

COM413 – Mobile App Development – Workbook Practical 4

Introduction

This Practical is based around using GPS and Google Maps within an application. There are several ways of using GPS just as there are many reasons why we might use GPS.

GPS (Global Positioning System) provides geolocation and time information to a GPS receiver anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites.

The GPS does not require the user to transmit any data, and it operates independently of any telephonic or internet reception, though these technologies can enhance the usefulness of the GPS positioning information. The GPS provides critical positioning capabilities to military, civil, and commercial users around the world. The United States government created the system, maintains it, and makes it freely accessible to anyone with a GPS receiver.

Google Maps is a web mapping service developed by Google. It offers satellite imagery, aerial photography, street maps, 360° interactive panoramic views of streets, real-time traffic conditions, and route planning for traveling by foot, car, bicycle and air, or public transportation.

So, today's practical will be based on creating an app capable of using GPS and incorporating Google Maps.

Basic:

Create an app which will:

- Get a location using Google Maps
- Add markers to the display

Advanced:

Create an app which will:

- Get current location from user

Additional:

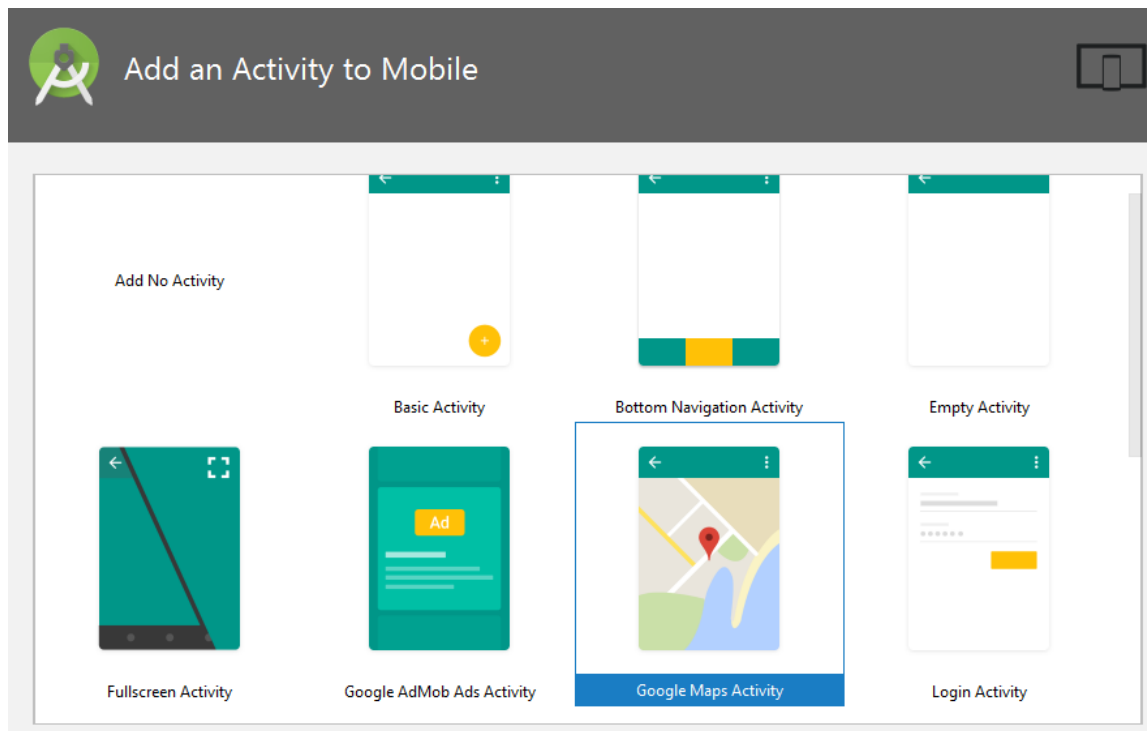
Create a transport tracker for a bus, package delivery service or even an app that allows you to store the location of your car and helps you get back to the car. You can use firebase to store locations, read your current location and give you a realtime update.

Basic Functionality:

In this practical, we will learn how to make a Google Maps App that detects a location. Google Maps will track a location and add a marker on it.

