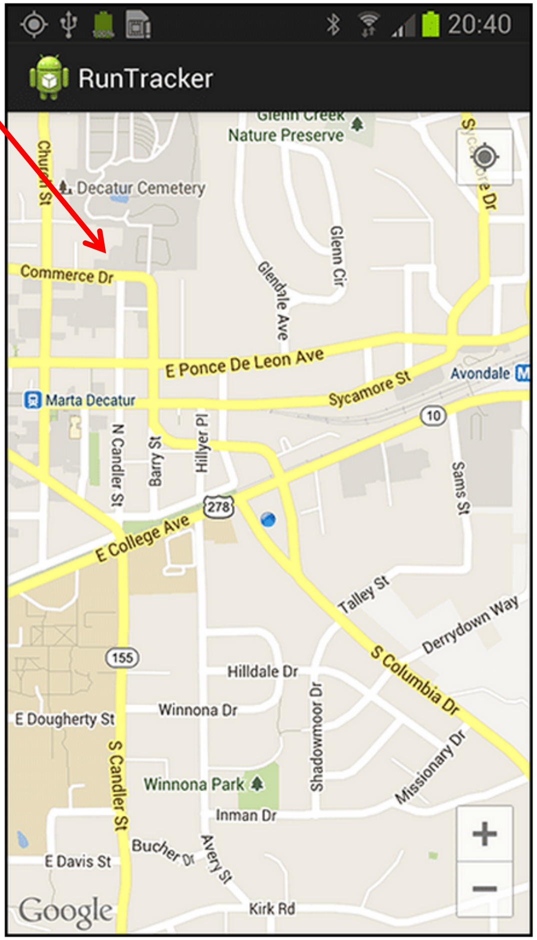
Using Maps

Slide courtesy of Emmanuel Agu

Introducing MapView and Map Activity

- **MapView:** UI widget that displays maps
- **MapActivity:** java class (extends Activity), handles map-related lifecycle and management for displaying maps.
- **Overlay:** java class used to annotate map, use a canvas to draw unto map layers
- **MapController:** enables map control, setting center location and zoom levels



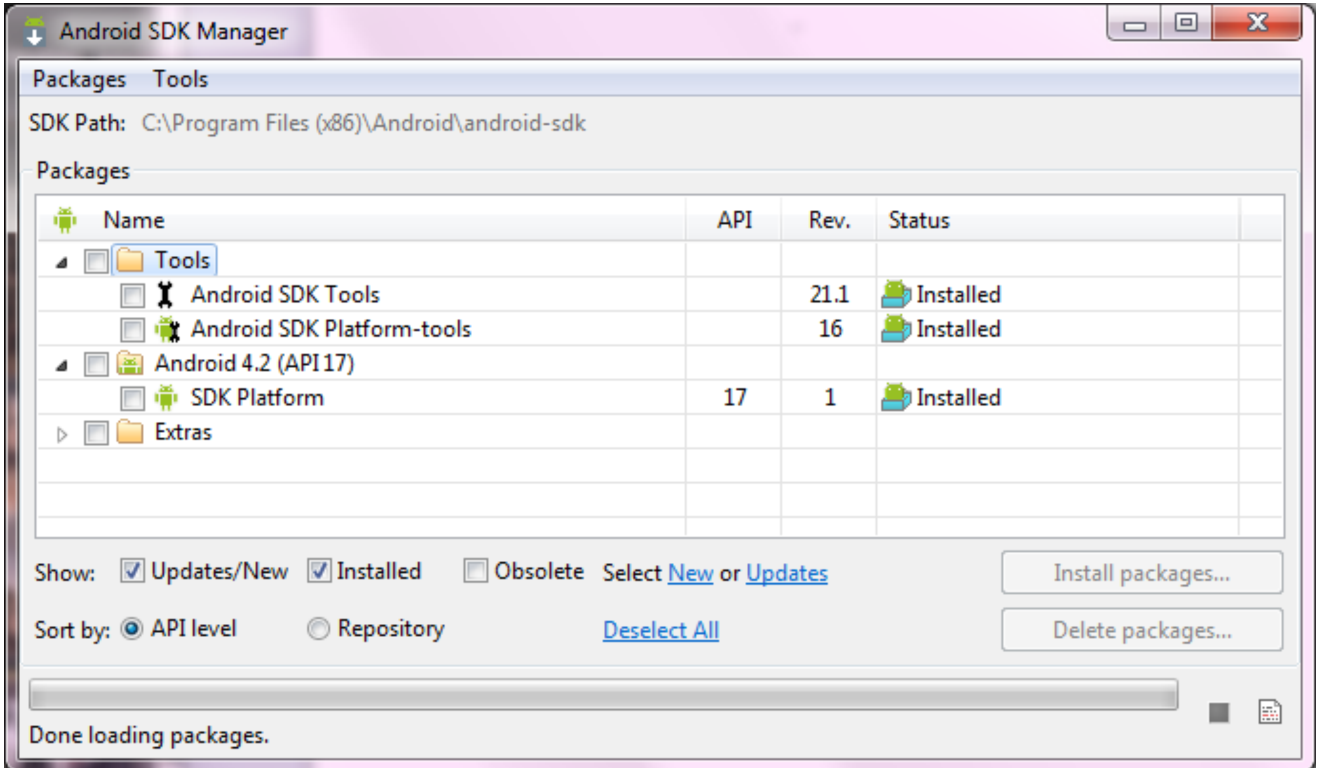
Steps for using Google Maps Android API v2

1. Install Android SDK (Done already!)
2. Use Android Studio SDK manager to add Google Play services
3. Obtain Google Maps API key
4. Add required settings (permissions, etc) to Android Manifest
5. Add a map to app

Step 2: Add Google Play Services to Your Project

- Google Maps API v2 is part of Google Play Services SDK
- Main steps to set up Google Play Services

  (See: https://developers.google.com/android/guides/setup)
- Use Android Studio SDK manager to download Google Play services



SDK Path: C:\Program Files (x86)\Android\android-sdk

Packages

Name	API	Rev.	Status
□ Tools			
□ Android SDK Tools		21.1	Installed
□ Android SDK Platform-tools		16	Installed
□ Android 4.2 (API 17)			
□ SDK Platform	17	1	Installed
□ Extras			

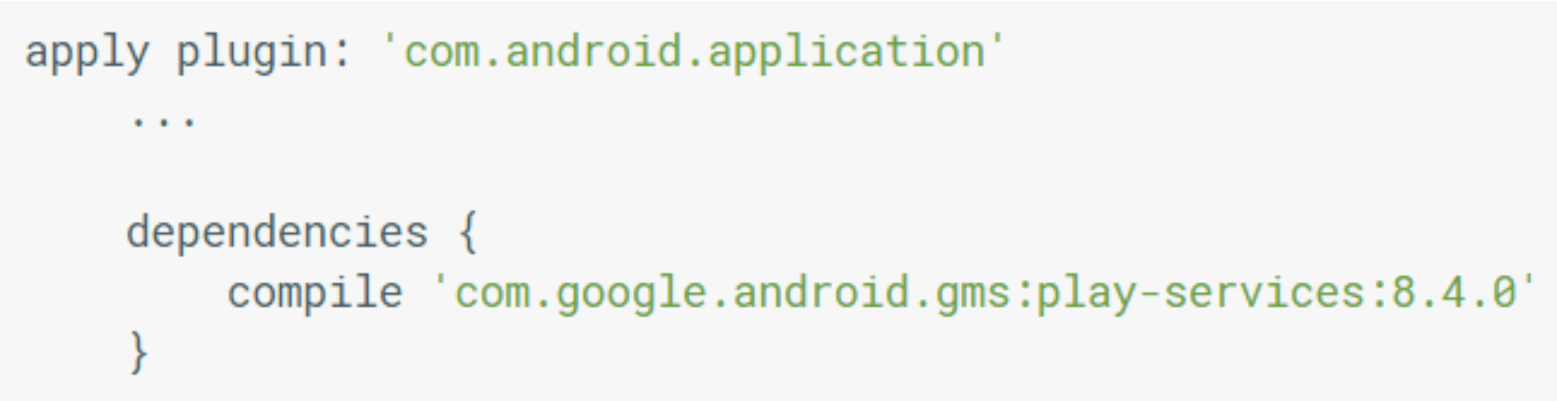
Show: ☑ Updates/New ☑ Installed □ Obsolete Select New or Updates

Sort by: ● API level ○ Repository Deselect All

Done loading packages.

