



# Programming with Android:
# The Google Maps Library

**Luca Bedogni** **Marco Di Felice**

**Dipartimento di Scienze dell'Informazione**

**Università di Bologna**

# Outline

- **Google Maps**: History and General Information
- **Google Maps Library**: Installation and Registration
- **Google Maps Library**: *MapView* and *MapActivity*
- **Google Maps Library**: *MapController*
- **Google Maps Library**: *Overlay* Definition
- **Google Maps Library**: *GPS* Localization
- **Google Maps Library**: *Geocoding*

# Android: Gmaps Important Dates …

- **2004** → Google Inc bought the australian company *Where 2 Technologies*, that developed a prototype WebMap system.
- **2005** (February) →Google Maps was announced
- **2006** → Google Maps updated to use the same satellite image database as Google Earth
- **2007** → Google Street View launched
- **2010** → On Christmas and New Years day, mobile usage of Google Maps surpassed desktop usage for the first time
- **NOW**: Google Maps, Google Sky, Google Moon, Google Mars, Google Transit, Google Aerial View, etc

# Android: Gmaps Stats and Information

- Maps are based on a variant of *Mercator* projections.
- Frequency of updates for satellite images ~ 3 years

SERVICE **COVERAGE**

**Map Tiles**: 209 countries over 218 → **~96**%

**Street View**: 23 countries over 218 → **~10**%

**Traffic View**: 22 countries over 218 → **~10**%

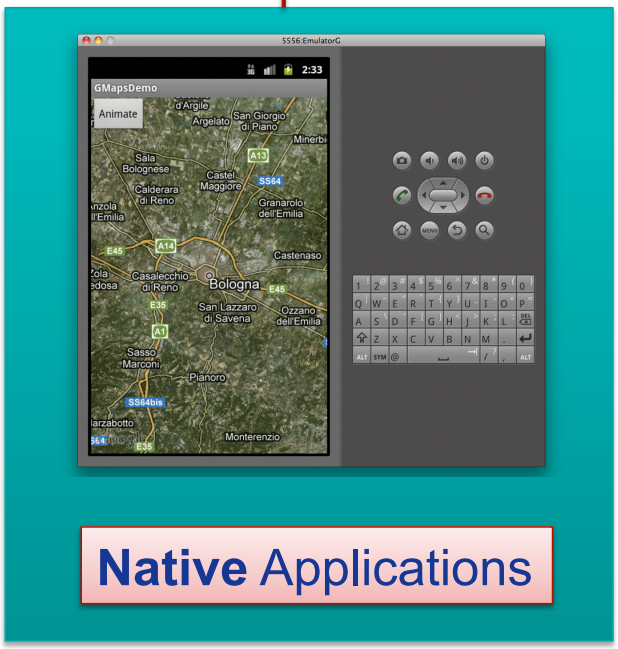
**Business info**: 37 countries over 218 → **~17**%

# Android: Deploying Map-based Apps

## Deploying **Map-based** Applications in Android


**WebView** +
Google Maps +
Web technologies

**Hybrid** Applications




**Native** Applications

# Android: Deploying Map-based Apps

Two versions of Android Google Maps API

**API v1**

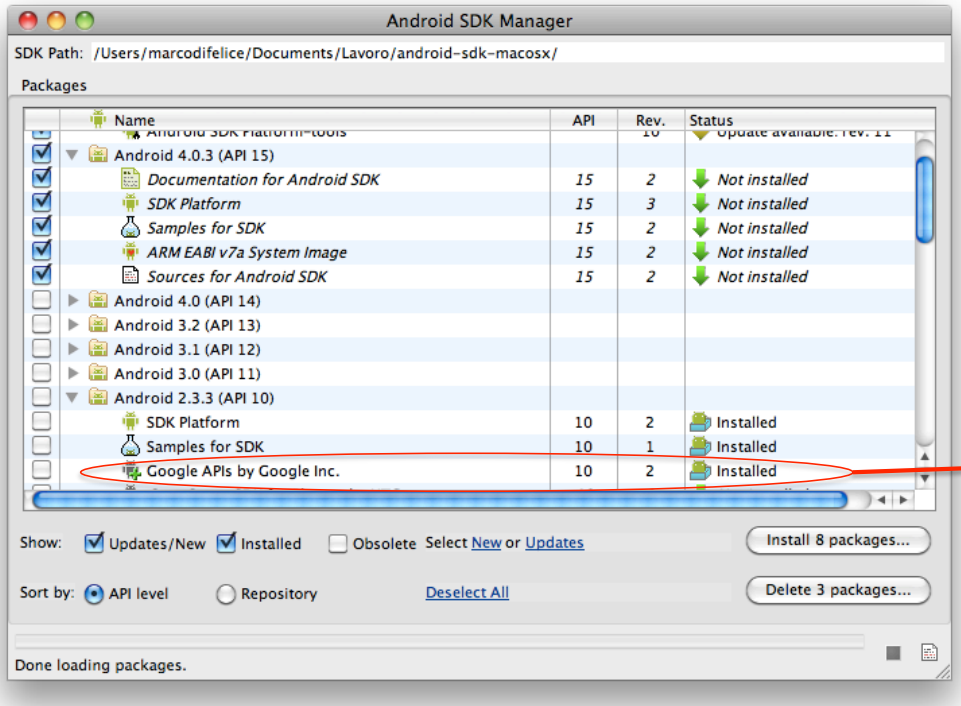
- Deprecated, not supported anymore since 18th March 2013.
- Still used for Android device with versions < 3.0 (unless API set is extended with support packages)

