Android Development

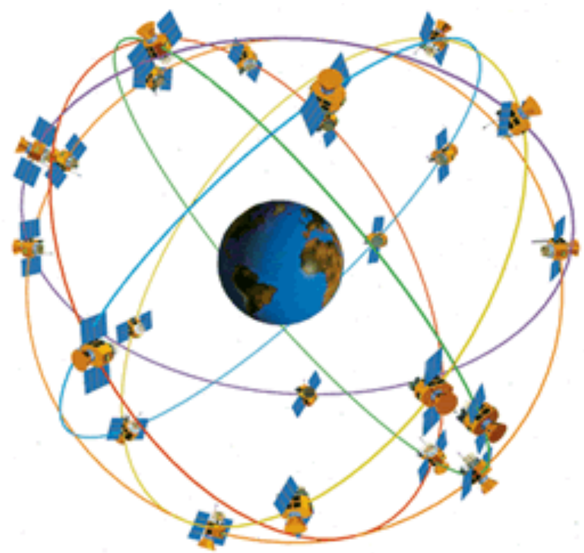
Lecture 6
Location and Maps

Lecture Summary

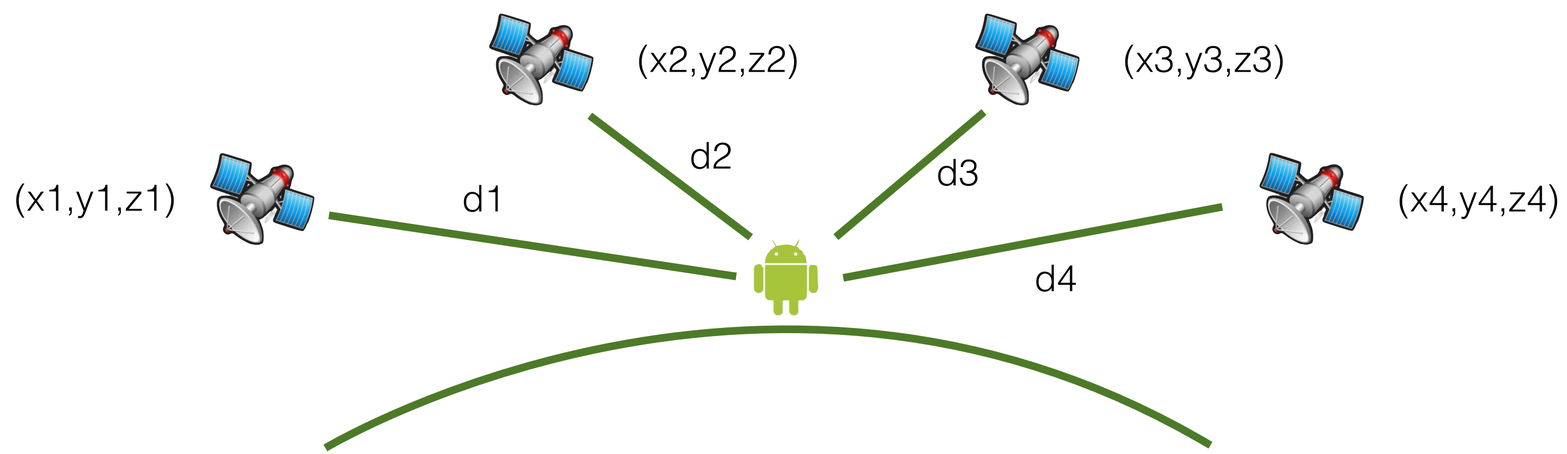
- GPS Introduction
- Location Based Services
- Android & Location
- Location Service
- LBS Application Model
- Google Maps Android API v2
- Setup
- MapFragment & Compatibility
- Marker
- Events
- Geocoder
- Map Application & Navigator

What is GPS ?

The Global Positioning System (GPS) is a satellite-based navigation system made up of a network of 24 satellites placed into orbit by the U.S. Department of Defense. GPS was originally intended for military applications, but in the 1980s, the government made the system available for civilian use.

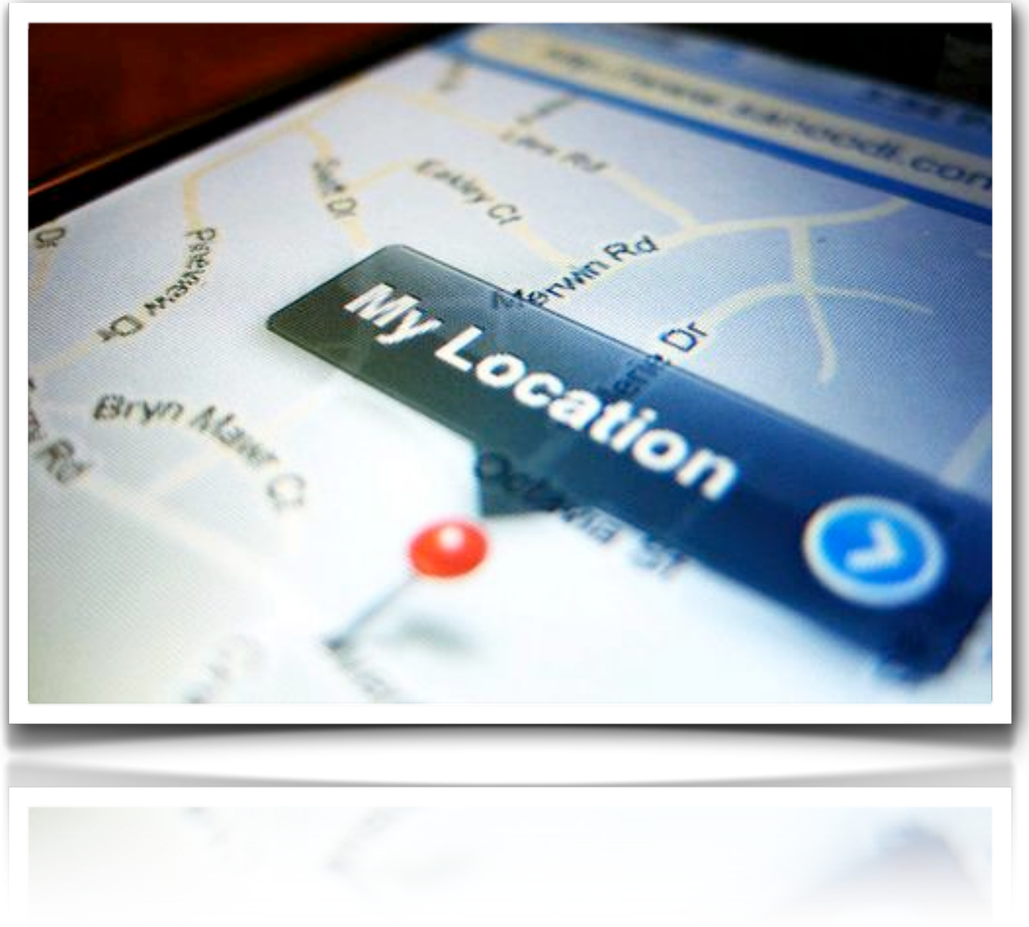


A GPS receiver's job is to locate four or more of these satellites, figure out the distance to each, and use this information to deduce its own location. This operation is based on a simple mathematical principle called trilateration.



Location Based Services

Location-Based Service (LBS) is an information or entertainment service, accessible with mobile devices through the mobile network and utilizing the ability to make use of the geographical position of the mobile device.