Step 2: Add Google Play Services to Your Project

2. Open **build.gradle** inside your application
3. Add new build rule under **dependencies**



```
apply plugin: 'com.android.application'
    ...

    dependencies {
        compile 'com.google.android.gms:play-services:8.4.0'
    }
```

Step 3: Get Google Maps API key

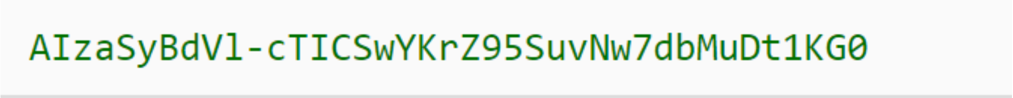
- To access Google Maps servers using Maps API, must add Maps API key to app
- Maps API key is free
- **Background:** Before they can be installed, android apps must be signed with digital certificate (developer holds private key)
- Digital certificates uniquely identify an app, used in tracking:
  - Apps within Google Play Store and
  - App's use of resources such as Google Map servers
- Android apps often use self-signed certificates, not authority
- **See: https://developers.google.com/maps/documentation/android-api/signup**

Step 3: Get Google Maps API key (Contd)

- To obtain a Maps API key, app developer provides:
  - App’s signing certificate + its package name
- Maps API keys linked to specific **certificate/package pairs**
- Steps to obtain a Maps API key:
  - Retrieve information about app’s certificate
  - Register a project in Google APIs console and add the Maps API as a service for the project
  - Request one or more keys
  - Add key to app and begin development
  - **See: https://developers.google.com/maps/documentation/android/start**

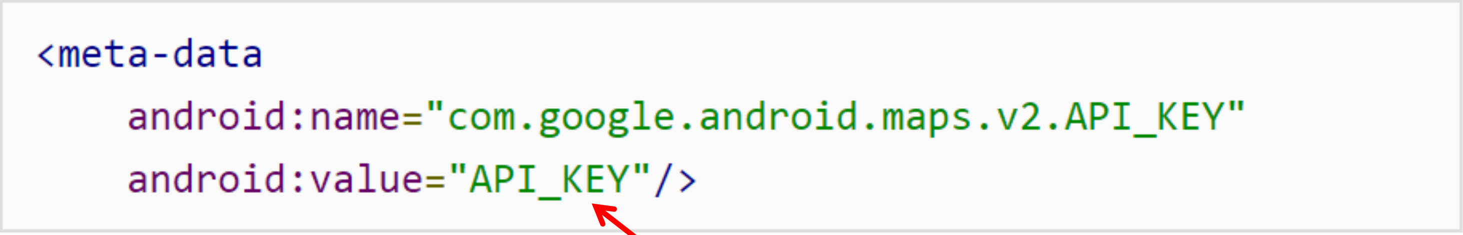
Step 3: Get Google Maps API key (Contd)

- If successful, 40-character API key generated, for example



```
AIzaSyBdVl-cTICSwYKrZ95SuvNw7dbMuDt1KG0
```

- Add this API key to app in order to use Maps API
- Include API key in AndroidManifest.xml
- To modify AndroidManifest.xml, add following between <application> … </application>



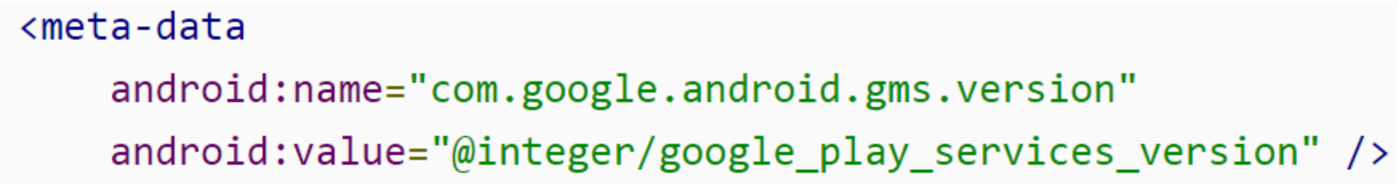
```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="API_KEY"/>
```

Insert Maps API key here
Makes API key visible to any MapFragment in app

- Maps API reads key value from AndroidManifest.xml, passes it to Google Maps server to authenticate access

Step 4: Add Settings to AndroidManifest.xml

- Add Google Play services version to AndroidManifest.xml



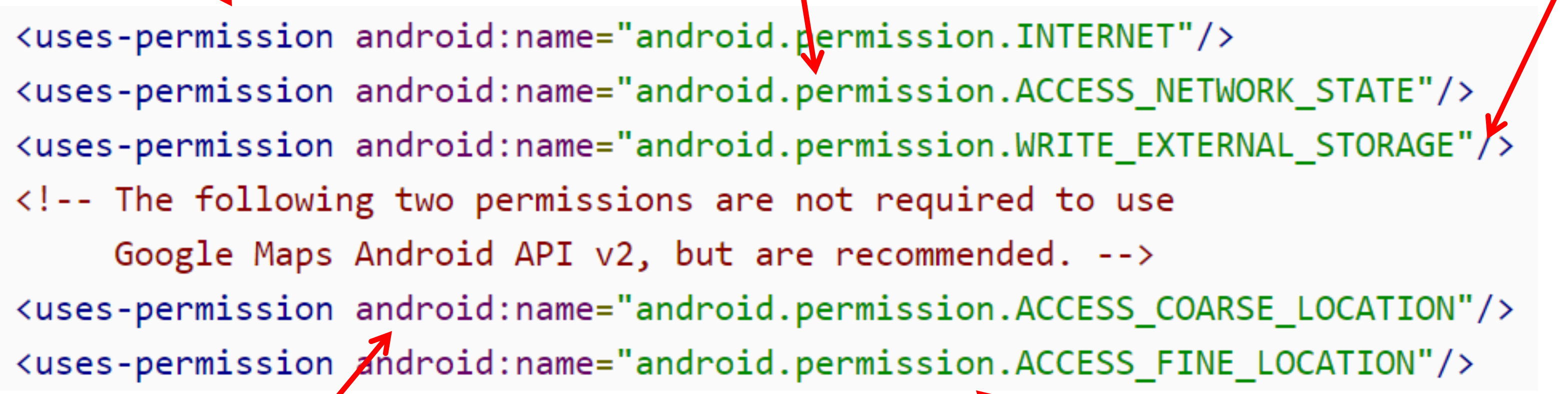
```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

- Request the following permissions:

Used by API to download map tiles from Google Maps servers

Allows the API to check the connection status to determine if data can be downloaded

Used by API to cache map tile data in device's external storage



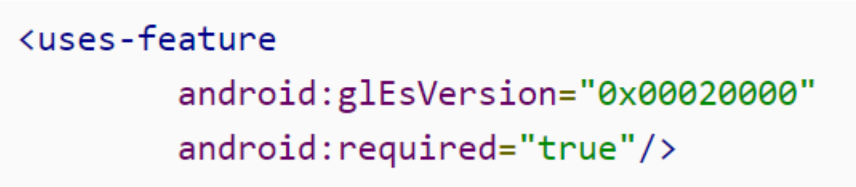
```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<!-- The following two permissions are not required to use
     Google Maps Android API v2, but are recommended. -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Allows API to use WiFI or mobile cell data (or both) to determine the device's location

Allows the API to use GPS to determine device's location within a small area

Step 4: Add Settings to AndroidManifest.xml (Contd)

- Specify that OpenGL ES version 2 is required
- Why? Google Maps Android API uses OpenGL ES version 2 to render the map

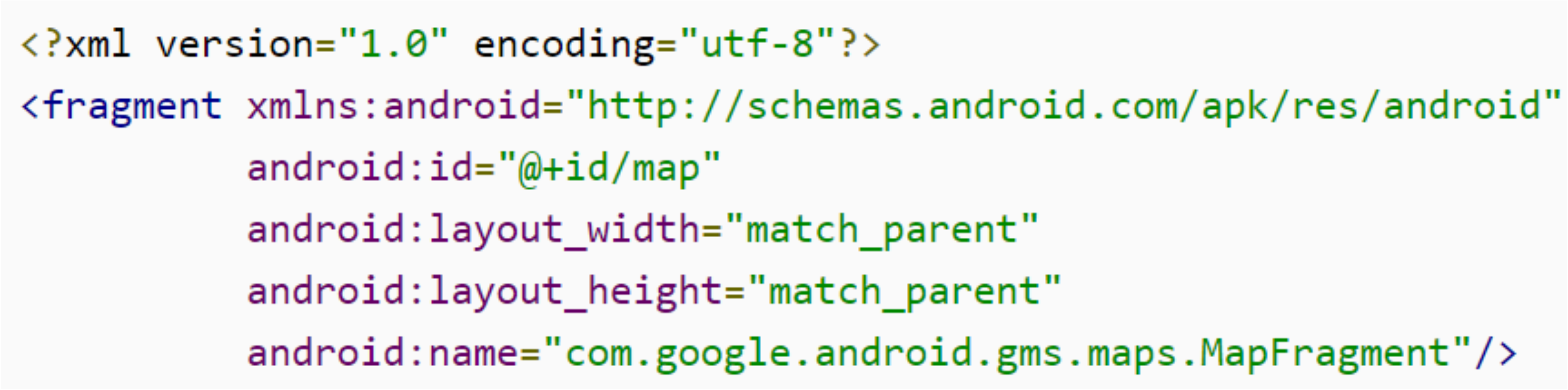


```
<uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>
```