**API v2**

- Different installation procedures.
- Novel methods to insert a Map inside an Android app.
- Improved caching and visualization capabilities.

# Android: Installing Google APIs

**STEP -1**: Install **Google APIs** to use the Maps in a native application.




Window → Android SDK Manager →Installed packages

Check GoogleAPIs is **installed**, or **install** it otherwise

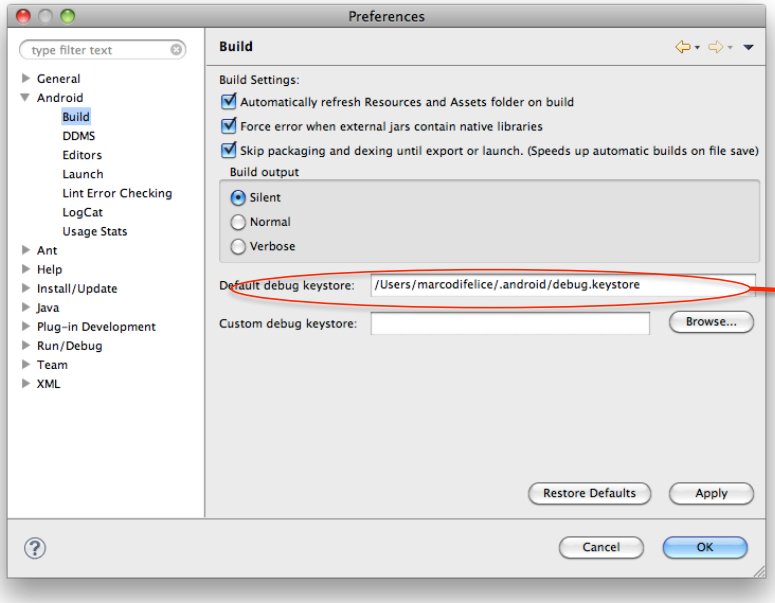
# Android: Getting a Google Maps API Key

## STEP 0: http://code.google.com/intl/it-IT/android/maps-api-signup.html

# Android: Getting a Google Maps API Key

**STEP 0**: Get a valid **Maps API Key** to utilize the Google Maps library.

**0.1**: Retrieve the fingerprint MD5 of the certificate used to sign the apps.




Window → Preferences → Android → Build

Get the debug **keystore** path

# Android: Getting a Google Maps API Key

**STEP 0**: Get a valid **Maps API Key** to utilize the Google Maps library.

**0.1**: Retrieve the fingerprint MD5 of the certificate used to sign the apps.

```
mylaptop:~ marco$ keytool -list -keystore /Users/
marcodifelice/.android/debug.keystore  -storepass
android -keypass android
…
androiddebugkey, Feb 1, 2011, PrivateKeyEntry,
Certificate fingerprint (MD5): A2:34:B1:A3:A5:BB:
11:21:21:B3:20:56:92:12:AB:DB
```

# Android: Google MAPs library overview

## What can I do with Google MAPs library in Android?

1. **Integrate** a Google Map into an Android application
2. **Control** the Map visualization **options**
3. **Customize** the Map
4. **Integrate** the Map with GPS data

