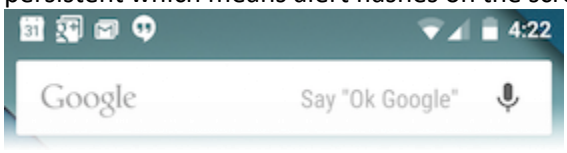
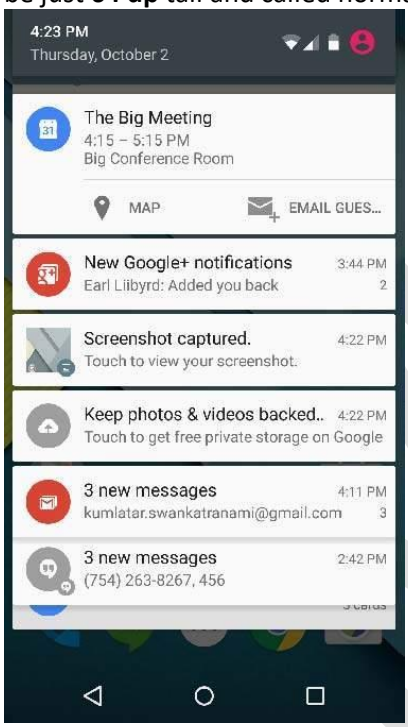
Android - Notifications

A **notification** is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Android **Toast** class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.



To see the details of the notification, you will have to select the icon which will display notification drawer having detail about the notification. While working with emulator with virtual device, you will have to click and drag down the status bar to expand it which will give you detail as follows. This will be just **64 dp** tall and called normal view.



Above expanded form can have a **Big View** which will have additional detail about the notification. You can add upto six additional lines in the notification. The following screen shot shows such notification.

Create and Send Notifications

You have simple way to create a notification. Follow the following steps in your application to create a notification −

Step 1 - Create Notification Builder

As a first step is to create a notification builder using *NotificationCompat.Builder.build()*. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)
```

Step 2 - Setting Notification Properties

Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following –

- A small icon, set by **setSmallIcon()**
- A title, set by **setContentTitle()**
- Detail text, set by **setContentText()**

```
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification Detail!");
```

You have plenty of optional properties which you can set for your notification. To learn more about them, see the reference documentation for NotificationCompat.Builder.

Step 3 - Attach Actions

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an **Activity** in your application, where they can look at one or more events or do further work.

The action is defined by a **PendingIntent** containing an **Intent** that starts an Activity in your application. To associate the PendingIntent with a gesture, call the appropriate method of *NotificationCompat.Builder*. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the PendingIntent by calling **setContentIntent()**.

A PendingIntent object helps you to perform an action on your applications behalf, often at a later time, without caring of whether or not your application is running.

We take help of stack builder object which will contain an artificial back stack for the started Activity. This ensures that navigating backward from the Activity leads out of your application to the Home screen.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class);

// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);
```

Step 4 - Issue the notification

Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification. Make sure you call **NotificationCompat.Builder.build()** method on builder object before notifying it. This method combines all of the options that have been set and return a new **Notification** object.

```
NotificationManager mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID, mBuilder.build());
```

The NotificationCompat.Builder Class

The NotificationCompat.Builder class allows easier control over all the flags, as well as help constructing the typical notification layouts. Following are few important and most frequently used methods available as a part of NotificationCompat.Builder class.

