

Mobile GIS

Utilizing Google Maps in Android

Ahmad SYAUQI Ahsan

Background

- Geographic Information System (GIS) has been adopted for many range of applications including facility management, emergency response, vehicle monitoring, and much more.
- Currently, mobile technology (both software and hardware) has been advanced rapidly.
 - There are many mobile-based services, including Mapping, has been announced.
 - The most powerful hardware released to market (as of October 2011) is 1,5GHz Dual Core Processor with 1GB RAM.
- According to [Canalys](#), Smartphone market growth 73% year-on-year in the second quarter of 2011. With Android as the number one platform with 48% from total 107.7 millions unit shipped worldwide.
- **There are more benefit if we could combine those two technologies → Mobile GIS.**

Objectives

- Build a basic mobile GIS application in Android by utilizing Google Maps.
 - Display the map
 - Enable Pan and Zoom
 - Set display mode into Satellite view
 - Get information from the map

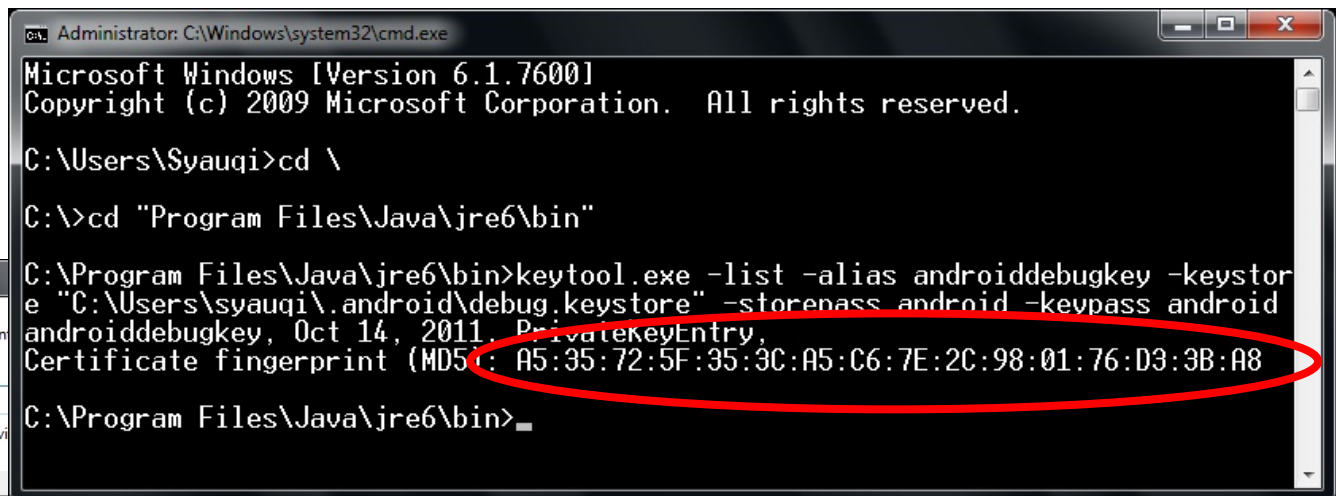
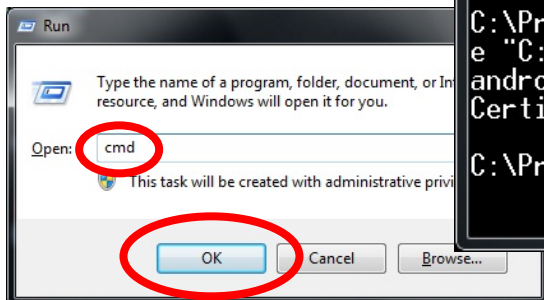
Google Maps API

- Google has provided an API (Application Programming interface) for accessing their map services.
- In order to use Maps Library/API in our Android application, we need an API Key.
- Each API Key is uniquely associated with a specific certificate, based on an MD5 fingerprint of our machine.

Obtain Maps API Key

1. Generate your MD5 certificate:
 - Run the Command Prompt (cmd.exe).
 - Go to "bin" directory under your "Java Development Kit" folder.
 - Issue the following command:

```
keytool.exe -list -alias androiddebugkey -keystore  
"C:\Users\username\.android\debug.keystore" -  
storepass android -keypass android
```

