**MediaPlayer**

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using [MediaPlayer](https://developer.android.com/reference/android/media/MediaPlayer.html) APIs.

The following classes are used to play sound and video in the Android framework:

[MediaPlayer](https://developer.android.com/reference/android/media/MediaPlayer.html)

This class is the primary API for playing sound and video.

[AudioManager](https://developer.android.com/reference/android/media/AudioManager.html)

This class manages audio sources and audio output on a device.

## Manifest declarations

Before starting development on your application using MediaPlayer, make sure your manifest has the appropriate declarations to allow use of related features.

* **Internet Permission** - If you are using MediaPlayer to stream network-based content, your application must request network access.

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

* **Wake Lock Permission** - If your player application needs to keep the screen from dimming or the processor from sleeping, or uses the [MediaPlayer.setScreenOnWhilePlaying()](https://developer.android.com/reference/android/media/MediaPlayer.html#setScreenOnWhilePlaying(boolean)) or [MediaPlayer.setWakeMode()](https://developer.android.com/reference/android/media/MediaPlayer.html#setWakeMode(android.content.Context,%20int)) methods, you must request this permission.

<uses-permission android:name="android.permission.WAKE\_LOCK" />

## Using MediaPlayer

One of the most important components of the media framework is the [MediaPlayer](https://developer.android.com/reference/android/media/MediaPlayer.html) class. An object of this class can fetch, decode, and play both audio and video with minimal setup. It supports several different media sources such as:

* Local resources
* Internal URIs, such as one you might obtain from a Content Resolver
* External URLs (streaming)

For a list of media formats that Android supports, see the [Supported Media Formats](https://developer.android.com/guide/topics/media/media-formats.html) page.

Here is an example of how to play audio that's available as a local raw resource (saved in your application's res/raw/ directory):

[JAVA](https://developer.android.com/guide/topics/media/mediaplayer#java)

MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound\_file\_1);  
mediaPlayer.start();

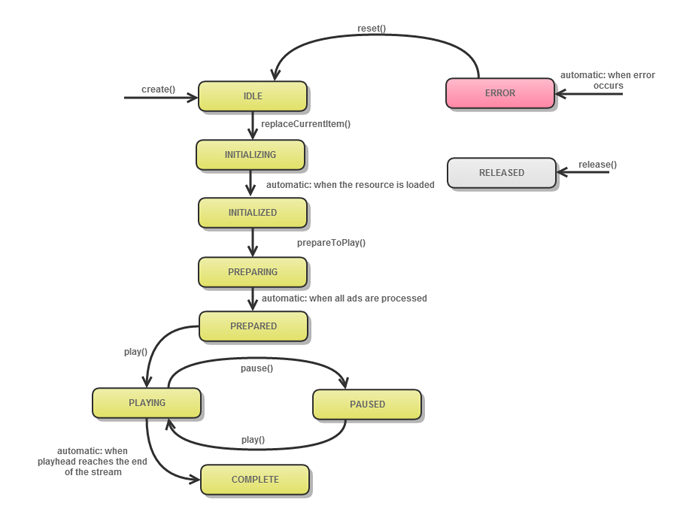
In this case, a "raw" resource is a file that the system does not try to parse in any particular way. However, the content of this resource should not be raw audio. It should be a properly encoded and formatted media file in one of the supported formats.

# MediaPlayer object lifecycle

State-transition diagram for the lifecycle of a MediaPlayer instance:

From the moment that you create the MediaPlayer instance to the moment when you terminate (reuse or remove) it, this instance completes a series of transitions between states.

Some operations are permitted only when the player is in a particular state. For example, calling play in IDLE is not allowed. You can call this status only after the player reaches the PREPARED state.



