dodela URLa servisu kroz serviceBinder objekat---
            serviceBinder.urls = urls;
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        startService(i);
    }

    public void onServiceDisconnected(ComponentName className) {
        //---poziva se kada se servis zaustavlja---
        serviceBinder = null;
    }
};


@Override
    public void onResume() {
        super.onResume();

        //---filtriranje sadržaja za preuzimanje datoteke---
        intentFilter = new IntentFilter();
        intentFilter.addAction("FILE_DOWNLOADED_ACTION");

        //---registrovanje prijemnika---
        registerReceiver(intentReceiver, intentFilter);
    }

    @Override
    public void onPause() {
```

```
        super.onPause();

        //---uklanjanje prijemnika---
        unregisterReceiver(intentReceiver);
    }

    public void startService(View view) {

        i = new Intent(MainActivity.this, MojServis.class);
        bindService(i, connection, Context.BIND_AUTO_CREATE);
       }

    public void stopService(View view) {
        stopService(new Intent(getBaseContext(), MojServis.class));
    }

    private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getBaseContext(), "Datoteka je preuzeta!",
                    Toast.LENGTH_LONG).show();
        }
    };

}
```

PROSLEĐIVANJE PODATAKA IZ SERVISA U AKTIVNOST - FUNKCIONISANJE

Da bi aktivnost bila povezana sa servisom, prvo je neophodno kreirati unutrašnju klasu u servisu koja nasleđuje klasu Binder.

Da bi aktivnost bila povezana sa servisom, prvo je neophodno kreirati unutrašnju klasu u servisu. Klasa se zove MyBinder.java i nasleđuje osnovnu klasu Binder.

```
public class MyBinder extends Binder {
        MojServis getService() {
            return MojServis.this;
        }
    }
```

Ova klasa implementira metodu getServices() koja vraća objekat servisa. U sledećem koraku kreira se objekat klase MyBinder :

private final IBinder binder = new MyBinder();

i modifukuje se metoda onBind() da bi bilo omogućeno vraćanje objekta klase MyBinder.

```
@Override
    public IBinder onBind(Intent arg0) {
```

```
        //return null;
        return binder;
    }
```

U metodi onStartCommand()izvršava se metoda execute(), primenjujući URL polje koje je definisano kao javni član u servisu (videti kod klase MojServis.java*)*.

Navedeno URL polje može se direktno postavljati iz aktivnosti, a to je urađeno na sledeći način:

1. Prvo su u klasi *ServicesActivity.java* deklarisani objekat servisa i *Intent objekat.*

```
....
MojServis serviceBinder;
    Intent i;
.....
```

Objekat serviceBinder je referenca servisa kojem se direktno pristupa.

2. Potom se kreira instanca klase ServiceConnection kojom je omogućeno praćenje stanja konkretnog servisa. Ovde je neophodno implementirati dve metode onServiceConnected() i onServiceDisconected(). Prva metoda se poziva kada se aktivnost poveže sa servisom, a druga kada se prekine veza između servisa i aktivnosti.

PROSLEĐIVANJE PODATAKA IZ SERVISA U AKTIVNOST – FUNKCIONISANJE - NASTAVAK

Metoda bindService() omogućava da aktivnost bude povezana sa servisom.

Sledi nastavak analize modifikovanog primera MojServisDemo. U metodi onServiceConnected(), prilikom povezivanja aktivnosti na servis, dobija se objekat servisa pomoću metode getService() *(*videti kod) koja se dodaje, potom, serviceBinder objektu. Objekat serviceBinderje referenca na servis. Svi članovi i metode u servisu mogu da se koriste pomoću ovog objekta. Ovde se kreira URL polje, a zatim se direktno dodaje javnom članu servisa.

```
....
URL[] urls = new URL[] {
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf")};
            //---dodela URLa servisu kroz serviceBinder objekat---
            serviceBinder.urls = urls;
.....
```

Nakon toga se pokreće servis pomoću sledeće instrukcije: ***startService(i);*** **-** koja je u vezi sa tasterom "Pokreni servis".

Pre nego što se pokrene servis, neophodno je povezati aktivnost i servis.

