Mobile Application Development - Topic 3

Maps

I need a map!

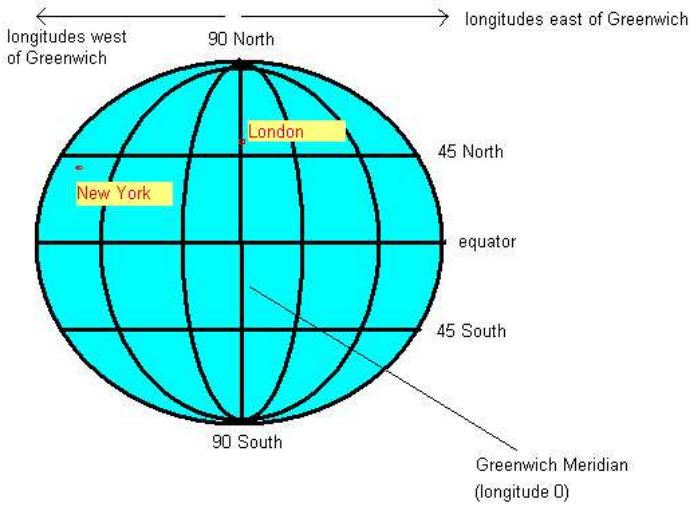
Most location-based apps include a map as the content view of their main activity. Android comes with inbuilt map functionality via Google Maps; however to use Google Maps you need to obtain an API key so we are going to use an alternative mapping library: ***osmdroid***, available here. (A ***librar*** is a collection of Java classes with related functionality, such as mapping).

osmdroid is a third-party open source library which uses maps from the OpenStreetMap project. OpenStreetMap is a global project to provide free and open mapping data which anyone can contribute to; see the website for more details. In using osmdroid, you will also see how to add external libraries to an Android project.

- Start a new project called ***Mapping***.

Latitude and Longitude

In order to understand location-based applications, it is important to understand the coordinate system used on the earth. The most common coordinate system uses ***latitude and longitude***. Latitude is a measure of how far north or south you are: the equator is at 0 degrees, while the North Pole is at 90 degrees North, we are at about 50 and Spain is at about 40. Longitude is a measure of how far east or west you are: 0 degrees of longitude is referred to as the ***Prime Meridian*** (or ***Greenwich Meridian***) and passes through Greenwich, London. By contrast Germany is located between approximately 7 degrees and 15 degrees East, while New York is at 74 degrees West and the west coast of North America at approximately 120 degrees West.



So a given point on the earth can be defined via its latitude and longitude. We are at, approximately, 50.9 North (latitude) and 1.4 West (longitude). By convention, latitudes north of the equator and longitudes east of Greenwich are treated as positive, so we can also define our position as **longitude -1.4, latitude +50.9**.

Mapping code

Now you can get down to some actual coding. Here is a sample app using the OSMDroid Android API:

```
package com.example.map;

import android.support.v7.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;

import org.osmdroid.config.Configuration;
import org.osmdroid.tileprovider.tilesource.TileSourceFactory;
import org.osmdroid.util.GeoPoint;
import org.osmdroid.views.MapView;

public class MainActivity extends AppCompatActivity
{


    MapView mv;
```

```
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {

        super.onCreate(savedInstanceState);

        // This line sets the user agent, a requirement to download OSM maps
        Configuration.getInstance().load(this, PreferenceManager.getDefaultSharedPreferences(this));

        setContentView(R.layout.activity_main);

        mv = findViewById(R.id.map1);

        mv.setMultiTouchControls(true);
        mv.getController().setZoom(16.0);
        mv.getController().setCenter(new GeoPoint(51.05,-0.72));
    }
}
```

