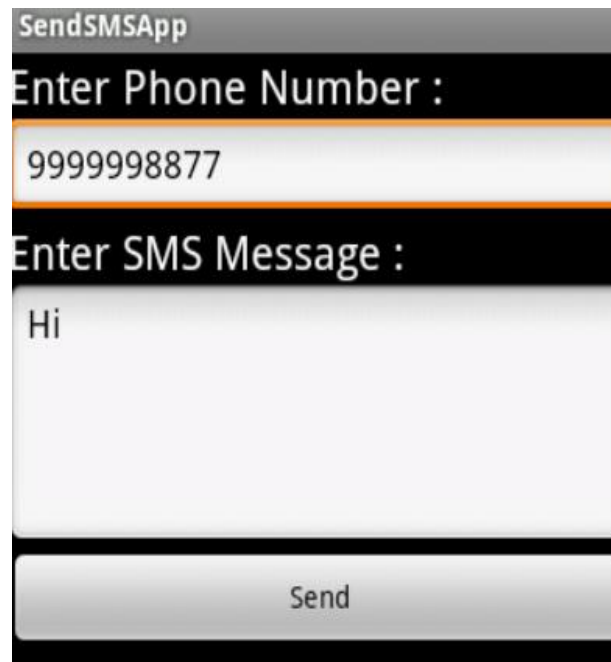


UNIT- VI Security n Application Development

SMS Telephony

- In Android, use SmsManager API or devices Built-in SMS application to send SMS's.

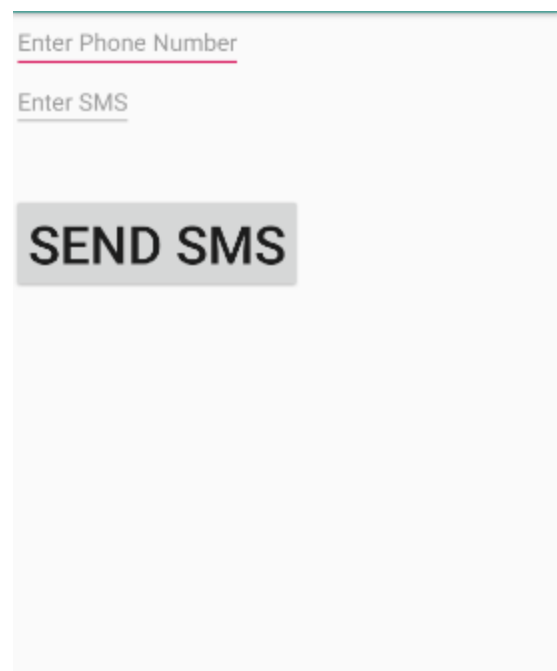


The screenshot shows an Android application interface with a black title bar at the top containing the text "SendSMSApp". Below the title bar, there are two input sections. The first section is labeled "Enter Phone Number :" in white text on a black background, followed by a white text box containing the number "9999998877". The second section is labeled "Enter SMS Message :" in white text on a black background, followed by a large white text box containing the text "Hi". At the bottom of the interface is a wide, light gray button with the word "Send" in black text.

- **Sending SMS using SmsManager API :**
- `SmsManager smsManager1 = SmsManager.getDefault();
smsManager1.sendTextMessage("phoneNo", null, "sms message", null, null);`
- **sendTextMessage Syntax :**
- `public void sendTextMessage (String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)`
- **destinationAddress:** You need to specify mobile number of the recipient.
- **ServiceCenterAddress:** You need to specify service center address. You can use null for default Service Center Address.
- **text:** You need to specify the content of your message.
- **sendIntent:** specify Pending Intent to invoke when the message is sent.
- **deliveryIntent:** specify the Pending Intent to invoke when the message has been delivered.
- A **Pending Intent** is a token you give to some app to perform an action on your apps' behalf irrespective of whether your application process is alive or not.
- By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified.
- In this case the ordering app uses a PendingIntent rather than sending an activity result because it could take a significant amount of time for the order to be delivered, and it doesn't make sense to force the user to wait while this is happening.
- **File :**
- `<uses-permission android:name="android.permission.SEND_SMS"/>`

