Lesson 11: Location-Aware Apps: Using GPS and Google Maps

Introduction

Most of the mobile applications nowadays rely on users' Geo-location and web mapping services. GPS is considered one of the most accurate Geo-location providers. In order to increase users' experience of location awareness, geo-coordinates should be represented graphically, and this can be achieved by using web-mapping services such as Google Maps.

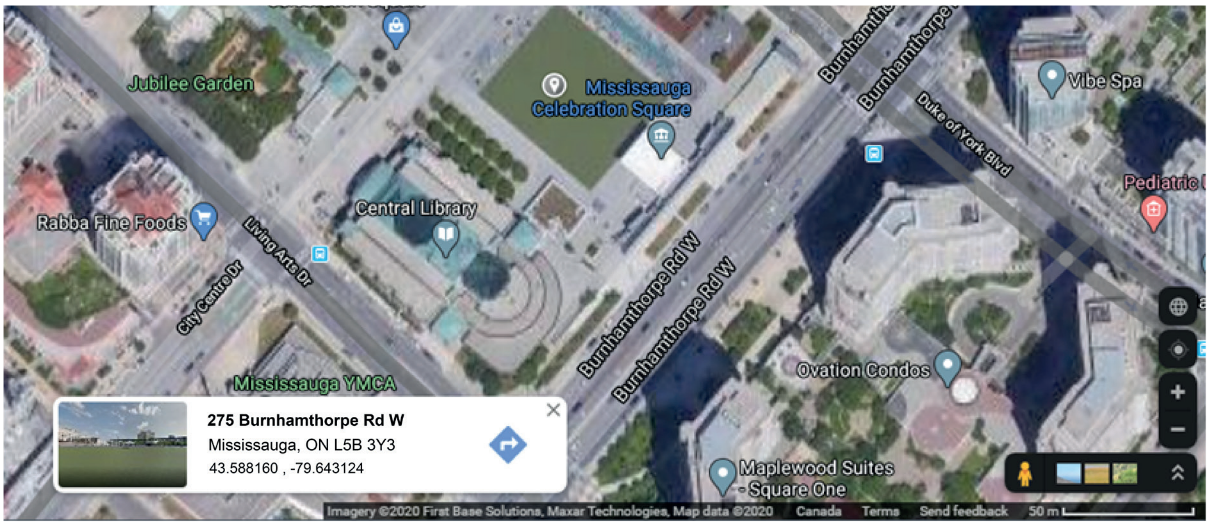
This lesson describes the different techniques used to capture geo-coordinates especially the GPS. It also covers some important topics related to Google Maps such as adding maps to your Android application, drawing shapes over maps, adding markers, capturing app users' location, etc...

What is GPS and how does it work?

GPS or Global Positioning System is a navigation system based on satellites. At over 20,000 kilometers above sea level (20,180 Kilometers), there is a constellation of satellites, each orbiting the Earth every 11 hours and 58 minutes. These satellites are continually sending data down to the Earth, which in turn is being received by GPS receivers such as mobile phones, allowing the device to calculate its position on earth.

For example, open Google maps: google.com/maps web site, then check your home address on Google maps.

The following figure displays the Latitude and Longitude values as follows:
Latitude: 43.588160
Longitude: -79.643124

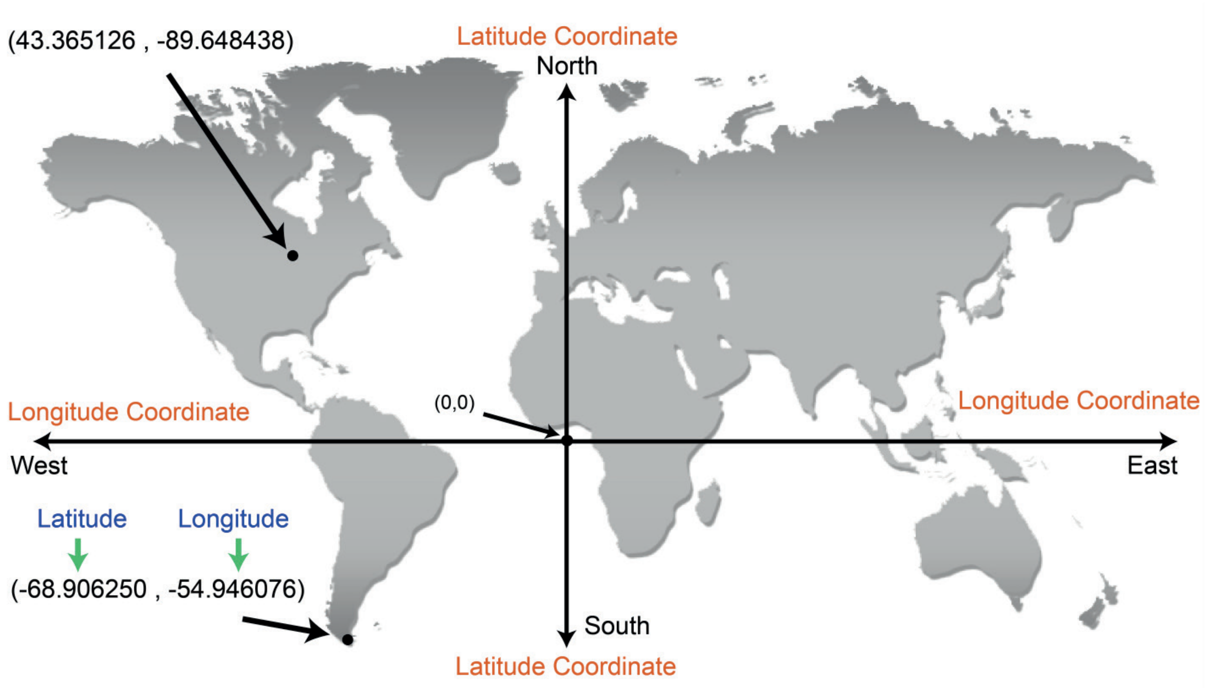


275 Burnhamthorpe Rd W
Mississauga, ON L5B 3Y3
43.588160 , -79.643124

Latitude and Longitude

The GPS Coordinates depend on the **latitude** and **longitude** values. Latitude and Longitude are the units that represent the coordinates in a geographic coordinate system. Just like every actual house has an address (which includes the number, the name of the street, the city, etc.), every single point on the surface of earth can be specified by ***latitude*** and ***longitude*** coordinates. Therefore, by using latitude and longitude, you can specify virtually any point on earth.

The following illustration displays the latitude and longitude coordinates:



As an Android developer, you can use the Geo-location term which is an identification or estimation of the real-world geographic location of an object, such as a radar source, mobile phone, or Internet-connected computer terminal. In its simplest form, a geo-location involves the generation of a set of geographic coordinates (Latitude and Longitude) and is closely related to the use of positioning systems.

The connection between a GPS satellite and receivers (such as smart devices) is a one-way connection; receivers don't send any information to satellites. The connection works through the use of a procedure called *Trilateration*.

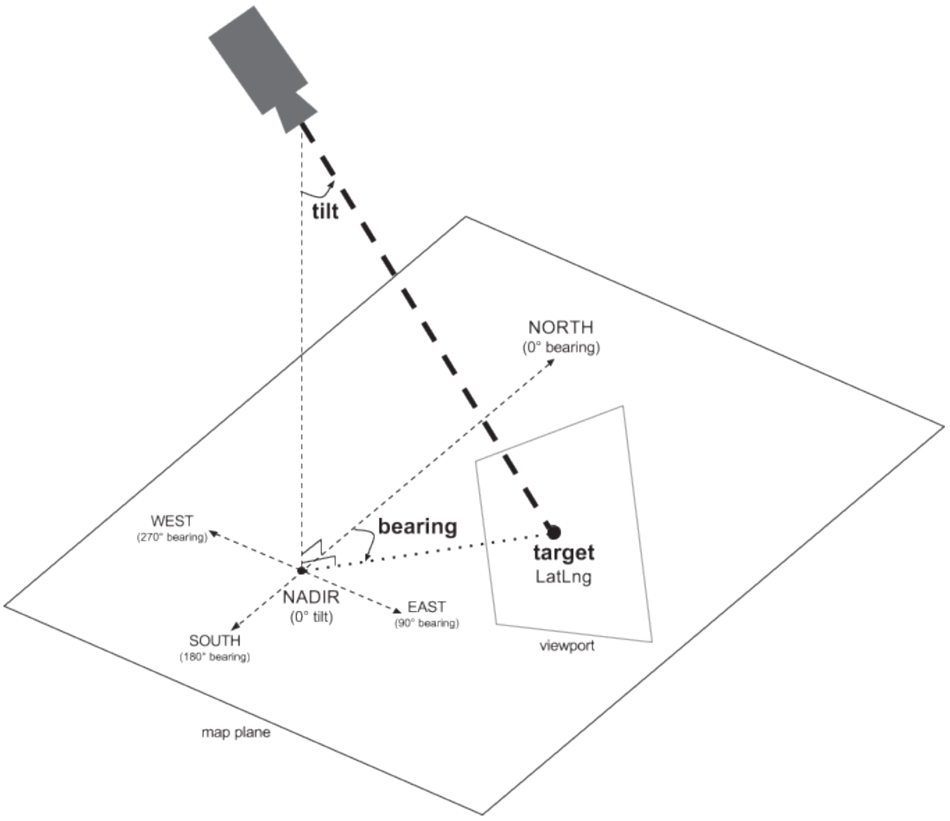
There are various types of navigational satellite systems. The most important one is **NAVSTAR**, which is basically used in all mobile devices. In **NAVSTAR** system, every single point on Earth is covered by at least four GPS satellites, which is enough to calculate the device geo-location coordinates.

As mentioned before, GPS is one of the most accurate geo-location systems. It has accuracy between 7.8~3 meters.

The Android system provides a reliable API that helps developers capture the user's coordinates using the GPS and other location service providers, which will be listed in the next topic.

Camera Position

The map view is modeled as a **camera** looking down on a flat plane. The position of the camera (and hence the rendering of the map) is specified by the following properties: **target** (latitude/longitude location), **bearing**, **tilt**, and **zoom**. You will use these camera attributes later in this lesson to control the camera position or motion.



Target (location)

The camera target is the location of the center of the map, specified as latitude and longitude co-ordinates.

Bearing (orientation)

The camera bearing is the direction in which a vertical line on the map points, measured in degrees clockwise from the north. Someone driving a car often turns a road map to align it with their direction of travel, while hikers use a map and a compass usually to orient the map so that a vertical line points north. The Maps API lets you change a map's alignment or bearing. In the "moving the camera" topic of this lesson, you will add this attribute to the map camera position and see how its value affects in changing the camera direction.

Tilt (viewing angle)

The tilt defines the camera's position on an arc between directly over the map's center position and the surface of the Earth, measured in degrees from the nadir (the direction pointing directly below the camera). When you change the viewing angle, the map appears in perspective, with far-away features appearing smaller, and nearby features appearing larger. In the "moving the camera" topic of this lesson, you will add this attribute to the map camera position and see how its value effects in changing the camera position.

Zoom

The zoom level of the camera determines the scale of the map. At larger zoom levels, more details can be seen on the screen, while at smaller zoom levels, more of the world can be seen on the screen. At zoom level 0, the scale of the map means the entire world has a width of approximately 256 dpi (density-independent pixels).

Adding Google Maps to an Android app

Android is Google's mobile app SDK for crafting high-quality native experiences on Android in record time.

The Google Maps Android API lets you display a Google map in your Android application. The maps are similar to those you see on Google Maps for mobile apps, and the API exposes many of the same features. With the **Google Maps Android plug-in**, you can add maps based on Google maps data to your application. The plug-in automatically handles access to the Google Maps servers, map display, and response to user gestures such as clicks and drags. You can also add markers to your map. These objects provide additional information for map locations, and allow the user to interact with the map.

Example:
In the following example, you are going to create a new Android project that includes a Google map. In this project you are going to learn about the classes and methods used to add and configure a Google map in Android applications, and you will also learn how to determine a specific location on the Google map (add a marker).