Sr.No.	Constants & Description
1	**Notification build()** Combine all of the options that have been set and return a new Notification object.
2	**NotificationCompat.Builder setAutoCancel (boolean autoCancel)** Setting this flag will make it so the notification is automatically canceled when the user clicks it in the panel.
3	**NotificationCompat.Builder setContent (RemoteViews views)** Supply a custom RemoteViews to use instead of the standard one.
4	**NotificationCompat.Builder setContentInfo (CharSequence info)** Set the large text at the right-hand side of the notification.
5	**NotificationCompat.Builder setContentIntent (PendingIntent intent)** Supply a PendingIntent to send when the notification is clicked.
6	**NotificationCompat.Builder setContentText (CharSequence text)** Set the text (second row) of the notification, in a standard notification.
7	**NotificationCompat.Builder setContentTitle (CharSequence title)** Set the text (first row) of the notification, in a standard notification.
8	**NotificationCompat.Builder setDefaults (int defaults)** Set the default notification options that will be used.
9	**NotificationCompat.Builder setLargeIcon (Bitmap icon)** Set the large icon that is shown in the ticker and notification.
10	**NotificationCompat.Builder setNumber (int number)** Set the large number at the right-hand side of the notification.
11	**NotificationCompat.Builder setOngoing (boolean ongoing)** Set whether this is an ongoing notification.
12	**NotificationCompat.Builder setSmallIcon (int icon)**

	Set the small icon to use in the notification layouts.
13	**NotificationCompat.Builder setStyle (NotificationCompat.Style style)** Add a rich notification style to be applied at build time.
14	**NotificationCompat.Builder setTicker (CharSequence tickerText)** Set the text that is displayed in the status bar when the notification first arrives.
15	**NotificationCompat.Builder setVibrate (long[] pattern)** Set the vibration pattern to use.
16	**NotificationCompat.Builder setWhen (long when)** Set the time that the event occurred. Notifications in the panel are sorted by this time.

Example

Following example shows the functionality of a Android notification using a **NotificationCompat.Builder** Class which has been introduced in Android 4.1.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *tutorialspoint* under a package *com.example.notificationdemo*.
2	Modify *src/MainActivity.java* file and add the code to notify(""), if user click on the button,it will call android notification service.
3	Create a new Java file *src/NotificationView.java*, which will be used to display new layout as a part of new activity which will be started when user will click any of the notifications
4	Modify layout XML file *res/layout/activity_main.xml* to add Notification button in relative layout.
5	Create a new layout XML file *res/layout/notification.xml*. This will be used as layout file for new activity which will start when user will click any of the notifications.
6	No need to change default string constants. Android studio takes care of default string constants
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.notificationdemo/MainActivity.java**. This file can include each of the fundamental lifecycle methods.

```
package com.example.notificationdemo;

import android.app.Activity;
```

```
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.support.v4.app.NotificationCompat;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {
  Button b1;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    b1 = (Button)findViewById(R.id.button);
    b1.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        addNotification();
      }
    });
  }

  private void addNotification() {
    NotificationCompat.Builder builder =
      new NotificationCompat.Builder(this)
      .setSmallIcon(R.drawable.abc)
      .setContentTitle("Notifications Example")
      .setContentText("This is a test notification");

    Intent notificationIntent = new Intent(this, MainActivity.class);
    PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent,
      PendingIntent.FLAG_UPDATE_CURRENT);
    builder.setContentIntent(contentIntent);

    // Add as notification
    NotificationManager manager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
    manager.notify(0, builder.build());
  }
}
```

Following will be the content of **res/layout/notification.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" >
```

```
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="400dp"
    android:text="Hi, Your Detailed notification view goes here...." />
</LinearLayout>
```

Following is the content of the modified main activity file **src/com.example.notificationdemo/NotificationView.java**.

```
package com.example.notificationdemo;

import android.os.Bundle;
import android.app.Activity;

public class NotificationView extends Activity{
  @Override
  public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.notification);
  }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="MainActivity">

  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Notification Example"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_below="@+id/textView1"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="48dp" />
```