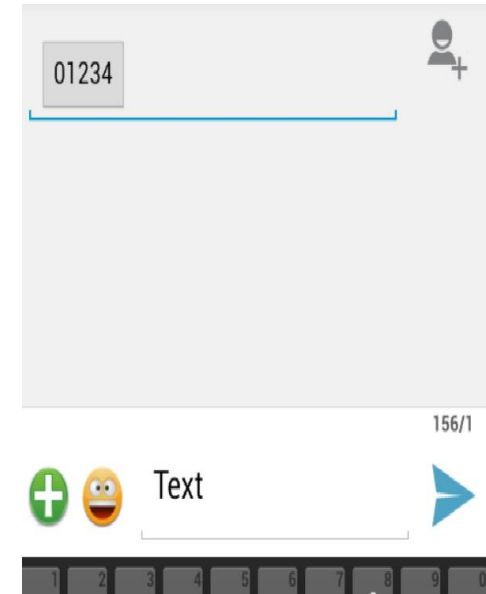
- **Built-in SMS application**
- `Intent sendIntent = new Intent(Intent.ACTION_VIEW);` *//Intent Object - Action to send SMS*
- *//To send an SMS you need to specify **smsto:** as URI using setData() method*
- `sendIntent.setData(Uri.parse("smsto:"));`
- `sendIntent.putExtra("sms_body", "default content");`
- *//This is used to create intents that only specify a type and not data, for example to indicate the type of data to return*
- *// data type will be to **vnd.android-dir/mms-sms** using setType()*
- `sendIntent.setType("vnd.android-dir/mms-sms");`
- `startActivity(sendIntent);`
- *Eg :*
- `sendIntent.putExtra("sms_body", "Test SMS to Angilla");`

```
• Button bt1;
  EditText phone_no;
  EditText msg;
  String phoneno , message;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_sms__api_exmaple1);
    bt1= (Button)findViewById(R.id.btnSendSMS);
    phone_no=(EditText)findViewById(R.id.editphono);
    msg=(EditText)findViewById(R.id.editmsg);
    bt1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        sendSMSMessage();
      }
    });
    protected void sendSMSMessage()
    {
      phoneno=phone_no.getText().toString();
      message= msg.getText().toString();
      SmsManager smsManager 1= SmsManager.getDefault(); // get the default instance of
SmsManager class
      smsManager1.sendTextMessage(phoneno, null, message, null, null);
      Toast.makeText(getApplicationContext(), "SMS sent.",
        Toast.LENGTH_LONG).show();
    }
  }
}
```



SMS by Intent –Exp -29

- `protected void sendSMS() {`
- `Log.i("Send SMS", "");`
- `Intent smsIntent = new Intent(Intent.ACTION_VIEW);`
- `smsIntent.setData(Uri.parse("smsto:"));`
- `smsIntent.setType("vnd.android-dir/mms-sms");`
- `smsIntent.putExtra("address", new String("9856342366"));`
- `//smsIntent.setData(Uri.parse("smsto:" + phoneNumber))`
- `smsIntent.putExtra("sms_body", "Test ");`
- `try { startActivity(smsIntent);`
- `finish();`
- `}`
- `catch (android.content.ActivityNotFoundException ex) { Toast.makeText(MainActivity.this,`
`"SMS failed, please try again later.", Toast.LENGTH_SHORT).show(); } }`
- `//setType : Set an explicit MIME data type.`
- `This is used to create intents that only specify a type and not data, for example to indicate the type of data to return`
- `//How different parts of a message, such as text and image, are combined into the message.`



Sending Email

- Email is messages distributed by electronic means from one system user to one or more recipients via a network.
- write an Activity that launches existing an email client(Gmail), using an implicit Intent with the right action and data.
- **Intent Object - Action to send Email**
- `Intent emailIntent = new Intent(Intent.ACTION_SEND);`
- **Intent Object - Data/Type to send Email**
- `emailIntent.setData(Uri.parse("mailto:"));`
`emailIntent.setType("text/plain");`
- **Intent Object - Extra to send Email**
- Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client.

Sr.No.	Extra Data & Description
1	EXTRA_BCC A String[] holding e-mail addresses that should be blind carbon copied.
2	EXTRA_CC A String[] holding e-mail addresses that should be carbon copied.
3	EXTRA_EMAIL A String[] holding e-mail addresses that should be delivered to.
4	EXTRA_HTML_TEXT A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text.
5	EXTRA_SUBJECT A constant string holding the desired subject line of a message.

Exp - 30

- ```
<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:id="@+id/l1"
 android:orientation="vertical"
 tools:context="MainActivity">

 <Button
 android:id="@+id/sendEmail"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Compose Email"
 android:textSize="25dp"
 android:layout_marginTop="30dp"
 android:layout_marginLeft="70dp"
 />

</LinearLayout>
```

My Application

COMPOSE EMAIL

- `public void onClick(View view) {`
- `sendEmail();`
- `}`
- `protected void sendEmail() { Log.i("Send email", "");`
- `String[] TO = {"abc@gmail.com"}; String[] CC = {"xyz@gmail.com"};`
- `Intent emailIntent = new Intent(Intent.ACTION_SEND);`
- `emailIntent.setData(Uri.parse("mailto:"));`
- `emailIntent.setType("text/plain");`
- `emailIntent.putExtra(Intent.EXTRA_EMAIL, TO`
- `emailIntent.putExtra(Intent.EXTRA_CC, CC);`
- `emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Your subject");`
- `emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message goes here");`
- `try { startActivity(Intent.createChooser(emailIntent, "Send mail...")); // title that will be displayed in the chooser`
- `finish(); Log.i("Finished sending email...", ""); } }`
- `catch (android.content.ActivityNotFoundException ex)`
- `{ Toast.makeText(MainActivity.this, "There is no email client installed.",`
- `Toast.LENGTH_SHORT).show(); } }`
- **Note : The chooser enables the user to pick another mail application than the default. Its very useful if you use normal gmail (privat) and email (work related) and you want to choose which one to take.**
- **user has both Chrome and Firefox installed on their Android device. user is presented with a chooser, so the user can choose which of the two Web browsers to use.**