1- Open Android Studio, and then click **File → New → New Project.**

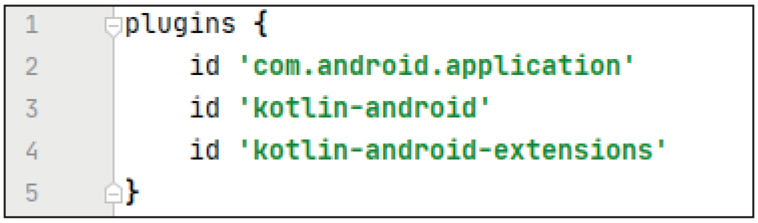
2- Select **Empty Activity**, and click **Next.**

3- Type: **Lesson11_GoogleMap** for the application name, then click **Finish**.

4- Before you start typing the Kotlin code in your app, check the **build.gardle (Module: Lesson11_GoogleMap.app)** file and be sure it has the Kotlin plugin. If not, add the following code:

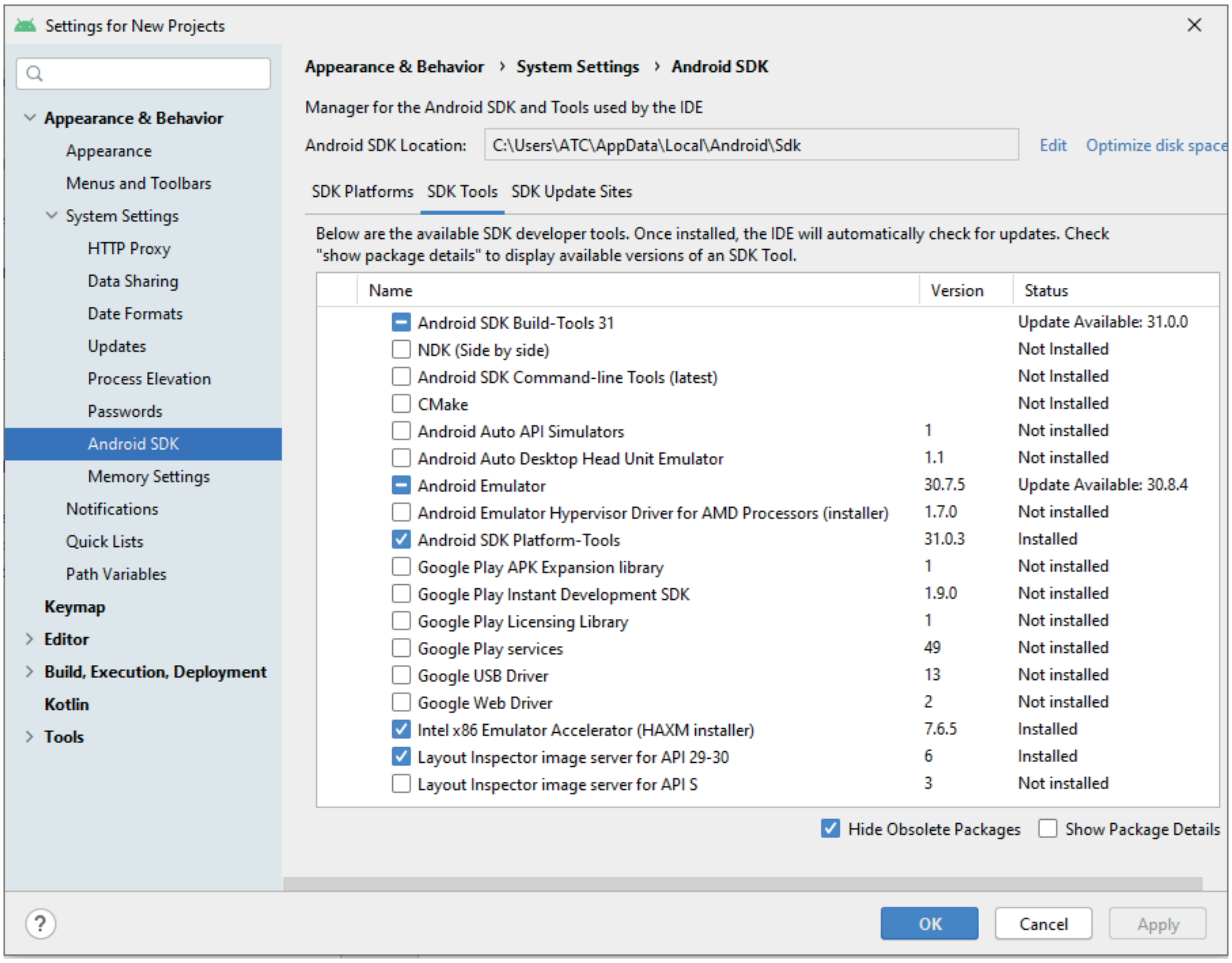
id 'kotlin-android-extensions'

Click the **Sync Now**. The configuration should be as follows:



```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

5- Install **Google Play service** on your Android Studio. Google Play Services is a layer of software that connects your apps with Google services. It runs on any Android device in the background of at all times and manages all Google services on this Android device such as determining the current GPS location of Android device on Google map. To install Google Play service on your Android Studio, open your Android Studio, click **SDK Manager** (**Tools → SDK Manager**), and as illustrated in the figure below, click **SDK Tools**, then select **Google Play services**, and click **Apply.**



Manager for the Android SDK and Tools used by the IDE

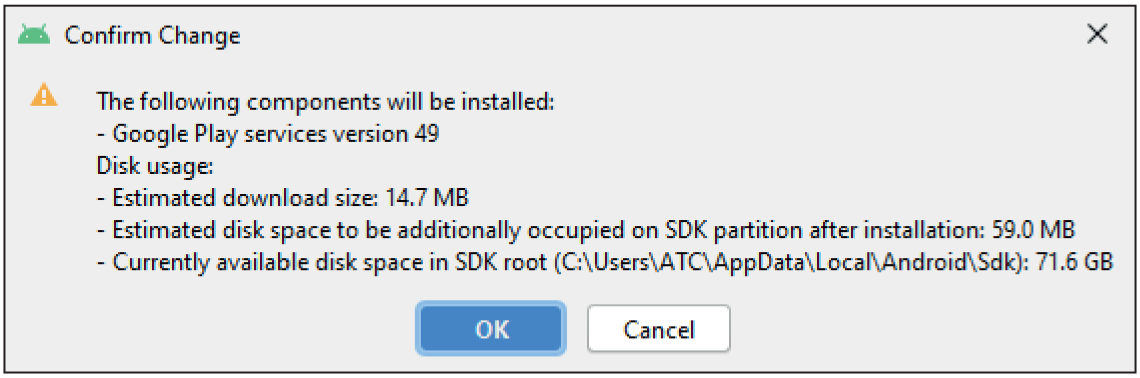
Android SDK Location: C:\Users\ATC\AppData\Local\Android\Sdk

Below are the available SDK developer tools. Once installed, the IDE will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
▬ Android SDK Build-Tools 31		Update Available: 31.0.0
☐ NDK (Side by side)		Not Installed
☐ Android SDK Command-line Tools (latest)		Not Installed
☐ CMake		Not Installed
☐ Android Auto API Simulators	1	Not installed
☐ Android Auto Desktop Head Unit Emulator	1.1	Not installed
▬ Android Emulator	30.7.5	Update Available: 30.8.4
☐ Android Emulator Hypervisor Driver for AMD Processors (installer)	1.7.0	Not installed
✓ Android SDK Platform-Tools	31.0.3	Installed
☐ Google Play APK Expansion library	1	Not installed
☐ Google Play Instant Development SDK	1.9.0	Not installed
☐ Google Play Licensing Library	1	Not installed
☐ Google Play services	49	Not installed
☐ Google USB Driver	13	Not installed
☐ Google Web Driver	2	Not installed
✓ Intel x86 Emulator Accelerator (HAXM installer)	7.6.5	Installed
✓ Layout Inspector image server for API 29-30	6	Installed
☐ Layout Inspector image server for API S	3	Not installed

✓ Hide Obsolete Packages ☐ Show Package Details

You should get the following dialog box. Click **OK.**



The following components will be installed:
- Google Play services version 49
Disk usage:
- Estimated download size: 14.7 MB
- Estimated disk space to be additionally occupied on SDK partition after installation: 59.0 MB
- Currently available disk space in SDK root (C:\Users\ATC\AppData\Local\Android\Sdk): 71.6 GB

After completing the installation process, click **Finish**, then click **OK**.

6- For each SDK that your app requires, include the dependency for that SDK. You should add Google Play dependencies to your Android project. To do that, open **build.gradle (Module:app)** file and add the following build rules under dependencies. These plugins are related to the Google Play services libraries.

```
implementation 'com.google.android.gms:play-services-location:18.0.0'
implementation 'com.google.android.gms:play-services-maps:17.0.1'
```

Click the **Sync Now**.

You can now begin developing features with Google Play services APIs (Google Map).

Map Fragment

You will configure your app to show the Google map in fragments. A Fragment represents a behavior or part of the user interface in one Activity. You can combine multiple fragments into a single activity to create a multi panel user interface, and you can also reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity that has its own life cycle, receives its own input events and that can be added or removed while the activity is running (It's like a "sub-activity" that you can reuse in different activities.).

7- Open the **activity_main.xml** file in **Design** mode and **delete** the "Hello World!" text.

8- Open the **activity_main.xml** file in the **Code** mode and Replace: **androidx.constraintlayout.widget.ConstraintLayout** in the first line with: **androidx.fragment.app.FragmentContainerView**

9- Add the following attribute to your fragment tag:

```
android:name="com.google.android.gms.maps.SupportMapFragment"
```

Your **activity_main.xml** file code should be as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"/>
```

10- Add a fragment **id** attribute with value: **myMap** to your **activity_main.xml** file. The full code of the **activity_main.xml** file will be as follows:

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

Later, you will use this fragment id: **myMap** to show where Google map will be displayed.

Getting a Google API key

Your app needs a Google API key to access or connect to the Google Maps servers. The type of key you need is an **API key** with restrictions for Android apps. This key is free, supports an unlimited number of users, and can be used with any of your Android apps that call the Google Maps database.

To get a Google Maps Android API key, you must register your Android project at the Google web site. Then, add that Google API key to your app configuration, then your app can use this API Key to connect and use all Google Play services.

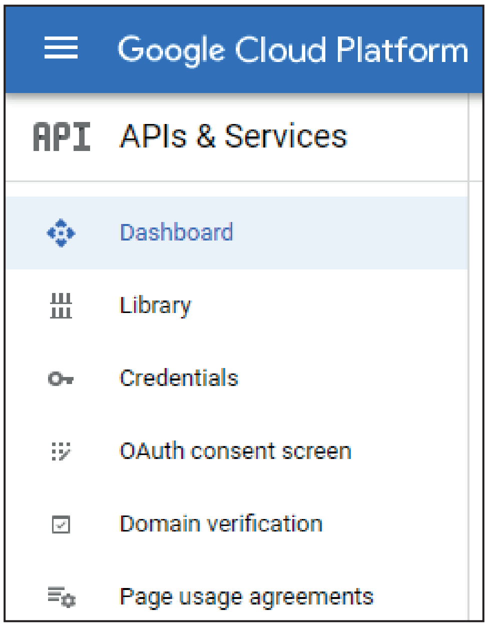
You may consider this API key like the password which makes the connection between your app and Google maps services or Google maps database. To get this API key, follow the steps below:

11- Go to : **https://console.developers.google.com**

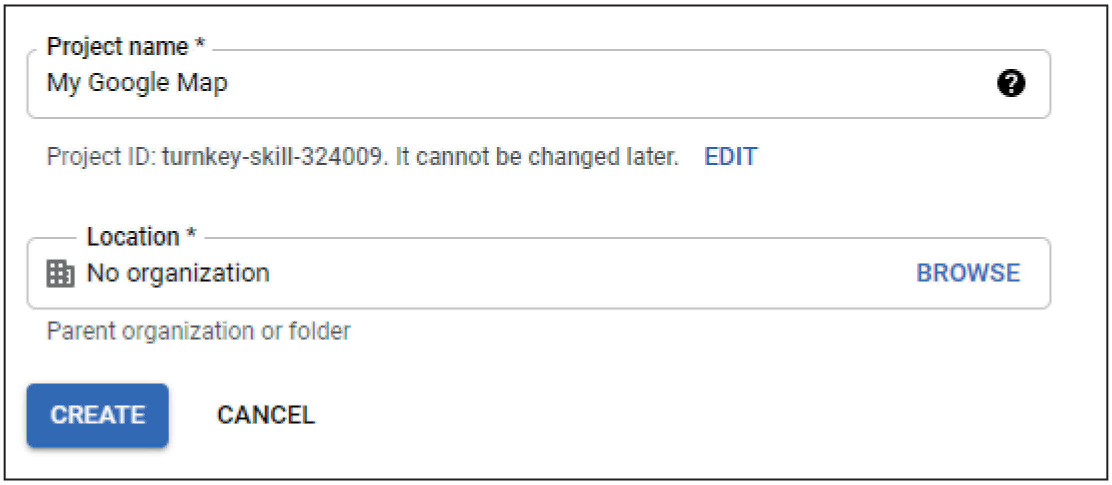
12- Login using your Gmail user name and password. If you don't have an account, create one.

