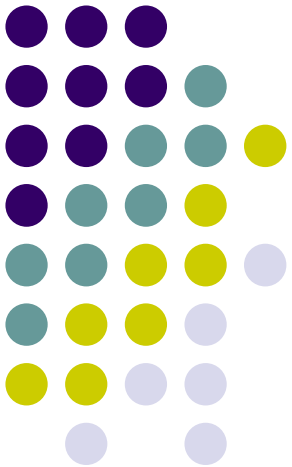
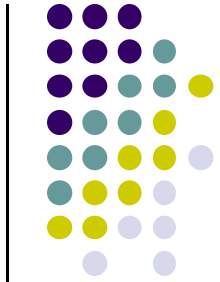
CS 528 Mobile and Ubiquitous Computing

Lecture 6: Tracking Location, Maps, Services, Audio/Video Playback, Presentation/Critique Guidelines

Emmanuel Agu



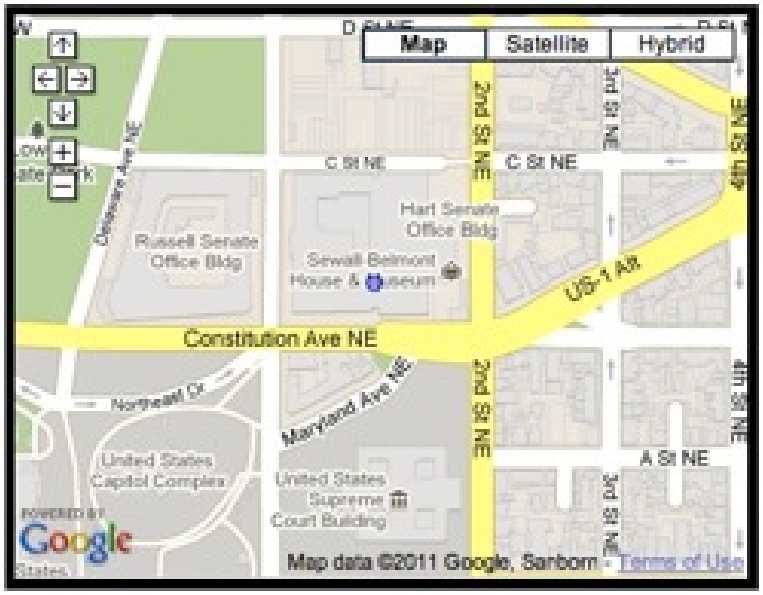


Tracking the Device's Location

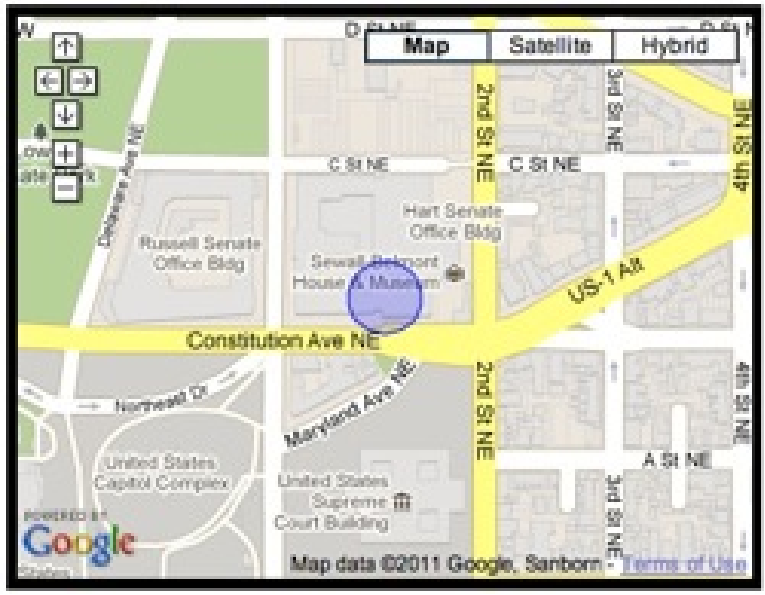
Location Tracking

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

GPS

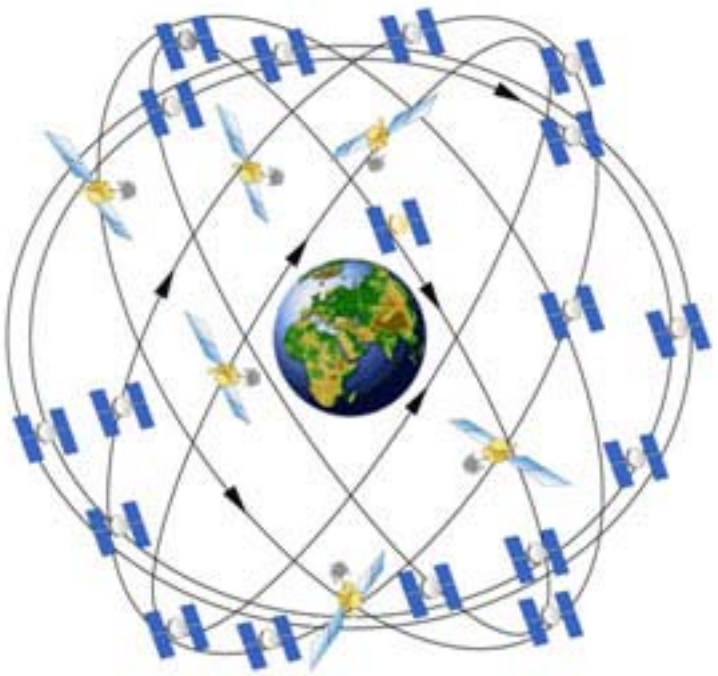


Wi-Fi



Global Positioning System (GPS)

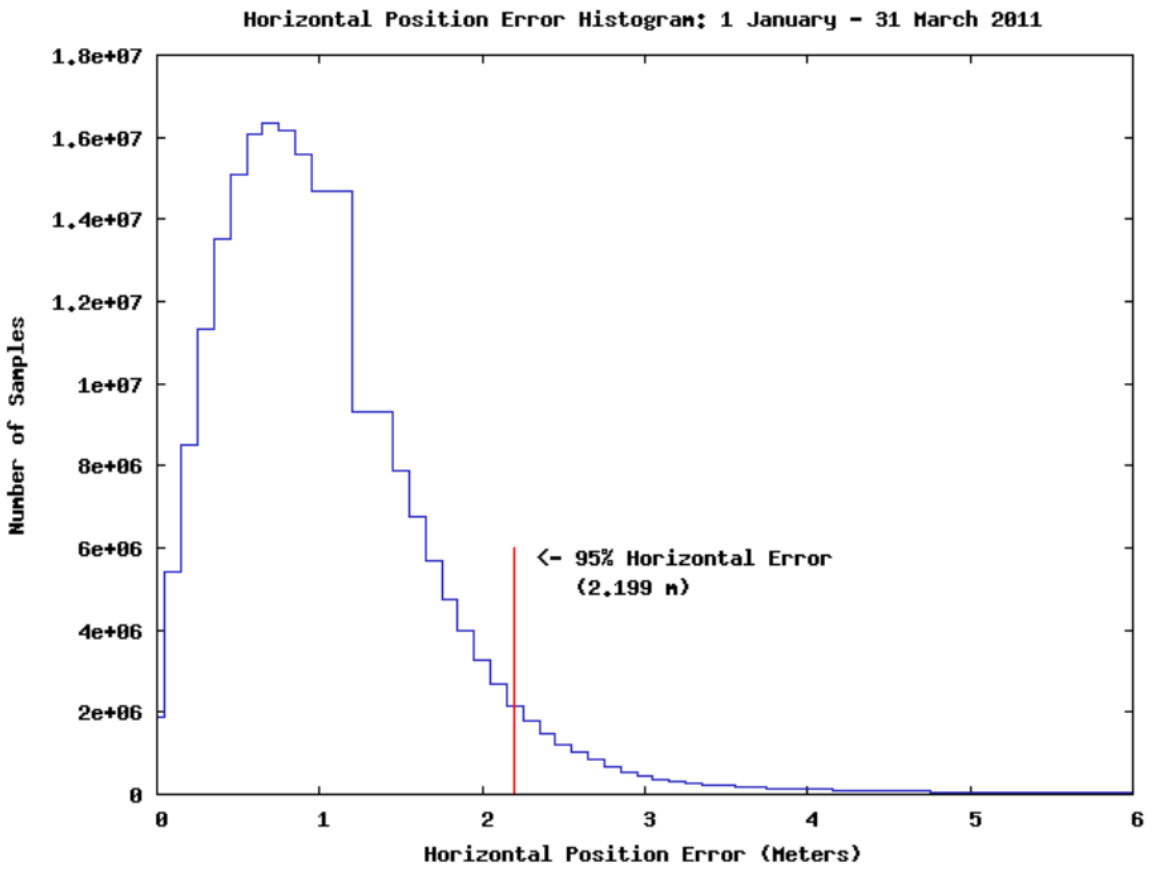
- 24 core satellites
- medium earth orbit, 20,000 km above earth
- 6 orbital planes with 4 satellites each
- 4 satellites visible from any spot on earth



