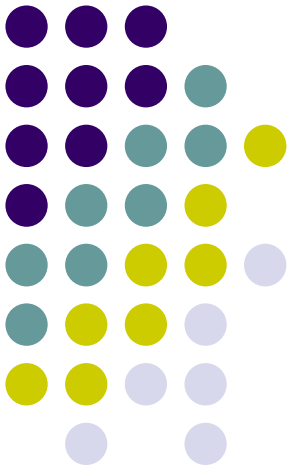
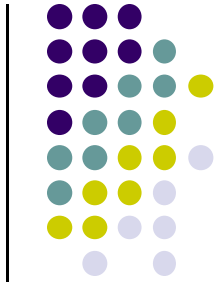
# CS 403X Mobile and Ubiquitous Computing

## Lecture 11: Tracking Location, Using Maps, Playing Back Sound and Video

**Emmanuel Agu**



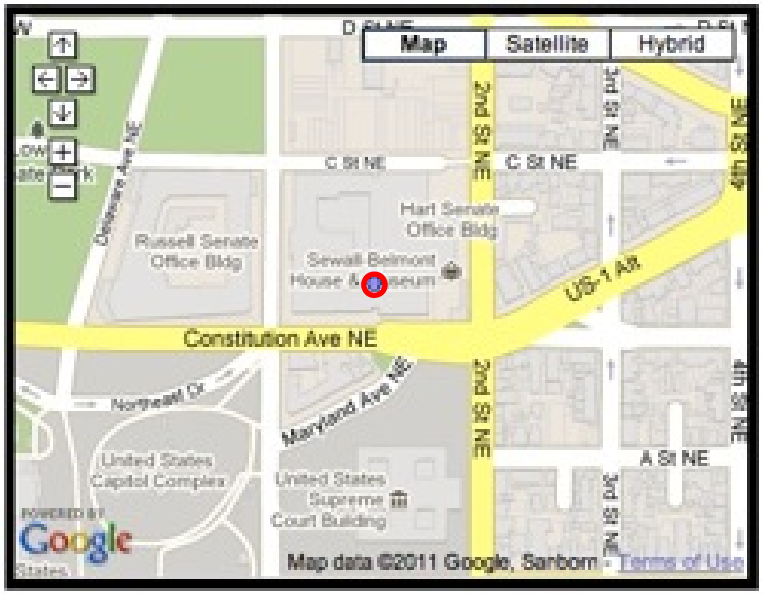


# Tracking the Device's Location

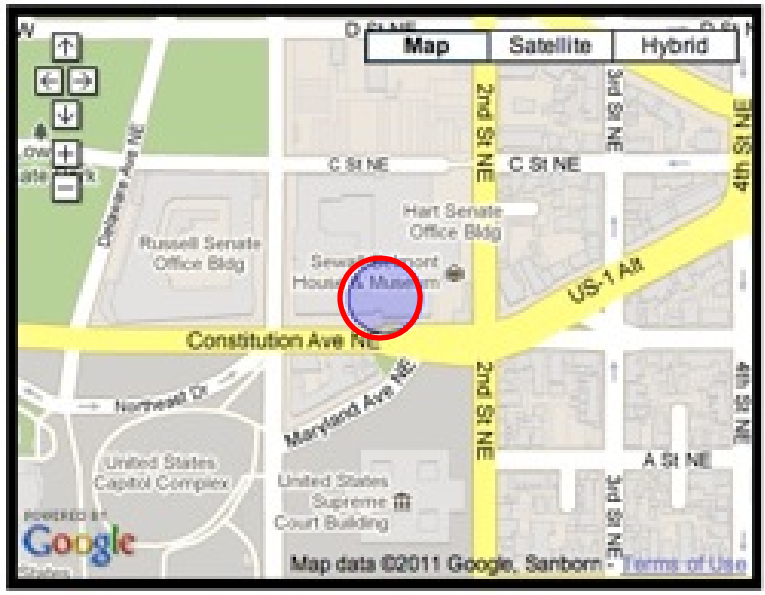
# Location Tracking

- **Outdoors:** Uses GPS (More accurate)
- **Indoors:** WiFi signals (called Assisted GPS, less accurate)