# Google Map

- Android allows us to integrate Google Maps in our application. For this Google provides us a library via Google Play Services for using maps.
- In order to use the Google Maps API, you must register your application on the Google Developer Console and enable the API.
- **Google Maps API :**
  - Is a robust tool , helps you take the power of Google Maps and put it directly on your own site.
  - It lets you add relevant content that is useful to your visitors and customize the look and feel of the map to fit with the style of your site.
  - **Api key :** need access to the Google Maps API key. Without this key, you will not be able to display Google Maps on your app.

# Steps to get API

- **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 SERVICES**.
- **Step 5:** Now open Google Map Android API
- **Step 6:** Now enable the Google Maps Android API.

- **Step 6:** Now go to **Credential**
- **Step 7:** Here click on Create credentials and choose API key

- 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.

- Now open **google\_maps\_api.xml** (debug) in values folder. Here enter your Google Maps API key in place of YOUR\_KEY\_HERE

# Permissions

- define internet and location permissions in Android Manifest.
- **INTERNET** – To determine if we are connected to Internet or not.
- **ACCESS\_FINE\_LOCATION** – To determine user's location using GPS. It will give us precise location.
- **ACCESS\_COARSE\_LOCATION** — Allows an app to access approximate location.
- `<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>`
- `<uses-permission android:name="android.permission.INTERNET"/>`



# MapsActivity.java file : callbacks in Google Maps:

- **OnMapReady Callback:** This callback is called when the map is ready to be used
- This is where we can add markers or lines, add listeners or move the camera
- @Override
- `public void onMapReady(GoogleMap googleMap) {}`
  
- **Connection/Suspend Callbacks:** This callback is called whenever device is connected and disconnected and implement `onConnected()` and `onConnectionSuspended()` functions.
  
- `//When the connect request has successfully completed`
- @Override
- `public void onConnected(Bundle bundle) {}`
  
- `//Called when the client is temporarily in a disconnected state.`
- @Override
  
- `public void onConnectionSuspended(int i) {`
- `}`
- **OnConnectionFailedListener:** Provides callbacks for scenarios that result in a failed attempt to connect the client to the service. Whenever connection is failed **`onConnectionFailed()`** will be called.
- @Override
- `public void onConnectionFailed(ConnectionResult connectionResult) {`
- `}`
- **LocationListener:** This callback have function **`onLocationChanged()`** that will be called whenever there is change in location of device.

# Method of Google Map

- Google map API methods help to customize google map
- `addMarker()` : add marker to map
- `addCircle()` : add circle to map
- `addPolygon()`:add polygon
- `getMyLocation()`: return the currently displayed user location.
- `Movecamera()`: reposition the camera
- `setTrafficEnabled()` : set traffic layer on or off.

# Zoom controls

- **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:
- **Eg : `mMap.getUiSettings().setZoomControlsEnabled(true);`**
- **Zoom Gestures:**
  - **Zoom In:** Double tap to increase the zoom level by 1.
  - **Zoom Out:** Two finger tap to decrease the zoom level by 1.
- Zoom control class for zoomin and zoomout functionality.
- **Methods:**
  - `Hide()`: hide ZoomControls from screen
  - `Show()`:show ZoomControls.

- **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);`
- **Zoom Gestures:**
  - **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);`
- **Compass:** You can set compass by calling below method
  - `mMap.getUiSettings().setCompassEnabled(true);`

# Changing the Map Type:

- `mMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);`
- `MAP_TYPE_NORMAL` : Represents a typical road map with street names and labels.
- `MAP_TYPE_SATELLITE`: Represents a Satellite View Area without street names and labels.
- `MAP_TYPE_TERRAIN`: Topographic data. The map includes colors, contour lines and labels, and perspective shading. Some roads and labels are also visible.
- `MAP_TYPE_HYBRID` : Combines a satellite View Area and Normal mode displaying satellite images of an area with all labels.
- `Map_TYPE_NONE` : No tiles. It is similar to a normal map, but doesn't display any labels or coloration for the type of environment in an area.

# Location Based Services

- To update location
- **requestLocationUpdates() :**
- Create Instance of LocationManager class and request location updates using **requestLocationUpdates()** method.
- Eg : `locationmgr.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0 , 0 , this);`
- Set type of location provider , number of seconds , distance and location listener object over which the location to be updated.
- **onLocationChanged() call back method :**
- Whenever user's location is changed. For that Google has predefined function **onLocationChanged** that will be called as soon as user's location change. Here we are getting the coordinates of current location using `getLatitude()` and `getLongitude()`
- Eg : `public void onLocationChanged(@NonNull Location location)`
- **Location class Object (location) :** it signifies a geographic location which consist of latitude , longitude , time stamp.
- **Methods of Location Class :**
- `getLatitude()`: get latitude in degree.
- `getLongitude()`: get longitude in degree.
- `getAccuracy()`: get accuracy in meters.
- `getSpeed()`: get speed in meters/second .