The following table provides additional details:

|  |  |
| --- | --- |
| MediaPlayer.PlayerState | Occurs when |
| IDLE | Your application requested a new media player by calling DefaultMediaPlayer.create . The newly created player is waiting for you to specify a media player item. This is the media player's initial state. |
| INITIALIZING | Your application called MediaPlayer.replaceCurrentItem , and the media player is loading. |
| INITIALIZED | TVSDK successfully set the media player item. |
| PREPARING | Your application called MediaPlayer.prepareToPlay . The media player is loading the media player item and the associated resources.  Tip: Some buffering of the main media might occur.  TVSDK is preparing the media stream and attempting to perform ad resolving and ad insertion, (if enabled).  Tip: To set the start time to a non-zero value, call prepareToPlay(startTime) with the time in milliseconds. |
| PREPARED | The content is prepared and ads have been inserted in the timeline, or the ad procedure failed. Buffering or playback can begin. |
| PLAYING | Your application has called play , so TVSDK is trying to play the video. Some buffering might occur before the video actually plays. |
| PAUSED | As your application plays and pauses the media, the media player moves between this state and PLAYING. |
| SUSPENDED | Your application navigated away from the playback, shut down the device, or switched applications while the player was playing or paused. The media player has been suspended and resources have been released. To continue, restore the media player. |
| COMPLETE | The player reached the end of the stream, and playback has stopped. |
| RELEASED | Your application has released the media player, which also releases any associated resources. You can no longer use this instance |
| ERROR | An error occurred during the process. An error also might affect what your application can do next. |

### Using wake locks

When designing applications that play media in the background, the device may go to sleep while your service is running. Because the Android system tries to conserve battery while the device is sleeping, the system tries to shut off any of the phone's features that are not necessary, including the CPU and the WiFi hardware. However, if your service is playing or streaming music, you want to prevent the system from interfering with your playback.

In order to ensure that your service continues to run under those conditions, you have to use "wake locks." A wake lock is a way to signal to the system that your application is using some feature that should stay available even if the phone is idle.

To ensure that the CPU continues running while your [MediaPlayer](https://developer.android.com/reference/android/media/MediaPlayer.html) is playing, call the [setWakeMode()](https://developer.android.com/reference/android/media/MediaPlayer.html" \l "setWakeMode(android.content.Context,%20int)) method when initializing your [MediaPlayer](https://developer.android.com/reference/android/media/MediaPlayer.html). Once you do, the [MediaPlayer](https://developer.android.com/reference/android/media/MediaPlayer.html) holds the specified lock while playing and releases the lock when paused or stopped:

[JAVA](https://developer.android.com/guide/topics/media/mediaplayer#java)

mediaPlayer = new MediaPlayer();  
// ... other initialization here ...  
mediaPlayer.setWakeMode(getApplicationContext(), PowerManager.PARTIAL\_WAKE\_LOCK);

**MediaRecorder**

In android, **MediaRecorder** class will provide a functionality to record audio or video files.

The android multimedia framework provides a built in support for capturing and encoding a variety of common audio and video formats. We have a multiple ways to record audio or video but by using **MediaRecorder** class we can easily implement audio or video recording.

In android, to record an audio we need to use device’s microphone along with **MediaRecorder** class. In case, if we want to record video, we need to use device’s camera along with **MediaRecorder** class.

## Android Set Permissions to Record Audio

To record an audio and save it in device, our app must tell the user that it will access the device’s audio input and storage, for that we need to set multiple permissions such as **RECORD\_AUDIO**, **STORAGE** and **WRITE\_EXTERNAL\_STORAGE** in our manifest file.

Following is the code snippet of defining the permissions in android manifest file to record audio and save it in device.

<manifest ... >  
<uses-permission android:name="android.permission.RECORD\_AUDIO"/>  
<uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"/>  
<uses-permission android:name="android.permission.STORAGE"/>  
...

</manifest>

The **RECORD\_AUDIO**, **WRITE\_EXTERNAL\_STORAGE** are considered as a **dangerous** permissions because it may pose a risk to the user’s privacy. Starting with Android 6.0 (API level 23) an app that uses a **dangerous** permission must ask the user for approval at run time. After the user has granted permission, the app should remember and not ask again.

## Android MediaRecorder Class

In android, we can record audio or video by using **MediaRecorder** class in our applications. This class will provide required API’s to record audio and video.

To use **MediaRecord** class to record an audio, we need to create an instance of **MediaRecorder** class and set the source, output, encoding format and output file to store the recorded audio in device. After that we need to call **prepare()**, **start()**, **stop()**, etc. to start the audio recording in our application.