GPS

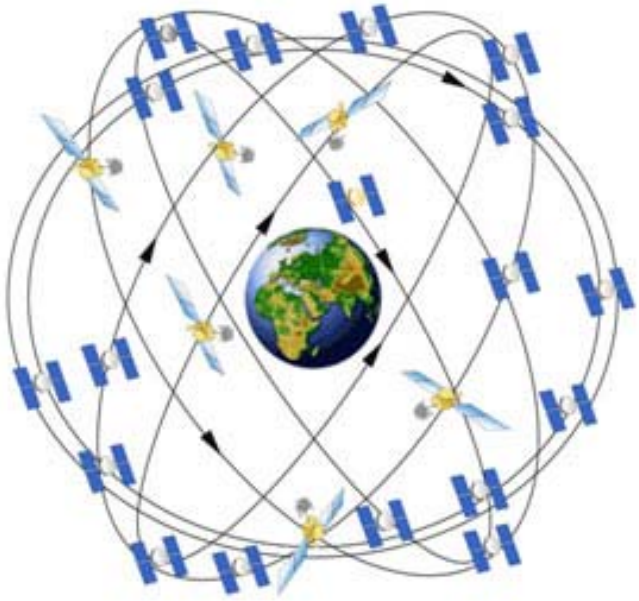



Wi-Fi



# Global Positioning System (GPS)

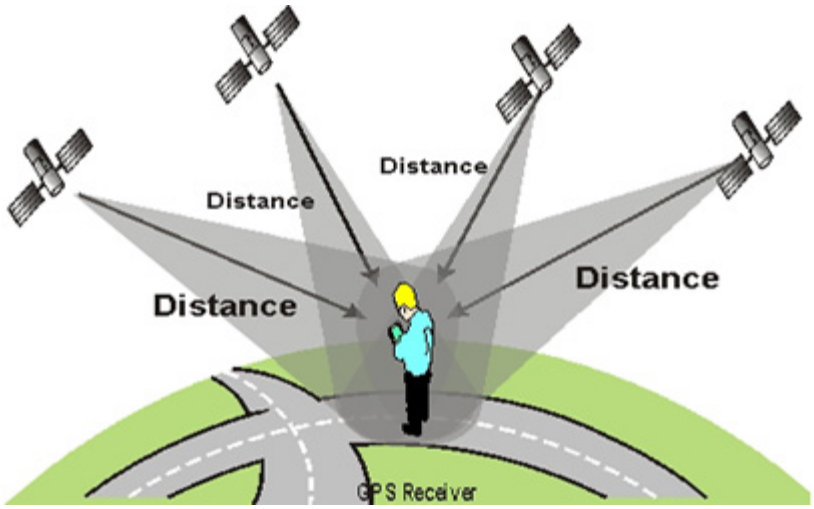
- 24 core satellites
- **20,000 km above earth** (Medium earth orbit)
- 6 orbital planes with 4 satellites each
- 4 satellites visible from any spot on earth
- Recently upgraded to 27 sats



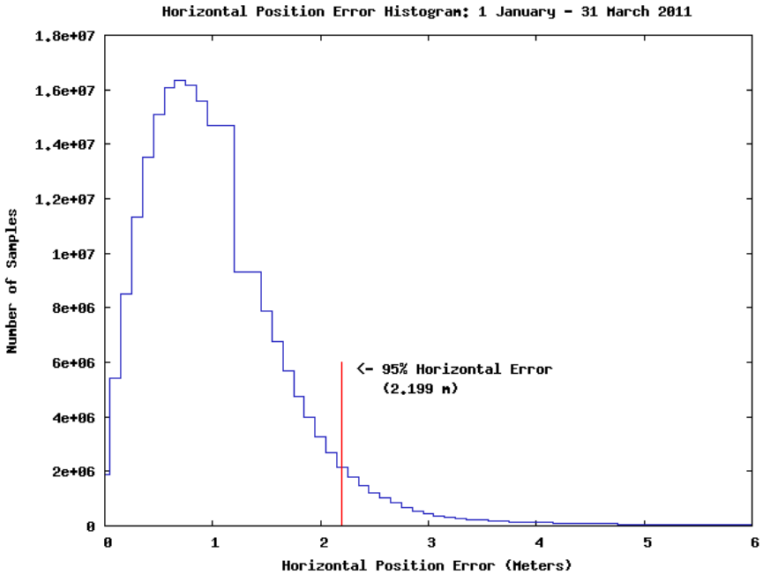
# GPS User Segment

- GPS receiver calculates user's position and travel path by comparing time signals from multiple satellites based on known positions of those satellites (called triangulation)

- Accuracy within 5 - 10 meters

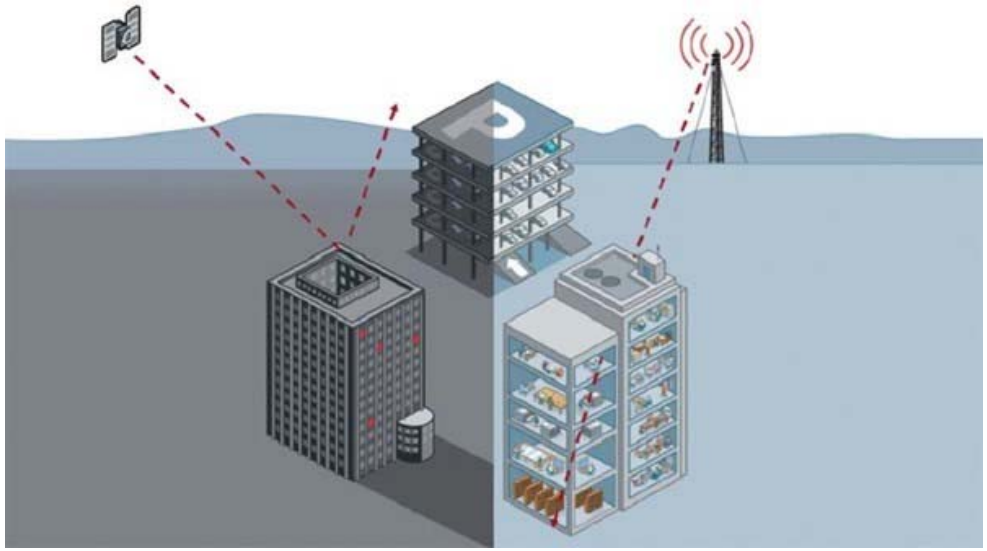


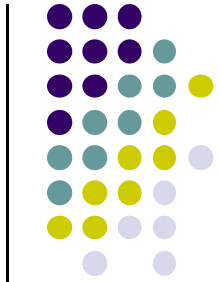

http://adamswalk.com/gpx-2/



# Determining User Location

- GPS most accurate but
  - Only works OUTDOORS (signals don't penetrate buildings)
  - Drains battery power
  - **Lag/delay** in acquiring satellites or re- acquiring if lost
- **Alternative:** Use Wi-Fi indoors
- Maps device's locations based on combination of wi-fi access points (known location) seen
- Also called **location fingerprinting**

