



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.



KI205 - JAVA 8: PROGRAMIRANJE U JAVI NA ANDROID PLATFORMI

Umrežavanje

Lekcija 09

PRIRUČNIK ZA STUDENTE

KI205 - JAVA 8: PROGRAMIRANJE U JAVI NA ANDROID PLATFORMI

Lekcija 09

UMREŽAVANJE

- ✓ Umrežavanje
- ✓ Poglavlje 1: Web servisi i HTTP protokol
- ✓ Poglavlje 2: Primena GET metode
- ✓ Poglavlje 3: Upotreba JSON servisa
- ✓ Poglavlje 4: Programiranje soketa
- ✓ Poglavlje 5: Domaći zadatak 9
- ✓ Pregled Lekcije10

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

▼ Uvod

UVOD

Lekcija će se baviti načinima preuzimanja tekstualnih i binarnih datoteka sa web servera.

U ovoj lekciji će cilj biti prikazivanje načina kako se koristi HTTP protokol za komuniciranje sa web serverima. Na taj način, kreirane Android aplikacije biće u mogućnosti da preuzimaju tekstualne i binarne podatke. Takođe, akcenat će biti na analizi XML datoteka sa ciljem preuzimanja relevantnih delova XML dokumenata. Pored XML web servisa biće opisan i **JSON (JAVA Script Object Notation)** kao jednostavnija alternativa za XML. Android SDK klase biće upotrebljene za manipulisanje *JSON* sadržajem.

Konačno, lekcija će se fokusirati na pisanje Android aplikacije za pristupanje serverima, primenom TSP soketa. Programiranjem soketa omogućeno je realizovanje sofisticiranih Android mrežnih aplikacija.

U ovoj lekciji biće razmotreni sledeći specifični zadaci:

- **Pristupanje web sadržaju pomoću HTTP protokola;**
- **Korišćenje XML web servisa;**
- **Korišćenje JSON web servisa;**
- **Pristupanje Socket serveru.**

Savladavanjem ove lekcije, studenti će biti u potpunosti obučeni da pristupaju web sadržaju pomoću HTTP protokola, sa koriste XML web servise, kao i JSON web servise. Poseban akcenat će biti na savladavanju korišćenja Socket - a i u tu svrhu biće razvijena i demonstrirana mala klijent - server **chat** mobilna aplikacija.

▼ Poglavlje 1

Web servisi i HTTP protokol

HTTP PROTOKOL

HTTP je veoma zaslužan za ekspanziju Interneta.

Aplikacije veoma često koriste **HTTP** (eng. HyperText Transfer Protocol) protokol za komuniciranje sa okruženjem. Pojam HTTP protokola je veoma poznat i on je zaslužan u velikoj meri za ekspanziju Interneta, budući da omogućava realizovanje velikog broja specifičnih zadataka. Koristeći HTTP moguće je obaviti brojne zadatke, među kojima je moguće istaći sledeće:

- Preuzimanje web stranica sa web servera;
- Preuzimanje binarnih podataka, i slično.

Za razumevanje upotrebe HTTP protokola u Android aplikacijama, biće uveden sledeći primer: *kreirati Android 6.0 projekat u kome se HTTP protokol koristi za pristupanje web sadržajima, a sa ciljem preuzimanja određenog sadržaja.*

Za razvoj traženog rešenja biće korišćen Android Studio IDE razvojno okruženje u kojem će biti kreirane sve datoteke koje se odnose na korisnički interfejs, podešavanja i dozvole i implementaciju logike aplikacije JAVA kodom. Kreiran projekat nosiće naziv *NetworkingDemo*. Sam programski kod i odgovarajuće funkcionalnosti biće detaljno elaboriran u narednom izlaganju.

HTTP PROTOKOL - PRIVILEGIJE

Neophodno je uvesti privilegiju da bi aplikacija mogla da pristupi web servisima.

Da bi bilo moguće pristupati web servisima, i generalno koristiti HTTP protokol, u Android aplikacijama, neophodno je u **AndroidManifest.xml** datoteci ugraditi odgovarajuće privilegije. Privilegijama je omogućeno da Android aplikacija direktno pristupa Internetu. Takođe, moguće je ugraditi još jednu privilegiju kojom bi aplikacija proveravala da li postoji Internet konekcija. Sledećim listingom je dat kod **AndroidManifest.xml** datoteke primera **NetworkingDemo** koji služi kao osnov za analizu i demonstraciju problematike ovog dela lekcije.

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

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

<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>
</application>

</manifest>
```

HTTP I KLASA AKTIVNOSTI

Klasa aktivnosti će implementirati metodu za rad sa web servisima.

Sledeći zadatak jeste kreiranje odgovarajuće klase aktivnosti. Ova klasa moraće da implementira metodu `OpenHttpConnection()` za otvaranje HTTP konekcije i preuzimanje URL stringa. Za početak, sledeće `import` naredbe je neophodno dodati u JAVA klasu `MainActivity.java`.

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

import java.io.IOException;
import java.io.InputStream;

import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import android.util.Log;
```

U nastavku, neophodno je prikazati listing klase `MainActivity.java` koja učitava korisnički interfejs, upravlja otvaranjem konekcije kroz implementaciju metode `OpenHttpConnection()` i omogućava preuzimanje URL stringa. Sledećim listingom prikazan je kod glavne klase aktivnosti.

```
public class NetworkingActivity extends Activity {

    ImageView img;

    private InputStream OpenHttpConnection(String urlString)
```

```
throws IOException
{
    InputStream in = null;
    int response = -1;

    URL url = new URL(urlString);
    URLConnection conn = url.openConnection();

    if (!(conn instanceof HttpURLConnection))
        throw new IOException("Nema HTTP Konekcije");
    try{
        HttpURLConnection httpConn = (HttpURLConnection) conn;
        httpConn.setAllowUserInteraction(false);
        httpConn.setInstanceFollowRedirects(true);
        httpConn.setRequestMethod("GET");
        httpConn.connect();
        response = httpConn.getResponseCode();
        if (response == HttpURLConnection.HTTP_OK) {
            in = httpConn.getInputStream();
        }
    }
    catch (Exception ex)
    {
        Log.d("Networking", ex.getLocalizedMessage());
        throw new IOException("Greška u povezivanju");
    }
    return in;
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
}
```

HTTP - FUNKCIONISANJE

Primenom `InputStream` objekta moguće je pokrenuti preuzimanje podataka sa servera

Pošto primer koristi HTTP protokol za povezivanje na web, aplikacija mora da ima odgovarajuću dozvolu za obavljanje ovog zadatka. Ta dozvola je ugrađena u `AndroidManifest.xml` datoteku kao sledeća instrukcija:

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

Po davanju tražene dozvole, kreirana je metoda `OpenHttpConnection()` koja preuzima URL string, a kao rezultat vraća objekat tipa `InputStream`. Primenom ovog objekta omogućeno je preuzimanje podataka učitavanjem pojedinačnih bajtova iz *stream* objekta. Metoda, dalje, koristi objekat tipa `HttpURLConnection` za uspostavljanje veze sa udaljenom lokacijom koja je određena URL adresom. Definisano je svojstvo konekcije *request* metoda (sledeći listing izdvojen iz glavne klase aktivnosti).

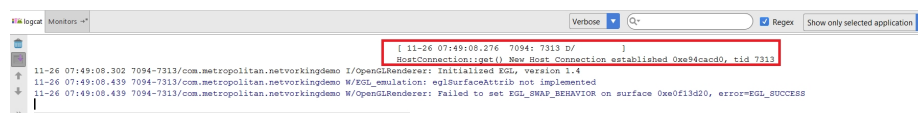
```
HttpURLConnection httpConn = (HttpURLConnection) conn;
httpConn.setAllowUserInteraction(false);
httpConn.setInstanceFollowRedirects(true);
httpConn.setRequestMethod("GET");
```

Nakon pokušaja uspostavljanja veze sa serverom, vraća se HTTP kod odgovora. Ako je konekcija uspostavljena (*HTTP_OK*), nastavlja se učitavanje `InputStream` objekta koji se odnosi na konekciju (sledeći listing izdvojen iz glavne klase aktivnosti).

Primenom `InputStream` objekta sada je moguće pokrenuti preuzimanje podataka sa servera.

```
httpConn.connect();
response = httpConn.getResponseCode();
if (response == HttpURLConnection.HTTP_OK) {
    in = httpConn.getInputStream();
}
```

Za sam početak je moguće izvršiti testiranje ove jednostavne aplikacije i pratiti u LogCat prozoru informacije u vezi sa njenim izvršavanjem. Na dobro poznati način, izborom opcije Run App ili Shift + F10, u Android Studio IDE razvojnom okruženju biće debugovana i pokrenuta aplikacija. Rezultat izvršavanja je moguće prikazati sledećom slikom.



Slika 1.1 Realizovana konekcija - logcat

PRIMER 1 - PREUZIMANJE BINARNIH PODATAKA

Preuzimanje binarnih datoteka je čest zadatak Android aplikacija.

U Android aplikacijama, kao česta funkcija, javlja se preuzimanje binarnih datoteka sa web servera. Aplikacija može da preuzme sliku, pesmu, film i slično. Sledeći primer će pokazati kako je to moguće realizovati oslanjajući se na već urađen deo programskog koda.

Ono što do sad nije učinjeno jeste kreiranje jednostavnog korisničkog interfejsa kojim će biti omogućeno prikazivanje prezetog binarnog sadržaja sa *weba*. Pa u tom svetlu neka GUI datoteka, sa podrazumevanim nazivom `activity_main.xml` ima XML kod priložen 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" >

        <ImageView
            android:id="@+id/img"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center" />

    </LinearLayout>

```

Klasa aktivnosti će biti proširena novim metodama, a imaće i jednu unutrašnju klasu za realizovanje priloženih softverskih zahteva. Takođe, klasa je dopunjena i **ImageView** objektom čija deklaracija predstavlja prvu naredbu u klasi. Dodatni uključeni paketi sa odgovarajućim klasama i deklaracija **ImageView** objekta, prikazani su sledećim listingom izolovanim iz klase aktivnosti primera.

```

import android.widget.ImageView;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;

public class MainActivity extends AppCompatActivity {

    ImageView img;
    ....
}

```

PROŠIRENJE KLASA AKTIVNOSTI

Za preuzimanje slika sa weba biće angažovani dodatna metoda i ugnježdjena klasa.

Klasa aktivnosti, koja omogućava preuzimanje binarnih datoteka sa weba, mora biti proširena odgovarajućim metodama. Nakon metode **OpenHTTPConnection()** koja je već ugrađena u klasu, uvodi se nova metoda **DownloadImage()** čiji zadatak jeste upravo preuzimanje binarne datoteke koja odgovara slici sa Interneta, a čije su definicija i logika dati sledećim listingom izolovanim iz glavne klase aktivnosti.

