UNIT 6

- **SMS telephony:**

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

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
text: You need to specify the content of your message.
sentIntent: specify Pending Intent to invoke when the message is successfully sent
deliveryIntent: specify the Pending Intent to invoke when the message is delivered to the recipient.

Manifest File :
<uses-permission android:name="android.permission.SEND_SMS"/>

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.

Example : 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.

The difference between PendingIntent and Intent: An Intent is something that is used right now; a PendingIntent is something that may create an Intent in the future. You will use a PendingIntent with Notifications

Permission in Manifest :
<uses-permission android:name="android.permission.SEND_SMS" />

SMS by Intent : 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");
Intent.putExtra("sms_body" , "Test SMS to Angilla");
startActivity(sendIntent);

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 faild, 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

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);

Intent Object - Data/Type
emailIntent.setData(Uri.parse("mailto:")); emailIntent.setType("text/plain");
```

Intent Object - Extra
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.

```
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(); } }
```

- **Google map:**

Method of Google Map to customize 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 Control

State syntax to display built in zoom control. (2M)

a) **Built in Zoom control in Google map can be displayed with** :

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:

UiSettings.setZoomControlsEnabled(true);

b) In any normal activity, **ZoomControl can be displayed as component** by following syntax :

```
ZoomControl zoomControls = (ZoomControl) findViewById(R.id.simpleZoomControl);
zoomControls.show()
```

Methods:

Hide(): hide ZoomControls from screen

Show():show ZoomControls.

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 : locationmgr1.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.

// update when the location has changed by distance in meters, AND byTime in milliseconds have passed.

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 (**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 postion
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.ACCESS_COARSE_LOCATION" />
 <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
```

2. Create location Manager instance as reference to the location service.

Eg :
location_Manager1=(LocationManager)getSystemService(Context.LOCATION_SERVICE) // LOCATION_SERVICE reference will be created.

3.Request current location from location manager:

// location updates are requested

location_Manager1.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());

To display Map

1. Map view : use <mapView> in XML

Java code :

```
public class MapsActivity extends AppcompatActivity implements
OnMapReadyCallback {
MapView mapview1;
Void onCreate(Bundle  saveInstanceState) {
mapview1=findViewById(R.id.map_view);
mapview1.getMapAsync(this);
}
```

2. SupportMapFragment : create google map activity . A Map component in an app . automatically handle the necessary life cycle needs. Being a fragment, this component can be added to an activity's layout file simply with the XML below.

XML :

```
<fragment
   name="com.google.android.gms.maps.SupportMapFragment"
   android:layout_width="match_parent"
   android:layout_height="match_parent“
   android:id=“@+id/fragment”
/>
```

Java code :

```
public class MapsActivity extends AppcompatActivity implements
OnMapReadyCallback {
Void onCreate(Bundle  saveInstanceState) {
 SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.fragment);
mapFragment.getMapAsync(this);
}}
```

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()

Geocoder : A class for handling geocoding and reverse geocoding.

Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate.

Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into a (partial) address.

The amount of detail in a reverse geocoded location description may vary, for example one might contain the full street address of the closest building, while another might contain only a city name and postal code.

a. getFromLocation

Syntax

public List<Address> getFromLocation (double latitude, double longitude, int maxResults)

This method returns an array of Addresses that attempt to describe the area immediately surrounding the given latitude and longitude.

b. **getFromLocationName**

Syntax :

- public List<Address> getFromLocationName (String locationName, int maxResults)

Returns an array of Addresses that attempt to describe the named location, which may be a place name such as "Dalvik, Iceland", an address such as "1600 Amphitheatre Parkway, Mountain View, CA", an airport code such as "SFO", and so forth.

- **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 SMS 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 :

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 on. If app declare in its manifest that it needs a normal permission , the system automatically grants the app that permission at installation 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
if you write App A that defends itself with a signature-level permission (e.g., a custom one), and you write App B that wants to talk to the defended portions of App A, you can do so, if you are signing App A and App B with the same signing key.
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

```
<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”> send SMS</string>
<string name=“desc_cellphone”>allows the app to send SMS</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 The label specifies the name of the permissions to be displayed to users
Description : defines a summary that will be shown to the end-user.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

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

- **Become a Publisher : Publishing is a process that makes your Android app available to users:**

  2 main task :

  1. Prepare the application for release

  Configuring app for release

  Building and signing a release version

  Testing the release version : Before you distribute your app, you should thoroughly test the release version on at least one target handset device and one target tablet device.

  Updating app resource for release : Make sure that all app resources, such as multimedia files and graphics, are updated

  Prepare remote servers and services on which app depend on : if your app depends on external servers or services, make sure they are secure and production ready.

  2.Release the application to users

  Release through an app market place : If you want to distribute your apps to the broadest possible audience such as Google Play

  1. Prepare promotional materials: To fully leverage the marketing and publicity capabilities of Google Play, you need to create promotional materials for your app such as screenshots, videos, graphics.

  2. Configure options and uploading assets : By configuring various Google Play settings, you can choose the countries you want to reach, the listing languages you want to use, and the price you want to charge in each country.

  3. Publish the release version of your app :

  Release through a website : make the app available for download on your own website

  1. Prepare your app for release.
  2. Host the release-ready APK file on your website.
  3. Provide a download link to users.

- **List and elaborate steps to deploy an Android application on Google play store:**

Step 1: Create a Developer Account

Before you can publish any app on Google Play, you need to create a Developer Account. need to pay a one-time registration fee of $25

Step 2: Plan to Sell? Link Your Merchant Account

If you want to publish a paid app or plan to sell in-app purchases, you need to create a payments center profile, i.e. a merchant account.

manage your app sales and monthly payouts, as well as analyze your sales reports

Step 3: Create an App

Now you have create an application by clicking on 'Create Application'. Here you have to select your app's default language and type title . Title will show on Google Play after you've published

Step 4: Prepare Store Listing

details that will show up to customers on your app's listing on Google Play.

a.Product Details containing title, short and full description of the app,

Use the right keywords, but don't overdo it. Make sure your app doesn't come across as spam-y or promotional, or it will risk getting suspended on the Play Store.

b. Graphic Assets where you can add screenshots, images, videos, promotional graphics, and icons that showcase your app's features and functionality.

c.Languages & Translations, Categorization where in category can be selected to which your app belong to. Contact Details , Privacy Policy for apps that request access to sensitive user data or permissions,

Step 5: Upload APK to an App Release

Finally upload your app, by uploading APK file. Before you upload APK, you need to create an app release. You need to select the type of release you want to upload your first app version to. You can choose between an internal test, a closed test, an open test, and a production release. The first three releases allow you to test out your app among a select

Step 6: Provide an Appropriate Content Rating

If you don't assign a rating to your app, it will be listed as 'Unrated'. Apps that are 'Unrated' may get removed from Google Play.

To rate your app, you need to fill out a content rating questionnaire

Step 7: Set Up Pricing & Distribution

set up your app as free or paid.

You can always change your app from paid to free later, but you cannot change a free app to paid. For that, you'll need to create a new app and set its price.

Step 8: Rollout Release to Publish Your App

Once you're sure about the correctness of the details, select your app and navigate to 'Release management' – 'App releases.' You can always opt for reviews by clicking on 'Review' to be taken to the 'Review and rollout release' screen. Here, you can see if there are any issues or warnings you might have missed out on.

Finally, select 'Confirm rollout.'