

Ubiquitous Computing

COMP H4025

Lecturer: Simon McLoughlin

Lecture 4



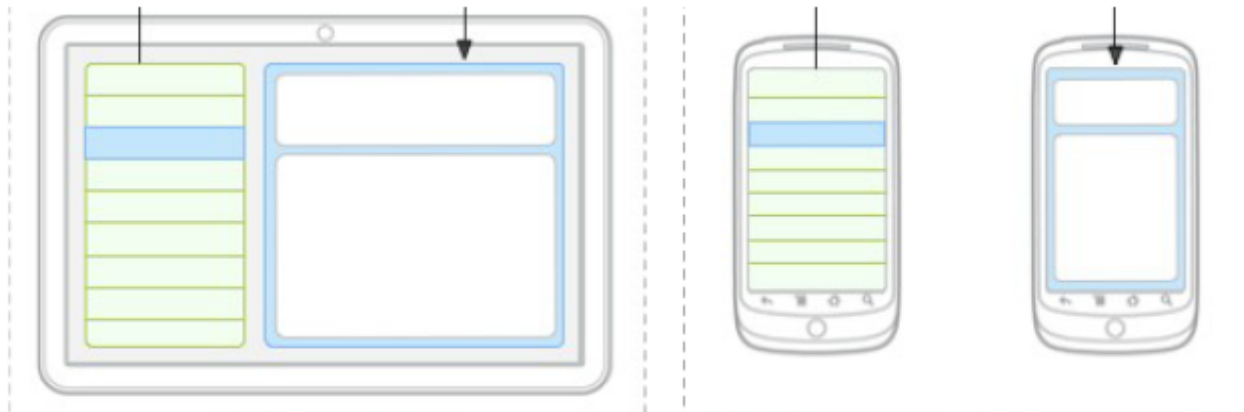
Lecture Overview

This week:

- Fragments
- Locations Based Services and Maps

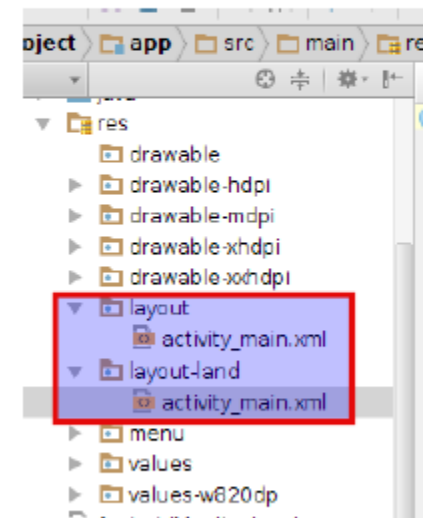
Layouts for different situations

- Your app can use different layout in different situations:
 - different device type (tablet vs phone vs watch)
 - different screen size
 - different orientation (portrait vs. landscape)
 - different country or locale (language, etc.)



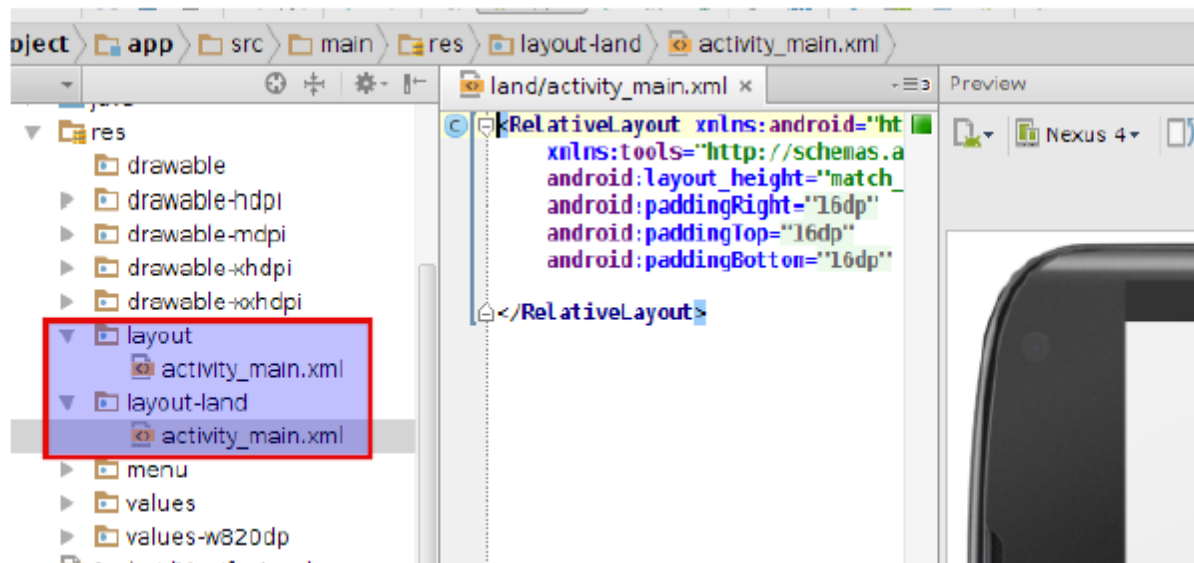
Layouts for different situations

- Your app will look for resource folder names with suffixes:
 - screen density (e.g. **drawable-hdpi**) ([link](#))
 - xhdpi: 2.0 (twice as many pixels/dots per inch)
 - hdpi: 1.5
 - mdpi: 1.0 (baseline)
 - ldpi: 0.75
 - screen size (e.g. **layout-large**) ([link](#))
 - small, normal, large, xlarge
 - orientation (e.g. **layout-land**)
 - portrait (), land (landscape)



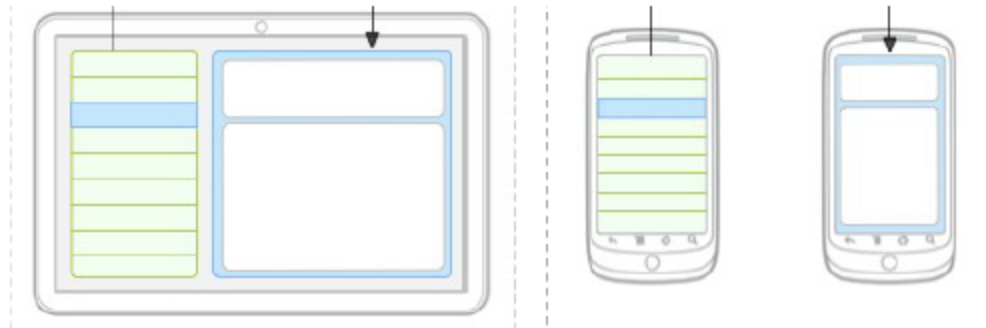
Portrait vs Landscape Layouts

- To create a different layout in landscape mode:
 - create a folder in your project called **res/layout-land**
 - place another copy of your activity's **layout XML file** there
 - modify it as needed to represent the differences



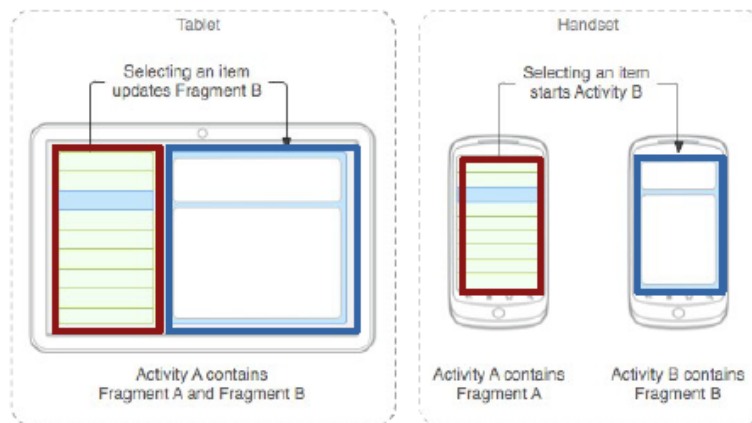
Layouts for different situations

- You sometimes want your code to behave **situationally**.
 - In portrait mode, clicking a button should launch a new **activity**.
 - In landscape mode, clicking a button should launch a new **view**.