```
  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="42dp" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Notification"
    android:id="@+id/button"
    android:layout_marginTop="62dp"
    android:layout_below="@+id/imageButton"
    android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="action_settings">Settings</string>
  <string name="app_name">tutorialspoint </string>
</resources>
```

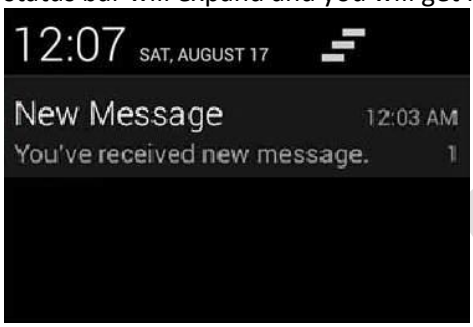
Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.notificationdemo.MainActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

    </activity>

    <activity android:name=".NotificationView"
      android:label="Details of notification"
      android:parentActivityName=".MainActivity">
```

```
      <meta-data
      android:name="android.support.PARENT_ACTIVITY"
      android:value=".MainActivity"/>
    </activity>

  </application>
</manifest>
```

Now click **button**, you will see at the top a message "New Message Alert!" will display momentarily and after that you will have following screen having a small icon at the top left corner.

Now lets expand the view, long click on the small icon, after a second it will display date information and this is the time when you should drag status bar down without releasing mouse. You will see status bar will expand and you will get following screen –



Big View Notification

The following code snippet demonstrates how to alter the notification created in the previous snippet to use the Inbox big view style. I'm going to update displayNotification() modification method to show this functionality –

```
protected void displayNotification() {
  Log.i("Start", "notification");

  /* Invoking the default notification service */
  NotificationCompat.Builder  mBuilder = new NotificationCompat.Builder(this);

  mBuilder.setContentTitle("New Message");
  mBuilder.setContentText("You've received new message.");
  mBuilder.setTicker("New Message Alert!");
  mBuilder.setSmallIcon(R.drawable.woman);

  /* Increase notification number every time a new notification arrives */
  mBuilder.setNumber(++numMessages);

  /* Add Big View Specific Configuration */
  NotificationCompat.InboxStyle inboxStyle = new NotificationCompat.InboxStyle();

  String[] events = new String[6];
  events[0] = new String("This is first line....");
  events[1] = new String("This is second line...");
  events[2] = new String("This is third line...");
  events[3] = new String("This is 4th line...");
  events[4] = new String("This is 5th line...");
```

```
  events[5] = new String("This is 6th line...");

  // Sets a title for the Inbox style big view
  inboxStyle.setBigContentTitle("Big Title Details:");

  // Moves events into the big view
  for (int i=0; i < events.length; i++) {
    inboxStyle.addLine(events[i]);
  }

  mBuilder.setStyle(inboxStyle);

  /* Creates an explicit intent for an Activity in your app */
  Intent resultIntent = new Intent(this, NotificationView.class);

  TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
  stackBuilder.addParentStack(NotificationView.class);

  /* Adds the Intent that starts the Activity to the top of the stack */
  stackBuilder.addNextIntent(resultIntent);
  PendingIntent resultPendingIntent
=stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);

  mBuilder.setContentIntent(resultPendingIntent);
  mNotificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

  /* notificationID allows you to update the notification later on. */
  mNotificationManager.notify(notificationID, mBuilder.build());
}
```

Location based Services

Android location APIs make it easy for you to build location-aware applications, without needing to focus on the details of the underlying location technology.

This becomes possible with the help of **Google Play services**, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

This tutorial shows you how to use Location Services in your APP to get the current location, get periodic location updates, look up addresses etc.

The Location Object

The **Location** object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information −

Sr.No.	Method & Description
1	**float distanceTo(Location dest)**

	Returns the approximate distance in meters between this location and the given location.
2	**float getAccuracy()** Get the estimated accuracy of this location, in meters.
3	**double getAltitude()** Get the altitude if available, in meters above sea level.
4	**float getBearing()** Get the bearing, in degrees.
5	**double getLatitude()** Get the latitude, in degrees.
6	**double getLongitude()** Get the longitude, in degrees.
7	**float getSpeed()** Get the speed if it is available, in meters/second over ground.
8	**boolean hasAccuracy()** True if this location has an accuracy.
9	**boolean hasAltitude()** True if this location has an altitude.
10	**boolean hasBearing()** True if this location has a bearing.
11	**boolean hasSpeed()** True if this location has a speed.
12	**void reset()** Clears the contents of the location.
13	**void setAccuracy(float accuracy)** Set the estimated accuracy of this location, meters.
14	**void setAltitude(double altitude)**