A screenshot of a Windows Command Prompt window running the keytool command. The output shows the certificate fingerprint (MD5) as A5:35:72:5F:35:3C:A5:C6:7E:2C:98:01:76:D3:3B:A8, which is circled in red.

```
C:\Windows\system32\cmd.exe  
Microsoft Windows [Version 6.1.7600]  
Copyright (c) 2009 Microsoft Corporation. All rights reserved.  
  
C:\Users\Syauqi>cd \  
  
C:\>cd "Program Files\Java\jre6\bin"  
  
C:\Program Files\Java\jre6\bin>keytool.exe -list -alias androiddebugkey -keystore  
"C:\Users\syauqi\.android\debug.keystore" -storepass android -keypass android  
androiddebugkey, Oct 14, 2011, PrivateKeyEntry,  
Certificate fingerprint (MD5): A5:35:72:5F:35:3C:A5:C6:7E:2C:98:01:76:D3:3B:A8  
  
C:\Program Files\Java\jre6\bin>
```

Obtain Maps API Key (cont')

2. Open your browser and navigate to <http://code.google.com/android/maps-api-signup.html>
3. Check the "I have read and agree with the terms and conditions (printable version)"
4. Put your MD5 certificate into the textfield.
5. Click the "Generate API Key" → you'll get your API Key.

The image shows two screenshots from the Google Maps API signup process. The left screenshot shows the 'I have read and agree with the terms and conditions' checkbox checked, with a red circle around it. Below it, the MD5 fingerprint field contains '75:EC:91:F1:E6:04:CE:67:14:16:DA:48:F6:63:1A:0E' and a 'Generate API Key' button is highlighted with a red arrow. The right screenshot shows the 'Thank you for signing up for an Android Maps API key!' message, with the generated API key '0w059uWmY4fnc2Djy5hWsE01StwYstuLmDAIs3g' circled in red. Below the key, it states 'This key is good for all apps signed with your certificate whose fingerprint is:' followed by the fingerprint '75:EC:91:F1:E6:04:CE:67:14:16:DA:48:F6:63:1A:0E'.

Google Maps API
Google Code Home > Google Maps API > Google Maps API Signup

Thank you for signing up for an Android Maps API key!

Your key is:

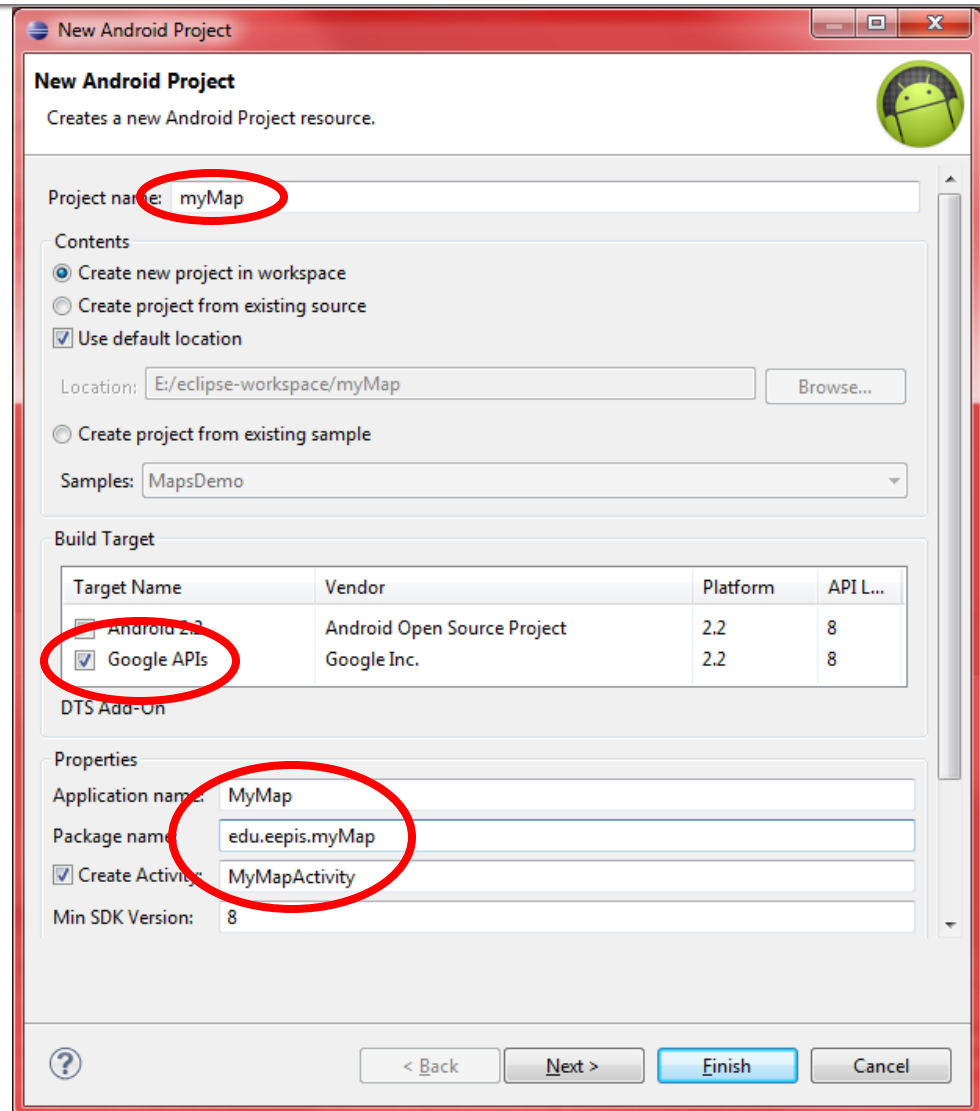
0w059uWmY4fnc2Djy5hWsE01StwYstuLmDAIs3g

This key is good for all apps signed with your certificate whose fingerprint is:

75:EC:91:F1:E6:04:CE:67:14:16:DA:48:F6:63:1A:0E

Create New Project

- Create new Android project:
File → New → Android Project
- Type "myMap" for the project name.
- For Build target, use Google APIs platform 2.2.



Edit the Android Manifest

- Google Maps library is not a part of the standard Android library, we must declare it in the Android Manifest.

Open the `AndroidManifest.xml` file and add the following as a child of the `<application>` element:

```
<uses-library android:name="com.google.android.maps" />
```

- Since we'll retrieve map data from the internet, we also need an access to the Internet. In the Android Manifest file, add the following as a child of the `<manifest>` element:

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


Edit the Android Manifest (cont')

- The `android-manifest.xml` should become like below:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.eepis.myMap"
    android:versionCode="1"
    android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MyMapActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <uses-library android:name="com.google.android.maps" />
    </application>
    <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Edit the layout

- Open the `res/layout/main.xml`.
- Delete the preconfigured `<TextView>` element.
- Add `<com.google.android.maps.MapView>` element.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <com.google.android.maps.MapView
    android:id="@+id/myMapView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:apiKey="put_your_API_Key_here" />
</LinearLayout>
```

Edit the MyMapActivity.java

- Open `src/edu.eepis.myMap/MyMapActivity.java`
- Change the `MyMapActivity` class to extends `MapActivity` instead of `Activity`
- Add the following inside the `MyMapActivity` class.

```
protected boolean isRouteDisplayed() {return false;}
```

- The `MyMapActivity.java` should become like below.

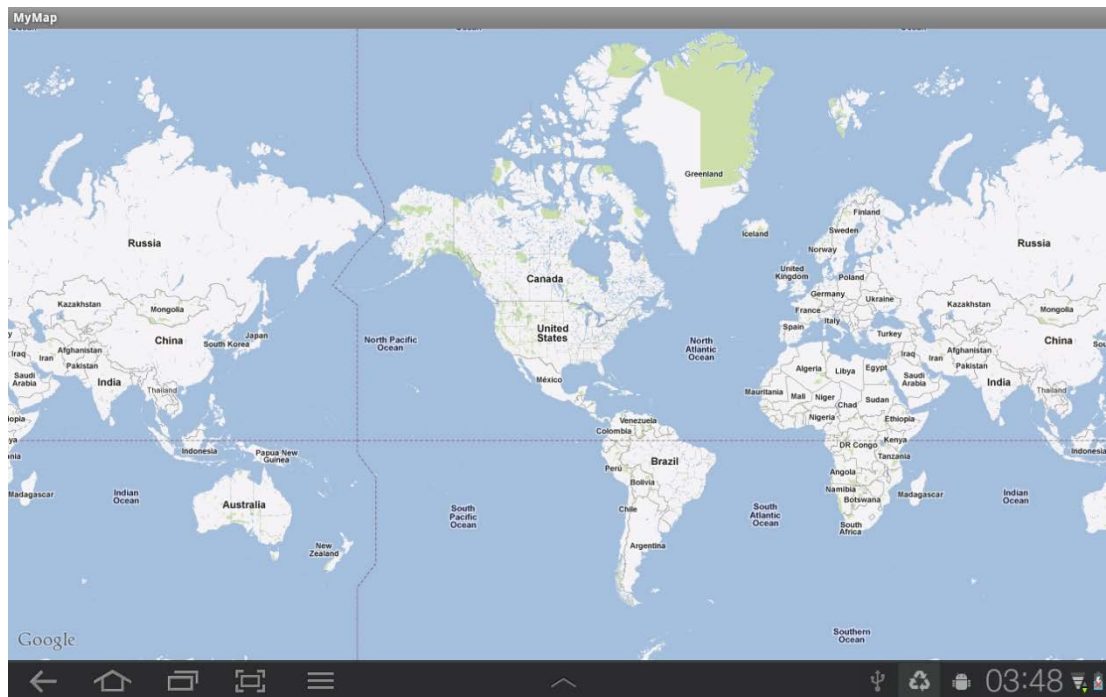
```
package edu.eepis.myMap;

