Using Location and Map APIs

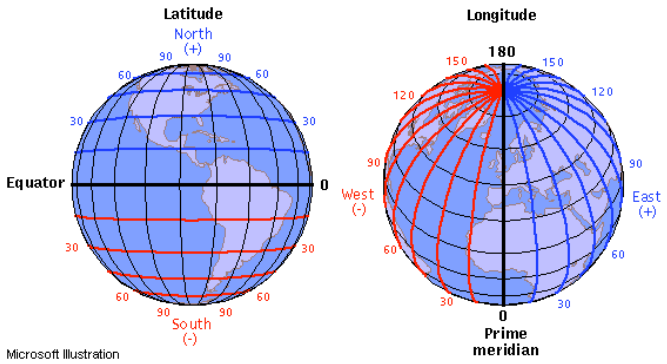
Santa Clara University

Center for Science, Technology, and Society

What and Why

- Use Maps to display locations
- GPS to sense current location (LocationManger class in Android)
- Geocoding = translation from and to

addresses ←→ latitude, longitude



Google Maps Android API v2

- Part of the Google Play services SDK,
- Uses MapFragment class
- Version 2 uses vector tiles (vectiles). Their data representation is smaller, so maps appear in your apps faster, and use less bandwidth (VS. tiled web maps)
  - For more: check Mapbox
- Caching is improved, so users will typically see a map without empty areas.
- Maps are now 3D. You can change user perspective

V2 is not supported on some Android Emulator

- Generally best to have a physical device to test.

- ALSO: devices not supporting OpenGL ES2.0 won't be able to run apps with maps (older devices)

Adding Map to Project

1. Download and configure the Google Play services SDK. The Google Maps Android API is distributed as part of this SDK.
2. Obtain an API key. To do this, you will need to register a project in the Google APIs Console, and get a signing certificate for your app.
3. Setup your Project to say it is using the GooglePlay Services
4. Specify settings in the Application Manifest. Add a map to a new or existing Android project.

Register For Google API Key

- Need to register your app (packagename.app) in google console to use maps to get api key
- Update your Android project (typically google_maps_api.xml file) to contain this new api key
- Add MapFragment (or other related class) to display and configure a Google map.

signing your app

- **See** https://developer.android.com/tools/publishing/app-signing.html
- Android Studio signs your app in debug mode automatically when you run or debug your project from the IDE.
- **Debug key info (**The debug keystore is located in $HOME/.android/debug.keystore**)**