- Recently upgraded to 27 sats

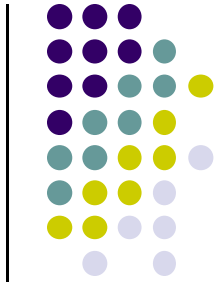
GPS User Segment

- Receiver calculates position and course by comparing time signals from multiple satellites based on known positions of those satellites
- Accuracy normally within 5 - 10 meters



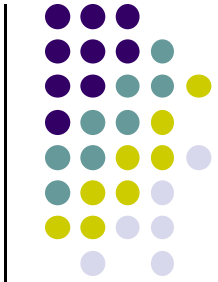
Android and Location

- Obtaining User Location
- GPS most accurate but
  - only works OUTDOORS
  - quickly consumes battery power
  - delay in acquiring satellites or re-acquiring if lost
- Can use Wi-Fi indoors
- Maps device's locations based on combination of wi-fi access points (known location) seen
- Also called **location fingerprinting**



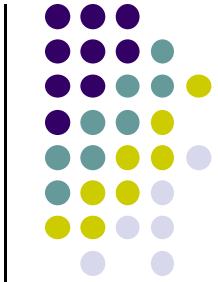
Services and Location
Example from Head First Android

Services (Ref: Head First Android pg 541)

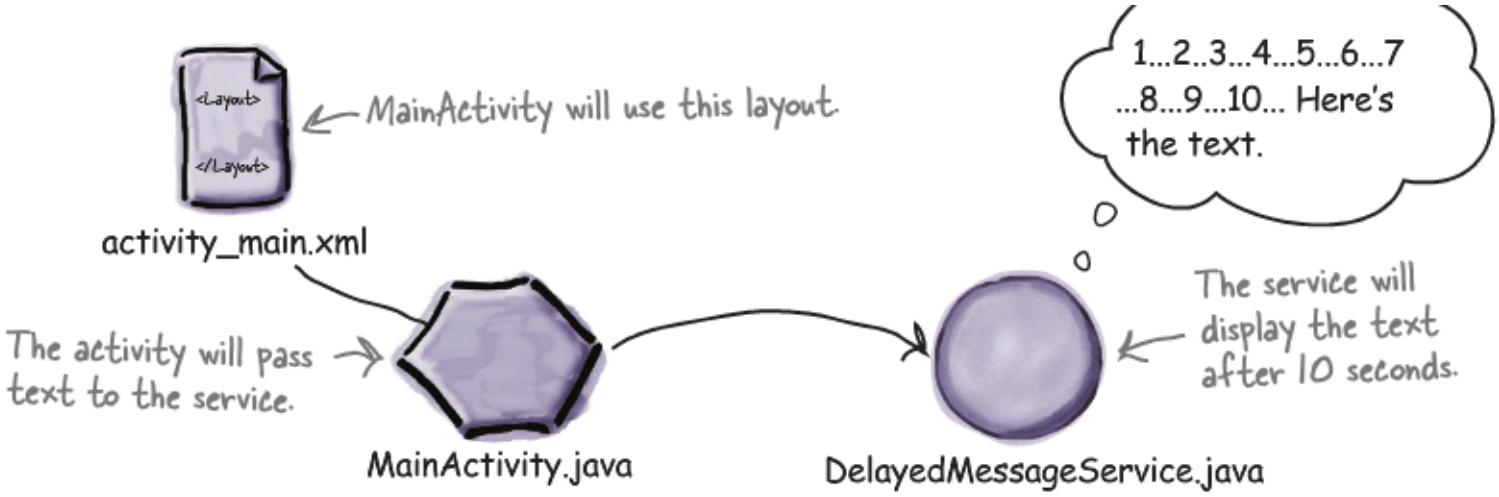


- **Services:** long running background processes, no UI
- Example uses:
  - Download a large file
  - Stream music
- Two types of services
  - **Started services:** Not tied to any activity, runs in background indefinitely, even after parent activity exits
  - **Bound services:** Exits when parent activity exits. Parent activity can interact with it, send requests, get requests

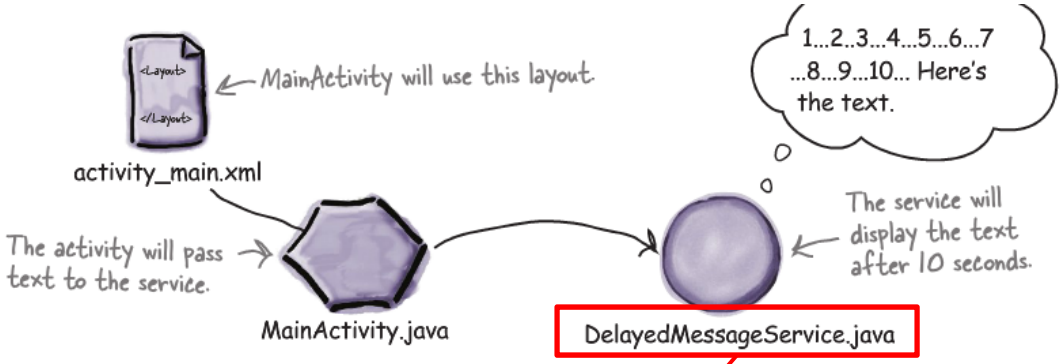
Example Service App



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



Example Service App



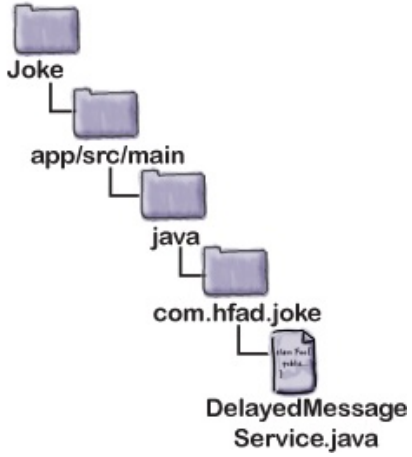
- Create **IntentService**, subclass of Service, can handle Intents

```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;

public class DelayedMessageService extends IntentService {

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

    @Override
    protected void onHandleIntent(Intent intent) {
        //Code to do something
    }
}
```

Extend the IntentService class.

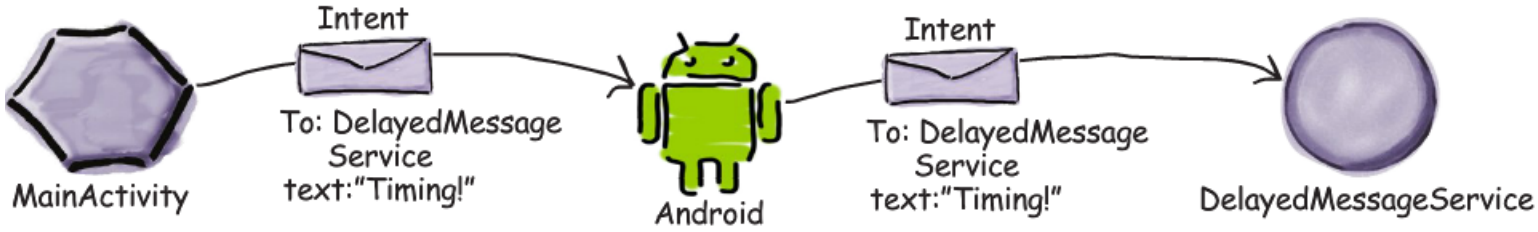
Put the code you want the service to run in the onHandleIntent() method.