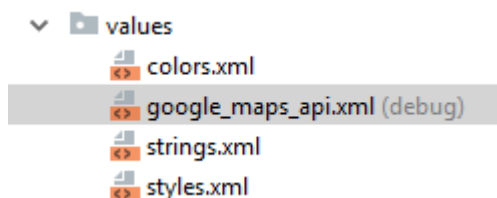
To get started, create a new project in Android Studio

Use the default Phone & Tablet options and select Google Maps Activity when prompted for the Activity Type:



Call it: Google Maps App.

In the Configure Activity screen, keep the default values and click Finish. Now you will be able to see `google_maps_api.xml` in the values folder.



Configuring Google Play Services

Open `google_maps_api.xml`. Here you will find a lot of information along with a link.

```
MapsActivity.java x google_maps_api.xml x
resources
<resources>
<!--
TODO: Before you run your application, you need a Google Maps API key.

To get one, follow this link, follow the directions and press "Create" at the end:
https://console.developers.google.com/flows/enableapi?apiid=maps\_android\_backend&keyType=CLIENT\_SIDE\_ANDROID&r=CD:39:72:69:C4:4A:27:93:5B:F7:22:A1:01:10:19:1E:14:53:B0:86:3Bcom.example.se245547.googlemapsapp

You can also add your credentials to an existing key, using this line:
CD:39:72:69:C4:4A:27:93:5B:F7:22:A1:01:10:19:1E:14:53:B0:86:com.example.se245547.googlemapsapp

Alternatively, follow the directions here:
https://developers.google.com/maps/documentation/android/start#get-key

Once you have your key (it starts with "AIza"), replace the "google_maps_key"
string in this file.
-->
<string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">YOUR_KEY_HERE</string>
</resources>
```

Copy-Paste this link into your web browser. Also, make a Gmail account through which you will configure google play services.

Register your application for Maps SDK for Android in Google API Console

Google API Console allows you to manage your application and monitor API usage.

Select a project where your application will be registered

You can use one project to manage all of your applications, or you can create a different project for each application.

Create a project

Terms of Service

☒ I agree to the Google Cloud Platform Terms of Service, and the terms of service of any applicable services and APIs.

Country of residence

United Kingdom

I would like to receive periodic emails on news, product updates and special offers from Google Cloud and Google Cloud Partners.

☐ Yes ☒ No

Agree and continue

Now at the browser choose **Create a Project** and click **Agree and Continue**. Following screen will be displayed. Clicking **API key**.

API Keys

<input type="checkbox"/>	Name	Creation date ↓	Restrictions	Key
<input type="checkbox"/>	<div>API key 1</div>	Oct 1, 2020	Android apps	AIzaSyAnwa...KubGuYkCHQ

OAuth 2.0 Client IDs

<input type="checkbox"/>	Name	Creation date ↓
No OAuth clients to display		

Service Accounts


<input type="checkbox"/>	Email	Name ↑	Usage with all services (last 30 days) ⓘ
No service accounts to display			

From here we want to restrict the key:

API restrictions

API restrictions specify the enabled APIs that this key can call

- ☐ Don't restrict key
This key can call any API
- ☒ Restrict key


 Type to filter

☒ Maps SDK for Android

Note: It may take up to 5 minutes for settings to take effect

SAVE CANCEL

Click on Save. Now a key will be created that you shall copy and paste in `google_maps_api.xml`.

Key	↑
AIzaSyB...	

Copy the key and paste it in place where **YOUR_KEY_HERE** is written:

```
<string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">YOUR_KEY_HERE</string>  
</resources>
```

The code inside `google_maps_api.xml` is now complete.

If you go inside **AndroidManifest.xml** then this key will be displayed in meta tags.

Here you need to add permissions for accessing location of device. The required permission should be as follows:

INTERNET – If we are connected to Internet or not. **ACCESS_COARSE_LOCATION** – To determine user's location using WiFi and mobile. It will give us approximate location.

OpenGL ES V2 – Required for Google Maps V2

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<!--
    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" />

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
```

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

    // Add a marker in Sydney and move the camera
    LatLng sydney = new LatLng(-34, 151);
    mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
}
```

Running Google Maps App for the first time:

We can run this App on an Emulator or a mobile device. Now click on **Run**. You will see the following image on an Emulator or a mobile device:



So, our basic Google Maps App is running fine with a marker placed at a particular location.

