VJTech Academy

Inspiring Your Success...

UNIT-VI Security and Application Deployment

SMS Telephony

- In android, we can send SMS from our android application in two ways either by using SMSManager API or Intents based on our requirements.
- If we use SMSManager API, it will directly send SMS from our application. In case if we use Intent with proper action (ACTION_VIEW), it will invoke a built-in SMS app to send SMS from our application.

Classes of SMS Telephony

1. Telephony.Sms.Conversations - Contains a view of SMS conversations (also referred to as threads).
2. Telephony.Sms.Draft - Contains all draft text-based SMS messages in the SMS app.
3. Telephony.Sms.Inbox - Contains all text-based SMS messages in the SMS app inbox
4. Telephony.Sms.Intents - Contains constants for SMS related Intents that are broadcast
5. Telephony.Sms.Outbox - Contains all pending outgoing text-based SMS messages
6. Telephony.Sms.Sent - Contains all sent text-based SMS messages in the SMS app

Android Send SMS using SMSManager API

- In android, to send SMS using SMSManager API we need to write the code like as shown below.
  SmsManager smgr = SmsManager.getDefault();
  smgr.sendTextMessage(MobileNumber,null,Message,null,null);
- SMSManager API required SEND_SMS permission in our android manifest to sendSMS. Following is the code snippet to set SEND_SMS permissions in manifest file.
  <uses-permission android:name="android.permission.SEND_SMS"/>

Android Send SMS using Intent

- In android, Intent is a messaging object which is used to request an action from another app components such as activities, services, broadcast receivers, and content providers.
- To send SMS using the Intent object, we need to write the code like as shown below.

```
Intent sInt = new Intent(Intent.ACTION_VIEW);
sInt.putExtra("address", new String[]{txtMobile.getText().toString()});
sInt.putExtra("sms_body",txtMessage.getText().toString());
sInt.setType("vnd.android-dir/mms-sms");
```

Android Send SMS Example

STEP-1: Create .xml file

```
<?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:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context=".MainActivity">
   <TextView
      android:layout_width="250dp"
      android:gravity="center"
      android:layout_height="wrap_content"
      android:text="Messages"
      android:textSize="25dp"
      android:textColor="@color/white"
      android:layout_gravity="center_horizontal"
      android:layout_marginTop="15dp"
      android:background="#2196F3"
      />
   <TextView
      android:layout_width="250dp"
      android:gravity="center"
      android:layout_height="wrap_content"
      android:text="Phone No:"
      android:textSize="25dp"
      android:textColor="@color/white"
      android:layout_gravity="center_horizontal"
      android:layout_marginTop="100dp"
      android:background="@color/teal_200"
      />
   <EditText
      android:layout_width="250dp"
      android:layout_height="wrap_content"
      android:id="@+id/phone_no"
      android:layout_marginTop="20dp"
      android:ems="10"
```

```
    android:inputType="number"
    android:layout_gravity="center_horizontal"
    android:backgroundTint="@color/purple_500"/>
  <TextView
    android:layout_width="250dp"
    android:gravity="center"
    android:layout_height="wrap_content"
    android:text="Message:"
    android:textSize="25dp"
    android:textColor="@color/white"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="50dp"
    android:background="@color/teal_200"
    />
  <EditText
    android:layout_width="250dp"
    android:layout_height="wrap_content"
    android:id="@+id/msg"
    android:layout_marginTop="20dp"
    android:inputType="text"
    android:ems="50"
    android:layout_gravity="center_horizontal"
    android:backgroundTint="@color/purple_500"/>
  <com.google.android.material.button.MaterialButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/sent"
    android:text="SENT"
    android:textSize="20dp"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="35dp"/>
</LinearLayout>
```

STEP-2: Create .java file

```
package com.vjtech.sendsms;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;

import android.os.Bundle;

import android.telephony.SmsManager;

import android.view.View;

import android.widget.EditText;

import android.widget.Toast;

import com.google.android.material.button.MaterialButton;
```

```
public class MainActivity extends AppCompatActivity {
   MaterialButton sent,recive;
   EditText phone,msg;
   SmsManager manager;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      sent=findViewById(R.id.sent);
     // recive=findViewById(R.id.recive);
      phone=findViewById(R.id.phone_no);
      msg=findViewById(R.id.msg);

      sent.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            try{
               manager=SmsManager.getDefault();
               manager.sendTextMessage(phone.getText().toString(),null,msg.getText().toString(),null,null);
               Toast.makeText(MainActivity.this, "Sms Send Successfully ", Toast.LENGTH_SHORT).show();
               phone.setText(null);
               msg.setText(null);
            }
            catch (Exception e){
               Toast.makeText(MainActivity.this, "Sms Send Not Successfully ", Toast.LENGTH_SHORT).show();
            }
         }
      });
   }
}
```

STEP-3: AndroidManifest.xml file

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

Location Based Services

- Location Based Services are provided by Android through its location framework.
- The framework provides a location API which consists of certain classes and interface.
- These classes and interface are the key components which allow us to develop Location Based Application in Android.