```
...
public void startService(View view) {

        i = new Intent(MainActivity.this, MojServis.class);
        bindService(i, connection, Context.BIND_AUTO_CREATE);
    }
...
```

Metoda bindService() omogućava da aktivnost bude povezana sa servisom. Metoda preuzima tri argumenta: Intent objekat, ServiceConnection obekat i indikator načina povezivanja servisa.

POVEZIVANJE AKTIVNOSTI SA SERVISIMA - VIDEO MATERIJALI

Video materijali koji pokazuju upotrebu IntentFilter-a i BroadcastReceover-a

Introduction To Service And BroadcastReceiver - IntentFilter - Part 1.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Introduction To Service And BroadcastReceiver - IntentFilter - Part 2

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK 7

Pokušajte sami!!!

Proširite vaš primer primenom prosleđivanja podataka iz servisa u aktivnost.

Poglavlje 8

Primena niti

PRIMER 8 - NITI U ANDROID APLIKACIJI

Biće uveden novi Android projekat, pod nazivom Threading, za demonstraciju primene niti u Android aplikacijama.

U dosadašnjem izlaganju pokazano je kako se servisi kreiraju i kako se upravlja zadacima koji dugo traju, posebno ako se radi o zadacima koji ažuriraju nit kojom se definiše korisnički interfejs. U tom svetlu, korišćena je klasa AsyncTask kojom je omogućeno izvršavanje dugotrajnih zadataka u pozadini. Potom se pomenute različite metode kojima su upravljani dugotrajni zadaci na različite načine.

Da bi prethodna razmatranja otišla korak dalje, biće uveden novi Android projekat, pod nazivom NitiDemo, za demonstraciju primene niti u Android aplikacijama.

Za početak, aplikacija mora da ima definisan korisnički interfejs koji će biti učitan u glavnu aktivnost. Sledećim listinom priložen je XML kod datoteke activity_main.xml kojim je realizovan korisnički interfejs aplikacije NitiDemo.

```
<?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="NitiDemo - primer!!!" />

    <Button
        android:id="@+id/btnStartCounter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="startCounter"
        android:text="Start" />

    <Button
        android:id="@+id/btnStopCounter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="stopCounter"
        android:text="Stop" />
```

```
    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Brojač" />

</LinearLayout>
```

KLASA MAINACTIVITY.JAVA PRIMERA

Realizovanje brojača u aktivnosti biće prepušteno klasi aktivnosti.

U nastavku, bitno je osmisliti način prezentacije primene niti u kreiranoj aplikaciji. Neka to bude jednostavan brojač kojim je omogućeno kontinuirano smenjivanje i prikazivanje vrednosti od 0 do 1000.

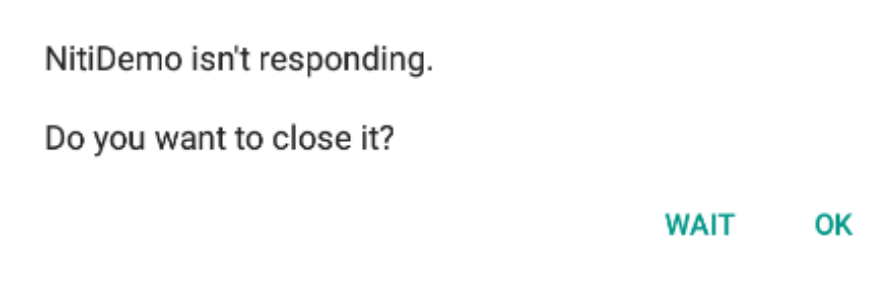
Sa ciljem realizovanja brojača u aktivnosti, od 0 do 1000, biće kreirana klasa MainActivity.java, klasa projekta, čiji je kod priložen sledećim listingom.

```
package com.metropolitan.nitidemo;

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

import android.util.Log;
import android.view.View;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    static TextView txtView1;

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

        txtView1 = (TextView) findViewById(R.id.textView1);
    }

    public void startCounter(View view) {
        for (int i=0; i<=1000; i++) {
            txtView1.setText(String.valueOf(i));
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Log.d("Threading", e.getLocalizedMessage());
            }
```