Location Based Services

foursquare

Google latitude

facebook Places

bing

waze way to go

tomtom

Google places

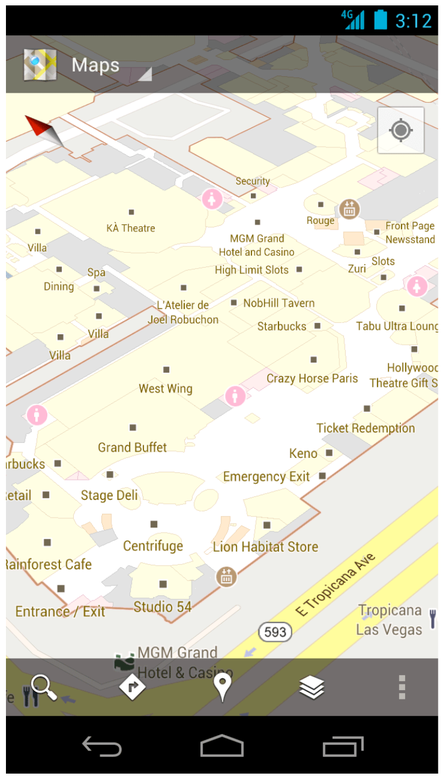
Google

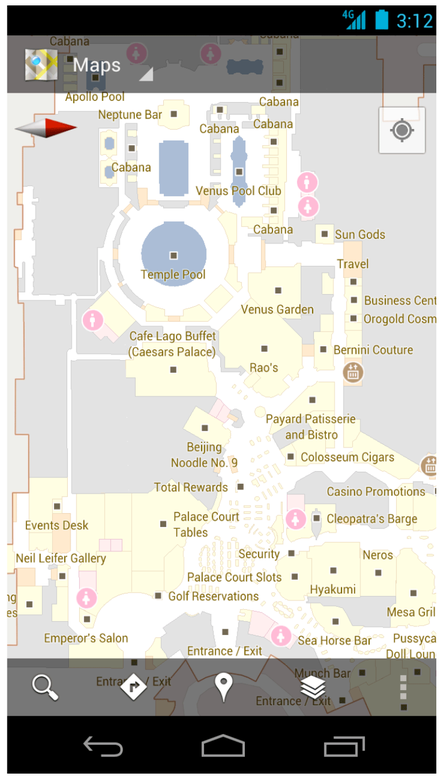
Location Based Services

- Mobile Phones use several methods to determine device location. They are:
  - Cell ID: Each cell tower worldwide has a unique identifier and knows its latitude and longitude. It allows a mobile phone to easily retrieve and know an approximate estimation of its location inside the range of the cell tower (Km).
  - Triangulation: When your mobile device is in range of more than one cell tower, the tower has the ability to evaluate the direction and distance of your signal and triangulate to determine user location.
  - GPS: Using Global Positioning System your mobile phone is able to determine device location with an high accuracy. The downside of this solution are:
    - Reduced Battery Life
    - Unreliable Availability: GPS works only if your device is able to “see” and receive the signal of at least 4 satellites.

Android & Location

Hardware Support
Location API





Native Map Library

Integrated Navigator

Android & Location

- Knowing where the user is allows your application to be smarter and deliver better information to the user. When developing a location-aware application for Android, you can utilize GPS and Android's Network Location Provider to acquire the user location.
- The GPS is most accurate, it only works outdoors, it quickly consumes battery power, and doesn't return the location as quickly as users want.
- Android's Network Location Provider determines user location using cell tower and Wi-Fi signals, providing location information in a way that works indoors and outdoors, responds faster, and uses less battery power.
- To obtain the user location in your application, you can use both GPS and the Network Location Provider, or just one.

Location Services

- Android gives your applications access to the location services supported by the device through the classes in the android.location package. The central component of the location framework is the LocationManager system service, which provides APIs to determine location and bearing of the underlying device (if available).
- As with other system services, you do not instantiate a LocationManager directly. Rather, you request an instance from the system by calling getSystemService(Context.LOCATION_SERVICE). The method returns a handle to a new LocationManager instance.
- Once your application has a LocationManager, your application is able to do three things:
  - Query for the list of all LocationProviders for the last known user location.
  - Register/unregister for periodic updates of the user's current location from a location provider (specified either by criteria or name).
  - Register/unregister for a given Intent to be fired if the device comes within a given proximity (specified by radius in meters) of a given lat/long.

Location Services

- User location on Android works with callbacks. You indicate that you'd like to receive location updates from the LocationManager("Location Manager") by calling requestLocationUpdates(), passing it a LocationListener. Your LocationListener must implement several callback methods that the Location Manager calls when the user location changes or when the status of the service changes.

```
// Acquire a reference to the system Location Manager
LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to location updates
LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
      // Called when a new location is found by the network location provider.
      ...
    }
    public void onStatusChanged(String provider, int status, Bundle extras) {}
    public void onProviderEnabled(String provider) {}
    public void onProviderDisabled(String provider) {}
  };
// Register the listener with the Location Manager to receive location updates
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, locationListener);
```

requestLocationUpdates(...)

- The frequency of notification or new locations may be controlled using the minTime and minDistance parameters.
- If minTime is greater than 0, the LocationManager could potentially rest for minTime milliseconds between location updates to conserve power.
- If minDistance is greater than 0, a location will only be broadcast if the device moves by minDistance meters.
- To obtain notifications as frequently as possible, set both parameters to 0.

requestLocationUpdates(long minTime, float minDistance, Criteria criteria, PendingIntent intent)

requestLocationUpdates(long minTime, float minDistance, Criteria criteria, LocationListener listener, Looper looper)

requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)

requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener, Looper looper)

requestLocationUpdates(String provider, long minTime, float minDistance, PendingIntent intent)

Location Permissions

- In order to receive location updates from NETWORK_PROVIDER or GPS_PROVIDER, you must request user permission by declaring either theACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permission, respectively, in your Android manifest file.
- If you are using both NETWORK_PROVIDER and GPS_PROVIDER, then you need to request only the ACCESS_FINE_LOCATION permission, because it includes permission for both providers. (Permission for ACCESS_COARSE_LOCATION includes permission only forNETWORK_PROVIDER.)

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

LBS Challenges

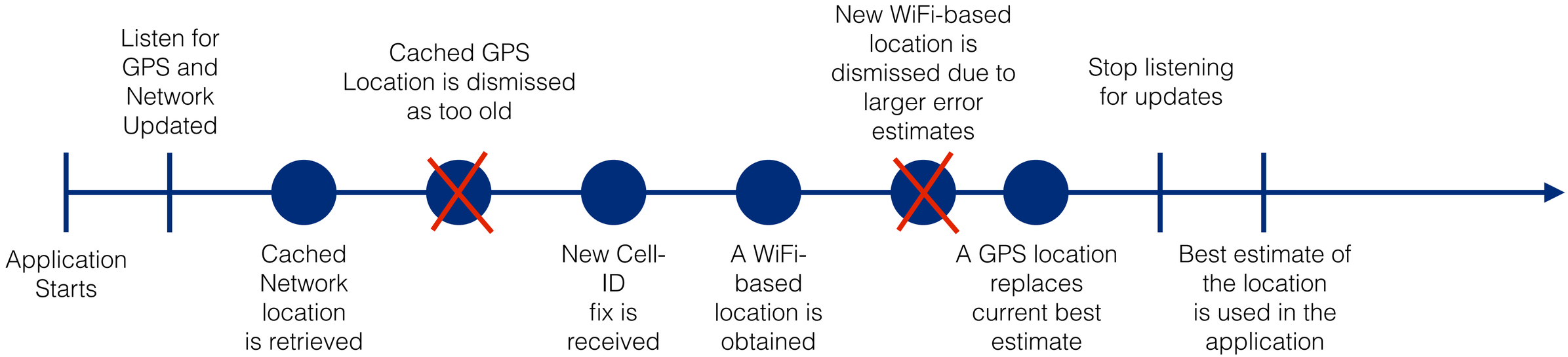
- Obtaining user location from a mobile device can be complicated. There are several reasons why a location reading (regardless of the source) can contain errors and be inaccurate. Some sources of error in the user location include:
  - Multitude of location sources: GPS, Cell-ID, and Wi-Fi can each provide a clue to users location. Determining which to use and trust is a matter of trade-offs in accuracy, speed, and battery-efficiency.
  - User movement: Because the user location changes, you must account for movement by re-estimating user location every so often.
  - Varying accuracy: Location estimates coming from each location source are not consistent in their accuracy. A location obtained 10 seconds ago from one source might be more accurate than the newest location from another or same source.

LBS Application Model

- Location-based applications are now commonplace, but due to the less than optimal accuracy, user movement, the multitude of methods to obtain the location, and the desire to conserve battery, getting user location is complicated. To overcome the obstacles of obtaining a good user location while preserving battery power, you must define a consistent model that specifies how your application obtains the user location. This model includes when you start and stop listening for updates and when to use cached location data.
- A common flow to obtain user location could be:
  - Start application.
  - Sometime later, start listening for updates from desired location providers.
  - Keep a "current best estimate" of location by filtering out new, but less accurate fixes.
  - Stop listening for location updates.
  - Take advantage of the last best location estimate.

http://developer.android.com/guide/topics/location/obtaining-user-location.html

LBS Application Model



- Show a timeline for an example model that visualizes the period in which an application is listening for location updates and the events that occur during that time.