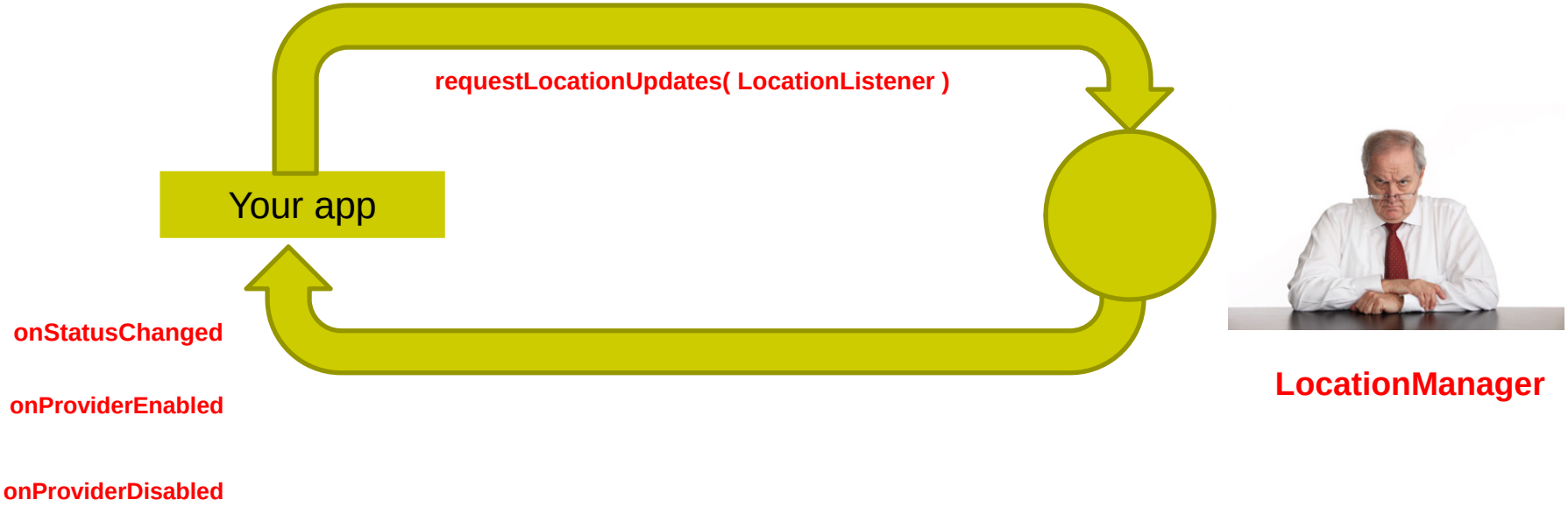




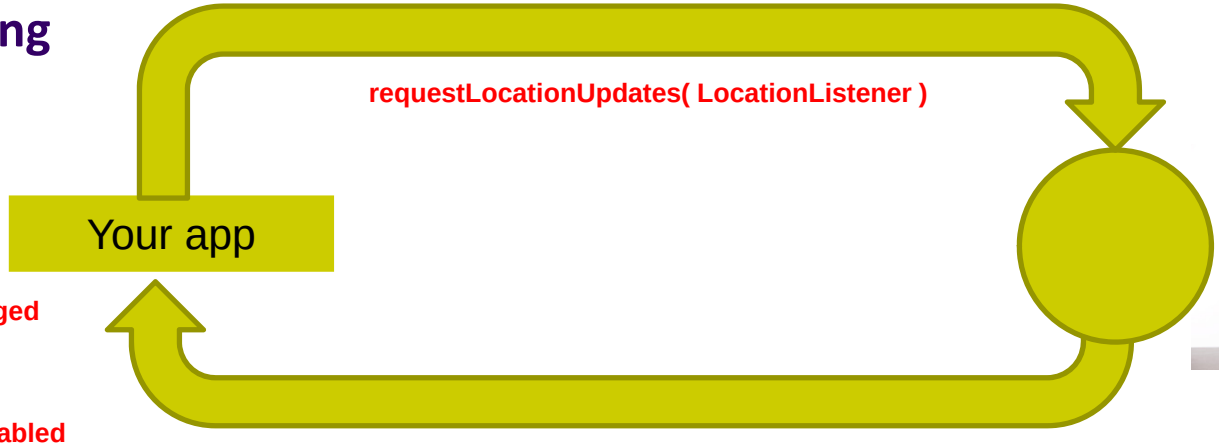
# Determing and Using Location in Android Apps