Message to be displayed
Will be passed as Intent
So declare IntentService



Big Picture

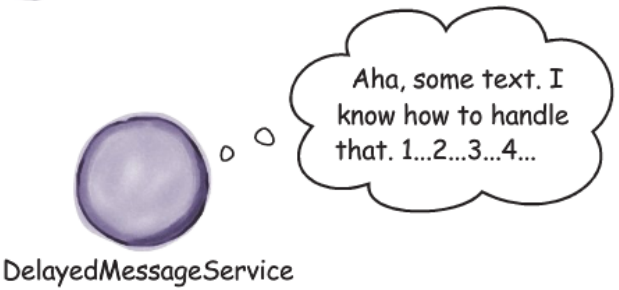
- Activity creates explicit intent, passes text “Timing” to Service



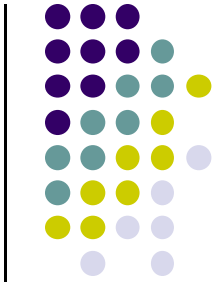
- IntentService **onHandleIntent( )** method is called,

```
public class DelayedMessageService extends IntentService {

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

    @Override
    protected void onHandleIntent(Intent intent) {
        //Code to do something
    }
}
```

Put the code you want the service to run in the onHandleIntent() method.



Full DelayedMessageService Code



```
package com.hfad.joke;

import android.app.IntentService;
import android.content.Intent;
import android.util.Log;

public class DelayedMessageService extends IntentService {

    public static final String EXTRA_MESSAGE = "message";

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

    @Override
    protected void onHandleIntent(Intent intent) {
        synchronized (this) {
            try {
                wait(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        String text = intent.getStringExtra(EXTRA_MESSAGE);
        showText(text);
    }

    private void showText(final String text) {
        Log.v("DelayedMessageService", "The message is: " + text);
    }
}
```

Extend the IntentService class.

Use a constant to pass a message from the activity to the service.

Call the super constructor.

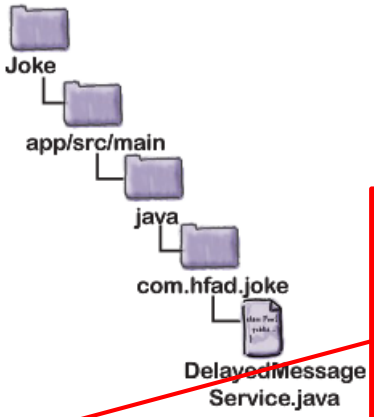
This method contains the code you want to run when the service receives an intent.

Wait 10 seconds.

Get the text from the intent

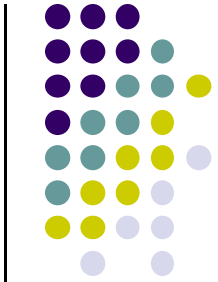
Call the showText() method.

This logs a piece of text so we can see it in the logcat through Android Studio.



- showText( ) method displays text from intent as log message

Declare Services in AndroidManifest.xml



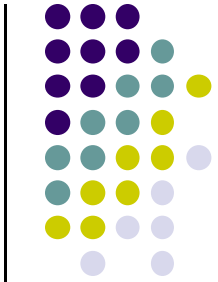
- Service declaration has:
  - **android:name:** name of service (e.g. DelayedMessageService)
  - **android:exported:** Can other apps use this service? (True/false)

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

        <activity
            ...
        </activity>

        <service
            android:name=".DelayedMessageService"
            android:exported="false" >
        </service>
    </application>
</manifest>
```

You declare a service in AndroidManifest.xml like this. Android Studio should do this for you automatically.

The service name has a . in front of it so that Android can combine it with the package name to derive the fully qualified class name.

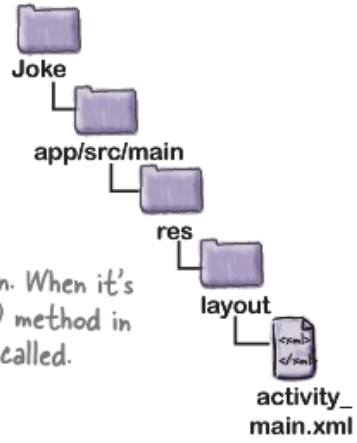
Add Button to activity_main.xml



- **Would like:** to start DelayedMessageService by clicking button on MainActivity
- Add code for this button

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_text"
        android:id="@+id/button"
        android:onClick="onClick"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```

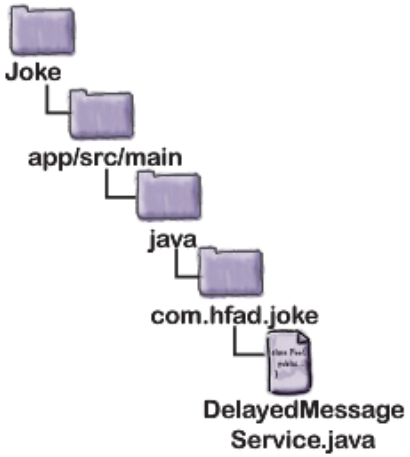
This creates a button. When it's clicked, the onClick() method in the activity will get called.



In Activity, Start Service using StartService( )

```
package com.hfad.joke;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends Activity {