import android.os.Bundle;
import com.google.android.maps.MapActivity;

public class MyMapActivity extends MapActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
    protected boolean isRouteDisplayed() {return false;}
}
```

First Run

- Run the application.
- The application should be able to display the map, but we cannot do anything with the map.
- Next step: enabling pan and zoom function.



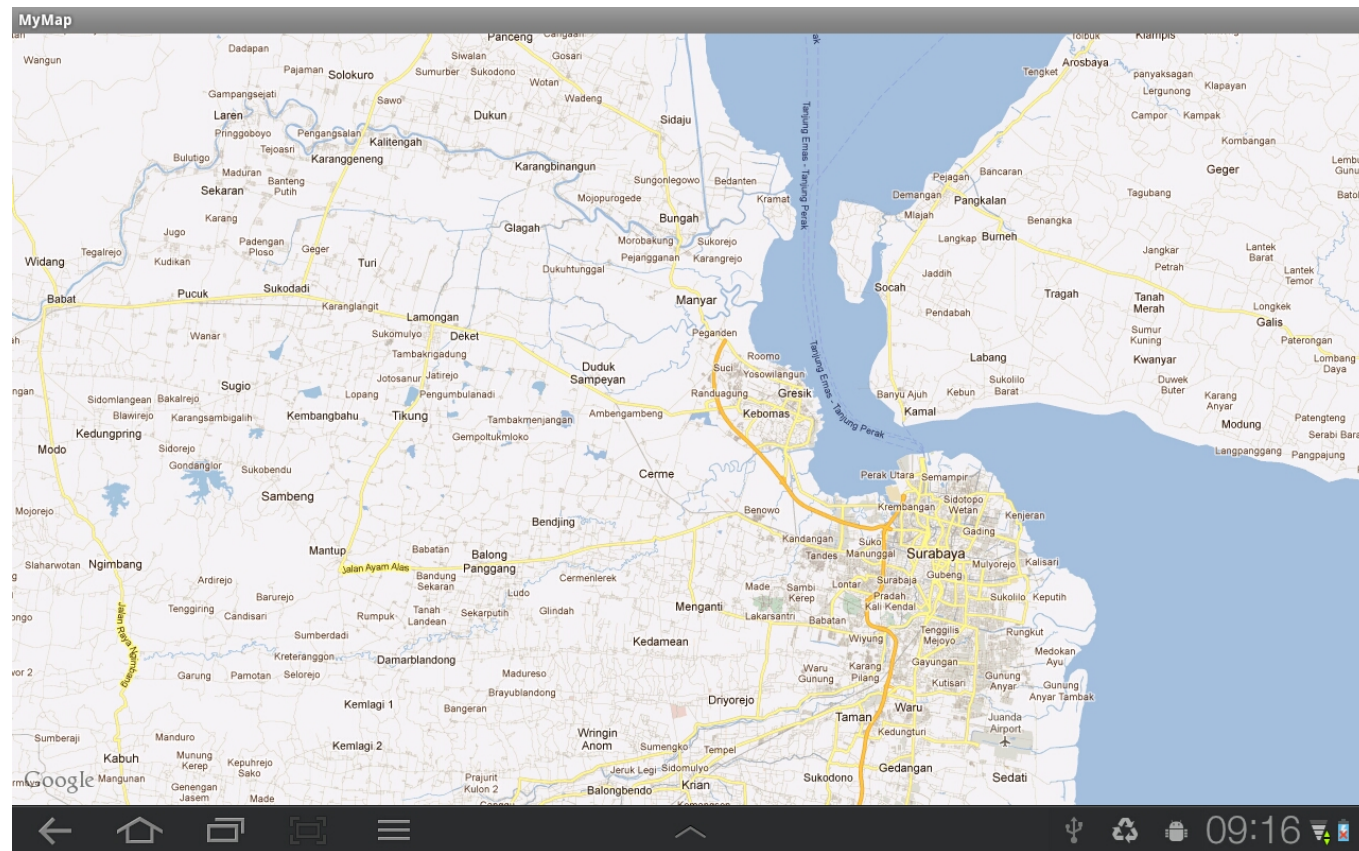
Enabling Pan and Zoom

- To enable Pan and Zoom, fortunately, Google Maps library has provided built-in these two function.
- So, we only need to add one single line into the layout file:
 - Open the `res/layout/main.xml`
 - Add the following line as an attribute of `<com.google.android.maps.MapView>` element.