# Android: Google MAPs library overview

**Instantiate these objects** to integrate a Google Map

1. **MapView** (com.google.android.maps.MapView)
   - A **View** that displays a Map
2. **MapActivity**(com.google.android.maps.MapActivity)
   - Extension of the **Activity** class
   - Base class with code to manage the necessities of any Activity that displays a MapView …

# Android: Google Maps library overview

Define a **MapView** in the layout file (main.xml)

```
<LinearLayout>
…
< com.google.android.maps.MapView
                    android:layout_width="fill_parent"
                    android:layout_height="fill_parent"
                    android:id="@+id/map
                    android:apiKey="**********"
/>
…
</LinearLayout>
```

Paste your **Google API Key** here

# Android: Google Maps library overview

Define a **MapActivity** in the Java code ..

```
public class MapDemoActivity extends MapActivity {
… … … … … …
}
```

Implement the method **isRouteDisplayed()**:

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

*Is my app giving routing information?*

# Android: Google Maps library overview

## Set the permissions in the **AndroidManifest.xml**

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

## Set the libraries in the **AndroidManifest.xml**

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

# Android: Google Maps library overview

Some **methods** of a MapView …

MapView options:

- public void **setSatellite**(boolean enable)
- public void **setTraffic**(boolean enable)
- public void **setStreetView**(boolean enable)

MapView interaction modes:

- public void **setClickable**(boolean enable)
- public void **setBuiltInZoomControls** (boolean enable)

# Android: Google Maps library overview

- How to **control** the Google Map **visualization**?

… Through the **MapController** object!

  - **Center** the Map at a given location
  - **Zoom** in/out operations
  - **Enable** animations on the map

How to get a MapController from a MapView?

```
public MapController getController()
```

# Android: Google Maps library overview

Some **methods** of a MapController …

Center the map at a given location:

- public void **setCenter(Geopoint p)**

A **GeoPoint** defines a location on the Map …

```
GeoPoint BOLOGNA=new GeoPoint(44494290,11346526);
```

***<latitude, longitude>*** *in microgrades, i.e. grade*10$^6$*

# Android: Google Maps library overview

Some **methods** of a MapController …

Control the **Zoom** IN/OUT operations

- public void **zoomIn()**
- public void **zoomOut**()

Enable animations on the map

- public void **animateTo(GeoPoint gp)**
- public void **animateTo**(Geopoint gp, Message msg)
- public void **animateTo**(Geopoint gp, Runnable runnable)

# Android: Google Maps library overview

**Overlays** → Map customizations, markers with icon, title, snippet and associated events (e.g. touch, tap, etc).

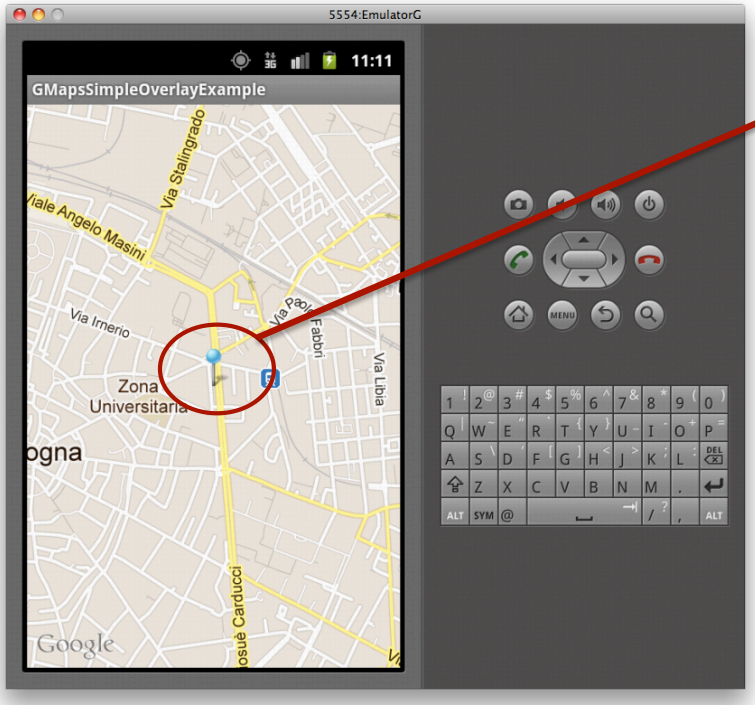
- **Overlay** (Base class representing an Overlay on the map)
- **ItemizedOverlay** (Extension of Overlay, List of **OverlayItems**)
- **MyLocationOverlay** (Extension of Overlay for drawing user's **current location** on the map, and/or a **compass-rose** inset)