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

    public void onClick(View view) {
        Intent intent = new Intent(this, DelayedMessageService.class);
        intent.putExtra(DelayedMessageService.EXTRA_MESSAGE,
                        getResources().getString(R.string.button_response));
        startService(intent);
    }
}
```

We're using these classes.

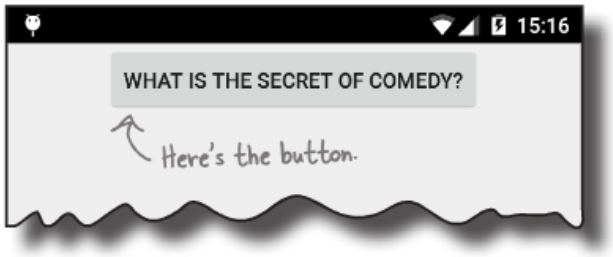


This will run when the button gets clicked.

Create the intent.

Add text to the intent.

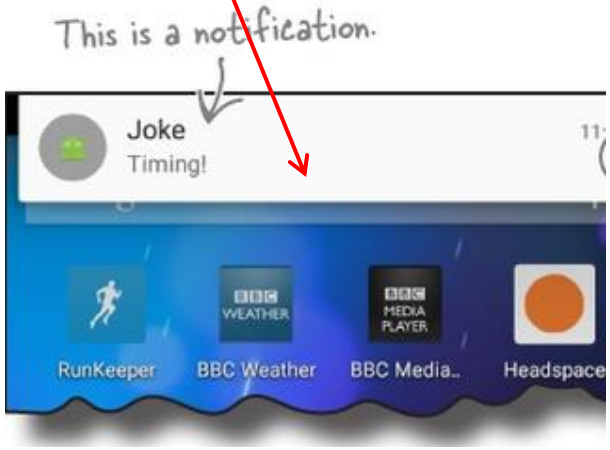
Start the service.



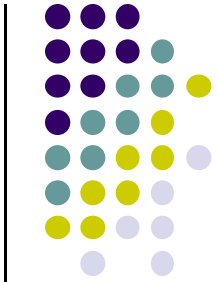
Contains text “Timing”

More in Book

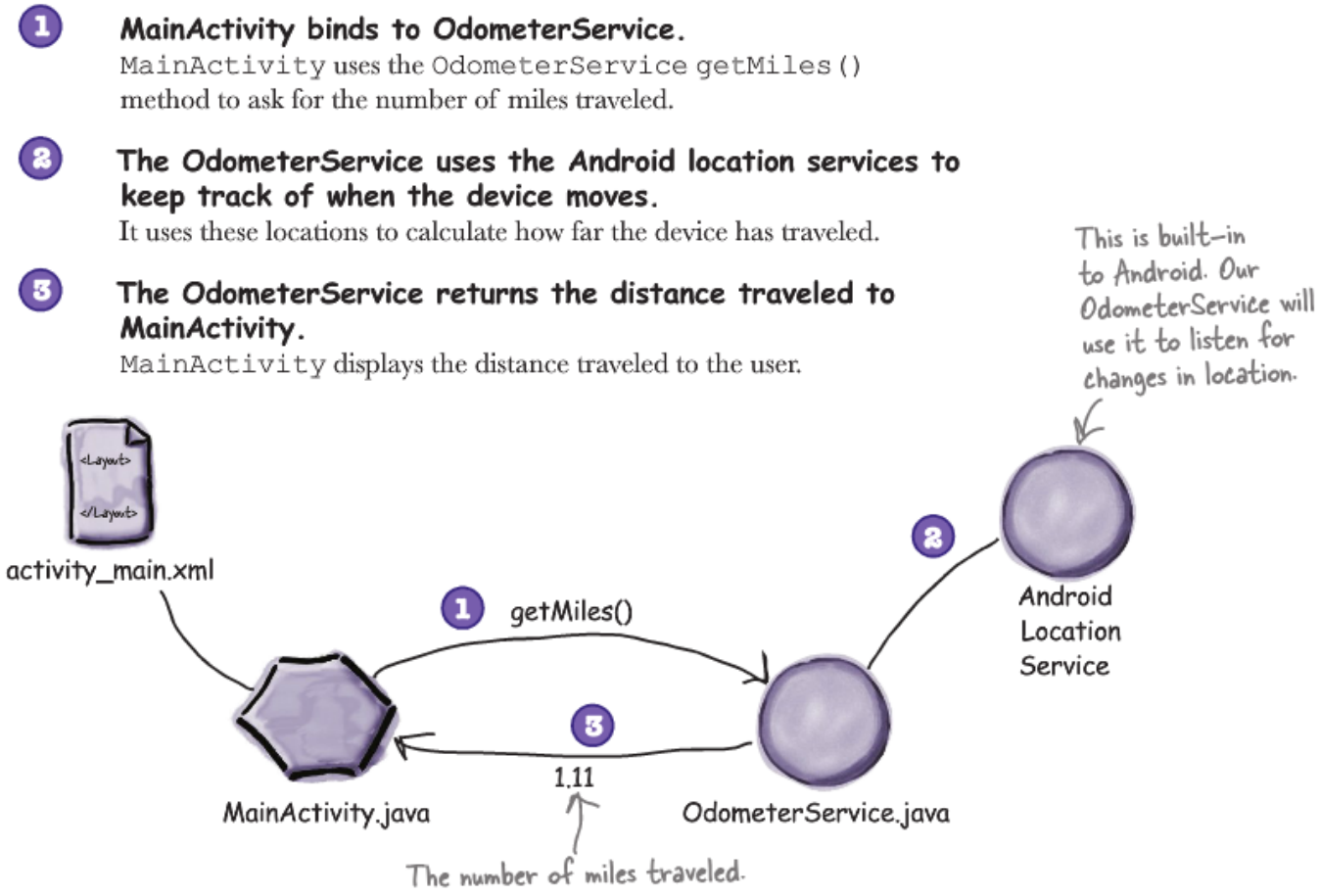
- Book also contains how to:
  - Display delayed message as **toast** or as **notification** to use



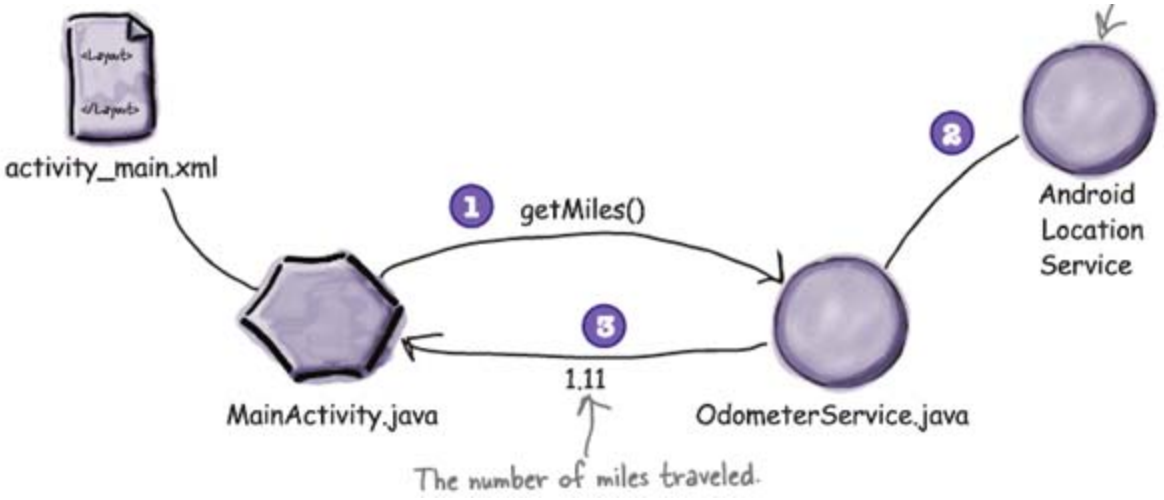
Bound Services are More Interactive



- **Bound services:** Created by activity, when parent activity exits.
  - Parent activity can interact with it, send requests, get requests
- Next, create odometer app that tracks distance travelled

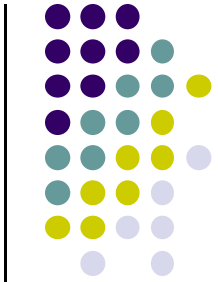


Steps to Create OdometerService



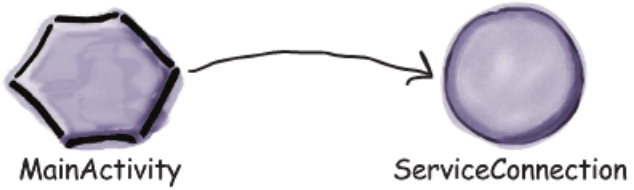
- 3 parts of OdometerService solution
  - Define OdometerBinder (to attach activity to Service)
  - Create LocationListener, register it
  - Create public getMiles( ) method

How Binding Works



1 **The activity creates a ServiceConnection object.**

A `ServiceConnection` is used to form a connection with the service.

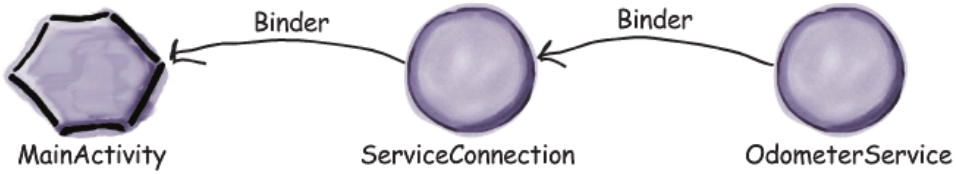


2 **The activity passes an Intent down the connection to the service.**

The intent contains any additional information the activity needs to pass to the service.

3 **The bound service creates a Binder object.**

The Binder contains a reference to the bound service. The service sends the Binder back along the connection.



4 **When the activity receives the Binder, it takes out the Service object and starts to use the service directly.**

How it Works

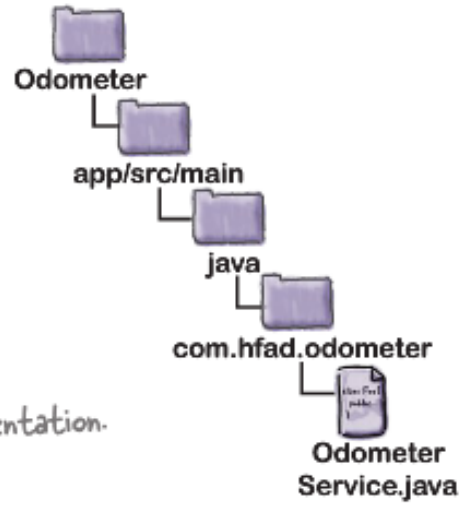
- Activity calls the Service's **onBind( )** method to ask to bind
- onBind returns binder object (**OdometerBinder**)
- Activity then calls **getOdometer( )** method to get **OdometerService** object

```
...
import android.os.Binder;
import android.os.IBinder;

public class OdometerService extends Service {
    private final IBinder binder = new OdometerBinder();