# Google Location APIs

- Location API is part of Google Play Services (newer! Recommended)
- Older Android framework location APIs (**android.location**)
  - Used by most books, online sources. We will use that
  - http://developer.android.com/guide/topics/location/strategies.html
- Requesting Location Updates



# Requesting Location Updates




LocationManager

```
// 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);
```

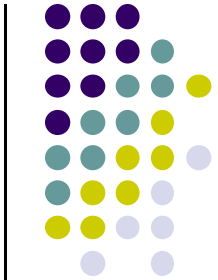
Create listener for Location info

Callback methods called by Location manager (e.g. when location changes))

Type of location Provider (e.g. cell tower and Wi-Fi based)

Listener that receives callbacks

# Requesting User Permissions

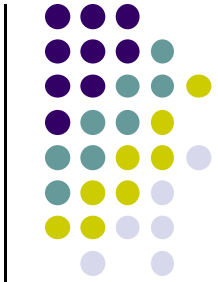


- To get smartphone owner's permission to use their GPS

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

- **ACCESS_FINE_LOCATION:** GPS
- **ACCESS_COARSE_LOCATION:** WiFi or cell towers

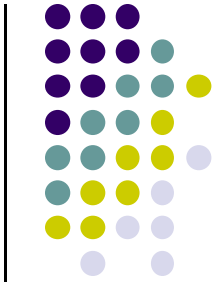
# Getting Cached Copy of Location (Fast)



- Getting current location may take a while
- Can choose to use location cached at Location Manager