```
        }
    }
}
```

U nastavku, u Android Studio IDE razvojnom okruženju, kreirani kod se prevodi i testira u izabranom emulatoru, u konkretnom slučaju biće korišćen emulator visokih performansi Genymotion.

Prilikom pokretanja aplikacije, i klikom na dugme Start, aplikacija će se nakratko zaustaviti. Nakon određenog vremena pojaviće se poruka prikazana sledećom slikom.



Slika 8.1 Aplikacija je prekinula sa izvršavanjem

Korisnički interfejs se kratko *zamrzava* iz razloga što aplikacija uporno pokušava da prikaže vrednost brojača, a istovremeno se realizuje pauza od jedne sekunde nakon prikazivanja vrednosti. Na ovaj način je onemogućeno izvršavanje korisničkog interfejsa koji čeka da se završi prikazivanje brojeva.

MODIFIKACIJA METODE STARTCOUNTER() - NOVI PROBLEM I REŠENJE

Neophodno je naći opcije za rešavanje uočenog problema.

U daljem radu je neophodno istražiti mehanizme kojima će biti omogućeno prikazivanje i funkcionisanje brojača, a da pri tome ne dolazi do zauzimanja glavne niti i otkazivanja aplikacije.

Kao moguće rešenje jeste implementiranje Runnableinterfejsa i petlje sadržane u niti, u okviru metode startCounter(). To je prikazano sledećim listingom navedene metode.

```
public void startCounter(View view) {
        Log.d("Threading", "running");
        new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i=0; i<=1000; i++) {
                    Log.d("Threading", "running");
                    txtView1.setText(String.valueOf(i));
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
```

```
                    Log.d("Threading", e.getLocalizedMessage());
                }
            }
        }
    }).start();
}
```

U navedenom kodu, prvo je kreirana klasa koja implementira Runnable interfejs. Klasa sadrži kod koji se dugo izvršava u okviru run()metode. Blok Runnable se, zatim, pokreće pomoću Thread klase.

Moguće je primetiti, pokretanjem programa, da se ova aplikacija neće izvršavati.

Kod, koji je postavljen u *Runnable* blok, predstavlja posebnu nit. Ažuriranje korisničkog interfejsa, iz druge niti, nije bezbedno iz razloga što Android korisnički interfejs nije otporan na greške u nitima. Za rešavanje ovog problema moguće je uposliti metodupost() klase View za kreiranje drugog Runnable bloka koji se dodaje redu sa porukama. Drugim rečima, ovaj Runnable blok će se izvršavati u niti korisničkog interfejsa i aplikacija može bezbedno da se izvrši.

```
public void startCounter(View view) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i=0; i<=1000; i++) {
                    final int valueOfi = i;

                    //---ažuriranje UI u glavnoj aktivnosti---
                    txtView1.post(new Runnable() {
                        public void run() {
                            //---UI thread for updating---
                            txtView1.setText(String.valueOf(valueOfi));
                        }
                    });

                    //---odlaganje
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        Log.d("Threading", e.getLocalizedMessage());
                    }
                }
            }
        }).start();
    }
```

PRIMENA HANDLER KLASE - LEPŠA REŠENJA

Kreirana aplikacija funkcioniše ispravno ali je veoma složena, sa kodom komplikovanim za održavanje.

Kreirana aplikacija funkcioniše ispravno ali je veoma složena, sa kodom komplikovanim za održavanje. Otuda, bilo je nophodno tražiti nove načine za ažuriranje korisničkog interfejsa.

Ažuriranje korisničkog interfejsa iz druge niti omogućeno je primenom Handlerklase. Ova klasa omogućava slanje i obradu poruka, kao i *post()* metoda klase *View.* U sledećem kodu prikazana je primena klase Handler, kroz instancu UIUpdaterkoja ažurira korisnički interfejs na osnovu primljene poruke.

```
//---ažuriranje UI u glavnoj aktivnosti---
    static Handler UIupdater = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            byte[] buffer = (byte[]) msg.obj;

            //---konverzija niza bajtova u string---
            String strReceived = new String(buffer);

            //---prikazuje primljeni tekst u TextView---
            txtView1.setText(strReceived);
            Log.d("Threading", "pokrenuto...");
        }
    };
```

U nastavku je neophodno modifikovati metodu startCounter() koja dobija novu definiciju nakon prethodnog razmatranja.