```

private Bitmap DownloadImage(String URL)
{
    Bitmap bitmap = null;
    InputStream in = null;
    try {
        in = OpenHttpConnection(URL);
        bitmap = BitmapFactory.decodeStream(in);
        in.close();
    } catch (IOException e1) {
        Log.d("NetworkingActivity", e1.getLocalizedMessage());
    }
}

```

```
    }  
    return bitmap;  
}
```

Poseban zadatak ima ugnježena klasa `DownloadImageTask` čiji je kod prikazan sledećim listingom i o kojoj će u najskorijem izlaganju biti diskutovano. Takođe, definicija metode `onCreate()` je dopunjena objektom `DownloadImageTask` koji ukazuje na URL sa koga se preuzima sadržaj.

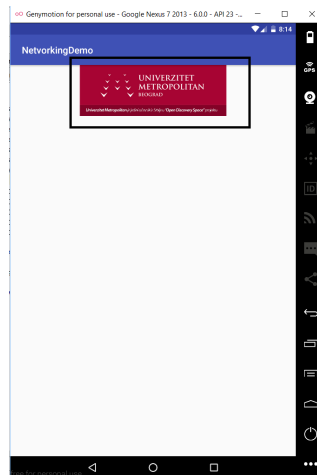
```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {  
    protected Bitmap doInBackground(String... urls) {  
        return DownloadImage(urls[0]);  
    }  
  
    protected void onPostExecute(Bitmap result) {  
        ImageView img = (ImageView) findViewById(R.id.img);  
        img.setImageBitmap(result);  
    }  
}  
  
/** Called when the activity is first created. */  
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    new DownloadImageTask().execute(  
        "http://ods.metropolitan.ac.rs/wp-content/uploads/2014/09/"  
        + "univerzitet-metropolitan.png");  
}  
  
} // kraj klase aktivnosti
```

PREUZIMANJE BINARNE DATOTEKE - FUNKCIONISANJE

Za preuzimanje slike i njeno prikazivanje u aktivnosti izvršava se metoda `DownloadImage()`.

Sledeći zadatak je proveravanje ispravnosti ugrađenih novih funkcionalnosti u kreiranu Android aplikaciju. To znači da je, korišćenjem razvojnog okruženja Android Studio, neophodno ponovo prevesti i pokrenuti aplikaciju izabranim emulatorom ili realnim mobilnim uređajem.

Klikom na Shift + F10 ili izborom opcije Run App pokreće se program koji preuzima sliku sa date lokacije i prikazuje je u `ImageView` pogledu (sledeća slika).



Slika 1.2 Preuzeta binarna datoteka sa Interneta

Metoda `DownloadImage()` preuzima URL adresu slike koja će biti prebačena na mobilni uređaj. Nakon toga, uspostavlja konekciju sa serverom, koristeći prethodno definisanu metodu `OpenHttpConnection()`. Dalje, metoda `decodeStream()` `BitmapFactory` klase, koristeći kreirani objekat tipa `InputStream`, preuzima i prevodi (dekodira) podatke u `Bitmap` objekat. Metoda `DownloadImage()`, upravo, vraća objekat tipa `Bitmap`.

Za preuzimanje slike i njeno prikazivanje u aktivnosti izvršava se metoda `DownloadImage()`. U aktuelnim verzijama Android operativnog sistema, sinhronne operacije ne mogu da se direktno izvršavaju iz niti namenjene korisničkom interfejsu. Ukoliko se pokuša direktno izvršavanje metode `DownloadImage()`, aplikacija će prekinuti izvršavanje. Sledeći kod bi doveo do prekida izvršavanja aplikacije.

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

    Bitmap bitmap =
        DownloadImage("http://www.mayoff.com/5-01cablecarDCP01934.jpg");
    img = (ImageView) findViewById(R.id.img);
    img.setImageBitmap(bitmap);
}
```

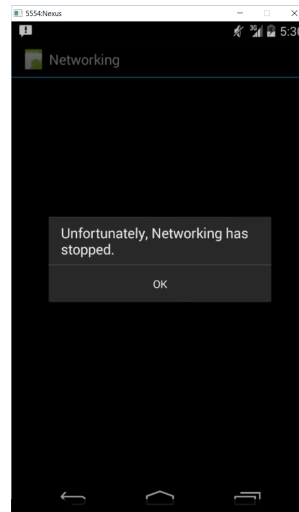
Slika 1.3 Direktan poziv sinhronne metode

PREUZIMANJE BINARNE DATOTEKE – ASYNCTASK KLASA

Direktno izvršavanje sinhronih metoda, iz UI niti, nije dozvoljeno u aktuelnim Android verzijama.

Ukoliko se pokrene aplikacija sa prethodno definisanom sinhronom metodom `DownloadImage()` metodom doći će do otkazivanja programa (sledeća slika). Razlog je veoma jednostavan. Pošto je `DownloadImage()` sinhrona metoda, njeno direktno izvršavanje će se

desiti u glavnoj niti aplikacije koja je rezervisana za glavnu aktivnost, odnosno korisnički interfejs aplikacije. Na ovaj način glavna aktivnost će biti blokirana.



Slika 1.4 Otkazivanje aplikacije

U novijim verzijama operativnog sistema, sinhroni kod mora da bude definisan pomoću `AsyncTask` klase. Ova klasa omogućava izvršavanje zadataka u pozadini, u posebnoj niti, a zatim vraćanje rezultata izvršavanja niti koja definiše korisnički interfejs. Na ovaj način se izvršavaju operacije u pozadini, bez potrebe za upravljanjem složenim problemima koji se odnose na upravljanje nitima.

Navedeni zadaci su prepušteni klasi `DownloadImageTask` koja nasleđuje osnovnu klasu `AsyncTask` i implementira metode `doInBackground()` i `onPostExecute()`. Kod koji se izvršava asinhrono postavlja se u metodu `doInBackground()`. Nakon što je zadatak obavljen, metodom `onPostExecute()` vraća se rezultat, a to je u ovom slučaju `ImageView` za prikazivanje slike. Navedeno je prikazano sledećim listingom.

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
        return DownloadImage(urls[0]);
    }

    protected void onPostExecute(Bitmap result) {
        ImageView img = (ImageView) findViewById(R.id.img);
        img.setImageBitmap(result);
    }
}
```

Na kraju, u `onCreate()` metodi, kreiran je `DownloadImageTask` objekat koji izvršava metodu `execute()`, prosleđujući joj URL adresu slike koja se preuzima.

```
new DownloadImageTask().execute(
    "http://ods.metropolitan.ac.rs/wp-content/uploads/2014/09/"
    + "univerzitet-metropolitan.png");
```

PREUZIMANJE SERIJE BINARNIH DATOTEKA

Asinhroni zadaci mogu dugo da traju.

Ponekada je neophodno u seriji izvršiti preuzimanje izvesnog binarnog sadržaja sa Interneta. Ovaj zadatak je, takođe, moguće asinhrono izvršiti u Android aplikacijama na lak i elegantan način.

Ukoliko je cilj da Android aplikacija asinhrono preuzme seriju slika, neophodno je izvršiti modifikaciju `DownloadImageTask` klase. Klasa će sadržati metodu `onProgressUpdate()`. Pošto asinhroni zadaci mogu dugo da traju, biće ugrađena i metoda `publishProgress()` koja će pratiti trenutno stanje izvršavanja operacije, a to će inicirati izvršavanje `onProgressUpdate()` metode – prikazivanje slike preuzete sa weba. Metoda `onProgressUpdate()` izvršava se u niti korisničkog interfejsa i potpuno bezbedno se ažurira `ImageView` prikazivanjem bitmapa preuzetih sa web servera. Sledećim listingom je prikazan kod klase naslednice `AsyncTask` klase zadužene za implementaciju pomenute logike.

```
private class DownloadImageTask extends AsyncTask <String, Bitmap, Long> {
    ///---preuzimanje liste URL adresa slika---
    protected Long doInBackground(String... urls) {
        long imagesCount = 0;
        for (int i = 0; i < urls.length; i++) {
            ///---preuzimanje slike---
            Bitmap imageDownloaded = DownloadImage(urls[i]);
            if (imageDownloaded != null) {
                ///---uvećanje rednog broja slike---
                imagesCount++;
                try {
                    ///---pauza 3s---
                    Thread.sleep(3000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                ///---vraćanje preuzete slike---
                publishProgress(imageDownloaded);
            }
        }
        ///---vraćanje ukupnog broja preuzetih slika---
        return imagesCount;
    }

    ///---prikazivanje preuzete slike---
    protected void onProgressUpdate(Bitmap... bitmap) {
        img.setImageBitmap(bitmap[0]);
    }

    ///---sve slike su preuzete---
    protected void onPostExecute(Long imagesDownloaded) {
        Toast.makeText(getBaseContext(),
            "Ukupno je " + imagesDownloaded +
            " slika primljeno" ,

```

```

        Toast.LENGTH_LONG).show();
    }
}

```

PREUZIMANJE SERIJE BINARNIH DATOTEKA – ONCREATE() METODA

URL lista za preuzimanje binarnih datoteka smeštena je u metodu onCreate() klase aktivnosti.

Sledeći zadatak je testiranje modifikovane aplikacije i demonstracija preuzimanja niza binarnih datoteka, u ovom slučaju slika sa Interneta i njihovo prikazivanje, sekvencijalno, u **ImageView** pogledu kreiranog korisničkog interfejsa.

Da bi niz slika asinhrono bio preuzet u pozadini, neophodno je kreirati objekat klase **DownloadImageTask** i izvršiti njenu **execute()** metodu. Ovo se dešava u metodi **onCreate()** klase aktivnosti (prikazano sledećim listingom).

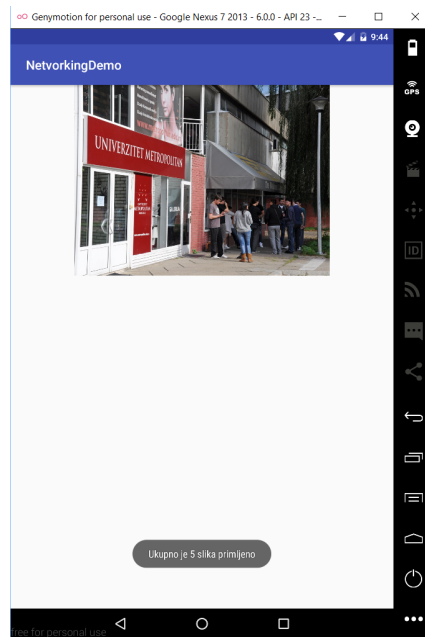
```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //serija slika za preuzimanje
    img = (ImageView) findViewById(R.id.img);
    new DownloadImageTask().execute(
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-01.jpg",
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-03.jpg",
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-10.jpg",
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-28.jpg",
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-28.jpg"
    );
}

```

Sada je moguće proveriti rezultate prethodnog rada. U Android Studio IDE razvojnom okruženju, neophodno je ponovo prevesti program, klikom na Run App ili Shift + F10, pokreće se program, prikazuje seriju preuzetih slika kao **ImageView** i, na kraju, vraća ukupan broj preuzetih slika. Navedeno je prikazano sledećom slikom.



Slika 1.5 Prikaz serije slika

PRIMER 2 - PREUZIMANJE TEKSTUALNOG SADRŽAJA

Sa web servera je moguće preuzeti i tekstualne podatke.

Pored binarnih podataka, sa web servera je moguće preuzeti i tekstualne podatke. Na primer, može se pristupiti web servisu koji vraća string sa slučajno izabranim citatima. Tekući primer će biti iskorišćen tako da će sve metode i ugnježdene klase koje se odnose za preuzimanje binarnog sadržaja sa weba, biće zamenjeni odgovarajućim metodama i ugnježdenom klasom za preuzimanje tekstualnih podataka. Metoda koja će biti uvedena je `DownloadText()`, koja vraća podatak klase `String`, dok će `onCreate()`, pored standardnih zadataka, dobiti novu logiku. Klasa koja će biti ugnježdjena u klasu aktivnosti zvaće se `DownloadTextTask`. Nepohodno je izvršiti sledeći uvoz: `import java.io.InputStreamReader` na postojeći skup uključenih klasa. Kod kojim će biti modifikovana glavna klasa aktivnosti, a u svetlu prethodnog izlaganja, prikazan je sledećim listingom.