    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }
...
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
}
```

We're using these classes.

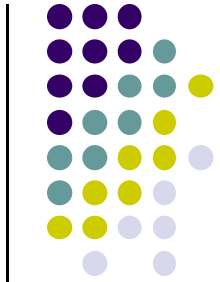
The Binder implementation.

The onBind() method returns an IBinder. This is an interface the Binder class implements.



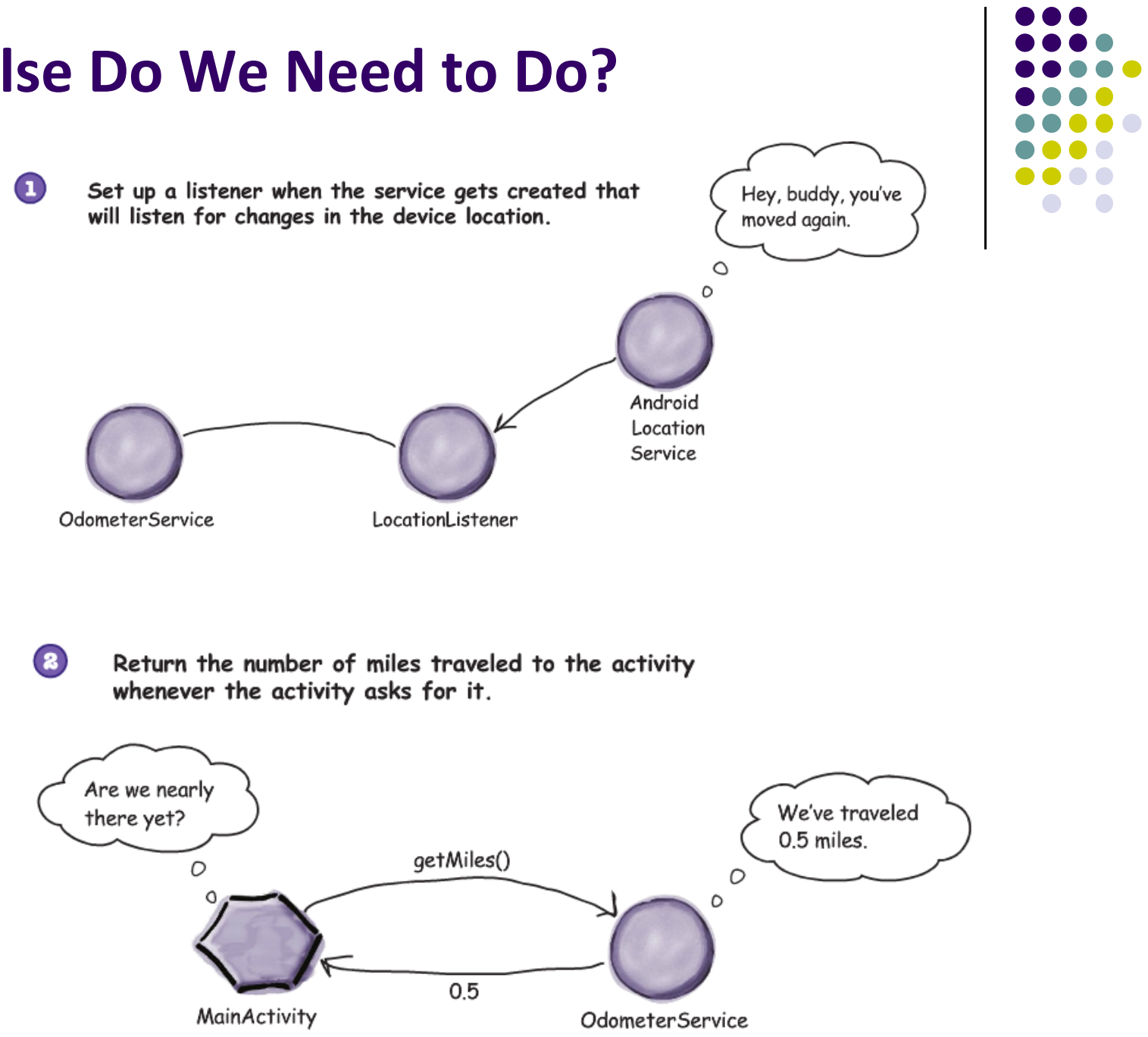
Activity calls onBind to request to bind to service

Main Methods of Service Class

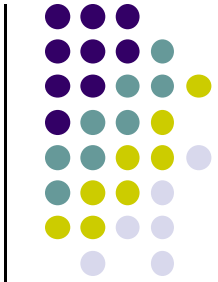


Method	When it's called
onCreate()	When the service is first created
onStartCommand()	When an activity starts the service using the `startService()` method
onBind()	When an activity wants to bind to the service
onDestroy()	When the service is no longer being used and is about to be destroyed

What Else Do We Need to Do?



Need to Create LocationListener



- Find device's location using Android Location Service
- Create LocationListener( ) to get updates whenever location changes

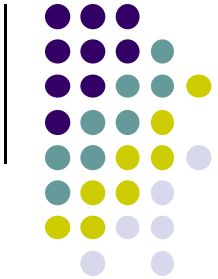
```
LocationListener listener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {
        //Code to keep track of the distance
    }

    @Override
    public void onProviderDisabled(String arg0) {}

    @Override
    public void onProviderEnabled(String arg0) {}

    @Override
    public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
};
```

This is the new LocationListener.

This method gets called whenever the LocationListener is told the device location has changed. The Location parameter describes the current location

You need to overide these methods too, but they can be left empty. They get called when the GPS is enabled or disabled, or if its status has changed. We don't need to react to any of these events.

Add LocationListener to the Service



```
...
public class OdometerService extends Service {

    private static double distanceInMeters;
    private static Location lastLocation = null;
    private LocationListener listener;
    ...

    @Override
    public void onCreate() {
        listener = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                if (lastLocation == null) {
                    lastLocation = location;
                }
                distanceInMeters += location.distanceTo(lastLocation);
                lastLocation = location;
            }

            @Override
            public void onProviderDisabled(String arg0) {}

            @Override
            public void onProviderEnabled(String arg0) {}

            @Override
            public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
        };
    }
}
```

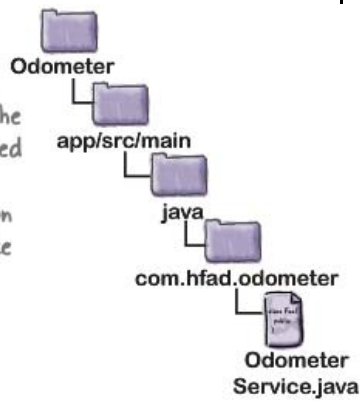
We're storing the distance traveled in meters and the last location as static private variables.

Create the listener as a private variable and initialize it.

If it's our first location, set lastLocation to the current Location.

Add the distance between this location and the last to the distanceInMeters variable, and set lastLocation to the current Location.

We need to override these methods, as they're part of the LocationListener interface.



Create Location Listener, Register it!

- Register location listener with Android Location service using **LocationManager** object
- Create LocationManager object

```
LocationManager locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
```

This is how you access the Android location service.

We used the getSystemService() method earlier to get access to Android's notification service.

- Use its **requestLocationUpdates** method to get updates every second

```
locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                                  1000,
                                  1,
                                  listener);
```

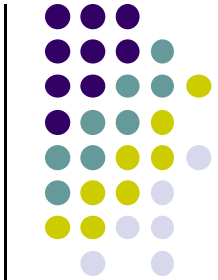
This is the GPS provider.

The time in milliseconds.

The distance in meters.

This is the LocationListener we created.

Put it Together



- Service onCreate method may look like this

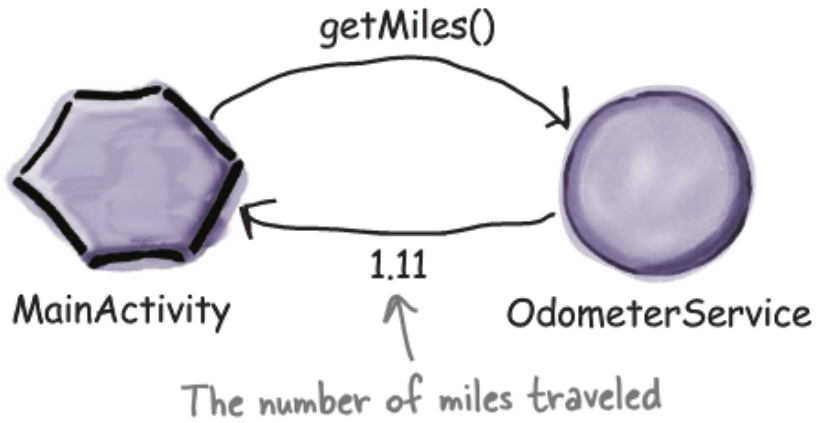
```
...
private LocationManager locManager;

@Override
public void onCreate() {
    listener = new LocationListener() {...};
    locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, listener);
    }
...
```

Create the location manager as a private variable.

This is the listener code we had before.

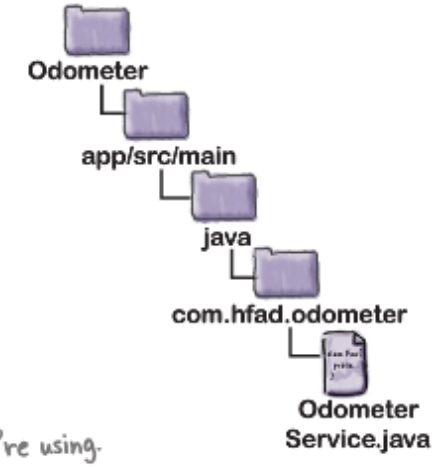
We want to register the listener with the location service when the service is created, and get updates every second when the device has moved more than a meter.

Service Informs Activity of Distance Traveled




```
public double getMiles() {
    return this.distanceInMeters / 1609.344;
}
```

This converts the distance traveled in meters into miles. We could make this calculation more precise if we wanted to, but it's accurate enough for our purposes.