- Due to above declaration, devices that don’t have OpenGL ES version 2 will not see the app on Google Play

Step 5: Add a map

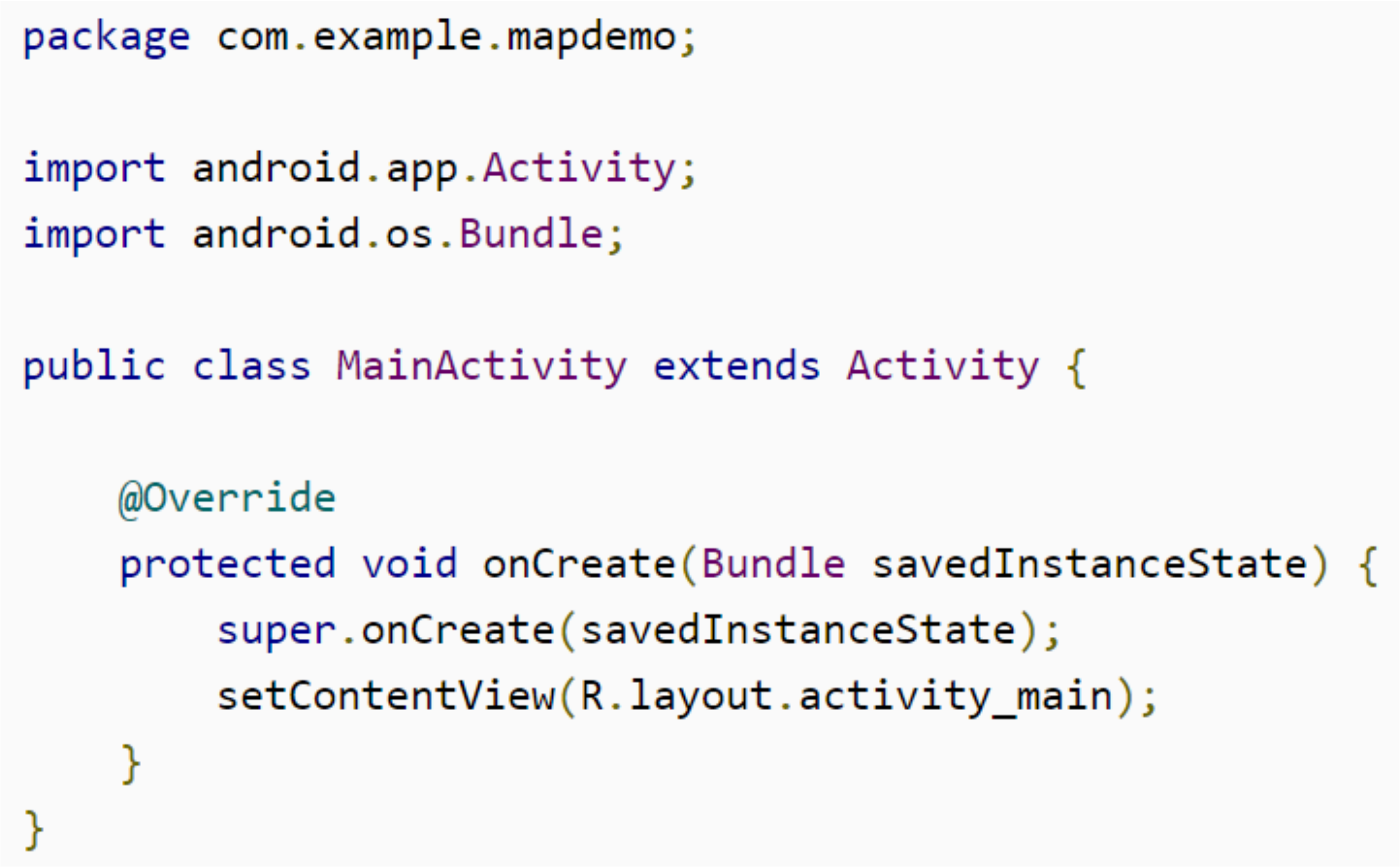
- To add a map, create XML layout file



```
<?xml version="1.0" encoding="utf-8"?>
<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"/>
```

Install & Configure Google Play Services SDK

- And create MainActivity.java



```java
package com.example.mapdemo;

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

public class MainActivity extends Activity {

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

Playing Audio and Video

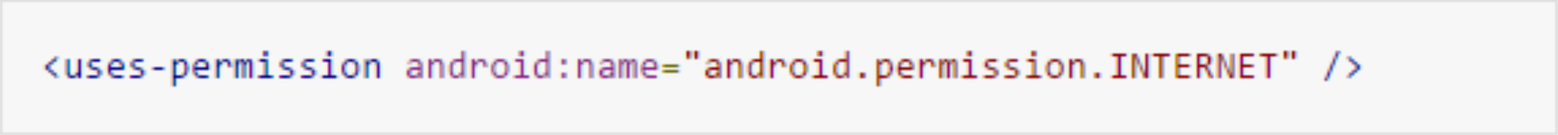
Media Playback

Ref:http://developer.android.com/guide/topics/media/mediaplayer.html

- Controls playback of audio/video files & streams
- Audio/video files stored in app’s resource folders
- App can use Media Playback APIs (e.g. MediaPlayer APIs), functionality easily integrated
- Classes used to play sound and video in Android
  - **MediaPlayer:** Primary class for playing sound and video
  - **AudioManager:** plays audio

Media Player: Manifest Declarations

- If MediaPlayer streams network-based content, request network access permission



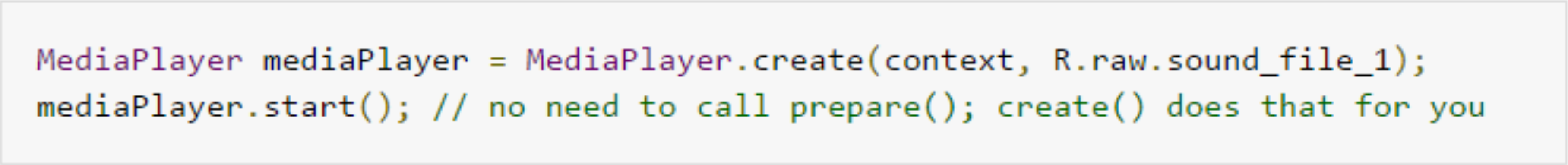
```
<uses-permission android:name="android.permission.INTERNET" />
```

Using MediaPlayer

- A MediaPlayer object can fetch, decode and play audio and video from:
  - Local resources
  - External URLs
- Supports:
  - **Network protocols:** RTSP, HTTP streaming
  - **Media Formats:** Audio (AAC, MP3, MIDI, etc), image (JPEG, GIF, PNG, BMP, etc) and video (H.263, H.264, H.265 AVC, MPEG-4, etc)

Using MediaPlayer

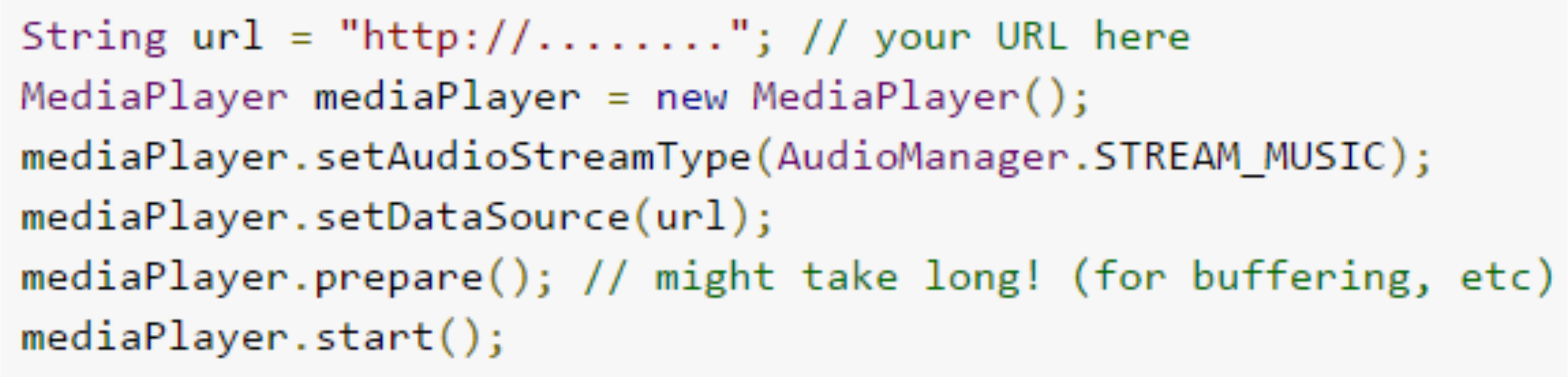
- To play audio file saved in app’s **res/raw/** directory