```
private String DownloadText(String URL)
{
    int BUFFER_SIZE = 2000;
    InputStream in = null;
    try {
        in = OpenHttpConnection(URL);
    } catch (IOException e) {
        Log.d("NetworkingActivity", e.getLocalizedMessage());
        return "";
    }

    InputStreamReader isr = new InputStreamReader(in);
    int charRead;
    String str = "";
```

```

        char[] inputBuffer = new char[BUFFER_SIZE];
        try {
            while ((charRead = isr.read(inputBuffer))>0) {
                //--convert the chars to a String--
                String readString =
                    String.valueOf(inputBuffer, 0, charRead);
                str += readString;
                inputBuffer = new char[BUFFER_SIZE];
            }
            in.close();
        } catch (IOException e) {
            Log.d("NetworkingActivity", e.getLocalizedMessage());
            return "";
        }
        return str;
    }

    private class DownloadTextTask extends AsyncTask<String, Void, String> {

        protected String doInBackground(String... urls) {
            return DownloadText(urls[0]);
        }

        @Override
        protected void onPostExecute(String result) {
            Toast.makeText(getBaseContext(), result, Toast.LENGTH_LONG).show();
        }
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //--download text--
        new DownloadTextTask().execute ("http://www.njegoss.org/petrovics/
gvijenac.htm#posveta"+
"/random?max_characters=256&";
        }
    }

} //kraj klase aktivnosti

```

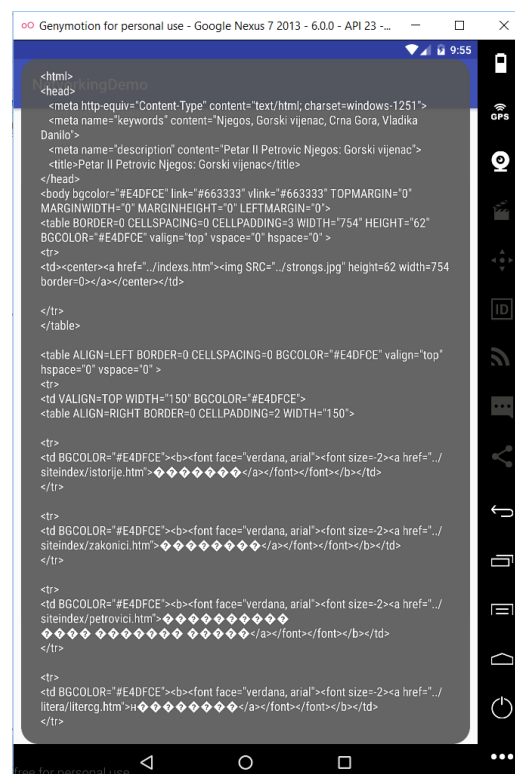
PREUZIMANJE TEKSTUALNOG SADRŽAJA - DEMONSTRACIJA

InputStream objekat se koristi za učitavanje svakog pojedinačnog karaktera iz toka i njegovo pakovanje u String objekat.

Metoda `DownloadText()` preuzima URL adresu tekstualne datoteke koju treba preuzeti na mobilni uređaj, ili emulator, a potom vraća kao rezultat *String* koji predstavlja naziv preuzete tekstualne datoteke. Konkretno, dolazi do uspostavljanja HTTP konekcije sa serverom, a zatim se upošljava objekat tipa *InputStream* za učitavanje svakog pojedinačnog karaktera iz toka i njegovo pakovanje u *String* objekat. Ovde se, takođe, koristi izvedena klasa tipa `AsyncTask` sa ciljem asinhronog izvršavanja metode `DownloadText()`. U suprotnom bi sinhrono izvršavanje blokiralo korisnički interfejs i došlo bi do otkazivanja programa.

Da bi rezultati manipulacije sa klasom aktivnosti, u novom kontekstu, bili vidljivi, neophodno je ponovo izvršiti prevođenje i pokretanje programa u Android Studio IDE razvojnom okruženju.

Pokreće se program i prikazuje preuzeti string sa web servera (sledeća slika).



Slika 1.6 Preuzeti tekst sa web servera

Izlaganje u ovom delu lekcije je moguće zaokružiti video materijalom kojim je podvučen značaj `AsyncTask` klase kod preuzimanja web sadržaja Android aplikacijom.

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

PRIMER 3 - VEŽBANJE PREUZIMANJA BINARNIH I TEKSTUALNIH DATOTEKA

Vežbanje na računaru preuzimanja sadržaja preko web servera

ZADATAK:

1. Kreirati Android aplikaciju koja koristi web servise i HTTP protokol za preuzimanje binarnih i tekstualnih datoteka.
2. Obezbediti Internet privilegiju aplikaciji

Sledećim listingom je dat AndroidManifest.xml sa Internet privilegijom.

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

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

<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>
</application>

</manifest>
```

Jednostavan korisnički Interfejs, sa ugrađenim logom našeg Univerziteta kao pozadinom, može biti prikazan 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"
    android:background="@drawable/metpozadina">

    <ImageView
        android:id="@+id/img"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />

</LinearLayout>
```

Glavna klasa aktivnosti sa asinhronom podrškom izvršavanja zadataka, priložena je sledećim kodom.

```
package com.metropolitan.networkingdemo;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.widget.ImageView;
import android.widget.Toast;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

public class MainActivity extends AppCompatActivity {

    ImageView img;

    private InputStream OpenHttpConnection(String urlString)
        throws IOException
    {
        InputStream in = null;
        int response = -1;

        URL url = new URL(urlString);
        URLConnection conn = url.openConnection();

        if (!(conn instanceof HttpURLConnection))
            throw new IOException("Nema HTTP Konekcije");
        try{
            HttpURLConnection httpConn = (HttpURLConnection) conn;
            httpConn.setAllowUserInteraction(false);
            httpConn.setInstanceFollowRedirects(true);
            httpConn.setRequestMethod("GET");
            httpConn.connect();
            response = httpConn.getResponseCode();
            if (response == HttpURLConnection.HTTP_OK) {
                in = httpConn.getInputStream();
            }
        }
    }
}
```

```

        catch (Exception ex)
        {
            Log.d("Networking", ex.getLocalizedMessage());
            throw new IOException("Greška u povezivanju");
        }
        return in;
    }

    private Bitmap DownloadImage(String URL)
    {
        Bitmap bitmap = null;
        InputStream in = null;
        try {
            in = OpenHttpConnection(URL);
            bitmap = BitmapFactory.decodeStream(in);
            in.close();
        } catch (IOException e1) {
            Log.d("NetworkingActivity", e1.getLocalizedMessage());
        }
        return bitmap;
    }

    /* private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
        protected Bitmap doInBackground(String... urls) {
            return DownloadImage(urls[0]);
        }

        protected void onPostExecute(Bitmap result) {
            ImageView img = (ImageView) findViewById(R.id.img);
            img.setImageBitmap(result);
        }
    }*/

    /* private class DownloadImageTask extends AsyncTask <String, Bitmap, Long> {
        //---preuzimanje liste URL adresa slika---
        protected Long doInBackground(String... urls) {
            long imagesCount = 0;
            for (int i = 0; i < urls.length; i++) {
                //---preuzimanje slike---
                Bitmap imageDownloaded = DownloadImage(urls[i]);
                if (imageDownloaded != null) {
                    //---uvećanje rednog broja slike---
                    imagesCount++;
                    try {
                        //---pauza 3s---
                        Thread.sleep(3000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    //---vraćanje preuzete slike---
                    publishProgress(imageDownloaded);
                }
            }
        }
    }*/

```

```

        }
    }
    //---vraćanje ukupnog broja preuzetih slika---
    return imagesCount;
}

//---prikazivanje preuzete slike---
protected void onProgressUpdate(Bitmap... bitmap) {
    img.setImageBitmap(bitmap[0]);
}

//---sve slike su preuzete---
protected void onPostExecute(Long imagesDownloaded) {
    Toast.makeText(getBaseContext(),
        "Ukupno je " + imagesDownloaded +
        " slika primljeno" ,
        Toast.LENGTH_LONG).show();
}
}*/

/* private String DownloadText(String URL)
{
    int BUFFER_SIZE = 2000;
    InputStream in = null;
    try {
        in = OpenHttpConnection(URL);
    } catch (IOException e) {
        Log.d("NetworkingActivity", e.getLocalizedMessage());
        return "";
    }

    InputStreamReader isr = new InputStreamReader(in);
    int charRead;
    String str = "";
    char[] inputBuffer = new char[BUFFER_SIZE];
    try {
        while ((charRead = isr.read(inputBuffer))>0) {
            //---convert the chars to a String---
            String readString =
                String.valueOf(inputBuffer, 0, charRead);
            str += readString;
            inputBuffer = new char[BUFFER_SIZE];
        }
        in.close();
    } catch (IOException e) {
        Log.d("NetworkingActivity", e.getLocalizedMessage());
        return "";
    }
    return str;
}

private class DownloadTextTask extends AsyncTask<String, Void, String> {

```

```

        protected String doInBackground(String... urls) {
            return DownloadText(urls[0]);
        }

        @Override
        protected void onPostExecute(String result) {
            Toast.makeText(getBaseContext(), result, Toast.LENGTH_LONG).show();
        }
    }*/

    private String WordDefinition(String word) {
        InputStream in = null;
        String strDefinition = "";
        try {
            in = OpenHttpConnection(
                "http://services.aonaware.com/DictService/"
                + "DictService.asmx/Define?word=" + word);
            Document doc = null;
            DocumentBuilderFactory dbf =
                DocumentBuilderFactory.newInstance();
            DocumentBuilder db;
            try {
                db = dbf.newDocumentBuilder();
                doc = db.parse(in);
            } catch (ParserConfigurationException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            doc.getDocumentElement().normalize();

            //--prihvatanje svih <Definition> elemenata--
            NodeList definitionElements =
                doc.getElementsByTagName("Definition");

            //--iteracija kroz svaki <Definition> element--
            for (int i = 0; i < definitionElements.getLength(); i++) {
                Node itemNode = definitionElements.item(i);
                if (itemNode.getNodeType() == Node.ELEMENT_NODE)
                {
                    //--konverzija Definition čvora u Element--
                    Element definitionElement = (Element) itemNode;

                    //--preuzimanje svih <WordDefinition> elemenata iz
                    // <Definition> elemenata--
                    NodeList wordDefinitionElements =
                        (definitionElement).getElementsByTagName(
                            "WordDefinition");
                }
            }
        }
    }

```

```

        strDefinition = "";
        ///---iteracija kroz svaki <WordDefinition> element---
        for (int j = 0; j < wordDefinitionElements.getLength(); j++) {
            ///---konverzija <WordDefinition> Ā?vora u Element---
            Element wordDefinitionElement =
                (Element) wordDefinitionElements.item(j);

            ///---pruzimanje izvedenih ĉvorova iz
            /// <WordDefinition> elementa---
            NodeList textNodes =
                ((Node) wordDefinitionElement).getChildNodes();
            strDefinition +=
                ((Node) textNodes.item(0)).getNodeValue() + ". \n";
        }
    }
} catch (IOException e1) {
    Log.d("NetworkingActivity", e1.getLocalizedMessage());
}
///---vraćanje definicije reči---
return strDefinition;
}

private class AccessWebServiceTask extends AsyncTask<String,
    Void, String> {
    protected String doInBackground(String... urls) {
        return WordDefinition(urls[0]);
    }

    protected void onPostExecute(String result) {
        Toast.makeText(getBaseContext(), result,
            Toast.LENGTH_LONG).show();
    }
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    /* new DownloadImageTask().execute(
        "http://ods.metropolitan.ac.rs/wp-content/uploads/2014/09/"
        + "univerzitet-metropolitan.png");*/
    //http://www.mayoff.com/5-01cablecarDCP01934.jpg*/

    /* //serija slika za preuzimanje
    img = (ImageView) findViewById(R.id.img);
    new DownloadImageTask().execute(
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-01.jpg",
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-03.jpg",

```

```

        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-10.jpg",
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-28.jpg",
        "http://www.metropolitan.edu.rs/files/fotografije/galerija/"
        + "okruzenje-univerziteta/Okru%C5%BEenje-Univerziteta-28.jpg"
    );*/

    /* ---download text---
    new DownloadTextTask().execute(
        "http://www.njegoss.org/petrovics/gvijenac.htm#posveta"
        +
        "/random?max_characters=256&";
    );

    //---download an image---
    //---code will not run in Android 3.0 and beyond---
    /* Bitmap bitmap =
        DownloadImage("http://www.mayoff.com/5-01cablecarDCP01934.jpg");
    img = (ImageView) findViewById(R.id.img);
    img.setImageBitmap(bitmap);*/