Note: Please see your build.gradle (Module.app) file. It should have following code:

```
dependencies {  
    .....  
    implementation 'com.google.android.gms:play-services-maps:17.0.0'  
    .....  
}
```

The line '**com.google.android.gms:play-services-maps:17.0.0**' is responsible for inserting Google Play Services. Please make sure this line is present in build.gradle (Module.app)

Location-Based Services (LBS)

You have seen the explosive growth of mobile apps in recent years. One category of apps that is very popular is Location-Based Services, commonly known as LBS. LBS apps track your location, and might offer additional services such as locating amenities nearby, offering suggestions for route planning, and so on.

This document shows you how to make use of Google Maps in your android application, as well as how to manipulate the map view programmatically. In addition, you find out how to obtain your geographical location using the LocationManager class available in the Android SDK.

In the previous practical, we have created a project with name “Google Maps App”. As we already developed basic Google Map application in the “Google Maps App” project, we are going to use “Google Maps App” project to continue with this practical.

Open the “Google Maps App” in the Android Studio and run the program in an emulator or a device. You will be seeing an output similar to the figure below:



Please go through this document and amend/change the code in the relevant section of the MapsActivity.java.

Changing Views

By default, Google Maps is displayed in *map view*, which is basically drawing of streets and places of interest. You can also set Google Maps to display in *satellite view* using the `setMapType()` method of the `GoogleMap` class:

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

    // Add a marker in Sydney and move the camera
    LatLng sydney = new LatLng(-34, 151);
    mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));

    mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
}
```

The figure below shows Google Maps displayed in satellite view.



Note: More details about Map Objects can be found via the link below

<https://developers.google.com/maps/documentation/android-api/map>

Navigating to a specific Location

By default, Google Maps displays the map of Australia when it is first loaded. However, you can set Google Maps to display a particular location. To do so, you can use the `moveCamera()` method of `GoogleMap` class.

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

    // Add a marker in Belfast City Airport and move the camera
    LatLng bca = new LatLng( 54.617611, -5.8718491);
    mMap.addMarker(new MarkerOptions().position(bca).title("Belfast City Airport"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(bca));
}
```

Run the program and touch on the red pin. When the map is loaded, observe that it now animates to a particular location – Belfast City Airport.



The above figure shows the location of Belfast City Airport, but we need to increase the zoom level, so that we can view the location of the Belfast city Airport more clearly and precisely.

How to increase the zoom level?

Instead of `newLatLng()`, we need to use `newLatLngZoom()`. This will return a `CameraUpdate` that moves the center of the screen to a latitude and longitude specified by a `LatLng` object, and moves to the given zoom level. The range of the zoom level will be 2.0 to 21.0.

For example:

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

    // Add a marker in Belfast City Airport and move the camera
    LatLng bca = new LatLng( 54.617611, -5.8718491);
    mMap.addMarker(new MarkerOptions().position(bca).title("Belfast City Airport"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(bca));
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(bca, 12.0f));
}
```

Run the program. When the map is loaded, you can clearly see the location of Belfast City Airport.



Getting the Location that Was Touched

After using google Maps for a while, you might want to know the latitude and longitude of a location corresponding to the position on the screen that was just touched. Knowing this information is very useful because you can determine a location's address – a process known as reverse geocoding (you find out how this is done in the next section).

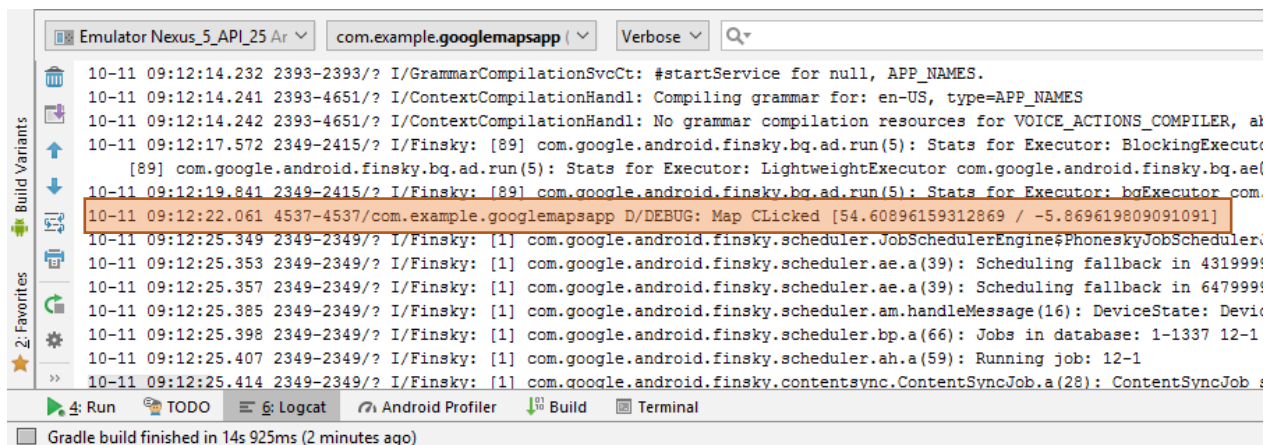
To get the latitude and longitude of a point on the Google Map that was touched, you must set an `onMapClickListener`:

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

    // Add a marker in Belfast City Airport and move the camera
    LatLng bca = new LatLng( 54.617611, -5.8718491);
    mMap.addMarker(new MarkerOptions().position(bca).title("Belfast City Airport"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(bca));
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(bca, 12.0f));

    mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
        @Override
        public void onMapClick(LatLng point) {
            Log.d( tag: "DEBUG", msg: "Map CLicked [" + point.latitude + " / " + point.longitude + "]);
        }
    });
}
```

If you run the program and touch anywhere on the google map screen then you should see a logcat entry similar to this in the **Logcat**.



Geocoding and Reverse Geocoding

As mentioned in the preceding section, if you know the latitude and longitude of a location, you can find out its address using a process known as reverse geocoding. Google Maps in Android supports reverse geocoding via the Geocoder class. The following code snippet shows how you can retrieve the address of a location just touched using the `getFromLocation()` method:

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

    // Add a marker in Belfast City Airport and move the camera
    LatLng bca = new LatLng(54.617611, -5.8718491);
    mMap.addMarker(new MarkerOptions().position(bca).title("Belfast City Airport"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(bca));
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(bca, 12.0f));

    mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
        @Override
        public void onMapClick(LatLng point) {
            Geocoder geoCoder = new Geocoder(getBaseContext(), Locale.getDefault());
            try{
                List<Address> addresses = geoCoder.getFromLocation(point.latitude, point.longitude, maxResults: 1);
                Address address = addresses.get(0);
                String add = "";

                if (addresses.size() > 0){
                    add += address.getAddressLine(0) + "\n";
                }
                Toast.makeText(getBaseContext(), add, Toast.LENGTH_SHORT).show();
            } catch (IOException e){
                e.printStackTrace();
            }
        }
    });
}
```

The Geocoder object converts the latitude and longitude into an address using the `getFromLocation()` method. After the address is obtained, you display it using the Toast class. Keep in mind that pin will not move. In this example, we are only getting the address of a location that you touch. The figure below shows the application displaying the address of a location that was touched on the map.



If you wish to place a pin to the places you have touched then we need to include the following code in the onMapReady() method.

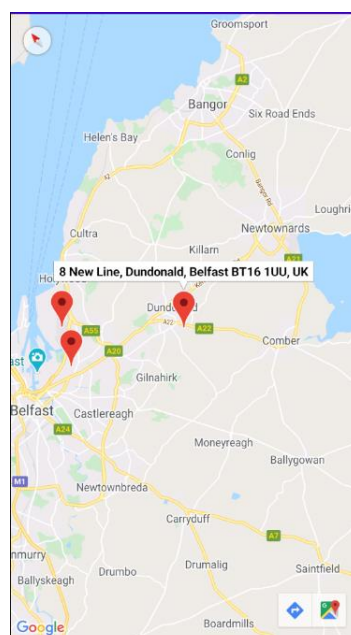
```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    // Add a marker in Sydney and move the camera
    LatLng bca = new LatLng( v: 54.617611, vl: -5.8718491);
    mMap.addMarker(new MarkerOptions().position(bca).title("Belfast City Airport"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(bca));
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(bca, v: 12.0f));

    mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
        @Override
        public void onMapClick(LatLng point) {
            Geocoder geoCoder = new Geocoder(getBaseContext(), Locale.getDefault());
            try{
                List<Address> addresses = geoCoder.getFromLocation(point.latitude, point.longitude, maxResults: 1);
                Address address = addresses.get(0);
                String add = "";

                if (addresses.size() > 0){
                    add += address.getAddressLine( index: 0) + "\n";
                    LatLng p = new LatLng(address.getLatitude(), address.getLongitude());
                    mMap.addMarker(new MarkerOptions().position(p).title(add));
                    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(p, v: 12.0f));
                }
                Toast.makeText(getBaseContext(), add, Toast.LENGTH_SHORT).show();
            } catch (IOException e){
                e.printStackTrace();
            }
        }
    });
}
```

The figures below show the application displaying the address and pin of a location that was touched.



If you know the address of a location but want to know its latitude and longitude, you can do so via geocoding. Again, you can use the Geocoder class for this purpose. The following code shows how you can find the exact location of the “Titanic Belfast” building by using the `getFromLocationName()` method:

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

    // Add a marker in Belfast City Airport and move the camera
    LatLng bca = new LatLng( v: 54.617611, vl: -5.8718491);
    mMap.addMarker(new MarkerOptions().position(bca).title("Belfast City Airport"));
    mMap.moveCamera(CameraUpdateFactory.newLatLng(bca));
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(bca, v: 12.0f));

    Geocoder geoCoder = new Geocoder(getBaseContext(), Locale.getDefault());
    try {
        String placeName = "Titanic Belfast";
        List<Address> addresses = geoCoder.getFromLocationName(placeName, maxResults: 1);
        Address address = addresses.get(0);

        if (addresses.size() > 0) {
            LatLng p = new LatLng(address.getLatitude(), address.getLongitude());
            mMap.addMarker(new MarkerOptions().position(p).title(placeName));
            mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(p, v: 12.0f));
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

The figure below shows the search result of “Titanic Belfast”.



Advanced Functionality:

Getting Location Data

Nowadays, mobile devices are commonly equipped with GPS receivers. Because of the many satellites orbiting the earth, you can use a GPS receiver to find your location easily. However, GPS requires a clear sky to work and hence does not always work indoors or places like a tunnel through a mountain.

Another effective way to locate your position is through *cell tower triangulation*. When a mobile phone is switched on, it is constantly in contact with base stations surrounding it. By knowing the identity of cell towers, it is possible to translate this information into a physical location through the use of various databases containing cell towers' identities and their exact geographical locations. The advantage of cell tower triangulation is that it works indoors, without the need to obtain information from satellites. However, it is not as precise as GPS because its accuracy depends on overlapping signal coverage, which varies quite a bit. Cell tower triangulation works best in densely populated areas where the cell towers are closely located.

The third method of locating your position is to rely on Wi-Fi triangulation. Rather than connect to cell towers, the device connects to a Wi-Fi network and checks the service provider against databases to determine the location serviced by the provider. Of the three methods described here, Wi-Fi triangulation is the least accurate.

On the Android platform, the SDK provides the LocationManger class to help your device determine the user's physical location. The following shows how this done in code.

Please make sure you have included the following permissions in the AndroidManifest.xml.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<!--
    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.ACCESS_COARSE_LOCATION" />

<application
    android:allowBackup="true"
```

