



Mobile Apps Development

COMP-304

Fall 2018



Review of Lecture 8

- ❑ Using the **Google Maps Android API v2**

- Google Maps Android API is distributed as part of **Google Play services** SDK.

- ❑ Obtain an **API key**. To do this, you will need to register a project in the **Google APIs Console**, and get a signing certificate for your app.

- ❑ Specify **settings in the Application Manifest**.

- Permissions
- API key

- ❑ **Add a map** to a new or existing Android project.

- Use **FragmentMap** component

- ❑ Use **Google API** as target platform

- ❑ Test your app on an Android device

- ❑ **Finding Your Location**

- Create an instance of the **LocationManager** using **getSystemService()** method



Review of Lecture 8

- ❑ Provide an implementation of **LocationListener** interface (four methods):
 - **onLocationChanged()** method

location.**getLatitude()**
location.**getLongitude()**
location.**getAltitude()**
location.**distanceTo**
- ❑ **Add a map** to a new or existing Android project.
 - Use **FragmentManager** component
- ❑ Use **Google API** as target platform
- ❑ Test your app on an Android device
- ❑ **Geocoder** object to find geographic coordinates from street number, etc.
 - coder.**getFromLocationName**(placeName, 3);
- ❑ **Mapping Intents**
//create a String that conforms to the URI
// handled by the mapping application
String geoURI =
String.format("geo:%f,%f", lat, lon);
//create a new Uri object for creating a
// new ACTION_VIEW Intent
Uri geo = Uri.parse(geoURI);
Intent geoMap = new
Intent(Intent.ACTION_VIEW, geo);
startActivity(geoMap);



Using Android Networking APIs

Objectives:

- ❑ Explain **Mobile networking fundamentals**.
- ❑ Develop **secure mobile apps that connect to web resources and process tasks asynchronously**:
 - How to connect to the web using **HTTP**
 - How to consume **XML** web services
 - How to consume **JSON** web services



Using Android Networking APIs

- ❑ Networking on the Android platform is based on using **java.net** classes
- ❑ Android SDK provides a number of tools and classes for ensuring **stable and responsive apps**.
- ❑ The most common way to transfer data to and from the network is to use **HTTP**:
 - secure the data with **Secure Sockets Layer (SSL)**
- ❑ For networking to work in any Android application, **internet permission is required**.
 - Add the following statement in its AndroidManifest.xml file:

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



Reading Data from the Web

- ❑ **Read a fixed amount of text from a file on a web server, like this:**

```
URL text = new URL(
    "http://api.flickr.com/services/feeds/photos_public.gn
    e" +
    "?id=26648248@N04&lang=en-us&format=atom");
InputStream isText = text.openStream();
byte[] bText = new byte[250];
int readSize = isText.read(bText);
Log.i("Net", "readSize = " + readSize);
Log.i("Net", "bText = " + new String(bText));
isText.close();
```



Using **HttpURLConnection**

- ❑ We can use the **HttpURLConnection** object to check on our URL before we transfer too much data.
- ❑ **HttpURLConnection** retrieves some information about the resource referenced by the URL object, including HTTP status and header information.
 - **length** of the content
 - content **type**
 - **date-time** information



Using HttpURLConnection

```
URL text = new URL(
    "http://api.flickr.com/services/feeds/photos_public.gne
    ➡ ?id=26648248@N04&lang=en-us&format=atom");
HttpURLConnection http =
    (HttpURLConnection)text.openConnection();
Log.i("Net", "length = " + http.getContentLength());
Log.i("Net", "respCode = " + http.getResponseCode());
Log.i("Net", "contentType = " + http.getContentType());
Log.i("Net", "content = "+http.getContent());
```




Using HttpURLConnection

```
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("Not an HTTP connection");
    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("Error connecting");
    }
    return in;
}
```



Parsing XML from the Network

- ❑ A large portion of data transmitted between network resources is stored in a structured fashion in Extensible Markup Language (**XML**).
- ❑ In particular, RSS feeds are provided in a standardized XML format, and many web services provide data using these feeds.
- ❑ Android SDK provides a variety of XML parsing utilities
 - XML Pull Parser
 - SAX
 - DOM



Processing Asynchronously

- ❑ Users demand responsive applications, so time-intensive operations such as **networking should not block the main UI thread**.
- ❑ The style of networking presented so far causes the UI thread it runs on to block until the operation finishes.
- ❑ For small tasks, this might be acceptable.
- ❑ However, when timeouts, large amounts of data, or additional processing, such as parsing XML, is added into the mix, you should **move these time-intensive operations off of the main UI thread**.