    /*
    img = (ImageView) findViewById(R.id.img);
    new DoBackgroundTask().execute(
        "http://www.mayoff.com/5-01cablecarDCP01934.jpg",
        "http://www.hartiesinfo.net/greybox/Cable_Car_Hartbeespoort.jpg",
        "http://mcmanuslab.ucsf.edu/sites/default/files/imagepicker/m/
mmcmmanus/CaliforniaSanFranciscoPaintedLadiesHz.jpg",
        "http://www.fantom-xp.com/wallpapers/63/San_Francisco_-_Sunset.jpg",
        "http://travel.roro44.com/europe/france/Paris_France.jpg",
        "http://designheaven.files.wordpress.com/2010/04/
eiffel_tower_paris_france.jpg",
        "http://wwp.greenwichmeantime.com/time-zone/usa/nevada/las-vegas/
hotel/the-strip/paris-las-vegas/paris-las-vegas-hotel.jpg");
    */

    /*
    new DownloadTextTask()
        .execute("http://iheartquotes.com/api/v1/
random?max_characters=256&";
    );

    //---pristup web servisu primenom GET---
    new AccessWebServiceTask().execute("Belgrade");

}

}

```


ZADATAK 1

Pokušajte sami!!!

Pokušajte da samostalno kreirate Android aplikaciju sa funkcionalnostima preuzimanja binarnih i tekstualnih podataka.

▼ Poglavlje 2

Primena GET metode

UVOD U PRIMENU GET METODE

Preuzimanje XML datoteka sa web servera je čest zadatak Android aplikacija.

U dosadašnjem izlaganju, naučeno je kako se preuzimaju binarne datoteke, u konkretnom slučaju slike, i tekst sa weba. Međutim, često se sreće potreba da se preuzmu XML datoteke i da se analizira njihov sadržaj. Iz navedenog razloga, neophodno je naučiti kako je moguće izvršiti konekciju sa web serverom primenom *HTTP GET metode*.

Nakon što izvesni web servis vrati rezultat, u formi XML koda, biće omogućeno preuzimanje relevantnih delova i prikazivanje sadržaja. Za prikazivanje željenog sadržaja biće upotrebljena *Toast* klasa. Navedeni zahtevi tražiće i ponovno modifikovanje klase aktivnosti polaznog primera, nakon čega će ponovo uslediti prevođenje i testiranje aplikacije sa inoviranim kodom.

PRISTUP WEB SERVISU

Preuzimanje zahteva web metodom je strogo formatirano.

Za demonstraciju pristupa web servisima i korišćenja GET metode biće iskorišćena web metoda sa `http://services.aonaware.com/DictService/DictService.asmx/Define?adrese`. Ova web metoda je sastavni deo web servisa kojim je realizovan web rečnik. Zadavanjem određene reči, servis vraća njenu definiciju koja se nakon toka prezentuje korisniku koji je i zadao ključnu reč.

Web metoda preuzima zahtev u sledećem formatu.

```
GET /DictService/DictService.asmx/Define?word=string HTTP/1.1
Host: services.aonaware.com

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

Slika 2.1 Format GET zahteva

Da bi definicija reči bila dobijena, neophodno je uspostaviti HTTP konekciju i izvršiti web metodu, a zatim i analizu dobijenog XML koda. Odgovor ima sledeći format.

```
<?xml version="1.0" encoding="utf-8"?>
<WordDefinition xmlns="http://services.aonaware.com/webservices/">
  <Word>string</Word>
  <Definitions>
    <Definition>
      <Word>string</Word>
      <Dictionary>
        <Id>string</Id>
        <Name>string</Name>
      </Dictionary>
      <WordDefinition>string</WordDefinition>
    </Definition>
    <Definition>
      <Word>string</Word>
      <Dictionary>
        <Id>string</Id>
        <Name>string</Name>
      </Dictionary>
      <WordDefinition>string</WordDefinition>
    </Definition>
  </Definitions>
</WordDefinition>
```

Slika 2.2 Format odgovora

PRIMER 4 - KORIŠĆENJE WEB SERVISA

U klasi aktivnosti se definišu metode i ugnježdjena klasa.

Za angažovanje GET metode i prihvatanje xml podataka sa navedenog web servisa, neophodno je izvršiti nove modifikacije na tekućem primeru **NetworkingDemo**. Za početak, potrebno je uvesti još neke *import* instrukcije koje su priložene sledećim listingom.

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
```

Nakon obavljenog uvoza neophodnih paketa i klasa, potrebno je tekuće metode i ugnježdenu klasu zameniti novim. Sledećim kodom data je ugnježdjena klasa **AccessWebServiceTask** i prilagođena **onCreate()** metoda. Ovaj kod će, sada, biti sastavni deo inovirane **MainActivity.java** klase. Navedeni kod je priložen u formi sledećih listinga. Prvo sledi metoda **onCreate()**.

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    new AccessWebServiceTask().execute("Belgrade");
}
```

U sledećem koraku sledi klasa **AccessWebServiceTask** naslednica **AsyncTask** klase.

```
private class AccessWebServiceTask extends AsyncTask<String,
    Void, String> {
    protected String doInBackground(String... urls) {
        return WordDefinition(urls[0]);
    }

    protected void onPostExecute(String result) {
        Toast.makeText(getBaseContext(), result,
            Toast.LENGTH_LONG).show();
    }
}
```

METODA WORDDEFINITION()

WordDefinition() metodom se uspostavlja HTTP konekcija sa web servisom i prosleđuje reč za čiju definiciju postoji interesovanje.

Neposredno pre ugnježdene klase **AccessWebServiceTask** biće implementirana metoda **WordDefinition** čiji će zadatak biti uspostavljanje HTTP konekcije i prosleđivanje reči od interesa. Upravo, kreiranje ove metode predstavlja najobimniji i najzahtevniji zadatak tokom procesa modifikacije klase aktivnosti za demonstraciju primene GET metode. Kod ove metode, izolovan iz klase aktivnosti, priložen je sledećim listingom.

```
private String WordDefinition(String word) {
    InputStream in = null;
    String strDefinition = "";
    try {
        in = OpenHttpConnection(
            "http://services.aonaware.com/DictService/"
            + "DictService.asmx/Define?word=" + word);
        Document doc = null;
        DocumentBuilderFactory dbf =
            DocumentBuilderFactory.newInstance();
        DocumentBuilder db;
        try {
            db = dbf.newDocumentBuilder();
            doc = db.parse(in);
        } catch (ParserConfigurationException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        doc.getDocumentElement().normalize();

        //--prihvatanje svih <Definition> elemenata--
        NodeList definitionElements =
            doc.getElementsByTagName("Definition");
    }
```

```
//---iteracija kroz svaki <Definition> element---
for (int i = 0; i < definitionElements.getLength(); i++) {
    Node itemNode = definitionElements.item(i);
    if (itemNode.getNodeType() == Node.ELEMENT_NODE)
    {
        //---konverzija Definition čvora u Element---
        Element definitionElement = (Element) itemNode;

        //---preuzimanje svih <WordDefinition> elemenata iz
        // <Definition> elementa---
        NodeList wordDefinitionElements =
            (definitionElement).getElementsByTagName(
                "WordDefinition");
        strDefinition = "";
        //---iteracija kroz svaki <WordDefinition> element---
        for (int j = 0; j < wordDefinitionElements.getLength(); j++) {
            //---konverzija <WordDefinition> čvora u Element---
            Element wordDefinitionElement =
                (Element) wordDefinitionElements.item(j);

            //---pruzimanje izvedenih čvorova iz
            // <WordDefinition> elementa---
            NodeList textNodes =
                ((Node) wordDefinitionElement).getChildNodes();
            strDefinition +=
                ((Node) textNodes.item(0)).getNodeValue() + ". \n";
        }
    }
}
} catch (IOException e1) {
    Log.d("NetworkingActivity", e1.getLocalizedMessage());
}
}
//---vraćanje definicije reči---
return strDefinition;
}
```

PRIMENA GET METODA - FUNKCIONISANJE

*Da bi se dobio objekat tipa Document koriste se
DocumentBuilderFactory i DocumentBuilder objekti.*

Prvi korak prilikom izvršavanja programa jeste angažovanje metode **WordDefinition()** koja uspostavlja HTTP konekciju i prosleđuje reč čiju bi definiciju trebalo dobiti kao rezultat. Navedeno je realizovano sledećim linijama koda metode:

```
in = OpenHttpConnection (http://services.aonaware.com/DictService/DictService.asmx/  
Define?word=" + word);
```

Da bi se dobio objekat tipa *Document* (DOM) iz XML datoteke (vraćene kao rezultat izvršavanja korisničkog zahteva), koriste se **DocumentBuilderFactory** i **DocumentBuilder** objekti (sledeći listing).

```
Document doc = null;
    DocumentBuilderFactory dbf =
        DocumentBuilderFactory.newInstance();
    DocumentBuilder db;
    try {
        db = dbf.newDocumentBuilder();
        doc = db.parse(in);
    } catch (ParserConfigurationException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    doc.getDocumentElement().normalize();
```

Nakon dobijanja objekta tipa *Document*, vrši se pronalaženje svih elemenata koji sadrže **<Definition>** tag:

//---prihvatanje svih <Definition> elemenata---

NodeList definitionElements =

doc.getElementsByTagName("Definition");

Struktura XML dokumenta kojeg vraća web servis data je sledećom slikom.

```
<Word>Belgrade</Word>
- <Definitions>
- <Definition>
  <Word>Belgrade</Word>
  - <Dictionary>
    <Id>wn</Id>
    <Name>WordNet (r) 2.0</Name>
  </Dictionary>
  - <WordDefinition>
    Belgrade n : capital and largest city of Yugoslavia [syn: {Beograd}, {capital of Yugoslavia}]
  </WordDefinition>
</Definition>
</Definitions>
</WordDefinition>
```

Slika 2.3 Struktura vraćenog XML dokumenta

PRIMENA GET METODA – FUNKCIONISANJE - NASTAVAK

Za zadatu reč neophodno je izdvojiti sve definicije.

Pošto se definicija reči nalazi u **<WordDefinition>** elementu, neophodno je nastaviti sa izolovanjem svih definicija za datu reč. Prvi korak je pristup svim **<Definition>** elementima

i izvlačenje **<WordDefinition>** elemenata iz njih. Navedena logika je realizovana sledećim programskim kodom.

```
//---iteracija kroz svaki <Definition> element---
for (int i = 0; i < definitionElements.getLength(); i++) {
    Node itemNode = definitionElements.item(i);
    if (itemNode.getNodeType() == Node.ELEMENT_NODE)
    {
        //---konverzija Definition čvora u Element---
        Element definitionElement = (Element) itemNode;

        //---preuzimanje svih <WordDefinition> elemenata iz
        // <Definition> elementa---
        NodeList wordDefinitionElements =
            (definitionElement).getElementsByTagName(
                "WordDefinition");
        strDefinition = "";
```

Nakon toga, neophodno je pristupiti svim, pojedinačnim, **<WordDefinition>** elementima, a to je prikazano sledećim listingom.

```
//---iteracija kroz svaki <WordDefinition> element---
for (int j = 0; j < wordDefinitionElements.getLength(); j++) {
    //---konverzija <WordDefinition> čvora u Element---
    Element wordDefinitionElement =
        (Element) wordDefinitionElements.item(j);

    //---pruzimanje izvedenih čvorova iz
    // <WordDefinition> elementa---
    NodeList textNodes =
        ((Node) wordDefinitionElement).getChildNodes();
    strDefinition +=
        ((Node) textNodes.item(0)).getNodeValue() + ". \n";
    }
}
}
```

PRIMENA GET METODA – FUNKCIONISANJE - KRAJ

*Neophodno je bilo kreiranje klase, naslednice osnovne klase **AsyncTask**, koja omogućava asinhrono izvođenje metode **WordDefinition()**.*

Prikazanim kodom omogućeno je iterativno pristupanje svim **<Definition>** elementima, u potrazi za **<WordDefinition>** elementima. Sadržaj elementa **<WordDefinition>** predstavlja definiciju odgovarajuće reči. Definicije se nadovezuju i vraćaju pomoću metode **WordDefinition()**, a za to je zadužen sledeći kod:

//---vraćanje definicije reči---

return strDefinition;

Na kraju, neophodno je bilo kreiranje klase, naslednice osnovne klase `AsyncTask`, koja omogućava asinhrono izvođenje metode `WordDefinition()`.