- This model could be extended to be applied to different type applications that use the location for example to tag user generated content with a location or that want to track the user while he/she is moving.

http://developer.android.com/guide/topics/location/obtaining-user-location.html

Getting a fast fix with the last known location

- The time it takes for your location listener to receive the first location fix is often too long for users wait.
- Until a more accurate location is provided to your location listener, you could use a cached location by calling getLastKnownLocation(String)
- Returns a Location indicating the data from the last known location fix obtained from the given provider. This can be done without starting the provider. Note that this location could be out-of-date, for example if the device was turned off and moved to another location.

```
LocationProvider locationProvider = LocationManager.NETWORK_PROVIDER;
// Or use LocationManager.GPS_PROVIDER

Location lastKnownLocation = locationManager.getLastKnownLocation(locationProvider);
```

Deciding when to stop listening for updates

- The logic of deciding when new fixes are no longer necessary might range from very simple to very complex depending on your application.
- A short gap between when the location is acquired and when the location is used, improves the accuracy of the estimate. Always beware that listening for a long time consumes a lot of battery power, so as soon as you have the information you need, you should stop listening for updates by calling removeUpdates(PendingIntent).
- Removes any current registration for location updates of the current activity with the given PendingIntent. Following this call, updates will no longer occur for this intent.

```
// Remove the listener you previously added
locationManager.removeUpdates(locationListener);
```

Maintaining a current best estimate

- You might expect that the most recent location fix is the most accurate. However, because the accuracy of a location fix varies, the most recent fix is not always the best.
- You should include logic for choosing location fixes based on several criteria. The criteria also varies depending on the use-cases of the application and field testing.
- You can validate the accuracy of a location fix:
  - Check if the location retrieved is significantly newer than the previous estimate.
  - Check if the accuracy claimed by the location is better or worse than the previous estimate.
  - Check which provider the new location is from and determine if you trust it more.

http://developer.android.com/guide/topics/location/obtaining-user-location.html

Maintaining a current best estimate

```
private static final int TWO_MINUTES = 1000 * 60 * 2;

/** Determines whether one Location reading is better than the current
Location fix
  * @param location  The new Location that you want to evaluate
  * @param currentBestLocation  The current Location fix, to which you want
to compare the new one
  */
protected boolean isBetterLocation(Location location, Location
currentBestLocation) {
    if (currentBestLocation == null) {
        // A new location is always better than no location
        return true;
    }

    // Check whether the new location fix is newer or older
    long timeDelta = location.getTime() - currentBestLocation.getTime();
    boolean isSignificantlyNewer = timeDelta > TWO_MINUTES;
    boolean isSignificantlyOlder = timeDelta < -TWO_MINUTES;
    boolean isNewer = timeDelta > 0;

    // If it's been more than two minutes since the current location, use
the new location
    // because the user has likely moved
    if (isSignificantlyNewer) {
        return true;
    // If the new location is more than two minutes older, it must be worse
    } else if (isSignificantlyOlder) {
        return false;
    }

    // Check whether the new location fix is more or less accurate
    int accuracyDelta = (int) (location.getAccuracy() -
currentBestLocation.getAccuracy());
    boolean isLessAccurate = accuracyDelta > 0;
    boolean isMoreAccurate = accuracyDelta < 0;
    boolean isSignificantlyLessAccurate = accuracyDelta > 200;

    // Check if the old and new location are from the same provider
    boolean isFromSameProvider = isSameProvider(location.getProvider(),
            currentBestLocation.getProvider());

    // Determine location quality using a combination of timeliness and
accuracy
    if (isMoreAccurate) {
        return true;
    } else if (isNewer && !isLessAccurate) {
        return true;
    } else if (isNewer && !isSignificantlyLessAccurate &&
isFromSameProvider) {
        return true;
    }
    return false;
}

/** Checks whether two providers are the same */
private boolean isSameProvider(String provider1, String provider2) {
    if (provider1 == null) {
      return provider2 == null;
    }
    return provider1.equals(provider2);
}
```

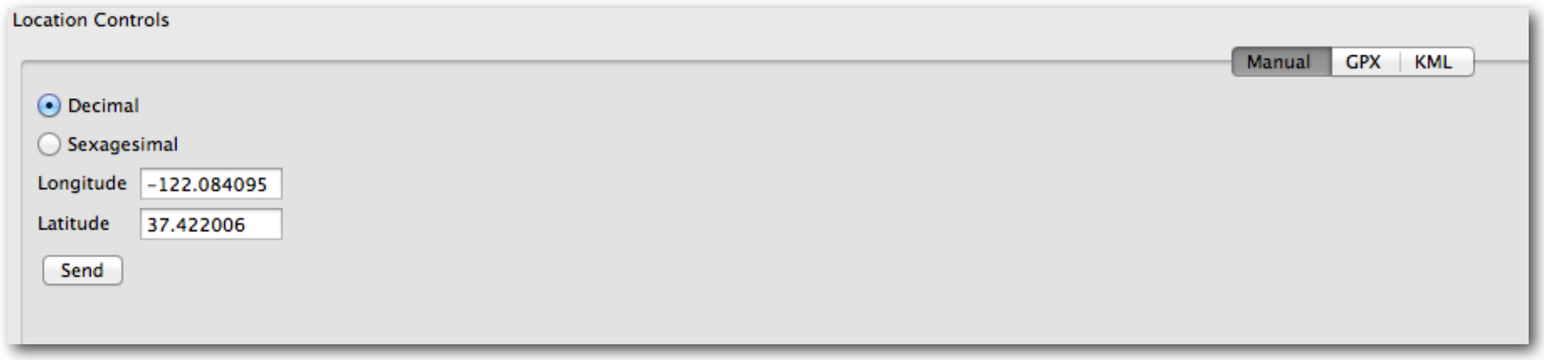
http://developer.android.com/guide/topics/location/obtaining-user-location.html

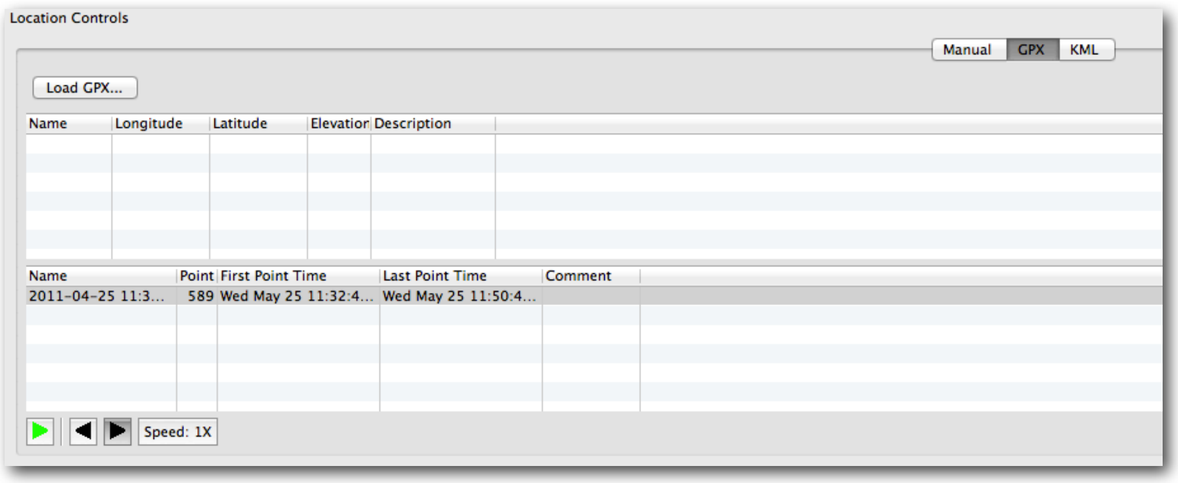
Adjusting the model to save battery and data exchange