Fragments

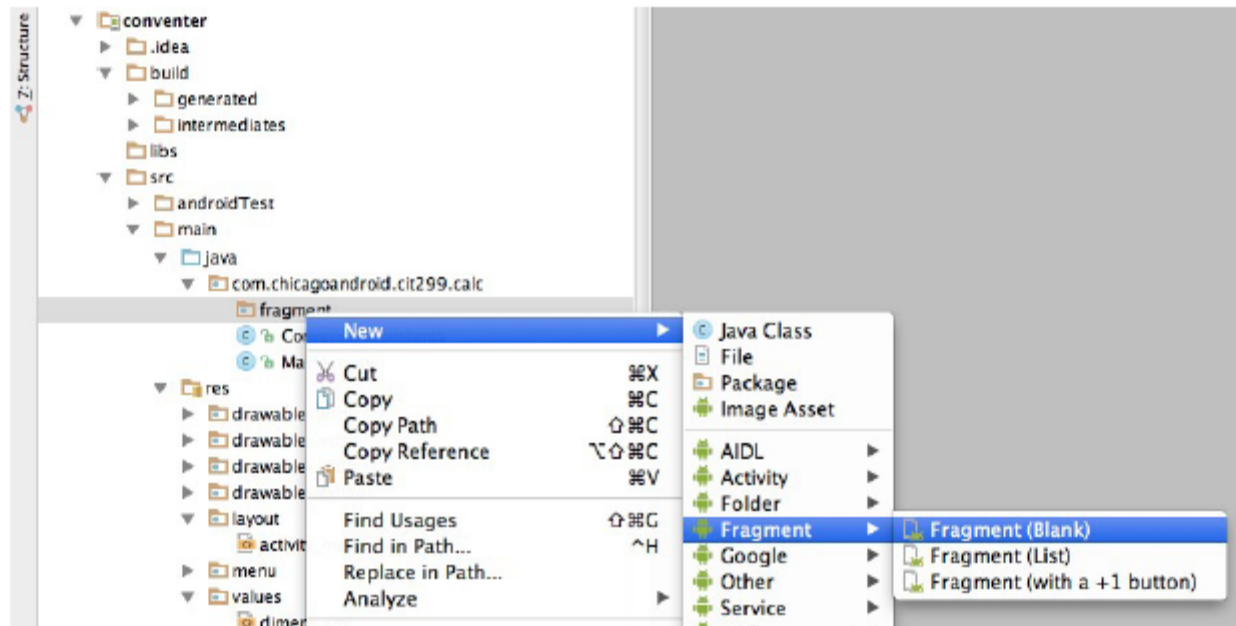
- **fragment:** A reusable segment of Android UI that can appear in an activity.
 - can help handle different devices and screen sizes
 - can reuse a common fragment across multiple activities
 - first added in Android 3.0 (*usable in older versions if necessary*)



- Fragments exist primarily to support more dynamic and flexible UI designs on large screens, such as tablets. Because a tablet's screen is much larger than that of a handset, there's more room to combine and interchange UI components.
- By dividing the layout of an activity into fragments, you become able to modify the activity's appearance at runtime.

Creating a Fragment

- In Android Studio, right-click app, click:
New → Fragment → Fragment (blank)
 - un-check boxes about "Include __ methods"
 - now create layout XML and Java event code as in an Activity



Using Fragments in Activity layout

- Activity layout XML can include fragments.



```
<!-- activity_name.xml -->
<LinearLayout ...>
    <fragment ...
        android:id="@+id/id1"
        android:name="ClassName1"
        tools:layout="@layout/name1" />
    <fragment ...
        android:id="@+id/id2"
        android:name="ClassName2"
        tools:layout="@layout/name2" />
</LinearLayout>
```

Fragment lifecycle

- Fragments have a similar **life cycle** and events as activities.
- Important methods:
 - **onAttach** to glue fragment to its surrounding activity
 - **onCreate** when fragment is loading
 - **onCreateView** method that must return fragment's root UI view
 - **onActivityCreated** method that indicates the enclosing activity is ready
 - **onPause** when fragment is being left/exited
 - **onDetach** just as fragment is being deleted



Fragment template

```
public class Name extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
        ViewGroup vg, Bundle bundle) {  
        // load the GUI layout from the XML  
        return inflater.inflate(R.layout.id, vg, false);  
    }  
  
    public void onActivityCreated(Bundle savedInstanceState) {  
        super.onActivityCreated(savedInstanceState);  
        // ... any other GUI initialization needed  
    }  
  
    // any other code (e.g. event-handling)  
}
```

Fragment vs Activity

- Fragment code is similar to activity code, with a few changes:
 - Many activity methods aren't present in the fragment, but you can call **getActivity** to access the activity the fragment is inside of.

```
Button b = (Button) findViewById(R.id.but);  
Button b = (Button) getActivity().findViewById(R.id.but);
```
 - Sometimes also use `getView` to refer to the activity's layout
 - Event handlers cannot be attached in the XML any more. :-(
 - Must be attached in Java code instead.
 - Passing information to a fragment (via Intents/Bundles) is trickier.
 - The fragment must ask its enclosing activity for the information.
 - Fragment initialization code must be mindful of order of execution.
 - Does it depend on the surrounding activity being loaded? Etc.
 - Typically move `onCreate` code to `onActivityCreated`.

Fragment onClickListener

- Activity:

```
<Button android:id="@+id/b1"  
        android:onClick="onClickB1" ... />
```

- Fragment:

```
<Button android:id="@+id/b1" ... />
```

```
// in fragment's Java file
```

```
Button b = (Button) getActivity().findViewById(r.id.b1);  
b.setOnClickListener(new View.OnClickListener() {  
    @Override public void onClick(View view) {  
        // whatever code would have been in onClickB1  
    }  
});
```

Activity with Intent Data

```
public class Name extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.name);

        // extract parameters passed to activity from intent
        Intent intent = getIntent();
        int name1 = intent.getIntExtra("id1", default);
        String name2 = intent.getStringExtra("id2", "default");

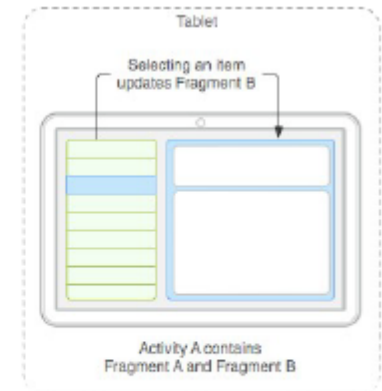
        // use parameters to set up the initial state
        ...
    }
    ...
}
```

Fragment with Intent Data

```
public class Name extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater,  
        ViewGroup container, Bundle savedInstanceState) {  
        return inflater.inflate(R.layout.name, container, false);  
    }  
  
    @Override  
    public void onActivityCreated(Bundle savedInstanceState) {  
        super.onActivityCreated(savedInstanceState);  
  
        // extract parameters passed to activity from intent  
        Intent intent = getActivity().getIntent();  
        int name1 = intent.getIntExtra("id1", default);  
        String name2 = intent.getStringExtra("id2", "default");  
  
        // use parameters to set up the initial state  
        ...  
    }  
}
```