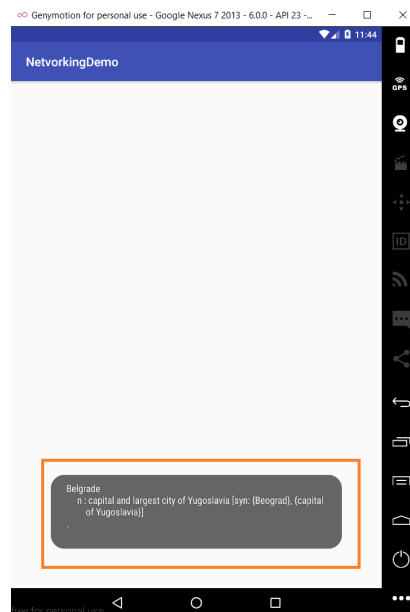
Asinhrono pristupanje web servisu realizovano je instrukcijom, u okviru metode `onCreate()` klase aktivnosti:

```
new AccessWebServiceTask().execute("Belgrade");
```

Takođe, na ovaj način se sprečeno otkazivanje aplikacije jer su zadaci, koji bi mogli da blokiraju korisnički interfejs, prepušteni asinhronoj klasi.

Konačno, ovo je poslednja modifikacija polazne klase aktivnosti pa je neophodno da se izvrši ponovno prevođenje aplikacije u Android Studio IDE razvojnom okruženju da bi nove funkcionalnosti bile dostupne.

Pokretanje programa emulatorom, ili mobilnim uređajem, prikazano je sledećom slikom.



Slika 2.4 Vraćena definicija zadate reči klasom Toast

ZADATAK 2

Pokušajte sami!!!

Pokušajte da samostalno kreirate Android aplikaciju koja na prikazani način koristi GET metodu.

▼ Poglavlje 3

Upotreba JSON servisa

UVOD U PRIMENU JSON SERVISA

DOM troši značajno memorijske resurse i uzima veliku količinu procesorskog vremena.

U prethodnom izlaganju je pokazano kako se koriste XML servisi, uspostavljanjem konekcije sa HTTP serverom i preuzimanjem rezultata u formi XML koda. Pokazano je, takođe, kako se koristi DOM za analiziranje rezultata prikazanih XML dokumentom. Međutim, korišćenje i upravljanje XML dokumentima, iz perspektive mobilnih uređaja, je veoma kompleksno i može dovesti do većih troškova korišćenja mobilnog uređaja. Razlozi su sledeći:

- XML dokumenti su dugi i koriste tagove za oblikovanje informacija. Za veći dokument je neophodno više vremena za preuzimanje sa Interneta, a to, potencijalno, povlači i veće troškove upotrebe mobilnog uređaja;
- XML dokumenti se teško obrađuju. Neophodno je koristiti DOM za pristupanje stablu dokumenta da bi konkretna informacija bila preuzeta i iskorišćena. DOM mora da izgradi celokupno stablo dokumenta u memoriji pre nego što počne da pristupa pojedinačnim elementima stabla. Navedena akcija troši značajno memorijske resurse i uzima veliku količinu procesorskog vremena.

Kao efikasniji način za predstavljanje informacija je **JSON (JavaScript Object Notation)**. Radi se o jednostavnom formatu za razmenu podataka, koji je jednostavan za čitanje i kreiranje dokumenata. Takođe, računar jednostavno može da analizira i kreira podatke.

Upravo, u narednom izlaganju će biti prikazani načini i mehanizmi korišćenja JSON dokumenta u Android mobilnim aplikacijama, a celokupna analiza i izlaganje biće praćeni odgovarajućim primerom koji će u tu svrhu biti kreiran, preveden i testiran.

STRUKTURA JSON DOKUMENTA

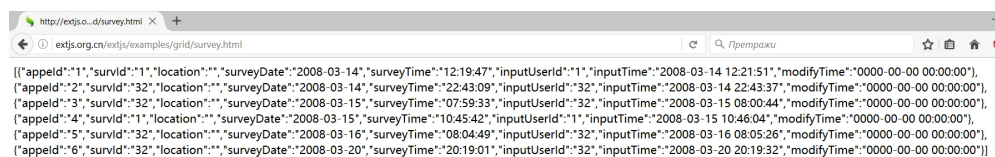
Za razliku od XML koda u JSON dokumentu ne postoje dugi nazivi tagova.

U prvom delu izlaganja, neophodno je pokazati i objasniti izgled i strukturu JSON dokumenta. Sledeći dokument prikazuje izgled JSON poruke. Dokumentom je prikazan programski blok za skup podataka koji se koriste u anketi (izvori: <http://extjs.org.cn/extjs/examples/grid/survey.html> i Wei-Meng L, *Android 4 – razvoj aplikacija*, Kombib 2012). Sve informacije prikazane su kao par (ključ, vrednost) i svaki par je grupisan u uređenu listu objekata.

Za razliku od XML koda u JSON dokumentu ne postoje dugi nazivi tagova, već zagrade i interpunkcija.

```
[{"appeId": "1", "survId": "1", "location": "", "surveyDate": "2008-03-14", "surveyTime": "12:19:47", "inputUserId": "1", "inputTime": "2008-03-14 12:21:51", "modifyTime": "0000-00-00 00:00:00"}, {"appeId": "2", "survId": "32", "location": "", "surveyDate": "2008-03-14", "surveyTime": "22:43:09", "inputUserId": "32", "inputTime": "2008-03-14 22:43:37", "modifyTime": "0000-00-00 00:00:00"}, {"appeId": "3", "survId": "32", "location": "", "surveyDate": "2008-03-15", "surveyTime": "07:59:33", "inputUserId": "32", "inputTime": "2008-03-15 08:00:44", "modifyTime": "0000-00-00 00:00:00"}, {"appeId": "4", "survId": "1", "location": "", "surveyDate": "2008-03-15", "surveyTime": "10:45:42", "inputUserId": "1", "inputTime": "2008-03-15 10:46:04", "modifyTime": "0000-00-00 00:00:00"}, {"appeId": "5", "survId": "32", "location": "", "surveyDate": "2008-03-16", "surveyTime": "08:04:49", "inputUserId": "32", "inputTime": "2008-03-16 08:05:26", "modifyTime": "0000-00-00 00:00:00"}, {"appeId": "6", "survId": "32", "location": "", "surveyDate": "2008-03-20", "surveyTime": "20:19:01", "inputUserId": "32", "inputTime": "2008-03-20 20:19:32", "modifyTime": "0000-00-00 00:00:00"}]
```

Na pomenutoj lokaciji je moguće izlistati i prikazati priloženi JSON listing koji odgovara izvođenju izvesne ankete. Navedeno je prikazano sledećom slikom.



Slika 3.1 Struktura JSON dokumeta sa <http://extjs.org.cn/extjs/examples/grid/survey.html>

JSON SERVISI - PRIVILEGIJE

Za primenu JSON servisa, u Android projektu, neophodno je dodeliti privilegije za pristup Internetu.

Da bi kreirana Android aplikacija bila u mogućnosti da koristi JSON servise, neophodno joj je dodeliti privilegije za pristup Internetu. Privilegije se, kao što je naučeno, dodeljuju u datoteci **AndroidManifest.xml** u okviru taga `<uses-permission ... />`. Sledećim listingom je prikazana odgovarajuća dozvola za pristup Internetu aplikacije koja koristi JSON servise, u konkretnom slučaju radi se o aplikaciji JsonDemo koja će biti kreirana kao podrška tekućem izlaganju.

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

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

    <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>
    </application>

</manifest>

```

PRIMER 5 - JSON SERVISI – KLASA AKTIVNOSTI I ANDROID 6 PODRŠKA

Klasa aktivnosti objedinjuje sve akcije vezane za realizovanje pristupa sadržaju Interneta.

Klasom aktivnosti biće obuhvaćene ključne aktivnosti koje se odnose na pristup i manipulisanje sadržajem preuzetim sa Interneta. Elementi klase koji će, po navedenom redosledu, biti ugrađivani u klasu su:

- Uvoz neophodnih paketa i klasa;
- Metoda za uspostavljanje HTTP konekcije;
- Ugnježdena klasa, naslednice klase *AsyncTask*;
- Izvršavanje asinhronih zadataka implementiranih u *onCreate()* metodu klase aktivnosti.

Sledećim listingom su prikazane sve import instrukcije koje se odnose na klasu aktivnosti primera. Klasa aktivnosti će zadržati podrazumevani naziv *MainActivity.java*.

```

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.StatusLine;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONArray;
import org.json.JSONObject;

```

```
import android.app.Activity;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;
```

Međutim, ovde se javlja jedan problem na koji je Android Studio IDE razvojno okruženje ukazao odmah. Paket `org.apache.http` nije direktno podržan Android API nivoom 23 koji odgovara verziji operativnog sistema Android 6.0. Pošto nije cilj rad sa starijim API verzijama, rešenje ovog problema je traženo u okviru najnovijih API verzija dodavanjem odgovarajuće biblioteke u build.Gradle. Dodavanjem instrukcije `useLibrary 'org.apache.http.legacy'` u blok `android {...}`, sve klase iz paketa `org.apache.http`, koje su prikazane prethodnim listingom, postaće dostupne u Android Studio IDE razvojnom okruženju za kreirani projekat. Navedeno je prikazano sledećim listingom.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 24
    buildToolsVersion "24.0.2"

    useLibrary 'org.apache.http.legacy'

    defaultConfig {
        applicationId "com.metropolitan.jsondemo"
        minSdkVersion 15
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:24.2.1'
    testCompile 'junit:junit:4.12'
}
```

ČITANJE JSON TOKA

Čitanje JSON toka i uspostavljanje HTTP konekcije prepušteno je posebnoj metodi.

Pošto je rešen problem sa paketom i njegovim klasama, neophodnim za rešavanje problema rada sa JSON dokumentima u Android aplikacijama, moguće je nastaviti sa kreiranjem glavne klase aktivnosti.

Sledeći korak jeste definisanje metode, koja će nositi naziv, `readJSONFeed()` čiji će zadatak biti uspostavljanje HTTP konekcije i pristupanje resursu koji će naknadno biti određen navođenjem konkretnog URL. Sledećim listingom, prikazan je programski kod koji odgovara `readJSONFeed()` metodi. On se dodaje odmah iza deklaracije klase aktivnosti.

```
public String readJSONFeed(String URL) {
    StringBuilder stringBuilder = new StringBuilder();
    HttpClient client = new DefaultHttpClient();
    HttpGet httpGet = new HttpGet(URL);
    try {
        HttpResponse response = client.execute(httpGet);
        StatusLine statusLine = response.getStatusLine();
        int statusCode = statusLine.getStatusCode();
        if (statusCode == 200) {
            HttpEntity entity = response.getEntity();
            InputStream content = entity.getContent();
            BufferedReader reader = new BufferedReader(
                new InputStreamReader(content));
            String line;
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line);
            }
        } else {
            Log.e("JSON", "Neuspešno učitavanje datoteke");
        }
    } catch (ClientProtocolException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return stringBuilder.toString();
}
```

UGNJEŽDENA KLASA – ASINHRONO IZVRŠAVANJE

Klasa `ReadJSONFeed` nasleđuje osnovnu klasu `AsyncTask`.

Tokom rešavanja postavljenih softverskih zahteva, ponovo mora da se vodi računa o zadacima koji mogu da upadnu u glavnu nit aplikacije, blokiraju izvršavanje glavne aktivnosti

i dovedu do otkazivanja programa. U tu svrhu biće neophodno definisati novu klasu, ugnježdenu u klasi aktivnosti, čiji je zadatak prosleđivanje svih zadataka koji bi mogli da prekinu glavnu aktivnost u pozadinsku nit.