We need to add the following in the MapsActivity.java:

```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

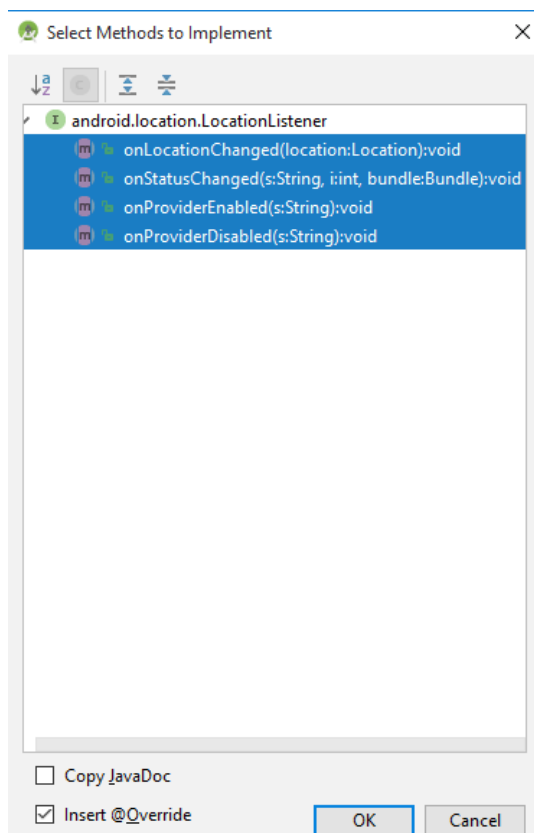
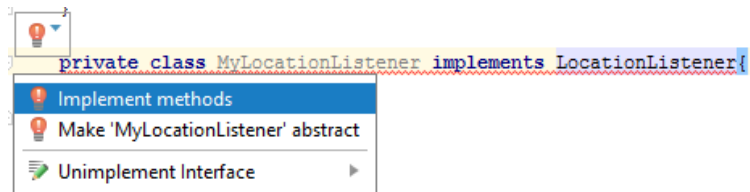
    private GoogleMap mMap;

    final private int REQUEST_COARSE_ACCESS = 123;
    boolean permissionGranted = false;
    LocationManager lm;
    LocationListener locationListener;
```

Now we need to create a MyLocationListner method to listen the current locations.

```
private class MyLocationListner implements LocationListener{  
  
}
```

The redline indicates that there is an error. We need to fix this error by clicking on implement methods:



Click OK. Now the method will look like as given below:


```

private class MyLocationListener implements LocationListener{

    @Override
    public void onLocationChanged(Location location) {

    }

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

    }

    @Override
    public void onProviderEnabled(String provider) {

    }

    @Override
    public void onProviderDisabled(String provider) {

    }

}

```

The following code needs to be included into the **onLocationChanged()**:

```

@Override
public void onLocationChanged(Location location) {
    if (location != null) {
        Toast.makeText(getBaseContext(),
            text: "Current Location : Lat: " + location.getLatitude() +
                " Lng: " + location.getLongitude(), Toast.LENGTH_LONG).show();
        LatLng p = new LatLng(location.getLatitude(), location.getLongitude());
        mMap.addMarker(new MarkerOptions().position(p).title("Current Location"));
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(p, 12.0f));
    }
}

```

The following code needs to be included into the **onMapReady()**:

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

    //---use the LocationManagers class to obtain locations data---
    lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
    locationlistener = new MyLocationListener();

    if (ActivityCompat.checkSelfPermission( context: this, ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED
        && ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION)
        != PackageManager.PERMISSION_GRANTED){
        ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.ACCESS_COARSE_LOCATION,
            Manifest.permission.ACCESS_FINE_LOCATION}, REQUEST_COARSE_ACCESS);
    }
    return;

}

else{
    permissionGranted = true;
}

if(permissionGranted){
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, minTimeMs: 0, minDistanceM: 0, locationlistener);
}
}
```

Two methods (i.e. **onRequestPermissionsResult**, **onPause**) for this application are:

Click ctrl + o:

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    switch (requestCode){
        case REQUEST_COARSE_ACCESS:
            if(grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED){
                permissionGranted = true;
                if (ActivityCompat.checkSelfPermission( context: this, ACCESS_FINE_LOCATION)
                    != PackageManager.PERMISSION_GRANTED
                    && ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION)
                    != PackageManager.PERMISSION_GRANTED){
                    return;
                }
                lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, minTimeMs: 0, minDistanceM: 0, locationlistener);
            }
            else{
                permissionGranted = false;
            }
            break;
        default:
            super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```

```

@Override
public void onPause() {
    super.onPause();