```
android:clickable="true"
```

Second Run

- Run the application.
- Now we should be able to pan and zoom the application.



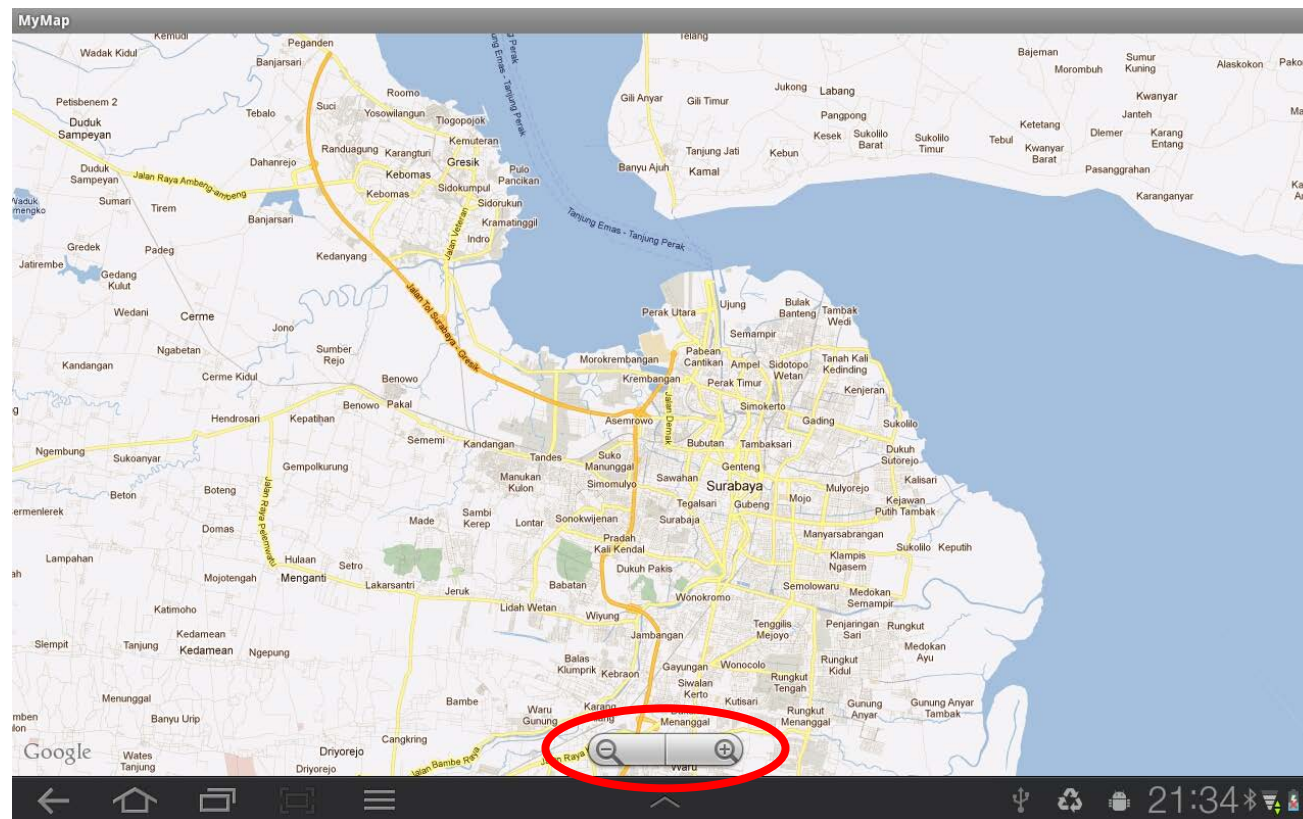
Using built-in zoom control

- We also could utilize built-in zoom control of the `MapView` component.
- Follow these steps to enable this control:
 1. Open the `MyMapActivity.java`.
 2. Add the following lines inside `onCreate` method for `MyMapActivity` class.

