Chapter 7

7. Location in Android

7.1. Location-Based Service and Geocoding

What makes mobile so different is we can build advanced services that only mobile device with sensing can deliver. We do this all the time on our phone: do a location-based search for say supermarkets, cafe, cinema, users etc. Users take their devices everywhere and are constantly using them on the go, and as developers we can capitalize on that by providing a more contextual experience based on their current location. Accessing the current location of an Android device is easier than ever, but it can still be a little tricky, especially for the first time. Android phone uses the location of the phone as an input to the search. This is called location-based service. Examples DarkSky, MapMyFitness, Glympse, Uber etc. Android provides a number of building blocks for location-based services.

Android has two basic ways to determine a user's location.

The first is to use the **built-in location APIs** that have been available since Android was first introduced. These still work, but not as well as the newer location APIs bundled as part of the best known as **Google Play Services**

The Google Location Services API, part of Google Play Services, provides a more powerful, high-level framework that automates tasks such as location provider choice and power management. Location Services also provides new features such as activity detection that aren't available in the framework API. Developers who are using the framework API, as well as developers who are just now adding location-awareness to their apps, should strongly consider using the Location Services API.

Google Play Services is a bundle of services and APIs from Google that we can use for lots of features in Android apps. They are installed on a majority of devices and run on Android 2.3 and up. There are a lot of different parts of Google Play Services, but we will see how to include them in a project and use them for detecting location in a quick and effective way.

We need to download the Google Play Services SDK using the SDK Manager. It's a separate download just like a new version of the Android SDK. If you don't have it already, check out the Setting Up Google Play Services page on the Android developer site for details on how to download it. You may also want to download an emulator (AVD) image that uses the Google APIs.

Location manager

The Android location manager gives location in terms of longitude and latitude for the location of the phone. Depending on the location provider selected (could be based on GPS, WiFi or Cellular) the accuracy of the location will vary.

The key Android plumbing for location is:

- Location Manager, which provides the coordinates
- Location Providers, which can make a number of tradeoffs to offer the user the capability they want

A number of services can be built using these simple components:

- get the user's current location
- periodically get the user location as the move around -- provides a trail of bread crumbs
- use proximity alerts when you move in and out of a predefined area (e.g., Time Square)

The user gets a location manager by specifying the LOCATION_SERVICE as input to the getSystemService() of the activity. A location manager is returned.

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  LocationManager locationManager;
  String svcName = Context.LOCATION_SERVICE;
  locationManager = (LocationManager)getSystemService(svcName);
```

In the Manifest you will see that it is necessary to get the user's permission to track their location or get a location reading:

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

There are fine and coarse permissions that the developer can use. The defaults is: if you ask for *ACCESS_FINE_LOCATION* then by default you get coarse. FINE is typically associated with GPS and coarse location with Cellular or network.

Location Provider

Mobile phones can provide location from a set of providers that make a number of tradeoffs. For example, GPS has good accuracy outdoors but is costly in terms of battery consumption for using the GPS chips on the phone. In contrast, Cellular is cheap in terms of energy consumption but could provide very coarse location information (say in the upper valley) because of the lack of cell tower density but could be great in the city. There are a number of tradeoffs that the user might want to make when selecting a location provider. Basically, depending on what location device, the user selects there are a number of different tradeoffs in:

- power consumption
- longitude/latitude accuracy
- altitude accuracy
- speed
- direction information

The user can specify the location provider explicitly in the code using a number of constants:

- LocationManager.GPS_PROVIDER
- LocationManager.NETWORK_PROVIDER
- LocationManager.PASSIVE_PROVIDER

It would be poor programming to rigidly specify the provider -- for example, the user might turn off GPS. It is better to let the Android systems match the user's needs to what providers are on offer. To do this we use *Criteria* as shown below. The code states that the user requires:

- coarse accuracy
- low power consumption
- no altitude, bearing or speed

The code snippet is taken again from onCreate(). The user specifies the level of location information and then asks the system for the best provider given what is currently available and the user's needs.