```
package com.hfad.odometer;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Binder;
import android.os.Bundle;
import android.os.IBinder;

public class OdometerService extends Service {

    private final IBinder binder = new OdometerBinder();
    private static double distanceInMeters;
    private static Location lastLocation = null;
    private LocationListener listener;
    private LocationManager locManager;

    public class OdometerBinder extends Binder {
        OdometerService getOdometer() {
            return OdometerService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
```

These are all the classes we're using.

These are the private variables we're using.

When you create a bound service, you have to define a Binder object. It enables the activity to bind to the service.

This gets called when the activity binds to the service.

Full code

Full Code Contd

```
@Override
public void onCreate() {
    listener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            if (lastLocation == null) {
                lastLocation = location;
            }
            distanceInMeters += location.distanceTo(lastLocation);
            lastLocation = location;
        }

        @Override
        public void onProviderDisabled(String arg0) {}

        @Override
        public void onProviderEnabled(String arg0) {}

        @Override
        public void onStatusChanged(String arg0, int arg1, Bundle bundle) {}
    };
    locManager = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000, 1, listener);
}

@Override
public void onDestroy() {
    if (locManager != null && listener != null) {
        locManager.removeUpdates(listener);
        locManager = null;
        listener = null;
    }
}

public double getMiles() {
    return this.distanceInMeters / 1609.344;
}
```

Initialize the location listener when the service is created.

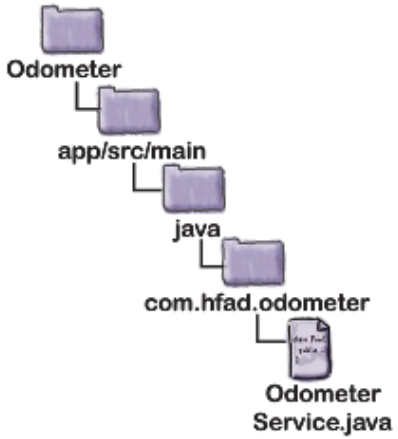
This is our implementaton of the location listener.

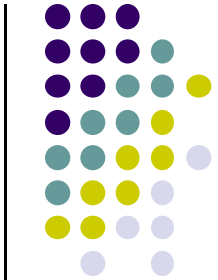
This gets called when the service is no longer being used and is about to be destroyed.

Register the location listener with the location service.

Stop sending location updates to the listener when the service is about to be destroyed.

Convert the distance traveled to miles and return it.





Update AndroidManifest.xml

- Request permission to use GPS

We're adding this because we're using the device GPS in our app.

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

- Service declared in AndroidManifest.xml

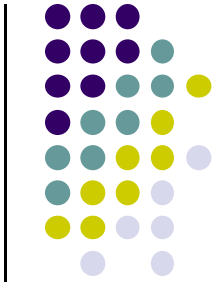
```
<manifest ... >

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

    <application
        ... >
        <activity
            ...
        </activity>

        <service
            android:name=".OdometerService"
            android:exported="false"
            android:enabled="true" >
        </service>
    </application>
</manifest>
```

All services need to be declared in AndroidManifest.xml.

We're setting this to false, as only this app will use the service.

The android:enabled attribute must either be set to true or omitted completely. If you set it to false, your app won't be able to use the service.

AndroidManifest.xml

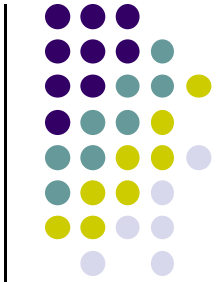


- Location Permissions in manifest

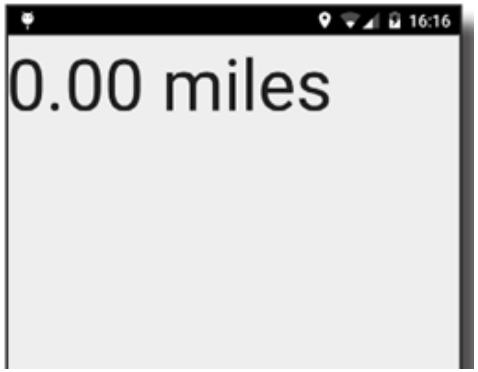
```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
</manifest>
```

- **Options:** ACCESS_FINE_LOCATION or ACCESS_COARSE_LOCATION
  - ACCESS_COARSE_LOCATION: use cell-ID and Wi-Fi
  - ACCESS_FINE_LOCATION: use GPS

More Details in Head First Android (pg 586)

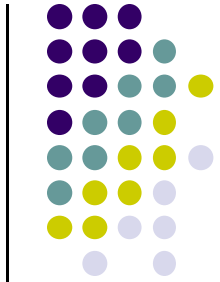


- Display distance traveled on Activity



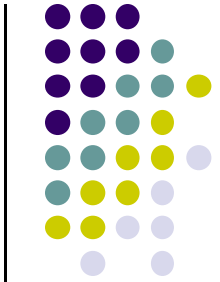
- How to create ServiceConnection

- How to bind activity to service on startup, unbind at end

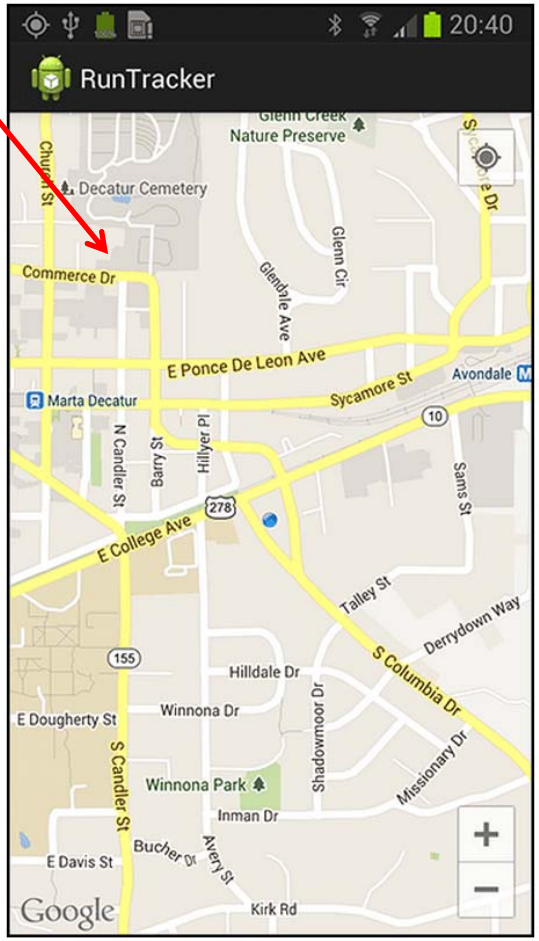


Using Maps

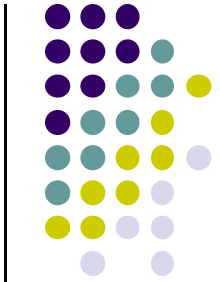
Introducing MapView and Map Activity



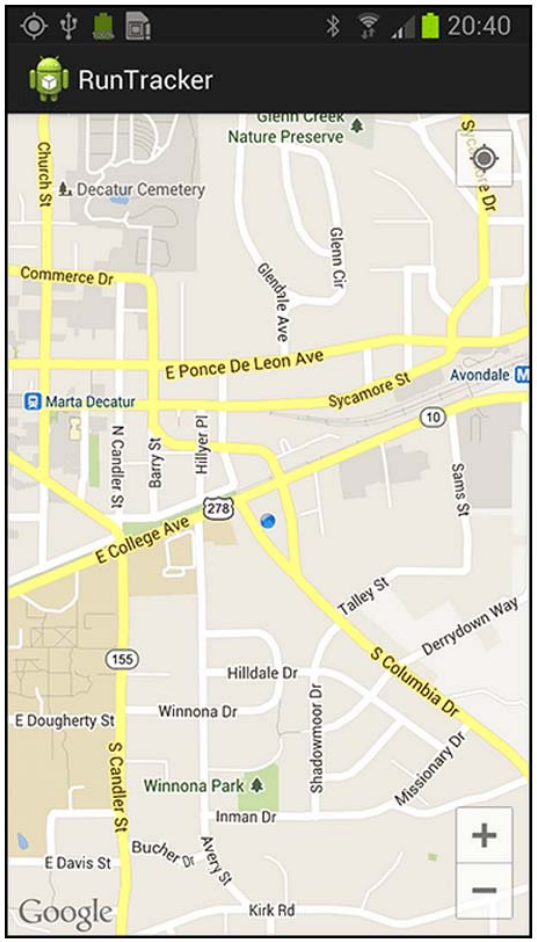
- **MapView:** UI widget that displays maps
- **MapActivity:** java class (extends Activity), handles map-related lifecycle and management for displaying maps.
- **Overlay:** java class used to annotate map, use a canvas to draw unto map layers



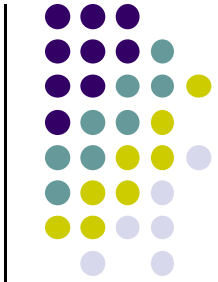
Introducing MapView and Map Activity



- **MapController:** enables map control, setting center location and zoom levels
- **MyLocationOverlay:** Display current location and device orientation
- **ItemizedOverlays and OverlayItems:** used to create markers on map

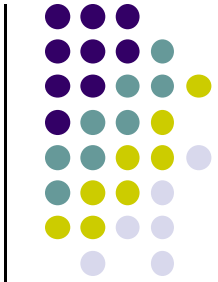


Steps for using Google Maps Android API v2



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



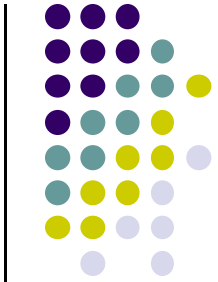
- Google Maps API v2 is part of Google Services SDK
- Main steps to set up Google Play Services

  (See: https://developers.google.com/android/guides/setup)

  1. Use Android Studio SDK manager to download Google Play services
  2. Open **build.gradle** inside your application
  3. Add new build rule under **dependencies**

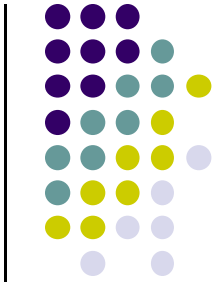
```
apply plugin: 'com.android.application'
    ...