Classes and Interfaces of Location Based Services:

LocationManager – This class helps to get access to the location service of the system.

LocationListener – This interface acts as the listener which receives notification from the location manager when the location changes or the location provider is disabled or enabled.

Location – This is the class which represents the geographic location returned at a particular time

Android Google Map

- Android provides facility to integrate Google map in our application. Google map displays your current location, navigate location direction, search location etc.
- We can also customize Google map according to our requirement.

Types of Google Maps

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

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

Syntax of different types of map:

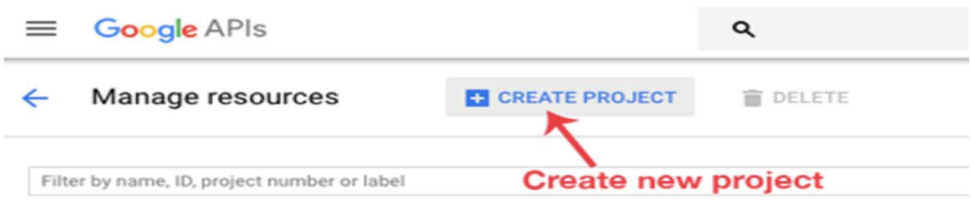
1. googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
2. googleMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
3. googleMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
4. googleMap.setMapType(GoogleMap.MAP_TYPE_TERRAIN);

Methods of Google map

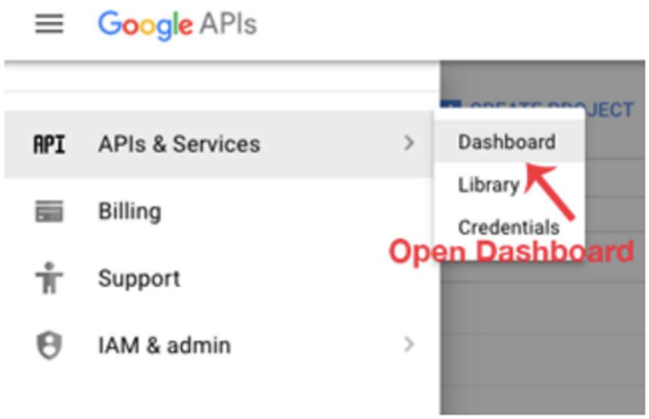
1. addCircle(CircleOptions options) – This method add circle to map.
2. addPolygon(PolygonOptions options) – This method add polygon to map.
3. addTileOverlay(TileOverlayOptions options) – This method add title overlay to the map.
4. animateCamera(CameraUpdate update) – This method move the map according to the update with an animation.
5. clear() - This method removes everything from the map.
6. getMyLocation() - This method returns the currently displayed user location.
7. moveCamera(CameraUpdate update) – Reposition the camera according to the instructions defined in the update.
8. setTrafficEnabled(boolean enabled) – This method set the traffic layer on or off.
9. snapshot(GoogleMap.SnapshotReadyCallback callback) - This method takes a snapshot of the map
10. stopAnimation() - This method stops the camera animation if there is any progress.

Getting The Google Maps API Key:

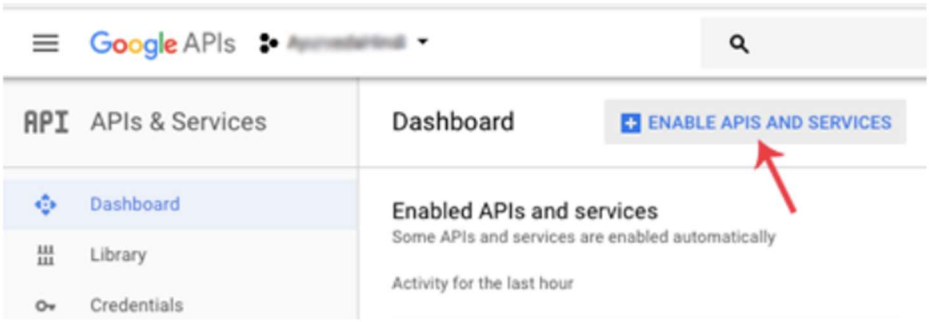
- **Step 1:** Open Google developer console and sign in with your Gmail. account: https://console.developers.google.com/project
- **Step 2:** Now create new project. You can create new project by clicking on the **Create Project** button and give name to your project



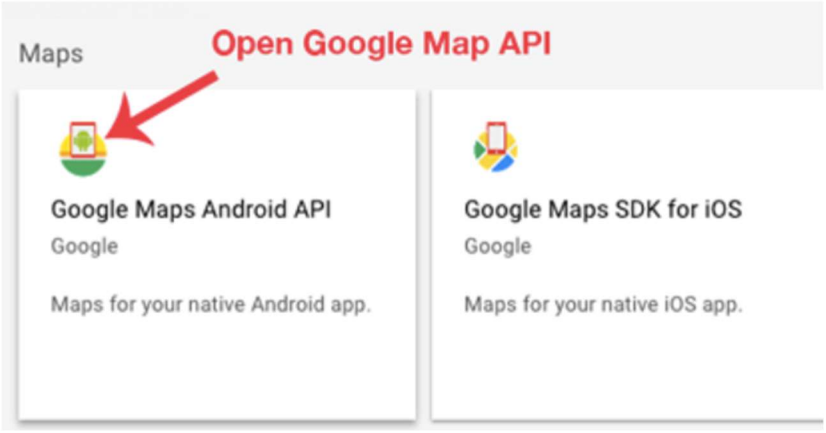
- **Step 3:** Now click on APIs & Services and open **Dashboard** from it.