```
    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_FINE);
    criteria.setPowerRequirement(Criteria.POWER_LOW);
    criteria.setAltitudeRequired(false);
    criteria.setBearingRequired(false);
    criteria.setSpeedRequired(false);
    criteria.setCostAllowed(true);

    String provider = locationManager.getBestProvider(criteria, true);
```

Geocoding

Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate. Geocoder supports two services:

- forward geocoding: from address to longitude/latitude
- reverse geocoding: from longitude/latitude to address. Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a (partial) address. The amount of detail in a reverse geocoded location description may vary, for example one might contain the full street address of the closest building, while another might contain only a city name and postal code. The Geocoder class requires a backend service that is not included in the core android framework.

The Geocoder class comes with the Google Maps library. To use the library you have to import it into the application. In addition, the Geocoder class uses a server to translate over the Internet so you need to add the following permission to the Manifest:

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

Example: Getting current location. The following example displays longitude and latitude of your current location.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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="match_parent"
        android:layout_height="wrap_content"
        android:text="Get Current Location"
        android:id="@+id/getLocationBtn"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/locationText"
        android:layout_below="@id/getLocationBtn"/>
</RelativeLayout>
```

MainActivity.java

```
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity implements LocationListener {
    Button getLocationBtn;
    TextView locationText;

    LocationManager locationManager;

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

        getLocationBtn = (Button)findViewById(R.id.getLocationBtn);
        locationText = (TextView)findViewById(R.id.locationText);

        getLocationBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                getLocation();
            }
        });
    }

    void getLocation() {
        try {
            locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
            locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 5000,
5, this);
        }
        catch(SecurityException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onLocationChanged(Location location) {
        locationText.setText("Current Location: " + location.getLatitude() + ", " +
location.getLongitude());
    }

    @Override
    public void onProviderDisabled(String provider) {
        Toast.makeText(MainActivity.this, "Please Enable GPS and Internet",
Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {

    }

    @Override
    public void onProviderEnabled(String provider) {
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.currentlocationtest">
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <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">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

7.2. Integrating Google Map with android application

Google Maps is one of the many applications bundled with the Android platform. Android provides facility to integrate Google map in our application. Google map displays your current location, navigate location direction, search location etc. We can also customize Google map according to our requirement.

Types of Google Maps

There are four different types of Google maps, as well as an optional to no map at all. Each of them gives different view on map. These maps are as follow:

- **Normal:** This type of map displays typical road map, natural features like river and some features build by humans.
- **Hybrid:** This type of map displays satellite photograph data with typical road maps. It also displays road and feature labels.
- **Satellite:** Satellite type displays satellite photograph data, but doesn't display road and feature labels.
- **Terrain:** This type displays photographic data. This includes colors, contour lines and labels and perspective shading.
- **None:** This type displays an empty grid with no tiles loaded.

Syntax of different types of map

```
googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

Methods of Google map

Google map API provides several methods that help to customize Google map. These methods are as following:

Methods	Description
addCircle(CircleOptions options)	This method add circle to map.
addPolygon(PolygonOptions options)	This method add polygon to map.
addTileOverlay(TileOverlayOptions options)	This method add tile overlay to the map.
animateCamera(CameraUpdate update)	This method moves the map according to the update with an animation.
clear()	This method removes everything from the map.
getMyLocation()	This method returns the currently displayed user location.
moveCamera(CameraUpdate update)	This method reposition the camera according to the instructions defined in the update.
setTrafficEnabled(boolean enabled)	This method set the traffic layer on or off.
snapshot(GoogleMap.SnapshotReadyCallback callback)	This method takes a snapshot of the map.
stopAnimation()	This method stops the camera animation if there is any progress.

To use google maps in our android applications we need to install **Google Play Services** SDK in our Android Studio because google made **Google Mas API** as a part of Google Play Services SDK.

To install Google Play Services, open **Android Studio** → Go to **Tools** menu → **Android** → click **SDK Manager**, then new window will open in that select **SDK Tools** tab → Select **Google Play Services** → click **OK.** Once we are done with **Google Play Services** installation in android studio, now we will see how to integrate google map in android app.