ADDING an OVERLAY to a MAPVIEW

```
mapView.getOverlays().add(newOverlay);
```

# Android: Google Maps library overview

**Overlay** → Basic class to add a Marker to the Map …




- **Extend** the Overlay class
- **Override** the method:

```
draw(Canvas c, MapView m,
boolean b)
```

- **Add** the Overlay to the Map:

```
mapView.getOverlay.add
(mySimpleOverlay)
```

# Android: Google Maps library overview

```
class SimpleOverlay extends Overlay {
       private GeoPoint gp;
       private Bitmap marker;

       public SimpleOverlay(GeoPoint p, Bitmap d) {
          gp=p;
          marker=d;
       }

       public void draw(Canvas canvas, MapView mapView, boolean
shadow) {
          super.draw(canvas, mapView, shadow);
          Point point=new Point();
          mapView.getProjection().toPixels(gp, point);
          canvas.drawBitmap(marker, point.x-24, point.y-48, null);

       }
}
```

# Android: Google Maps library overview

**ItemizedOverlay** → Overlay Extension, Collection of OverlayItem

OVERLAYITEM Constructor

```
OverlayItem(Geopoint gp, String title, String snippet)
```

ITEMIZEDITEM Constructor

```
ItemizedOverlay(Drawable defaultMarker)
```

**Extend** the ItemizedOverlay and **Override** the following methods:

- public int **size**()
- protectedOverlayItem **createItem**(int i)

# Android: Google Maps library overview

**ItemizedOverlay** → Overlay Extension, Collection of OverlayItem

OVERLAYITEM Constructor

```
OverlayItem(Geopoint gp, String title, String snippet)
```

ITEMIZEDITEM Constructor

```
ItemizedOverlay(Drawable defaultMarker)
```

Other methods:

- protected void **populate**()
- protected boolean **onTap**(int index)

# Android: Google Maps library overview

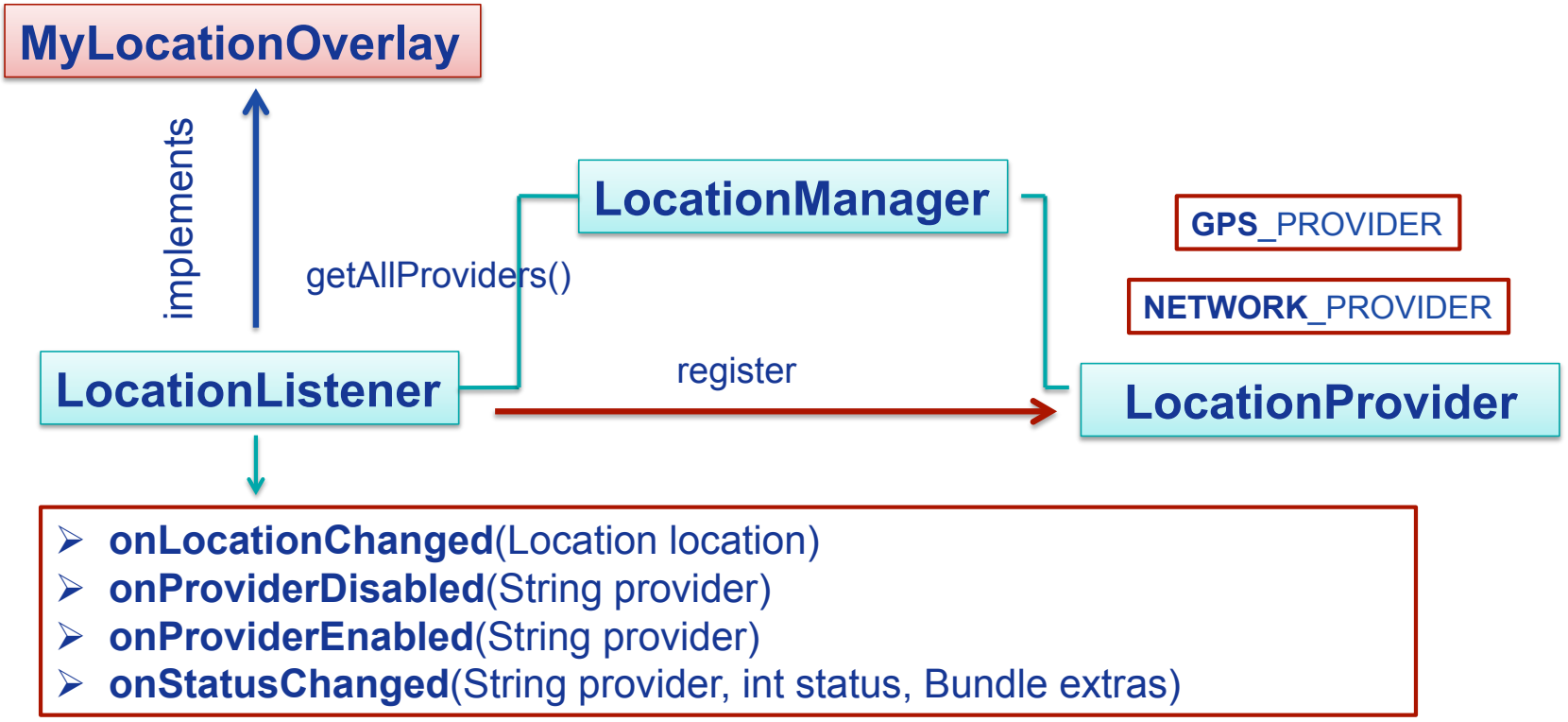
**MyLocationOverlay** → Overlay Extension, Draw user's position