# Show marker on a location

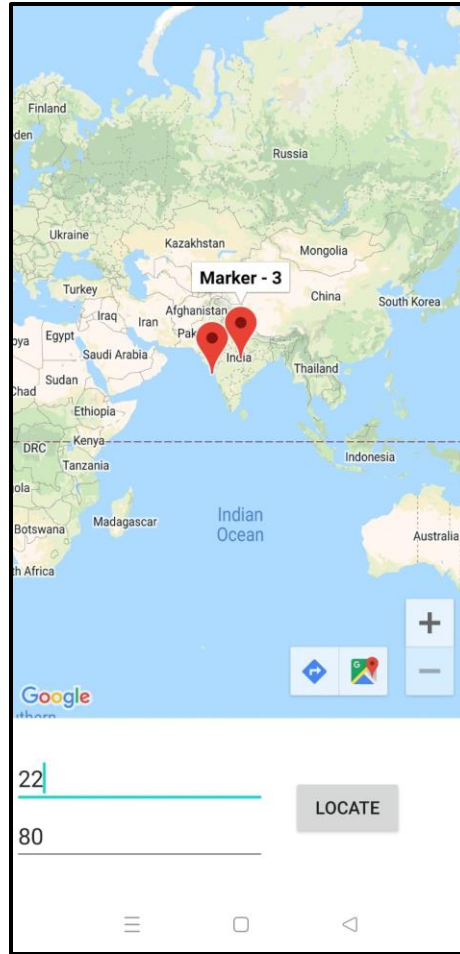
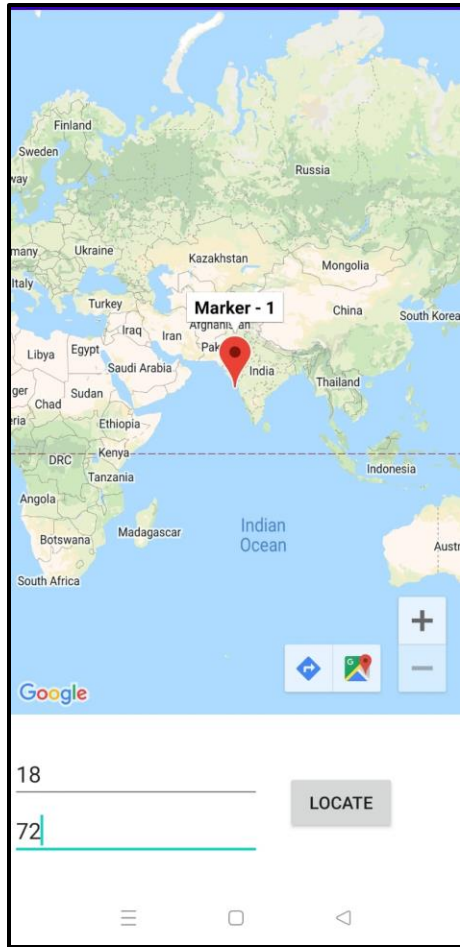
- **Addmarker ():**
- Create instance of GoogleMap and call AddMarker method . Use LatLng class object to define the position to add the marker.
- *//This callback is triggered when the map is ready to be used.*
- ```
public void onMapReady(GoogleMap googleMap) {  
    mMap = googleMap;  
    // Add a marker in Sydney and move the camera  
    LatLng sydney = new LatLng(-34, 151); // define instance of LatLng class and add position  
    mMap.addMarker(new MarkerOptions().position(sydney).title("Marker in Sydney"));  
    mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
```

Steps to get location in Android :

- **1. Provide permission in manifest for receiving location update.**
- `<uses-permission android:name="android.permission.INTERNET" />`
- `<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />`
- `<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"
/>`
- `<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />`
- `<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />`
- `<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />`

- **2. Create location Manager instance as reference to the location service.**
- Eg :
`location_Manager=(LocationManager)getSystemService(Context.LOCATION_SERVICE) // LOCATION_SERVICE reference will be created.`
- **3.Request current location from location manager:**
- `// location updates are requested`
- `location_Manager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, this);`
- Parameters : location provider , number of seconds , distance ,
- object over which the location to be updated
- **4. Receive location update**
- Update is notified.
- `onLocationChanged(Location location)`
- `Text1.setText(("Latitude" + location.getLatitude() + "Longitude : " +location.getLongitude()));`