```
MapView myMapView = (MapView) findViewById(R.id.myMapView);  
myMapView.setBuiltInZoomControls(true);
```

Third Run

- Now we have zoom in and zoom out buttons located at bottom-center of screen.
- These buttons automatically disappear if there are no activity .



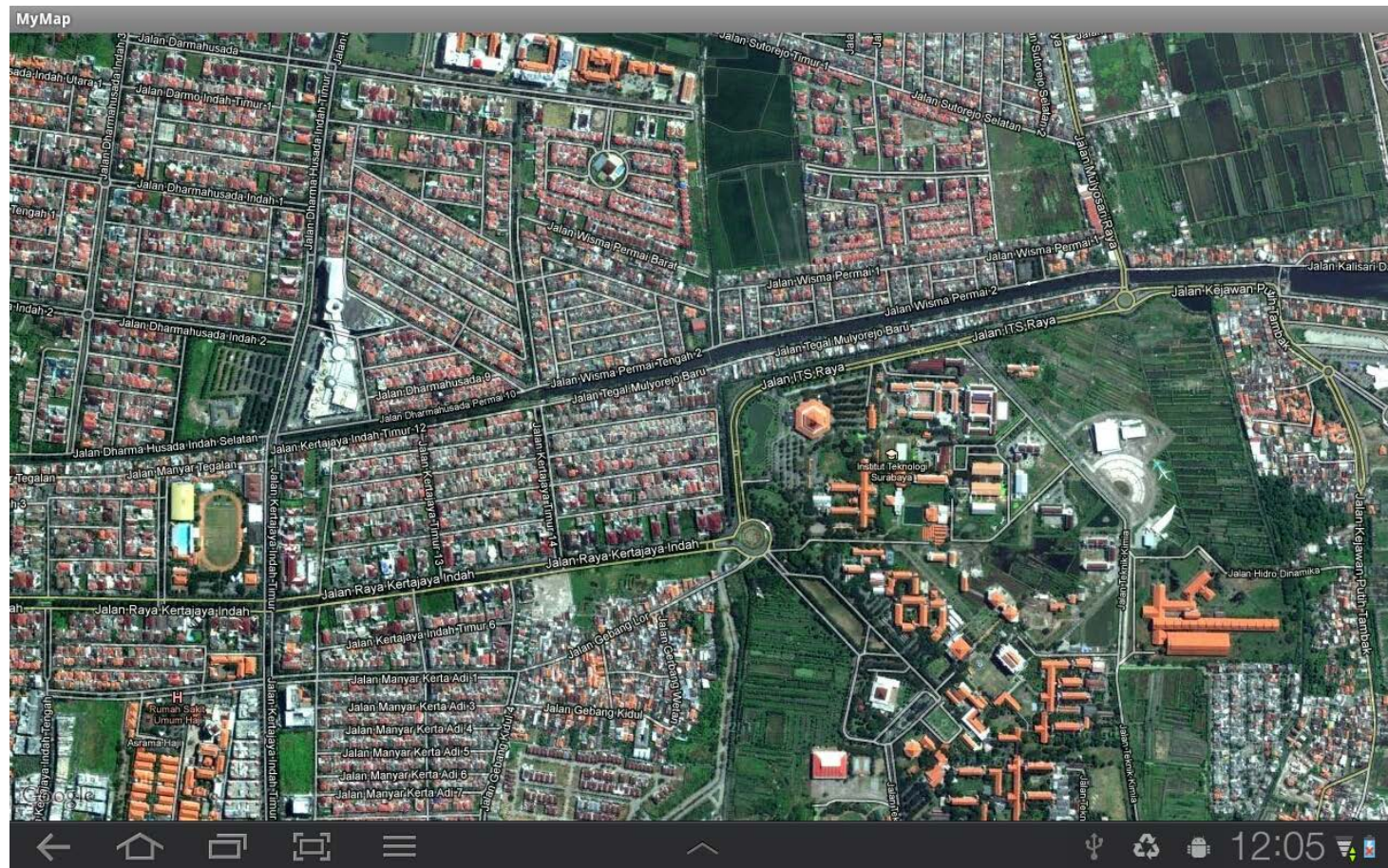
Displaying satellite view

- The **MapView** component displays a street view as a default.
- If we need to display the map in satellite view, follow these steps:
 1. Open the **MyMapActivity.java**.
 2. Add the following lines inside **onCreate** method for **MyMapActivity** class.

```
myMapView.setSatellite(true);
```

Fourth Run

- The application will display the map in satellite view.



Getting information from the map

- In GIS applications, it is common to get information from the map.
- In this example, we'll make the application shows the coordinate of a place that get touched.

Creating overlay class

- In order to put an item that lays over the map, we need to create an overlay class.
- The following code will create an overlay class along with `onTouchEvent` that show the coordinate using `Toast`.
Put this code inside the `MyMapActivity` class.