Following is the code snippet to use **MediaRecorder** to record audio in android applications.

MediaRecorder recorder = new MediaRecorder();  
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);  
recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE\_GPP);  
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR\_NB);  
recorder.setOutputFile(PATH\_NAME);  
recorder.prepare();  
recorder.start();   // Recording will start  
...  
recorder.stop();  
recorder.reset();   // You can reuse the object by going back to setAudioSource() step  
recorder.release(); // Now the object cannot be reused

If you observe above code snippet, we create an instance of **MediaRecorder** class and added required audio source, output format, encoding format, audio file saving path, etc. to record an audio and save it in device.

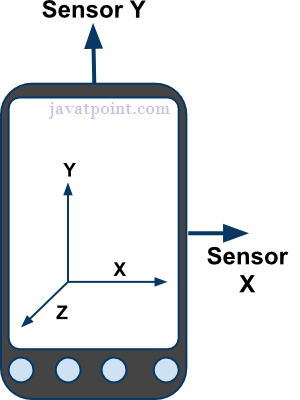
| **Method** | **Description** |
| --- | --- |
| setAudioSource() | It is used to specify the source of audio to be recorded. |
| setVideoSource() | It is used to specify the source of video to be recorded. |
| setOutputFormat() | It is used to specify the audio / video output format. |
| setAudioEncoder() | It is used to specify the audio encoder. |
| setVideoEncoder() | it is used to specify the video encoder. |
| setOutputFile() | It is used to specify the path of recorded audio / video file to be stored. |
| stop() | It is used to stop the recording process. |
| start() | it is used to start the recording process. |
| release() | It is used to releases the resources which are associated with MediaRecorder object. |

Apart from above methods, **MediaRecorder** class provides a different type of methods to control audio and video recording based on requirements.

**Sensors**

**Sensors** can be used to monitor the three-dimensional device movement or change in the environment of the device.

Android provides sensor api to work with different types of sensors.



## Types of Sensors

Android supports three types of sensors:

#### 1) Motion Sensors

These are used to measure acceleration forces and rotational forces along with three axes.

#### 2) Position Sensors

These are used to measure the physical position of device.

#### 3) Environmental Sensors

These are used to measure the environmental changes such as temperature, humidity etc.

## Android Sensor API

Android sensor api provides many classes and interface. The important classes and interfaces of sensor api are as follows:

#### 1) SensorManager class

The **android.hardware.SensorManager** class provides methods :

* to get sensor instance,
* to access and list sensors,
* to register and unregister sensor listeners etc.

You can get the instance of SensorManager by calling the method getSystemService() and passing the SENSOR\_SERVICE constant in it.

1. SensorManager sm = (SensorManager)getSystemService(SENSOR\_SERVICE);

#### 2) Sensor class

The **android.hardware.Sensor** class provides methods to get information of the sensor such as sensor name, sensor type, sensor resolution, sensor type etc.

#### 3) SensorEvent class

Its instance is created by the system. It provides information about the sensor.

#### 4) SensorEventListener interface

It provides two call back methods to get information when sensor values (x,y and z) change or sensor accuracy changes.

|  |  |
| --- | --- |
| **Public and abstract methods** | **Description** |
| **void onAccuracyChanged(Sensor sensor, int accuracy)** | it is called when sensor accuracy is changed. |
| **void onSensorChanged(SensorEvent event)** | it is called when sensor values are changed. |

**Proximity Sensor**

Detects when an object is near to the phone. Most commonly used to sense when a phone is held up to the users ear to turn off the display. This saves both battery life and prevents accidental screen touches.

**Accelerometer and gyroscope**

Accelerometers in mobile phones are used to detect the orientation of the phone. The gyroscope, or gyro for short, adds an additional dimension to the information supplied by the accelerometer by tracking rotation or twist.

An accelerometer measures linear acceleration of movement, while a gyro on the other hand measures the angular rotational velocity. Both sensors measure rate of change; they just measure the rate of change for different things.

In practice, that means that an accelerometer will measure the directional movement of a device but will not be able to resolve its lateral orientation or tilt during that movement accurately unless a gyro is there to fill in that info.