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

Step 3: Get Google Maps API key



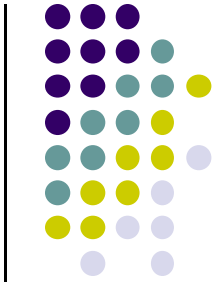
- To access Google Maps servers using Maps API, must add Maps API key to app
- Maps API key is free
- **Background:** Before they can be installed, android apps must be signed with digital certificate (developer holds private key)
- Digital certificates uniquely identify an app, used in tracking:
  -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

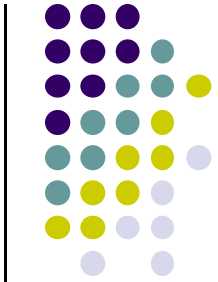
- Add this API key to app in order to use Maps API
- Include API key in AndroidManifest.xml
- To modify AndroidManifest.xml, add following between <application> … </application>

```
<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

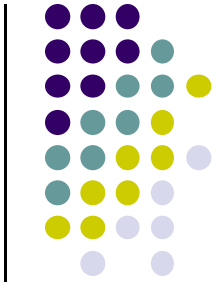
Used by API to cache map tile data in device's external storage

```
<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)

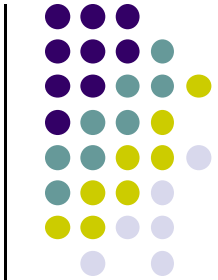


- Specify that OpenGL ES version 2 is required
- Why? Google Maps Android API uses OpenGL ES version 2 to render the map

```
<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

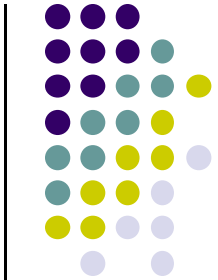
Step 5: Add a map



- To add a map, create XML layout file

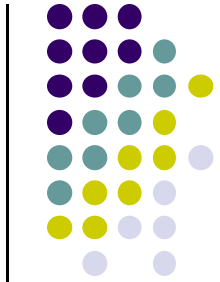
```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
          android:id="@+id/map"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:name="com.google.android.gms.maps.MapFragment"/>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

Install & Configure Google Play Services SDK



- And create MainActivity.java

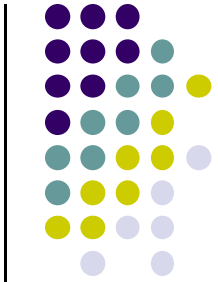
```
package com.example.mapdemo;

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

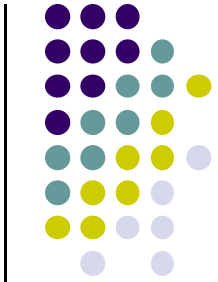
Media Playback

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

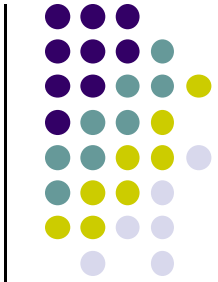
Media Player: Manifest Declarations



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

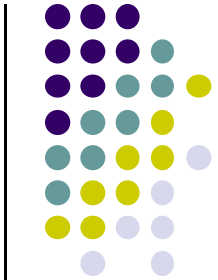
```
<uses-permission android:name="android.permission.INTERNET" />
```

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

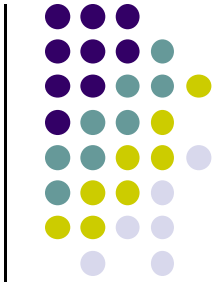
```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you
```

- 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();
```

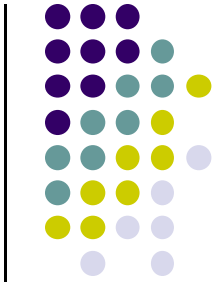
Releasing the MediaPlayer



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

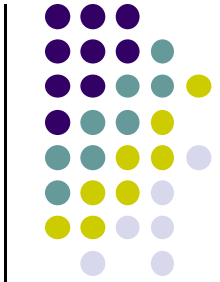
```
mediaPlayer.release();
mediaPlayer = null;
```

- 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

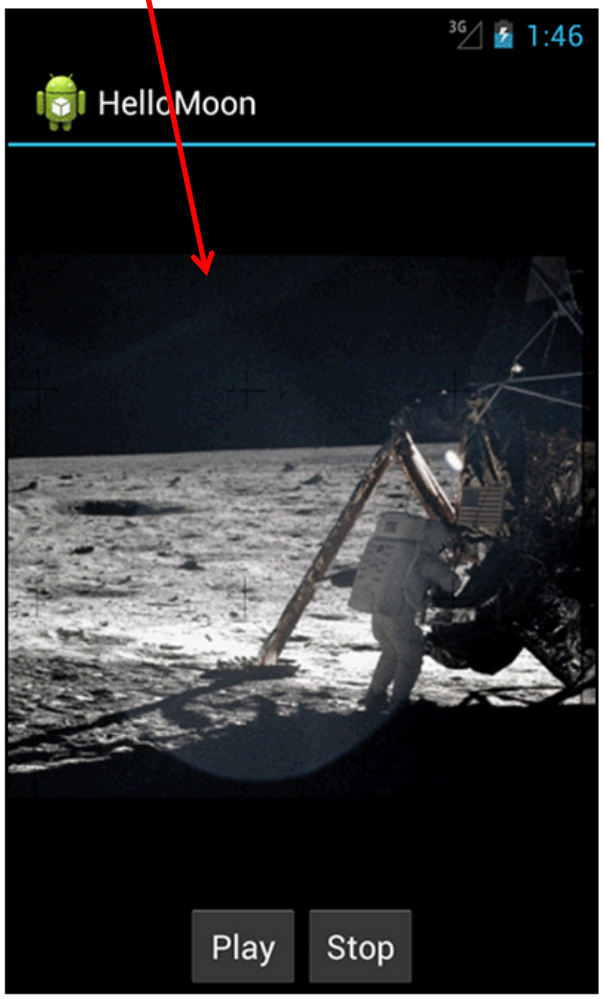


- Example creates **HelloMoon app** that uses **MediaPlayer** to play audio file
- Android Class for audio and video playback
- **Source:** Can play local files, or streamed over Internet
- **Supported formats:** WAV, MP3, Ogg, Vorbis, MPEG-4, 3GPP, etc