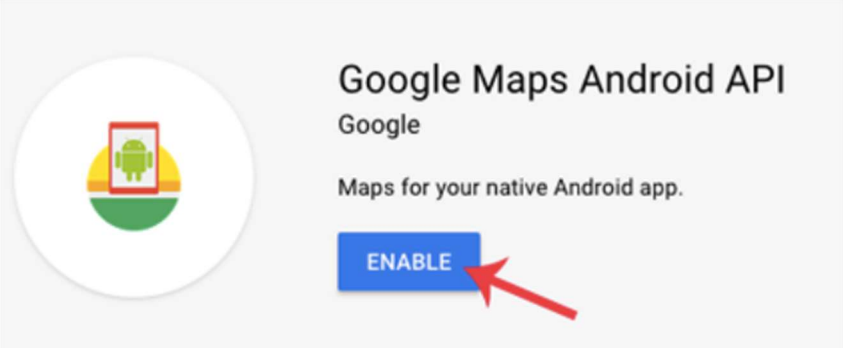
- **Step 4:** In this open **Enable APIS AND SERICES**.



- **Step 5:** Now open Google Map Android API.

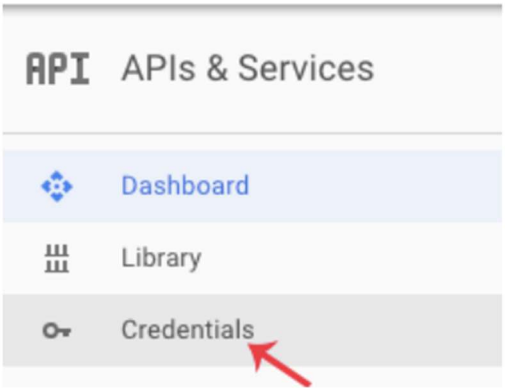


- **Step 6:** Now enable the Google Maps Android API.

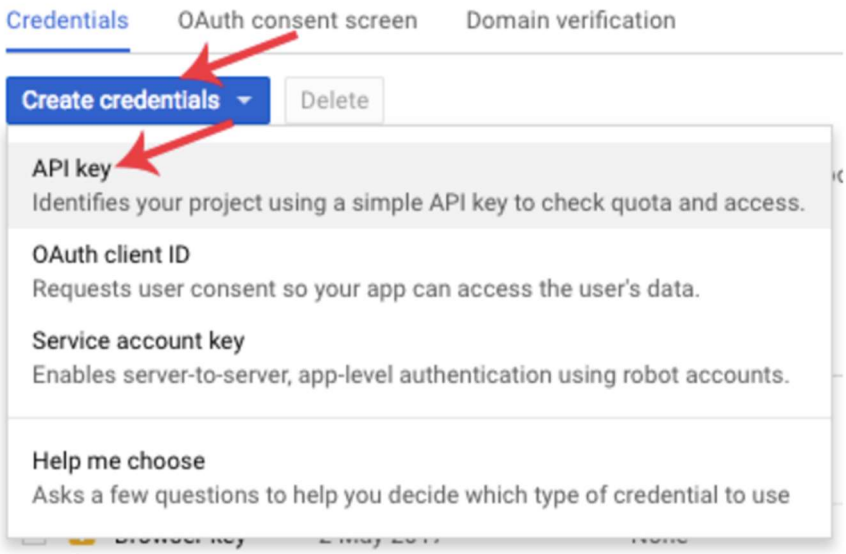


\

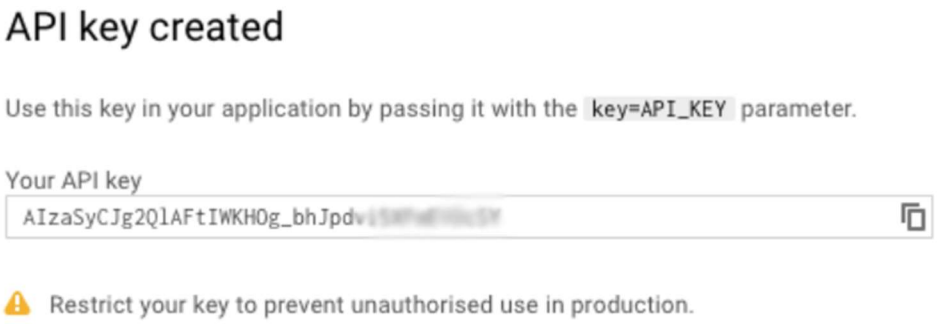
- **Step 6:** Now go to **Credentials**



- **Step 7:** Here click on Create credentials and choose API key



- **Step 8:** Now API your API key will be generated. Copy it and save it somewhere as we will need it when implementing Google Map in our Android project.