```
public void startCounter(View view) {
        new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i=0; i<=1000; i++) {
                    //---ažuriranje UI u glavnoj aktivnosti---
                    ThreadingActivity.UIupdater.obtainMessage(
                        0,  String.valueOf(i).getBytes() ).sendToTarget();

                    //---odlaganje
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        Log.d("Threading", e.getLocalizedMessage());
                    }
                }
            }
        }).start();
    }*/
```

Mnogo više podataka o klasi Handler moguće je pronaći u dokumentaciji sa adrese:

http://developer.android.com/reference/android/os/Handler.html

PRIMENA KLASE ASYNCTASK U RADU SA NITIMA

Primenom jednostavne AsyncTask klase moguće je ažurirati UI iz posebne niti.

U daljem izlaganju traže se još kvalitetnija rešenja. Android operativni sistem dozvoljava primenu AsyncTask klase za ažuriranje korisničkog interfejsa iz posebne niti. U tom svetlu će biti kreirana klasa DoCountingTask koja nasleđuje klasu AsyncTask i čiji je kod prikazan sledećim listingom.

```
private class DoCountingTask extends AsyncTask<Void, Integer, Void> {
        protected Void doInBackground(Void... params) {
            for (int i = 0; i < 1000; i++) {
                //---izveštaj o napretku---
                publishProgress(i);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    Log.d("Threading", e.getLocalizedMessage());
                }
                if (isCancelled()) break;
            }
            return null;
        }

        protected void onProgressUpdate(Integer... progress) {
            txtView1.setText(progress[0].toString());
            Log.d("Threading", "ažuriranje...");
        }
    }
```

Zaustavljanje izvedene AsyncTask klase zahteva dodavanje njenog objekta i izvršavanje cancel() metode. Moguće je koristiti metodu isCancelled() za proveru potrebe prekida zadatka. Navedene aktivnosti su u domenu klase ThreadingActivity.java.

Konačno je neophodno priložiti konačnu implementaciju klase MainActivity.java izgrađene da demonstrira primenu niti u Android aplikacijama. Kod klase je priložen sledećim listingom.

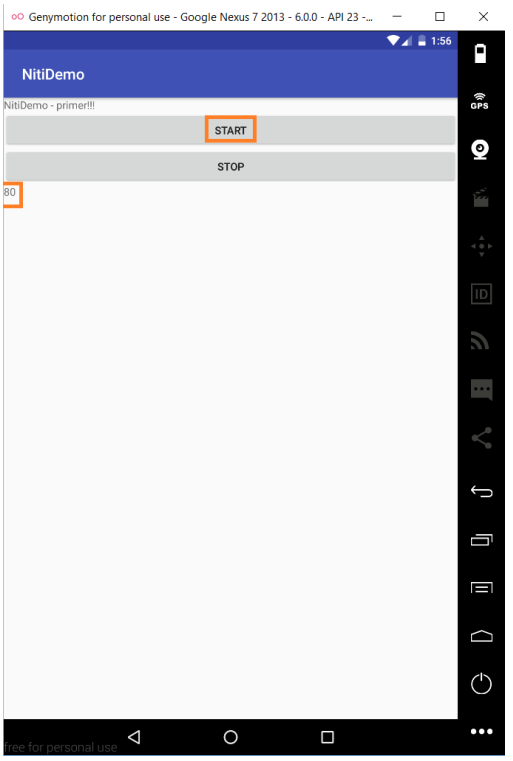
```
package com.metropolitan.nitidemo;

import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.TextView;


public class MainActivity extends AppCompatActivity {

```

```
    static TextView txtView1;
    //CountingThread thread;

    DoCountingTask task;


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

        txtView1 = (TextView) findViewById(R.id.textView1);
    }

    //--- Primer  ---
    public void startCounter(View view) {
        task = (DoCountingTask) new DoCountingTask().execute();
    }

    public void stopCounter(View view) {
        task.cancel(true);
    }

    private class DoCountingTask extends AsyncTask<Void, Integer, Void> {
        protected Void doInBackground(Void... params) {
            for (int i = 0; i < 1000; i++) {
                //---izveštaj o napretku---
                publishProgress(i);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    Log.d("Threading", e.getLocalizedMessage());
                }
                if (isCancelled()) break;
            }
            return null;
        }