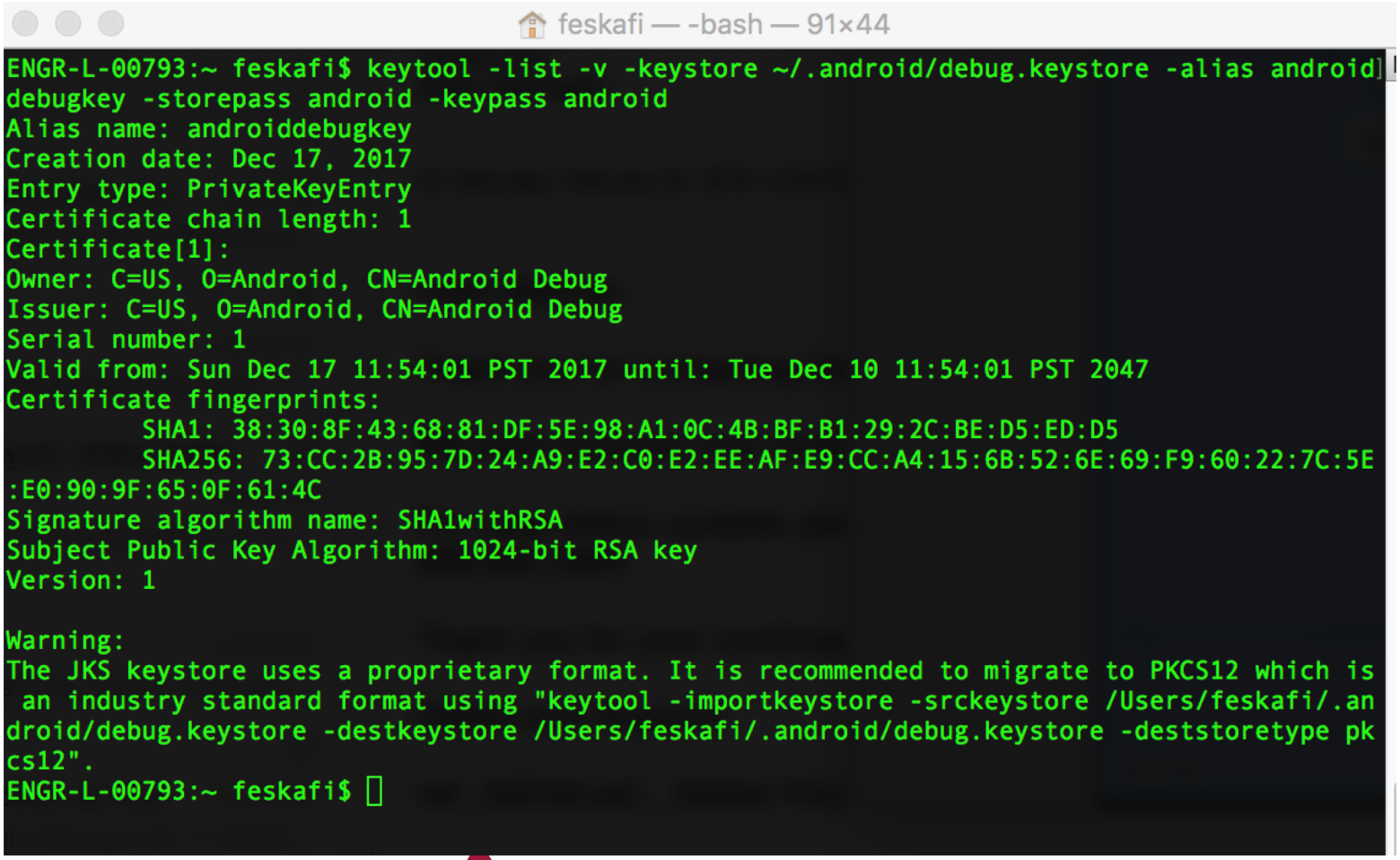
You can't publish the app with this debug key

Certificate Information

List the SHA-1 fingerprint.

- For Linux or OS X, open a terminal window and enter the following:
  - keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
- For Windows, run:
  - keytool -list -v -keystore "C:\Users\your_user_name\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android

Android debug key



App must ask for permissions

- In AndroidManifest.xml file (right after <manifest tag):

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

<!-- 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"/>
```

Specify Use of OpenGL

- The Google Maps Android API uses OpenGL ES version 2 to render the map.
- If OpenGL ES version 2 is not installed, your map will not appear.
- Add the following <uses-feature> element as a child of the <manifest> element inAndroidManifest.xml:

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

Mapping Locations

- The Android SDK provides two different methods to show a location with Google Maps.
- The first method is to use a location URI to launch the built-in Google Maps application with the specified location.
- The second method is to use a `MapFragment` embedded within your application to display the map location.

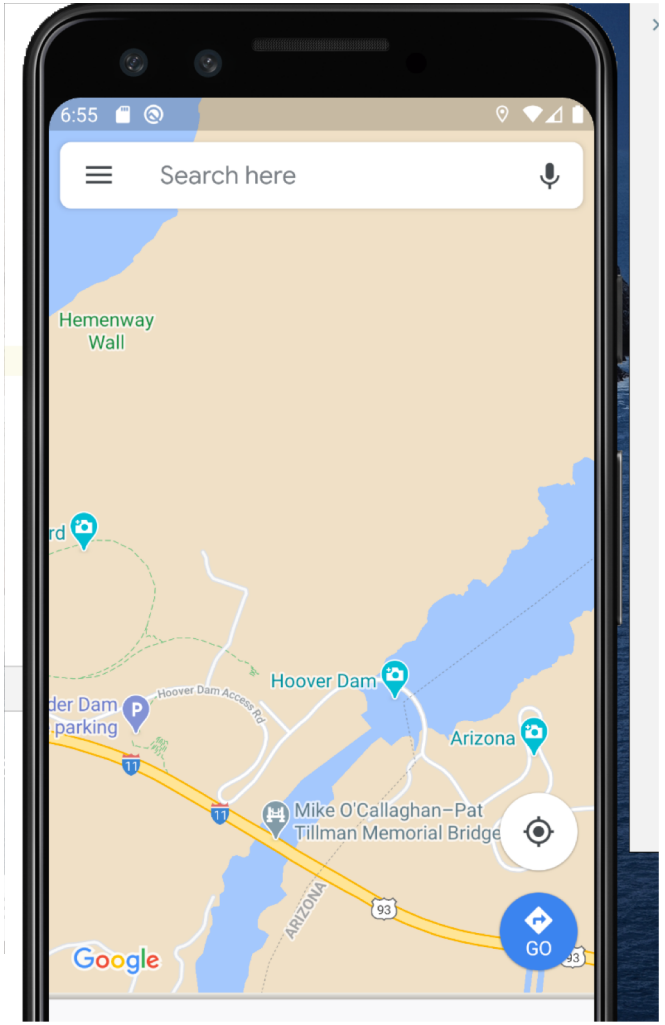
Mapping Intents

- The first task is to create a String that conforms to the URI handled by the mapping application. In this case, it's geo: followed by the latitude and longitude
- This URI is then used to create a new Uri object for creating new ACTION_VIEW intent
- Call the startActivity() method