MYLOCATIONOVERLAY Constructor

```
MyLocationOverlay(Context context, MapView mapview)
```

- public boolean **enableMyLocation**()
- public boolean **disableMyLocation**()
- public GeoPoint **getMyLocation**()
- public Location **getLastFix()**
- public boolean **runOnFirstFix(Runnable)**

# Android: Google Maps library overview



# Android: Google Maps library overview

**GeoCoding** → Technique to convert an Address into a GeoPoint, or viceversa …

Implemented by the Geocoder class

```
public Geocoder(Context contex)
```

Main methods:

- public **List<Address> getFromLocation**(double latitude, double longitude, int maxResults)
- public **List<Address> getFromLocationName**(String locationName, int maxResults)

# Android: Deploying Map-based Apps

Two versions of Android Google Maps API

**API v1**

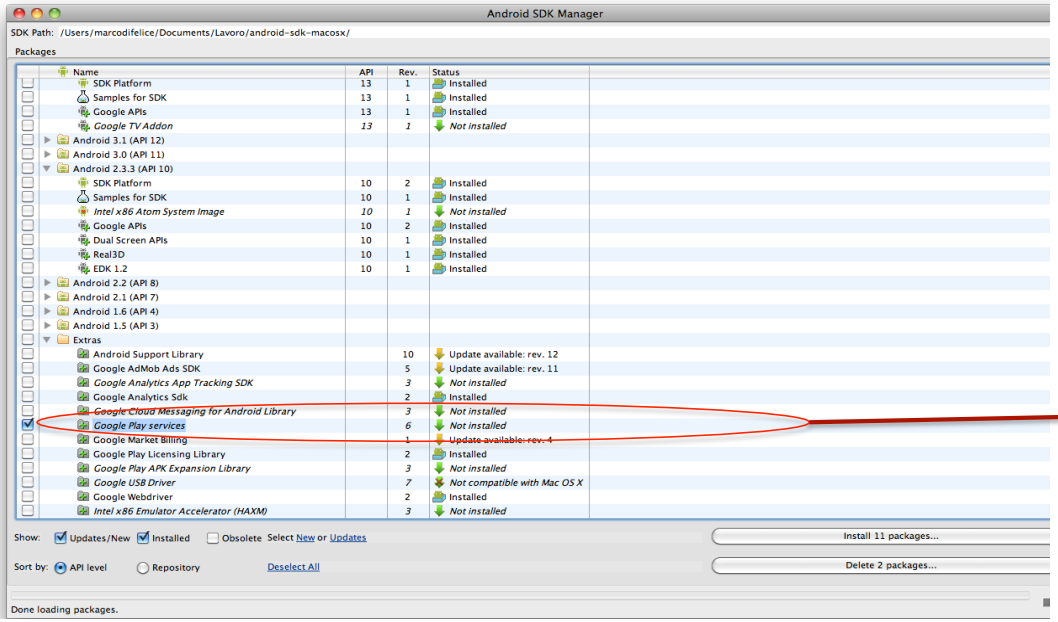
- Deprecated, not supported anymore since 18th March 2013.
- Still used for Android device with versions < 3.0 (unless API set is extended with support packages)

**API v2**

- Different installation procedures.
- Novel methods to insert a Map inside an Android app.
- Improved caching and visualization capabilities.