Definicija klase aktivnosti završava se kreiranjem klase `ReadJSONFeed` koja nasleđuje osnovnu klasu `AsyncTask`. Takođe, poseban zadatak ima osnovna metoda `onCreate()`, kojom se učitava korisnički interfejs u aktivnost. Ova metoda će biti proširena linijama programskog koda koje omogućavaju asinhrono izvršavanje metode `readJSONFeed()`. Sledećim listingom data je metoda `onCreate()` glavne klase aktivnosti.

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

    new ReadJSONFeedTask().execute("http://extjs.org.cn"
        + "/extjs/"
        + "examples/grid/"
        + "survey.html");
}
```

Kod ugnježdene klase `ReadJSONFeed`, sa kojim će biti zaokružena definicija klase aktivnosti, dat je sledećim listingom.

```
private class ReadJSONFeedTask extends AsyncTask<String, Void, String> {
    protected String doInBackground(String... urls) {
        return readJSONFeed(urls[0]);
    }

    protected void onPostExecute(String result) {
        try {
            JSONArray jsonArray = new JSONArray(result);
            Log.i("JSON", "Broj nizova u toku: " +
                jsonArray.length());

            //---Prikazuje sadržaj JSON toka---
            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject jsonObject = jsonArray.getJSONObject(i);

                Toast.makeText(getBaseContext(), jsonObject.getString("appId")
+
                    " - " + jsonObject.getString("inputTime"),
                    Toast.LENGTH_SHORT).show();

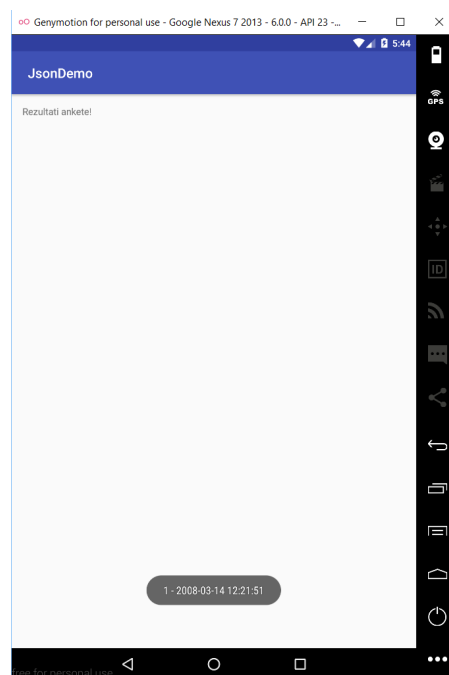
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
}  
}
```

FUNKCIONISANJE I DEMONSTRACIJA PRIMERA

Pojedinačni JSON objekti se čuvaju u listi i pristupa im se primenom `JSONArray` klase.

Sledeći zadatak je, koristeći razvojno okruženje Android Studio IDE, prevođenje i testiranje kreiranog programa. Nakon obavljenih navedenih zadataka, ukoliko ne postoje prijavljene greške, program će funkcionisati na način prikazan sledećom slikom.



Slika 3.2 Pokretanje JSON servisa - demonstracija

Sledi dodatno objašnjenje funkcionisanja kreiranog programa. U prvom koraku je definisana metoda `readJSONFeed()` (videti kod) koja omogućava jednostavno pristupanje resursu koji se nalazi na navedenoj URL adresi (videti kod `onCreate()` metode). Nakon toga se vrši očitavanje sadržaja sa web servera i rezultat se vraća kao `String` objekat.

Da bi kreirana metoda `readJSONFeed()` imala sposobnost asinhronog izvršavanja bilo je neophodno kreirati klasu `ReadJSONFeed`. Navedena klasa je izvedena iz klase `AsyncTask`. Metoda `readJSONFeed()` se izvršava u metodi `doInBackground()`, a potom se identifikovani JSON string prosleđuje pomoću `onPostExecute()` metode (videti priloženi kod). JSON string je definisan u metodi `execute()` koju izvršava objekat klase `ReadJSONFeed` u metodi `onCreate()`.

Lista objekata u JSON stringu dobija se primenom klase `JSONArray` prosleđivanjem JSON podataka u konstruktor navedene klase (videti kod u `onPostExecute()`). Metoda `length()` vraća broj objekata u `JSONArray` objektu.

Metodom `getJSONObject()` se očitavaju pojedinačni objekti iz liste i metoda vraća rezultat tipa `JSONObject`. Za preuzimanje vrednosti iz para (ključ, vrednost) neophodno je primeniti metodu `getString()` (za druge tipove podatak moguće je koristiti i `getInt()`, `getLong()` itd).

JSON U ANDROID APLIKACIJAMA - VIDEO MATERIJALI

U ovom delu lekcije su priloženi izabrani video materijali za temu JSON u Android aplikacijama.

Prvo će biti priložen materijal JSON Parsing, `JSONObjects` i `JsonArrays`, `URLConnection` - Android Studio.

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

Sledi materijal za JSON Parsing, Creating a `URLConnection` - Android Studio.

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

U nastavku je priložen materijal za JSON Parsing, `NetworkOnMainThreadException`, `AsyncTask` - Android Studio.

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

I konačno, JSON Parsing, `Finally Parsing JSON` - Android Studio.

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

PRIMER 6 - VEŽBANJE PREUZIMANJA JSON SADRŽAJA

Vežbanje na računaru preuzimanja i obrade JSON sadržaja.

Zadatak:

1. Sa unapred zadatog URL učitati JSON dokument u Android aplikaciju i prikazati rezultate njegove obrade u Toast klasi.
2. Omogućiti asinhrono izvršavanje zadataka i odgovarajuću Internet privilegiju.

Sledećim listingom je priložen `AndroidManifest.xml` sa Internet privilegijom.

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



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

<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>
</application>

</manifest>
```

Aplikacija ima jednostavan GUI:

```
?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.metropolitan.jsondemo.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Rezultati ankete!" />
</RelativeLayout>
```

Zadatak je rešen prilaganjem listinga koji odgovara glavnoj klasi aktivnosti:

```
package com.metropolitan.jsondemo;

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

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.StatusLine;
```

```
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONArray;
import org.json.JSONObject;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;

public class MainActivity extends AppCompatActivity {

    public String readJSONFeed(String URL) {
        StringBuilder stringBuilder = new StringBuilder();
        HttpClient client = new DefaultHttpClient();
        HttpGet httpGet = new HttpGet(URL);
        try {
            HttpResponse response = client.execute(httpGet);
            StatusLine statusLine = response.getStatusLine();
            int statusCode = statusLine.getStatusCode();
            if (statusCode == 200) {
                HttpEntity entity = response.getEntity();
                InputStream content = entity.getContent();
                BufferedReader reader = new BufferedReader(
                    new InputStreamReader(content));
                String line;
                while ((line = reader.readLine()) != null) {
                    stringBuilder.append(line);
                }
            } else {
                Log.e("JSON", "Neuspešno učitavanje datoteke");
            }
        } catch (ClientProtocolException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return stringBuilder.toString();
    }

    private class ReadJSONFeedTask extends AsyncTask<String, Void, String> {
        protected String doInBackground(String... urls) {
            return readJSONFeed(urls[0]);
        }

        protected void onPostExecute(String result) {
            try {
                JSONArray jsonArray = new JSONArray(result);
                Log.i("JSON", "Broj nizova u toku: " +
                    jsonArray.length());
            }
        }
    }
}
```

```

        //--Prikazuje sadržaj JSON toka--
        for (int i = 0; i < jsonArray.length(); i++) {
            JSONObject jsonObject = jsonArray.getJSONObject(i);

            Toast.makeText(getBaseContext(), jsonObject.getString("appeId")
+
                " - " + jsonObject.getString("inputTime"),
                Toast.LENGTH_SHORT).show();

            /* Toast.makeText(getBaseContext(),
jsonObject.getString("text") +
                " - " + jsonObject.getString("kreiran "),
                Toast.LENGTH_SHORT).show();*/
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

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

    new ReadJSONFeedTask().execute("http://extjs.org.cn"
        + "/extjs/"
        + "examples/grid/"
        + "survey.html");

    /*new ReadJSONFeedTask().execute("https://twitter.com/statuses/
user_timeline/"
        + "weimenglee.json");*/
}
}

```

ZADATAK 3

Pokušajte sami!!!

Pokušajte da samostalno kreirate Android aplikaciju koja koristi predstavljanje informacija pomoću JSON (JavaScript Object Notation).

▼ Poglavlje 4

Programiranje soketa

UVOD U PROGRAMIRANJE SOCKET-A

Svaki pristup web servisu preko HTTP protokola je nova konekcija.

U dosadašnjem izlaganju prikazani su web servisi koji koriste HTTP protokol za komunikaciju i ovo se dešava u najvećem broju slučajeva. Međutim, ovakav način komuniciranja ima jedan ozbiljan nedostatak. Svaki pristup web serveru preko HTTP protokola tretira se kao nova konekcija, a to znači da web server ne održava trajnu komunikaciju sa korisnicima tj. ne postoje stanja.

Ako se zamisli slučaj mobilne aplikacije kojom se pristupa sistemu za rezervaciju karata. Kada klijent rezerviše mesto, ostali klijenti nisu upoznati sa obavljenom registracijom sve dok se ponovo ne povežu na web servis sa ciljem rezervisanja svojih mesta. Stalno povezivanje na web servis dovodi do nepotrebnog trošenja propusnog opsega, a samu aplikaciju čini sporom i neefikasnom. Kao kvalitetnije rešenje nameće se server koji ostvaruje pojedinačne konekcije sa svakim klijentom i šalje poruke klijentima kada je neko od njih rezervisao određeno sedište.

Za održavanje stalne veze klijenata sa serverom, i slanje poruka na gore navedeni način, neophodno je primeniti tehniku poznatu kao programiranje soketa. Za demonstriranje navedene tehnike biće kreirana aplikacija za časkanje koja se povezuje na server pomoću soketa. Više klijent aplikacija se povezuje sa serverom i između njih se realizuje časkanje. Aplikacije će biti pokrenute na više emulatora, ili realnih uređaja, a potom će tokom testiranja kreirane aplikacije svi uređaji, realni ili virtualni, biti uključeni u razmenu poruka.

PRIMER 5 - PROGRAMIRANJE SOCKETA – XML DATOTEKE

Prvi korak je definisanje privilegija i korisničkog interfejsa.

Kao prvi zahtev, koji je neophodno razmotriti prilikom kreiranja **SocketDemo** aplikacije, jeste dozvole neophodne za njeno funkcionisanje.

Aplikacija koja omogućava trajnu konekciju sa web serverom mora da ima privilegije za pristup Internetu. Sledećim listingom dat je kod **AndroidManifest.xml** datoteke sa odgovarajućom dozvolom.

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

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

<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>
</application>

</manifest>
```

U nastavku, biće kreiran korisnički interfejs klijent Android aplikacije za **chat** realizovanje servisa. Datotekom, čiji je podrazumevani naziv **activity_main.xml** zadržan, definisan je korisnički interfejs aplikacije za časkanje. Sledećim listingom je priložen njen kod.

```
<?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" >

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

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Pošalji poruku"
        android:onClick="onClickSend"/>

    <TextView
        android:id="@+id/txtMessagesReceived"
        android:layout_width="fill_parent"
        android:layout_height="200dp"
        android:scrollbars = "vertical" />

</LinearLayout>
```

COMMSTHREAD KLASA

Neophodno je kreirati posebnu klasu za upravljanje specifičnom komunikacijom preko socket-a.

Iz ovog primera moguće je videti da se radi o malo složenijem Android rešenju i sam projekat će biti sastavljen od više klasa, pored glavne klase aktivnosti. Sledeći zadatak jeste kreiranje klase, koja nasleđuje osnovnu klasu `Thread`, čiji je zadatak upravljanje specifičnim komunikacijama koje se odvijaju preko `socket`-a. Kod klase, pod nazivom `CommsThread`, priložen je sledećim listingom koji će u nekom od narednih izlaganja biti predmet analize.