    //----remove the location listener----
    if (ActivityCompat.checkSelfPermission( context: this, android.Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED
        && ActivityCompat.checkSelfPermission( context: this, android.Manifest.permission.ACCESS_COARSE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, new String[]{
            android.Manifest.permission.ACCESS_COARSE_LOCATION
        }, REQUEST_COARSE_ACCESS);
        return;
    } else {
        permissionGranted = true;
    }
    if (permissionGranted) {
        lm.removeUpdates(locationListener);
    }
}

```

Note: Use the Emulator map to set location for this application.



How It Works

In Android, location-based services are provided by the LocationManager class, located in the android.location package. Using the LocationManager class, your application can obtain periodic updates of the device's geographical locations, as well as fire an intent when it enters the proximity of a certain location.

In the MapsActivity.java file, you first check for permission to use the Coarse Locations. Then you obtain a reference to the LocationManager class using the getSystemService () method. You do this in the onCreate () method:

```
//---use the LocationManager class to obtain locations data---
lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
locationListener = new MyLocationListener();
```

Next, you create an instance of the MyLocationListener class, which you define later in the class.

The MyLocationListener class implements the LocationListener abstract class. You need to override four methods in this implementation:

onLocationChanged (Location location) – Called when the location has changed.

onProviderDisabled (String provider) – Called when the provider is disabled by the user

onProviderEnabled (String provider) – Called when the provider is enabled by the user

onStatusChanged (String provider, int status, Bundle extras) – Called when the provider status changes

In this example, you're more interested in what happens when a location changes, so you write your code in the onLocationChanged () method. Specifically, when a location changes, you display a small dialog on the screen showing the new location information: latitude and longitude. You show this dialog using the Toast class:

```
@Override
public void onLocationChanged(Location location) {
    if (location != null) {
        Toast.makeText(getBaseContext(),
            text: "Current Location : Lat: " + location.getLatitude() +
                " Lng: " + location.getLongitude(), Toast.LENGTH_LONG).show();
        LatLng p = new LatLng(location.getLatitude(), location.getLongitude());
        mMap.addMarker(new MarkerOptions().position(p).title("Current Location"));
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(p, 12.0f));
    }
}
```

In the preceding method, you also navigate the map to the location that you have received.

To be notified whenever there is a change in location, you needed to register a request for location changes so that your program can be notified periodically. You do this via the requestLocationUpdates () method:

```
if(permissionGranted){
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, minTimeMs: 0, minDistanceM: 0, locationlistener);
}
}
```

The requestLocationUpdates () method takes four arguments:

Provider – the name of the provider with which you register. In this case, you are using NETWORK to obtain your geographical location data.

minTime – The minimum time interval for notifications, in milliseconds. 0 indicates that you want to be continually informed of location changes.

MinDistance – the minimum distance interval for notifications, in meters. 0 indicates that you want to be continually informed of location changes.

Listener – An object whose onLocationChanged () method will be called for each location update.

Finally, in the onPause () method, you remove the listener when the activity is destroyed or goes into the background (so that the application no longer listens for changes in location, thereby saving the battery of the device). You do that using the removeUpdates() method.

```
@Override
public void onPause() {
    super.onPause();

    //----remove the location listener----
    if (ActivityCompat.checkSelfPermission( context: this, android.Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED
        && ActivityCompat.checkSelfPermission( context: this, android.Manifest.permission.ACCESS_COARSE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, new String[]{
            android.Manifest.permission.ACCESS_COARSE_LOCATION
        },REQUEST_COARSE_ACCESS);
        return;
    } else {
        permissionGranted = true;
    }
    if (permissionGranted) {
        lm.removeUpdates(locationListener);
    }
}
```

Based on the above code, when click on the red pin, it will only show a message – “Current Location”. If you want to change the colour of the pin to Blue and display the address of the current location then you need to include the following code:

```
@Override
public void onLocationChanged(Location location) {
    if (location != null){
        Toast.makeText(getBaseContext(), text: "Current Location: Lat:" + location.getLatitude() + "Long:" + location.getLongitude(),Toast.LENGTH_LONG).show();
        LatLng p = new LatLng(location.getLatitude(), location.getLongitude());