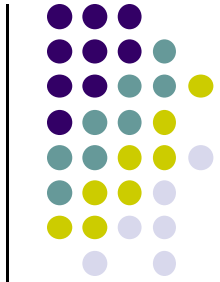
```
String locationProvider = LocationManager.NETWORK_PROVIDER;
// Or use LocationManager.GPS_PROVIDER

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

# Stopping Listening for Location Updates



- Location updates consume battery power
- Stop listening for location updates whenever you no longer need

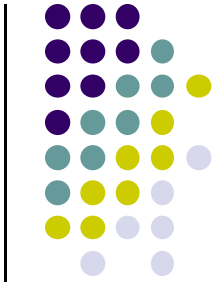
```
// Remove the listener you previously added
locationManager.removeUpdates(locationListener);
```



# Services and Location
# Example from Head First Android

# Example: Odometer (Distance Travelled) updates as a Services

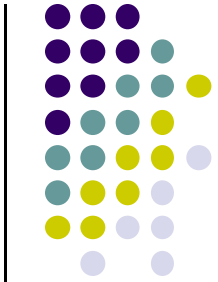
## (Ref: Head First Android pg 541)



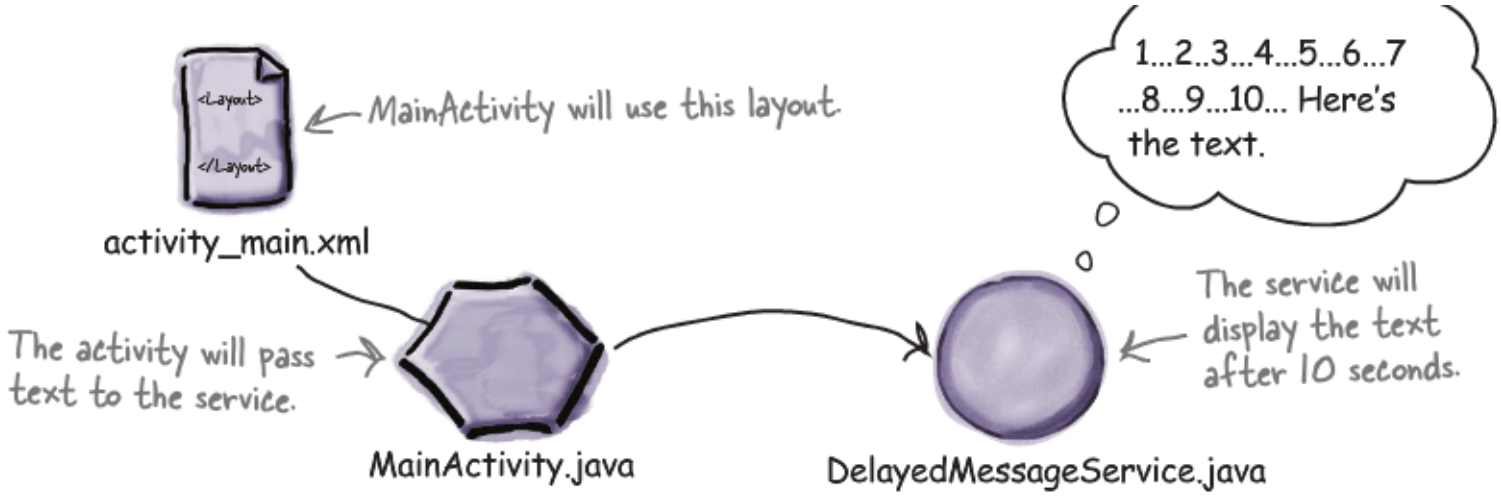
- **Services:** long running background processes, no UI
- May want background service (a module in our app) to continuously retrieve location updates from LocationManager, forward our Activity updates
- Ref: Head First Android pg 541
  - Example of using a Service
  - Nice Example app using Odometer Service

# Example Service App

## (Ref: Head First Android pg 541)

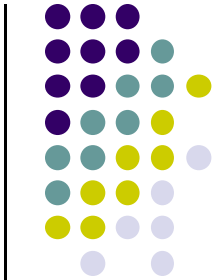


- App has:
  - MainActivity
  - DelayedMessageService