Navigating to a Specific Location, Set/Unset Zoom Controls



Navigating to a Specific Location, Set/Unset Zoom Controls

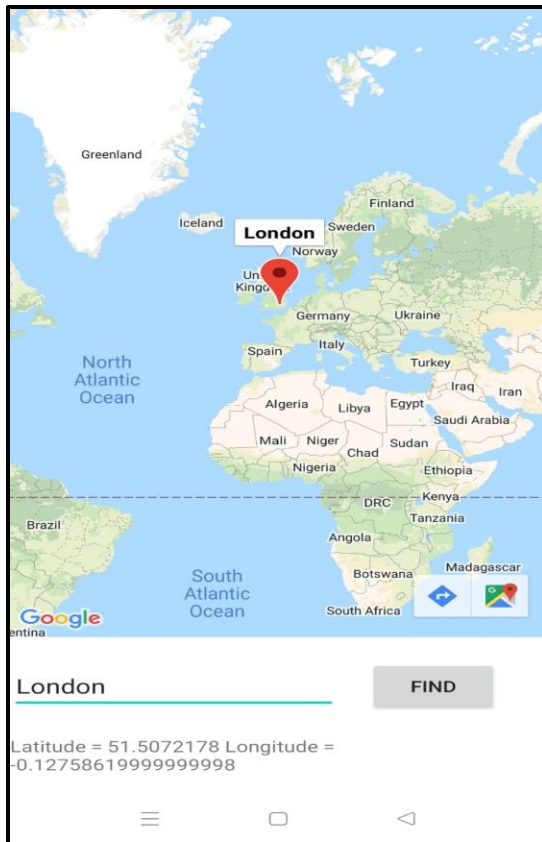
- public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {
 private GoogleMap mMap;
 private ActivityMapsBinding binding;
 EditText la, lo;
 int count = 0;
 Button b1;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 binding = ActivityMapsBinding.inflate(getLayoutInflater());
 setContentView(binding.getRoot());
 // 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);
 }
}

- ```
public void onMapReady(GoogleMap googleMap) {
 mMap = googleMap;
 lo = findViewById(R.id.ed_long);
 la = findViewById(R.id.ed_lat);
 b1 = findViewById(R.id.b1);
 b1.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 count++;
```
- ```
//LatLang is a class
LatLng temp = new LatLng(Float.parseFloat(la.getText().toString()),
Float.parseFloat(lo.getText().toString()));
```
- ```
mMap.addMarker(new MarkerOptions().position(temp).title("Marker - " + count));
mMap.moveCamera(CameraUpdateFactory.newLatLng(temp));
 }
});
```
- ```
mMap.getUiSettings().setZoomControlsEnabled(true);
//displays + - option used to zoom in & out on map (default - false)
```
- ```
mMap.getUiSettings().setZoomGesturesEnabled(false);
//sets the pinch in & out gestures (default - true)
```
- ```
mMap.getUiSettings().setCompassEnabled(false);
//sets the compass gesture, visible when map is rotated (default - true)
```

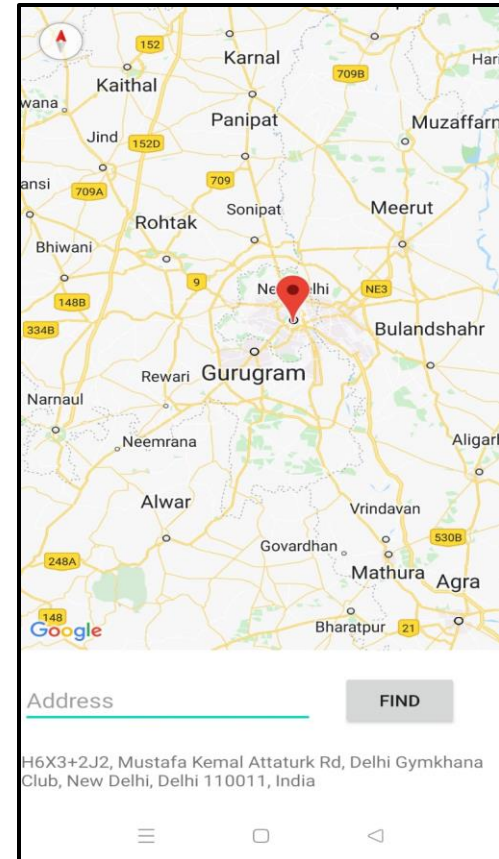
Geocode and Reverse Geocoding

- Geocode : finding the geographical co ordinate (latitude and longitude) of a given location is called Geocode.
- Reverse Geocode : opposite of geocode.
- Pair of latitude and longitude is converted into location called Reverse Geocode.
- **Methods of Geocoder class**
- **getFromLocationName()**
- **getFromLocation()**

- GeoCode



Reverse GeoCode



Geocoding & Reverse Geocoding

- ```
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {
 private GoogleMap mMap;
 private ActivityMapsBinding binding;
 private Geocoder geo;
 EditText ed_ad;
 Button b1;
 TextView t1;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 binding = ActivityMapsBinding.inflate(getLayoutInflater());
 setContentView(binding.getRoot());

 // 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);

 geo = new Geocoder(this);
 ed_ad = findViewById(R.id.ed_ad);
 b1 = findViewById(R.id.b1);
 t1 = findViewById(R.id.t1);
 }
}
```