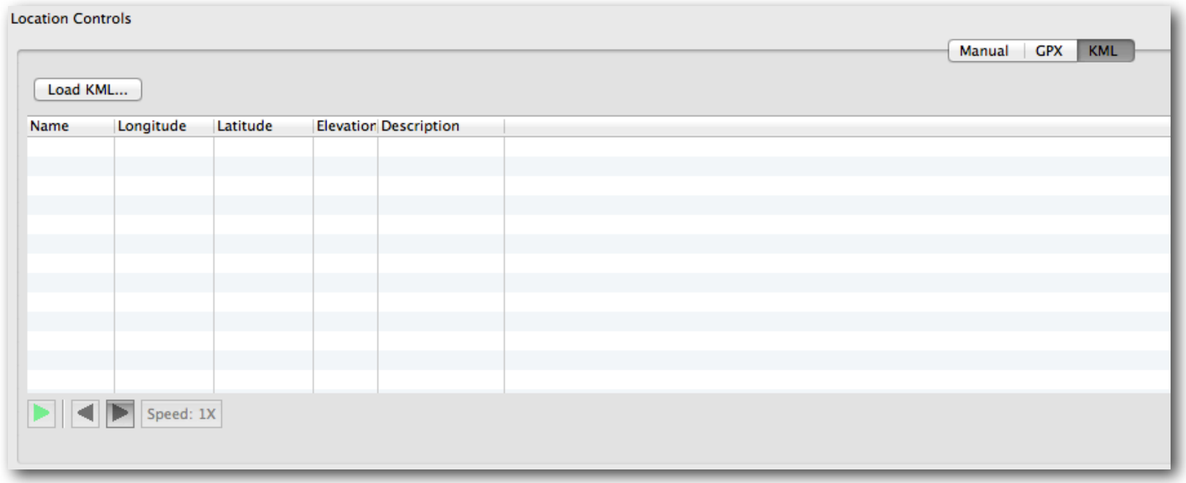
- As you test your application, you might find that your model for providing good location and good performance needs some adjustment. Here are some things you might change to find a good balance between the two.
  - Reduce the size of the window
    - A smaller window in which you listen for location updates means less interaction with GPS and network location services, thus, preserving battery life. But it also allows for fewer locations from which to choose a best estimate.
  - Set the location providers to return updates less frequently
    - Reducing the rate at which new updates appear during the window can also improve battery efficiency, but at the cost of accuracy. The value of the trade-off depends on how your application is used. You can reduce the rate of updates by increasing the parameters in requestLocationUpdates() that specify the interval time and minimum distance change.
  - Restrict a set of providers
    - Depending on the environment where your application is used or the desired level of accuracy, you might choose to use only the Network Location Provider or only GPS, instead of both. Interacting with only one of the services reduces battery usage at a potential cost of accuracy.

Location & AVD

- Eclipse ADT allows you to control the virtual device emulating a change of location. You can use a single location, a GPX or KML file.

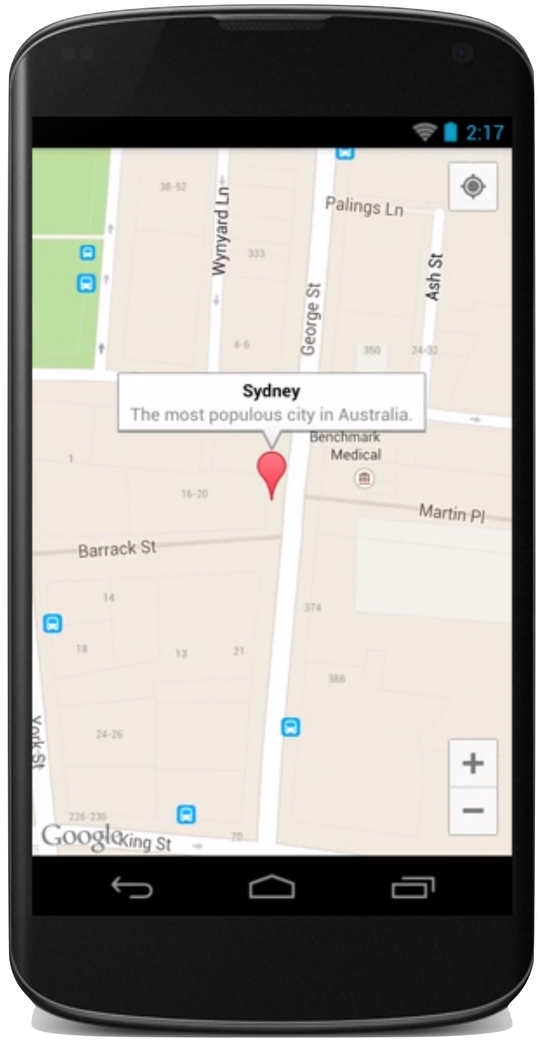






Google Maps API

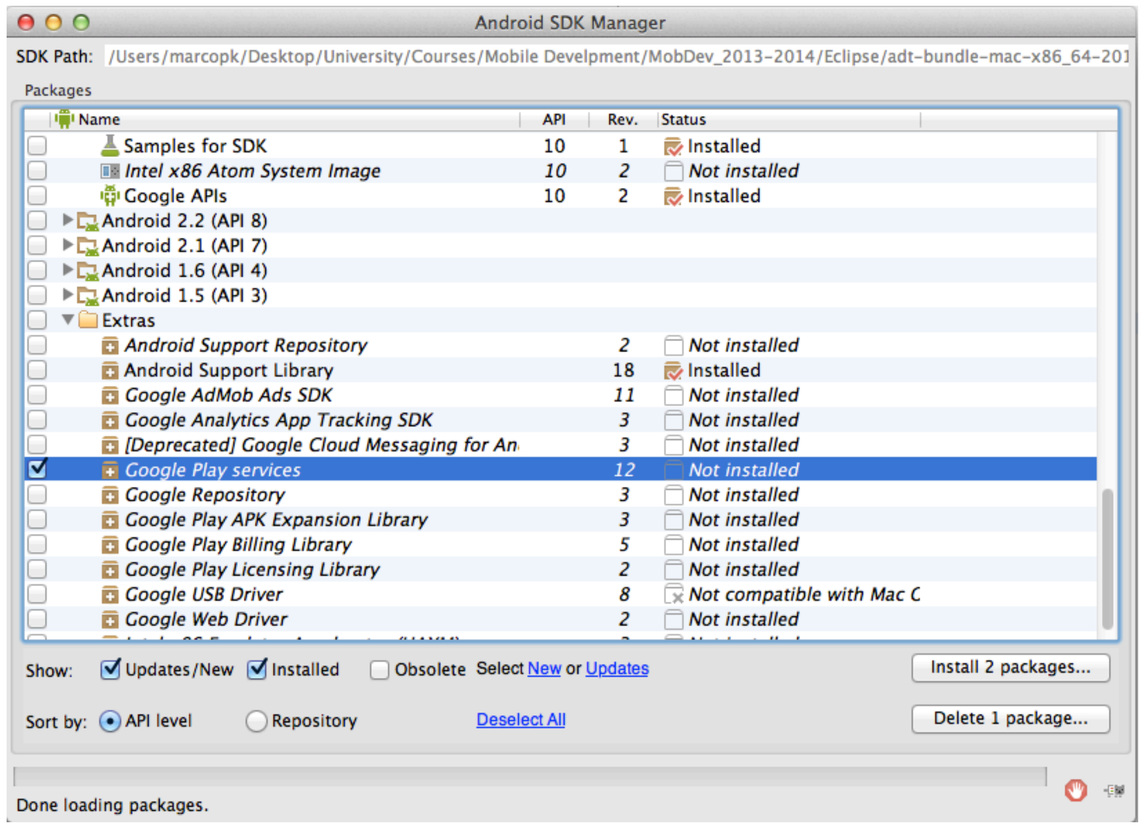
- Google Maps external library allows to add Maps and mapping capabilities to your application.
- The Google Maps APIs add-on includes a Maps external library,com.google.android.maps.
- Provided classes have built-in features such as:
  - download Maps
  - render Maps
  - cache of Maps tiles
  - multiple display options and controls



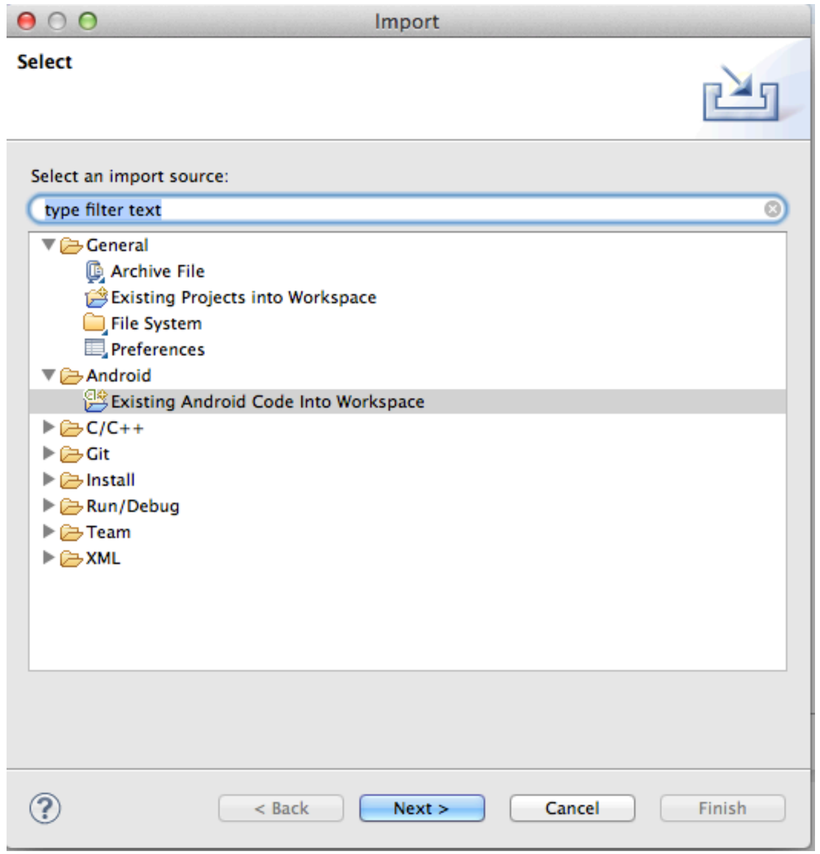
https://developers.google.com/maps/documentation/android/