# Android: Installing Google APIs

**STEP -1**: Install and Setup **Google Play Service SDK**




Window → Android SDK Manager →Installed packages

Check Google Play is **installed**, or **install** it otherwise

http://developer.android.com/google/play-services/setup.html

# Android: Getting a Google Play API Key

**STEP 0**: Get a valid Google Play **API Key** to utilize the Google Maps library.

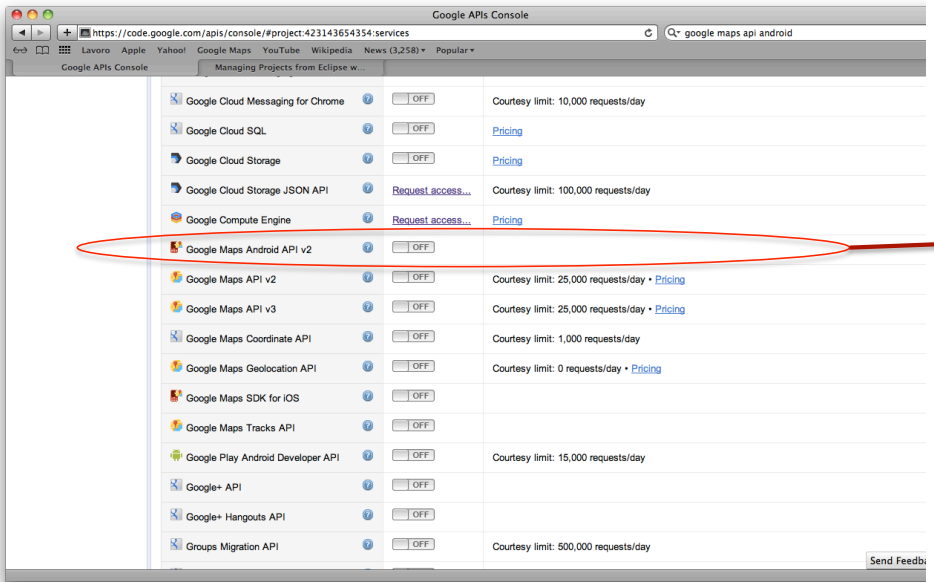
**0.1**: Retrieve the fingerprint SHA1 of the certificate used to sign the apps.

```
mylaptop:~ marco$ keytool -list -keystore /Users/
marcodifelice/.android/debug.keystore  -storepass
android -keypass android
…
androiddebugkey, Feb 1, 2011, PrivateKeyEntry,
Certificate fingerprint (SHA1): A2:34:B1:A3:A5:BB:
11:21:21:B3:20:56:92:12:AB:DB
```

# Android: Getting a Google Play API Key

**STEP 1**: Navigate with a browser to https://accounts.google.com/

**1.1**: Select the Google service you intend to use for your apps.




Enable Google Maps Android v2 API

# Android: Getting a Google Play API Key

**STEP 1**: Navigate with a browser to https://accounts.google.com/

**1.2**: Get an **Google Play API Activation Key**

- Select the API Acess

- Insert the SHA1 Key, followed by the package's name:

BB:0D:AC:74:D3:21:E1:43:67:71:9B:62:91:AF:A1:66:6E:44:5D:75;
**com.example.android.mapexample**

- Generate and save the obtained Activation Key
- **For each application/package → get a new Activation Key.**

# Android: Google MAPs library overview

## What can I do with Google MAPs v2 library in Android?

1. **Integrate** a Google Map into an Android application
2. **Manage** the camera
3. **Add** information layers to the Map
4. **Manage** user events

# Android: Inserting a Map inside the App

**Permissions** should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

- Internet Access
- Localization capabilities
- Access to Google Web services
- OpenGL ES version 2 libraries
- Access to network state

# Android: Inserting a Map inside the App

**Permissions** should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

```
<meta-data
 android:name="com.google.android.maps.v2.API_KEY"
 android:value="API_activation_key"/>
```

```
<permission
android:name="com.example.mapdemo.permission.MAPS_RECEIVE"
android:protectionLevel="signature"/>
<uses-permission
android:name="com.example.mapdemo.permission.MAPS_RECEIVE"/>
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true"/>
```