If you get any dialog box related to the Google terms and services, select your country, click **I Agree**, then click **AGREE AND CONTINUE**.

You should get the panel below at the left side of Google cloud web site:



13- Click **CREATE PROJECT**, then you should get the following dialog box. Type: **My Google Map** (or any project name you want) for the project name and click **CREATE**.



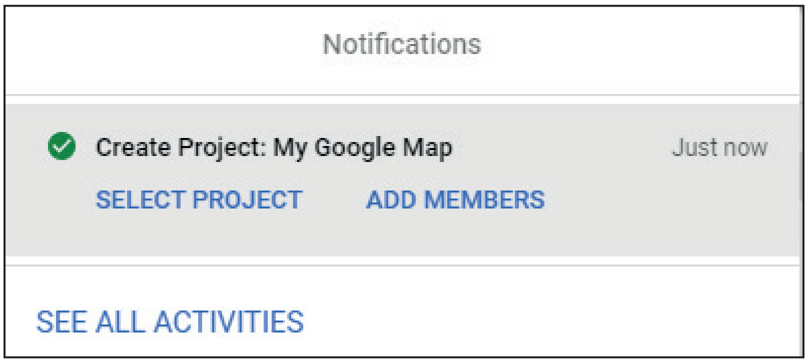
Project name *
My Google Map

Project ID: turnkey-skill-324009. It cannot be changed later. EDIT

Location *
No organization BROWSE

Parent organization or folder

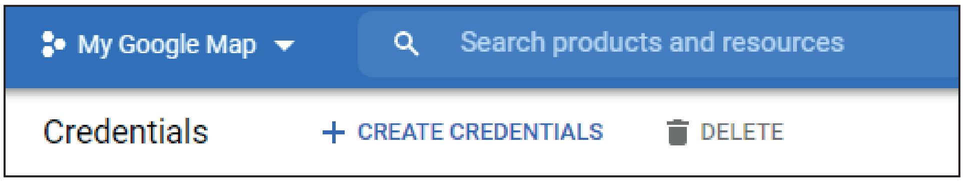
You should get the following notification:



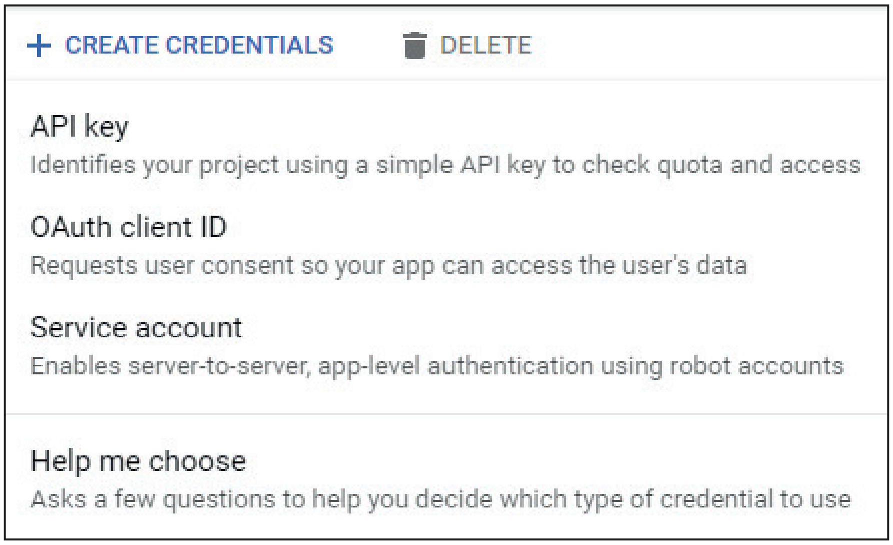
Notifications

Create Project: My Google Map Just now

14- Click **Credentials** from the left panel. Then, as illustrated in the figure below and at the top of your web browser, click **+ CREATE CREDENTIALS.**



You should get a menu as illustrated in the figure below. Select **API Key.**



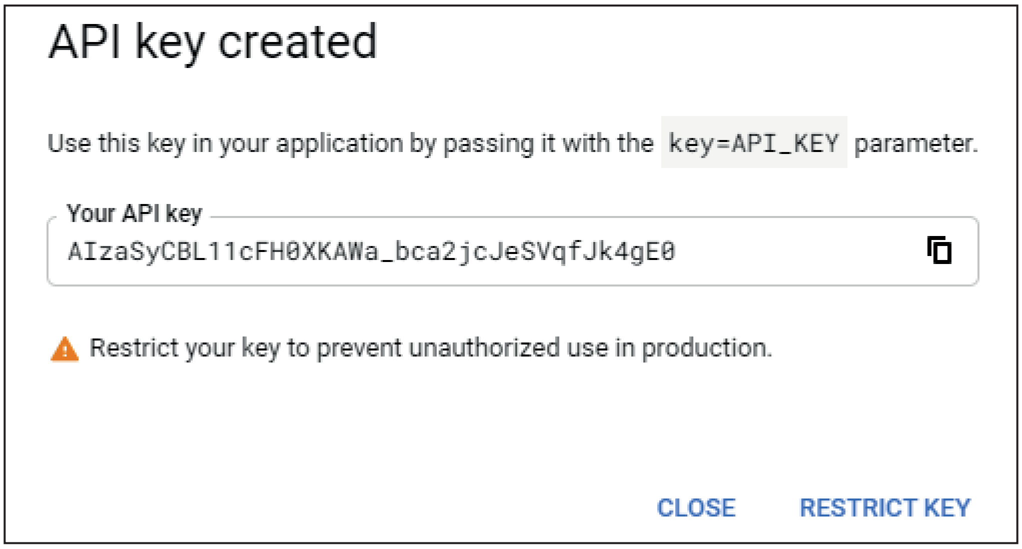
API key
Identifies your project using a simple API key to check quota and access

OAuth client ID
Requests user consent so your app can access the user's data

Service account
Enables server-to-server, app-level authentication using robot accounts

Help me choose
Asks a few questions to help you decide which type of credential to use

Then, you will get your API Key as illustrated in the following figure:



API key created

Use this key in your application by passing it with the `key=API_KEY` parameter.

Your API key
`AIzaSyCBL11cFH0XKAWa_bca2jcJeSVqfJk4gE0`

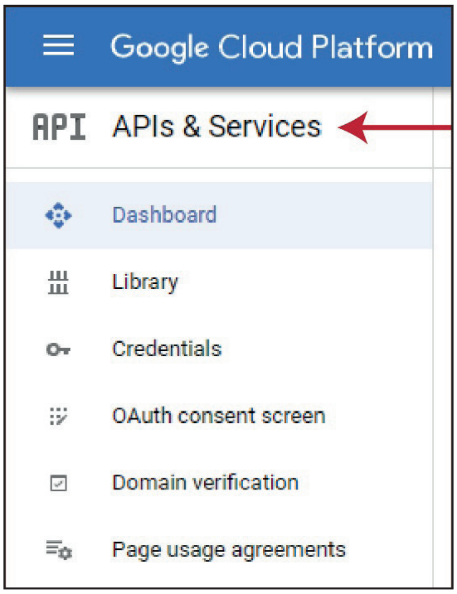
Restrict your key to prevent unauthorized use in production.

Copy your API key, and paste it in a separate Notepad file to use it later in your Android app configuration.

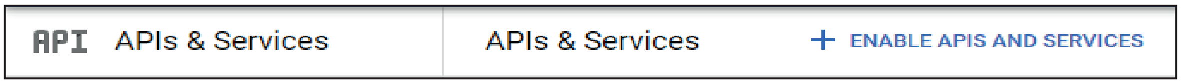
Click **Close**.

Note: The API key above is provided as an example only. You cannot use exactly the same key. Instead, use the one that you have generated in your Google API console.

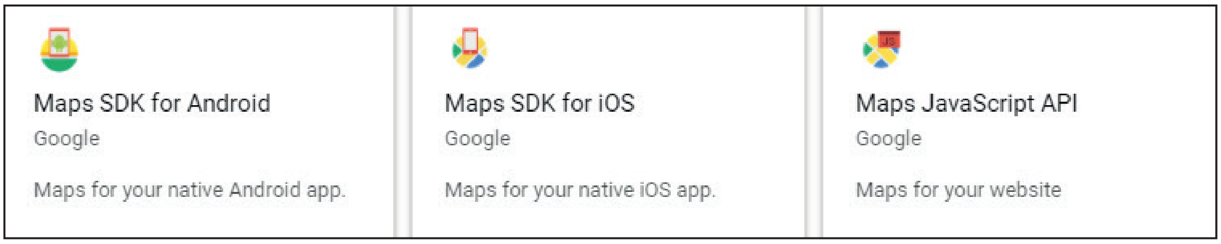
15- Now, to enable **Maps SDK** for your Android app, as illustrated in the figure below, click: **Google Cloud Platform → APIs & Services.**



Then, at the top of the web site and as illustrated in the figure below, click **+ ENABLE APIS AND SERVICES.**



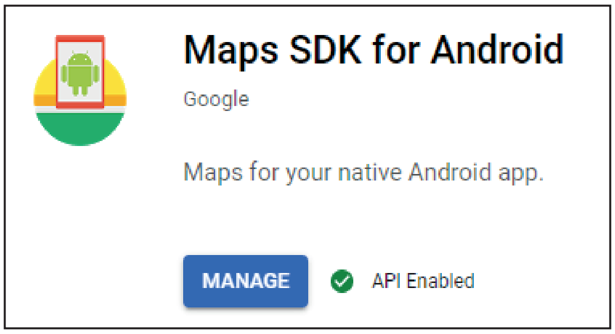
Then, you should get the figure below. Click **Maps SDK for Android**.


Maps SDK for Android
Google
Maps for your native Android app.

Maps SDK for iOS
Google
Maps for your native iOS app.

Maps JavaScript API
Google
Maps for your website

Then, click **Enable**. You should have the following status of your Maps configuration at Google cloud:


Maps SDK for Android
Google
Maps for your native Android app.

These are all the required configuration to make your Android app to have an access on Google Maps.

Now, back to your Android Studio to complete the other configuration to display and use Google map on your Android app.

16-To add your API key to your app, go back to Android Studio, open the **AndroidManifest.xml** and add the gray highlighted meta-data tag configuration as illustrated in the following XML code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidatc.lesson11_googlemap">

    <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/Theme.Lesson11_GoogleMap">

        <meta-data android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyCBL11cFH0XKAWa_bca2jcJeSVqfJk4gE0"/>

        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

17- Now, you are going to write the code which will show the Google map inside your fragment (**myMap**).