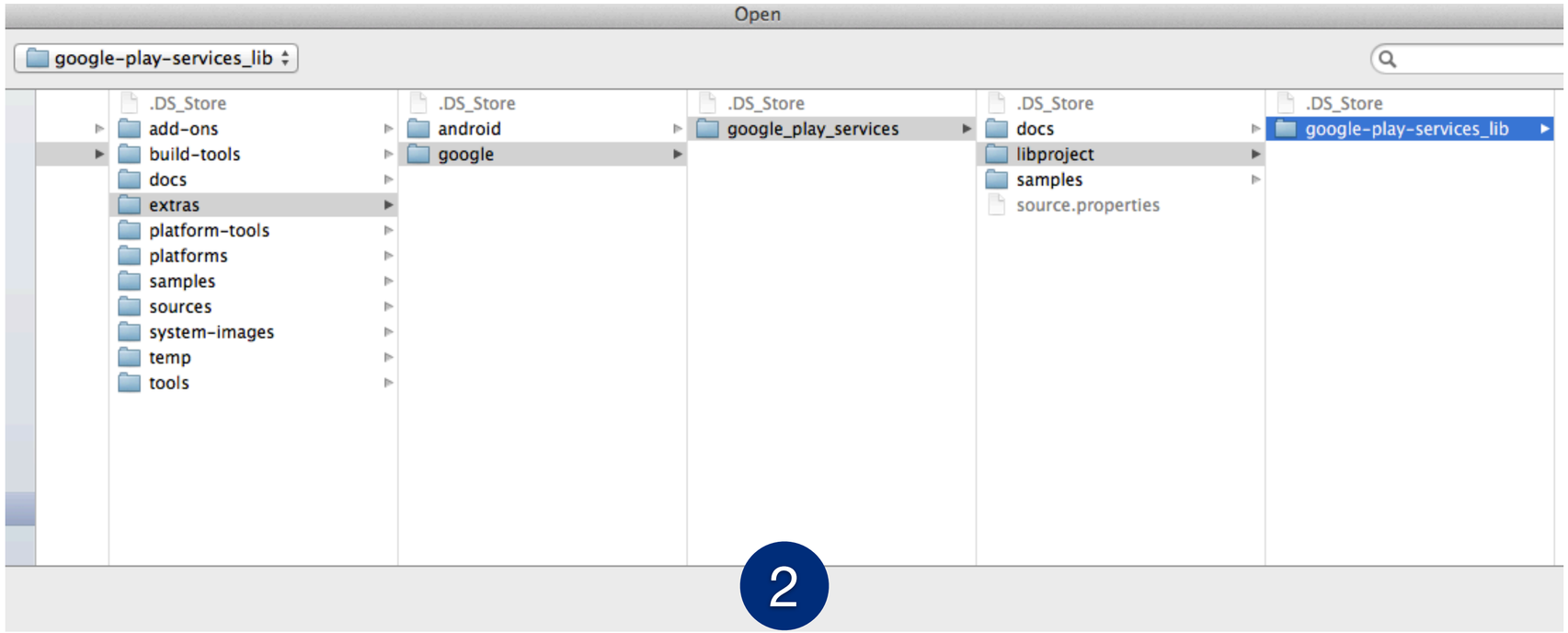
Google Maps Android API v2

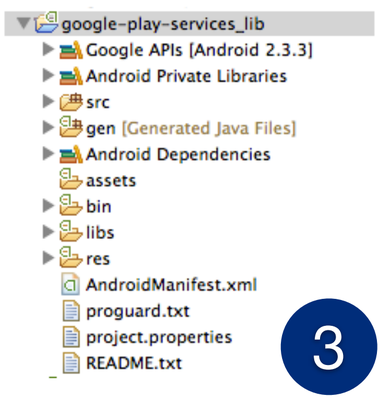
- The creation of a new Android application based on the Google Maps Android API v2 requires the following steps (many of them will only have to be performed once):
  - Download and configure the Google Play services SDK.
  - Obtain an API key.
  - Specify settings in the Application Manifest.
  - Add a map to a new or existing Android project.
  - Publish your application!



Google Play Service Lib. Project







Uncheck the “Copy to workspace” option (in this case)

The Google Maps API Key

- In order to have access to the Google Maps servers with the Maps API, it is necessary to obtain and add a Maps API key to your application.
- The key is free, and can be used with any of your applications that call the Maps API.
- It supports an unlimited number of users.
- To obtain a Maps API key from the Google APIs Console you should provide your application's signing certificate and its package name.
- Once you have the key, you add it to your application by adding an element to your application's manifest file AndroidManifest.xml.
- Maps API keys are linked to specific certificate/package pairs, rather than to users or applications.
- You only need one key for each certificate, no matter how many users you have for an application. Applications that use the same certificate can use the same API key.
- The recommended practice is to sign each of your applications with a different certificate and get a different key for each one.

Certificates

- The Maps API key is based on a short form of your application's digital certificate, called SHA-1 fingerprint. The fingerprint is a unique text string generated from the commonly-used SHA-1 hashing algorithm. Since the fingerprint is itself unique, Google Maps uses it as a way to identify your application.

- **Debug certificate:** The Android SDK tools generate this certificate automatically when you do a "debug" build from the command line, or when you build and run a project from Eclipse without exporting it as a released application. Only use this certificate with apps that you're testing; do not attempt to publish an app that's signed with a debug certificate.

- **Release certificate:** The Android SDK tools generate this certificate when you do a "release" build with either ant program or Eclipse. You can also generate this certificate using the keytool program. Use this certificate when you are ready to release your app to the world.

Displaying the debug certificate fingerprint

The file name is **debug.keystore**, and is created automatically the first time you build your project. By default, it is stored in the same directory as your Android Virtual Device (AVD) files:

OS X and Linux: `~/.android/`
Windows Vista and Windows 7: `C:\Users\your_user_name\.android\`

*If you are using Eclipse with ADT, and you're not sure where your debug keystore is located, you can select **Windows** > **Prefs** > **Android** > **Build** to check the full path, which you can then paste into a file explorer to locate the directory containing the keystore.*

List the SHA-1 fingerprint.