	Set the altitude, in meters above sea level.
15	**void setBearing(float bearing)** Set the bearing, in degrees.
16	**void setLatitude(double latitude)** Set the latitude, in degrees.
17	**void setLongitude(double longitude)** Set the longitude, in degrees.
18	**void setSpeed(float speed)** Set the speed, in meters/second over ground.
19	**String toString()** Returns a string containing a concise, human-readable description of this object.

Get the Current Location

To get the current location, create a location client which is **LocationClient** object, connect it to Location Services using **connect()** method, and then call its **getLastLocation()** method. This method returns the most recent location in the form of **Location** object that contains latitude and longitude coordinates and other information as explained above. To have location based functionality in your activity, you will have to implement two interfaces −

- GooglePlayServicesClient.ConnectionCallbacks
- GooglePlayServicesClient.OnConnectionFailedListener

These interfaces provide following important callback methods, which you need to implement in your activity class −

Sr.No.	Callback Methods & Description
1	**abstract void onConnected(Bundle connectionHint)** This callback method is called when location service is connected to the location client successfully. You will use **connect()** method to connect to the location client.
2	**abstract void onDisconnected()** This callback method is called when the client is disconnected. You will use **disconnect()** method to disconnect from the location client.
3	**abstract void onConnectionFailed(ConnectionResult result)** This callback method is called when there was an error connecting the client to the service.

You should create the location client in **onCreate()** method of your activity class, then connect it in **onStart()**, so that Location Services maintains the current location while your activity is fully visible. You should disconnect the client in **onStop()** method, so that when your app is not visible, Location Services is not maintaining the current location. This helps in saving battery power up-to a large extent.

Get the Updated Location

If you are willing to have location updates, then apart from above mentioned interfaces, you will need to implement **LocationListener** interface as well. This interface provide following callback method, which you need to implement in your activity class −

Sr.No.	Callback Method & Description
1	**abstract void onLocationChanged(Location location)** This callback method is used for receiving notifications from the LocationClient when the location has changed.

Location Quality of Service

The **LocationRequest** object is used to request a quality of service (QoS) for location updates from the **LocationClient**. There are following useful setter methods which you can use to handle QoS. There are equivalent getter methods available which you can check in Android official documentation.

Sr.No.	Method & Description
1	**setExpirationDuration(long millis)** Set the duration of this request, in milliseconds.
2	**setExpirationTime(long millis)** Set the request expiration time, in millisecond since boot.
3	**setFastestInterval(long millis)** Explicitly set the fastest interval for location updates, in milliseconds.
4	**setInterval(long millis)** Set the desired interval for active location updates, in milliseconds.
5	**setNumUpdates(int numUpdates)** Set the number of location updates.
6	**setPriority(int priority)** Set the priority of the request.

Now for example, if your application wants high accuracy location it should create a location request with **setPriority(int)** set to PRIORITY_HIGH_ACCURACY and **setInterval(long)** to 5 seconds. You can

also use bigger interval and/or other priorities like PRIORITY_LOW_POWER for to request "city" level accuracy or PRIORITY_BALANCED_POWER_ACCURACY for "block" level accuracy.
Activities should strongly consider removing all location request when entering the background (for example at onPause()), or at least swap the request to a larger interval and lower quality to save power consumption.

Displaying a Location Address

Once you have **Location** object, you can use **Geocoder.getFromLocation()** method to get an address for a given latitude and longitude. This method is synchronous, and may take a long time to do its work, so you should call the method from the **doInBackground()** method of an **AsyncTask** class.