Communication between Fragments

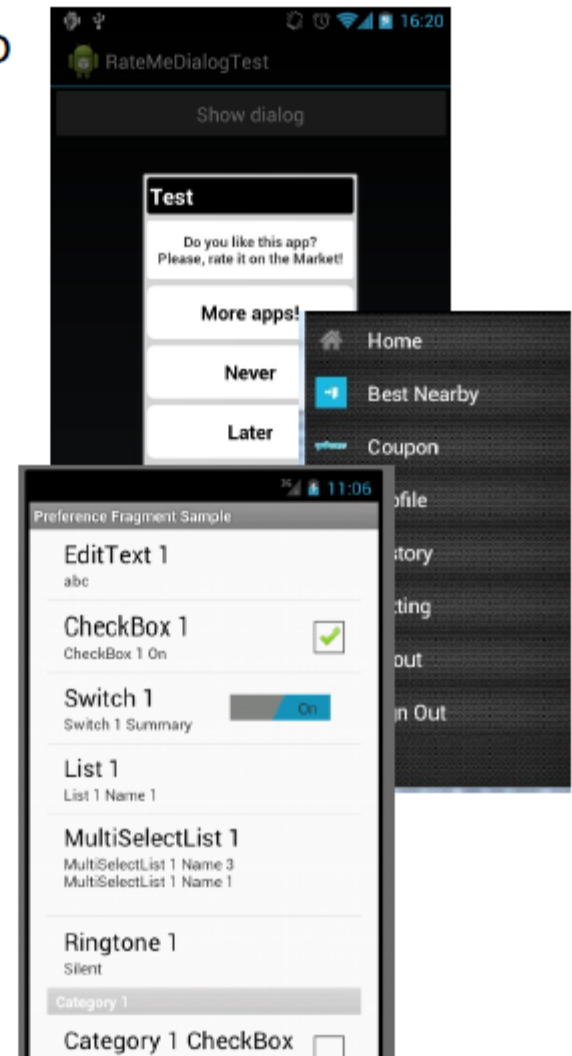
- One activity might contain multiple fragments.
- The fragments may want to talk to each other.
 - Use activity's `getFragmentManager` method.
 - its `findFragmentById` method can access any fragment that has an id.



```
Activity act = getActivity();
if (act.getResources().getConfiguration().orientation ==
    Configuration.ORIENTATION_LANDSCAPE) {
    // update other fragment within this same activity
    FragmentClass fragment = (FragmentClass)
        act.getFragmentManager().findFragmentById(R.id.id);
    fragment.methodName(parameters);
}
```


Fragment Subclasses

- `DialogFragment` - a fragment meant to be shown as a dialog box that pops up on top of the current activity.
- `ListFragment` - a fragment that shows a list of items as its main content.
- `PreferenceFragment` - a fragment whose main content is meant to allow the user to change settings for the app.



StatusFragment

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.marakana.android.yamba.clientlib.YambaClient;
import com.marakana.android.yamba.clientlib.YambaClientException;

public class StatusFragment extends Fragment implements OnClickListener {
    private static final String TAG = "StatusFragment";
    private EditText editStatus;
    private Button buttonTweet;
    private TextView textCount;
    private int defaultTextColor;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {

        View view = inflater
            .inflate(R.layout.fragment_status, container, false);

        editStatus = (EditText) view.findViewById(R.id.editStatus);
        buttonTweet = (Button) view.findViewById(R.id.buttonTweet);
        textCount = (TextView) view.findViewById(R.id.textCount);

        buttonTweet.setOnClickListener(this);

        defaultTextColor = textCount.getTextColors().getDefaultColor();
        editStatus.addTextChangedListener(new TextWatcher() {
```

StatusFragment

```
defaultTextColor = textCount.getTextColors().getDefaultColor();
editStatus.addTextChangedListener(new TextWatcher() {

    @Override
    public void afterTextChanged(Editable s) {
        int count = 140 - editStatus.length();
        textCount.setText(Integer.toString(count));
        textCount.setTextColor(Color.GREEN);
        if (count < 10)
            textCount.setTextColor(Color.RED);
        else
            textCount.setTextColor(defaultTextColor);
    }

    @Override
    public void beforeTextChanged(CharSequence s,
                                   int start, int count,
                                   int after) {

    }

    @Override
    public void onTextChanged(CharSequence s,
                               int start, int before,
                               int count) {

    }

});

return view;
}

@Override
public void onClick(View view) {
    String status = editStatus.getText().toString();
    Log.d(TAG, "onClicked with status: " + status);

    new PostTask().execute(status);
}
```

StatusFragment

```
private final class PostTask extends AsyncTask<String, Void, String> {

    @Override
    protected String doInBackground(String... params) {
        YambaClient yambaCloud =
            new YambaClient("student", "password");
        try {
            yambaCloud.postStatus(params[0]);
            return "Successfully posted";
        } catch (YambaClientException e) {
            e.printStackTrace();
            return "Failed to post to yamba service";
        }
    }

    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        Toast.makeText(StatusFragment.this.getActivity(),
            result, Toast.LENGTH_LONG).show();
    }
}
```

New StatusActivity

```
package com.marakana.android.yamba;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

public class StatusActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.status, menu);
        return true;
    }
}

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <fragment
        android:id="@+id/fragment_status"
        android:name="com.marakana.android.yamba.StatusFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</FrameLayout>
```

Dynamically Creating Fragments

- You can also dynamically create the Fragment in Java code. We do this by creating a new instance of the StatusFragment class, obtain the fragment manager from the current context, start a transaction, attach this new fragment to the root of this activity identified by the system ID android.R.id.content, and commit this transaction.

```
public class StatusActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Check if this activity was created before  
        if (savedInstanceState == null) { // ❶  
            // Create a fragment  
            StatusFragment fragment = new StatusFragment(); // ❷  
            getSupportFragmentManager()  
                .beginTransaction()  
                .add(android.R.id.content, fragment,  
                    fragment.getClass().getSimpleName())  
                .commit(); // ❸  
        }  
    }  
}
```

Dynamically Creating Fragments

- You can also dynamically create the Fragment in Java code. We do this by creating a new instance of the StatusFragment class, obtain the fragment manager from the current context, start a transaction, attach this new fragment to the root of this activity identified by the system ID android.R.id.content, and commit this transaction.

```
public class StatusActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        // Check if this activity was created before  
        if (savedInstanceState == null) { // ❶  
            // Create a fragment  
            StatusFragment fragment = new StatusFragment(); // ❷  
            getSupportFragmentManager()  
                .beginTransaction()  
                .add(android.R.id.content, fragment,  
                    fragment.getClass().getSimpleName())  
                .commit(); // ❸  
        }  
    }  
}
```

Location Based Services

- When developing a location-aware application for Android, you can utilize **GPS** and Android's **Network Location Provider** to acquire the user location.
- Although 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.
- We will look at the simplest approach for accessing location data first, using **the android.location API**. The newer Google Play Services APIs is the more up to date machine for this purpose but there is more involved (later)

Challenges in Determining User Location

- 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. These problems can make it difficult to obtain a reliable user location reading.

Requesting User Permissions

- In order to receive location update from NETWORK_PROVIDER or GPS_PROVIDER, you must request user permission by declaring either the ACCESS_COARSE_LOCATION or ACCESS_FINE_LOCATION permission, respectively, in your Android manifest file.

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

- 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 for NETWORK_PROVIDER.)

Requesting Location Updates

- Getting user location in Android works by means of **callbacks**. You indicate that you'd like to receive location updates from the [LocationManager](#) by calling [requestLocationUpdates\(\)](#) and 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. For example, the following code shows how to define a [LocationListener](#) and request location updates:

```
// 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.
        makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {}

    public void onProviderEnabled(String provider) {}

    public void onProviderDisabled(String provider) {}
};