Zoom Controls:

The Maps API provides built-in zoom controls that appear in the bottom right-hand corner of the map. These can be enabled by calling:

```
mMap.getUiSettings().setZoomControlsEnabled(true);
```

ZoomIn: Double tap to increase the zoom level by 1.

Zoom Out: Two finger tap to decrease the zoom level by 1.

```
mMap.getUiSettings().setZoomGesturesEnabled(true);
```

Navigating to a specific location

- Navigating to a destination is done using a NavController, an object that manages app navigation within a NavHost.
- Each NavHost has its own corresponding NavController.
- NavController provides a few different ways to navigate to a destination, which are further described in the sections below.
- To retrieve the NavController for a fragment, activity, or view, use one of the following methods:
  - NavHostFragment.findNavController(Fragment)
  - Navigation.findNavController(Activity, @IdRes int viewId)
  - Navigation.findNavController(View)

- After you've retrieved a NavController, you can call one of the overloads of navigate() to navigate between destinations.

Adding Marker

- You can place a maker with some text over it displaying your location on the map. It can be done by via addMarker() method. Its syntax is given below –

```
final LatLng vjtech = new LatLng(21 , 57);
Marker TP = googleMap.addMarker(new MarkerOptions().position(vjtech).title("vjtech"));
```

Getting Location:

- Appropriate use of location information can be beneficial to users of your app.
- For example, if your app helps the user find their way while walking or driving, or if your app tracks the location of assets, it needs to get the location of the device at regular intervals.
- As well as the geographical location (latitude and longitude), you may want to give the user further information such as the bearing (horizontal direction of travel), altitude, or velocity of the device.
- This information, and more, is available in the Location object that your app can retrieve from the fused location provider. In response, the API updates your app periodically with the best available location, based on the currently-available location providers such as WiFi and GPS (Global Positioning System).
- The accuracy of the location is determined by the providers, the location permissions you've requested, and the options you set in the location request.

Geocoding and reverse Geocoding

- **Geocoding** is the process of converting addresses (like a street address) into geographic coordinates (like latitude and longitude), which you can use to place markers on a map, or position the map.
- **Reverse geocoding** is the process of converting geographic coordinates into a human readable address.
- Geocoding is the process of finding the geographical coordinates (latitude and longitude) of a given address or location.

- Reverse Geocoding is the opposite of geocoding where a pair of latitude and longitude is converted into an address or location.
- For achieving Geocode or Reverse Geocode you must first import the proper package.

  import android.location.Geocoder;

Getting Location data:

There are two types of location providers:

1. GPS Location Provider
2. Network Location Provider

- Any one of the above providers is enough to get current location of the user or user's device.
- But it is recommended to use both providers as they both have different advantages.
- Because, GPS provider will take time to get location at indoor area. And, the Network Location Provider will not get location when the network connectivity is poor

Network Location Provider vs GPS Location Provider

- Network Location provider is comparatively faster than the GPS provider in providing the location co-ordinates.
- GPS provider may be very slow in in-door locations and will drain the mobile battery.
- Network location provider depends on the cell tower and will return our nearest tower location.
- GPS location provider, will give our location accurately

Steps to get location in Android

1. Provide permissions in manifest file for receiving location update
2. Create LocationManager instance as reference to the location service
3. Requestlocation from LocationManager
4. Receive location update from LocationListener on change of location

Provide permissions for receiving location update

To access current location information through location providers, we need to set permissions with android manifest file.

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

ACCESS_COARSE_LOCATION is used when we use network location provider for our Android app. But, ACCESS_FINE_LOCATION is providing permission for both providers. INTERNET permission is must for the use of network provider.

Write a Program to get location data i.e. Latitude and Longitude.

Activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
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:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">

   <TextView
      android:id="@+id/textview1"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:textSize="20dp"
      android:textColor="#EF0C0C"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

MainActivity.java file

```
package com.vjtech.getgooglemaplocation;

import static android.location.LocationManager.GPS_PROVIDER;

import androidx.appcompat.app.AppCompatActivity;

import android.annotation.SuppressLint;
import android.app.LocaleManager;
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity implements LocationListener{

   LocationManager locationmanager;
   TextView tv1;
   double latitude;
   double longitude;
   @SuppressLint("MissingPermission")
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      tv1=findViewById(R.id.textview1);

      locationmanager=(LocationManager) getSystemService(Context.LOCATION_SERVICE);
      locationmanager.requestLocationUpdates(GPS_PROVIDER,0,0, this);

   }
   public void onLocationChanged(Location location)
   {
      latitude = location.getLatitude();
      longitude=location.getLongitude();
      tv1.setText("Latitude:"+latitude+"\nLongitude:"+longitude);
   }
}
```