# Android: Inserting a Map inside the App

**Permissions** should be added to the AndroidManifest.xml, and the Activation Key must be specified in the meta-data.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.REA
D_GSERVICES"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

# Android: Inserting a Map inside the App

In order to insert a Google Map into a mobile Application:

- Add a **MapFragment** to the current Activity:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
  android:id="@+id/map"

android:name="com.google.android.gms.maps.MapFragm
ent"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />
```

# Android: Fragments

**Fragment** → A portion of the user interface in an Activity.

Introduced from **Android 3.0** (API Level 11)

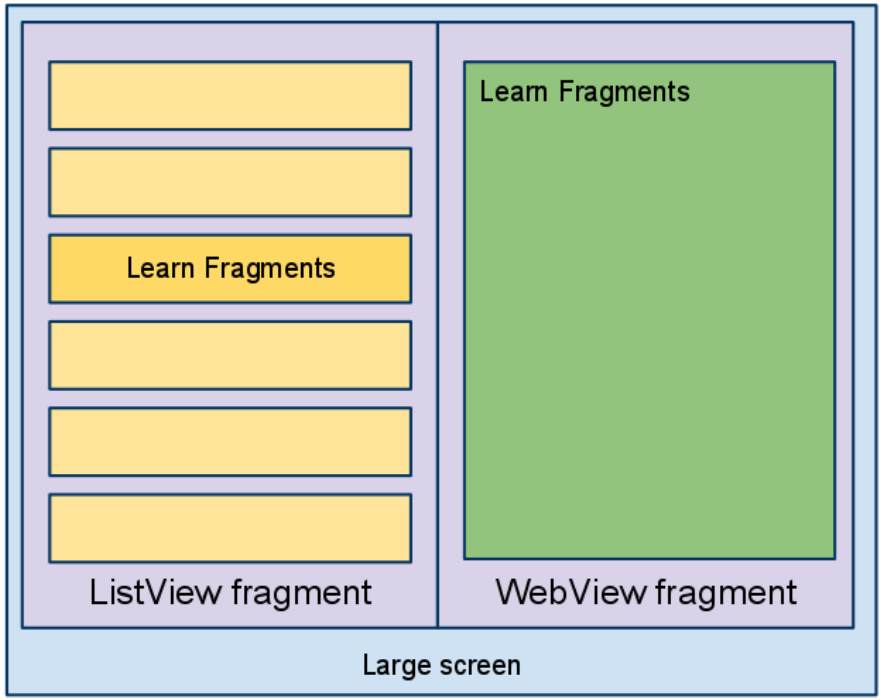
Practically, a Fragment is a modular section of an Activity.

DESIGN PHILOSOPHY

- **Structure** an Activity as a collection of Fragments.
- **Reuse** a Fragment on different Activities …

# Android: Fragments Design Philosophy

**EXAMPLE**: Structuring an Application using 1 Activity and 2 Fragments.



# Android: Inserting a Map inside the App

A MapFragment is a container of the **GoogleMap** object, which is a View containing the map and managing the events.

```
private GoogleMap mMap;
...
mMap = ((MapFragment) getFragmentManager
().findFragmentById(R.id.map)).getMap();
```

Differences with **Android Maps v1 libs**:

- No need to use a MapActivity, use a regular Activity instead.
- Improved caching and drawing functionalities.

# Android: Customize the Map

How to customize the Google Map?

- Define the **Map type**, governing the overall representation of the map

```
nMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
```

**Normal** → Typical road map.

**Hybrid** → Satellite photograph data with road maps added.

**Satellite** → Satellite photograph data. Road and feature labels are not visible.

**Terrain** → Topographic data. The map includes colors, contour lines and labels, and perspective shading.

**None** → no tiles, empty grid.

# Android: Customize the Map

The **LatLng** class allows to define a point on the map, expressed through the latitude/longitude coordinates.

```
private static final LatLng BOLOGNA_POINT = new
LatLng(44.496781,11.356387);
```

```
private static final LatLng FLORENCE_POINT = new
LatLng(43.771373,11.248069);
```

**LatLng** class (API v2) → **Geopoint** class (API v1)

# Android: Customize the Map

Developers can handle the **events** on the Google Map.

Events are managed through the **listener mechanism** seen so far …

**CLICK** events → Implement the `OnMapClickListener` interface and the `OnMapLongClickListener` method.

**CAMERA** events → Implement the `OnCameraChangeListener` interface and the `onCameraChange(CameraPosition)` method.

# Android: Customize the Map

Developers can handle the **events** on the Google Map.