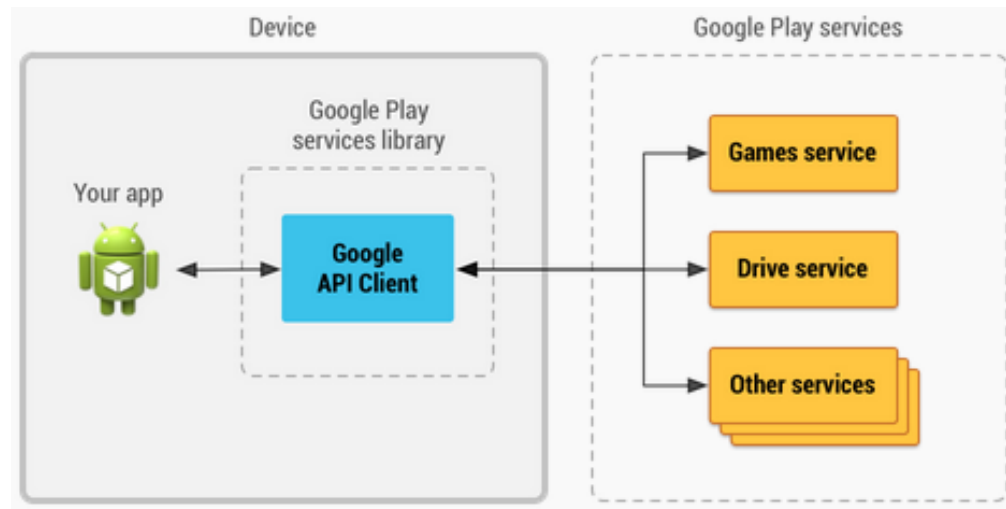
// Register the listener with the Location Manager to receive location updates
locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 0, 0, locationListener);
```

Extracting the raw location data

- Extracting the raw location data is then very straightforward. When the location provider finds a new location the `onLocationChanged` callback is called and a `Location` object is passed.
- This location object has methods to retrieve the latitude, longitude, height etc. these are `loc.getLatitude()` etc. which returns double variable representing the location values in degrees.
- You can also extract the users location in a synchronous manner (without the callbacks) by calling the method [`getLastKnownLocation`](#) from the `LocationManager` class.
- `Location loc = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);`

Google Play Services

- Google Play Services is a set of high level libraries that you can use in your android app (will only support the more recent android devices) to access features for areas such as, gaming, maps, Google+, location
- To use Google play services you must add the SDK and configure your project to use it, see <https://developer.android.com/google/play-services/setup.html> for more information.
- To connect to a Google Play Service you use a GoogleApiClient object or one of its subclasses, we will see an example of this when we access the Location Service.



Google Location Service API

- *Simple APIs*: Lets you specify high-level needs like "high accuracy" or "low power", instead of having to worry about location providers.
- *Immediately available*: Gives your apps immediate access to the best, most recent location.
- *Power-efficiency*: Minimizes your app's use of power. Based on all incoming location requests and available sensors, fused location provider chooses the most efficient way to meet those needs.
- *Versatility*: Meets a wide range of needs, from foreground uses that need highly accurate location to background uses that need periodic location updates with negligible power impact.
- It also adds geofencing which set up geographic boundaries around specific locations and then receives notifications when the user enters or leaves those areas.
- It also adds activity recognition which makes it easy to check the user's current activity—still, walking, cycling, and in-vehicle—with very efficient use of the battery

Checking for the Google Services API

- The first thing you need to do is check for Google Services on the device, to check that the APK is installed call [GooglePlayServicesUtil.isGooglePlayServicesAvailable\(\)](#), which returns an integer result code listed in the reference documentation for [ConnectionResult](#)

```
int resultCode =
GooglePlayServicesUtil.isGooglePlayServicesAvailable(this);
// If Google Play services is available
    if (ConnectionResult.SUCCESS == resultCode) {
        // In debug mode, log the status
        Log.d("Location Updates", "Google Play services is
available.");
```

Retrieving the Location using Google Services Location API

- First we need to create a client and connect it to the Service, and then call its [getLastLocation\(\)](#) method. The return value is the best, most recent location, based on the permissions your app requested and the currently-enabled location sensors.
- Before you create the location client, implement the interfaces that Location Services uses to communicate with your app:

[ConnectionCallbacks](#) - Specifies methods that Location Services calls when a location client is connected or disconnected.

[OnConnectionFailedListener](#) - Specifies a method that Location Services calls if an error occurs while attempting to connect the location client.

Retrieve User location

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener {
    ...
    /*
     * Called by Location Services when the request to connect the
     * client finishes successfully. At this point, you can
     * request the current location or start periodic updates
     */
    @Override
    public void onConnected(Bundle dataBundle) {
        // Display the connection status
        Toast.makeText(this, "Connected", Toast.LENGTH_SHORT).show();
    }
    @Override
    public void onDisconnected() {
        // Display the connection status
        Toast.makeText(this, "Disconnected. Please re-connect.",
            Toast.LENGTH_SHORT).show();
    }
    ...
    /*
     * Called by Location Services if the attempt to
     * Location Services fails.
     */
}
```

Retrieve User location

```
/*
 * Called by Location Services if the attempt to
 * Location Services fails.
 */
@Override
public void onConnectionFailed(ConnectionResult connectionResult) {
    /*
     * Google Play services can resolve some errors it detects.
     * If the error has a resolution, try sending an Intent to
     * start a Google Play services activity that can resolve
     * error.
     */
    if (connectionResult.hasResolution()) {
        try {
            // Start an Activity that tries to resolve the error
            connectionResult.startResolutionForResult(
                this,
                CONNECTION_FAILURE_RESOLUTION_REQUEST);
        } catch (IntentSender.SendIntentException e) {
            // Log the error
            e.printStackTrace();
        }
    } else {
        /*
         * If no resolution is available, display a dialog to the
         * user with the error.
         */
        showErrorDialog(connectionResult.getErrorCode());
    }
}
```

Retrieve User location

- Now that the callback methods are in place, create the location client and connect it to Location Services. You should create the location client in [onCreate\(\)](#), then connect it in [onStart\(\)](#), so that Location Services maintains the current location while your activity is fully visible.
- Disconnect the client in [onStop\(\)](#), so that when your app is not visible, Location Services is not maintaining the current location. Following this pattern of connection and disconnection helps save battery power.

Retrieve User location

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        /*
         * Create a new location client, using the enclosing class to
         * handle callbacks.
         */
        mLocationClient = new LocationClient(this, this, this);
        ...
    }
    ...
    /*
     * Called when the Activity becomes visible.
     */
    @Override
    protected void onStart() {
        super.onStart();
        // Connect the client.
        mLocationClient.connect();
    }
}
```

```
    ...
    /*
     * Called when the Activity becomes visible.
     */
    @Override
    protected void onStart() {
        super.onStart();
        // Connect the client.
        mLocationClient.connect();
    }
    ...
    /*
     * Called when the Activity is no longer visible.
     */
    @Override
    protected void onStop() {
        // Disconnecting the client invalidates it.
        mLocationClient.disconnect();
        super.onStop();
    }
}
```

Retrieve User location

- To get the current location call the [getLastLocation\(\)](#) method.

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener {
    ...
    // Global variable to hold the current location
    Location mCurrentLocation;
    ...
    mCurrentLocation = mLocationClient.getLastLocation();
    ...
}
```

- You can also receive location updates at regular intervals or when the location changes by implementing the `LocationListener` and the overriding `onLocationChanged`

```
public class MainActivity extends FragmentActivity implements
    GooglePlayServicesClient.ConnectionCallbacks,
    GooglePlayServicesClient.OnConnectionFailedListener,
    LocationListener {
    ...
    // Define the callback method that receives location updates
    @Override
    public void onLocationChanged(Location location) {
        // Report to the UI that the location was updated
        String msg = "Updated Location: " +
            Double.toString(location.getLatitude()) + ", " +
            Double.toString(location.getLongitude());
        Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
    }
    ...
}
```

Retrieve User location

- You also need to request for location updates even though you have the listener and callbacks set up, do this in the onConnected callback

```
//assuming we have the LocationClient and its connected  
mLocationClient.requestLocationUpdates(mLocationRequest, this);
```

- If you do not want location updates anymore you need to call the method:

```
// call this in the onStop() method of the Activity/FragmentActivity  
removeLocationUpdates(this);
```

- Note that the requestLocationUpdates method has a LocationRequest argument. This allows you to control the interval between updates and the location accuracy you want, by setting values in the [LocationRequest](#) object and then sending this object as part of your request to start updates.