- For Linux or OS X, open a terminal window and enter the following:
```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
```

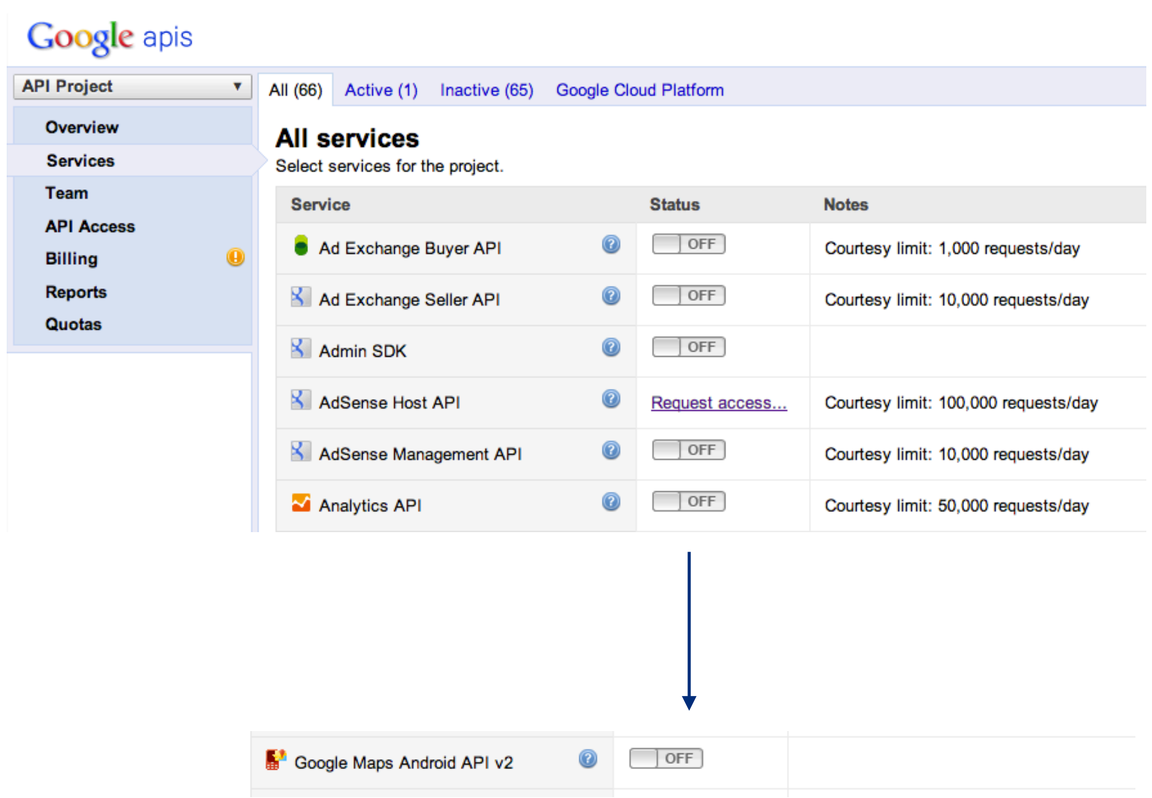
- For Windows Vista and Windows 7, run:
```
keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

Displaying the debug certificate fingerprint

```
Alias name: androiddebugkey
Creation date: Jan 01, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4aa9b300
Valid from: Mon Jan 01 08:04:04 UTC 2013 until: Mon Jan 01 18:04:04 PST 2033
Certificate fingerprints:
     MD5:  AE:9F:95:D0:A6:86:89:BC:A8:70:BA:34:FF:6A:AC:F9
     SHA1: BB:0D:AC:74:D3:21:E1:43:07:71:9B:62:90:AF:A1:66:6E:44:5D:75
     Signature algorithm name: SHA1withRSA
     Version: 3
```

Create an API Project

- Open Google APIs Console WebPage: https://code.google.com/apis/console/?noredirect.
- If you haven't used the Google APIs Console before, you're prompted to create a project that you use to track your usage of the Google Maps Android API. Click Create Project; the Console creates a new project called API Project.
- Open the list of APIs and services in the main window otherwise select Services from the left navigation bar.
- In the list of services displayed in the center of the page, scroll down until you see Google Maps Android API v2. To the right of the entry, click the switch indicator so that it is on.
- This displays the Google Maps Android API Terms of Service. If you agree to the terms of service, click the checkbox below the terms of service, then click Accept. This returns you to the list of APIs and services.



https://developers.google.com/maps/documentation/android/start#the_google_maps_api_key

Obtaining an API Key

- Navigate to your project in the Google APIs Console.
- In the left navigation bar, click API Access.
- In the resulting page, click Create New Android Key....
- In the resulting dialog, enter the SHA-1 fingerprint, then a semicolon, then your application's package name. For example:

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

- The Google APIs Console responds by displaying Key for Android apps (with certificates) followed by a forty-character API key, for example:

```
AIzaSyBdVl-cTICSwYKrZ95SuvNw7dbMuDt1KG0
```

Obtaining an API Key

API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allows you to exceed anonymous limits by connecting requests back to your project.

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. A single project may contain up to 20 client IDs.
Learn more

Create an OAuth 2.0 client ID...

Simple API Access

Use API keys to identify your project when you do not need to access user data. Learn more

Key for Android apps (with certificates)

API key: [redacted]
Android apps: [redacted]
Activated on: Oct 20, 2013 3:43 AM
Activated by: [redacted]

Generate new key...
Edit allowed Android apps...
Delete key...

Key for browser apps (with referers)

API key: [redacted]
Referers: Any referer allowed
Activated on: May 4, 2013 7:40 AM
Activated by: [redacted]

Generate new key...
Edit allowed referers...
Delete key...

Create new Server key... | Create new Browser key... | Create new Android key... | Create new iOS key...

Notification Endpoints

Use notification endpoints to identify domains that may receive webhook notifications from your API. Learn more

Allowed Domains: No domains allowed

Edit...

Adding the API Key to your App

- In AndroidManifest.xml, add the following element as a child of the <application> element, by inserting it just before the closing tag</application>:

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

- substituting your API key for API_KEY. This element sets the key com.google.android.maps.v2.API_KEY to the value API_KEY and makes the API key visible to any MapFragment in your application.

- Save AndroidManifest.xml and re-build your application.

Application Manifest Settings

- An Android application that uses the Google Maps Android API should specify the following settings in its manifest file, AndroidManifest.xml:
  - **Permissions** that give the application access to Android system features and to the Google Maps servers.
  - (**Recommended**) Notification that the application requires OpenGL ES version
    - The Google Maps Android API uses OpenGL ES version 2 to render the map. If OpenGL ES version 2 is not installed, your map will not appear. The recommendation is to add the following <uses-feature> element as a child of the <manifest> element in AndroidManifest.xml:

```
<uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
```

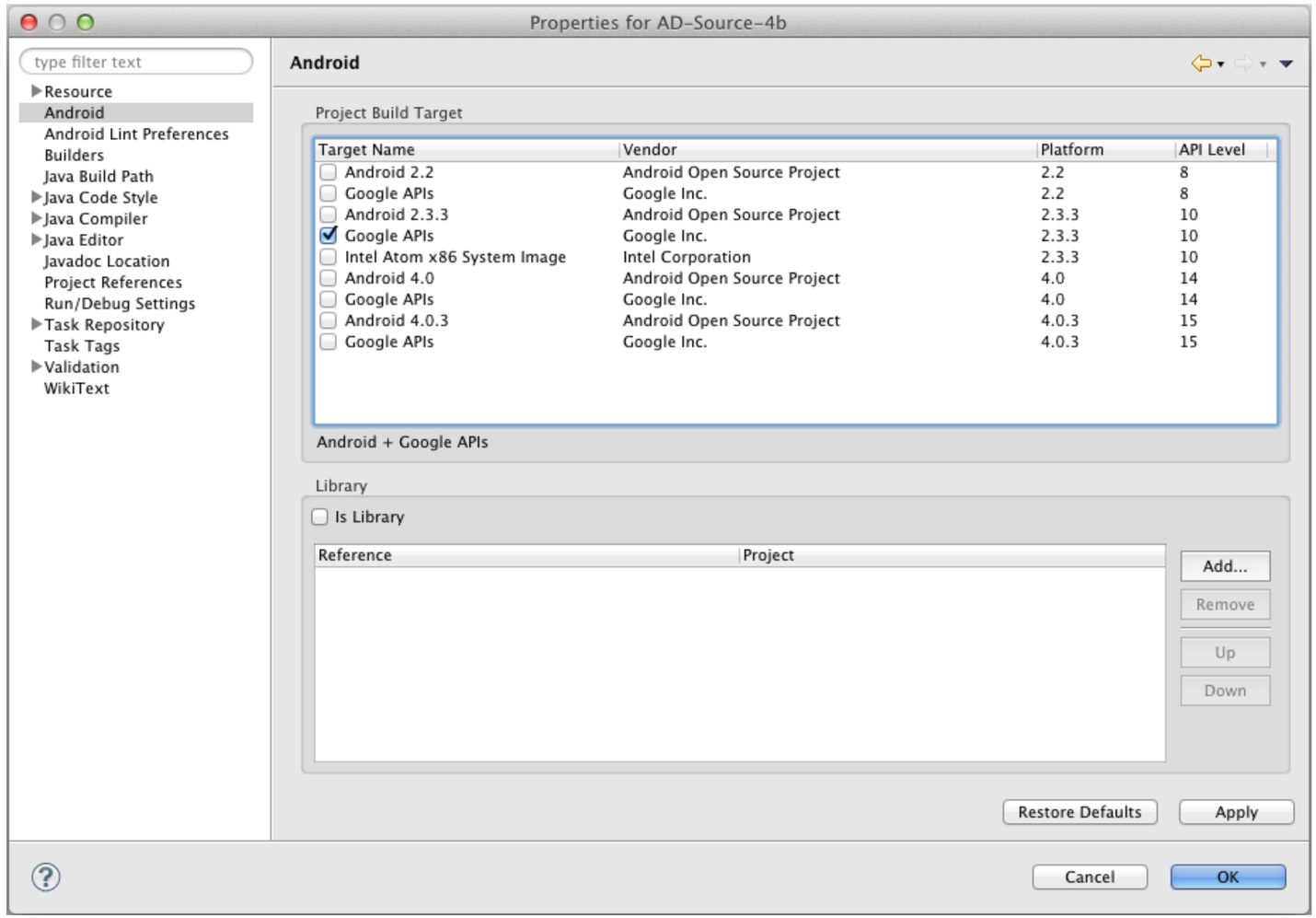
  - The Maps **API key** for the application.

Permissions