- **MainActivity:** calls DelayedMessage Service, passes text
- **Delayed Service:** waits 10 seconds, displays text

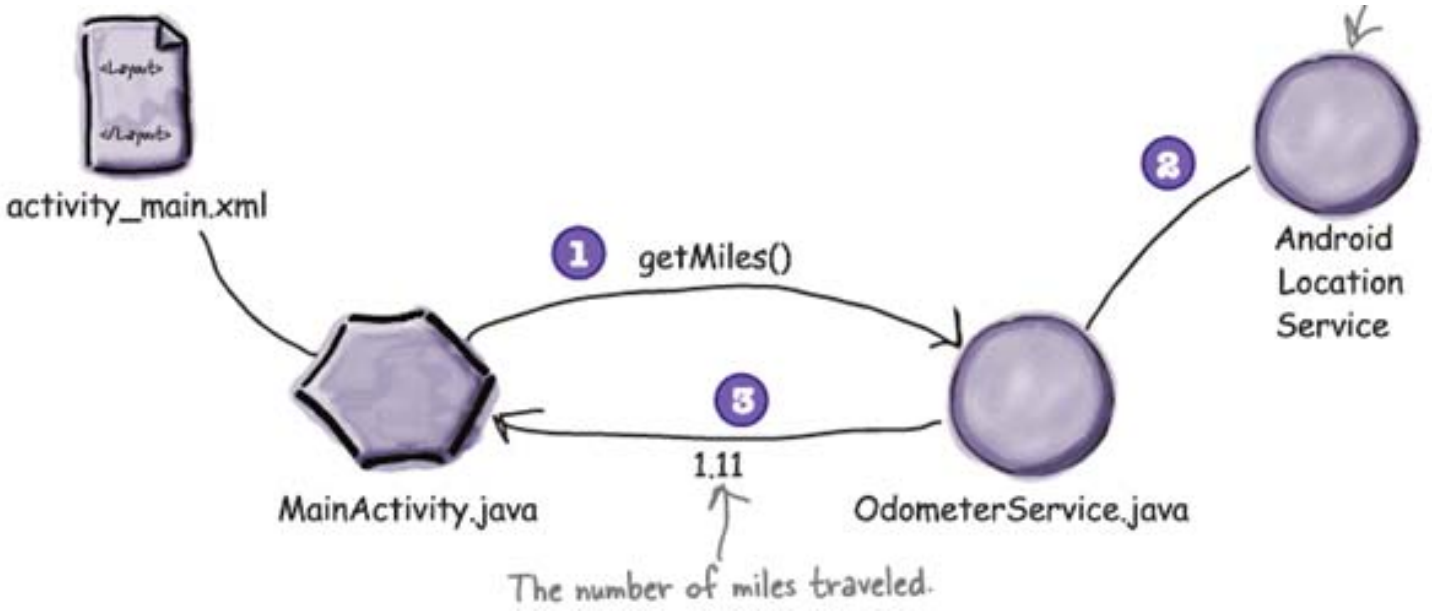


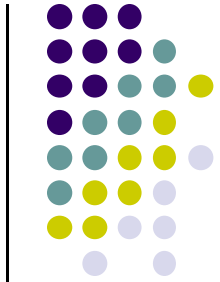
# Example: Odometer (Distance Travelled) updates as a Services

## (Ref: Head First Android pg 541)



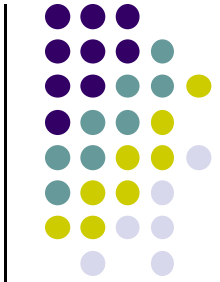
- Example odometer app that tracks distance travelled



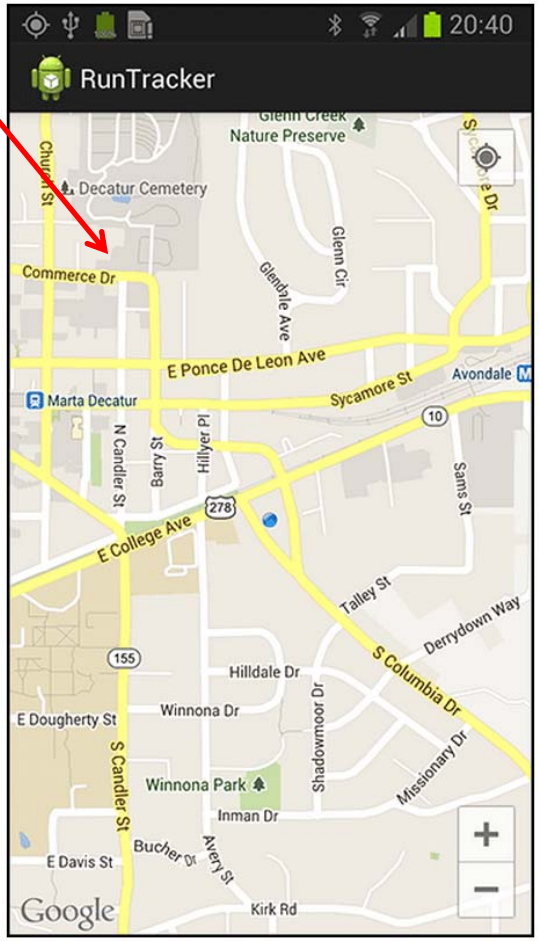


# Using Maps

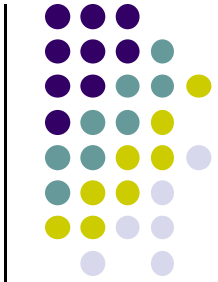
# Introducing MapView and Map Activity



- **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

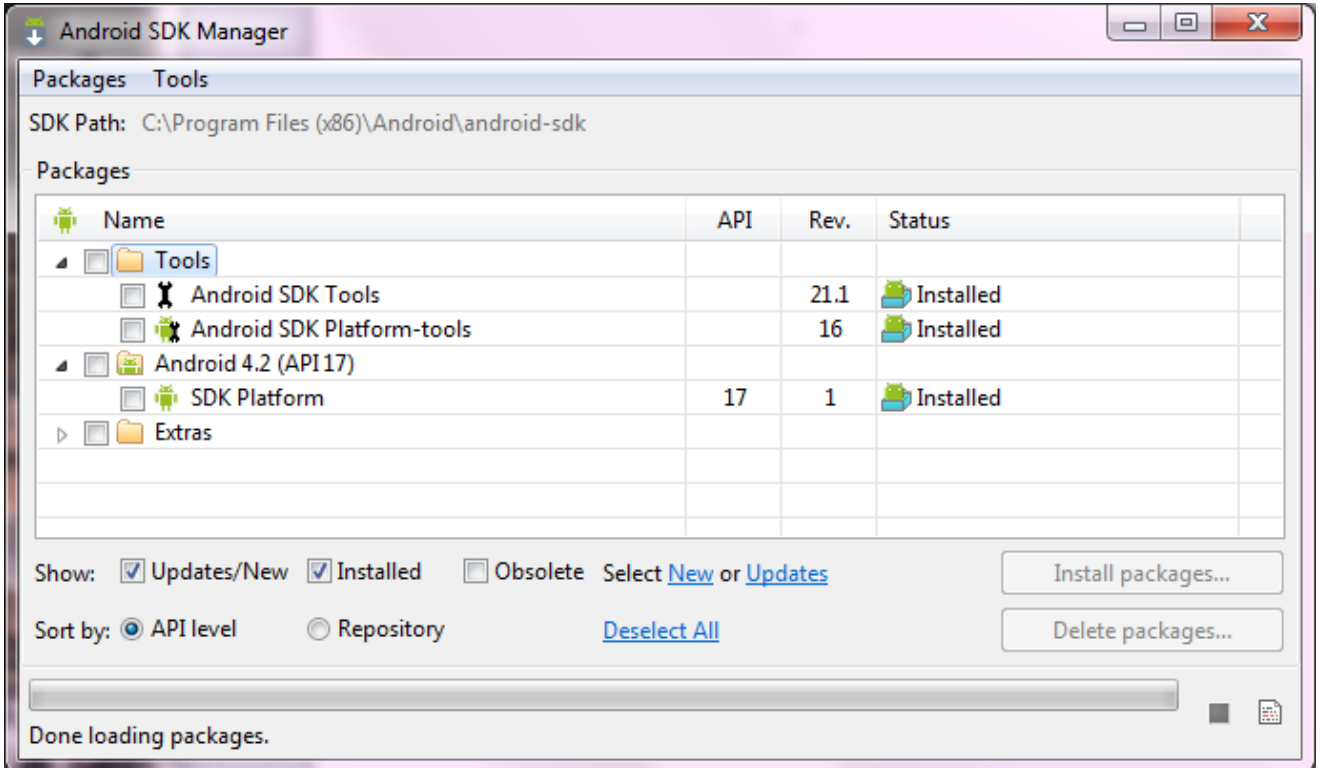


1. Install Android SDK (Done already!)
2. Use Android Studio SDK manager to add Google Play services
3. Obtain Google Maps API key
4. Add required settings (permissions, etc) to Android Manifest
5. Add a  map to app

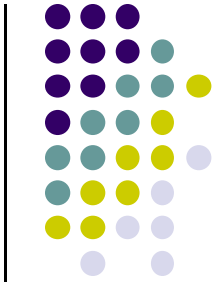
# Step 2: Add Google Play Services to Your Project

- 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



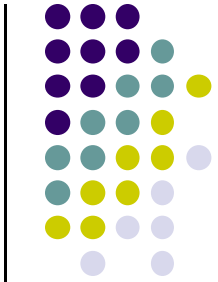
# Step 2: Add Google Play Services to Your Project



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

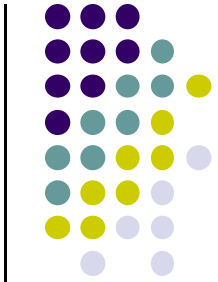
```
apply plugin: 'com.android.application'
    ...