Retrieve User location

```
LocationRequest mLocationRequest;
...
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Create the LocationRequest object
    mLocationRequest = LocationRequest.create();
    // Use high accuracy
    mLocationRequest.setPriority(
        LocationRequest.PRIORITY_HIGH_ACCURACY);
    // Set the update interval to 5 seconds
    mLocationRequest.setInterval(5);
    // Set the fastest update interval to 1 second, use this
    //to set an upper limit on the update rate in case other
    //apps have requested faster update rates
    mLocationRequest.setFastestInterval(1);
    ...
}
...
```

Displaying a Location Address

- Although latitude and longitude are useful for calculating distance or displaying a map position, in many cases the address of the location is more useful.
- The Android platform API provides a feature that returns an estimated street address for latitude and longitude values.
- To get an address for a given latitude and longitude, call [Geocoder.getFromLocation\(\)](#), which returns a list of addresses. The method may take a long time to do its work, so you should call the method from the [doInBackground\(\)](#) method of an [AsyncTask](#).
- You should really show a progress bar or some other indeterminate activity while your app is performing the lookup

Displaying a Location Address

```
//...in doInBackground of AsyncTask

Geocoder geocoder = new Geocoder(mContext, Locale.getDefault());
List<Address> addresses = null;
try {
    /*
     * Return 1 address.
     */
    addresses = geocoder.getFromLocation(loc.getLatitude(),loc.getLongitude(), 1);

}catch(Exception ex) {
    //handle exceptions in here
}


// If the geocoder returned an address
if (addresses != null && addresses.size() > 0) {
    // Get the first address
    Address address = addresses.get(0);
    /*
     * Format the first line of address (if available),
     * city, and country name.
     */
    String addressText = String.format("%s, %s, %s",
    // If there's a street address, add it
    address.getMaxAddressLineIndex() > 0 ?
        address.getAddressLine(0) : "",
    // Locality is usually a city
    address.getLocality(),
    // The country of the address
    address.getCountryName());
    // Return the text
    return addressText;
}
else {
    return "No address found";
}
```

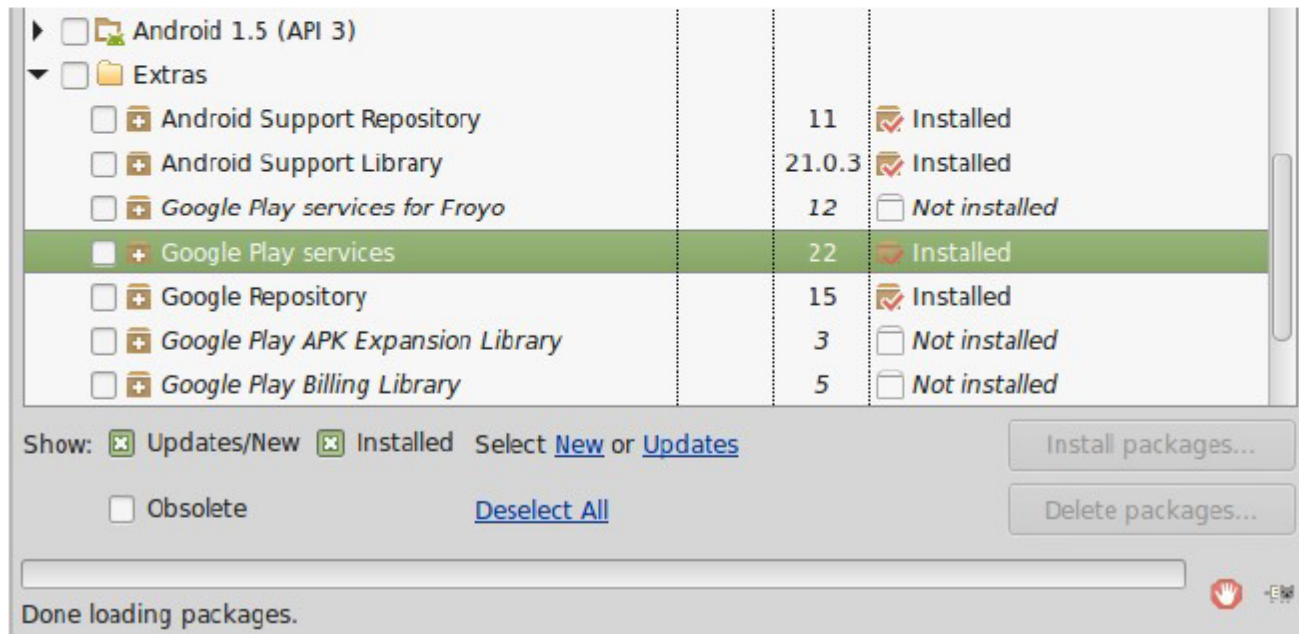
Geofences

- Geofencing combines awareness of the user's current location with awareness of nearby features, defined as the user's proximity to locations that may be of interest. To mark a location of interest, you specify its latitude and longitude. To adjust the proximity for the location, you add a radius. The latitude, longitude, and radius define a geofence. You can have multiple active geofences at one time.
- Location Services treats geofences as an area rather than as a points and proximity. This allows it to detect when the user enters or exits a geofence. For each geofence, you can ask Location Services to send you entrance events or exit events or both. You can also limit the duration of a geofence by specifying an expiration duration in milliseconds. After the geofence expires, Location Services automatically removes it.
- If you wish to add geofences to you applications, take a look at the following tutorial:

<https://developer.android.com/training/location/geofencing.html>

Google Play Services

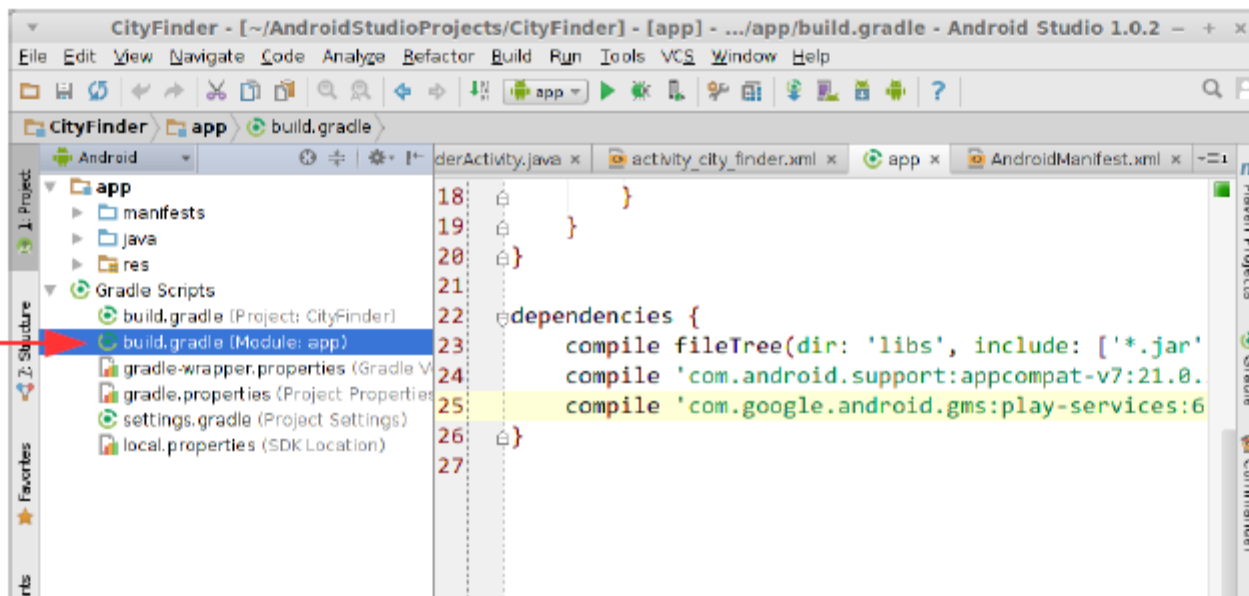
- need to install **Google Play** services
 - SDK Manager  → Extras → Google Play services (check box)
 - click Install packages...