- **//Geocoding to find lat,long from address**

```
b1.setOnClickListener(new View.OnClickListener() {
 @Override
 public void onClick(View v) {
 try {
List<Address> addresses = geo.getFromLocationName(ed_ad.getText().toString(), 1);
 if(addresses.size()>0){
 Address ads = addresses.get(0); // get value of 0th index address list like : roadno
```

- ```
        LatLng latlo = new LatLng(ads.getLatitude(), ads.getLongitude());  
        t1.setText("Latitude = "+ads.getLatitude()+" Longitude = "+ads.getLongitude());  
        mMap.addMarker(new MarkerOptions().position(latlo).title(ads.getLocality()));  
        mMap.moveCamera(CameraUpdateFactory.newLatLng(latlo));  
    }  
} catch (IOException e) {  
    e.printStackTrace();  
}  
}  
});
```


//REVERSE GEO-CODING (to find address from lat,long)

```
mMap.setOnMapLongClickListener(new GoogleMap.OnMapLongClickListener() {
    @Override
    public void onMapLongClick(@NonNull LatLng latLng) {
        Toast.makeText(getApplicationContext(), latLng.toString(),
        Toast.LENGTH_SHORT).show();
        try {
            List<Address> addresses = geo.getFromLocation(latLng.latitude, latLng.longitude, 1);
            if (addresses.size()>0){
                Address ads = addresses.get(0);
                String txt = ads.getAddressLine(0);
                mMap.addMarker(new MarkerOptions().position(latLng).title(txt));
                t1.setText(txt);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
});
```

Login Validation

- **public class MainActivity extends AppCompatActivity {**
 Button **b1,b2;**
 EditText **ed1,ed2;**
 TextView **tx1;**
 int counter = 3;
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.**activity_main**);
 b1 = (Button)findViewById(R.id.button);
 ed1 = (EditText)findViewById(R.id.editText);
 ed2 = (EditText)findViewById(R.id.editText2);

 b2 = (Button)findViewById(R.id.button2);
 tx1 = (TextView)findViewById(R.id.textView3);
 tx1.setVisibility(View.GONE);
 }



- b1.setOnClickListener(**new** View.OnClickListener() {
 @Override

• **public void** onClick(View v) {
 if(ed1.getText().toString().equals("admin") &&
 ed2.getText().toString().equals("admin")) {
 Toast.makeText(getApplicationContext(),
 "**Redirecting...**", Toast.LENGTH_SHORT).show();
 } else {
 Toast.makeText(getApplicationContext(), "**Wrong Credentials**", Toast.LENGTH_SHORT).show();

 tx1.setVisibility(View.VISIBLE);
 tx1.setBackgroundColor(Color.RED);
 counter--;
 tx1.setText(Integer.toString(counter));

 if (counter == 0) {
 b1.setEnabled(false);
 }
 }
 }
});

Android Security Model :Permission

- The purpose of a permission is to protect the privacy of an android user.
- App permissions help support user privacy by protecting access to the following:
 - **Restricted data**, such as system state and a user's contact information.
 - **Restricted actions**, such as connecting to a paired device and recording audio.
- Android apps must request permissions to access sensitive user data(contact ,SMS) as well as certain system features(camera , internet).
- To make use of protected features of device , must include in AndroidManifest.xml <uses-permission> tags declaring permissions.
- Permissions are divided into several protection levels
- Protection levels :
 - --normal
 - --signature
 - --dangerous

- `<manifest xmlns:android="http://schemas.android.com/apk/res/android"`
- `package="com.example.practiceprograms">`
- `<uses-permission android:name="android.permission.SEND_SMS"`
- `android:label="@string/label_cellphone"`
- `android:description="@string/desc_cellphone"`
- `android:permissionGroup="android.permission-group.COST_MONEY"`
- `android:protectionLevel="dangerous" />`
- `</manifest>`

- `<string name="label_cellphone">directly call phone number</string>`
- `<string name="desc_cellphone">allows the app to call</string>`

- **Label and description** should be supplied for the permission. These are string resources that can be displayed to the user when they are viewing a list of permissions.
- **Label** should be short describing the key piece of the functionality the permission is protecting.
- **Description** : what the permission allows a holder to do.

- **permissionGroup** : The permission-group tag allows you to create a group of custom permissions. Declares a name for a logical grouping of related permissions.
- If user allows permission from a group and your app requests for second permission from the same group, system will automatically grant the permission.
- **MESSAGES** : The messages related permissions, say `android.permission.SEND_SMS`, `RECEIVE_SMS` and all the permissions related to messages are grouped under **`android.permission-group.MESSAGES`**
- For example, System will automatically grant `WRITE_CALENDAR` permission without prompting for user acceptance. This is because `READ_CALENDAR` and `WRITE_CALENDAR` permissions belong to **`CALENDAR`** group.

| Permission Group | Description |
|-------------------------|------------------------------------|
| Calendar | Managing calendars |
| Camera | Taking photos and recording videos |
| Contacts | Managing contacts |
| Location | Current device location |
| Microphone | Audio recording |
| Phone | Dialing and managing phone calls |