        protected void onProgressUpdate(Integer... progress) {
            txtView1.setText(progress[0].toString());
            Log.d("Threading", "ažuriranje...");
        }
    }
    @Override
    protected void onPause() {
        // TODO Auto-generated method stub
        super.onPause();
        stopCounter(txtView1);
    }
}
```

DEMONSTRACIJA PRIMERA

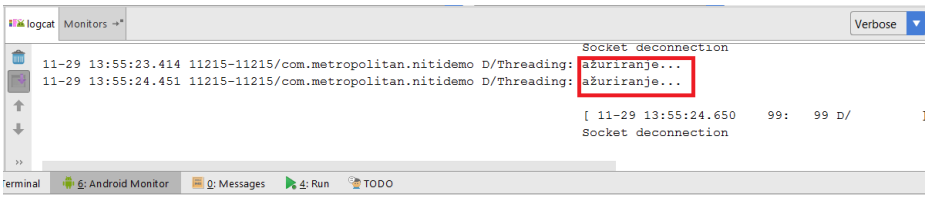
Pokrtanjem primera i primenom dva dugmeta Start i Stop upravlja se nitima.

Poslednji korak jeste testiranje kreiranog primera. U Android Studio IDE razvojnom okruženju, aplikacija je prevedena, nije bilo grešaka i u izabranom AVD uređaju je pokrenuta. Rezultat izvršavanja na emulatoru je priložen sledećom slikom.



Slika 8.2 Servis je pokrenut i brojač radi

Tokom izvršavanja aplikacije u LogCat prozoru je moguće posmatrati ažuriranje aktivnosti, a to je prikazano sledećom slikom.



Slika 8.3 Praćenje ažuriranja aktivnosti

Sledećim video materijalom je prezentovano korišćenje niti u aplikacijama sa Android servisima.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ZADATAK 7

Pokušajte sami!!!

Proširite vaš primer primenom Niti u Android aplikaciji.

Poglavlje 9

Razvoj Android servisa - rezime

PRIMER 9 - VEŽBANJE PRIMENE SERVISA U ANDROID APLIKACIJI

Vežbanje naučenih načina upravljanja servisima u Android aplikacijama.

ZADATAK:

1. Koristeći kao primer zadatak sa predavanja, realizovati projekat koji objedinjuje sve koncepte obrađene na časovima predavanja.

2. Aplikaciju razviti i testirati u Android Studio okruženju, koristeći API verziju koja odgovara Android 6.0 operativnom sistemu.

3. Primer menjati i testirati po scenarijima obrađenim na časovima predavanja.

REŠENJE:

Prvo se prilaže jednostavan GUI:

```
<?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/btnStartService"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Pokreni servis"
        android:onClick="startService"/>

    <Button android:id="@+id/btnStopService"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Zaustavi servis"
        android:onClick="stopService" />

</LinearLayout>
```

U sledećem izlaganju slede klase servisa. Klasa MyServices.java priložena je sledećim kodom. U nastavku, deo koda koji se ne koristi, u zavisnosti od scenarija primene, biće tretiran kao JAVA komentar.

```
package net.learn2develop.Services;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.Timer;
import java.util.TimerTask;
import android.app.Service;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

public class MyService extends Service {
    int counter = 0;
    URL[] urls;

    static final int UPDATE_INTERVAL = 1000;
    private Timer timer = new Timer();

    private final IBinder binder = new MyBinder();

    public class MyBinder extends Binder {
        MyService getService() {
            return MyService.this;
        }
    }
    @Override
    public IBinder onBind(Intent arg0) {
        //return null;
        return binder;
    }
//    @Override
//    public int onStartCommand(Intent intent, int flags, int startId) {
//        // Želimo da servis bude aktivan dok se eksplicitno ne zaustavi
//        // zato vraća vrednost sticky.
//        Toast.makeText(this, "Servis je pokrenut", Toast.LENGTH_LONG).show();
//        new DoBackgroundTask().execute(urls);
//        return START_STICKY;
//    }


    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // Želimo da servis bude aktivan dok se eksplicitno ne zaustavi
        // zato vraća vrednost sticky.

        Toast.makeText(this, "Servis je pokrenut", Toast.LENGTH_LONG).show();


       doSomethingRepeatedly();
```

```
        try {
            new DoBackgroundTask().execute(
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf"));

        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return START_STICKY;
    }
    /*@Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // Želimo da servis bude aktivan dok se eksplicitno ne zaustavi
        // zato vraća vrednost sticky.