1. Create an Android project and select Google maps activity.
2. Get a Google Map API key: Once project is created, Android Studio will open **google_maps_api.xml** and **MapsActivity.java** files in the editor. The **google_maps_api.xml** file will contains an instructions to generate a Google Maps API key to access Google Maps servers. Copy the link provided in the **google_maps_api.xml** file and paste the console URL in browser and it will take you to **Google API Console**. Follow the instructions to create a **new project** on **Google API Console**. Once we click on **continue** it will create a project and Google Maps Android API will be enabled. Now we need to create an API key to call the API for that click on **Create API Key**. Once we click on **Create API Key**, it will create an API key.

3. Now copy the API Key, go back to android studio and paste the API key into the **<string>** element in **google_maps_api.xml** file like as shown below.

```
<string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">AIzaSyC
KPTaBv41DKqr9qxMPWOQAsqp0Q4NHMER</string>
```

4. We need to modify AndroidManifest.xml file by adding some user permission like:

- INTERNET: To determine if we are connected to the internet or not.
- ACCESS_FINE_LOCATION: To determine a user’s location using GPS.
- ACCESS_COARSE_LOCATION: To determine user”s location using Wi-Fi and mobile data.

AndroidManifext.xml

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

    <!--
         The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
         Google Maps Android API v2, but you must specify either coarse or fine
         location permissions for the 'MyLocation' functionality.
    -->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>

    <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">
     <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>
```

activity_maps.xml

We are going to implement the android MapView in a Fragment and subsequently add the Fragment to the MainActivity class.

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

MapsActivity.java

In MapsActivity.java class to get the Google map object, we need to implement the OnMapReadyCallback interface and override the onMapReady() callback method, these are called when the map is ready to be used.

Zoom Controls: We can control Zooming gestures on the map by using built-in *setZoomControlsEnabled()* function. These can be enabled by calling:

```
googleMap.getUiSettings().setZoomGesturesEnabled(true); // false to disable
```

Add Marker on Map: **We can place a marker to display location on the map by** addMarker()**method.**

```
LatLng addisababa= new LatLng(9.02497, 38.74689);
mMap.addMarker(new MarkerOptions().position(addisababa).title("Marker in
Addis Ababa"));
```

MapsActivity.java

```
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

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);
    }

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

        LatLng addisababa = new LatLng(9.02497, 38.74689);
        mMap.addMarker(new MarkerOptions().position(addisababa).title("Addis Ababa in
Ethiopia"));
        mMap.moveCamera(CameraUpdateFactory.newLatLng(addisababa));

    }
}
```

build.gradle

In the app build.gradle file, we will add a Map dependency library. The modified version of the file is as shown:

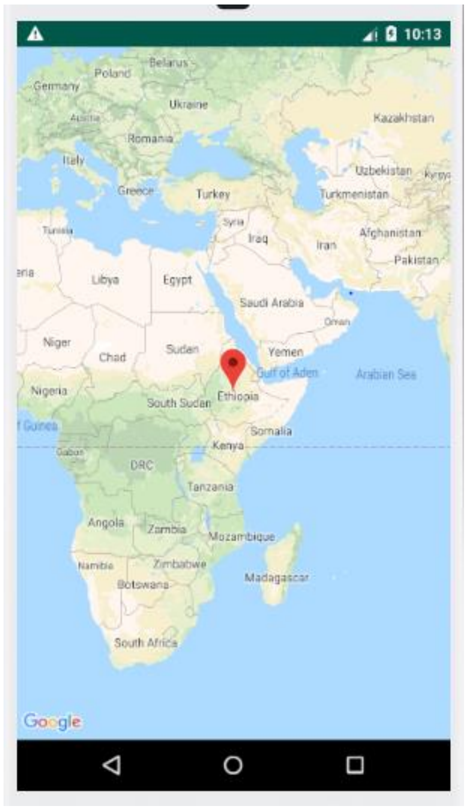
```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.example.mygooglemapapplication"
        minSdkVersion 14
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.google.android.gms:play-services-maps:16.0.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

}
```

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity. The output is as shown below.


10:13
Google