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

# Step 3: Get Google Maps API key



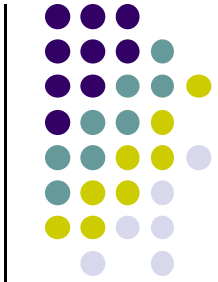
- To access Google Maps servers using Maps API, must add Maps API key to app
- Maps API key is free
- **Background:** Before they can be installed, android apps must be signed with digital certificate (developer holds private key)
- Digital certificates uniquely identify an app, used in tracking:
  - 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
```

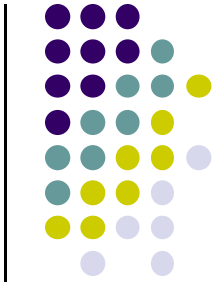
- Add this API key to app in order to use Maps API
- Include API key in AndroidManifest.xml
- To modify AndroidManifest.xml, add following between <application> … </application>

```
<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

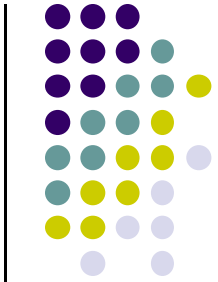
Used by API to cache map tile data in device's external storage

```
<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)

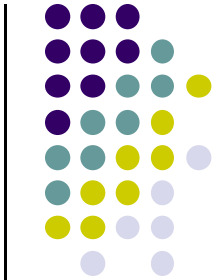


- Specify that OpenGL ES version 2 is required
- Why? Google Maps Android API uses OpenGL ES version 2 to render the map

```
<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

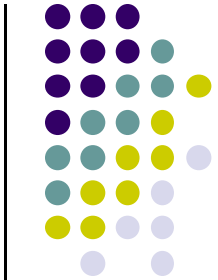
# Step 5: Add a map



- To add a map, create XML layout file

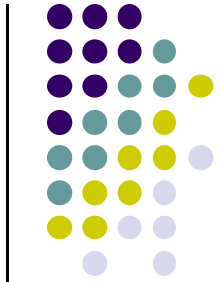
```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:name="com.google.android.gms.maps.MapFragment"/>
```

# Install & Configure Google Play Services SDK



- And create MainActivity.java

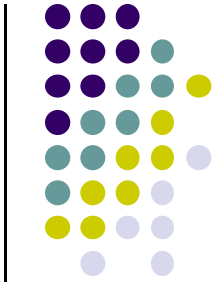
```
package com.example.mapdemo;

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

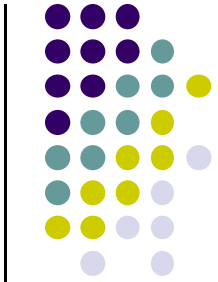
# Media Playback

**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

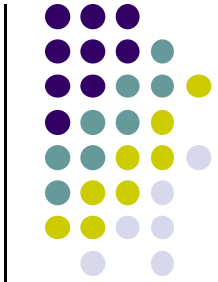
# Media Player: Manifest Declarations



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

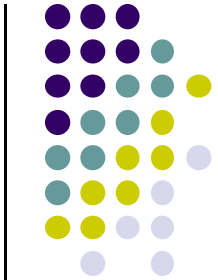
```
<uses-permission android:name="android.permission.INTERNET" />
```

# 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

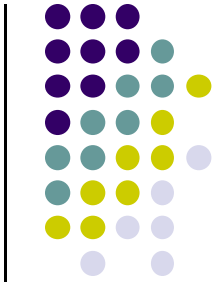
```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you
```

- 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();
```

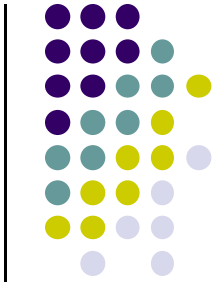
# Releasing the MediaPlayer



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

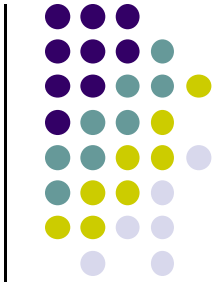
```
mediaPlayer.release();
mediaPlayer = null;
```