AndroidManifest.xml file

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

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.GetGoogleMapLocation"
        tools:targetApi="31">
        <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>
```

❖ Displaying the MAP: Write a Program to accept source and destination city and display its map.

Activity_main.xml file

```
<?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:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context=".MainActivity">

   <EditText
      android:id="@+id/source"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="100dp"
      android:hint="Enter Source City"/>
   <EditText
      android:id="@+id/destination"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="100dp"
      android:hint="Enter Destination City"/>
   <Button
      android:id="@+id/b1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginTop="100dp"
      android:layout_marginLeft="130dp"
      android:text="Display Map"/>

</LinearLayout>
```

MainActivity.java file

```
package com.vjtech.getlocationdirection;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity
{
   EditText et1,et2;
   Button b1;
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      et1=findViewById(R.id.source);
      et2=findViewById(R.id.destination);
      b1=findViewById(R.id.b1);
      b1.setOnClickListener(new View.OnClickListener()
      {
         public void onClick(View view)
         {
            String src=et1.getText().toString().trim();
            String dest=et2.getText().toString().trim();
            if(src.equals("")||dest.equals(""))
            {
               Toast.makeText(MainActivity.this,"Enter both location",Toast.LENGTH_LONG).show();
            }
            else
            {
               Uri url= Uri.parse("https://www.google.com/maps/dir/"+src+"/"+dest);
               Intent obj=new Intent(Intent.ACTION_VIEW,url);
               obj.setPackage("com.google.android.apps.maps");
               startActivity(obj);
            }
         }
      });
   }
}
```

AndroidManifest.xml file

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

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

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@drawable/vjtech"
        android:label="@string/app_name"
        android:roundIcon="@drawable/vjtech"
        android:supportsRtl="true"
        android:theme="@style/Theme.GetLocationDirection"
        tools:targetApi="31">
        <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>
```

Android Security Model

- The Android security model is primarily based on a sandbox and permission mechanism.
- Each application is running in a specific Dalvik virtual machine with a unique user ID assigned to it, which means the application code runs in isolation from the code of all others applications.
- As a consequence, one application has not granted access to other applications' files.
- Android application has been signed with a certificate with a private key Know the owner of the application is unique.
- This allows the author of the application will be identified if needed. When an application is installed in the phone is assigned a user ID, thus avoiding it from affecting it other applications by creating a sandbox for it.
- This user ID is permanent on which devices and applications with the same user ID are allowed to run in a single process. This is a way to ensure that a malicious application has cannot access / compromise the data of the genuine application
- It is mandatory for an application to list all the resources it will Access during installation. Terms are required of an application, in the installation process should be user-based or interactive. Check with the signature of the application.

Declaring and Using Permissions

- The purpose of a permission is to protect the privacy of an Android user.
- Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet).
- Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.
- Permissions are divided into several protection levels. The protection level affects whether runtime permission requests are required.
- There are three protection levels that affect third-party apps: normal, signature, and dangerous permissions.
- **Normal permissions:** cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps. For example, permission to set the time zone is a normal permission. If an app declares in

its manifest that it needs a normal permission, the system automatically grants the app that permission at install time. The system doesn't prompt the user to grant normal permissions, and users cannot revoke these permissions.

- **Signature permissions:** The system grants these app permissions at install time, but only when the app that attempts to use permission is signed by the same certificate as the app that defines the permission.
- **Dangerous permissions:** cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps. For example, the ability to read the user's contacts is a dangerous permission. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app. Until the user approves the permission, your app cannot provide functionality that depends on that permission. To use a dangerous permission, your app must prompt the user to grant permission at runtime. For more details about how the user is prompted, see Request prompt for dangerous permission.

Using Custom Permission:

- Apps can define their own custom permissions and request custom permissions from other apps by defining elements.
- To enforce your own permissions, you must first declare them in your AndroidManifest.xml using one or more elements.
- For example, an app that wants to control who can start one of its activities could declare a permission for this operation as follows:

```
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.example.myapp" >

    <permission
      android:name="com.example.myapp.permission.DEADLY_ACTIVITY"
      android:label="@string/permlab_deadlyActivity"
```