Example of permission group

- `<permission-group android:name="android.permission-group.MESSAGES"`
- `android:label="@string/permgrouplab_messages"`
- `android:icon="@drawable/perm_group_messages"`
- `android:description="@string/permgroupdesc_messages"/>`

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

Protection levels

- **1. Normal :** cover areas where our app needs to access data outside the apps sandbox , but there are very little risk to the users privacy
- Eg : permission to set Time zone is a normal permission. If app declare in its manifest that it needs a normal permission , the system automatically grants the app that permission at install time.
- Like : ACCESS_NETWORK_STATE , SET_ALARM ,SET_WALLPAPER , INTERNET
- **2. Signature :** the system grant these permissions at install time but only when the app that attempt to use permission is signed by the same certificate as the app that defines the permissions.
- It's granted automatically by the system if both applications are signed with the same certificate
- LIKE : BIND_ACCESSIBILITY_SERVICE , WRITE_SETTINGS , WRITE_VOICEMAIL
- **3. Dangerous :** cover areas where the app want data that involves the users private information.
- Eg : access to Contact , SMS.
- If an app declare that it needs a dangerous permission , the user has to explicitly grant the permission .
- LIKE : READ_CONTACT , WRITE_CONTACT , RECORD_AUDIO

Custom Permission

- Android allows defining custom permission ..
- If you wanted to avoid certain users from starting one of the activities in your application , programmer could do that by defining custom permission.
- Define custom permissions for your app to share services with other apps. Like services , broadcast receivers

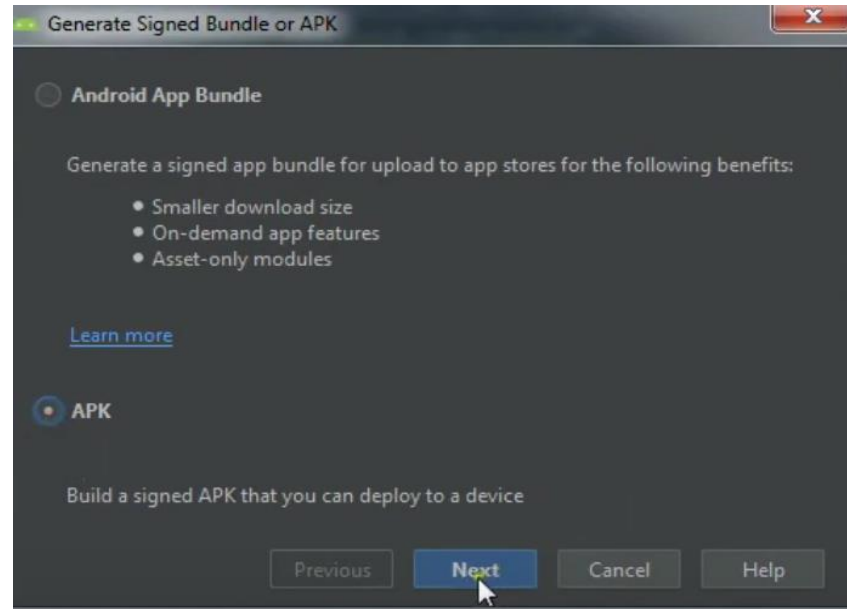
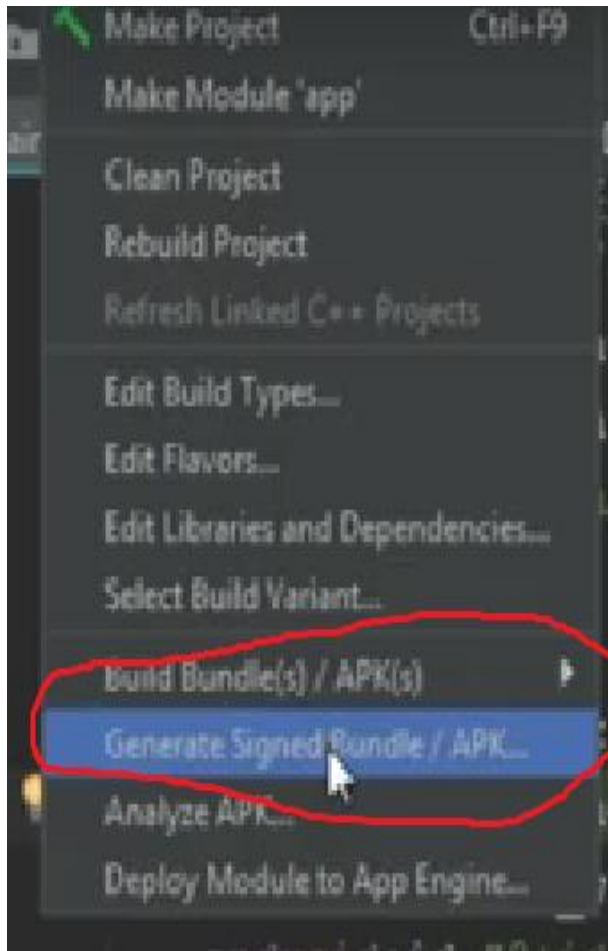
Become a Publisher