- 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

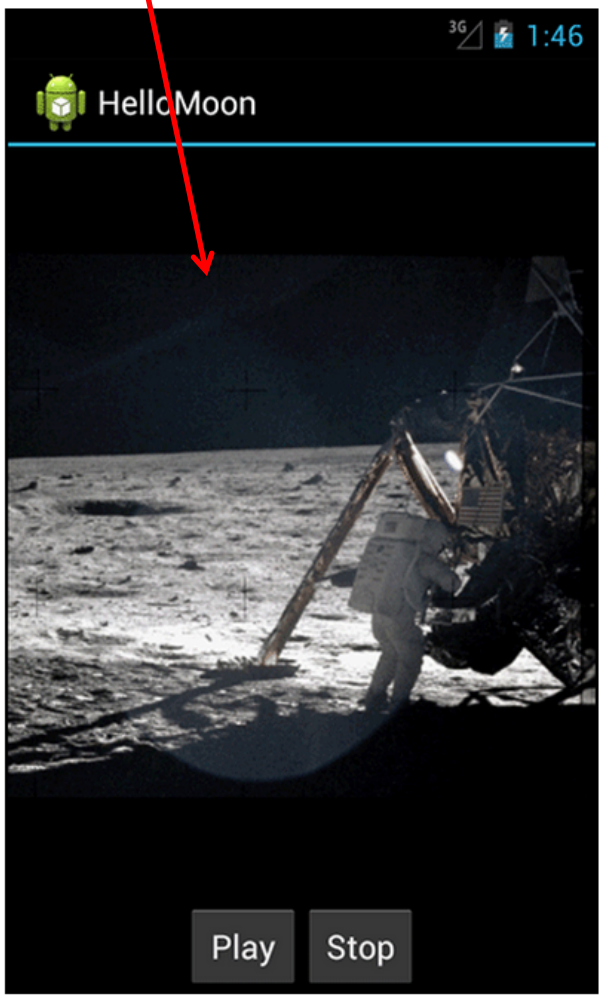


- Example creates **HelloMoon app** that uses **MediaPlayer** to play audio file
- Android Class for audio and video playback
- **Source:** Can play local files, or streamed over Internet
- **Supported formats:** WAV, MP3, Ogg, Vorbis, MPEG-4, 3GPP, etc

# 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>
```

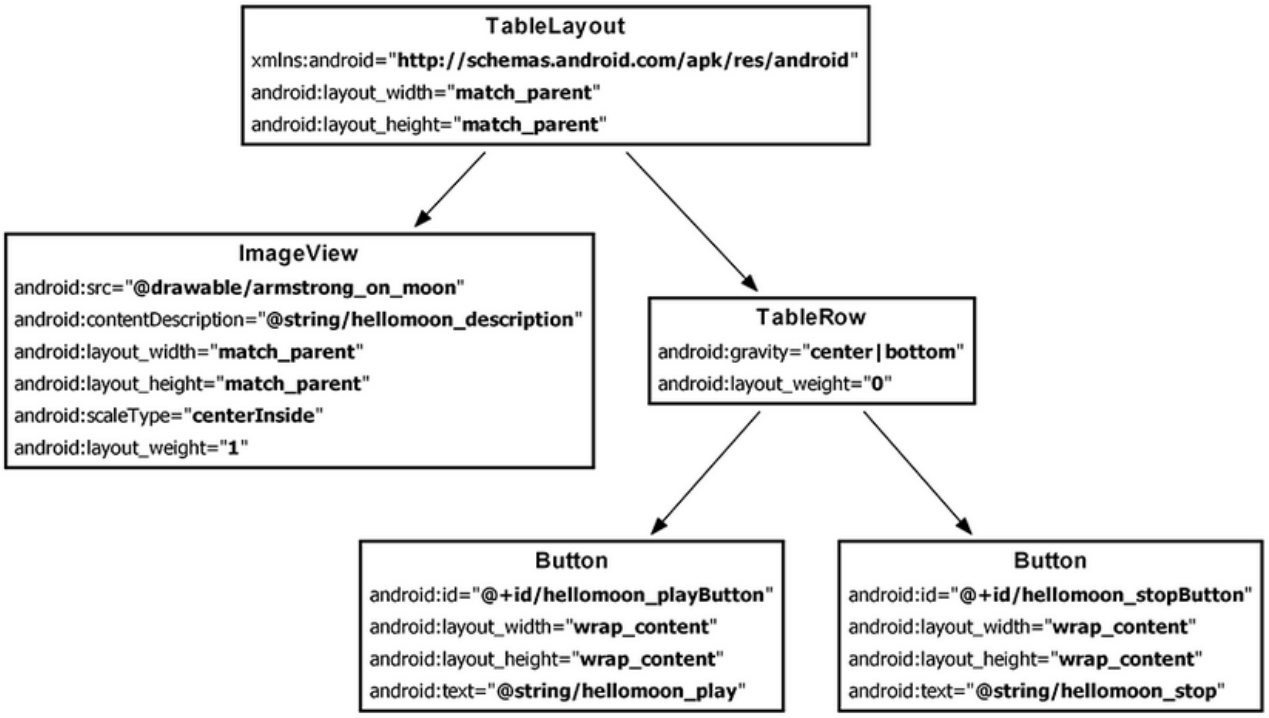


# HelloMoon App

- 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

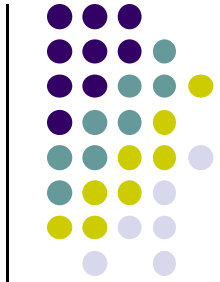



# Creating a Layout Fragment

- 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**

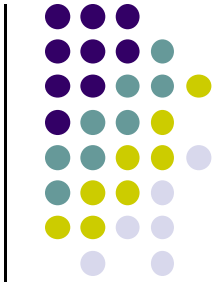
```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/helloMoonFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">

</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

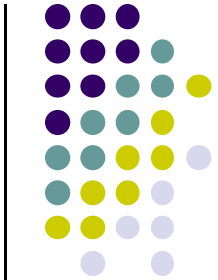


```
public class AudioPlayer {

    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;
    }
```

# References




- Paul A Zandbergen, Accuracy of iPhone Locations: A Comparison of Assisted GPS, Transactions in GIS, 2009, 13(s1): 5–26
- Head First Android
- Android Nerd Ranch, 2nd edition
- Busy Coder's guide to Android version 6.3
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014