MONASH University

Information Technology

FIT5183: Mobile and Distributed Computing Systems (MDCS)

# Lecture 10C
# Location and Maps on Android

# Overview

- Review approaches to handle Location Data on Android.
  - Where are we?
  - Which way are we facing?
- How to acquire the user's location via their mobile device based on the frameworks available on Android.
- How to integrate Maps into our mobile interfaces and interact with basic content such as markers.

# Mobile Location Awareness

# Working with Mobile Locations

- Due to the many sensors now available on mobile devices, they are becoming ubiquitous as navigation aids for indoor and outdoor areas.
- Many unique applications have found new ways of using location data to create new experiences or enhance our daily routines in life.
  - E.g. Fitness applications, Fire proximity warnings, ...
- Let's see how the Android platform support location based sensors and how to work with map interfaces in your applications.

# Challenges for Mobile Location Services

- Depending how the location data is sourced, there will be varying levels of accuracy.
- Often the more accurate the information, the higher the performance costs which can result in lower battery life for mobile devices.
- The user's location is dynamic and can change by different variables (such as walking or driving).
  - Location is constantly re-estimated in real-time to provide more meaningful data to the user.

# Android – Location Framework Google Maps Android API v2

# Location Framework

- Android is supported by the LocationFramework which provides access to location services supported by the device.
  - Applications using this framework can determine the approximate location of the device and listen for updates to the location at specific intervals.

- While the framework is useful for working with location data directly, it is often more useful in other location services such as Google Maps where the data is visualised in an effective manner.

Source: http://developer.android.com/guide/topics/location/index.html

# Location Permissions

- Like most functionality now being explored in the Android platform, more permissions need to be specifically outlined in the Android Manifest to become accessible to our application.

**ACCESS_COARSE_LOCATION**

- Enables access to an approximate location of the device provided through sources such as mobile towers and Wi-Fi routers.

**ACCESS_FINE_LOCATION**

- Enables access to more precise location data from sources such as GPS. Also grants access to sources listed above.

**ACCESS_NETWORK_STATE**

- Enables checking if your device is able to connect to a network to request information regarding location.

# Location API

**Criteria**

- Class indicating the app requirements for selecting a particular location provider. E.g. ACCURACY_HIGH, POWER_LOW

**Location**

- Class which represents a geographic location that was acquired through a sensor on the device.

**LocationManager**

- Class provides access to the location services running on the Android device.

**LocationListener**

- Interface for receiving updates from LocationManager when the provider being used has updated the location of the device.

# Example: Using Location Manager

1.Get access to LocationManager via service:

```
LocationManager locationManager =
(LocationManager)this.getSystemService(Context.LOCATION_SERVICE);
```

2.Determine which location provider is most appropriate:

```
Criteria criteria = new Criteria();
String provider =
locationManager.getBestProvider(criteria,false);
Location location =
locationManager.getLastKnownLocation(provider);
```

3.Update location and setup listener if we received a value:

```
if(location!=null){
onLocationChanged(location);
locationManager.requestLocationUpdates(provider,
0,0,this);}
```

# Example: Using Location Manager

4.Implement onLocationChanged(location) method from Activity implementing LocationListener:

```
@Override
Public void onLocationChanged(Location location) {
     TextView latitudeText =
     (TextView)findViewById(R.id.latitudeText);
     latitudeText.setText("Latitude:”+location.getLatitude());
     TextView longitudeText =
     (TextView)findViewById(R.id.longitudeText);
     longitudeText.setText("Longitude:”+location.getLongitude());
}
```

5.Override onPause() Activity lifecycle method to stop listening for location updates when outside of activity or application:

```
@Override
Protected void onPause() {
     /*IMPORTANT! We need to stop listening for location updates if
     we exit the activity.*/
     super.onPause();
     locationManager.removeUpdates(this); }
```

# Google Maps Android API v2

- Google Maps for Android was recently updated bringing faster performance and new features such as 3D mapping and improvements to caching.
- Working with Google Maps requires an API key to access the service.
  - Free! (As long as you stay within the terms of service)
- Unfortunately the latest version of Google Maps API requires access to the Google Play store which is unavailable on the emulator.
  - If you wish to use maps for Android in your assignment, you will need access to a local device.