```
<!-- Used by the API to download map tiles from Google Maps servers. -->
<uses-permission android:name="android.permission.INTERNET"/>

<!-- Allows the API to check the connection status in order to determine whether data can be downloaded. -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

<!-- Allows the API to access Google web-based services. -->
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>

<!-- Allows the API to cache map tile data in the device's external storage area. -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

<!-- Allows the API to use WiFi or mobile cell data (or both) to determine the device's location. -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

<!-- Allows the API to use the GPS to determine the device's location to within a very small area. -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Add a Map to your project ! (Finally :))

- In order to use the Google Maps Library and correctly compile your application you should use as target of your project the Google APIs Target Name corresponding to the right API level.



Add a Map to your project ! (Finally :))

- Using the latest version of the Google Maps for Android the MapFragment is the simplest way to place a map in an application.
- A MapFragment is a wrapper around a view of a map to automatically handle the necessary life cycle needs.
- Being a fragment, this component can be added to an activity's layout file simply with the XML below.

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:name="com.google.android.gms.maps.MapFragment"/>
```

Add a Map to your project ! (Finally :))

A GoogleMap object can be acquired only using the getMap() method when the underlying maps system is loaded and the underlying view in the fragment exists.

This class automatically initializes the maps system and the view;

If a GoogleMap is not available, getMap() will return null.

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

      // Get a handle to the Map Fragment
      GoogleMap map = ((MapFragment)getFragmentManager().findFragmentById(R.id.map)).getMap();

   }
```

Maps V2 & Compatibility

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:name="com.google.android.gms.maps.MapFragment"/>
```

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:name="com.google.android.gms.maps.SupportMapFragment"/>
```

Maps V2 & Compatibility

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

      // Get a handle to the Map Fragment
      GoogleMap map = ((MapFragment)getFragmentManager().findFragmentById(R.id.map)).getMap();
   }
```

```
[...]
import com.google.android.gms.maps.SupportMapFragment;
import android.support.v4.app.FragmentActivity;

public class MainActivity extends FragmentActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Get a handle to the Map Fragment
        GoogleMap map = ((SupportMapFragment)getSupportFragmentManager().findFragmentById(R.id.map)).getMap();
    }
}
```

Main Map Methods and Classes

- **LatLng Class:** A class representing a pair of latitude and longitude coordinates, stored as degrees.
- **setMyLocationEnabled(boolean enabled):** Enables or disables the my-location layer. While enabled, the my-location layer continuously draws an indication of a user's current location and bearing, and displays UI controls that allow a user to interact with their location.
- **moveCamera(CameraUpdate update)**: Repositions the camera according to the instructions defined in the update. The move is instantaneous, and a subsequent getCameraPosition() will reflect the new position. See CameraUpdateFactory for a set of updates.
- **setMapType(int type)**: Sets the type of map tiles that should be displayed (e.g. GoogleMap.MAP_TYPE_SATELLITE). The allowable values are:
  - MAP_TYPE_NORMAL: Basic map with roads.
  - MAP_TYPE_SATELLITE: Satellite view with roads.
  - MAP_TYPE_TERRAIN: Terrain view without roads.
- **public final void setOnMapClickListener (GoogleMap.OnMapClickListener listener)**: Sets a callback that's invoked when the map is tapped.
- **public final void setOnMapLongClickListener (GoogleMap.OnMapLongClickListener listener)**: Sets a callback that's invoked when a long click has been performed on the map.

Main Map Methods

```
// Get a handle to the Map Fragment
GoogleMap map = ((SupportMapFragment)getSupportFragmentManager().findFragmentById(R.id.map)).getMap();

//Create a geographic point starting from latitude and longitude coordinate
LatLng parmaUniversity = new LatLng(44.76516282282244,10.311720371246338);
//Enable or disable user location layer
map.setMyLocationEnabled(true);
//Move the map to a specific point
map.moveCamera(CameraUpdateFactory.newLatLngZoom(parmaUniversity, 13));
//Create and Assign the listener for onClick event on the Map
map.setOnMapClickListener(new OnMapClickListener() {
   //Listener method to receive the LatLng object associated to the clicked point on the map
   @Override
   public void onMapClick(LatLng point) {
         Log.d(MainActivity.TAG,"MainActivity ---> onMapClick: " + point.latitude + ";" + point.longitude);
      }
});

//Create and Assign the listener for onLongClick event on the Map
map.setOnMapLongClickListener(new OnMapLongClickListener() {
   @Override
   public void onMapLongClick(LatLng point) {
      Log.d(MainActivity.TAG,"MainActivity ---> onMapLongClick: " + point.latitude + ";" + point.longitude);
   }
});
```

Marker

- Markers indicate single locations on the map.
- Markers can be customized by changing
  - the default color
  - replacing the marker icon with a custom image
  - providing an Info windows with additional context to a marker.

```
//Create a new marker
MarkerOptions myMarkerOptions = new MarkerOptions();
myMarkerOptions.title("Unipr - 1");
myMarkerOptions.snippet("Universita' degli Studi di Parma - Facolta' di Ingegneria Sede Didattica");
myMarkerOptions.position(parmaUniversity);

//Add the marker to the Map
Marker marker = map.addMarker(myMarker);
```

Marker Customization

Marker Color

```
MarkerOptions myMarkerOptions = new MarkerOptions();
[...]
myMarkerOptions.position(parmaUniversity);
myMarkerOptions.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE));
```

Marker Icon

It is also possible to replace the default marker image with a custom marker image. Custom icons are always set as a BitmapDescriptor, and defined using one of four methods in the BitmapDescriptorFactory class.

fromAsset(String assetName): Creates a custom marker using an image in the assets directory.

fromBitmap (Bitmap image): Creates a custom marker from a Bitmap image.

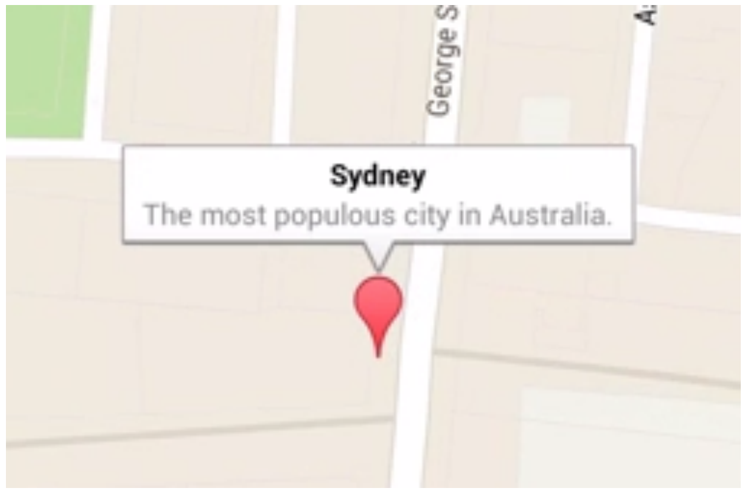
fromFile (String path): Creates a custom icon from a file at the specified path.

fromResource (int resourceId): Creates a custom marker using an existing resource.