With an accelerometer you can either get a really "noisy" info output that is responsive, or you can get a "clean" output that's sluggish. But when you combine the 3-axis accelerometer with a 3-axis gyro, you get an output that is both clean and responsive in the same time."

Accelerometers are also used to provide 'steps' information for a vendors 'health' application.

**Digital compass**

The digital compass that's usually based on a sensor called the magnetometer and provides mobile phones with a simple orientation in relation to the Earth's magnetic field. As a result, your phone always knows which way is North so it can auto rotate your digital maps depending on your physical orientation.

**Barometer**

The barometer assists the GPS chip inside the device to get a faster lock by instantly delivering altitude data. Additionally,

the barometer can be utilized to provide 'floors climbed' information to a phones 'health' app.

With the advent of more accurate indoor navigation, the barometer can assist in determine what floor a user is on within an airport for example.

**Bluetooth :**

Among many ways, Bluetooth is a way to send or receive data between two different devices. Android platform includes support for the Bluetooth framework that allows a device to wirelessly exchange data with other Bluetooth devices.

Android provides Bluetooth API to perform these different operations.

* Scan for other Bluetooth devices
* Get a list of paired devices
* Connect to other devices through service discovery

Android provides BluetoothAdapter class to communicate with Bluetooth. Create an object of this calling by calling the static method getDefaultAdapter(). Its syntax is given below.

private BluetoothAdapter BA;

BA = BluetoothAdapter.getDefaultAdapter();

In order to enable the Bluetooth of your device, call the intent with the following Bluetooth constant ACTION\_REQUEST\_ENABLE. Its syntax is.

Intent turnOn = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(turnOn, 0);

Apart from this constant, there are other constants provided the API , that supports different tasks. They are listed below.

|  |  |
| --- | --- |
| **Sr.No** | **Constant & description** |
| 1 | **ACTION\_REQUEST\_DISCOVERABLE**  This constant is used for turn on discovering of bluetooth |
| 2 | **ACTION\_STATE\_CHANGED**  This constant will notify that Bluetooth state has been changed |
| 3 | **ACTION\_FOUND**  This constant is used for receiving information about each device that is discovered |

Once you enable the Bluetooth , you can get a list of paired devices by calling getBondedDevices() method. It returns a set of bluetooth devices. Its syntax is.

private Set<BluetoothDevice>pairedDevices;

pairedDevices = BA.getBondedDevices();

Apart form the parried Devices , there are other methods in the API that gives more control over Blueetooth. They are listed below.

|  |  |
| --- | --- |
| **Sr.No** | **Method & description** |
| 1 | **enable()**  This method enables the adapter if not enabled |
| 2 | **isEnabled()**  This method returns true if adapter is enabled |
| 3 | **disable()**  This method disables the adapter |
| 4 | **getName()**  This method returns the name of the Bluetooth adapter |
| 5 | **setName(String name)**  This method changes the Bluetooth name |
| 6 | **getState()**  This method returns the current state of the Bluetooth Adapter. |
| 7 | **startDiscovery()**  This method starts the discovery process of the Bluetooth for 120 seconds. |

Location Based services:

XML CODE

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:orientation="vertical"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity"**>  
  
  
 <**EditText  
 android:id="@+id/editText"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
  
 android:ems="10"  
 android:inputType="textPersonName"  
 android:text="Name"** />  
  
 <**Button  
 android:id="@+id/button"  
 android:layout\_width="244dp"  
 android:layout\_height="wrap\_content"  
  
 android:text="Button"** />  
  
 <**TextView  
 android:id="@+id/textView"  
 android:layout\_width="426dp"  
 android:layout\_height="421dp"  
 android:text="TextView"** />  
  
</**LinearLayout**>

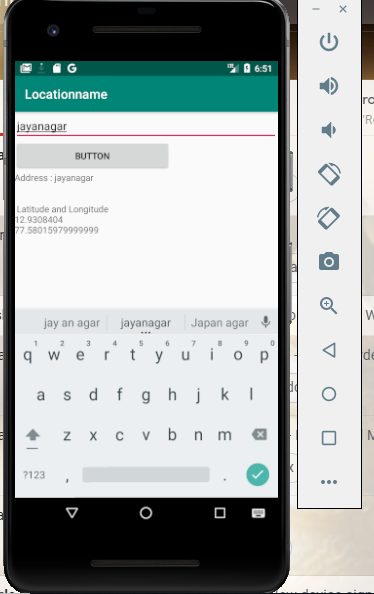
JAVA CODE:

MainActivity.java

**package** com.example.locationname;  
  
**import** androidx.annotation.NonNull;  
**import** androidx.appcompat.app.AppCompatActivity;  
  
**import** android.location.Geocoder;  
**import** android.os.Bundle;  
**import** android.os.Handler;  
**import** android.os.Message;  
**import** android.view.View;  
**import** android.widget.EditText;  
**import** android.widget.TextView;  
**import** android.widget.Button;  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
  
 Button bt;  
 TextView add;  
 EditText place;  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 place = (EditText) findViewById(R.id.editText);  
 add=(TextView)findViewById(R.id.textView);  
 bt=(Button)findViewById(R.id.button);  
 bt.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 String add= place.getText().toString();  
 GeoLocation geoLocation= **new** GeoLocation();  
 geoLocation.getAddress(add,getApplicationContext(),**new** GeoHandler());  
 }  
 });  
  
  
 }  
 **private class** GeoHandler **extends** Handler{  
 @Override  
 **public void** handleMessage(@NonNull Message msg) {  
 String address;  
 **switch** (msg.what)  
 {  
 **case** 1:  
 Bundle bundle= msg.getData();  
 address= bundle.getString(**"address"**);  
 **break**;  
 **default**:  
 address= **null**;  
 }  
 add.setText(address);  
 }  
 }  
}

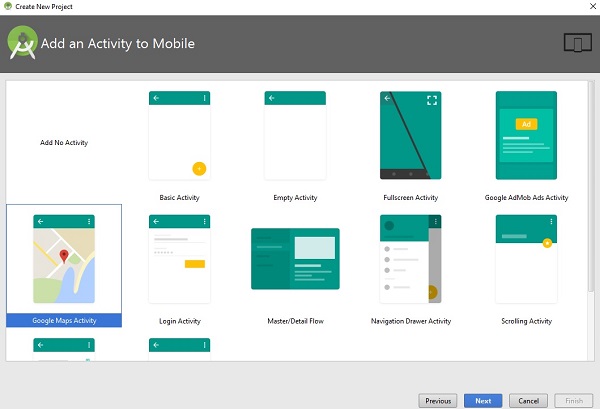
GeoLocation.java

**package** com.example.locationname;  
  
**import** android.content.Context;  
**import** android.location.Address;  
**import** android.location.Geocoder;  
**import** android.os.Bundle;  
**import** android.os.Message;  
**import** android.os.Handler;  
  
**import** java.io.IOException;  
**import** java.util.Locale;  
**import** java.util.List;  
  
  
**public class** GeoLocation {  
 **public static void** getAddress(**final** String locationAddress, **final** Context context, **final** Handler handler)  
 {  
 Thread thread=**new** Thread(){  
 @Override  
 **public void** run() {  
 Geocoder geocoder=**new** Geocoder(context, Locale.getDefault());  
 String result=**null**;  
 **try** {  
 List addressList= geocoder.getFromLocationName(locationAddress,1);  
 **if**(addressList!=**null** && addressList.size()>0){  
 Address address=(Address) addressList.get(0);  
 StringBuilder stringBuilder= **new** StringBuilder();  
 stringBuilder.append(address.getLatitude()).append(**"\n"**);  
 stringBuilder.append(address.getLongitude()).append(**"\n"**);  
 result= stringBuilder.toString();  
  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **finally** {  
 Message message= Message.obtain();  
 message.setTarget(handler);  
 **if**(result!=**null**){  
 message.what=1;  
 Bundle bundle= **new** Bundle();  
 result = **"Address : "**+locationAddress+**"\n\n\n Latitude and Longitude \n"**+result;  
 bundle.putString(**"address"**,result);  
 message.setData(bundle);  
 }  
  
 message.sendToTarget();  
  