```
public class MainActivity extends Activity
                              implements OnMapClickListener {
private GoogleMap mMap;

protected void onCreate(Bundle savedInstanceState) {
…
mMap.setOnMapClickListener(this);
…
}

public void onMapClick(LatLng position) {
      // Handle the click events here …
}
```

# Android: Customize the Map

How to customize the Google Map?

- Define the **properties of the Camera** applied to the Map.

**Location** → expressed in forms of latitude/longitude coordinates.

**Zoom** → defines the scale levels of the map.

**Bearing** → defines the map orientation, i.e. the direction in which a vertical line on the map points, measured in degrees clockwise from north.

**Tilt** → viewing angle, measured as degrees from the nadir.

# Android: Customize the Map

How to customize the Google Map?

- Define the **properties of the Camera** applied to the Map.

**Location** → expressed in forms of latitude/longitude coordinates.

**Zoom** → defines the scale levels of the map.

**Bearing** → defines the map orientation, i.e. the direction in which a vertical line on the map points, measured in degrees clockwise from north.

**Tilt** → viewing angle, measured as degrees from the nadir.

# Android: Customize the Map

Camera properties can be set individually, or collectively through the **CameraPosition** object.

```
private static final LatLng BOLOGNA_POINT = new
LatLng(44.496781,11.356387);
```

```
CameraPosition cameraPosition = new CameraPosition.
     Builder()
    .target(BOLOGNA_POINT)
     .zoom(17)
    .bearing(90)
     .tilt(30)
    .build();
```

# Android: Customize the Map

Two methods to modify the position of the camera:

```
mMap.moveCamera(cameraPosition);
```

- Update the camera properties immediately.

```
mMap.animateCamera(cameraPosition);
```

```
mMap.animateCamera(cameraPosition, duration, call);
```

- Update the camera properties through an animation, eventually adding a delay and a callback to be invoked when the animation stops.


# Android: Customize the Map

**Markers** can be used to identify locations on the GoogleMap.

Markers can be customized in terms of:

- **Icon** to be displayed
- **Position** of the marker on the map
- **Title** and text to be displayed
- **Events** to be managed

# Android: Customize the Map

**Markers** can be used to identify locations on the GoogleMap.

```
private static final LatLng BOLOGNA_POINT = new
LatLng(44.496781,11.356387);

Marker bologna = myMap.addMarker(newMarkerOptions
().position(BOLOGNA_POINT));
```

```
Marker bologna= mMap.addMarker(new MarkerOptions()
                  .position(Bologna)
                  .title("Bologna downtown")
                  .snippet("Visit the city centre"));
```

# Android: Customize the Map

**Markers** can be used to identify locations on the GoogleMap.

**EVENTS** associated to a Marker:

**ClickEvents** → implement the `OnMarkerClickListener` interface, and the `onMarkerClick(Marker)` method.

**DragEvents** → implement the `OnMarkerDragListener` interface, and the `onMarkerDragEnd(Marker)` method.

**InfoWindow Click Events** → implement the `OnInfoWindowClickListener` interface, and the `onInfoWindowClick(Marker)` method.

# Android: Customize the Map

**Shapes** can be used to identify sections of the GoogleMap.

**Polylines** → define a set of LatLong objects, and connect them through a set of lines. Possible to define the stroke and colors of the lines.

**Polygons** → define a set of LatLong objects, and connect them through a closed polygon. Possible to define the stroke and colors of the lines.

**Circles** → define a LatLong object and a radius, and draw a circle centered at the point. Define pen color/stroke as above.

# Android: Customize the Map

**Shapes** can be used to identify sections of the GoogleMap.

```
PolygonOptions rectOptions = new PolygonOptions()
        .add(BOLOGNA_P1)
        .add(BOLOGNA_P2)
        .add(BOLOGNA_P3);
Polygon polyline = mMap.addPolygon(rectOptions);

CircleOptions circleOptions = new CircleOptions()
       .center(BOLOGNA_P1)
       .radius(1000)
       .strokeColor(Color.RED);

Circle circle = mMap.addCircle(circleOptions);
```