```
MarkerOptions myMarkerOptions = new MarkerOptions();
[...]
myMarkerOptions.position(parmaUniversity);
myMarkerOptions.icon(BitmapDescriptorFactory.fromResource(R.drawable.ic_location_place));
```

Marker Events

- The Maps API allows to listen and respond to markers events.
- To listen to these events, it is necessary to set the right associated listener on the GoogleMap object to which the markers belong.
- When the event occurs on one of the markers on the map, the listener's callback will be invoked with the corresponding Marker object passed through as a parameter.
- To compare this Marker object with your own reference to a Marker object, you must use equals() and not ==.
- You can listen to the following events:
  - Marker click events
  - Marker drag events
  - Info window click events



http://developer.android.com/reference/android/location/Geocoder.html

Marker Events

```
//Create and Assign the listener for available markers
map.setOnMarkerClickListener(new OnMarkerClickListener() {

   @Override
   public boolean onMarkerClick(Marker marker) {

      if(myMarker != null && marker.equals(myMarker))
         Log.d(MainActivity.TAG,"MainActivity ---> onInfoWindowClick of MyMarker");

      //True if we want that the event has been consumed or false to propagate it
      return false;
   }
});

//Create and Assign the listener for InfoWindows of existing markers
map.setOnInfoWindowClickListener(new OnInfoWindowClickListener() {

   @Override
   public void onInfoWindowClick(Marker marker) {
      if(myMarker != null && marker.equals(myMarker))
         Log.d(MainActivity.TAG,"MainActivity ---> onInfoWindowClick of MyMarker");
   }
});
```

GeoCoding Locations

- The Android SDK provides some helper methods to :
  - turn raw location data into addresses and descriptive place names.
  - using named locations or address lines to generate latitude and longitude information.
- The provided object is called Geocoder. It can be used without any special permissions and according to the official API has 4 different methods:
  - **getFromLocation(double latitude, double longitude, int maxResults):** Returns an array of Addresses that are known to describe the area immediately surrounding the given latitude and longitude.
  - **getFromLocationName(String locationName, int maxResults, double lowerLeftLatitude, double lowerLeftLongitude, double upperRightLatitude, double upperRightLongitude):** Returns an array of Addresses that are known to describe the named location, which may be a place name such as "Dalvik, Iceland", an address such as "1600 Amphitheatre Parkway, Mountain View, CA", an airport code such as "SFO", etc..
  - **getFromLocationName(String locationName, int maxResults):** Returns an array of Addresses that are known to describe the named location, which may be a place name such as "Dalvik, Iceland", an address such as "1600 Amphitheatre Parkway, Mountain View, CA", an airport code such as "SFO", etc..
  - **isPresent():** Returns true if the Geocoder methods getFromLocation and getFromLocationName are implemented.

http://developer.android.com/reference/android/location/Geocoder.html

GeoCoding Locations

```
Geocoder coder = new Geocoder(this);
try {

   Iterator<Address> addresses = coder.getFromLocation(44.764531,10.312128,10).iterator();

   if(addresses != null)
   {
      while (addresses.hasNext()) {
         Address loc = addresses.next();
         String placeName = loc.getLocality();
         String featerName = loc.getFeatureName();
         String conString = loc.getCountryName();
         String road = loc.getThoroughfare();

         ...
      }
   }

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


http://developer.android.com/reference/android/location/Geocoder.html

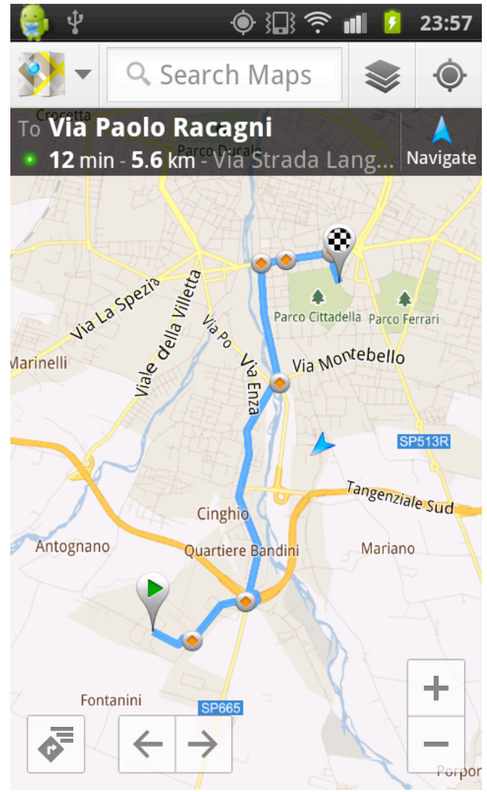
GeoCoding Locations

```
Geocoder coder = new Geocoder(this);
try {

   Iterator<Address> addresses = coder.getFromLocationName("Parma",10).iterator();

   if(addresses != null)
   {
      while (addresses.hasNext()) {
         Address loc = addresses.next();
         double lat = loc.getLatitude();
         double lng = loc.getLongitude();

         ...
      }
   }

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


http://developer.android.com/reference/android/location/Geocoder.html

Call Maps Application or Navigator

- Using Intents is possible to call from an application the maps application or if available the built-in navigator.
- The intent requires an url structured in the Google Maps API format specifing:
  - starting and destination addresses
  - starting and destination geographic location (lat,lng)

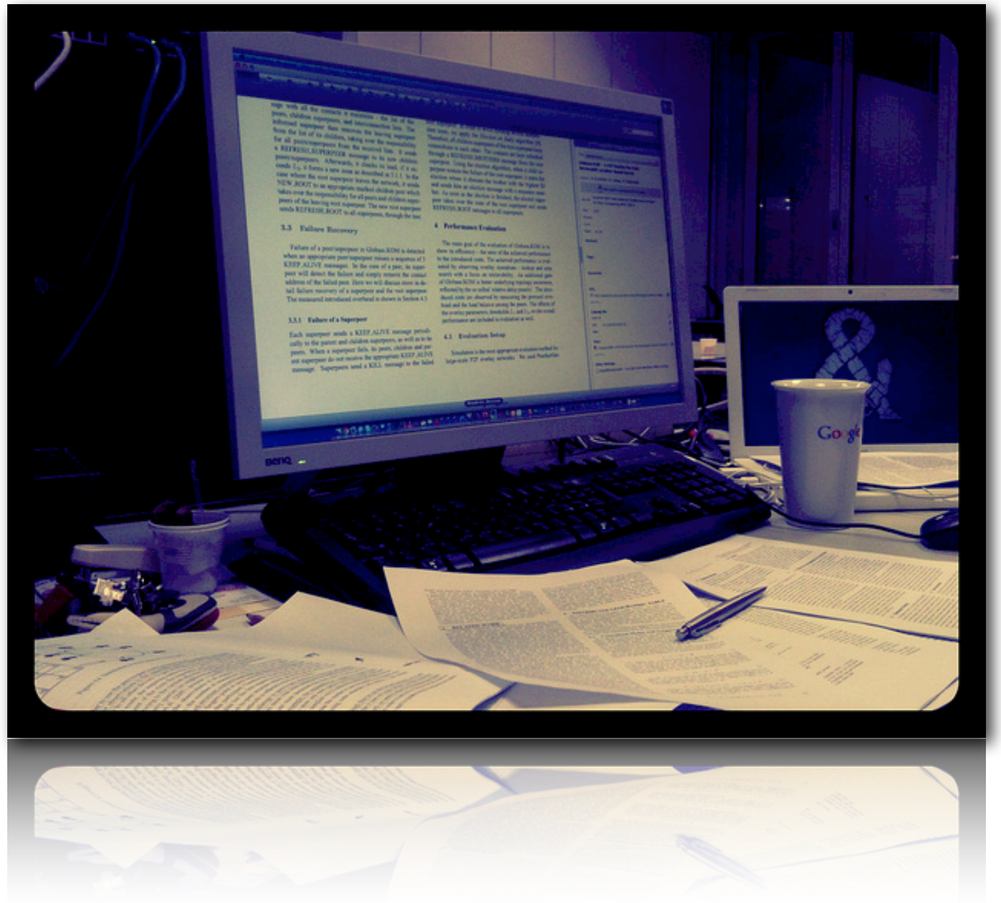



```
String url = "http://maps.google.com/maps?saddr="+startAddress+"&daddr="+destAddress;
Intent intent = new Intent(android.content.Intent.ACTION_VIEW,  Uri.parse(url));
startActivity(intent);
```

```
String url = "http://maps.google.com/maps?saddr=44.764531,10.312128&daddr=44.792343,10.331526";
Intent intent = new Intent(android.content.Intent.ACTION_VIEW,  Uri.parse(url));
startActivity(intent);
```

Coming Up

- Next Lecture
  - Data Persistence
- Homework
  - Review Location & Mapping Application
  - Add a Menu to dynamically change map types
  - Create an interface to search a location and show it on the map starting from an address.



Android Development

Lecture 6
Location and Maps