The **AsyncTask** must be subclassed to be used and the subclass will override **doInBackground(Params...)** method to perform a task in the background and **onPostExecute(Result)** method is invoked on the UI thread after the background computation finishes and at the time to display the result. There is one more important method available in AyncTask which is **execute(Params... params)**, this method executes the task with the specified parameters.

Example

Following example shows you in practical how to to use Location Services in your app to get the current location and its equivalent addresses etc.
To experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you will have to struggle with emulator which may not work.
Create Android Application

Step	Description
1	You will use Android studio IDE to create an Android application and name it as package *com.example.tutorialspoint7.myapplication*.
2	add *src/GPSTracker.java* file and add required code.
3	Modify *src/MainActivity.java* file and add required code as shown below to take care of getting current location and its equivalent address.
4	Modify layout XML file *res/layout/activity_main.xml* to add all GUI components which include three buttons and two text views to show location/address.
5	Modify *res/values/strings.xml* to define required constant values
6	Modify *AndroidManifest.xml* as shown below
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **MainActivity.java**.

```
package com.example.tutorialspoint7.myapplication;

import android.Manifest;
import android.app.Activity;
import android.os.Bundle;
import android.support.v4.app.ActivityCompat;
```

```
import android.test.mock.MockPackageManager;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

  Button btnShowLocation;
  private static final int REQUEST_CODE_PERMISSION = 2;
  String mPermission = Manifest.permission.ACCESS_FINE_LOCATION;

  // GPSTracker class
  GPSTracker gps;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    try {
      if (ActivityCompat.checkSelfPermission(this, mPermission)
          != MockPackageManager.PERMISSION_GRANTED) {

        ActivityCompat.requestPermissions(this, new String[]{mPermission},
            REQUEST_CODE_PERMISSION);

        // If any permission above not allowed by user, this condition will
           execute every time, else your else part will work
      }
    } catch (Exception e) {
      e.printStackTrace();
    }

    btnShowLocation = (Button) findViewById(R.id.button);

    // show location button click event
    btnShowLocation.setOnClickListener(new View.OnClickListener() {

      @Override
      public void onClick(View arg0) {
        // create class object
        gps = new GPSTracker(MainActivity.this);

        // check if GPS enabled
        if(gps.canGetLocation()){

          double latitude = gps.getLatitude();
          double longitude = gps.getLongitude();

          // \n is for new line
          Toast.makeText(getApplicationContext(), "Your Location is - \nLat: "
```

```
            + latitude + "\nLong: " + longitude, Toast.LENGTH_LONG).show();
        }else{
          // can't get location
          // GPS or Network is not enabled
          // Ask user to enable GPS/network in settings
          gps.showSettingsAlert();
        }

      }
    });
  }
}
```