The main XML layout file is here:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
<org.osmdroid.views.MapView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:enabled="true"
    android:clickable="true"
    android:id="@+id/map1"
    tilesource="Mapnik"
    />
</LinearLayout>
```

Note how this is working:

- In the XML layout, we add a component of type ***MapView***, but because MapView is not part of the standard Android library, we have to include its complete qualified name, including the package it belongs to, i.e. ***org.osmdroid.views.MapView***.
- The main activity is actually quite straightforward. We obtain the MapView using its ID (as we have done with other components before) and then set the zoom level (16; higher numbers are zoomed in further) and the centre point of the map, in latitude and longitude. For more on latitude and longitude, see here. For more on zoom levels, see below, and Google's article on zoom levels.

The manifest file and Permissions

The manifest file is an XML file describing the app and its components (e.g. the activities making up the app), as well as the app ***permissions*** (see below). It is called ***AndroidManifest.xml*** and can be found in ***manifests***.

Apps need to be granted ***permissions*** to perform sensitive operations. Sensitive operations can include:

- ***Using the internet*** - because this might incur a cost to the user, and have potential privacy/security concerns;
- ***Tracking your location*** - because this could be potentially abused (e.g. by stalkers)
- ***Reading from and writing to files*** - again this has security concerns.

Permissions are dealt with via two alternative mechanisms:

1. a simple mechanism, in which the user grants permissions when the app is installed, on Android 5 and downwards;
2. a more complex, but more flexible mechanism, in which users can turn permissions on and off at run time (i.e while the app is running) via the Android settings, on Android 6 upwards.

We will return to the latter mechanism later in the unit, but, to keep things simple, focus on the first mechanism now.

To add permissions, we add them to the manifest file. OSMDroid needs the following:

- READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE - because map tiles are cached on the device;
- ACCESS_FINE_LOCATION - to access the GPS
- ACCESS_WIFI_STATE and ACCESS_NETWORK_STATE - so that the wifi and network can be used for positioning if the GPS is not available.

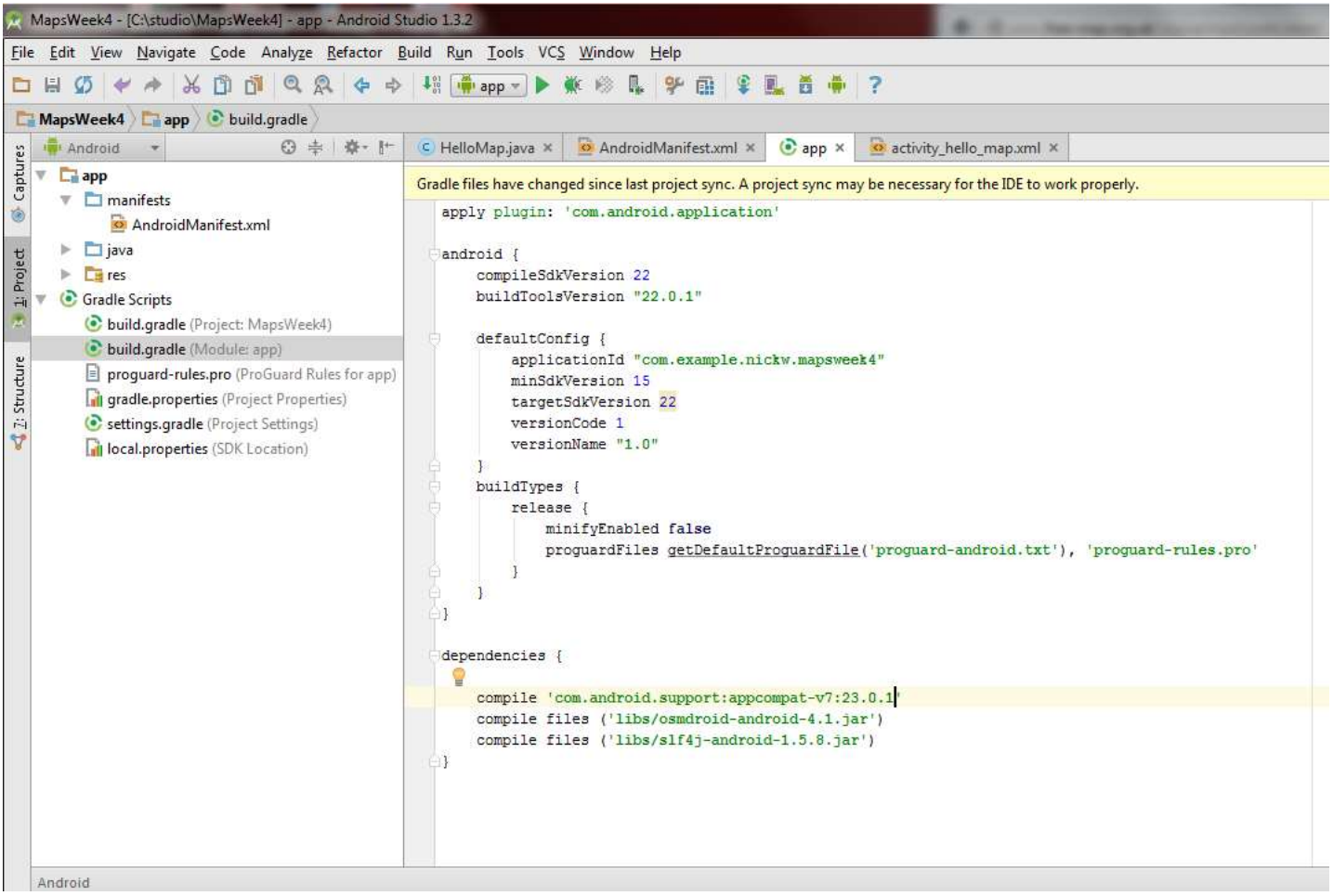
So ensure your manifest file contains the following permissions. They should go **before the <application> tag.**

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Linking the libraries to your app

You also need to link the OSMDroid mapping library to your app via the ***build.gradle*** file. ***build.gradle*** is a set of instructions used by the build tool ***Gradle*** to create your app from your code and any third-party libraries it may require. Gradle is a standalone tool (can be used outside Android Studio), but is included within Android Studio.

Note there are two ***build.gradle*** files, one for the whole project and one for your app specifically. It is the latter you need to edit. A build.gradle is shown below.



Look for the dependencies section in the build.gradle file. Change the dependencies section so it looks like this:

```
dependencies {
    // KEEP THE OTHER DEPENDENCIES AS THEY ARE - DO NOT CHANGE THEM!
    // .... other dependencies, e.g. support library, here ....

    // ADD THIS!
    implementation 'org.osmdroid:osmdroid-android:6.1.0'
}
```

Note you should use ***implementation*** if you are using Android Studio 3. ***compile*** is only used in Android Studio 2 or lower.

You also need to ensure that the targetSdkVersion is 22 or less in your build.gradle,as otherwise, Android will enforce runtime permission checking - see above.

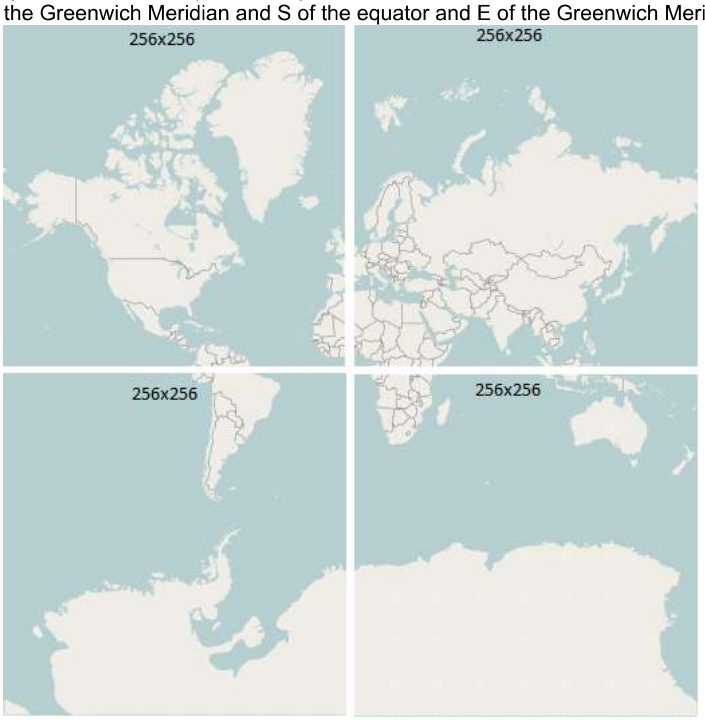
Where is the OSMDroid library coming from? If you have used a build system such as Maven in standard Java, you might recognise the technique used. It downloads the OSMDroid library from an ***online repository of Java libraries***. The specific repository used by default is ***Bintray JCenter***, which can be found here. Another common repository is Maven Central. Once the dependency has been downloaded, it will be saved on your compute so that it will not need to be downloaded next time you open the project.

The repositories used are specified in the ***project build.gradle*** file (the other one); you'll notice the ***jcenter()*** line in there which specifies that JCenter is being used.

Details on the zoom levels

If you are interested, this is what the zoom levels represent. There is a series of zoom levels, with 0 the most zoomed out and successive levels progressively zoomed in. How does this work? Basically, zoom level 0 is defined as a flat map of the entire world, occupying 256x256 pixels, so that 36 degrees of longitude becomes 256 pixels and 180 degrees of latitude becomes 256 pixels, as shown below:

Each successive zoom level zooms in by a factor of 2 in both directions, so that at zoom level 1, there are four 256x256 pixel tiles, each covering a quarter of the earth (N of the equator and W of the Greenwich Meridian; N of the equator and E of the Greenwich Meridian; S of the equator and W of the Greenwich Meridian and S of the equator and E of the Greenwich Meridian):



With progressive zoom levels, we continue zooming in by a factor of 2, so that zoom level 2 has 16 tiles (4x4), zoom level 3 has 64 (8x8), and so on. Each tile has an x and y coordinate where x=0 represents the leftmost column of tiles, y=0 represents the topmost row of tiles, and the tile with x=0 y=0 represents the top left tile (x=1 y=0 represents the second tile on the top row, and so on)

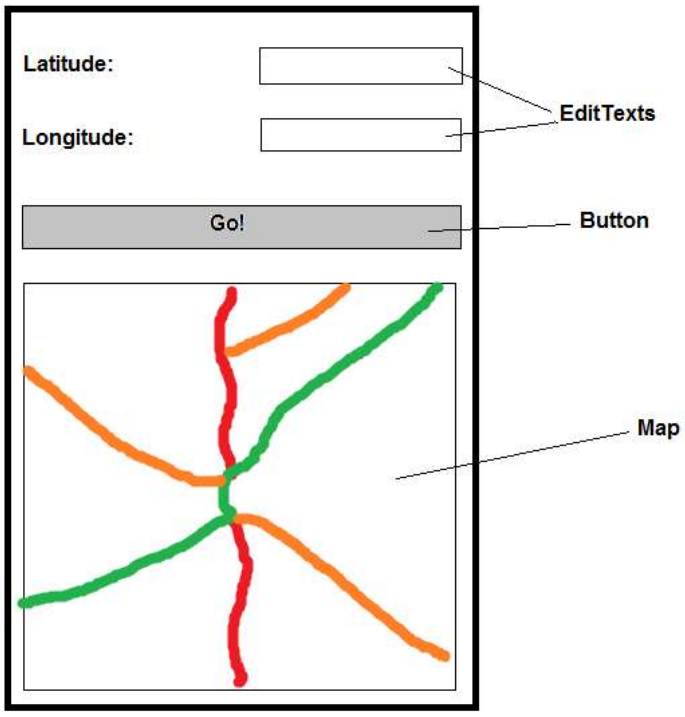
Images from OpenStreetMap, (c) OSM contributors, licenced under CC-by-SA (not ODBL as they are old images)

Exercise 1

- Try the example above out. It should show a map of Fernhurst, West Sussex (about 30 miles/50km east of here).
- Use the site informationfreeway.org to look up the latitude and longitude of your home town. Make the map centre on your home town instead.
- **Publish your work on GitHub**. See the Git Cheat Sheet for guidance. We will go through this in class.

Exercise 2 - Consolidation

This exercise allows you to revise the previous layouts and event handling topic. Enhance your app so that it has a UI like this. ***Create the XML by hand; do not use the UI designer.***



When the user clicks "Go", the map should move to the latitude and longitude that the user entered in the two EditTexts**. When finished, commit the changes and push your work to GitHub again!**