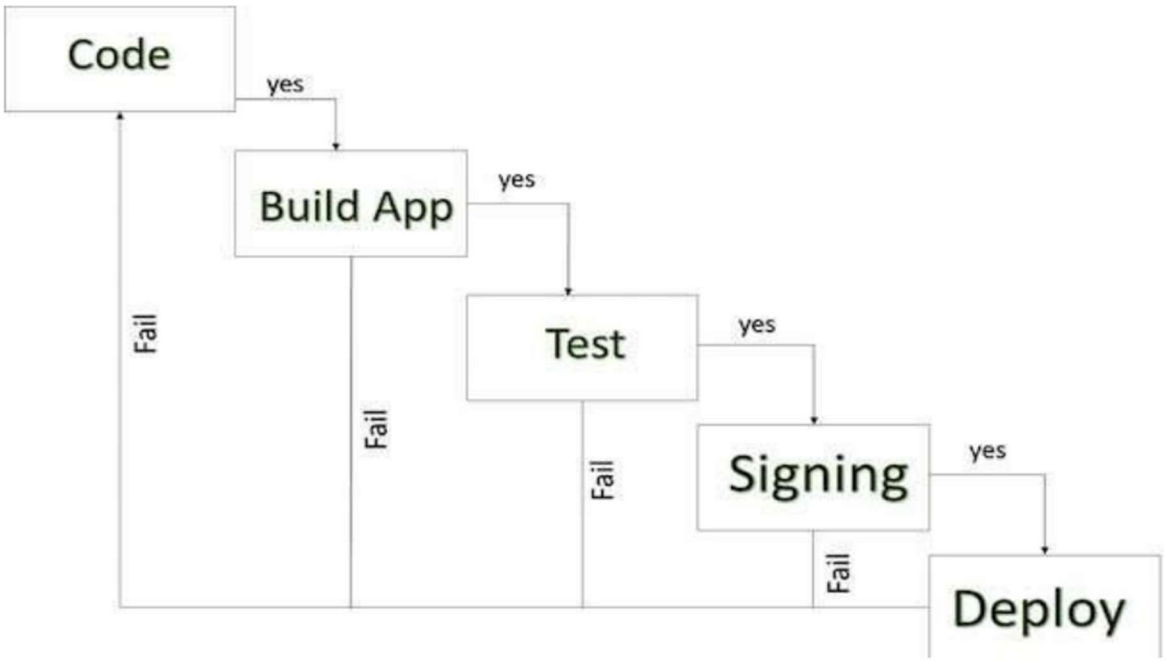
```
        android:description="@string/permdesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST_MONEY"
        android:protectionLevel="dangerous" />
    ...
</manifest>
```

- The **protectionLevel** attribute is required, telling the system how the user is to be informed of apps requiring the permission, or who is allowed to hold that permission.
- The **android:permissionGroup** attribute is optional, and only used to help the system display permissions to the user.
- You need to supply both a label and description for the permission. These are string resources that the user can see when they are viewing a list of permissions (android:label) or details on a single permission (android:description).
- The label should be short; a few words describing the key piece of functionality the permission is protecting. The description should be a couple of sentences describing what the permission allows a holder to do.
- Our convention is a two-sentence description: the first sentence describes the permission, and the second sentence warns the user of the type of things that can go wrong if an app is granted the permission.

```
<string name="permlab_callPhone">directly call phone numbers</string>
<string name="permdesc_callPhone">Allows the app to call
    phone numbers without your intervention. Malicious apps may
    cause unexpected calls on your phone bill. Note that this does not
    allow the app to call emergency numbers.</string>
```

Application Deployment

- Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.



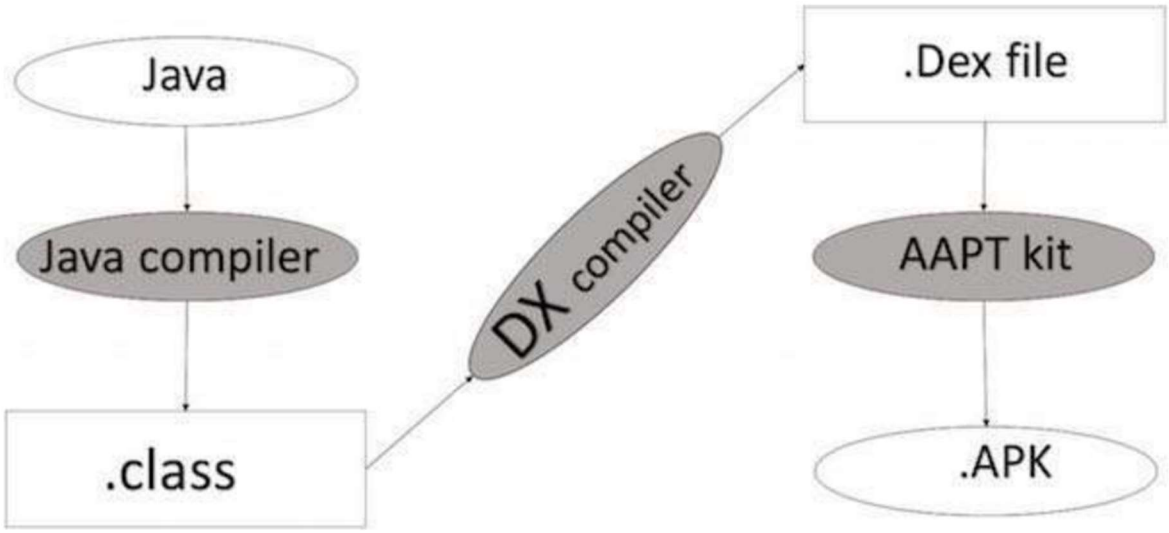
Android development life cycle

- Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace).
- You can also release your applications by sending them directly to users or by letting users download them from your own website.
- Here is a simplified check list which will help you in launching your Android application

Step	Activity
1	**Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.
2	**Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level.

	Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.
3	**Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly, you must take care of setting up time zone, localization or any other specific requirement as per the targeted region.
4	**Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices.
5	**SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target.
6	**Application Pricing** Deciding whether your app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies.
7	**Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere.
8	**Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users.
9	**Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide.

Export Android Application Process



Before exporting the apps, you must some of tools

- **Dx tools**(Dalvik executable tools): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time
- **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file** to**.Apk**
- **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

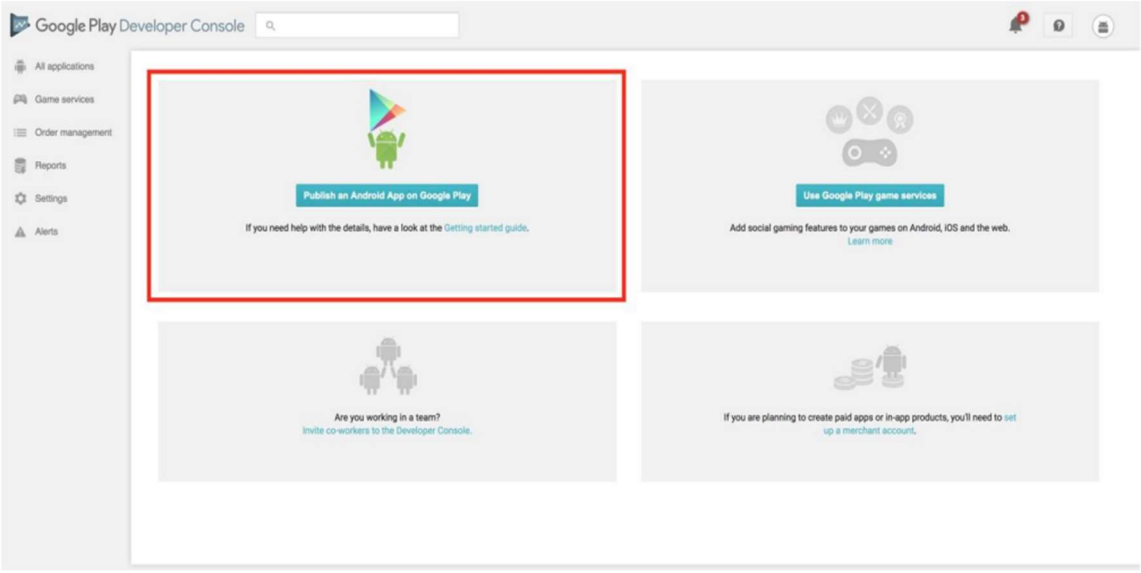
- You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.
- To export an application, just open that application project in Android studio and select **Build → Generate Signed APK** from your Android studio and follow the simple steps to export your application –
- Next select, **Generate Signed APK** option as shown in the above screen shot and then click it so that you get following screen where you will choose **Create new keystore** to store your application.
- Enter your key store path, key store password, key alias and key password to protect your application and click on **Next** button once again. It will display following screen to let you create an application –
- Once you filled up all the information, like app destination, build type and flavours click **finish** button
- Finally, it will generate your Android Application as APK format File which will be uploaded at Google Play marketplace

Steps to publish the Android application.

Step 1: Sign up Sign up for an account on the Android Developer Console. Creating an account costs $25

Step 2: Create a new application

- On the Developer Console select the Publish an Android Application option.
- Fill out the details: Title, Short Description, Full Description.



Step 3: Prepare multimedia

- Screenshots: I used the android emulator to take screenshots of my app.
- Hi-res icon: I used the launcher icon. It was an SVG file, so I converted it to PNG using GIMP.
- Feature graphic: This is an image that shows up on the top of the app download page in Google Play on mobile phones.

Step 4: Prepare code for release

- Remove log statements.
- Remove the android:debuggable attribute from your manifest file. I didn't have to do this because Android Studio automatically sets this attribute based on the kind of APK its building. Neat!

- Set the android:versionCode attribute in the manifest tag in manifest.xml. Two important notes: (1) This must be an integer that increases with each release. (2) This number is not displayed to users. I chose “1”.

- Set the android:versionName attribute in the manifest tag in manifest.xml. This string is shown to users and has no other purpose. I chose “1.0”.

Step 5: Build a release-ready APK:

The release-ready APK is different from the debug APK in that it is signed with certificate that is owned by the developer. This is done to ensure that updates to the app come from a verified source, i.e. a developer with access to the private key.

I recommend you follow the instructions here to create a signed APK.