```
class MapOverlay extends Overlay {  
  
    public boolean onTouchEvent(MotionEvent event, MapView mapView) {  
        if (event.getAction() == 1) {  
            GeoPoint p = mapView.getProjection().fromPixels((int)  
                event.getX(), (int) event.getY());  
            Toast.makeText(getApplicationContext(), p.getLatitudeE6() / 1E6 +  
                ", " + p.getLongitudeE6() / 1E6, Toast.LENGTH_SHORT).show();  
        }  
        return false;  
    }  
}
```

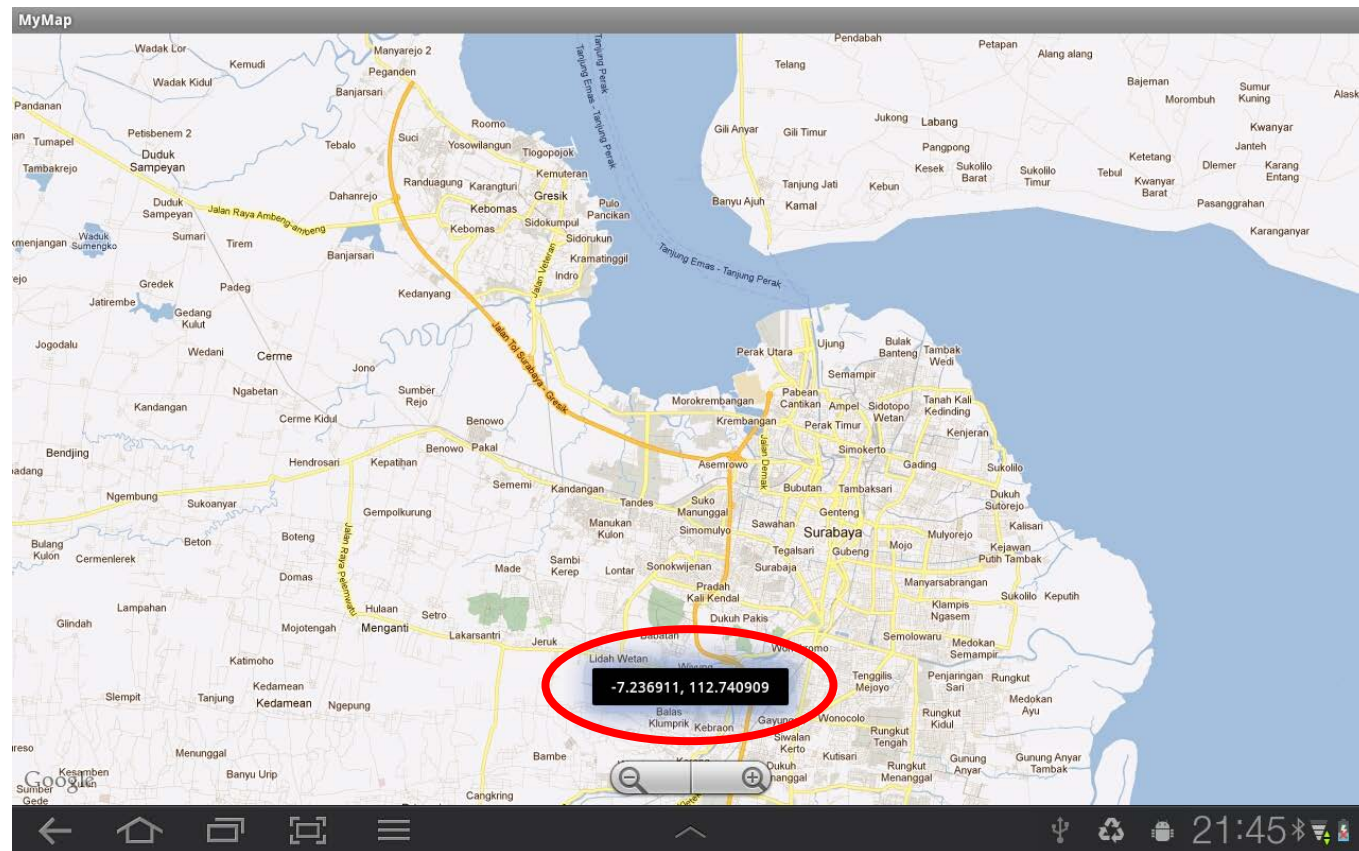
Using and arranging overlays

- We have created the overlay class in previous slide.
- Then, we need to use that class and arrange it with overlay of the map class.
- Insert the following code inside the `onCreate` method of `MyMapActivity` class (after the `myMapView.setSatellite(true);` line):