    //  Toast.makeText(this, "Servis je pokrenut", Toast.LENGTH_LONG).show();



        try {
            int result = DownloadFile (new URL(
                    "http://www.amazon.com/somefiles.pdf"));
            Toast.makeText(getBaseContext(),
                    "Preuzeto je " + result + " bajtova",
                    Toast.LENGTH_LONG).show();
        } catch (MalformedURLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        */

        /*
        Object[] objUrls = (Object[]) intent.getExtras().get("URLs");
        URL[] urls = new URL[objUrls.length];
        for (int i=0; i<objUrls.length-1; i++) {
            urls[i] = (URL) objUrls[i];
        }
        new DoBackgroundTask().execute(urls);


        return START_STICKY;
    }
    */

    private void doSomethingRepeatedly() {
        timer.scheduleAtFixedRate( new TimerTask() {
            public void run() {
                Log.d("MyService", String.valueOf(++counter));
            }
        }, 0, UPDATE_INTERVAL);
```

```
    }

    private int DownloadFile(URL url) {
        try {
            //---simulacija proticanja vremena dok se fajl preuzima---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
             e.printStackTrace();
        }
        //---vraćanje proizvoljnog broja koji
        // predstavlja veličinu preuzete datoteke---
        return 100;
    }

    private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
        protected Long doInBackground(URL... urls) {
            int count = urls.length;
            long totalBytesDownloaded = 0;
            for (int i = 0; i < count; i++) {
                totalBytesDownloaded += DownloadFile(urls[i]);
                //---računa procenat preuzimanja i
                // izveštava o napretku---
                publishProgress((int) (((i+1) / (float) count) * 100));
            }
            return totalBytesDownloaded;
        }

       /* protected void onProgressUpdate(Integer... progress) {
            Log.d("Preuzimanje fajlova",
                    String.valueOf(progress[0]) + "% preuzeto");
            Toast.makeText(getBaseContext(),
                String.valueOf(progress[0]) + "% preuzeto",
                Toast.LENGTH_LONG).show();
        }

        protected void onPostExecute(Long result) {
            Toast.makeText(getBaseContext(),
                    "Preuzeto je " + result + " bajtova",
                    Toast.LENGTH_LONG).show();
            stopSelf();
        }*/    }

    @Override
    public void onDestroy() {
        super.onDestroy();

       if (timer != null){
            timer.cancel();
        }

        /*Toast.makeText(this, "Servis je zaustavljen",
                Toast.LENGTH_LONG).show();*/
```

```
    }
}
```

Klasa koja nasleđuje IntentSevice klasu, data je sledećim listingom.

```
package net.learn2develop.Services;
import java.net.MalformedURLException;
import java.net.URL;
import android.app.IntentService;
import android.content.Intent;
import android.util.Log;
public class MyIntentService extends IntentService {
    public MyIntentService() {
        super("MyIntentServiceName");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        try {
            int result =
                DownloadFile(new URL("http://www.amazon.com/somefile.pdf"));
            Log.d("IntentService", "Preuzeo je " + result + " bajtova");

            //---slanje poruke kojom se aktivnost obaveštava
            // da je datoteka preuzeta---
            Intent broadcastIntent = new Intent();
            broadcastIntent.setAction("FILE_DOWNLOADED_ACTION");
            getBaseContext().sendBroadcast(broadcastIntent);

        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
    }

    private int DownloadFile(URL url) {
        try {
            //---simulacija preuzimanja datoteke---
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return 100;
    }
}
```

PRIMER 9 - KLASA AKTIVNOSTI I ANDROIDMANIFEST.XML

Testiranje nije moguće bez glavne klase aktivnosti i deklarisanja servisa.

Da bi funkcionalnosti kreirane aplikacije bilo moguće testirati, neophodna je glavna klasa aktivnosti, čiji će još zadatak biti da učita i korisnički interfejs aplikacije. Sledi njen listing, uz napomenu da deo koda koji se ne koristi, u zavisnosti od scenarija primene, biće tretiran kao JAVA komentar.

```
package net.learn2develop.Services;

import java.net.MalformedURLException;
import java.net.URL;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;
import android.content.ComponentName;
import android.os.IBinder;
import android.content.ServiceConnection;

public class ServicesActivity extends Activity {
   IntentFilter intentFilter;
   MyService serviceBinder;
    Intent i;

    private ServiceConnection connection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            //---poziva se kada je konekcija napravljena---
            serviceBinder = ((MyService.MyBinder)service).getService();
            try {
                URL[] urls = new URL[] {
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf")};
                    //---dodela URLa servisu kroz serviceBinder objekat---
                    serviceBinder.urls = urls;
                } catch (MalformedURLException e) {
                    e.printStackTrace();
                }
                startService(i);
        }
        public void onServiceDisconnected(ComponentName className) {
```