```
package com.metropolitan.socketdemo;

import android.util.Log;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

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

public class CommsThread extends Thread {

    private final Socket socket;
    private final InputStream inputStream;
    private final OutputStream outputStream;

    public CommsThread(Socket sock) {
        socket = sock;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
        try {
            //---kreira inputstream i outputstream objekte
            // za čitanje i pisanje preko socket-a---
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.d("SocketChat", e.getLocalizedMessage());
        }
        inputStream = tmpIn;
        outputStream = tmpOut;
    }

    public void run() {
        //---bafer skladište za konkretan tok---
        byte[] buffer = new byte[1024];
        //---vraćeni bajtovi iz read()---
```

```

int bytes;
//---nastavak osluškivanja InputStream do
// pojave izuzetka---
while (true) {
    try {
        //---čitanje iz inputStream---
        bytes = inputStream.read(buffer);

        //---ažuriranje glavne aktivnosti - UI---
        MainActivity.UIUpdater.obtainMessage(
            0,bytes, -1, buffer).sendToTarget();
    } catch (IOException e) {
        break;
    }
}
}
//---poziv iz glavne aktivnosti
// slanje podataka na udaljeni uređaj---
public void write(byte[] bytes) {
    try {
        outputStream.write(bytes);
    } catch (IOException e) { }
}

//---pozivanje iz glavne aktivnosti
// za zatvaranje konekcije---
public void cancel() {
    try {
        socket.close();
    } catch (IOException e) { }
}
}

```

PROGRAMIRANJE SOCKET-A – KLASA AKTIVNOSTI

Klasa aktivnosti sadrži posebno definisane metode i ugnježdene klase naslednice klase `AsyncTask`.

Klasa aktivnosti objedinjuje ključne metode i ugnježdene klase, naslednice klase `AsyncTask`, za programiranje socket-a. Sledećim listingom su prikazani uveženi paketi i odgovarajuće klase, neophodni za funkcionisanje klase aktivnosti.

```

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

```

```
import android.widget.TextView;

import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;
```

U daljem izlaganju, korak - po - korak, biće građena definicija i struktura glavne klase aktivnosti. Uvodne instrukcije klase aktivnosti i metoda `handleMessage()` dati su sledećim kodom.

```
public class MainActivity extends AppCompatActivity {

    static final String NICKNAME = "Vlada";
    //---socket---
    InetAddress serverAddress;
    Socket socket;
    //---svi pogledi---
    static TextView txtMessagesReceived;
    EditText txtMessage;
    //---nit za komuniciranje sa socket-om---
    CommsThread commsThread;
    //---ažuriranje UI glavne aktivnosti---
    static Handler UIupdater = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            int numOfBytesReceived = msg.arg1;
            byte[] buffer = (byte[]) msg.obj;

            //---prevođenje byte niza u string---
            String strReceived = new String(buffer);

            //---pruzimanje aktuelnog stringa---
            strReceived = strReceived.substring(
                0, numOfBytesReceived);

            //---prikazivanje primljenog teksta u TextView---
            txtMessagesReceived.setText(
                txtMessagesReceived.getText().toString() +
                strReceived);
        }
    };
};
```

UGNJEŽDENE KLASÉ

Klasa aktivnosti objedinjuje tri ugnježdene klase naslednice klase `AsyncTask`.

Ovde je od posebnog značaja vođenje računa o aktivnostima koje pretenduju da prekinu izvršavanje glavne aktivnosti i dovedu, na taj način, do otkazivanja programa. U tu svrhu

je neophodno kreirati izvesne klase naslednice `AsyncTask` klase za upravljanje navedenim zadacima. Ove klase će biti posebno analizirane u narednim izlaganjima, a ovde će biti priložen njihov kod kao nastavak građenja strukture glavne klase aktivnosti.

Sledeći zadatak jeste kreiranje tri specifične ugnježdene klase. Kod klase `CreateCommThreadTask` dat je sledećim kodom.

```
private class CreateCommThreadTask extends AsyncTask
    <Void, Integer, Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            //---kreiranje soketa---
            serverAddress =
                InetAddress.getByName("192.168.1.101");
            socket = new Socket(serverAddress, 500);
            commsThread = new CommsThread(socket);
            commsThread.start();
            //---prijavljivanje korisnika; šalje nadimak---
            sendToServer(NICKNAME);
        } catch (UnknownHostException e) {
            Log.d("Sockets", e.getLocalizedMessage());
        } catch (IOException e) {
            Log.d("Sockets", e.getLocalizedMessage());
        }
        return null;
    }
}
```

Kodovi klasa `WriteToServerTask` i `CloseSocketTask`, koje takođe nasleđuju `AsyncTask` klasu, dati su sledećim listinzima.

```
private class WriteToServerTask extends AsyncTask
    <byte[], Void, Void> {
    protected Void doInBackground(byte[]...data) {
        commsThread.write(data[0]);
        return null;
    }
}

private class CloseSocketTask extends AsyncTask
    <Void, Void, Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            socket.close();
        } catch (IOException e) {
            Log.d("Sockets", e.getLocalizedMessage());
        }
        return null;
    }
}
```

KLASA AKTIVNOSTI – ONCREATE() I METODE KONTROLA UI.

Definicija klase aktivnosti zatvorena je metodama za upravljanje UI.

Konačno, definicija klase aktivnosti mora da se zaokruži programiranjem još nekih korisnih metoda. Sastavni deo ove klase je metoda koja učitava korisnički interfejs u glavnu aktivnost. Takođe, biće implementirana i metoda kojom je omogućeno slanje poruke nakon klika na odgovarajuće dugme.

Sledećim listingom prikazane su metode `onCreate()` i `onClickSend()`.

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

    //---preuzimanje pogleda---
    txtMessage = (EditText) findViewById(R.id.txtMessage);
    txtMessagesReceived = (TextView)
        findViewById(R.id.txtMessagesReceived);
}

public void onClickSend(View view) {
    //---prosleđivanje poruke na server---
    sendToServer(txtMessage.getText().toString());
}
```

U glavnu klasu aktivnosti će biti ugrađene i još neke korisne metode koje će biti analizirane u narednom izlaganju. Sledećim kodom date su metode `sendToServer()`, `onResume()` i `onPause()`.

```
private void sendToServer(String message) {
    byte[] theByteArray =
        message.getBytes();
    new WriteToServerTask().execute(theByteArray);
}

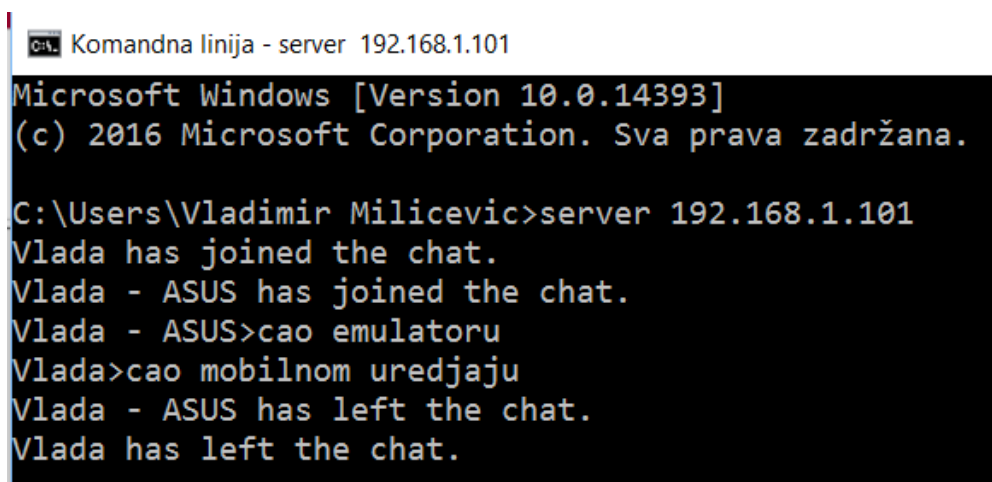
@Override
public void onResume() {
    super.onResume();
    new CreateCommThreadTask().execute();
}

@Override
public void onPause() {
    super.onPause();
    new CloseSocketTask().execute();
}
```

PROGRAMIRANJE SOCKET-A - DEMONSTRACIJA

Čet je pokrenut sa emulatora i realnog uređaja, a praćen je na serveru.

Studentima će biti dostupna i mala serverska aplikacija **Server.exe** koja se kopira u *Windows* folder i pokreće iz *cmd*. Preko ove aplikacije komuniciraju uređaji uključeni u dopisivanje. Veoma je važno da IP adresa, iz `doInBackground()` metode, u potpunosti odgovara IP adresi računara koji glumi server i pre prevođenja neophodno je uneti ispravnu IP adresu. Za svaki od uređaja neophodno je definisani drugačiji nadimak. Aplikacija je prevedena i instalirana, za različite vrednosti nadimaka, za emulator i realan uređaj koji su, potom, razmenili nekoliko poruka. Sledeća slika pokazuje severski monitoring.



```
Komandna linija - server 192.168.1.101
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. Sva prava zadržana.

C:\Users\Vladimir Milicevic>server 192.168.1.101
Vlada has joined the chat.
Vlada - ASUS has joined the chat.
Vlada - ASUS>cao emulatoru
Vlada>cao mobilnom uređaju
Vlada - ASUS has left the chat.
Vlada has left the chat.
```

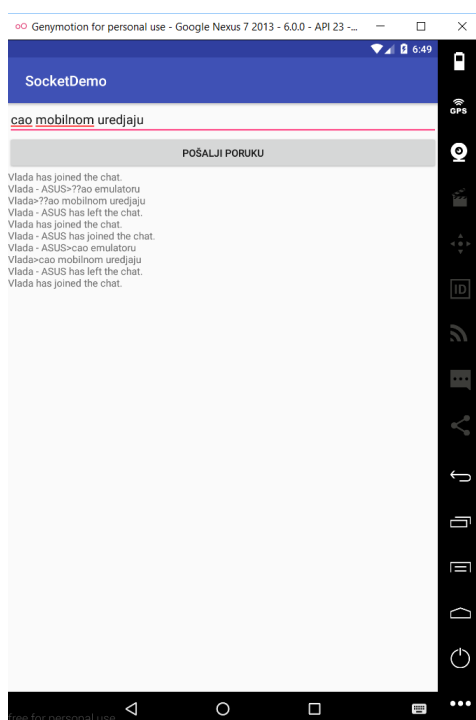
Slika 4.1 Sever aplikacija za čet

PROGRAMIRANJE SOCKET-A - DEMONSTRACIJA - NASTAVAK

Aplikacija je prevedena, posebno za svaki uređaj i za različite vrednosti nadimaka.

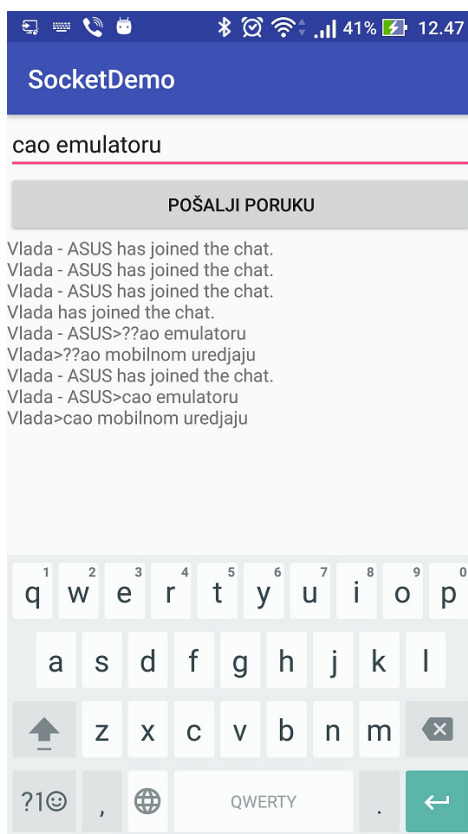
Kao što je već napomenuto. Kreirana klijent **chat** aplikacija mora da bude instalirana na svaki uređaj, realni ili virtuelni, koji učestvuje u ćaskanju.