```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you
```

- Audio file called by create must be encoded in one of supported media formats

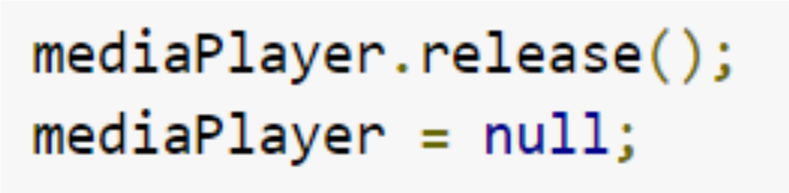
- To play from remote URL via HTTP streaming



```
String url = "http://........"; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // might take long! (for buffering, etc)
mediaPlayer.start();
```

Releasing the MediaPlayer

- MediaPlayer can consume valuable system resources
- When done, always call **release( )** to free up system resources



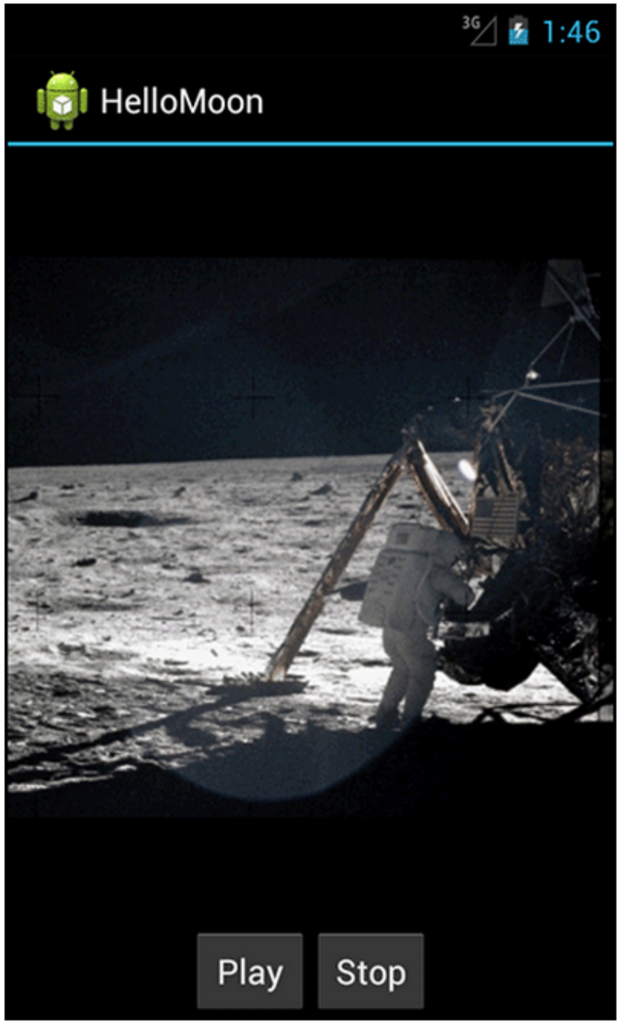
```
mediaPlayer.release();
mediaPlayer = null;
```

- Typically call **release( )** in **onStop( )** or **onDestroy( )** methods
- If you want playback even when app is not onscreen, start MediaPlayer from a Service

Playing Audio File using MediaPlayer Example from Android Nerd Ranch 1st edition

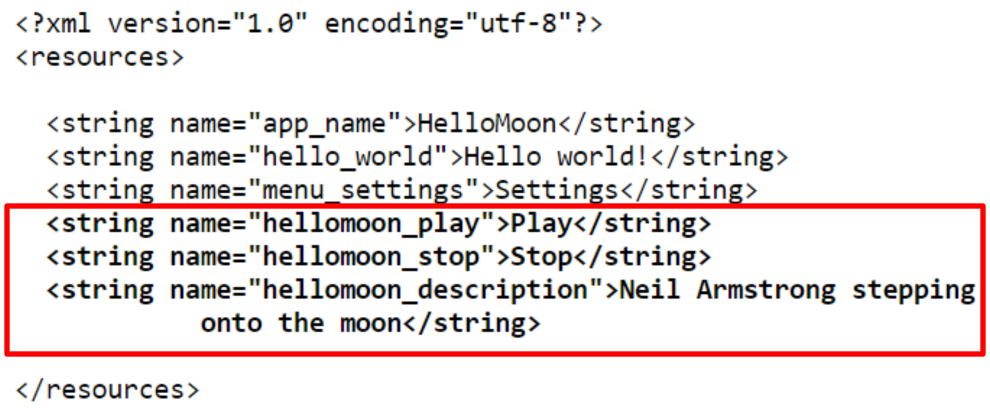
Example taken from Android Nerd Ranch Chapter 13

- Example creates **HelloMoon app** that uses **MediaPlayer** to play audio file
- Android Class for audio and video playback
- **Source:** Can play local files, or streamed over Internet
- **Supported formats:** WAV, MP3, Ogg, Vorbis, MPEG-4, 3GPP, etc



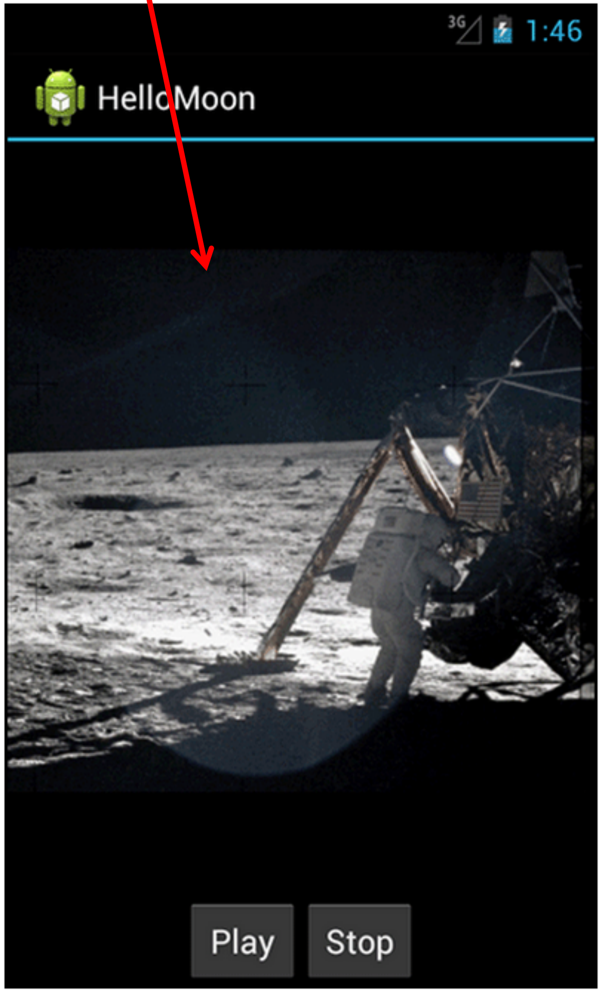
HelloMood App

- Put image **armstrong_on_moon.jpg** in **res/drawable-mdpi/** folder
- Place audio file to be played back (**one_small_step.wav**) in **res/raw** folder
- Can also copy mpeg file and play it back
- Create **strings.xml** file for app



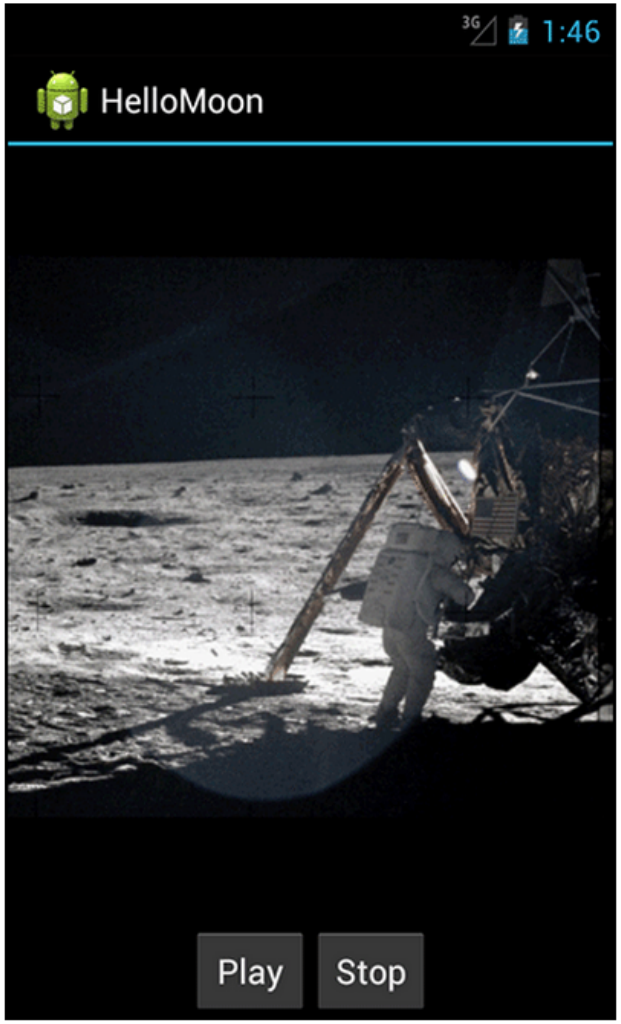
```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="app_name">HelloMoon</string>
  <string name="hello_world">Hello world!</string>
  <string name="menu_settings">Settings</string>
  <string name="hellomoon_play">Play</string>
  <string name="hellomoon_stop">Stop</string>
  <string name="hellomoon_description">Neil Armstrong stepping
            onto the moon</string>