```
            //---poziva se kada se servis zaustavlja---
            serviceBinder = null;
        }
    };

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

   @Override
    public void onResume() {
        super.onResume();

        //---filtriranje sadržaja za preuzimanje datoteke---
        intentFilter = new IntentFilter();
        intentFilter.addAction("FILE_DOWNLOADED_ACTION");

        //---registrovanje prijemnika---
        registerReceiver(intentReceiver, intentFilter);
    }

    @Override
    public void onPause() {
        super.onPause();

        //---uklanjanje prijemnika---
        unregisterReceiver(intentReceiver);
    }

    public void startService(View view) {
        //startService(new Intent(getBaseContext(), MyService.class));
        //OR
        //startService(new Intent("net.learn2develop.MyService"));
       // startService(new Intent(getBaseContext(), MyIntentService.class));

        /*
        Intent intent = new Intent(getBaseContext(), MyService.class);
        try {
            URL[] urls = new URL[] {
                    new URL("http://www.amazon.com/somefiles.pdf"),
                    new URL("http://www.wrox.com/somefiles.pdf"),
                    new URL("http://www.google.com/somefiles.pdf"),
                    new URL("http://www.learn2develop.net/somefiles.pdf")};
            intent.putExtra("URLs", urls);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        startService(intent);*/

        i = new Intent(ServicesActivity.this, MyService.class);
```

```
        bindService(i, connection, Context.BIND_AUTO_CREATE);
    }

    public void stopService(View view) {
        stopService(new Intent(getBaseContext(), MyService.class));
    }

  private BroadcastReceiver intentReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            Toast.makeText(getBaseContext(), "Datoteka je preuzeta!",
                    Toast.LENGTH_LONG).show();
        }
    };

}
```

Registrovanje servisa je obavezno AndroidManifest.xml datotekom čiji se kod prilaže sledećim listingom.

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

    <uses-sdk android:minSdkVersion="14" />

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

        <service android:name=".MyService">
            <intent-filter>
                <action android:name="net.learn2develop.MyService" />
            </intent-filter>
        </service>
        <service android:name=".MyIntentService" />
    </application>

</manifest>
```

PRIMER 10 - VEŽBANJE NITI U ANDROID APLIKACIJI

Vežbanje na računaru izrade zadatka sa nitima.

ZADATAK:

1. Kreirati Android aplikaciju u Android Studio IDE razvojnom okruženju koristeći API za Android 6.0;

2. Zadatak simulira odbrojavanje 10-0, a potom vraća Toast komentar polećemo;

3. Progres brojača realizovati primenom AsyncTask klase.

Slika 9.1 Izgled aplikacije

Grafički korisnički interfejs sa slike je realizovan sledećim listingom datoteke activity_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"
    android:background="@drawable/metpozadina">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="NitiDemo - primer!!!" />

    <Button
        android:id="@+id/btnStartCounter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="startCounter"
        android:text="Start" />

    <Button
```

```
        android:id="@+id/btnStopCounter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:onClick="stopCounter"
        android:text="Stop" />

    <TextView
        android:id="@+id/textView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Brojač" />

</LinearLayout>
```

Programska logika je realizovana sledećom JAVA klasom glavne aktivnosti, sa podrazumevanim nazivom MainActivity.java.

```
package com.metropolitan.nitidemo;

import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;


public class MainActivity extends AppCompatActivity {

    static TextView txtView1;
    //CountingThread thread;

    DoCountingTask task;


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

        txtView1 = (TextView) findViewById(R.id.textView1);
            }

    //--- Primer  ---
    public void startCounter(View view) {
        task = (DoCountingTask) new DoCountingTask().execute();

        }

    public void stopCounter(View view) {
        task.cancel(true);
    }
```

```
    private class DoCountingTask extends AsyncTask<Void, Integer, Void> {
        protected Void doInBackground(Void... params) {
            for (int i = 10; i >= 0; i--) {
                //---izveštaj o napretku---
                publishProgress(i);

                try {
                    Thread.sleep(1000);

                } catch (InterruptedException e) {
                    Log.d("Threading", e.getLocalizedMessage());
                }

                if (isCancelled()){
                    Toast.makeText(MainActivity.this, "Cool Ha?",
Toast.LENGTH_SHORT).show();
                    break;
                }