```
String geoURI = String.format("geo:%f,%f", 36.02, -
114.74);
Uri geo = Uri.parse(geoURI);
Intent geoMap = new Intent(Intent.ACTION_VIEW, geo);
startActivity(geoMap);
```

Mapping Intents

- Using this method of mapping launches the user into a built-in mapping application—in this case, Google Maps.
- If you do not want to bother with the details of a full mapping application or do not need to provide any further control over the map, this is a fast and easy method to use.



MapFragment

public class **MapFragment** extends Fragment

- A Map component in an app
- This fragment is the simplest way to place a map in an application
- It's a wrapper around a view of a map to automatically handle the necessary life cycle needs.
- Can be added to an activity's layout file simply with the XML below.

```
<fragmentclass="com.google.android.gms.maps.MapFragment”
android:layout_width="match_parent”
android:layout_height="match_parent"/>
```

MapFragment (2)

- A GoogleMap must be acquired using **getMapAsync(OnMapReadyCallback)**
- This class automatically initializes the maps system and the view
- A view can be removed when the MapFragment's **onDestroyView()** method is called and the **useViewLifecycleInFragment(boolean)** option is set.
  - MapFragment is no longer valid until the view is recreated again later
  - MapFragment's **onCreateView(LayoutInflater, ViewGroup, Bundle)** method is called.
- Any objects obtained from the GoogleMap are associated with the view
- It's important to not hold on to objects (e.g. Marker) beyond the view's life
  - memory leak as the view cannot be released.

Manifest for Maps

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

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

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

    <application

        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
```

Manifest (2)

```
<meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="@string/google_maps_key" />

    <activity
        android:name=".MapsActivity"
        android:label="@string/title_activity_maps">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Camera – changing location

- See https://developers.google.com/maps/documentation/android/views

- Change location

```
private static final LatLng SCU = new LatLng(37,-122);

private GoogleMap map;
... // Obtain the map from a MapFragment or MapView.

// Move the camera instantly to SCU with a zoom of 15.
map.moveCamera(CameraUpdateFactory.newLatLngZoom(SCU, 15));
```

Changing Map in Code

- **MapsActivity**

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified
        // when the map is ready to be used.
        SupportMapFragment mapFragment =
                                (SupportMapFragment) getSupportFragmentManager()
                                                        .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
}
```

MapsActivity (2)

```
@Override
public void onMapReady(GoogleMap googleMap) {
   mMap = googleMap;

   // Add a marker on SCU and move the camera
   LatLng scu = new LatLng(37.35, -121.94);
   mMap.addMarker(new MarkerOptions().position(scu).title("Marker in
SCU"));
   mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(scu,15));

}
```

Map Features

- Zoom in/out
- Location on map
- Markers

Camera –zoom

- See https://developers.google.com/maps/documentation/android/views

- **Zoom in** and **out**

```
// Zoom in, animating the camera.
map.animateCamera(CameraUpdateFactory.zoomIn());

// Zoom out to zoom level 10, animating with a duration of 2 seconds.
map.animateCamera(CameraUpdateFactory.zoomTo(10), 2000, null);
```