Following is the content of the modified main activity file **GPSTracker.java**.

```
package com.example.tutorialspoint7.myapplication;

import android.app.AlertDialog;
import android.app.Service;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.provider.Settings;
import android.util.Log;

public class GPSTracker extends Service implements LocationListener {

  private final Context mContext;

  // flag for GPS status
  boolean isGPSEnabled = false;

  // flag for network status
  boolean isNetworkEnabled = false;

  // flag for GPS status
  boolean canGetLocation = false;

  Location location; // location
  double latitude; // latitude
  double longitude; // longitude

  // The minimum distance to change Updates in meters
  private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 10; // 10 meters

  // The minimum time between updates in milliseconds
  private static final long MIN_TIME_BW_UPDATES = 1000 * 60 * 1; // 1 minute
```

```
// Declaring a Location Manager
protected LocationManager locationManager;

public GPSTracker(Context context) {
  this.mContext = context;
  getLocation();
}

public Location getLocation() {
  try {
    locationManager = (LocationManager) mContext.getSystemService(LOCATION_SERVICE);

    // getting GPS status
    isGPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

    // getting network status
    isNetworkEnabled = locationManager
      .isProviderEnabled(LocationManager.NETWORK_PROVIDER);

    if (!isGPSEnabled && !isNetworkEnabled) {
      // no network provider is enabled
    } else {
      this.canGetLocation = true;
      // First get location from Network Provider
      if (isNetworkEnabled) {
        locationManager.requestLocationUpdates(
          LocationManager.NETWORK_PROVIDER,
          MIN_TIME_BW_UPDATES,
          MIN_DISTANCE_CHANGE_FOR_UPDATES, this);

        Log.d("Network", "Network");
        if (locationManager != null) {
          location = locationManager
            .getLastKnownLocation(LocationManager.NETWORK_PROVIDER);

          if (location != null) {
            latitude = location.getLatitude();
            longitude = location.getLongitude();
          }
        }
      }

      // if GPS Enabled get lat/long using GPS Services
      if (isGPSEnabled) {
        if (location == null) {
          locationManager.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,
            MIN_TIME_BW_UPDATES,
            MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
```

```
            Log.d("GPS Enabled", "GPS Enabled");
            if (locationManager != null) {
              location = locationManager
                .getLastKnownLocation(LocationManager.GPS_PROVIDER);

              if (location != null) {
                latitude = location.getLatitude();
                longitude = location.getLongitude();
              }
            }
          }
        }
      }

    } catch (Exception e) {
      e.printStackTrace();
    }

    return location;
  }

  /**
    * Stop using GPS listener
    * Calling this function will stop using GPS in your app
  * */

  public void stopUsingGPS(){
    if(locationManager != null){
      locationManager.removeUpdates(GPSTracker.this);
    }
  }

  /**
    * Function to get latitude
  * */

  public double getLatitude(){
    if(location != null){
      latitude = location.getLatitude();
    }

    // return latitude
    return latitude;
  }

  /**
    * Function to get longitude
  * */

  public double getLongitude(){
    if(location != null){
```

```
        longitude = location.getLongitude();
    }

    // return longitude
    return longitude;
}

/**
 * Function to check GPS/wifi enabled
 * @return boolean
* */

public boolean canGetLocation() {
    return this.canGetLocation;
}

/**
 * Function to show settings alert dialog
 * On pressing Settings button will lauch Settings Options
* */

public void showSettingsAlert(){
    AlertDialog.Builder alertDialog = new AlertDialog.Builder(mContext);

    // Setting Dialog Title
    alertDialog.setTitle("GPS is settings");

    // Setting Dialog Message
    alertDialog.setMessage("GPS is not enabled. Do you want to go to settings menu?");

    // On pressing Settings button
    alertDialog.setPositiveButton("Settings", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog,int which) {
            Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
            mContext.startActivity(intent);
        }
    });

    // on pressing cancel button
    alertDialog.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
            dialog.cancel();
        }
    });

    // Showing Alert Message
    alertDialog.show();
}

@Override
public void onLocationChanged(Location location) {
```

```
  }

  @Override
  public void onProviderDisabled(String provider) {
  }

  @Override
  public void onProviderEnabled(String provider) {
  }

  @Override
  public void onStatusChanged(String provider, int status, Bundle extras) {
  }

  @Override
  public IBinder onBind(Intent arg0) {
    return null;
  }
}
```

Following will be the content of **res/layout/activity_main.xml** file −

```
<?xml version = "1.0" encoding = "utf-8"?>
<LinearLayout xmlns:android = "http://schemas.android.com/apk/res/android"
  android:layout_width = "fill_parent"
  android:layout_height = "fill_parent"
  android:orientation = "vertical" >


  <Button
    android:id = "@+id/button"
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:text = "getlocation"/>

</LinearLayout>
```