Open the **MainActivity** file, then add the following code inside `onCreate()` method:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.myMap) as SupportMapFragment
        mapFragment.getMapAsync(this)

    }
}
```

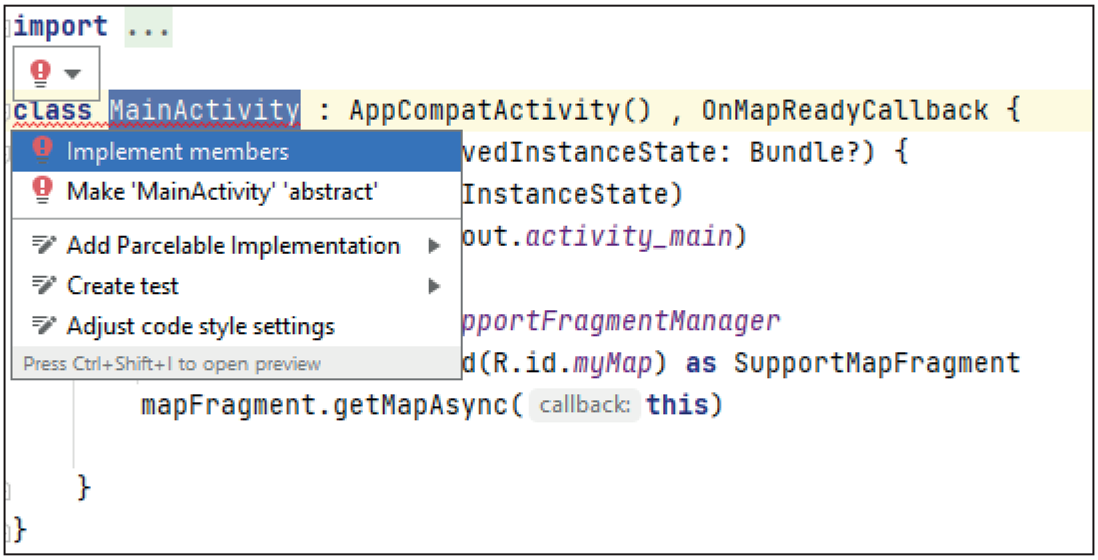
Double click the **`SupportMapFragment`**, click the red pop-up lamp, and select **Import.**

18- You will get a red line under: **`this`** in your code because you need to add the **`OnMapReadyCallback`** class. Add its configuration as illustrated in the gray highlighted part in the following code:

```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {

override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


val mapFragment = supportFragmentManager
.findFragmentById(R.id.myMap) as SupportMapFragment
        mapFragment.getMapAsync(this)
    }
}
```

Double click the **`OnMapReadyCallback`**, click the red pop-up lamp, and select **Import.**

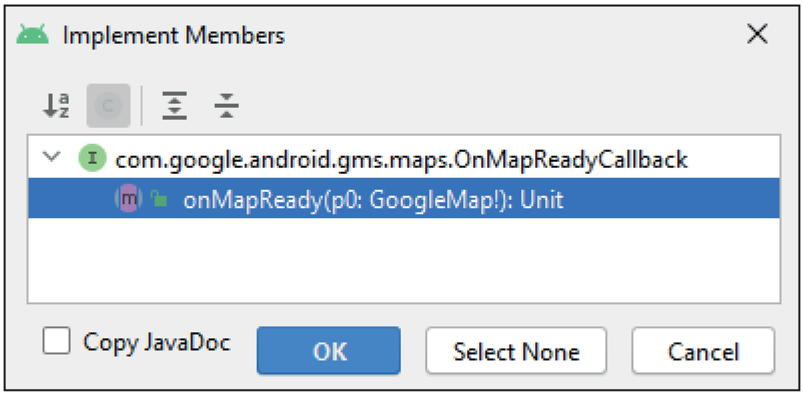
19- As illustrated in the figure below, double click the **MainActivity** class, click the red pop-up lamp, and select **Implement members**.



```
import ...

class MainActivity : AppCompatActivity() , OnMapReadyCallback {
                                         vedInstanceState: Bundle?) {
                                         InstanceState)
                                         out.activity_main)

                                         pportFragmentManager
                                         d(R.id.myMap) as SupportMapFragment
        mapFragment.getMapAsync( callback: this)

    }
}
```

Implement members
Make 'MainActivity' 'abstract'
Add Parcelable Implementation
Create test
Adjust code style settings
Press Ctrl+Shift+I to open preview

20- In the following dialog box and as illustrated in the figure below, select: **onMapReady** method and click **OK**.



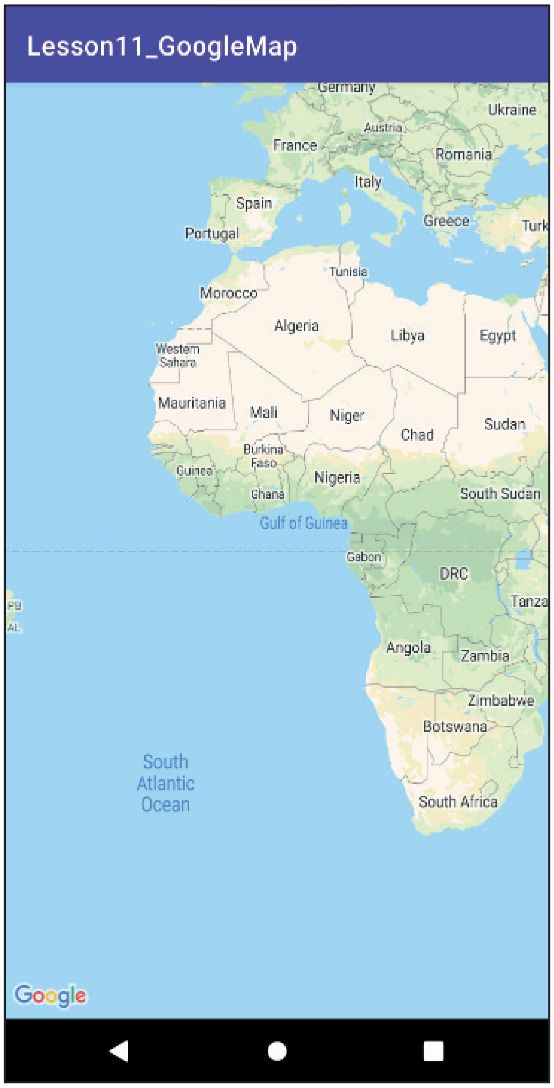
Implement Members
com.google.android.gms.maps.OnMapReadyCallback
onMapReady(p0: GoogleMap!): Unit
Copy JavaDoc
OK
Select None
Cancel

Until this step, the full code of your **MainActivity** file should have the following code:

```
class MainActivity : AppCompatActivity() , OnMapReadyCallback {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.myMap) as SupportMapFragment
        mapFragment.getMapAsync(this)

    }

    override fun onMapReady(p0: GoogleMap) {
        TODO("Not yet implemented")
    }
}
```

21- Delete the `TODO("Not yet implemented")` of your **`onMapReady`** function.

22- **Run** your app. You will get a Google map as illustrated in the figure below shows (0, 0) latitude and longitude values.



Adding a Google Map Marker

The Google map which you got shows (0, 0) latitude and longitude values by default, but you can configure your app to show a specific address like a restaurant or any other place you have its latitude and longitude values.

The method **`onMapReady`** which you imported in this example can be configured to show a specific location when you run your app. Also, you can add a marker (pin) to your app Google map to display the exactly location you want on this map. To do this, follow the next steps:

23-Open the **MainActivity** file. Declare a new variable **GMap**. This variable will inherit all the properties of the **GoogleMap** class:

```
private lateinit var mMap: GoogleMap
```

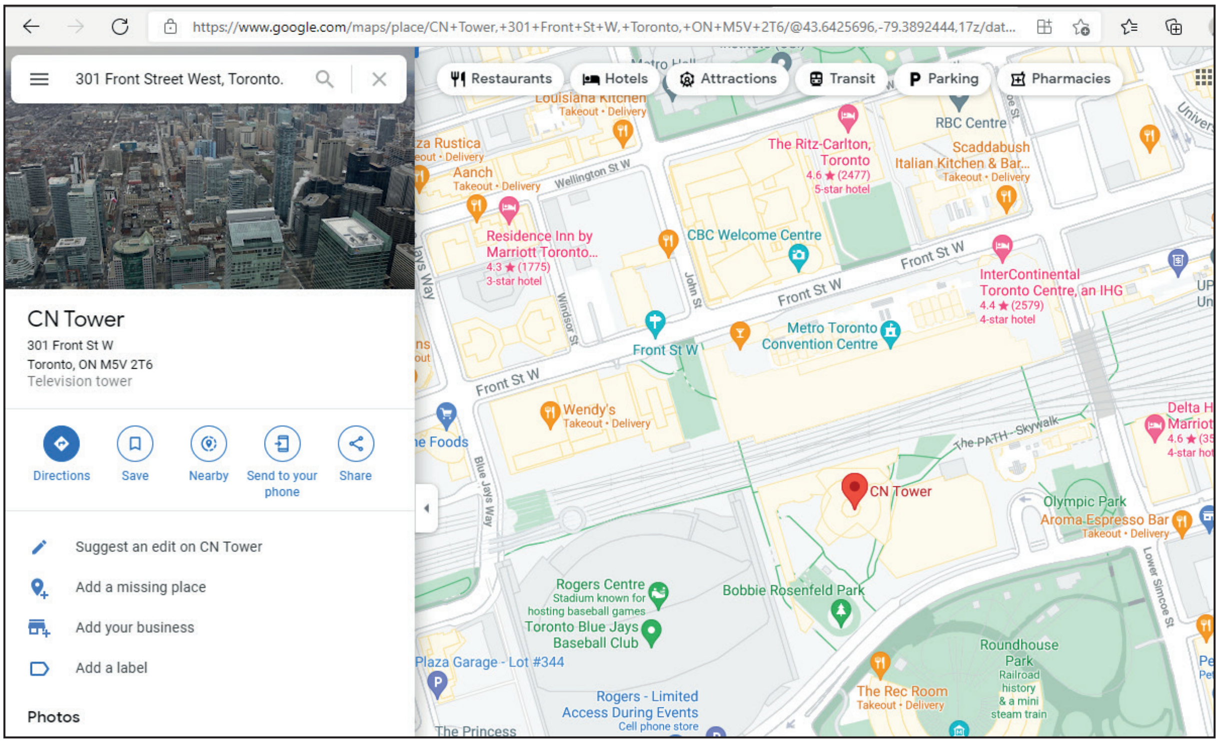
Then, replace the "**p0**" with "**googleMap**" in the `onMapReady` method to make the code easier. The `onMapReady` method will be as follows:

```
override fun onMapReady(googleMap: GoogleMap)
```

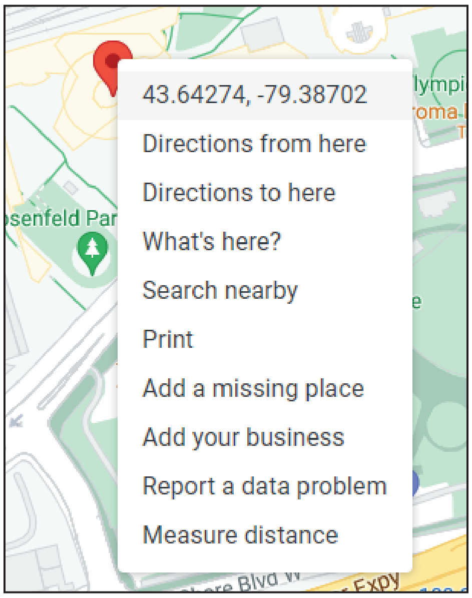
The code of the **MainActivity** file should be as follows:

```
class MainActivity : AppCompatActivity() , OnMapReadyCallback {

    private lateinit var mMap: GoogleMap

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.myMap) as SupportMapFragment
        mapFragment.getMapAsync(this)


    }

    override fun onMapReady(googleMap: GoogleMap) {


    }
}
```

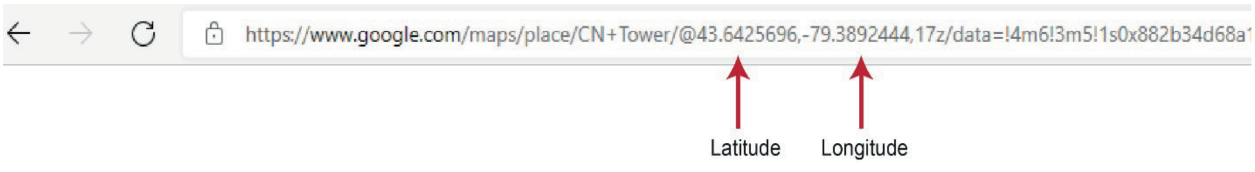
24- In this example, you are going to add a map maker for Toronto CN tower in Toronto, Canada. You must have the latitude and longitude for this location to add it to your Google map. To get the latitude and longitude coordinates for any location on the Google map, you can simply go to **www.google.com/maps** and type the address you want to know its latitude and longitude coordinates. In this example, the address of Toronto CN tower is: 301 Front Street West, Toronto. The following figure shows how Google map displays that address:



To get this address latitude and longitude coordinates, right click the maker (red pin) as illustrated below, the first choice of this shortcut menu is the latitude and longitude coordinates for this place.



Or you can go to the address bar and copy the values that come after the **@** sign. The first value is the latitude and the second value is longitude.


https://www.google.com/maps/place/CN+Tower/@43.6425696,-79.3892444,17z/data=!4m6!3m5!1s0x882b34d68a1
Latitude
Longitude

Note: There are a lot of web sites that provide free location services, and all you need to do is typing the address, then get the latitude and longitude coordinates values.

Now, you can represent your location by a variable and its latitude and longitude coordinates as follows:

```
val cnTower =LatLng(43.6425696,-79.389244)
```

`LatLng` means Latitude and Longitude.

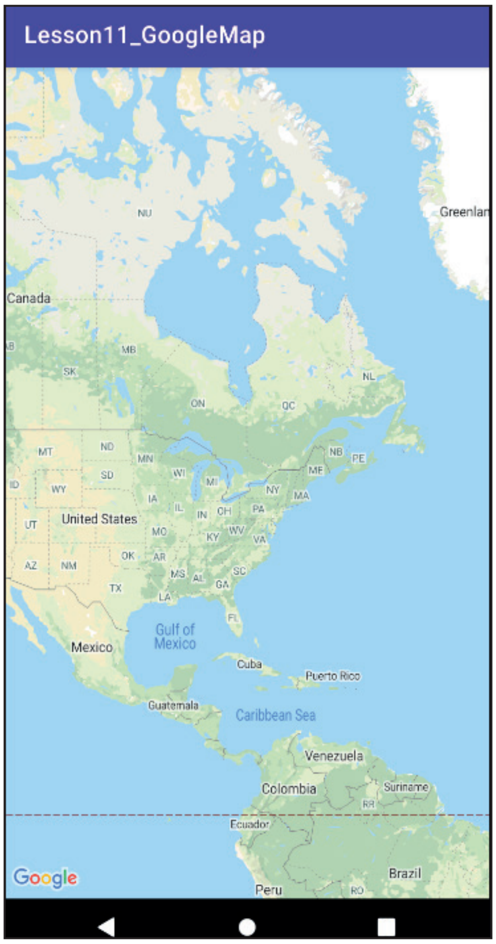
The code of the `onMapReady` method is:

```
override fun onMapReady(googleMap: GoogleMap) {

    mMap = googleMap
    val cnTower = LatLng(43.6425696, -79.389244)
    mMap.moveCamera(CameraUpdateFactory.newLatLng(cnTower))
}
```

Double click **`LatLng`**, click the red pop-up lamp, and select **Import**.

Double click **`CameraUpdateFactory`**, click the red pop-up lamp, and select **Import**.

25- Stop, then Run your app. The following figure displays the map camera on Toronto, Canada.



Remember: Maps are modelled as a flat plane on the screen, based on the Mercator projection. The map view is that of a camera looking straight down on this plane. The position of the camera can be controlled by changing the location, zoom, tilt, and bearing.

You can show your Google map in the zoom in state. All you need is to replace the following code:

```
mMap.moveCamera(CameraUpdateFactory.newLatLng(cnTower))
```

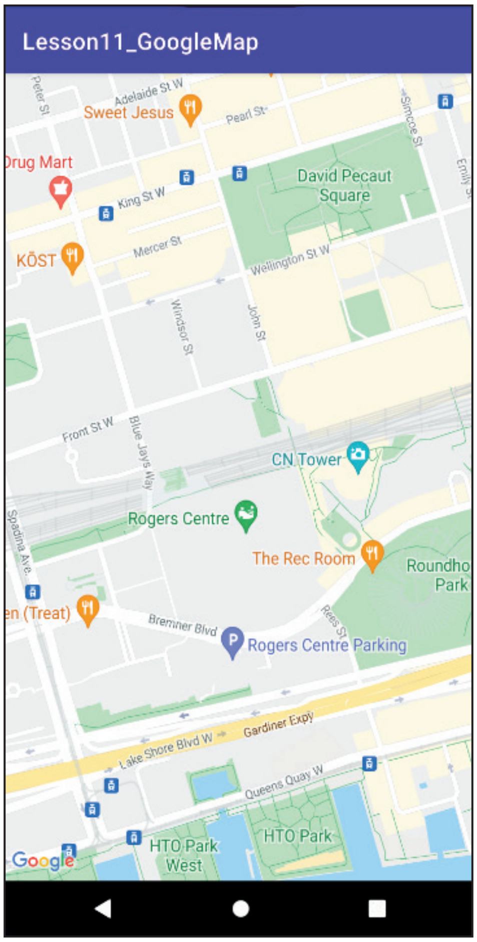
with:

```
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(cnTower,16f))
```

Note: f means: Float (data type).

You can change the zoom level more or less than 16f depending on the purpose of showing the map in your app.

26- Stop, then Run your app. The following figure displays the part of map which includes the CN Tower.



27- To add a map marker to your Google map address, add the following code to your **onMapReady** function:

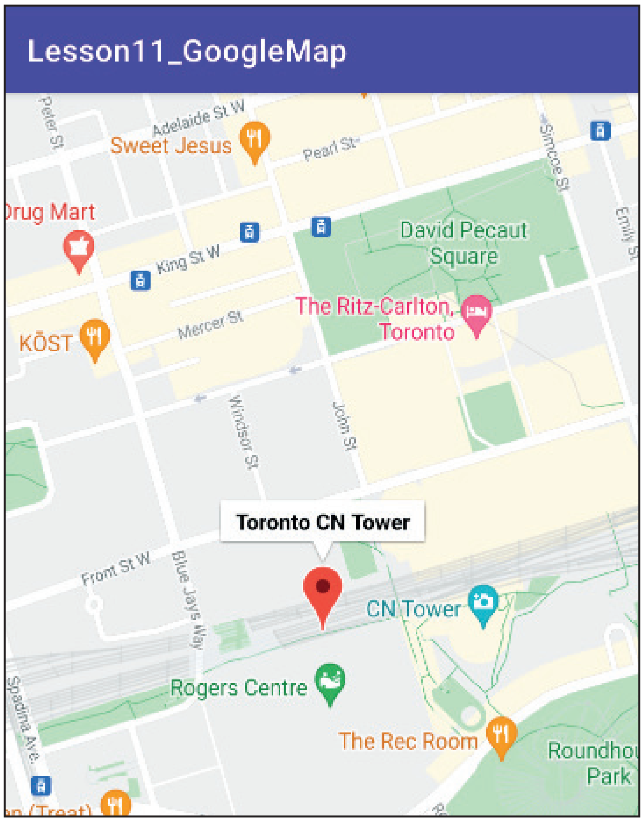
```
mMap.addMarker(MarkerOptions().position(cnTower).title("Toronto CN
Tower"))
```

Double click **MarkerOptions**, click the red pop-up lamp, and select **Import**.

The full code of the **onMapReady** function will be as follows:

```
override fun onMapReady(googleMap: GoogleMap) {

    mMap = googleMap
    val cnTower = LatLng(43.6425696, -79.389244)
    mMap.moveCamera(CameraUpdateFactory.
newLatLngZoom(cnTower,16f))

    mMap.addMarker(MarkerOptions().position(cnTower).
title("Toronto CN Tower"))
}
```

28- Stop, then Run your app. The following figure displays the map with the CN Tower map maker. If you click the map maker, you will get the map maker title: Toronto CN Tower.



Capture a User’s Location on Google Maps

Some apps use the users’ location coordinates to give them information related to their locations such as nearby restaurant, fuel stations, health or transportation services, etc…

The fused location provider manages the underlying location technologies, such as GPS and Wi-Fi, and provides a simple API that you can use to specify the required quality of service. For example, you can request the most accurate data available or the best accuracy possible with no additional power consumption.

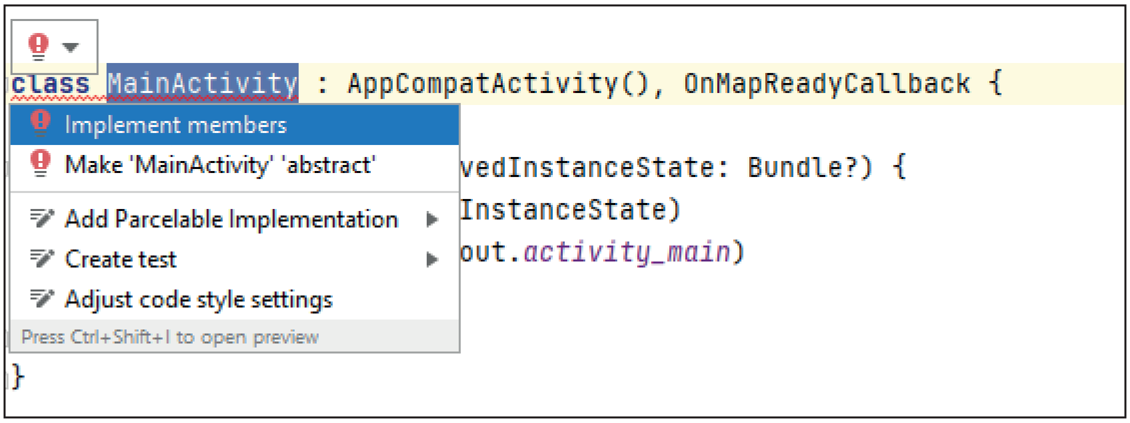
In this topic, you will use `FusedLocationProviderClient` class and `getFusedLocationProviderClient` method to connect to the app user (client) Internet provider to get his/her location on the Google maps.

You may create a new Android app, create a new or use the same Google Map API key which you have created in the previous topic, and make the same previous configuration for the **activity_main.xml** and **AndroidManifest.xml** files. However, to save the time and effort, and focus on the purpose of this topic which is understanding how configuring an Android app to display the app user location on the Google map, we will use the same previous app with little changes as follows:

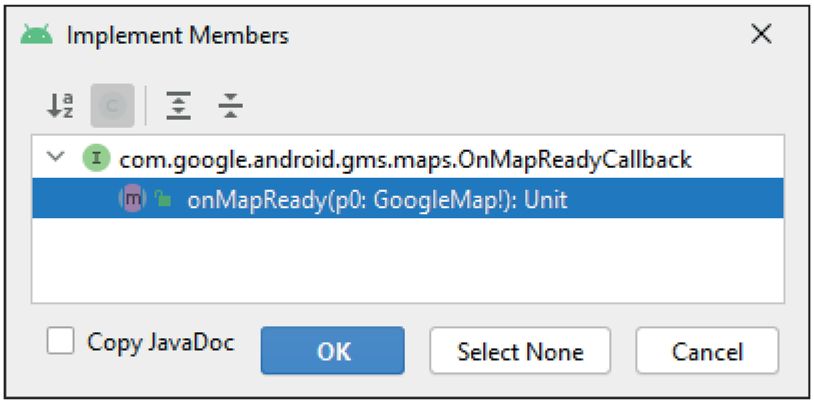
29- Open the **MainActivity** file. Delete the previous configuration and keep your file as follows:

```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

    }

}
```

30- Double click **`MainActivity`**, click the red pop-up lamp, and as illustrated in the figure below, select: **Implement members**


```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {
                    vedInstanceState: Bundle?) {
                    InstanceState)
                    out.activity_main)
}
```
Implement members
Make 'MainActivity' 'abstract'
Add Parcelable Implementation
Create test
Adjust code style settings
Press Ctrl+Shift+I to open preview

Then as illustrated in the figure below, select **onMapReady**, then click **OK**.


Implement Members
com.google.android.gms.maps.OnMapReadyCallback
onMapReady(p0: GoogleMap!): Unit
Copy JavaDoc
OK
Select None
Cancel

31- Declare the variables: `userLocation`, `fusedLocationClient`, and `REQUEST_CODE` as follows:

```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {

 var currentLocation: Location? = null
 var fusedLocationProviderClient: FusedLocationProviderClient? =
null
 private val REQUEST_CODE = 101


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

    }

    override fun onMapReady(p0: GoogleMap) {
        TODO("Not yet implemented")
    }
}
```

Double click `Location`, click the red pop-up lamp, and select **Import**.

Double click `FusedLocationProviderClient`, click the red pop-up lamp, and select **Import.**

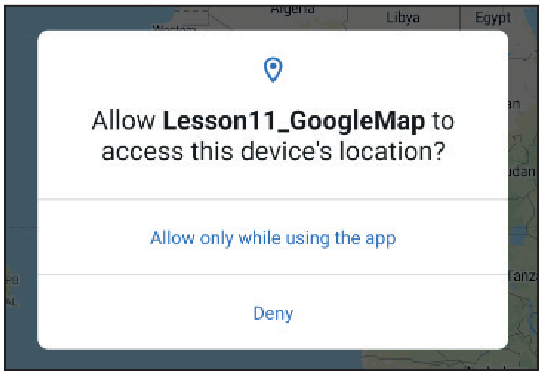
Note: The request code is any integer value. It identifies the return result when the result arrives. You can call your activity more than once before you get any results. When results arrive, you use the request code to distinguish one result from another.

32- Configure your onCreate class with `LocationServices` class and `getFusedLocationProviderClient` method as follows:

```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {

 var currentLocation: Location? = null
 var fusedLocationProviderClient: FusedLocationProviderClient? =
null
 private val REQUEST_CODE = 101

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

 // Initialize fused location
        fusedLocationProviderClient = LocationServices.
getFusedLocationProviderClient(this)

    }

    override fun onMapReady(p0: GoogleMap) {
        TODO("Not yet implemented")
    }
}
```

Double click **`LocationServices`**, click the red pop-up lamp, and select **Import**.

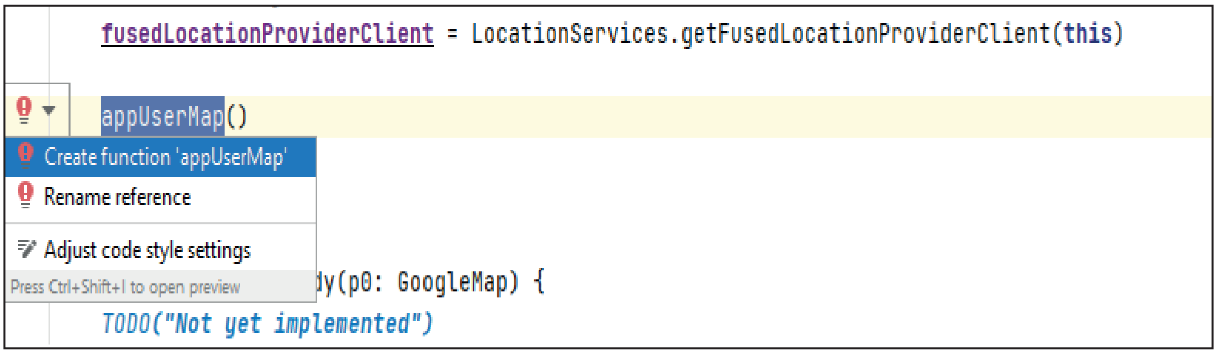
33- You should ask the app user to accept or allow the Android app to display his/her location on the Google map. This is what is called location access permission. In this step, you will add a function called **`appUserMap`** (or any method name) to ask the app user a permission to display his/her coordinates on Google map. This function or method will appear a dialog box similar to the figure below. In case the app user selects the allow option, the app user location should be displayed on the Google map.


Allow **Lesson11_GoogleMap** to access this device's location?

This function should be added as illustrated in the gray highlighted part of the following code:

```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {

 var currentLocation: Location? = null
 var fusedLocationProviderClient: FusedLocationProviderClient? =
null
 private val REQUEST_CODE = 101


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Initialize fused location
        fusedLocationProviderClient = LocationServices.
getFusedLocationProviderClient(this)

        appUserMap()

    }

    override fun onMapReady(p0: GoogleMap) {
        TODO("Not yet implemented")
    }
}
```

34- As illustrated in the figure below, double click the function **`appUserMap()`**, click the red pop-up lamp, and select: **`Create function 'appUserMap'`**



```
    fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient(this)

    appUserMap()
```

Create function 'appUserMap'
Rename reference
Adjust code style settings
Press Ctrl+Shift+I to open preview

```
...dy(p0: GoogleMap) {
    TODO("Not yet implemented")
```

The following function should be added to your code:

```
private fun appUserMap() {
    TODO("Not yet implemented")
}
```

35- Delete the TODO part of the **`appUserMap()`** function, and add to this function the following check permission code:

```
private fun appUserMap() {

    if (ActivityCompat.checkSelfPermission(this, Manifest.
permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED && ActivityCompat.
checkSelfPermission(
                this,
                Manifest.permission.ACCESS_COARSE_LOCATION
            )
            != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            this,
            arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
            REQUEST_CODE
        )
        return
    }
    // initialize task location
    val task = fusedLocationProviderClient!!.lastLocation
    task.addOnSuccessListener { location ->
        if (location != null) {
            currentLocation = location
            val supportMapFragment =
                (supportFragmentManager.findFragmentById(R.
id.myMap) as SupportMapFragment?)
            supportMapFragment!!.getMapAsync(this)
        }
    }

}
```

Double click **`ActivityCompat`**, click the red pop-up lamp, and select **Import**.

Double click **`Manifest`**, click the red pop-up lamp, and select **Import**.

Double click **`PackageManager`**, click the red pop-up lamp, and select **Import**.

Double click **`SupportMapFragment`**, click the red pop-up lamp, and select **Import**.

36- Now in the **`onMapReady`** function, delete the *`TODO`*`(`**`"Not yet implemented"`**, and add the following configuration for this function and for the `onRequestPermissionsResult function`:

```
override fun onMapReady(mMap: GoogleMap) {
    val myUserLocation = LatLng(currentLocation!!.latitude,
currentLocation!!.longitude)


    val markerOptions = MarkerOptions().position(myUserLocation).
title("Your Location")


//zoom map
mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(myUserLocation,
15f))


// add the maker on the map
    mMap.addMarker(markerOptions)
}

override fun onRequestPermissionsResult(
    requestCode: Int,
    permissions: Array<out String>,
    grantResults: IntArray) {
    super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
    when (requestCode) {
        REQUEST_CODE -> {
            if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                appUserMap()
            }
        }
    }

}
```

Double click **`LatLng`**, click the red pop-up lamp, and select **Import**.

Double click **`MarkerOptions`**, click the red pop-up lamp, and select **Import**.

Double click **`CameraUpdateFactory`**, click the red pop-up lamp, and select **Import**.

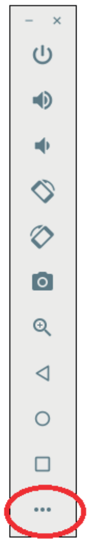
37- The full code of your **MainActivity** file will be as follows:

```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {

 var currentLocation: Location? = null
 var fusedLocationProviderClient: FusedLocationProviderClient? = null
 private val REQUEST_CODE = 101


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

 // Initialize fused location
        fusedLocationProviderClient = LocationServices.
getFusedLocationProviderClient(this)

        appUserMap()

    }

    private fun appUserMap() {

        if (ActivityCompat.checkSelfPermission(this, Manifest.
permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED && ActivityCompat.
checkSelfPermission(
                this,
                Manifest.permission.ACCESS_COARSE_LOCATION
            )
            != PackageManager.PERMISSION_GRANTED
        ) {
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
                REQUEST_CODE
            )
            return
        }

 // initialize task location
        val task = fusedLocationProviderClient!!.lastLocation
        task.addOnSuccessListener { location ->
            if (location != null) {
                currentLocation = location
                val supportMapFragment =
                    (supportFragmentManager.findFragmentById(R.
id.myMap) as SupportMapFragment?)
                supportMapFragment!!.getMapAsync(this)
                }
            }

        }

        override fun onMapReady(mMap: GoogleMap) {

            val myUserLocation = LatLng(currentLocation!!.latitude,
currentLocation!!.longitude)

            val markerOptions = MarkerOptions().position(myUserLocation).
title("Your Location")

//zoom map
mMap.animateCamera(CameraUpdateFactory.newLatLngZoom(myUserLocation,
15f))

// add the maker on the map
            mMap.addMarker(markerOptions)
        }

        override fun onRequestPermissionsResult(
            requestCode: Int,
            permissions: Array<out String>,
            grantResults: IntArray) {
            super.onRequestPermissionsResult(requestCode, permissions,
grantResults)
            when (requestCode) {
                REQUEST_CODE -> {
                    if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
                        appUserMap()
                    }
                }
            }

        }

}
```

38- Now, you should configure your **AndroidManifest.xml** file to let your app access the app user's precise location by adding a user permission tag before the **<application>** tag as illustrated in the gray highlighted part of the following code:

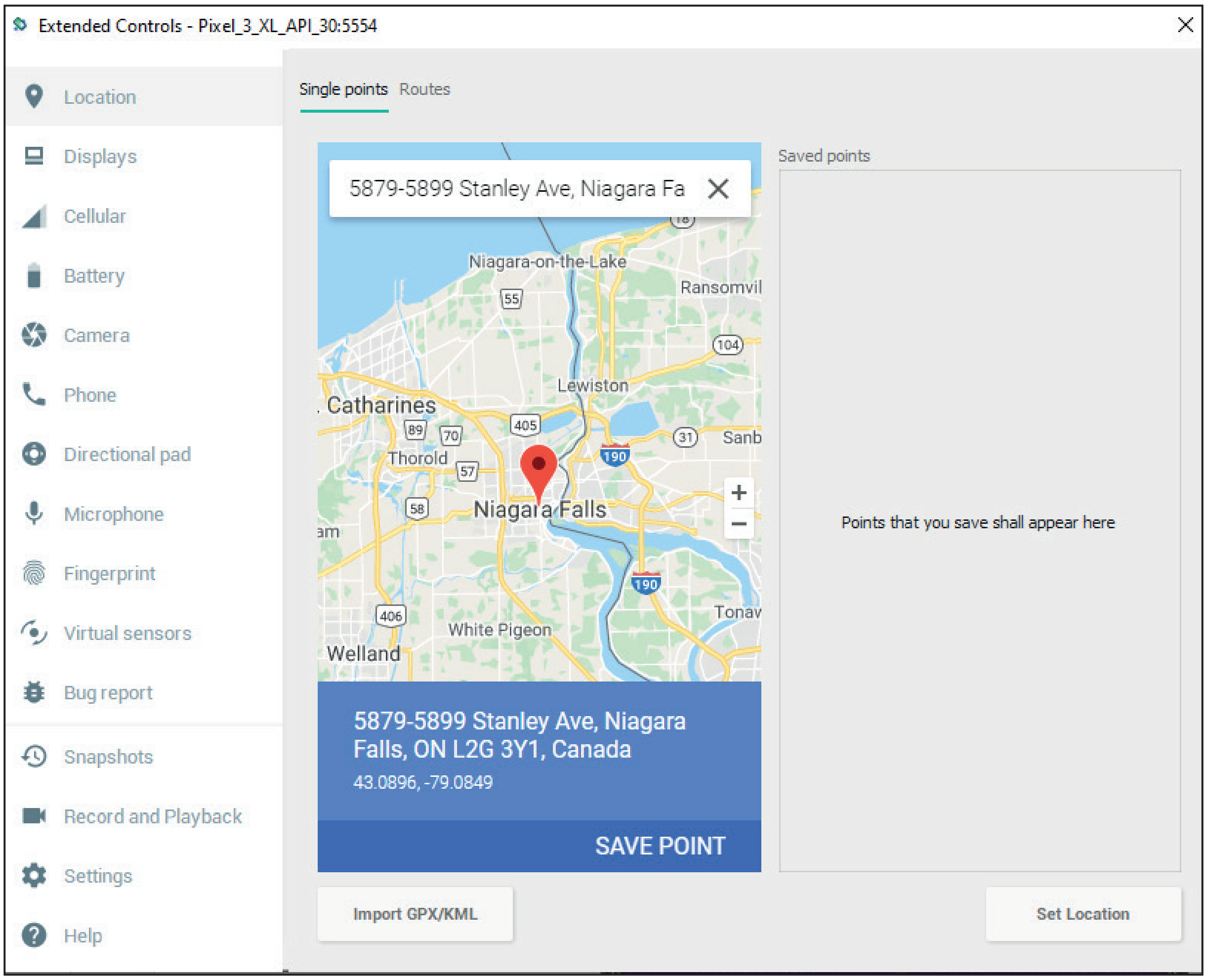
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidatc.lesson11_googlemap">
<uses-permission android:name="android.permission.ACCESS_FINE_
LOCATION"/>

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

    <application
```

39- Before running your app, remember that you are working with a virtual device (emulator); therefore, configure your location (latitude and longitude) manually by clicking the **more** button of your virtual device tools bar. (See illustration below.)