```
MapOverlay myOverlay = new MapOverlay();  
List<Overlay> myListOverlays = myMapView.getOverlays();  
myListOverlays.add(myOverlay);
```

Fifth Run

- The application will displays the coordinate (latitude & longitude) of location that we have just touched.



Displaying address

- In some case, we may want to display an address instead of a coordinate.
- In order to do that, we'll use a **Geocoder** class.
- **Geocoder** class has a **getFromLocation** method that will return an address from the given location.

Displaying address (cont')

- Modify `onTouchEvent()` method of `MapOverlay` class by replacing the

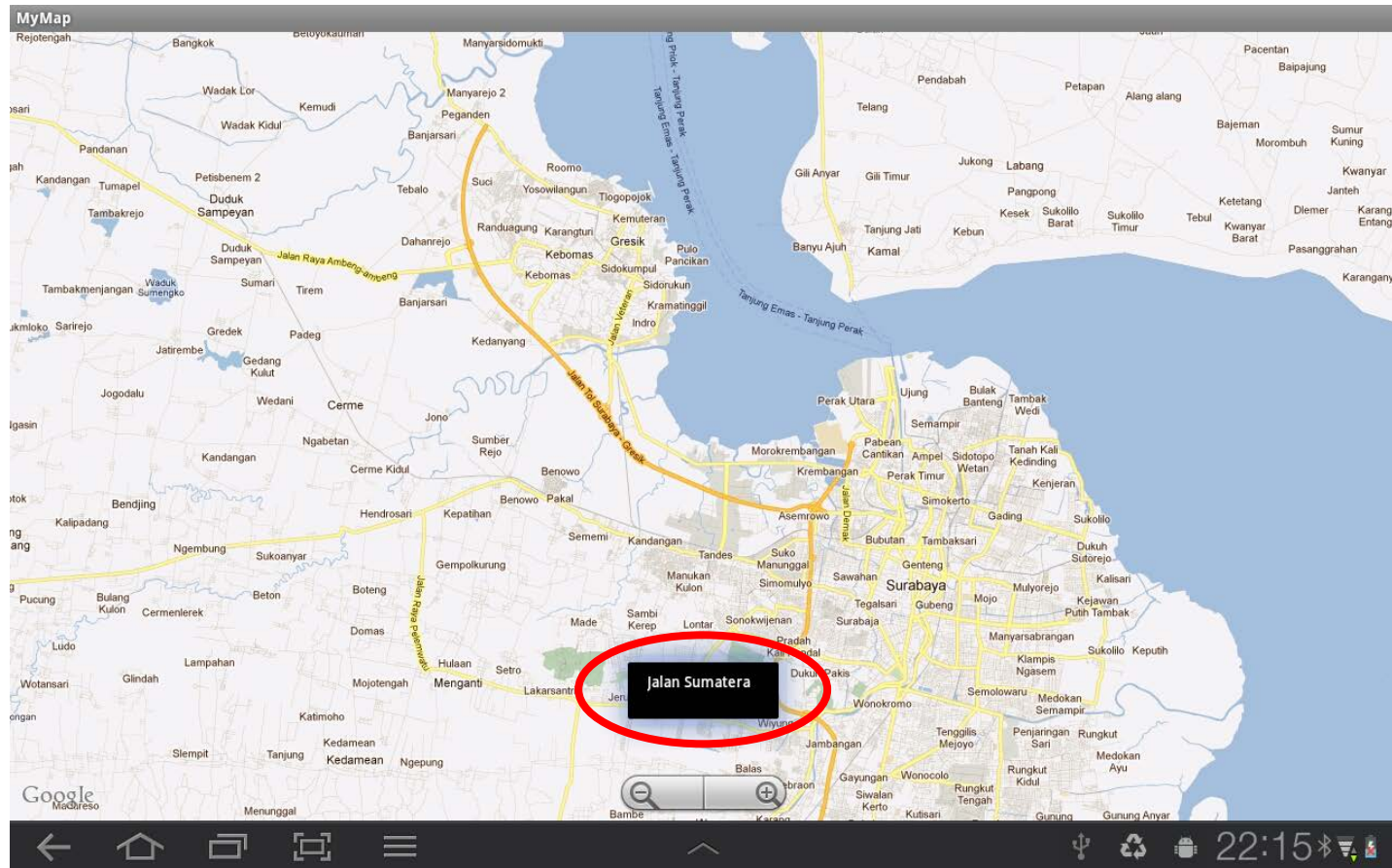
```
Toast.makeText(getBaseContext(), p.getLatitudeE6() / 1E6 +  
    ", " + p.getLongitudeE6() / 1E6, Toast.LENGTH_SHORT).show();
```

line with the following code.

```
Geocoder myGeocoder = new Geocoder(getBaseContext(),  
Locale.getDefault());  
try {  
    String myText = "";  
    List<Address> myAddresses =  
        myGeocoder.getFromLocation(p.getLatitudeE6() / 1E6,  
            p.getLongitudeE6() / 1E6, 1);  
    if (myAddresses.size() > 0) {  
        for (int i=0; i<myAddresses.get(0).getMaxAddressLineIndex()-1; i++)  
            myText += myAddresses.get(0).getAddressLine(i) + "\n";  
    }  
    Toast.makeText(getBaseContext(), myText, Toast.LENGTH_SHORT).show();  
} catch (IOException e){}
```


Sixth Run

- When we touch the screen, an address shows up.



Q&A