# Getting Started with Google Maps

- To integrate Google Maps into an Android application, the following steps need to be followed:
  - Configure your project to use Google Play services SDK.
  - Obtain an API key to work with Google Maps.
  - Setup permissions and API key with AndroidManifest.
  - Add a MapFragment into your project.
- Instructions to configure your Android projects and setup your personal API key (tied to a Google account)
  - https://developers.google.com/maps/documentation/android-api/start

# Working with Latitude and Longitude

- Map values are represented by both latitude and longitude values which are used to pinpoint specific locations and place custom markers on the map.
- The library comes with a useful class LatLng which is a simple data representation of those two values which Google uses throughout their Android API.
- Example: Monash Campus Locations

```
static final LatLng BERWICK_CAMPUS = new
LatLng(-38.041, 145.339);
static final LatLng CAULFIELD_CAMPUS = new
LatLng(-37.877, 145.045);
static final LatLng CLAYTON_CAMPUS = new
LatLng(-37.912, 145.133);
...
```

# Permissions in AndroidManifest.xml

- Current version of Google Maps requires devices with OpenGL ES 2.0 support to display the maps correctly:

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

- Then we add all the assorted permissions that Google Maps will need to work optimally:

```
<!– Permission element needs to match project namespace! -->
<permission android:name="monash.campusmap.permission.MAPS_RECEIVE"
android:protectionLevel="signature"/>
<uses-permission android:name="monash.campusmap.permission.MAPS_RECEIVE"/>
<uses-permission android:name="android.permission.INTERNET" />
<!– External storage useful for caching maps -->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"
/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- Lastly, we add our Google Maps API key:

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

# Example: Monash Campus Map

- Now that our permissions are setup, let's look at an example!
- Application will have a MapFragment which displays the interface on the right.
  - We ultimately want to display a marker for each Monash campus!
- We start by adding a MapFragment to our MainActivity layout:

```
<fragment
android:id="@+id/map"
android:layout_width="match_parent"
android:layout_height="match_parent"
class="com.google.android.gms.maps.MapF
ragment"/>
```

# Working with MapFragments

- Let's start setting up our Activity to support the MapFragment:

```
private MapFragment mMapFragment;
private GoogleMap mMap;
```

- We start with the onCreate() method which is executed on the Activity launch and when the device orientation changes:

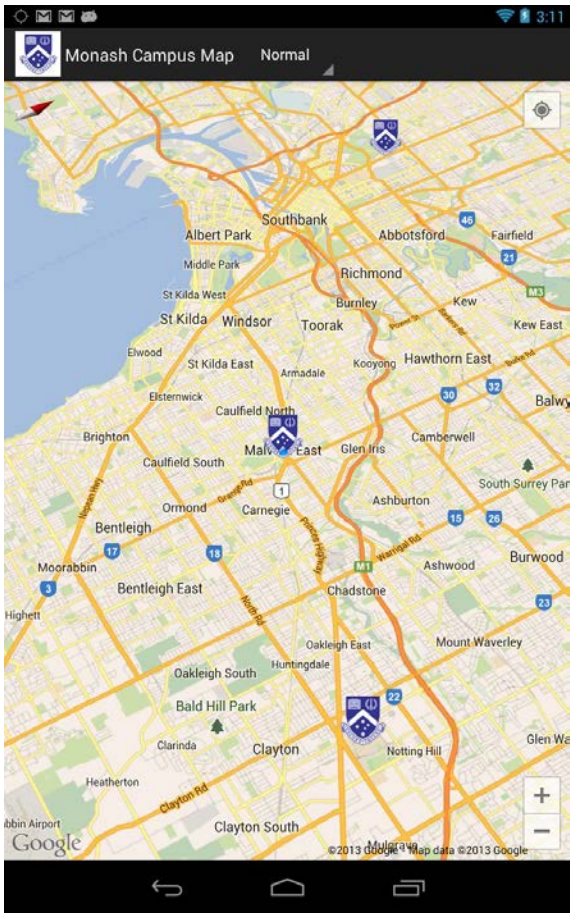
```
mMapFragment =
((MapFragment)getFragmentManager().findFragmentById(R.id.map));

if (savedInstanceState == null) {
    // First time activity is being run, setup map fragment to
    retain data
    mMapFragment.setRetainInstance(true);
    mMapFragment.getMap().setMyLocationEnabled(true);
} else {
    // Repeated activity, we can obtain current fragment instance
    of map
    mMap = mMapFragment.getMap();
}
```

# Adding Markers

- Markers are used to represent locations on the map which can either by system or user generated.
  - Can be assigned textual information and custom views upon selection.
  - Also can be represented by a custom icon.
- Adding custom markers is quite easy:

```
public void addCampusMarker(String title,
LatLng position) {
      MarkerOptions options = new
      MarkerOptions();
      options.position(position);
      options.title(title);
      options.icon(BitmapDescriptorFactory
      .fromResource(R.drawable.monash_shield));
      // Add marker to map
      mMap.addMarker(options);
}
```



# Adding Markers

- Let's create a method to initialise the GoogleMap with markers to represent each Monash campus and call the method in onCreate():

```
private void setupMap() {
if (mMap == null) {
        mMap =
        ((MapFragment)getFragmentManager().findFragmentById(R.id.map)).getMap();
if (mMap != null) {
        // Add campus makers
        addCampusMarker("Berwick Campus", BERWICK_CAMPUS);
        addCampusMarker("Caulfield Campus", CAULFIELD_CAMPUS);
        addCampusMarker("Clayton Campus", CLAYTON_CAMPUS);
        addCampusMarker("Gippsland Campus", GIPPSLAND_CAMPUS);
        addCampusMarker("Parkville Campus", PARKVILLE_CAMPUS);
        addCampusMarker("Peninsula Campus", PENINSULA_CAMPUS);
        addCampusMarker("South Africa Campus", SOUTH_AFRICA_CAMPUS);
        addCampusMarker("Sunway Campus", SUNWAY_CAMPUS);
        }
}
```

# Navigating between Map Types

- Google Maps provide support for four map types: Normal, Hybrid, Satellite and Terrain.
- Let's do something different and create a Navigation Drop Down for the ActionBar to let us change between different Map Types.
  - An array (R.array.mapTypes) representing each map type was created inside the values folder in the project.

```
private void setupActionBar() {
      // Configure ActionBar navigation for changing map types
      ActionBar actionBar = getActionBar();
      actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
      SpinnerAdapter mSpinnerAdapter =
      ArrayAdapter.createFromResource(this.getActionBar().getThemedC
      ontext(),
      R.array.mapTypes,
      android.R.layout.simple_spinner_dropdown_item);
      actionBar.setListNavigationCallbacks(mSpinnerAdapter, this);
}
```

# Navigating between Map Types

```
@Override
public boolean onNavigationItemSelected(int itemPosition, long
itemId) {
      // Get strings used to populate dropdown and selected  item
      String[] mapTypes =
      getResources().getStringArray(R.array.mapTypes);
      String selection = mapTypes[itemPosition];

      // Change map type depending on selection
      if(selection.equals("Normal"))
      "mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
      else if(selection.equals("Hybrid"))
      "mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
      else if(selection.equals("Satellite"))
      "mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
      else if(selection.equals("Terrain"))
      "mMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);

      return true;
}
```

# Navigating between Map Types

- Lastly, we override the Activity lifecycle methods to ensure that the navigation index is saved should the Activity get paused or recreated so that the Map Type selected is persistent.

```
// Need to overwrite these methods to ensure NavigationItem index is
saved
@Override
protected void onSaveInstanceState(Bundle outState) {
      // Set current mapType value so it can be restored!
      super.onSaveInstanceState(outState);
      outState.putInt("selectedIndex",
      getActionBar().getSelectedNavigationIndex());
}
@Override
protected void onRestoreInstanceState(Bundle inState) {
      super.onSaveInstanceState(inState);
      getActionBar().setSelectedNavigationItem(inState.getInt("selec
      tedIndex"));
}
```

# References

- Lecture Notes of FIT3027/FIT4039: Sensors and Location Awareness
- Location and Maps: http://developer.android.com/guide/topics/location/index.html