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>
```

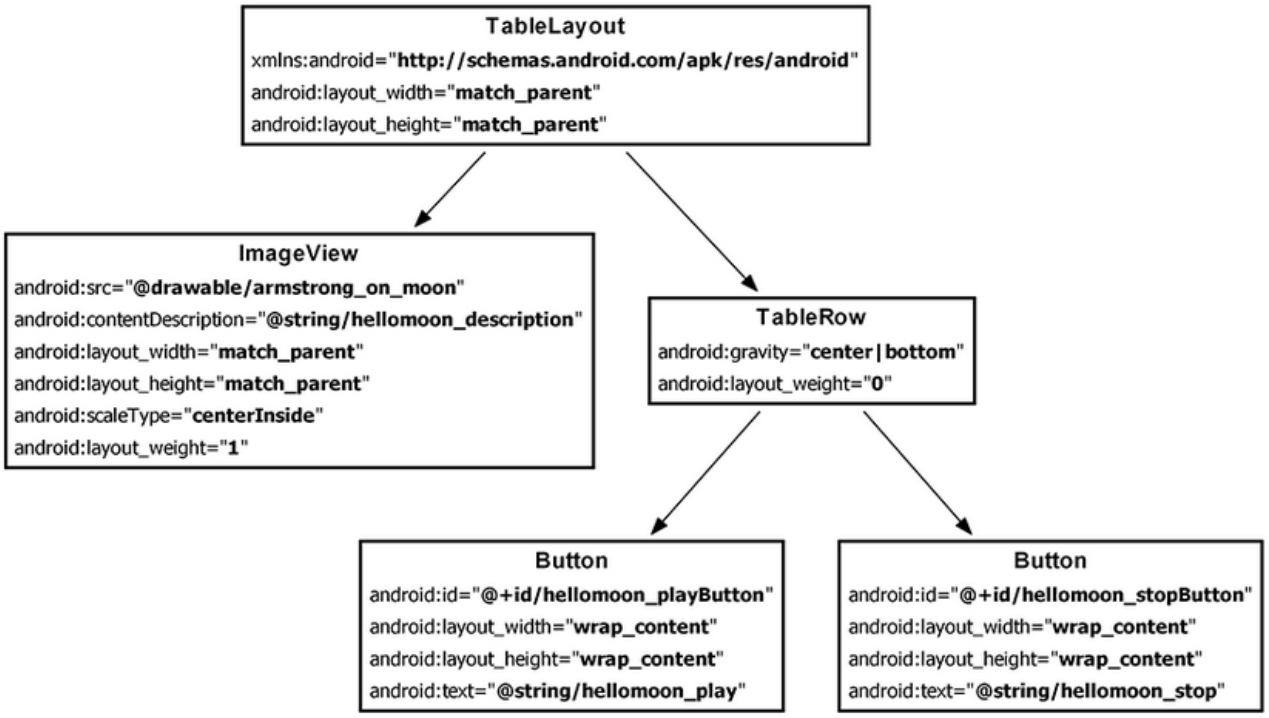


HelloMoon App

- 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

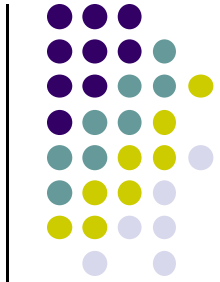


Creating a Layout Fragment

- 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**

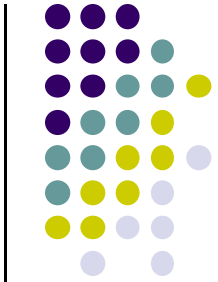
```
<?xml version="1.0" encoding="utf-8"?>
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/helloMoonFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:name="com.bignerdranch.android.hellomoon.HelloMoonFragment">

</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;
    }
}
```

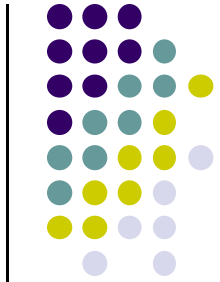
Create AudioPlayer Class to Wrap MediaPlayer



```
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();
    }
}
```

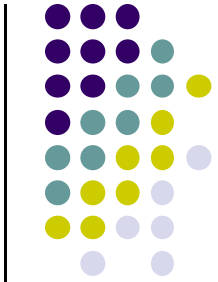
Hook up Play and Stop Buttons

```
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;
    }
```



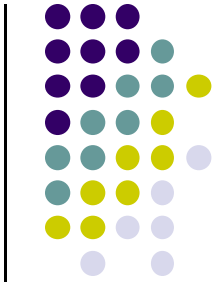
Presentation Guidelines

Overview



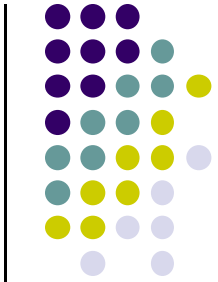
- Next week, class enters new phase of class featuring
  - Paper presentations
  - Writing critiques of papers
- Each week, about 3-6 presenters (see presentation schedule)
- All **students not presenting each week**, pick ANY ONE of the papers presented and write critiques of them
- Next, I provide guidelines on presenting papers and writing critiques

Your Presentation



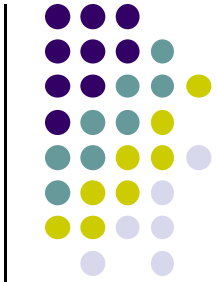
- About 20 mins
- Estimate: about 2 mins per slide
- About 10 slides should be enough + **front page and references** (~ 12 pages)
- Allow 10 mins for questions, discussions

Main Points Presentation Should Cover



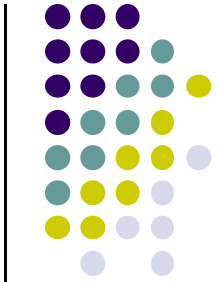
- Introduction/motivation:
  - What was the main problem addressed by the paper?
  - Why is the problem solved important
  - How will the solution be used eventually? How will this new approach save time, resources, incovenience, etc?
  - Focus on what is **new**:
    - scientific results, what was learned
    - Engineering results: new design + justification for choices

Main Points Presentation Should Cover



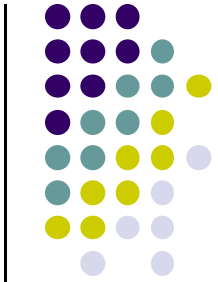
- Related Work:
  - What have other researchers done to solve this problem?
  - How is the approach proposed in this paper different or novel?
    - New approach:
    - New algorithm
    - New technique
    - New experiments

Main Points Presentation Should Cover



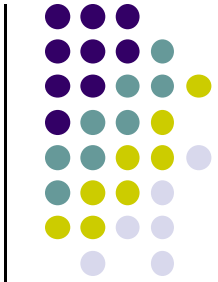
- Methodology/Approach:
  - Summarize the approach/design
  - If a system is described **“how does it work?”**
  - Describe the implementation used (languages, libraries, etc)
  - State any assumptions of the authors and limitations of the proposed work
  - What are the design tradeoffs?

Main Points Presentation Should Cover

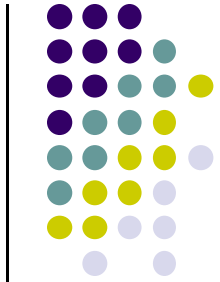


- Results:
  - Present a few of the most significant results/graphs
  - Results should show how well proposed approach worked or findings
  - Do the presented results back up the claims of the authors?

Main Points Presentation Should Cover

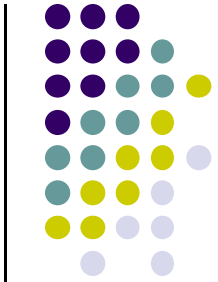


- Discussions/Conclusions/Future Work
  - Summarize what was achieved
  - What did you learn from this paper?
  - What extensions do the authors plan for future work?
  - Brief comments on the paper



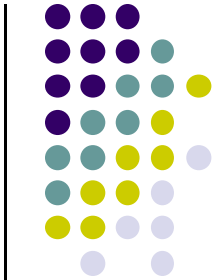
Critique Guidelines

Critique Guidelines



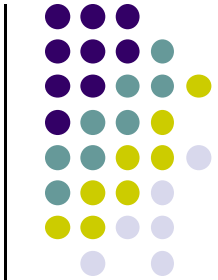
- Capture key points of paper
- Should not exceed half a page
- Don’t just cut-and-paste abstract blindly
- In 1 year’s time, summary should recall key aspects of paper, refresh memory without re-reading paper
- Provide key important details:
  - New idea, concepts, algorithms tools proposed?
- See guidelines on course website

Critique Guidelines (Contd)



- Are assumptions fine?
- Design trade-offs?
- How is the organization of the paper, clarity of writing?
- Did the graphs, results support the claims by authors?
- What was good/Bad about paper?
- Suggestions for improvement?

References



- 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