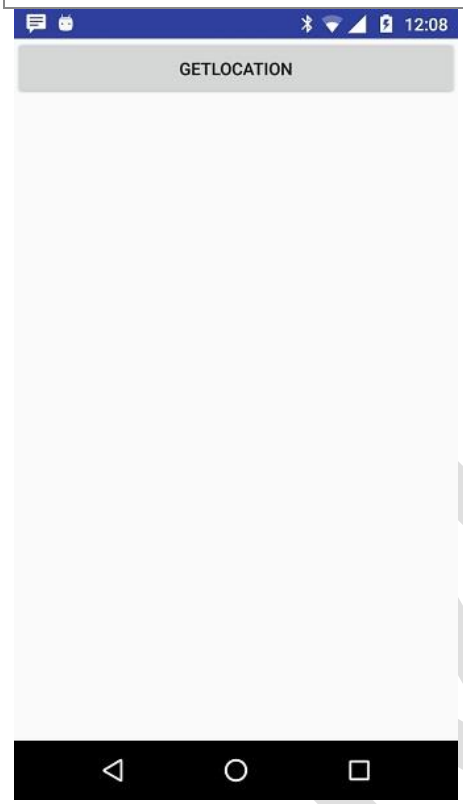
Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version = "1.0" encoding = "utf-8"?>
<resources>
  <string name = "app_name">Tutorialspoint</string>
</resources>
```

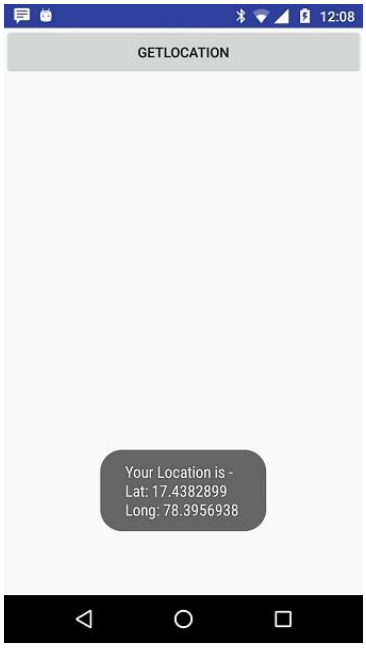
Following is the default content of **AndroidManifest.xml** −

```
<?xml version = "1.0" encoding = "utf-8"?>
<manifest xmlns:android = "http://schemas.android.com/apk/res/android"
  package = "com.example.tutorialspoint7.myapplication">
  <uses-permission android:name = "android.permission.ACCESS_FINE_LOCATION" />
  <uses-permission android:name = "android.permission.INTERNET" />
  <application
    android:allowBackup = "true"
    android:icon = "@mipmap/ic_launcher"
    android:label = "@string/app_name"
```

```
    android:supportsRtl = "true"
    android:theme = "@style/AppTheme">

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

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

</manifest>
```



Now to see location select Get Location Button which will display location information as follows –



Android - Sending Email

Email is messages distributed by electronic means from one system user to one or more recipients via a network.

Before starting Email Activity, You must know Email functionality with intent, Intent is carrying data from one component to another component with-in the application or outside the application.

To send an email from your application, you don't have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc. For this purpose, we need to write an Activity that launches an email client, using an implicit Intent with the right action and data. In this example, we are going to send an email from our app by using an Intent object that launches existing email clients.

Following section explains different parts of our Intent object required to send an email.

Intent Object - Action to send Email

You will use **ACTION_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION_SEND action.
Intent emailIntent = new Intent(Intent.ACTION_SEND);

Intent Object - Data/Type to send Email

To send an email you need to specify **mailto:** as URI using setData() method and data type will be to **text/plain** using setType() method as follows −
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.setType("text/plain");

Intent Object - Extra to send Email

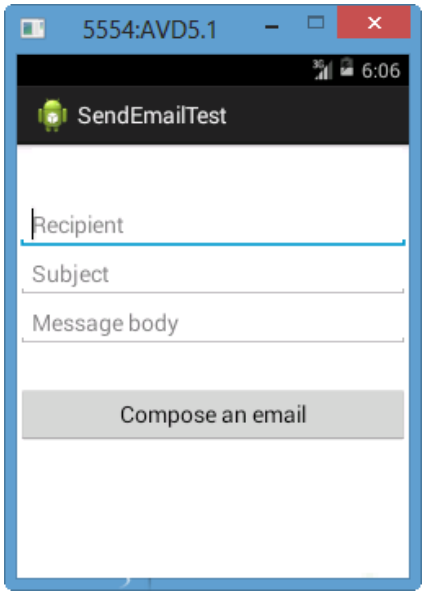
Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client. You can use following extra fields in your email −

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.
6	**EXTRA_TEXT** A constant CharSequence that is associated with the Intent, used with ACTION_SEND to supply the literal data to be sent.
7	**EXTRA_TITLE** A CharSequence dialog title to provide to the user when used with a ACTION_CHOOSER.

Here is an example showing you how to assign extra data to your intent −