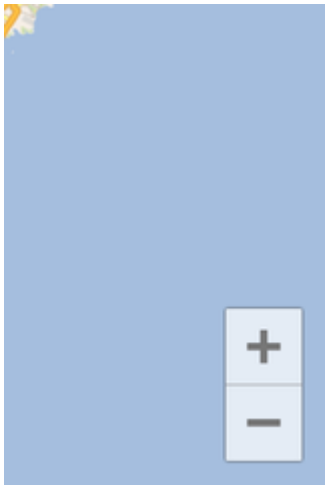
Animate to New Location/Zoom/Bearing/Tilt

```
//Construct a CameraPosition focusing on Mountain View and animate the
//camera to that position.
private static final LatLng MOUNTAIN_VIEW = new LatLng(37.4, -122.1);

CameraPosition cameraPosition = new CameraPosition.Builder()
    .target(MOUNTAIN_VIEW)    // Sets the center of map to Mountain View
    .zoom(17)                 // Sets the zoom
    .bearing(90)              // Sets the orientation of the camera to east
    .tilt(30)                 // Sets the tilt of the camera to 30 degrees
    .build();                 // Creates a CameraPosition from the builder


map.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition)
);
```

Zoom Control

- On real device you can pinch to zoom in and out but, not on an emulator
- The Maps API provides built-in zoom controls that appear in the bottom right hand corner of the map
- These are disabled by default, but can be enabled by calling

  UiSettings.setZoomControlsEnabled(true).



Hit (bottom of map)
– icon to zoom in
+ icon to zoom out

Adding a Marker

```
//grab GoogleMap instance from MapFragment or SupportMapFragment or….


//Add marker at location with title Hello world
map.addMarker(new MarkerOptions()
        .position(new LatLng(10, 10))
        .title("Hello world"));
```

- **See https://developers.google.com/maps/documentation/android/marker for other options including:**
  - Position (required)
  - Title = A string that's displayed in the info window when the user taps the marker.
  - Snippet = Additional text that's displayed below the title.
  - Icon =A bitmap that's displayed in place of the default marker

Changing Marker color

```
static final LatLng MELBOURNE = new LatLng(-37.813, 144.962);

Marker melbourne = mMap.addMarker(new MarkerOptions()
                .position(MELBOURNE)
                .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE)))
;
```

Register and Implement OnMapClickListener

- GoogleMap.setOnMapClickListener(OnMapClickListener)
- Implement onMapClick(LatLng) method
  - When a user clicks (taps) somewhere on the map, you will receive an event that indicates the location on the map that the user clicked.
  - you can obtain a Projection from the map which allows you to convert between latitude/longitude coordinates and screen pixel coordinates.
    - **Point toScreenLocation(Latlng location**)
  - **https://developers.google.com/android/reference/com/google/android/gms/maps/Projection**

Using the Global Positioning System (GPS)

- The Android location APIs provide the means for accessing location via built-in GPS hardware, when it's available.
- Just about every Android phone has some location capabilities.
  - For example, in the United States, emergency services use mobile phone location information.
- Not all Android devices are phones, nor do all phones enable consumer usage of location services.
- If GPS features are disabled, or an Android device does not have location hardware, the Android SDK provides additional APIs for determining alternative location providers.
  - These other providers might have advantages and disadvantages in terms of power use, speed, and accuracy of reporting.

Using GPS Features

```
<uses-feature
android:name="android.hardware.location" />
```

or

```
<uses-feature
android:name="android.hardware.location.gps" />
```

Location Scenarios

- Last Known Location
  - Request last known location of the user's device
- Location Settings
  - Many different location sources: GPS, WiFi
  - Must specify desired level of service
- Location Updates
  - Provide location update to a callback in your app at specific intervals
  - Interval can be a function of quality of service
  - Use it to provide direction and velocity

Fused Location Provider

- A location API in Google Play services that intelligently combines different signals to provide the location information that your app needs
- Manages the underlying location technologies, such as GPS and Wi-Fi
- Provides a simple API that you can use to specify the required quality of service
- No need to be concerned with managing power consumption on their own
- Ability to locate with fine-grained accuracy with very little coding effort.