As illustrated in the figure below, type: **Niagara Falls, Canada**, then click **SAVE POINT.** Click **OK.,** click **Set Location**, then close this dialog box.

Note: You may configure the location for your emulator using your address or any address if you want to test your app configuration. However, if you test your app on your physical connected Android phone, it is enough to enable the Location setting in your phone, then the app will determine exactly your location on the Google map.

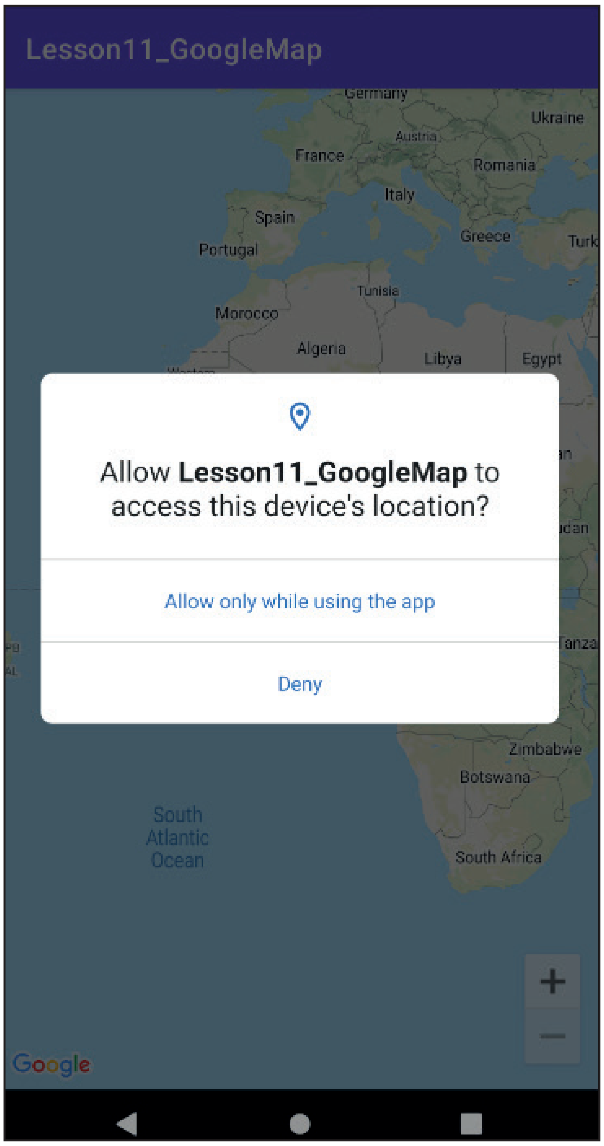


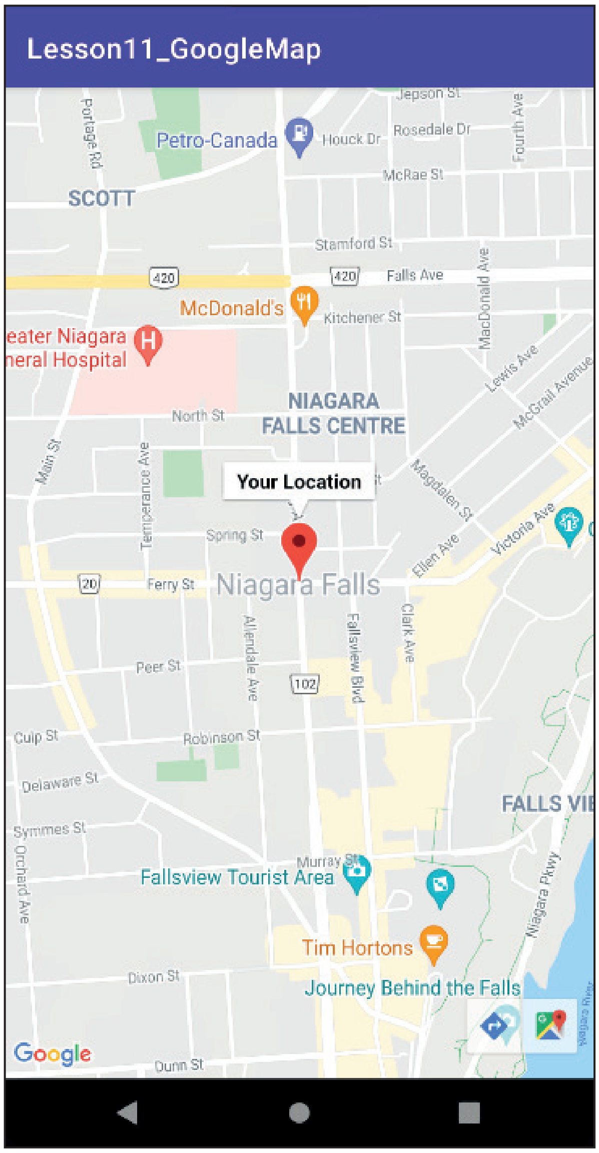
5879-5899 Stanley Ave, Niagara Fa

5879-5899 Stanley Ave, Niagara Falls, ON L2G 3Y1, Canada
43.0896, -79.0849

Points that you save shall appear here

40- **Stop**, then **Run** your app. Tap **Allow** to get the next figure which displays the user's current location. Click the map maker to get the location title.


Lesson11_GoogleMap
Allow **Lesson11_GoogleMap** to access this device's location?
Allow only while using the app
Deny



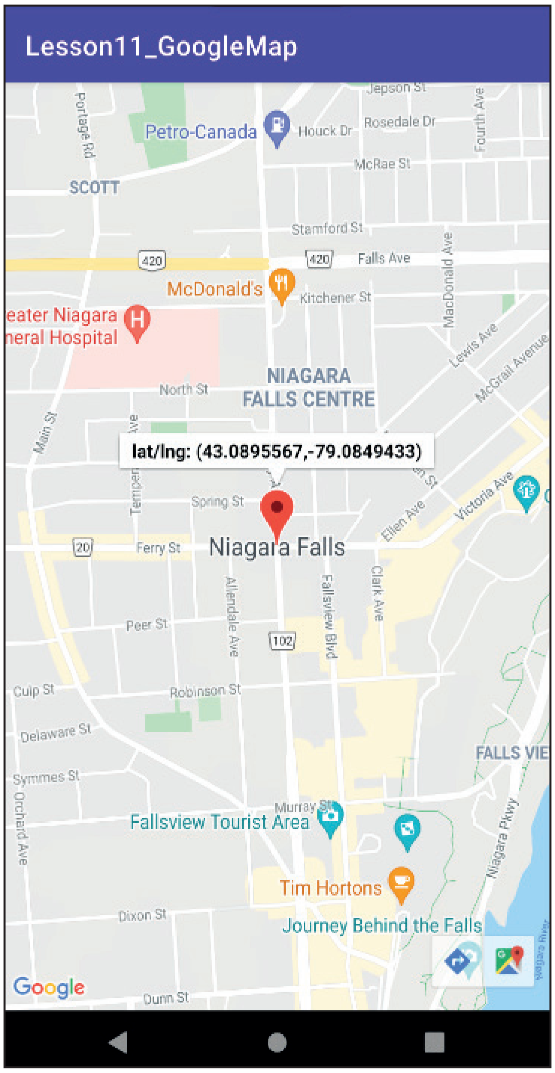
Reverse Geolocation on Google Map

Reverse geocoding is the process of retrieving a readable address from the geo-coordinates (Latitude and Longitude).

In the previous code, just replace the **title** method content in the **`onMapReady`** function with the following:

```
val markerOptions = MarkerOptions().position(myUserLocation).
title("$myUserLocation")
```

Then, stop and Run your app again. Tap the map maker. You should get the address coordinates (Latitude and Longitude) as illustrated in the figure below:



Retrieving a Readable Address on Google Map

Reverse geo-coding is the process of retrieving a readable address, place name, street address, or subdivision from the geo-coordinates (Latitude and Longitude).

Android SDK provides a class called **Geocoder** which handles the reverse geo-location.

You are going to add create a new Android project and use the same previous Google API key. This app code will enable your app to read any readable address on the Google map and show it as a marker title. The app user just taps any location on the Google map and get the address name in the map marker title. It is amazing for many applications especially for tracking and tours apps.

The following steps show how you may create this app:

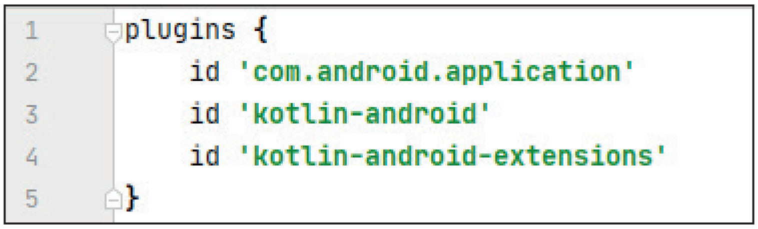
1- Open **Android Studio**, and then click **File → New → New Project.**

2- Select **Empty Activity**, and click **Next.**

3- Type: **Readable_Addresses** for the application name, then click **Finish**.

4- Before you start typing the Kotlin code in your app, check the **build.gardle (Module: Readable_Addresses.app)** file and be sure it has the Kotlin plugin. If not, add the following code:

id 'kotlin-android-extensions' as illustrated in the following figure:



```
plugins {
    id 'com.android.application'
    id 'kotlin-android'
    id 'kotlin-android-extensions'
}
```

And add the following Google maps dependencies:

```
implementation 'com.google.android.gms:play-services-
location:18.0.0'
implementation 'com.google.android.gms:play-services-maps:17.0.1'
```

Then, click the **Sync Now**.

5- Open the **AndroidManifest.xml** file and add the gray highlighted code to its configuration. This code is related to configure the allow permissions to display the user's app location on the Google map and adding the Google API key to your app. You may use the same API key that you created in the previous steps of this lesson.

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

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

        <meta-data android:name="com.google.android.geo.API_KEY"
            android:value="AIzaSyCBL11cFH0XKAWa_bca2jcJeSVqfJk4gE0"/>

        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

Note: Here, you should use your Google API Key.

6- Open the **activity_main.xml** file in **Design** mode, and then **delete** the "Hello World!" text.

7- Open the **activity_main.xml** file in the **Code** mode.

8- Replace: `androidx.constraintlayout.widget.ConstraintLayout` in the first line with: `androidx.fragment.app.FragmentContainerView`

9- Add the following attribute to your fragment tag:

```
android:name="com.google.android.gms.maps.SupportMapFragment"
```

Your **activity_main.xml** file code should be as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"


    android:name="com.google.android.gms.maps.SupportMapFragment"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"/>
```

10- Add a fragment **id** attribute with value: **myMap** to your **activity_main.xml** file. The full code of the **activity_main.xml** file will be as follows:

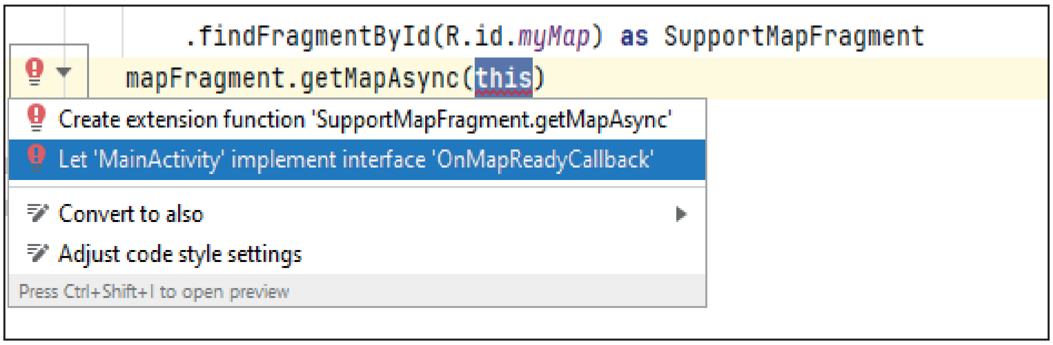
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.fragment.app.FragmentContainerView
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"

    android:id="@+id/myMap"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"/>
```

11- Open the **MainActivity** file and add the following code:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val mapFragment = supportFragmentManager

            .findFragmentById(R.id.myMap) as SupportMapFragment
        mapFragment.getMapAsync(this)

    }
}
```

Double click `SupportMapFragment`, click the red pop-up lamp, and select **Import**.

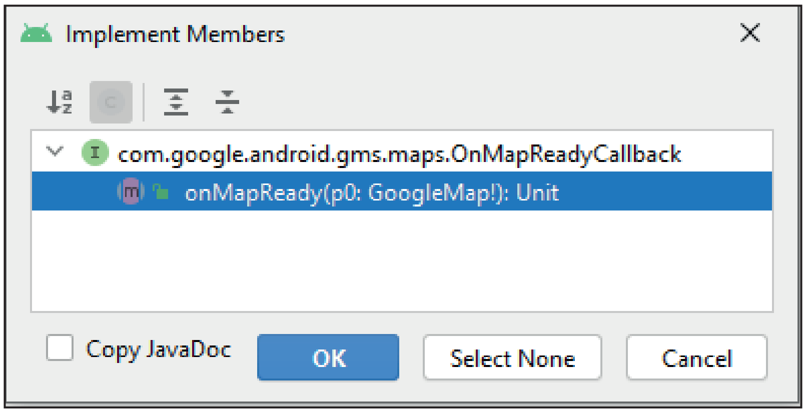
12- In your code, double click: `this`, click the red pop-up lamp, and as illustrated in the figure below, select `Let'MainActivity' implement interface'onMapReadyCallback'`.