Google Play Services

- add Google Play to project in app's **build.gradle** file

```
dependencies {  
    compile fileTree(dir: 'libs', include: ['*.jar'])  
    compile 'com.android.support:appcompat-v7:21.0.3'  
    compile 'com.google.android.gms:play-services:6.5.87'  
}
```

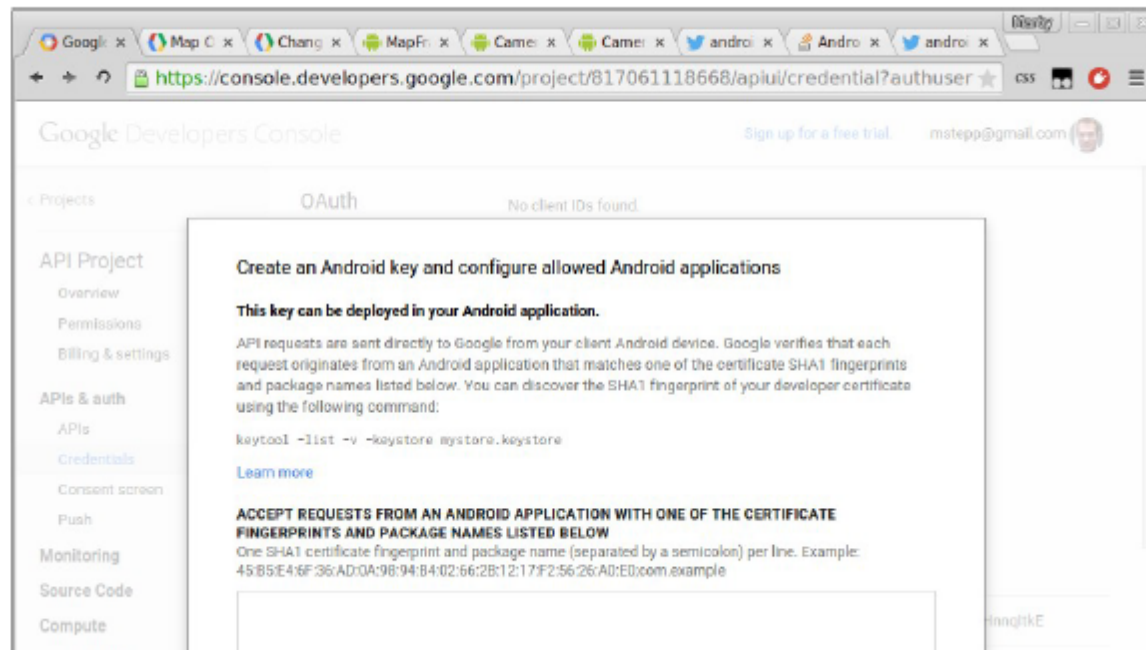


Google Play Services – API key

- Google won't allow you to fetch map data without an **API key**.
- To get a key, open a Terminal and find the file **debug.keystore**:
 - Windows (new): C:\Users\USERNAME\.android
 - Windows (old): C:\Documents and Settings\USERNAME\.android
 - Linux: /home/USERNAME/.android/
 - Mac: /Users/USERNAME/.android/ (?)
- In the terminal, **cd** to that directory, then type:
`keytool -list -v -keystore debug.keystore`
(it asks for a password, so just press Enter)
- Find the line with your "Certificate fingerprint" for "SHA-1". It should contain a long string in this format. Copy it down.
 - BD:2B:3F:4B:.....

Google Play Services – API key

- Go to the Google APIs developer console:
 - <https://code.google.com/apis/console/>
 - click APIs and Auth → Credentials → Create new Key
 - choose Android Key
 - paste in the SHA-1 key you got from the previous slide



Manifest changes

- To use maps in your app, must make some manifest changes:

```
<manifest ...>
```

```
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-feature android:glEsVersion="0x00020000" android:required="true" />
```

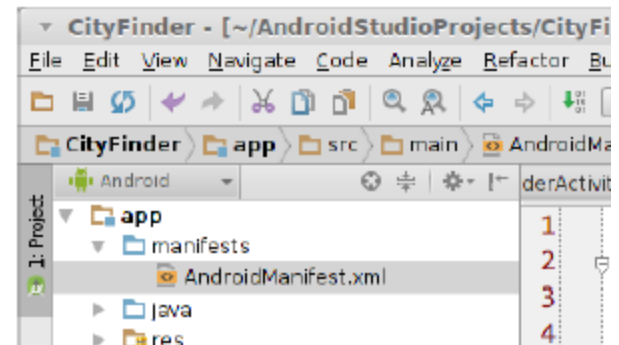
```
<application ...>
```

```
    <meta-data android:name="com.google.android.gms.version"
               android:value="@integer/google_play_services_version" />
    <meta-data android:name="com.google.android.maps.v2.API_KEY"
               android:value="your API key" />
```

```
    <activity ...> ... </activity>
```

```
</application>
```

```
</manifest>
```



Map Fragment

- Google Maps API provides a fragment class named MapFragment for displaying a map within an activity.

```
<LinearLayout ...  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:map="http://schemas.android.com/apk/res-auto"  
    tools:ignore="MissingPrefix">  
  
    <fragment ...  
        android:name="com.google.android.gms.maps.MapFragment"  
        android:id="@+id/ID" />  
  
</LinearLayout>
```

- (There is also a MapView class that we won't cover)*



Waiting for Map to load

```
public class Name extends Activity
    implements OnMapReadyCallback, GoogleMap.OnMapLoadedCallback {
    private GoogleMap map = null;

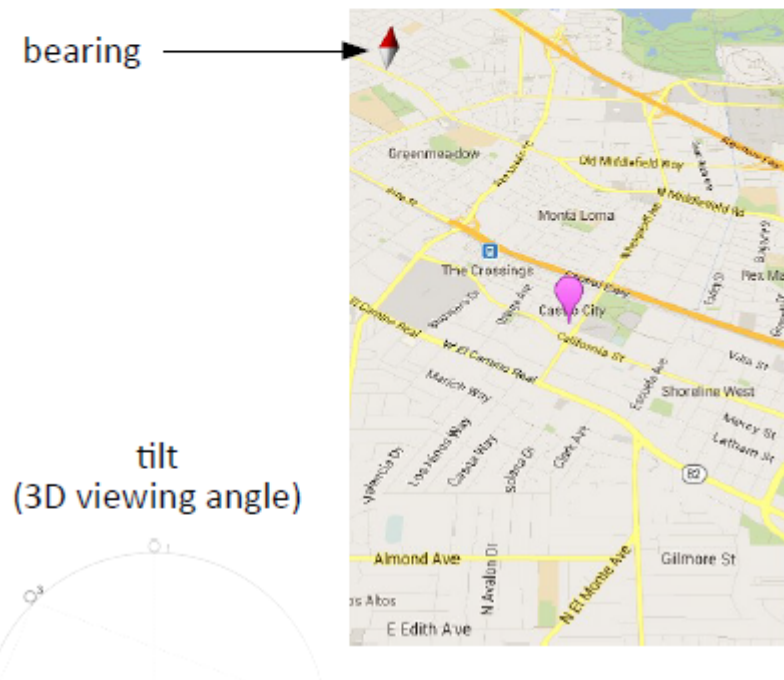
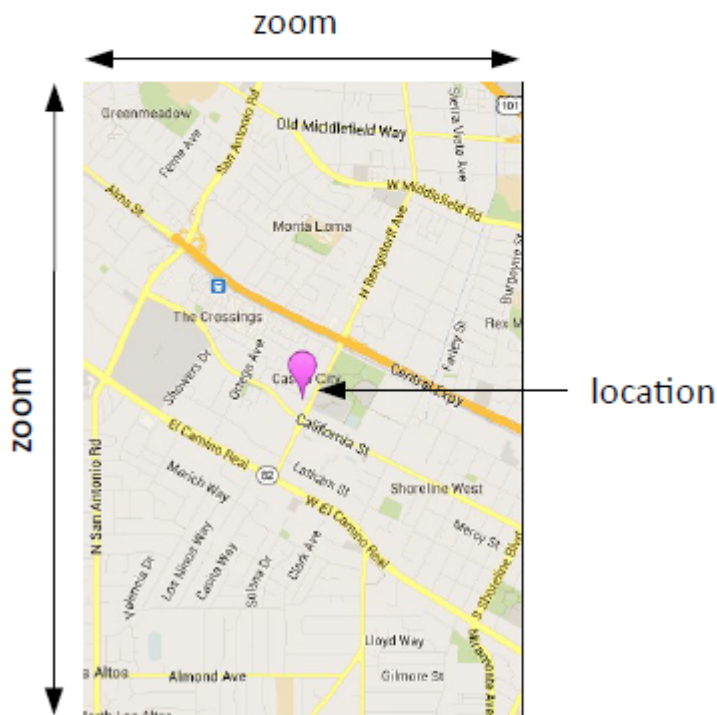
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        MapFragment mf = (MapFragment) getFragmentManager().findFragmentById(R.id.ID);
        mf.getMapAsync(this);           // calls onMapReady when loaded
    }
    ↓
    @Override
    public void onMapReady(GoogleMap map) { // map is loaded but not laid out yet
        map.setOnMapLoadedCallback(this); // calls onMapLoaded when layout done
    }
    ↓
    @Override
    public void onMapLoaded() {
        code to run when the map has loaded;
    }
}
```