Locating with the Fused Location Provider

- To make use of the fused location provider, there are a few simple steps you need to follow:
  1. Add the appropriate permissions to the Android manifest file, declaring the accuracy that your application requires, such as `ACCESS_COARSE_LOCATION` or `ACCESS_FINE_LOCATION`.
  2. Make sure to check that Google Play services has been installed on the device; otherwise, your application will crash.
  3. Implement the appropriate callback methods for the `OnConnectionFailedListener` and `ConnectionCallbacks`.
  4. Create a location client using the `LocationClient` class in the `onCreate()` method of your application, and handle the appropriate lifecycle events to `connect()` and `disconnect()` the location client where appropriate.
  5. Use the `getLastLocation()` method of the location client to retrieve the location of the device.

Getting the Last Known Location

- Inside Activity

  public class MainActivity extends AppCompatActivity implements
  ConnectionCallbacks, OnConnectionFailedListener {

- Step 1: specify permission in Android Manifest file:
  - <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

- Step 2: create instance Google Play services API client
  - **INSIDE ACTIVITIES onCreate(*) method:**

```
mGoogleApiClient = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(this)
    .addOnConnectionFailedListener(this)
    .addApi(LocationServices.API)
    .build();
```

NOTE: once the GoogleApiClient is ready for use will call this Activity as it is the ConnectionCallbacks invoke onConnected(*) method

We add LocationServices as Google API we want to access in client

Getting Last Known Location (2)

- STEP 3: create instance of FusedLocationProviderAPI to call its getLastLocation method inside the callback onConnected method that is called when GoogleApiClient is ready

```
  //class variable:   Location mLastLocation
  //inside Activity acts as ConnectionCallbacks
   @Override
   public void onConnected(Bundle connectionHint) {
      mLastLocation = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);

      if (mLastLocation != null) {
         mLatitudeText.setText(String.valueOf(mLastLocation.getLatitude()));
         mLongitudeText.setText(String.valueOf(mLastLocation.getLongitude()));
      }
   }
}
```

Retrieving Location Updates

- As phone/device moves, you get updates to your app
- Besides Lat/Long , get bearing, velocity and altitude
- Can use WiFi, GPS, cell info
- Accuracy determined by provider

- Uses **FusedLocationProviderApi.requestLocationUpdates(*)**

Retrieving Location Updates (2)

- Step 1: Activity declare it as a LocationListener

  public class MainActivity extends AppCompatActivity implements
  ConnectionCallbacks, OnConnectionFailedListener, **LocationListene**r {

- Step 2: specify permission in Android Manifest file:
  - <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
- Step 3: create instance Google Play services API client
  - **INSIDE ACTIVITIES onCreate(*) method:**

```
mGoogleApiClient = new GoogleApiClient.Builder(this)
    .addConnectionCallbacks(this)
    .addOnConnectionFailedListener(this)
    .addApi(LocationServices.API)
    .build();
```

NOTE: once the GoogleApiClient is ready for use will call this Activity as it is the ConnectionCallbacks invoke onConnected(*) method

NOTE: we add LocationServices as Google API we want to access in client

Retrieving Location Updates (3)

- STEP 4: now create instance of FusedLocationProviderAPI to call its requestLocationUpdates method inside the callback onConnected method that is called when GoogleApiClient is ready

```
    ...
    @Override
    public void onConnected(Bundle connectionHint) {
      //whatever other code you have here……
•     if (mRequestingLocationUpdates) {
          startLocationUpdates();
      }

    protected void startLocationUpdates() {
        LocationServices.FusedLocationApi.requestLocationUpdates(
            mGoogleApiClient, mLocationRequest, this);
    }

  }
}
```

Retrieving Location Updates (4)

- STEP 5: implement the **onLocationChanged** method of the LocationListener interface in your activity

```
@Override
   public void onLocationChanged(Location location) {
      mCurrentLocation = location;
      mLastUpdateTime = DateFormat.getTimeInstance().format(new Date());
      updateUI();
   }

   private void updateUI() {    //just a method to alter GUI with new lat/long
      mLatitudeTextView.setText(String.valueOf(mCurrentLocation.getLatitude()));
      mLongitudeTextView.setText(String.valueOf(mCurrentLocation.getLongitude()));
      mLastUpdateTimeTextView.setText(mLastUpdateTime);
   }
```

NOTE: onLocationChanged called automatically each time a new Location event has occurred.