</resources>
```

armstrong_on_moon.jpg

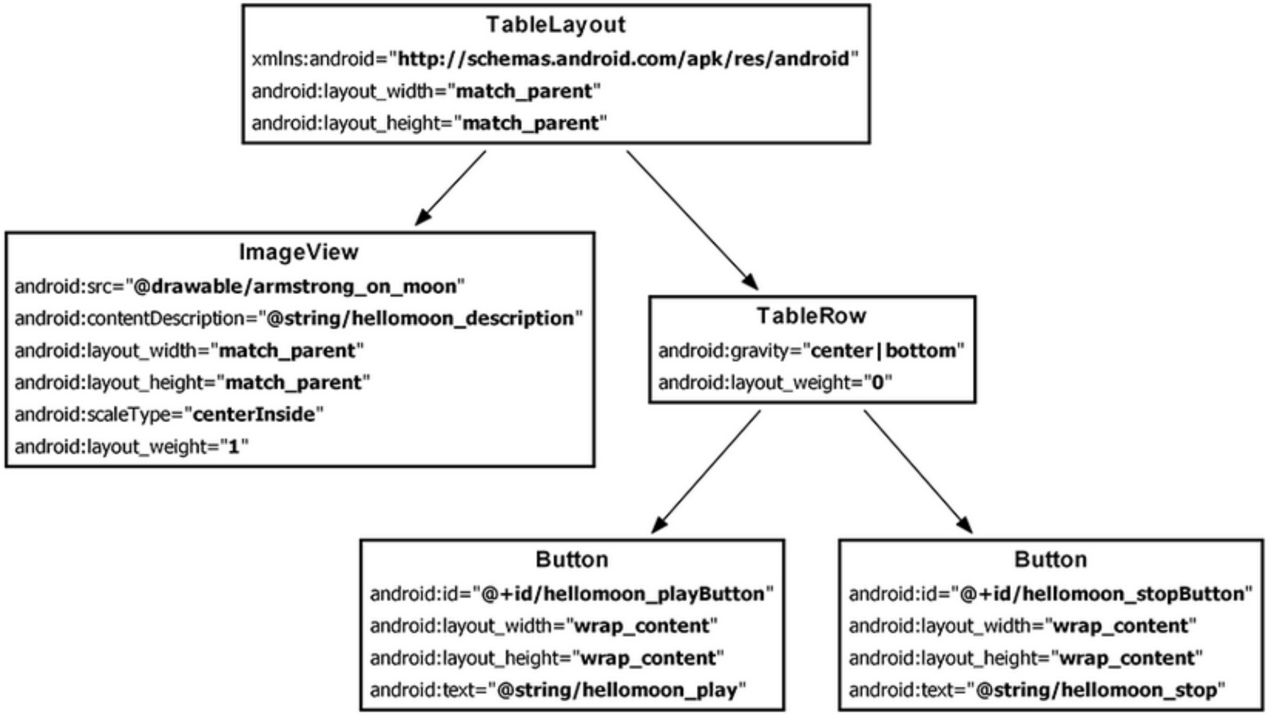


HelloMoon App

- HelloMoon app will have:
  - 1 activity (**HelloMoonActivity**) that hosts **HelloMoonFragment**
- **AudioPlayer** class will be created to encapsulate **MediaPlayer**
- First set up the rest of the app by
  1. Define a layout for the fragment
  2. Create the fragment class
  3. Modify the activity and its layout to host the fragment



Defining the Layout for HelloMoonFragment



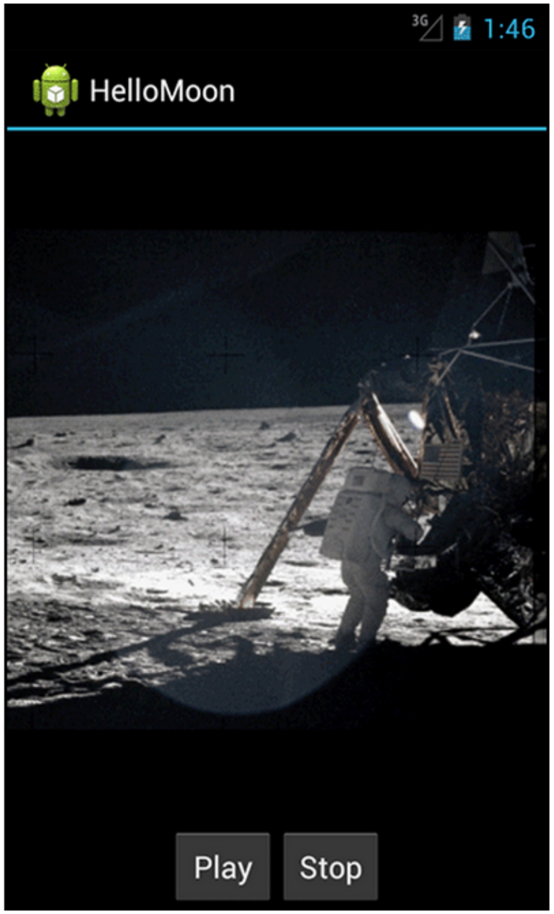
TableLayout
xmlns:android="**http://schemas.android.com/apk/res/android**"
android:layout_width="**match_parent**"
android:layout_height="**match_parent**"

ImageView
android:src="**@drawable/armstrong_on_moon**"
android:contentDescription="**@string/hellomoon_description**"
android:layout_width="**match_parent**"
android:layout_height="**match_parent**"
android:scaleType="**centerInside**"
android:layout_weight="**1**"

TableRow
android:gravity="**center|bottom**"
android:layout_weight="**0**"

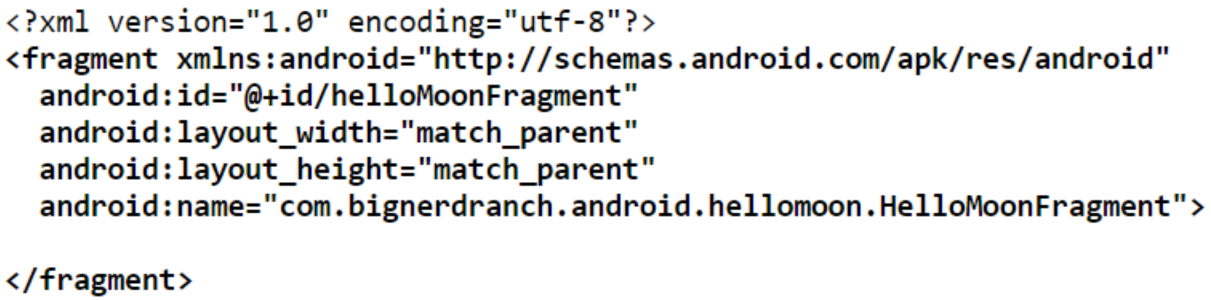
Button
android:id="**@+id/hellomoon_playButton**"
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:text="**@string/hellomoon_play**"

Button
android:id="**@+id/hellomoon_stopButton**"
android:layout_width="**wrap_content**"
android:layout_height="**wrap_content**"
android:text="**@string/hellomoon_stop**"

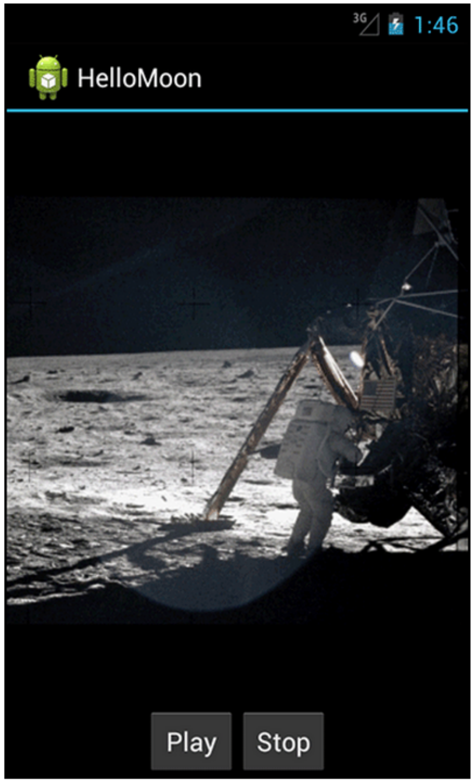


Creating a Layout Fragment

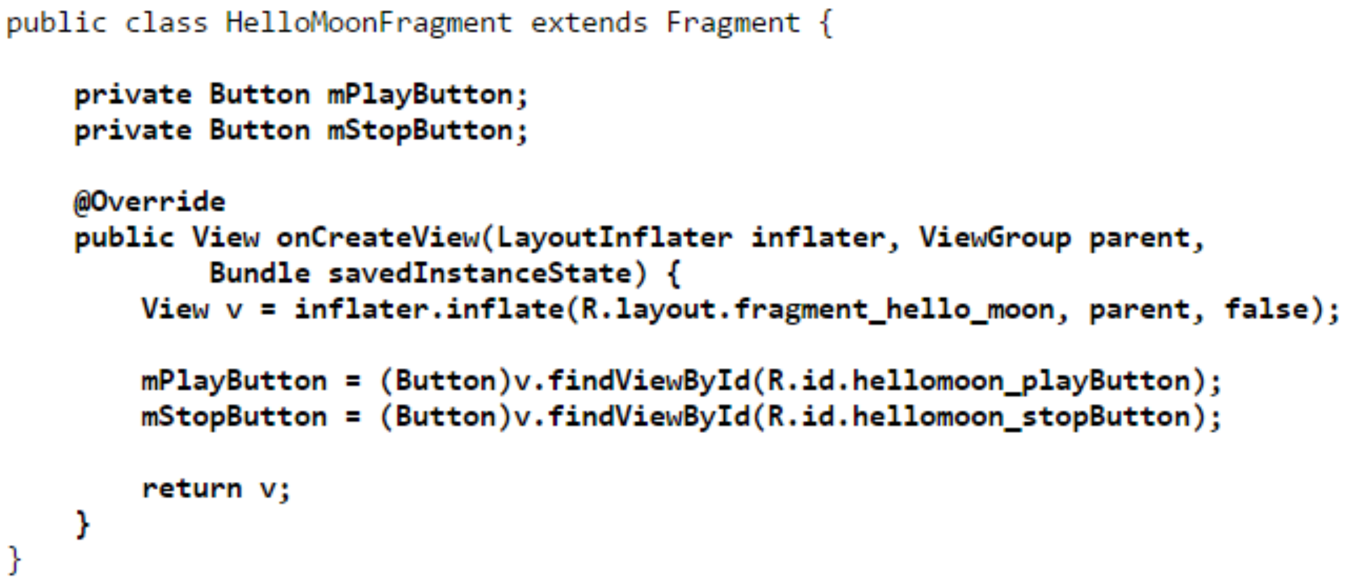
- Previously added Fragments to activity's java code
- Layout fragment enables fragment views to be inflated from XML file
- We will use a layout fragment instead
- Create layout fragment **activity_hello_moon.xml**



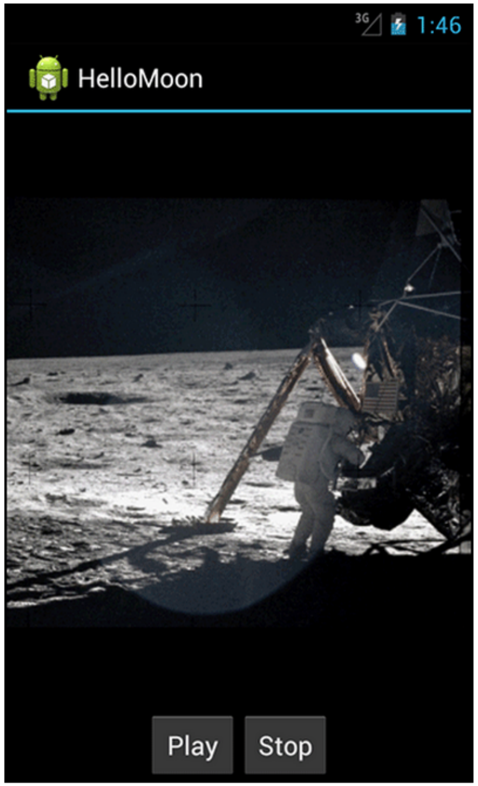
```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/helloMoonFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">