- 2 main task :
- Prepare the application for release
 - Configuring app for release
 - Building and signing a release version
 - Testing the release version
 - Updating app resource for release
 - Prepare remote servers and services on which app depend on.
- Release the application to users
 - Release through an app market place
 - Release through app google play

Generate Signed APK for Android App

- The Android system needs that all installed applications be digitally signed with a certificate whose private key is owned by the application's developer.
- The Android system applies the certificate as a means of recognizing the author of an application and establishing trust relations between applications.
- **Signed Apk** generates a key and this key can be used to release versions of the app, so it is important to save this key which will be used when the next version of the app is released.
- When developers are ready to publish the android application for end-users, they are required to sign it with a suitable private key.
- Applications can only be installed when they are signed. Android does not allow unsigned applications to get installed.
- Link :
<https://www.bing.com/videos/search?q=how+to+generate+signed+apk+in+android+studio&view=detail&mid=BFF19BBDE0A9374C0418BFF19BBDE0A9374C0418&FORM=VIRE>

Build -> Generate Signed Bundle / APK



| | | | |
|-------------|-------------------|-----------|-----------|
| app-release | 7/20/2019 2:20 PM | APK File | 26,210 KB |
| output.json | 7/20/2019 2:20 PM | JSON File | 1 KB |

How to publish App on Play Store

- First [generate signed apk](#) of your Android App to publish it on Play Store.
- Now you will need to sign up for Google Play Console to publish and manage your Android App.
- Login with your Gmail account that you want to use for publishing App on Play Store.

- After reading the Google play store developer distribution agreement agree to their terms by clicking on check box

The screenshot displays the Google Play Developer account setup interface. It is divided into three main sections:

- Accept developer agreement:** This section includes a document icon, the heading "Accept developer agreement", and the text "Read and agree to the [Google Play Developer distribution agreement](#)". Below this, there is a checked checkbox with the text "I agree and I am willing to associate my account with the Google Play Developer distribution agreement". A red arrow points to this checkbox. A red text overlay below the checkbox reads "First Click on check box then click on Continue to payment".
- Review distribution countries:** This section includes a globe and dollar sign icon, the heading "Review distribution countries", and the text "Review the distribution countries where you can distribute and sell applications. If you are planning to sell apps or in-app products, check if you can have a merchant account in your country."
- Credit card:** This section includes a credit card icon, the heading "Credit card", and the text "Make sure you have your credit card handy to pay the \$25 registration fee in the next step."

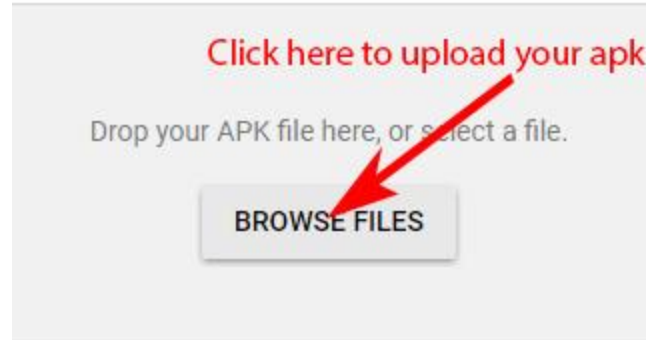
At the bottom of the form, there is a blue button labeled "CONTINUE TO PAYMENT".

- Now you will need to pay one time 'Developer Registration Fee' of \$25 to Google.
- Important Note: You can upload unlimited number of Android App on Play store from single account with a limit of uploading 15 apk/day.
- Complete your account details for Google developer account. Now click on Create Application For example see the below image:

- Enter the name of your App.
- Now fill store listing details of your App which include Title, Short description, and Full description.

- After this you need to put some App screenshots here. The minimum required are 2 screenshots and maximum limit is 8.
- After screenshot now you need to put a high Resolution icon or logo with a size of 512 * 512 pixel. This will be displayed on Play Store.
- Now scroll down and fill other details which include application type, category, website, email and phone no.
- After this check privacy policy because now we are not submitting and then click on save draft. If your App require user permission then it is mandatory to put privacy url.
- Click on Save Draft to save your work so far.
- After saving data on draft now go to app release and click on manage production.
- Now you will see create release now click on it.

- After click on create release you will see browse files click on it and upload your signed APK



- Once the upload is successful then scroll down and click on review to check..
- Now go to Content Rating and click on continue.

- Fill details which include email address and select your categories.
- Now click on apply rating
- Click on pricing and distribution and select free/paid based on how you want user to access your App.

- Now scroll down and see mandatory things with * you need to select After this click on save draft
- Now Click on ready on publish along with save draft and click on Manage release..
- Click on Manage Production.
- After Manage production click on edit release.