Processing Asynchronously

- ❑ The Android SDK provides two easy ways to manage offload processing from the main UI thread:
 - the **AsyncTask** class
 - and the standard Java **Thread** class.



Working with **AsyncTask**

- ❑ **AsyncTask** is an abstract helper class for **managing background operations** that eventually post back to the UI thread.
- ❑ It creates a simpler interface for asynchronous operations than manually creating a Java Thread class.
- ❑ Create a subclass of **AsyncTask** and implement the appropriate event methods.
 - The **onPreExecute()** method runs on the UI thread **before background processing begins**.
 - The **doInBackground()** method **handles background processing**.
 - **publishProgress()** informs the UI thread periodically about the background processing progress.
 - When the background processing finishes, the **onPostExecute()** method runs on the UI thread to give a final update.



Working with **AsyncTask**

- ❑ AsyncTask uses **generics** and **varargs**:
 - The parameters are the following **AsyncTask** `<TypeOfVarArgParams , ProgressValue , ResultValue>` .
 - **TypeOfVarArgParams** is passed into the `doInBackground()`
 - **ProgressValue** is used for progress information
 - **ResultValue** must be returned from `doInBackground()` and is passed to `onPostExecute()` as parameter.



Downloading Binary Data

```
public class NetworkingActivity extends Activity {
    ImageView img;
    private InputStream OpenHttpConnection(String urlString) throws IOException
    {.....}
    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);
        }
    }
}
```



Downloading Binary Data

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    new DownloadImageTask().execute(  
        "http://www.mayoff.com/5-01cablecarDCP01934.jpg");  
}  
} //end of activity
```

- ❑ To call the DownloadImageTask class, create an instance of it and then call its **execute()** method, passing it the URL of the image to download:

```
new DownloadImageTask().execute("http://www.jfdimarzio.com/butterfly.png");
```




Working with AsyncTask

```
private class ImageLoader extends AsyncTask<URL, String,  
    String>  
{  
    @Override  
    protected String doInBackground(  
        URL... params) {  
        // just one param  
        try {  
            URL text = params[0];  
            // ... parsing code {  
            publishProgress(  
                "imgCount = " + curlImageCount);  
            // ... end parsing code }  
        }  
    }  
}
```



Working with AsyncTask

```
catch (Exception e ) {  
    Log.e("Net",  
        "Failed in parsing XML", e);  
    return "Finished with failure.";  
  
}  
return "Done...";  
}
```



Working with AsyncTask

```
protected void onCancelled() {  
    Log.e("Net", "Async task Cancelled");  
}  
protected void onPostExecute(String result) {  
    mStatus.setText(result);  
}  
protected void onPreExecute() {  
    mStatus.setText("About to load URL");  
}  
protected void onProgressUpdate(  
String... values) {  
    // just one value, please  
    mStatus.setText(values[0]);  
}  
}
```



Working with **AsyncTask**

- ❑ When launched with the `AsyncTask.execute()` method, `doInBackground()` runs in a background thread while the other methods run on the UI thread



Using Threads for Network Calls

- ❑ Executing both the parsing code and the networking code on a separate thread allows the **user interface to continue to behave in a responsive fashion** while the network and parsing operations are done behind the scenes, resulting in a **smooth and friendly user experience**.



Downloading Text Content

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

} 11/9/2018



Downloading Text Content