```
emailIntent.putExtra(Intent.EXTRA_EMAIL  , new String[]{"Recipient"});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "subject");
emailIntent.putExtra(Intent.EXTRA_TEXT   , "Message Body");
```

The out-put of above code is as below shown an image



Email Example

Example

Following example shows you in practical how to use Intent object to launch Email client to send an Email to the given recipients.
To Email experiment with this example, you will need actual Mobile device equipped with latest Android OS, otherwise you might get struggle with emulator which may not work properly. Second you will need to have an Email client like GMail(By default every android version having Gmail client App) or K9mail installed on your device.

Step	Description
1	You will use Android studio to create an Android application and name it as *Tutorialspoint* under a package *com.example.tutorialspoint*.
2	Modify *src/MainActivity.java* file and add required code to take care of sending email.
3	Modify layout XML file *res/layout/activity_main.xml* add any GUI component if required. I'm adding a simple button to launch Email Client.
4	Modify *res/values/strings.xml* to define required constant values
5	Modify *AndroidManifest.xml* as shown below
6	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.Tutorialspoint/MainActivity.java**.

```
package com.example.tutorialspoint;

import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
```

```
import android.content.Intent;
import android.util.Log;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button startBtn = (Button) findViewById(R.id.sendEmail);
    startBtn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        sendEmail();
      }
    });
  }

  protected void sendEmail() {
    Log.i("Send email", "");
    String[] TO = {""};
    String[] CC = {""};
    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..."));
      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();
    }
  }
}
```

Following will be the content of **res/layout/activity_main.xml** file –
Here abc indicates about tutorialspoint logo

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical" >
```

```
  <TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sending Mail Example"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp" />

  <TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials point "
    android:textColor="#ff87ff09"
    android:textSize="30dp"
    android:layout_above="@+id/imageButton"
    android:layout_alignRight="@+id/imageButton"
    android:layout_alignEnd="@+id/imageButton" />

  <ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageButton"
    android:src="@drawable/abc"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

  <Button
    android:id="@+id/sendEmail"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/compose_email"/>

</LinearLayout>
```

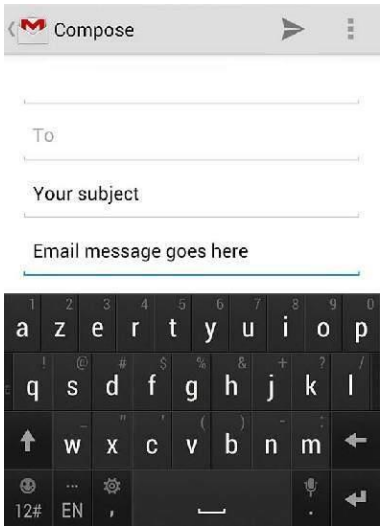
Following will be the content of **res/values/strings.xml** to define two new constants −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Tutorialspoint</string>
  <string name="compose_email">Compose Email</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

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

  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
```

```
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

    <activity
      android:name="com.example.tutorialspoint.MainActivity"
      android:label="@string/app_name" >

      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>

    </activity>

  </application>
</manifest>
```



****