</fragment>
```



Set up HelloMoonFragment

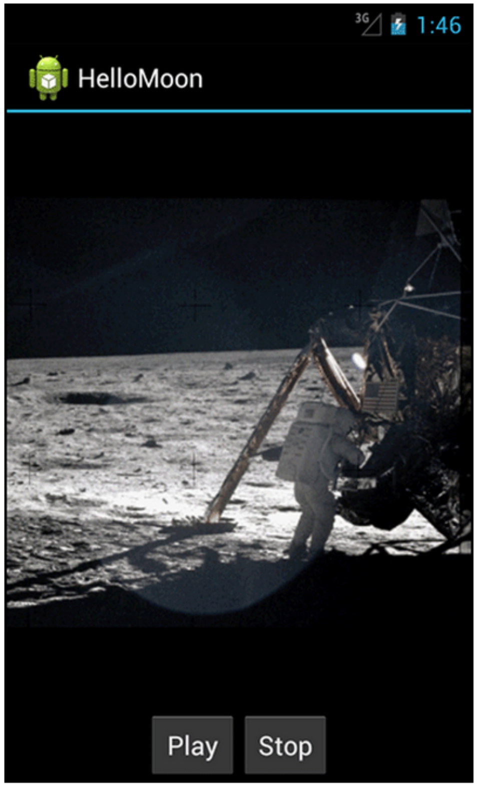


```
public class HelloMoonFragment extends Fragment {

    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);

        return v;
    }
}
```



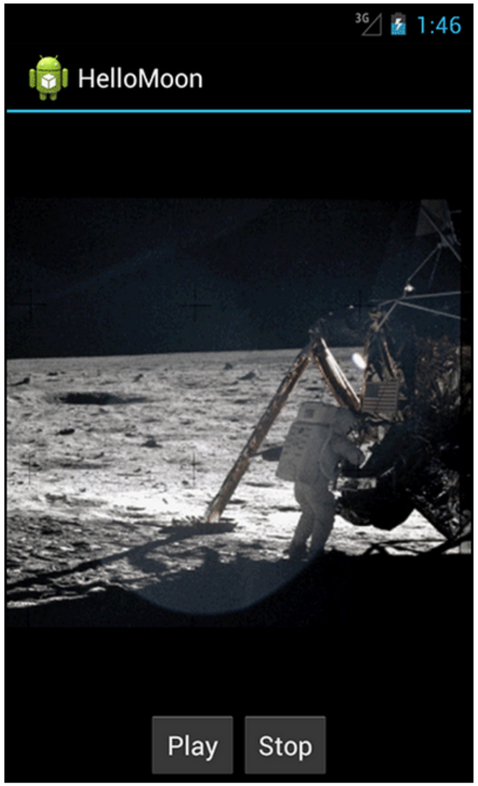
Create AudioPlayer Class to Wrap MediaPlayer

```
public class AudioPlayer {

    private MediaPlayer mPlayer;

    public void stop() {
        if (mPlayer != null) {
            mPlayer.release();
            mPlayer = null;
        }
    }

    public void play(Context c) {
        mPlayer = MediaPlayer.create(c, R.raw.one_small_step);
        mPlayer.start();
    }
}
```



Hook up Play and Stop Buttons