GoogleMap methods

- placing items on the map:
 - `addCircle`, `addGroundOverlay`, **`addMarker`**, `addPolygon`, **`addPolyline`**, `addTileOverlay`
 - **`clear`** - Removes all markers, polylines/polygons, overlays
- manipulating the camera:
 - `getCameraPosition`, **`moveCamera`**, **`animateCamera`**, `stopAnimation`
- map settings and appearance:
 - `setBuildingsEnabled`, `setIndoorEnabled`, `setMapType`, `setPadding`, `setTrafficEnabled`
- snapshot - take a screen shot of the map as a bitmap
- event listeners:
 - `setOnCameraChangeListener`, **`setOnMapClickListener`**, `setOnMapLoadedCallback`, `setOnMapLongClickListener`, **`setOnMarkerClickListener`**, `setOnMarkerDragListener`, `setOnMyLocationChangeListener`

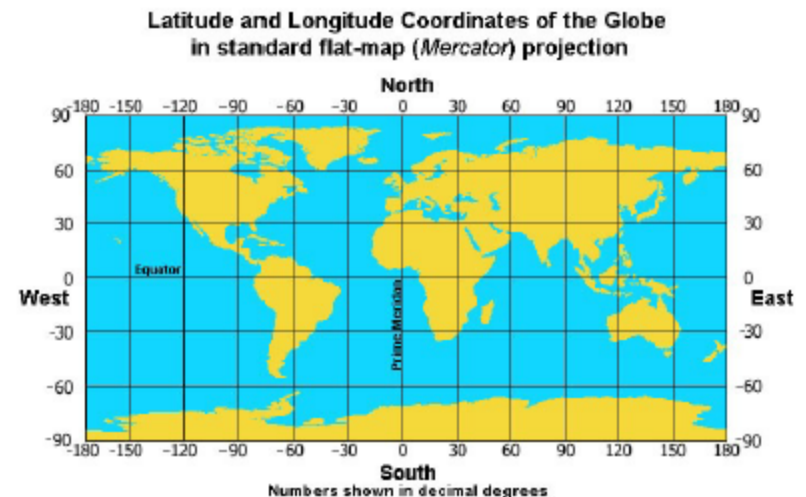
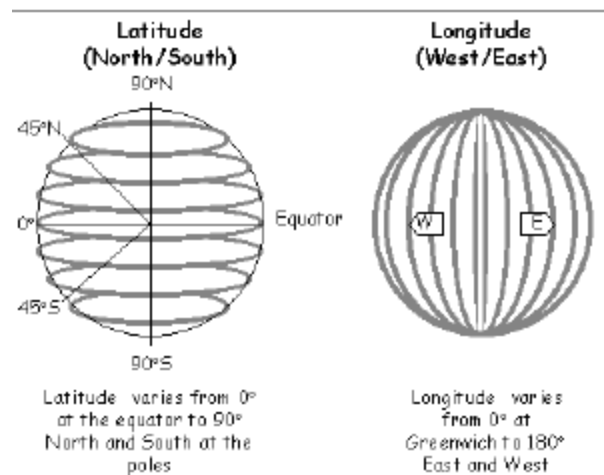
Map's Camera

- The current viewing window of a map's camera is defined by:
 - **target** location (latitude/longitude), **zoom** (2.0 - 21.0),
 - **bearing** (orientation/rotation), and **tilt** (degrees)



Latitude and Longitude

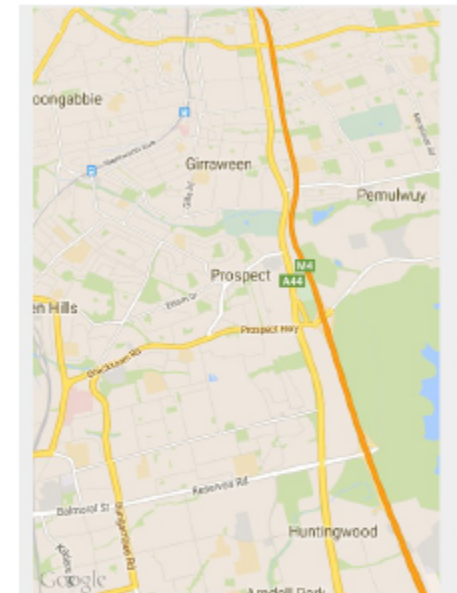
- **latitude:** N/S angle relative to the equator
 - North pole = +90; South pole = -90
- **longitude:** E/W angle relative to prime meridian
 - West = 0 \rightarrow -180; East = 0 \rightarrow 180
 - *find lat/long of a place on Google Maps in URL address bar*
see also: <http://itouchmap.com/latlong.html>



Set Camera in XML

- Set initial map settings and camera position in the layout XML:

```
<fragment ...  
    android:name="com.google.android.gms.maps.MapFragment"  
    android:id="@+id/ID"  
    map:cameraBearing="112.5"  
    map:cameraTargetLat="-33.796923"  
    map:cameraTargetLng="150.922433"  
    map:cameraTilt="30"  
    map:cameraZoom="13"  
    map:mapType="normal"  
    map:uiCompass="false"  
    map:uiRotateGestures="true"  
    map:uiScrollGestures="false"  
    map:uiTiltGestures="true"  
    map:uiZoomControls="false"  
    map:uiZoomGestures="true" />
```



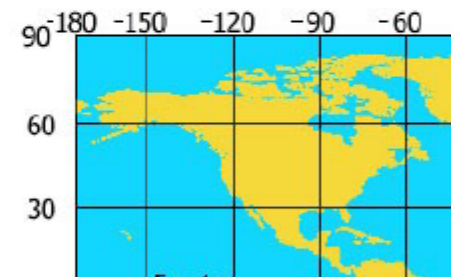
Set Camera in Java

- CameraUpdateFactory methods:
 - newLatLng(new LatLng(*lat*, *lng*))
 - newLatLngBounds(new LatLngBounds(*SW*, *NE*), *padding*)
 - newLatLngZoom(new LatLng(*lat*, *lng*), *zoom*)
 - newCameraPosition(*CameraPosition*)
 - others:

```
// example; show roughly the entire USA
LatLngBounds bounds = new LatLngBounds(
    new LatLng(20, -130.0),    // SW
    new LatLng(55, -70.0));    // NE
```

```
map.moveCamera(CameraUpdateFactory.newLatLngBounds(bounds, 50));
```

```
// try also: map.animateCamera
```



Placing Markers

- A `GoogleMap` object has an `addMarker` method that can let you put "push pin" markers at locations on the map.
 - The marker's methods return the marker, so you can chain them.
 - options: `alpha`, `draggable`, `icon`, `position`, `rotation`, `title`, `visible`, ...

```
map.addMarker(new MarkerOptions()  
    .position(new LatLng(40.801, -96.691))  
    .title("Lincoln, NE")  
);
```

```
// to modify/remove the marker later  
Marker mark = map.addMarker(new MarkerOptions()  
    ...);  
mark.remove();
```

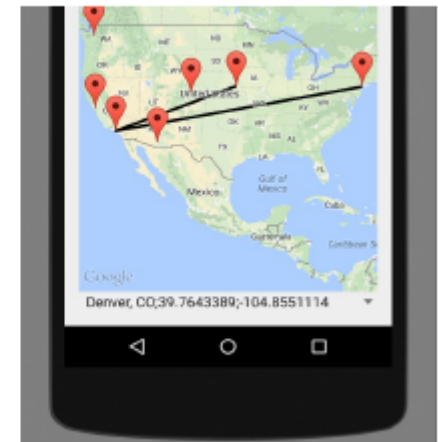


Lines and Paths

- A GoogleMap object has an addPolyline method that can let you put lines between locations on the map.
 - options: color, visible, width, zIndex, ...