Za vrednost nadimka **Vlada**, aplikacija je prevedena i instalirana na emulatoru koji je spreman za dopisivanje (sledeća slika).



Slika 4.2 Aplikacija na emulatoru

Za vrednost nadimka *Vlada - ASUS*, aplikacija je prevedena i instalirana na realnom mobilnom telefonu koji je spreman za dopisivanje (sledeća slika).



Slika 4.3 Aplikacija na telefonu

PROGRAMIRANJE SOCKET-A – FUNKCIONISANJE

Komuniciranje pomoću socket-a ostvaruje se u različitim nitima.

Za upravljanje specifičnom komunikacijom preko socket-a, prvo je kreirana klasa, naslednika osnovne klase `Thread`, pod nazivom `CommsThread`. Svaka komunikacija pomoću socket-a ostvaruje se u različitim nitima koje su izdvojene od glavne niti korisničkog interfejsa. U klasi su deklarirana tri objekta:

```
private final Socket socket;
```

```
private final InputStream inputStream;
```

```
private final OutputStream outputStream;
```

Prvi objekat obezbeđuje klijentski TCP soket. `InputStream` se koristi prilikom očitavanja podataka iz socket konekcije, a `OutputStream` obezbeđuje prosleđivanje podataka pomoću socket konekcije.

Konstruktor `CommsThread` klase preuzima `Socket` objekat i pokušava da učitava `InputStream` i `OutputStream` objekte socket konekcije (videti priloženi kod).

Metoda `run()`, izvršava se pozivom metode niti `start()`, nastavlja da prati dolazne podatke koristeći `InputStream` objekat. Potom, ažurira se korisnički interfejs prosleđivanjem primljene poruke (videti priloženi kod).

Metoda `write()` pomaže prilikom prosleđivanja podataka upotrebom socket konekcije.

Na kraju, primenom metode `cancel()` omogućava se prekidanje uspostavljene socket konekcije.

Za obe metode pogledati priloženi programski kod.

PROGRAMIRANJE SOCKET-A – FUNKCIONISANJE - NASTAVAK

Zadaci asinhronne komunikacije u nadležnosti su klase aktivnosti.

U glavnoj klasi aktivnosti definisane su tri klase koje su izvedene iz bazne klase `AsyncTask`.

Klasa `CreateCommThreadTask` asinhrono uspostavlja socket konekciju sa serverom. Prvi string koji prosleđuje predstavlja nadimak klijenta povezanog na server za dopisivanje (videti priloženi kod).

Klasa `WriteToServer` omogućava da se asinhrono pošalju podaci prema serveru, a klasa `CloseSocketTask` ima zadatak da prekine konekciju koja je uspostavljena pomoću socket-a (videti priložene kodove).

Metodom `sendToServer()` obezbeđeno je preuzimanje `String` argumenta i njegova konverzija u niz bajtova. Izvršavanjem metode `execute()`, ovaj niz bajtova se prosleđuje serveru (videti priloženi kod).

Na samom kraju, metodama `onPause()` i `onResume()` omogućeno je da posle kratkog prekida aktivnosti dolazi do prekidanja konekcije koja se realizuje preko socket-a i nakon ponovnog iniciranja aktivnosti dolazi do ponovnog uspostavljanja konekcije, respektivno (videti priloženi kod).

Sledećim video materijalom pokrivena je tema Android mrežnog programiranja - programiranje Socket -a.

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

PRIMER 6 - PROGRAMIRANJE SOCKET - A

Vježbanje programiranja Socketa u Android aplikacijama.

ZADATAK:

1. Kreiranje jednostavne klijent Android aplikacije za instant razmenu poruka između studenata;
2. Ugraditi odgovarajuće dozvole, kreirati GUI po uzoru na sledeću sliku.
3. Kreirati odgovarajuće klase naslednice klase `AsyncTask`, `CommsTread` klasu po uzoru na predavanje i glavnu klasu aktivnosti.



Slika 4.4 GUI klijent čet aplikacije.

Sledećim listingom je priložen `AndroidManifest.xml` da Internet privilegijom.

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

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

```
<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>
</application>

</manifest>
```

GUI aplikacije sa postavljenom pozadinom koja odgovara logu našeg Univerziteta 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"
    android:background="@drawable/metpozadina">

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

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Pošalji poruku"
        android:onClick="onClickSend"/>

    <TextView
        android:id="@+id/txtMessagesReceived"
        android:layout_width="fill_parent"
        android:layout_height="200dp"
        android:scrollbars = "vertical" />

</LinearLayout>
```

PRIMER 6 - NASTAVAK

Kreiranje JAVA klasa

Klasa CommsThread, naslednica osnovne klase Thread, data je sledećim listingom.

```
package com.metropolitan.socketdemo;

import android.util.Log;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

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

public class CommsThread extends Thread {

    private final Socket socket;
    private final InputStream inputStream;
    private final OutputStream outputStream;

    public CommsThread(Socket sock) {
        socket = sock;
        InputStream tmpIn = null;
        OutputStream tmpOut = null;
        try {
            //---kreira inputstream i outputstream objekte
            // za čitanje i pisanje preko socket-a---
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) {
            Log.d("SocketChat", e.getLocalizedMessage());
        }
        inputStream = tmpIn;
        outputStream = tmpOut;
    }

    public void run() {
        //---bafer skladište za konkretan tok---
        byte[] buffer = new byte[1024];
        //---vraćeni bajtovi iz read()---
        int bytes;
        //---nastavak osluškivanja InputStream do
        // pojave izuzetka---
        while (true) {
            try {
                //---čitanje iz inputStream---
                bytes = inputStream.read(buffer);

                //---ažuriranje glavne aktivnosti - UI---
                MainActivity.UIUpdater.obtainMessage(
                    0, bytes, -1, buffer).sendToTarget();
            } catch (IOException e) {
```



```

        break;
    }
}

//---poziv iz glavne aktivnosti
// slanje podataka na udaljeni uređaj---
public void write(byte[] bytes) {
    try {
        outputStream.write(bytes);
    } catch (IOException e) { }
}

//---pozivanje iz glavne aktivnosti
// za zatvaranje konekcije---
public void cancel() {
    try {
        socket.close();
    } catch (IOException e) { }
}
}

```

Konačno, zadatak je završen kreiranjem glavne klase aktivnosti.

```

package com.metropolitan.socketdemo;

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

import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;

public class MainActivity extends AppCompatActivity {

    static final String NICKNAME = "Student1";
    //---socket---
    InetAddress serverAddress;
    Socket socket;
    //---svi pogledi---
    static TextView txtMessagesReceived;
    EditText txtMessage;
    //---nit za komuniciranje sa socket-om---
    CommsThread commsThread;
    //---ažuriranje UI glavne aktivnosti---
    static Handler UIUpdater = new Handler() {

```

```

@Override
public void handleMessage(Message msg) {
    int numOfBytesReceived = msg.arg1;
    byte[] buffer = (byte[]) msg.obj;

    //---prevođenje byte niza u string---
    String strReceived = new String(buffer);

    //---pruzimanje aktuelnog stringa---
    strReceived = strReceived.substring(
        0, numOfBytesReceived);

    //---prikazivanje primljenog teksta u TextView---
    txtMessagesReceived.setText(
        txtMessagesReceived.getText().toString() +
        strReceived);
}
};

private class CreateCommThreadTask extends AsyncTask
    <Void, Integer, Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            //---kreiranje soketa---
            serverAddress =
                InetAddress.getByName("192.168.1.101");
            socket = new Socket(serverAddress, 500);
            commsThread = new CommsThread(socket);
            commsThread.start();
            //---prijavljivanje korisnika; šalje nadimak---
            sendToServer(NICKNAME);
        } catch (UnknownHostException e) {
            Log.d("Sockets", e.getLocalizedMessage());
        } catch (IOException e) {
            Log.d("Sockets", e.getLocalizedMessage());
        }
        return null;
    }
}

private class WriteToServerTask extends AsyncTask
    <byte[], Void, Void> {
    protected Void doInBackground(byte[]...data) {
        commsThread.write(data[0]);
        return null;
    }
}

private class CloseSocketTask extends AsyncTask
    <Void, Void, Void> {
    @Override

```

```

        protected Void doInBackground(Void... params) {
            try {
                socket.close();
            } catch (IOException e) {
                Log.d("Sockets", e.getMessage());
            }
            return null;
        }
    }

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

        //---preuzimanje pogleda---
        txtMessage = (EditText) findViewById(R.id.txtMessage);
        txtMessagesReceived = (TextView)
            findViewById(R.id.txtMessagesReceived);
    }

    public void onClickSend(View view) {
        //---prosleđivanje poruke na server---
        sendToServer(txtMessage.getText().toString());
    }

    private void sendToServer(String message) {
        byte[] theByteArray =
            message.getBytes();
        new WriteToServerTask().execute(theByteArray);
    }

    @Override
    public void onResume() {
        super.onResume();
        new CreateCommThreadTask().execute();
    }

    @Override
    public void onPause() {
        super.onPause();
        new CloseSocketTask().execute();
    }
}

```

ZADATAK 4

Pokušajte sami!!!

Priloženu chat aplikaciju stilizujte dijalogima dobrodošlice na chat. Dodajte podatke o trenutnom vremenu i datumu.

▼ Poglavlje 5

Domaći zadatak 9

ZADATAK 1

Vežbanje primene JSON u Android aplikaciji.

Razviti samostalno Android aplikaciju, koja koristi JSON string, po uzoru na primere iz lekcije i sa sledeće adrese:

http://www.tutorialspoint.com/android/android_json_parser.htm

ZADATAK 2

Vežbanje povezivanja pomoću socket-a

Kreirati Android aplikaciju koja ima sledeće elemente:

1. Raspored elemenata korisničkog interfejsa je po slobodnom izboru;
2. Glavna aktivnost je izdeljena na sledeće podaktivnosti: fragment na kojem se smenjuju slike okruženja našeg Univerziteta, fragment koji pokazuje datum i vreme i fragment koji realizuje simulaciju čet komunikacije između zaposlenih i/ili studenata sa našeg Univerziteta. Za čet fragment koristiti kao uzor primer iz ove lekcije.

▼ Pregled Lekcije10

PREGLED LEKCIJE 9

Lekcija je obradila načine komuniciranja mobilne aplikacije sa spoljašnjim svetom koristeći Internet privilegije.

Lekcija je stavila akcenat na korišćenje HTTP protokola prilikom komunikacije kreirane mobilne aplikacije sa okolinom. Pokazano je kako primenom HTTP protokola, aplikacija može da preuzme i koristi različite sadržaje sa web servera. Takođe, lekcija se fokusirala na analizu preuzetih XML datoteka i na upotrebu JSON servisa koji su se pokazali jednostavnijim za primenu nego što je slučaj sa XML web servisima.

Poseban zadatak lekcije je bio obrađivanje komunikacije pomoću socket-a. Na ovaj način obezbeđuje se da aplikacija ostane povezana sa serverom i da prima podatke uvek kada su oni dostupni. Veoma važna činjenica, obrađena ovom lekcijom, jeste da sve sinhronne operacije moraju da budu enkapsulirane primenom *AsyncTask* klase. U suprotnom aplikacija neće moći da bude izvršena ni na jednom uređaju kojeg pokreće novija generacija Android operativnog sistema.

LITERATURA

Za pripremanje lekcije korišćena je aktuelna 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. <http://services.aonaware.com/DictService/DictService.asmx/Define?>
6. <http://stackoverflow.com/questions/31433687/android-gradle-apache-httpclient-does-not-exist>