```
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(getApplicationContext(), result, Toast.LENGTH_LONG).show();  
    }  
}  
/** Called when the activity is first created. */  
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    //---download text---  
    new DownloadTextTask().execute(  
        "http://iheartquotes.com/api/v1/random?max_characters=256&max_lines=10");  
}
```

- ❑ The DownloadText() method takes the URL of the text file to download and then returns the string of the text file downloaded.
- ❑ It opens an HTTP connection to the server and uses an **InputStreamReader** object to read each character from the stream and save it in a **String** object.



Downloading Text Content

- ❑ You need to create a subclass of the AsyncTask class in order to **call the DownloadText() method asynchronously.**



Retrieving Android Network Status

- ❑ The Android SDK provides utilities for gathering information about the current **state of the network**.
- ❑ This is useful to determine if a network connection is even available before trying to use a network resource.
- ❑ The **ConnectivityManager** class provides a number of methods to do this.
- ❑ The following code determines if the mobile (cellular) network is available and connected and also if Wi-Fi network is available and connected :
 - The following statement is required to be in its AndroidManifest.xml file:

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



Retrieving Android Network Status

```
ConnectivityManager cm = (ConnectivityManager)  
getSystemService(Context.CONNECTIVITY_SERVICE);  
NetworkInfo ni =  
cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI);  
boolean isWifiAvail = ni.isAvailable();  
boolean isWifiConn = ni.isConnected();  
ni = cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);  
boolean isMobileAvail = ni.isAvailable();  
boolean isMobileConn = ni.isConnected();  
status.setText("WiFi\nAvail = "+ isWifiAvail +  
"\nConn = " + isWifiConn +  
"\nMobile\nAvail = "+ isMobileAvail +  
"\nConn = " + isMobileConn);
```



Accessing Web Services Using the GET Method

- ❑ You can connect to a web service using the HTTP GET method.
- ❑ Once the web service returns a result in XML, you extract the relevant parts and display its content using the **Toast** class

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



Accessing Web Services Using the GET Method

```
<?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>
```



Accessing Web Services Using the GET Method

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



CONSUMING JSON SERVICES

- ❑ Manipulating XML documents is a computationally expensive operation:
- ❑ The size of an XML document can get very big pretty quickly.
 - your device has to use more bandwidth to download it, which translates into higher cost.
- ❑ XML documents are more difficult to process.
 - Using DOM to traverse the tree in order to locate the information you want.
- ❑ DOM itself has to build the entire document in memory as a tree structure before you can traverse it.
 - This is both memory and CPU intensive.



CONSUMING JSON SERVICES

- ❑ Using JSON is much more efficient way to represent information exists in the form of JSON (JavaScript Object Notation).
- ❑ JSON is a **lightweight data-interchange format** that is easy for humans to read and write



CONSUMING JSON SERVICES

```
public class JSONActivity extends Activity {
    public String readJSONFeed(String URL) {
        URL url = null;
        try {
            url = new URL(address);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        };
        StringBuilder stringBuilder = new StringBuilder();
        HttpURLConnection urlConnection = null;
        try {
            urlConnection = (HttpURLConnection) url.openConnection();
        } catch (IOException e) { e.printStackTrace(); }
        try {
            InputStream content = new BufferedInputStream(
                urlConnection.getInputStream());
            BufferedReader reader = new BufferedReader(new InputStreamReader(content));
            String line;
            while ((line = reader.readLine()) != null) {
                stringBuilder.append(line);
            }
        } catch (IOException e) { e.printStackTrace(); }
        finally {
            urlConnection.disconnect();
        }
        return stringBuilder.toString();
    }
}
```




CONSUMING JSON SERVICES

```
{
  "appeld": "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"
},
{
  "appeld": "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"
}
]
```



CONSUMING JSON SERVICES

```
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", "Number of surveys in feed: " + jsonArray.length());
            //---print out the content of the json feed---
            for (int i = 0; i < jsonArray.length(); i++) {
                JSONObject jsonObject = jsonArray.getJSONObject(i);
                Toast.makeText(getBaseContext(), jsonObject.getString("appeld") + " - " + jsonObject.getString("inputTime"),
                    Toast.LENGTH_SHORT).show();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    new ReadJSONFeedTask().execute("http://extjs.org.cn/extjs/examples/grid/survey.html");
}
}
```

11/9/2018



Using OpenWeatherMap

- ❑ Use <https://openweathermap.org/>
- ❑ Register and obtain an API key.
- ❑ Use JSON service to get Weather information.

MainActivity

Latitude
37.77493

Longitude
-122.419416

Get Weather

Weather:
Postal Code
89118

Get Places

[{"id":800,"main":"Clear","description":"clear sky","icon":"01d"}]



References

- ☐ Textbook
- ☐ Android Documentation