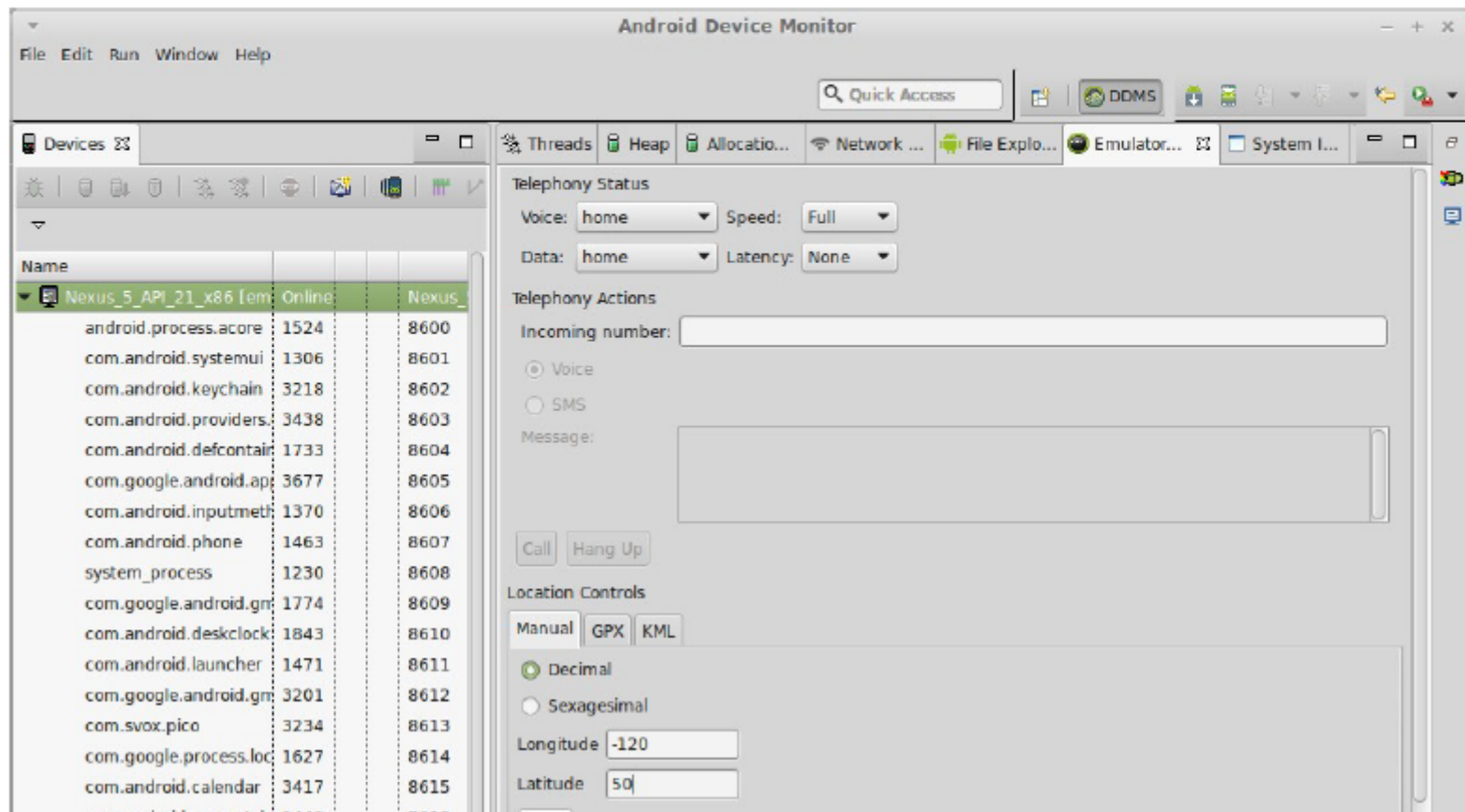
```
map.addPolyline(new PolylineOptions()  
    .add(new LatLng(40.801, -96.691))    // Lincoln, NE  
    .add(new LatLng(34.020, -118.412))   // Los Angeles, CA  
    .add(new LatLng(40.703, -73.980))    // New York, NY  
);
```

```
// to modify/remove the line later  
Polyline polly = map.addPolyline(...);  
polly.remove();
```



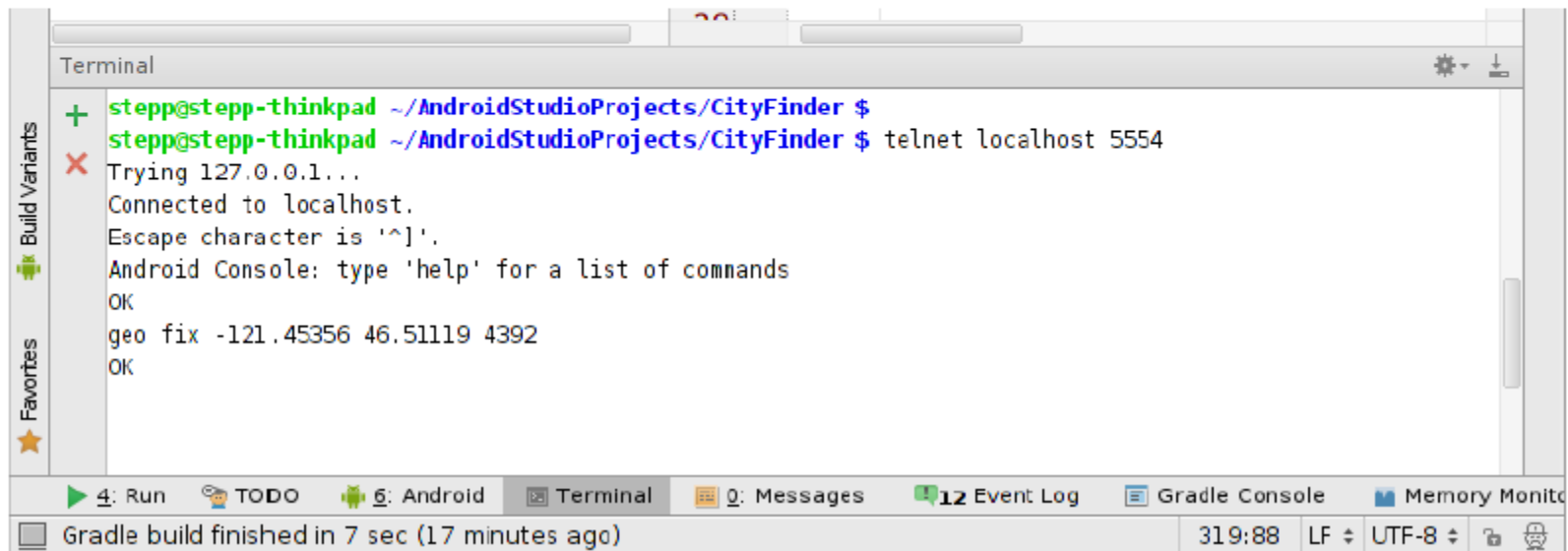
Mock Locations

- Android Device Monitor → Emulator Controls → Location
 - in device, click Settings → Location → On



Mock Locations

- Another way: Open a **Terminal**, and type:
`telnet localhost 5554`
- once connected, type: *(altitude is optional)*
`geo fix latitude longitude altitude`



```
Terminal
+ stepp@stepp-thinkpad ~/AndroidStudioProjects/CityFinder $
X stepp@stepp-thinkpad ~/AndroidStudioProjects/CityFinder $ telnet localhost 5554
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: type 'help' for a list of commands
OK
geo fix -121.45356 46.51119 4392
OK
```

The screenshot shows the Android Studio interface with the Terminal tab active. The terminal output shows a successful telnet connection to localhost on port 5554. After sending the 'geo fix' command with coordinates and altitude, the service responds with 'OK'. The bottom status bar indicates the Gradle build finished in 7 seconds.