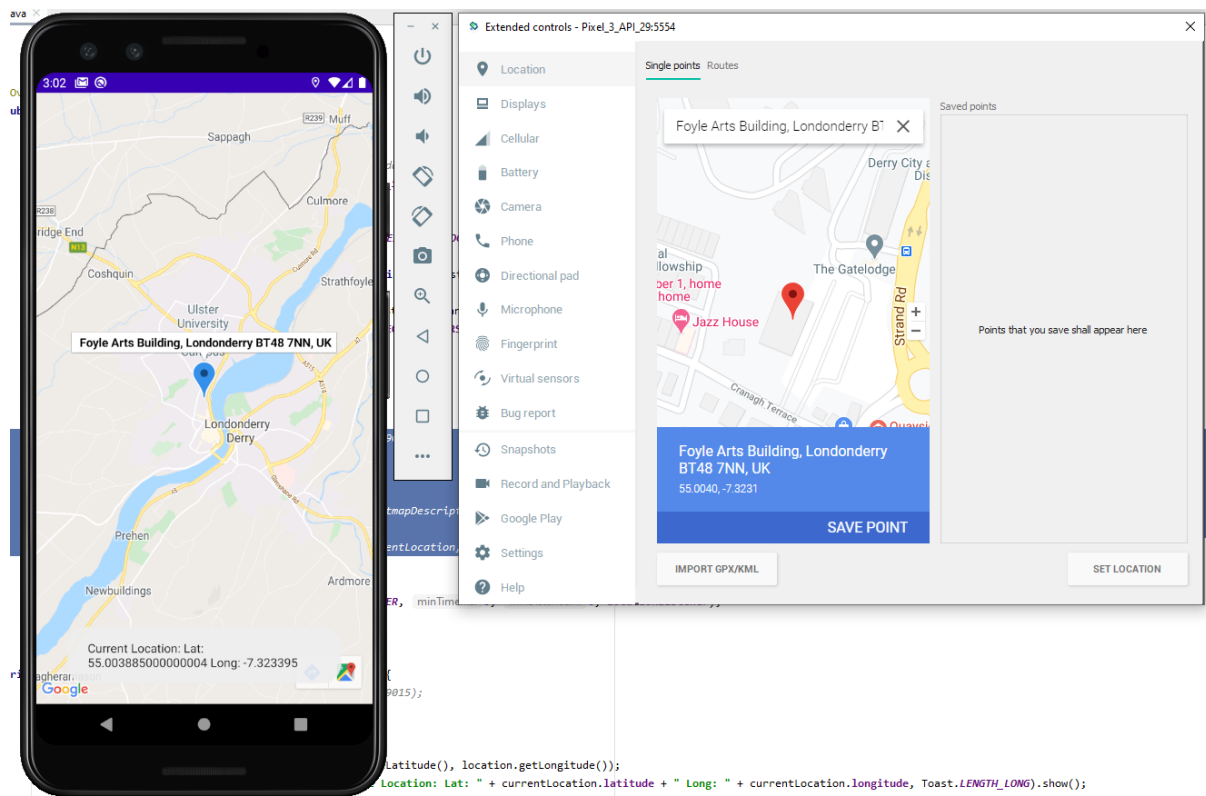
        Geocoder geocoder = new Geocoder(getBaseContext(), Locale.getDefault());
        List<Address> addresses = null;
        String add = "";
        try{
            addresses = geocoder.getFromLocation(location.getLatitude(), location.getLongitude(), maxResults: 1);
            Address address = addresses.get(0);

            if (addresses.size() > 0 ){
                add += address.getAddressLine( index: 0);
            }
        }catch (IOException e){
            e.printStackTrace();
        }

        mMap.addMarker(new MarkerOptions()
            .position(p)
            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE))
            .title(add));

        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(p, v: 12.0f));
    }
}
```

Run the program. You will see an output similar to the one shown below:



Additional Functionality:

Create a transport tracker for a bus, package delivery service or even an app that allows you to store the location of your car and helps you get back to the car.

You can use firebase to store locations, read your current location and give you a real time update on the other item to be located.

This can be done in a number of ways:

1. You could store locations in a database, read the longitude and latitude of the item and then read the users current location. This would work if you have many things like buses.
2. You could combine the Basics and Advanced in this practical. Read the current location of the user and hardcode the location of an item, let's say a package.

The goal of the advanced section is to show two markers on the screen. You'll want to use different coloured markers to separate the two items.

Finally, you want to give the user some real time information. You could look at distance, or draw a line on the map showing the user a path to the item, or you could look at the estimated time to get to the address, this part is up to you.