```
public class HelloMoonFragment extends Fragment {
    private AudioPlayer mPlayer = new AudioPlayer();
    private Button mPlayButton;
    private Button mStopButton;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_hello_moon, parent, false);

        mPlayButton = (Button)v.findViewById(R.id.hellomoon_playButton);
        mPlayButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.play(getActivity());
            }
        });

        mStopButton = (Button)v.findViewById(R.id.hellomoon_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                mPlayer.stop();
            }
        });
        return v;
    }
```



Activity Recognition Using Google API

Activity Recognition

- Activity Recognition? Detect what user is doing?
  - Part of user's context
- Examples: sitting, running, driving, walking
- Why? App can adapt it's behavior based on user behavior
- E.g. If user is driving, don't send notifications

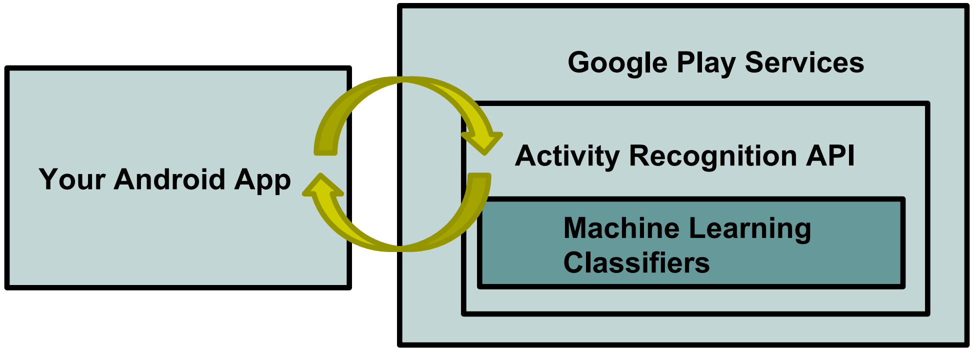
https://www.youtube.com/watch?v=S8sugXgUVEI

Google Activity Recognition API

- API to detect smartphone user's current activity
- Programmable, can be used by your Android app
- Currently detects 6 states:
  - In vehicle
  - On Bicycle
  - On Foot
  - Still
  - Tilting
  - Unknown

Google Activity Recognition API

- Deployed as part of Google Play Services



Your Android App

Google Play Services

Activity Recognition API

Machine Learning Classifiers

Activity Recognition Using Google Fit

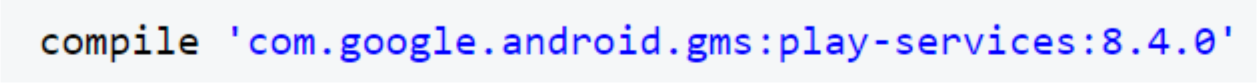
Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

- Example code for this tutorial on gitHub:

  https://github.com/tutsplus/Android-ActivityRecognition

- Google Activity Recognition can:
  - Recognize user’s current activity (Running, walking, in a vehicle or still)

- Project Setup:
  - Create Android Studio project with blank Activity (minimum SDK 14)
  - In **build.gradle** file, define latest Google Play services (8.4) as dependency

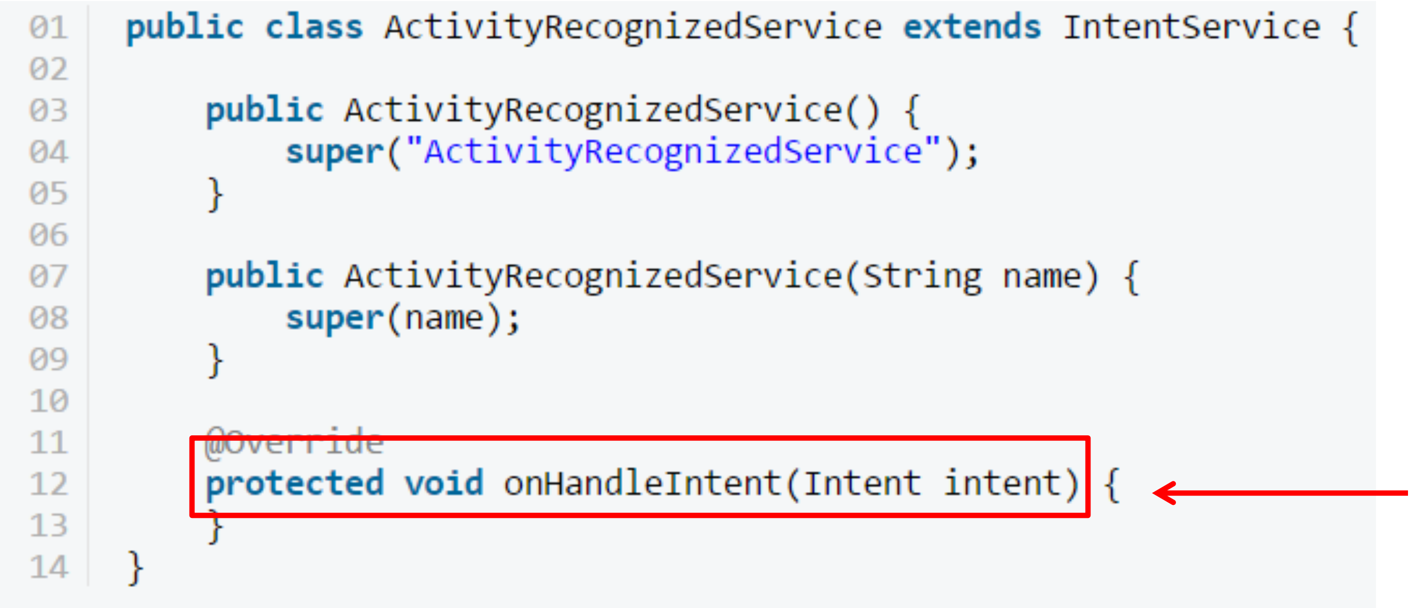


```
compile 'com.google.android.gms:play-services:8.4.0'
```

Activity Recognition Using Google Fit

Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

- Create new class **ActivityRecognizedService** which extends **IntentService**
- **IntentService:** type of service, asynchronously handles work off main thread as Intent requests.
- Throughout user's day, **Activity Recognition API** sends user's activity to this IntentService in the background
- Need to program this Intent to handle incoming user activity



```java
public class ActivityRecognizedService extends IntentService {

    public ActivityRecognizedService() {
        super("ActivityRecognizedService");
    }

    public ActivityRecognizedService(String name) {
        super(name);
    }

    @Override
    protected void onHandleIntent(Intent intent) {
    }
}
```

Called to deliver User's activity

Activity Recognition Using Google Fit

Ref: How to Recognize User Activity with Activity Recognition by Paul Trebilcox-Ruiz on Tutsplus.com tutorials

- Modify **AndroidManifest.xml** to
  - Declare **ActivityRecognizedService**
  - Add com.google.android.gms.permission.ACTIVITY_RECOGNITION permission

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.tutsplus.activityrecognition">

  <uses-permission android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />

  <application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <service android:name=".ActivityRecognizedService" />
  </application>

</manifest>
```

Requesting Activity Recognition

- In **MainActivity.java**, To connect to Google Play Services:
  - Provide **GoogleApiClient** variable type + implement callbacks

```
public class MainActivity extends AppCompatActivity implements GoogleApiClient.ConnectionCallbacks,
GoogleApiClient.OnConnectionFailedListener {

    public GoogleApiClient mApiClient;

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

    @Override
    public void onConnected(@Nullable Bundle bundle) {

    }

    @Override
    public void onConnectionSuspended(int i) {

    }

    @Override
    public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {

    }
}
```

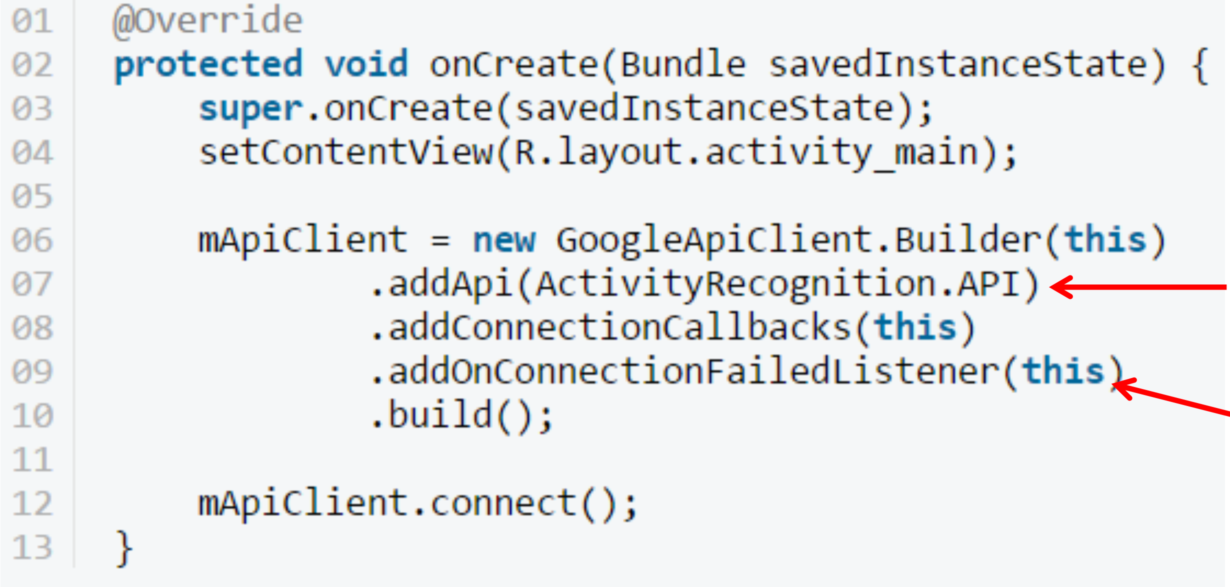
Handle to Google Activity Recognition client