```
        .findFragmentById(R.id.myMap) as SupportMapFragment
    mapFragment.getMapAsync(this)
```

Create extension function 'SupportMapFragment.getMapAsync'
Let 'MainActivity' implement interface 'OnMapReadyCallback'
Convert to also
Adjust code style settings
Press Ctrl+Shift+I to open preview

Then in the dialog box below, select **onMapReady** and click **OK.**


Implement Members
com.google.android.gms.maps.OnMapReadyCallback
onMapReady(p0: GoogleMap!): Unit
Copy JavaDoc
OK
Select None
Cancel

13- Declare the following variables to your code:

```
class MainActivity : AppCompatActivity(), OnMapReadyCallback {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val mapFragment = supportFragmentManager

            .findFragmentById(R.id.myMap) as SupportMapFragment
        mapFragment.getMapAsync(this)

    }

    private lateinit var GMap: GoogleMap
    var locationManager :LocationManager?=null
    var locationListener :LocationListener?=null


    override fun onMapReady(p0: GoogleMap) {
        TODO("Not yet implemented")
    }
}
```

Double click **`LocationManager`**, click the red pop-up lamp, and select **Import**.

Double click **`LocationListener`**, click the red pop-up lamp, and select **Import**.

14- Add to the imported classes (at the top of your code) the following code to import the location listener class:

```
import android.location.LocationListene
```

15- Configure your **`onMapReady`** function by replacing the *`TODO`*`(`**`"Not yet implemented"`**`)` with the following code:

```
override fun onMapReady(googleMap: GoogleMap) {

    GMap = googleMap

    locationManager = getSystemService(Context.LOCATION_SERVICE)
as LocationManager

    locationListener = object : LocationListener {

        override fun onLocationChanged(location: Location) {
            val userLocation = LatLng(location.latitude, location.
longitude)
            GMap.moveCamera(CameraUpdateFactory.
newLatLngZoom(userLocation, 16f))
            GMap.addMarker(MarkerOptions().position(userLocation).
title("Your Location"))}

    }

    if (ContextCompat.checkSelfPermission(this, Manifest.
permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_
GRANTED)
    {
        ActivityCompat.requestPermissions(this, arrayOf(Manifest.
permission.ACCESS_FINE_LOCATION), 1)}

else{
        locationManager!!.requestLocationUpdates(LocationManager.
GPS_PROVIDER,0,0f,locationListener as LocationListener
        )
    }

}
```

Double click **`Context`**, click the red pop-up lamp, and select **Import**.

Double click **`Location`**, click the red pop-up lamp, and select **Import**.

Double click **`LatLng`**, click the red pop-up lamp, and select **Import**.

Double click **`CameraUpdateFactory`**, click the red pop-up lamp, and select **Import**.

Double click **`MarkerOptions`**, click the red pop-up lamp, and select **Import**.

Double click **`ContextCompat`**, click the red pop-up lamp, and select **Import**.

Double click **`Manifest`**, click the red pop-up lamp, and select **Import**.

Double click **`PackageManager`**, click the red pop-up lamp, and select **Import**.

Double click **`ActivityCompat`**, click the red pop-up lamp, and select **Import**.

Double click **`Geocoder`**, click the red pop-up lamp, and select **Import**.

Double click **`Locale`**, click the red pop-up lamp, and select **Import**.

16- Add the following configuration for the `OnMapClickListener` method. Here in this part of the code, you will use the **Geocoder** class which handles the reverse geolocation on Google map.

```
// Click Listener

val myListener = object :GoogleMap.OnMapClickListener{

    override fun onMapClick(location: LatLng?) {
        GMap.clear()
        val geocoder=Geocoder(applicationContext, Locale.
getDefault())
        var address =""
        try{
            val addressList = geocoder.getFromLocation(location!!.
latitude,location.longitude,1)
            if (addressList!=null && addressList.size>0) {
                if (addressList[0].thoroughfare!=null){
                    address +=addressList[0].thoroughfare
                    if (addressList[0].subThoroughfare!=null){
                        address+=addressList[0].subThoroughfare
                    }
                }
            }

        } catch (e:Exception){
            e.printStackTrace()
        }
        if (address.equals("")){
            address="No address is available"
        }
        GMap.addMarker(MarkerOptions().position(location!!).
title(address))

    }
}
```

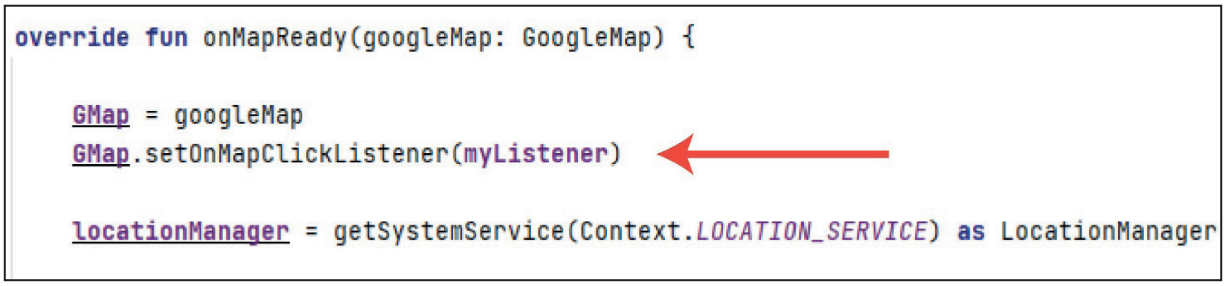
Double click **`Geocoder`**, click the red pop-up lamp, and select **Import**.

Double click **Locale**, click the red pop-up lamp, and select **Import**.

16- Add the `setOnMapClickListener` method to the `onMapReady` function using the following code:

```
GMap.setOnMapClickListener(myListener)
```

Add the code above as illustrated in the figure below:



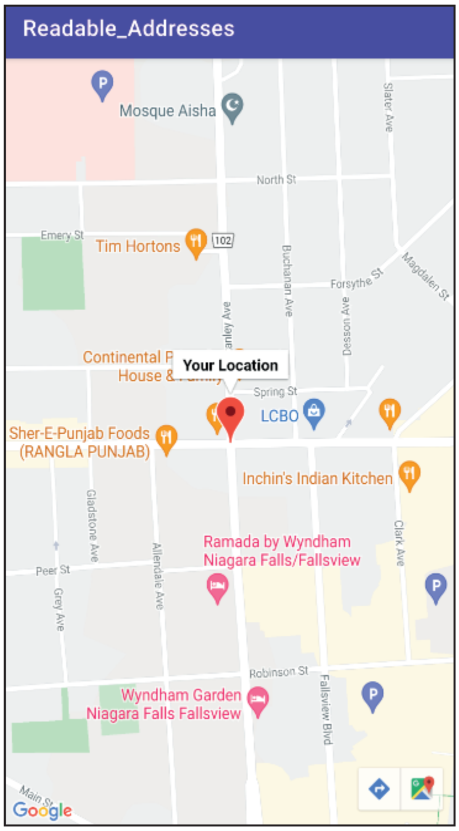
```
override fun onMapReady(googleMap: GoogleMap) {

    GMap = googleMap
    GMap.setOnMapClickListener(myListener)

    locationManager = getSystemService(Context.LOCATION_SERVICE) as LocationManager
```

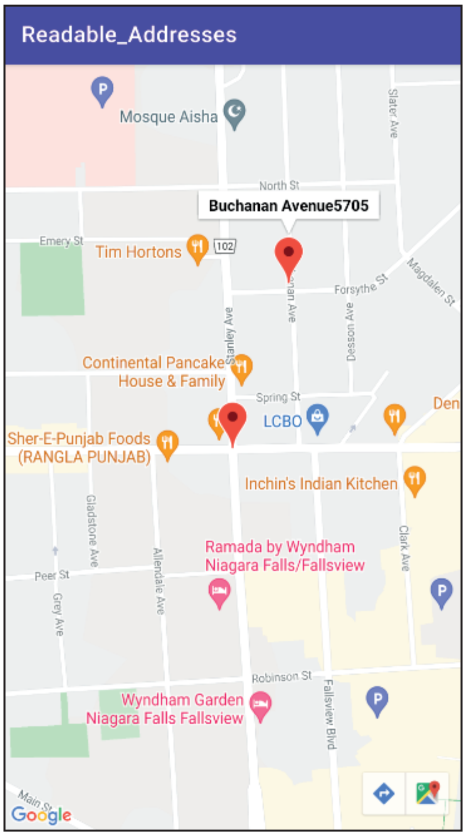
The full code of the **MainActivity** file will be as follows:

```
package com.androidatc.readable_addresses

import android.Manifest
import android.content.Context
import android.content.pm.PackageManager
import android.location.Geocoder
import android.location.Location
import android.location.LocationManager
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat

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
import java.util.*
import android.location.LocationListener

class MainActivity : AppCompatActivity(), OnMapReadyCallback {

     override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.myMap) as SupportMapFragment
        mapFragment.getMapAsync(this)

    }
    private lateinit var GMap: GoogleMap
    var locationManager : LocationManager?=null
    var locationListener : LocationListener?=null

    override fun onMapReady(googleMap: GoogleMap) {

        GMap = googleMap
        GMap.setOnMapClickListener(myListener)

        locationManager = getSystemService(Context.LOCATION_
SERVICE) as LocationManager

        locationListener = object : LocationListener {

            override fun onLocationChanged(location: Location) {
                val userLocation = LatLng(location.latitude,
location.longitude)
                GMap.moveCamera(CameraUpdateFactory.
newLatLngZoom(userLocation, 16f))
                GMap.addMarker(MarkerOptions().
position(userLocation).title("Your Location"))}

        }

        if (ContextCompat.checkSelfPermission(this, Manifest.
permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_
GRANTED)
        {
            ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.ACCESS_FINE_LOCATION), 1)}

        else{
            locationManager!!.
requestLocationUpdates(LocationManager.GPS_PROVIDER,0,0f,
locationListener as LocationListener
            )
        }

    }

    // Click Listener
    val myListener = object :GoogleMap.OnMapClickListener{
        override fun onMapClick(location: LatLng?) {
            GMap.clear()
            val geocoder=Geocoder(applicationContext, Locale.
getDefault())
            var address =""
            try{
                val addressList=geocoder.getFromLocation(location!!.
latitude,location.longitude,1)
```

```
                    if (addressList!=null && addressList.size>0) {
                        if (addressList[0].thoroughfare!=null){
                            address +=addressList[0].thoroughfare
                            if (addressList[0].subThoroughfare!=null){
                                address+=addressList[0].subThoroughfare
                            }
                        }
                    }

                } catch (e:Exception){
                    e.printStackTrace()
                }
                if (address.equals("")){
                    address="No address is available"
                }
                GMap.addMarker(MarkerOptions().position(location!!).
title(address))

        }
    }

}
```

17- Run you app. Tap allow option to display your location on Goggle map. You should get a Google map that displays your location depending on your location settings on your emulator. Click any location on the Google map, and if you click the marker of this location, you should get the location name as illustrated in the figures below:


Readable_Addresses
Your Location


Readable_Addresses
Buchanan Avenue5705