            }

            return null;
        }

        protected void onProgressUpdate(Integer... progress) {
            txtView1.setText(progress[0].toString());
            if(progress[0] == 0)
                Toast.makeText(MainActivity.this, "Polećemooo!!!", 0).show();
            Log.d("Threading", "ažuriranje...");
        }

    }
    @Override
    protected void onPause() {
        // TODO Auto-generated method stub
        super.onPause();
        stopCounter(txtView1);
    }
}
```

ZADATAK 8

Pokušajte sami!!!

Servis broji unazad od 10 do 0. Korisnik pokušava da odgovori na pitanje ponuđeno u glavnoj aktivnosti. Nakon isteka vremena, u Toast klasi se javlja informacije da li ste tačno ili netačno odgovorili na pitanje.

Poglavlje 10

Domaći zadatak 10

ZADATAK 1

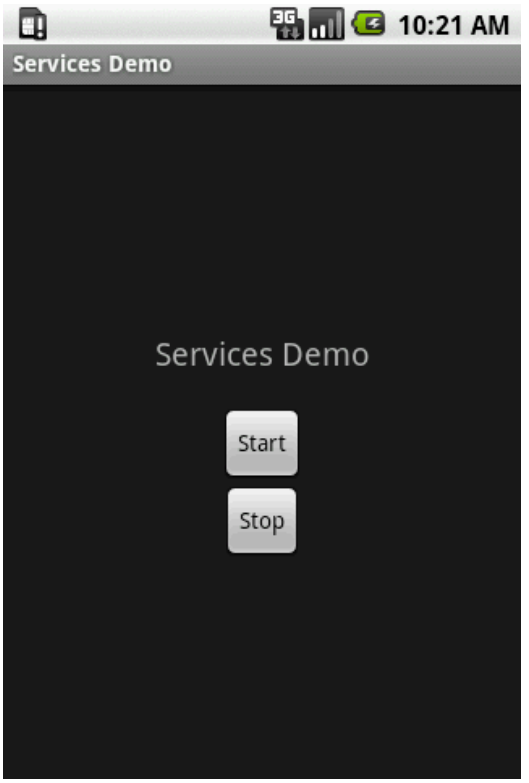
Vežbanje povezivanja aktivnosti sa servisima.

- Kreirati Android projekat MP3;
- Kreirati UI sa kontrolama za pokretanje i zaustavljanje servisa;
- Obezbediti asinhrono izvršavanje zadataka u servisu;
- Servis u pozadini pušta izabranu mp3 muziku;
- Obezbediti mehanizam da se nakon završene pesme, servis samostalno zaustavi.

Primer:

http://www.java2s.com/Code/Android/Media/Playmp3filewithinaservice.htm

Ideja za UI:



Slika 10.1 Ideja za UI

Zaključak

PREGLED LEKCIJE 10

Lekcijom su obrađeni Android servisi i njihovo povezivanje sa aktivnostima.

U ovoj lekciji je obrađeno kako se kreiraju servisi u okviru Android projekta i kako se u njima izvršavaju dugotrajni zadaci. Pokazano je nekoliko načina pomoću kojih je moguće obezbediti asinhrono izvršavanje zadataka u pozadini, a da pri tome ne dolazi do blokiranja aktivnosti koja inicira izvršavanje zadatka. Takođe, naučeno je kako se prosleđuju poruke u servis i koje su alternative za direktnije povezivanje aktivnosti i servisa.

U ovoj lekciji su posebno obrađene sledeće teme:

- Kreiranje servisa;
- Implementiranje metoda u servisu;
- Pokretanje servisa;
- Zaustavljanje servisa;
- Izvršavanje zadatka u posebnoj niti i automatsko zaustavljanje servisa;
- Komunikacija između aktivnosti i servisa;
- Povezivanje aktivnosti sa servisom;
- Ažuriranje korisničkog interfejsa iz *Runnable* bloka.

LITERATURA

Za pripremanje lekcije korišćena je sledeća literatura

1. Lee W. M. 2012. *Android 4 – razvoj aplikacija*, Wiley Publishing, INC
2. http://developer.android.com/training/index.html
3. http://www.tutorialspoint.com/android/
4. http://www.vogella.com/tutorials/android.html
5. Dietel P, Dietel H, Wald A. 2016. *Android* 6 for Programmers - An App Driven Approach, Dietel Developer Series