Called if sensor (accelerometer) connection

Called if Google Play connection fails

Requesting Activity Recognition

- In onCreate, initialize client and connect to Google Play Services



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

    mApiClient = new GoogleApiClient.Builder(this)
            .addApi(ActivityRecognition.API)
            .addConnectionCallbacks(this)
            .addOnConnectionFailedListener(this)
            .build();

    mApiClient.connect();
}
```

Request ActivityRecognition.API

Associate listeners with our instance of GoogleApiClient

Requesting Activity Recognition

- Once **GoogleApiClient** has connected, **onConnected( )** is called
- Need to create a **PendingIntent** that goes to our **IntentService**
- Also set how often API shold check user's activity in milliseconds

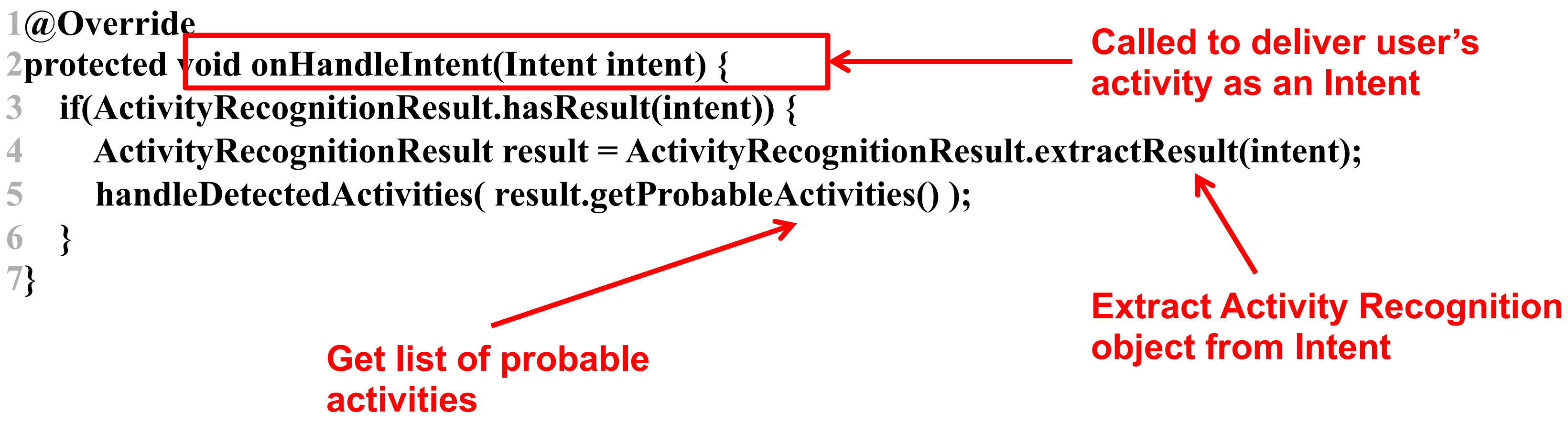
Build intent to send to IntentService

```
@Override
public void onConnected(@Nullable Bundle bundle) {
    Intent intent = new Intent( this, ActivityRecognizedService.class );
    PendingIntent pendingIntent = PendingIntent.getService( this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT );
    ActivityRecognition.ActivityRecognitionApi.requestActivityUpdates( mApiClient, 3000, pendingIntent );
}
```

How often to check user's activity
(in milliseconds)

Handling Activity Recognition

- Our app tries to recognize the user’s activity every 3 seconds
- **onHandleIntent** called every 3 seconds, Intent delivered
- In **onHandleIntent( )** method of **ActivityRecognizedService**
  - Validate that received intent contains activity recognition data
  - If so, extract **ActivityRecognitionResult** from the Intent
  - Retrieve list of possible activities by calling **getProbableActivities( )** on **ActivityRecognitionResult** object



```
@Override
protected void onHandleIntent(Intent intent) {
   if(ActivityRecognitionResult.hasResult(intent)) {
      ActivityRecognitionResult result = ActivityRecognitionResult.extractResult(intent);
      handleDetectedActivities( result.getProbableActivities() );
   }
}
```

Called to deliver user’s activity as an Intent

Extract Activity Recognition object from Intent

Get list of probable activities

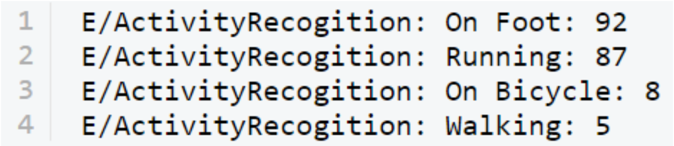
Handling Activity Recognition

- Simply log each detected activity and display how confident Google Play services is that user is performing this activity

```
private void handleDetectedActivities(List<DetectedActivity> probableActivities) {
    for( DetectedActivity activity : probableActivities ) {
        switch( activity.getType() ) {
            case DetectedActivity.IN_VEHICLE: {
                Log.e( "ActivityRecogition", "In Vehicle: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.ON_BICYCLE: {
                Log.e( "ActivityRecogition", "On Bicycle: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.ON_FOOT: {
                Log.e( "ActivityRecogition", "On Foot: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.RUNNING: {
                Log.e( "ActivityRecogition", "Running: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.STILL: {
                Log.e( "ActivityRecogition", "Still: " + activity.getConfidence() );
                break;
            }
            case DetectedActivity.TILTING: {
                Log.e( "ActivityRecogition", "Tilting: " + activity.getConfidence() );
                break;
            }
```

Switch statement on activity type

Sample output



```
E/ActivityRecogition: On Foot: 92
E/ActivityRecogition: Running: 87
E/ActivityRecogition: On Bicycle: 8
E/ActivityRecogition: Walking: 5
```

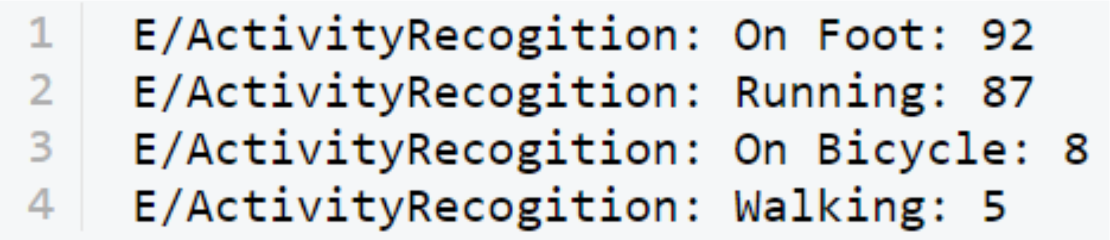
Handling Activity Recognition

- If confidence is > 75, activity detection is probably accurate
- If user is walking, ask “Are you walking?”

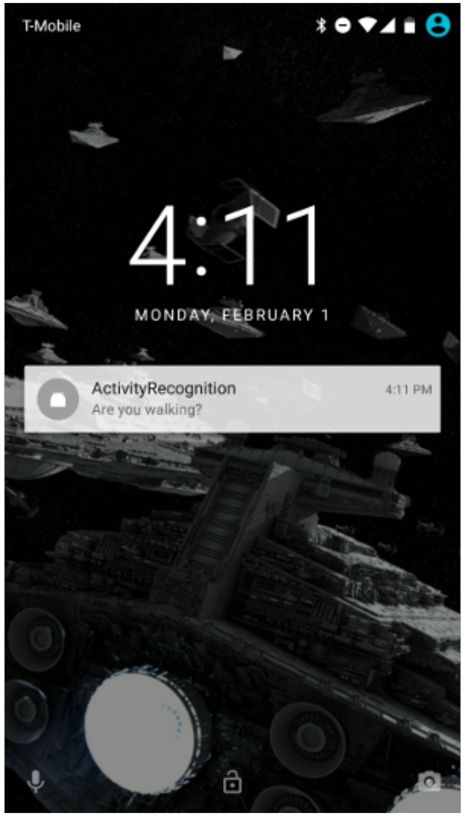
```
            case DetectedActivity.WALKING: {
                Log.e( "ActivityRecogition", "Walking: " + activity.getConfidence() );
                if( activity.getConfidence() >= 75 ) {
                    NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
                    builder.setContentText( "Are you walking?" );
                    builder.setSmallIcon( R.mipmap.ic_launcher );
                    builder.setContentTitle( getString( R.string.app_name ) );
                    NotificationManagerCompat.from(this).notify(0, builder.build());
                }
                break;
            }
            case DetectedActivity.UNKNOWN: {
                Log.e( "ActivityRecogition", "Unknown: " + activity.getConfidence() );
                break;
            }
        }
    }
}
```

Sample Output of Program

- Sample displayed on development console



```
E/ActivityRecognition: On Foot: 92
E/ActivityRecognition: Running: 87
E/ActivityRecognition: On Bicycle: 8
E/ActivityRecognition: Walking: 5
```



T-Mobile

4:11

MONDAY, FEBRUARY 1

ActivityRecognition 4:11 PM
Are you walking?

- Full code at: https://github.com/tutsplus/Android-ActivityRecognition