- Android Studio -> Build -> Generate Signed APK

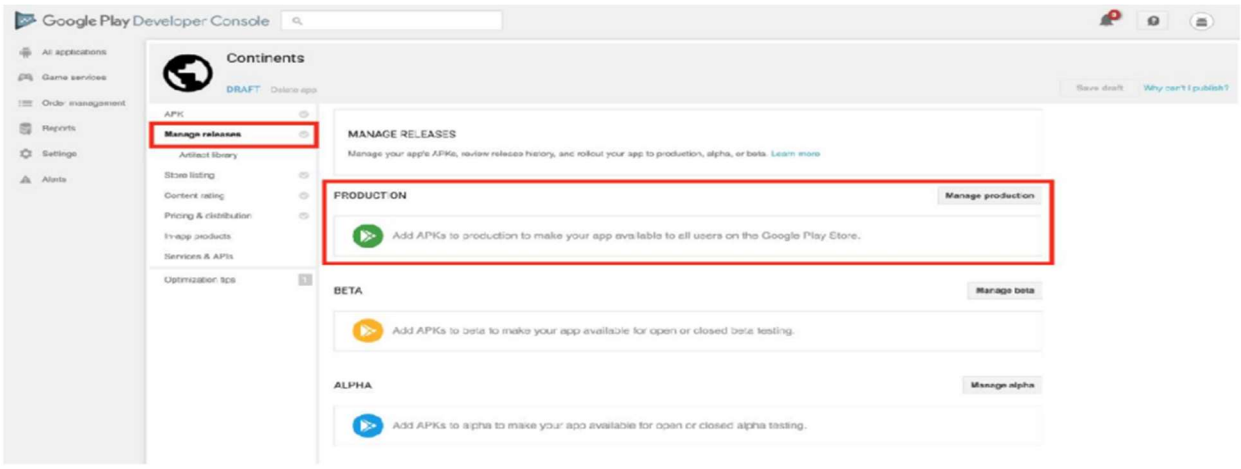
- A Java Keystore (JKS) is a repository of public-private key pairs.

- You must sign all APKs with the same key pair.

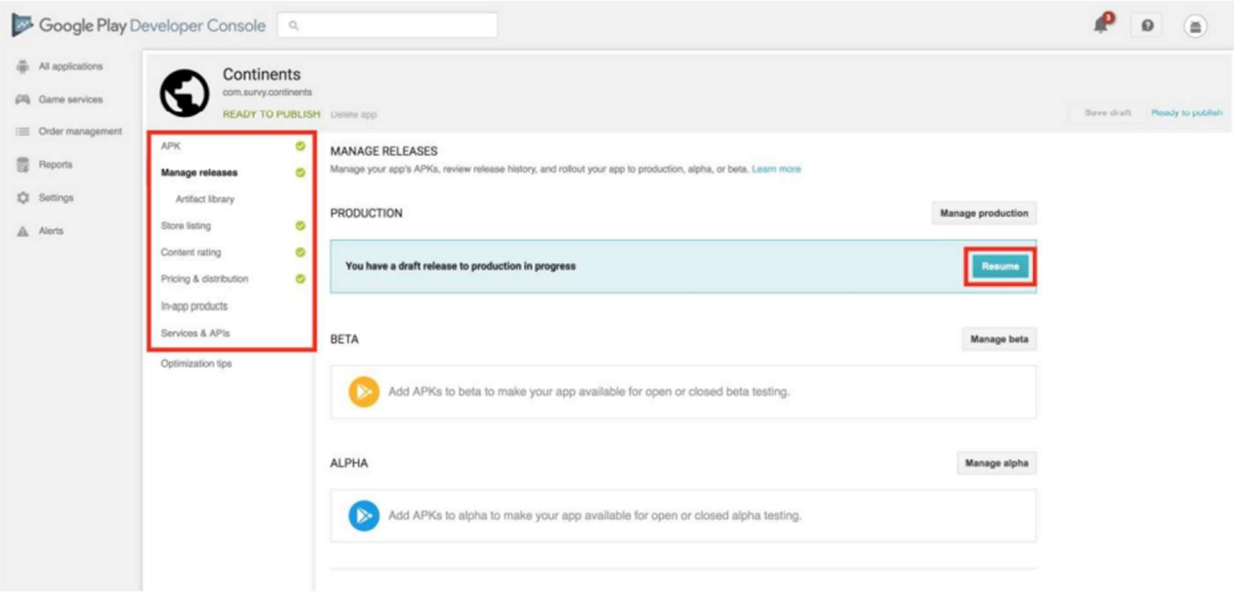
- Losing a key-pair consequences that you will not be able to push updates to your app.

Step 6: Upload APK

Go back to the Developer Console and click on Manage Releases. Then create a Production Release and upload your signed APK. Google will perform a check on the APK.



Step 7: Complete the checklist on the left until all the items have a green checkmark. The console re-evaluates the checklist every time you click Save Draft in the top right.



You are now ready to publish

Developer Console

The Android Things Console provides easy and secure deployment of updates to your connected devices. Google provides the infrastructure to host and deliver system and app updates with the developer in final control.

Using the console, developers and device makers can:

- Download and install the latest Android Things system image
- Build factory images that contain OEM applications along with the system image
- Push over-the-air (OTA) seamless updates, including OEM applications and the system image, to devices
- Manage and share OEM applications across products and owners
- Monitor informative analytics to understand how well products are performing