 }  
  
  
 }  
 };  
 thread.start();  
 }  
}

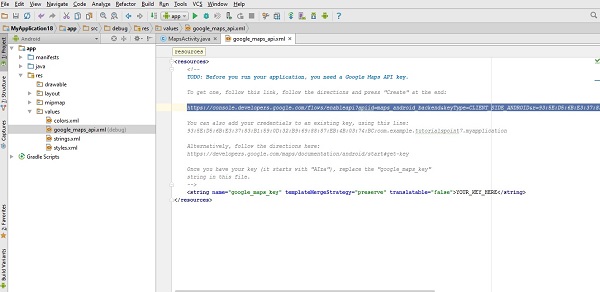


Create map activity and API Key

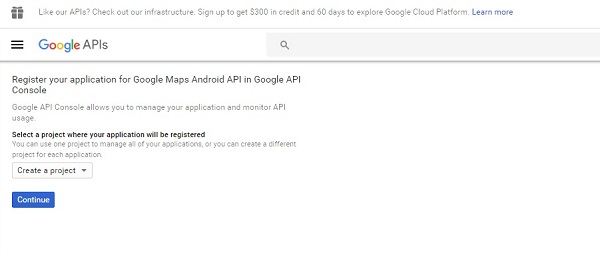
Create a project with google maps activity as shown below −



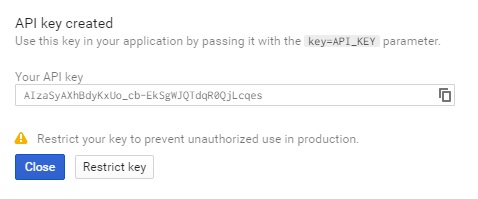
It will open the following screen and copy the console url for API Key as shown below −



Copy this and paste it to your browser. It will give the following screen −



Click on continue and click on Create API Key then it will show the following screen



Here is the content of **activity\_main.xml**.

<fragment xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:map="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/map"

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="com.example.tutorialspoint7.myapplication.MapsActivity" />

Here is the content of **MapActivity.java**.

In the below code we have given sample latitude and longitude details

package com.example.tutorialspoint7.myapplication;

import android.support.v4.app.FragmentActivity;

import android.os.Bundle;

import com.google.android.gms.maps.CameraUpdateFactory;

import com.google.android.gms.maps.GoogleMap;

import com.google.android.gms.maps.OnMapReadyCallback;

import com.google.android.gms.maps.SupportMapFragment;

import com.google.android.gms.maps.model.LatLng;

import com.google.android.gms.maps.model.MarkerOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

private GoogleMap mMap;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_maps);

// Obtain the SupportMapFragment and get notified when the map is ready to be used.

SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()

.findFragmentById(R.id.map);

mapFragment.getMapAsync(this);

}

/\*\*

\* Manipulates the map once available.

\* This callback is triggered when the map is ready to be used.

\* This is where we can add markers or lines, add listeners or move the camera.

\* In this case, we just add a marker near Sydney, Australia.

\* If Google Play services is not installed on the device.

\* This method will only be triggered once the user has installed

Google Play services and returned to the app.

\*/

@Override

public void onMapReady(GoogleMap googleMap) {

mMap = googleMap;

// Add a marker in Sydney and move the camera

LatLng TutorialsPoint = new LatLng(21, 57);

mMap.addMarker(new

MarkerOptions().position(TutorialsPoint).title("Tutorialspoint.com"));

mMap.moveCamera(CameraUpdateFactory.newLatLng(TutorialsPoint));

}

}

Following is the content of **AndroidManifest.xml** file.

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.tutorialspoint7.myapplication">

<!--

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" />

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

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/AppTheme">

<!--

The API key for Google Maps-based APIs is defined as a string resource.

(See the file "res/values/google\_maps\_api.xml").

Note that the API key is linked to the encryption key used to sign the APK.

You need a different API key for each encryption key, including the release key

that is used to sign the APK for publishing.

You can define the keys for the debug and

release targets in src/debug/ and src/release/.

-->

<meta-data

android:name="com.google.android.geo.API\_KEY"

android:value="AIzaSyAXhBdyKxUo\_cb-EkSgWJQTdqR0QjLcqes" />

<activity

android:name=".MapsActivity"

android:label="@string/title\_activity\_maps">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

Output should be like this −