Pause and Resume Updates

- Consider when Activity Paused and Resumed--

```
@Override
protected void onPause() {
   super.onPause();
   LocationServices.FusedLocationApi.removeLocationUpdates(
         mGoogleApiClient, this);
}


@Override
public void onResume() {
   super.onResume();
   if (mGoogleApiClient.isConnected() && !mRequestingLocationUpdates) {
      startLocationUpdates();
   }
}
```

Saving State When Activity Destroyed

- Changing orientation can cause Activity to be destroyed --- we need to save our current operational state (that we are using location services) and then in onCreate restore these values

```
public void onSaveInstanceState(Bundle savedInstanceState) {
   savedInstanceState.putBoolean(REQUESTING_LOCATION_UPDATES_KEY,
         mRequestingLocationUpdates);
   savedInstanceState.putParcelable(LOCATION_KEY, mCurrentLocation);
   savedInstanceState.putString(LAST_UPDATED_TIME_STRING_KEY, mLastUpdateTime);
   super.onSaveInstanceState(savedInstanceState);
}
```

Saving State When Activity Destroyed

- and then in onCreate restore these values

```
@Override
public void onCreate(Bundle savedInstanceState) {
   ...
   updateValuesFromBundle(savedInstanceState);
}

private void updateValuesFromBundle(Bundle
savedInstanceState) {
   if (savedInstanceState != null) {
      // Update the value mRequestingLocationUpdates
      // from the Bundle, & make sure that the Start
      //Updates and Stop Updates buttons are
      // correctly enabled or disabled.
      if
(savedInstanceState.keySet().contains(REQUESTING_LOCATI
ON_UPDATES_KEY)) {
         mRequestingLocationUpdates =
savedInstanceState.getBoolean(
               REQUESTING_LOCATION_UPDATES_KEY);
        setButtonsEnabledState ();
      }
// Update the value of mCurrentLocation from the Bundle and update the
// UI to show the correct latitude and longitude.
      if (savedInstanceState.keySet().contains(LOCATION_KEY)) {
         // Since LOCATION_KEY was found in the Bundle, we can be sure that
         // mCurrentLocationis not null.
         mCurrentLocation =
          savedInstanceState.getParcelable(LOCATION_KEY);
      }

      // Update the value of mLastUpdateTime from the Bundle and update
the UI.
      if
(savedInstanceState.keySet().contains(LAST_UPDATED_TIME_STRING_KEY)) {
         mLastUpdateTime = savedInstanceState.getString(
               LAST_UPDATED_TIME_STRING_KEY);
      }
      updateUI();
   }
}
```

Doing More with Google Location Services

- The fused location provider is just the first improvement of the Google location services API.
- In addition, there are two other APIs that you should be aware of:
  - The activity recognition APIs
  - The geofencing APIs

Understanding Activity Recognition APIs

- The activity recognition APIs are used for detecting the user's current physical activity.
- Here is a list of the activities, by their constant value and description, that your application is able to detect:
  - `IN_VEHICLE`: Determine if a user is driving a vehicle.
  - `ON_BICYCLE`: Determine if a user is riding on a bike.
  - `ON_FOOT`: Determine if a user is walking or running.
  - `STILL`: Determine if a user is not moving at all.
  - `TILTING`: Determine if a user is tilting the device.
  - `UNKNOWN`: If the user's current activity does not fit into one of these classifications, the activity is unknown.

Understanding Geofencing APIs

- A `Geofence` is an area that you define for detecting whether a user has entered or exited a particular latitude, longitude, and specified radius.
- To set up a `Geofence` for your application, there are a few things you need to do:
  1. Add the appropriate location permissions to the Android manifest file.
  2. Verify that your app is able to access Google Play services, as geofencing APIs are available only if the Google Play application is installed.
  3. Define a `Geofence` storage mechanism using `SharedPreferences`.
  4. Create a `Geofence` using a latitude, longitude, and radius.
  5. Monitor for when a user enters or exits a `Geofence` and handle the intents appropriately.
  6. Stop monitoring and remove the geofences.

My Location in Map

- To enable:
  - **map.setMyLocationEnabled(true);**

- The My Location button appears in the top right corner of the screen *only* when the My Location layer is enabled.

- Click goes to user location if known. A click will also trigger the GoogleMap.OnMyLocationButtonClickListener.

- disable the button from appearing altogether by calling **UiSettings.setMyLocationButtonEnabled(false)**