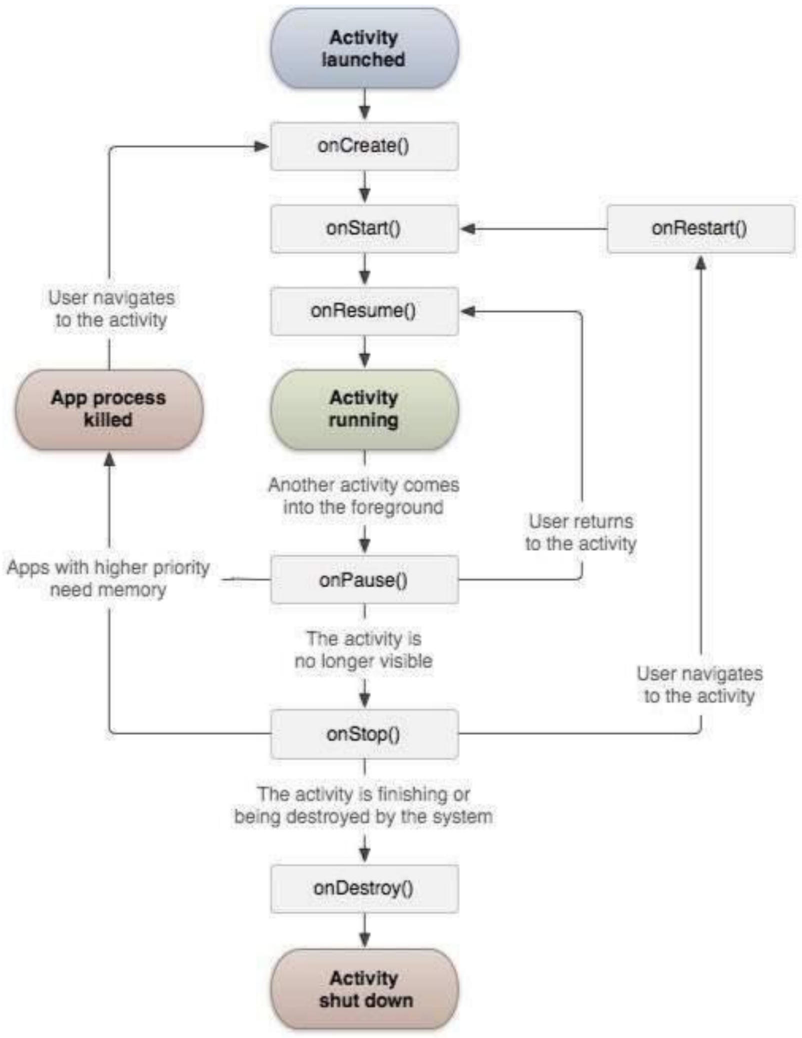
NOTES OF

ANDROID

INDEX

ACTIVITIES AND INTENT

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram: (*image courtesy : android.com*)


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

USER INTERFACE IN ANDROID

Views:-

The basic building block for user interface is a View object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

ViewsGroup:-

The ViewGroup is a subclass of View and provides invisible container that hold other Views or other ViewGroups and define their layout properties.

At third level we have different layouts which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using View/ViewGroup objects or you can declare your layout using simple XML file main_layout.xml which is located in the res/layout folder of your project.

ACTION BAR:-

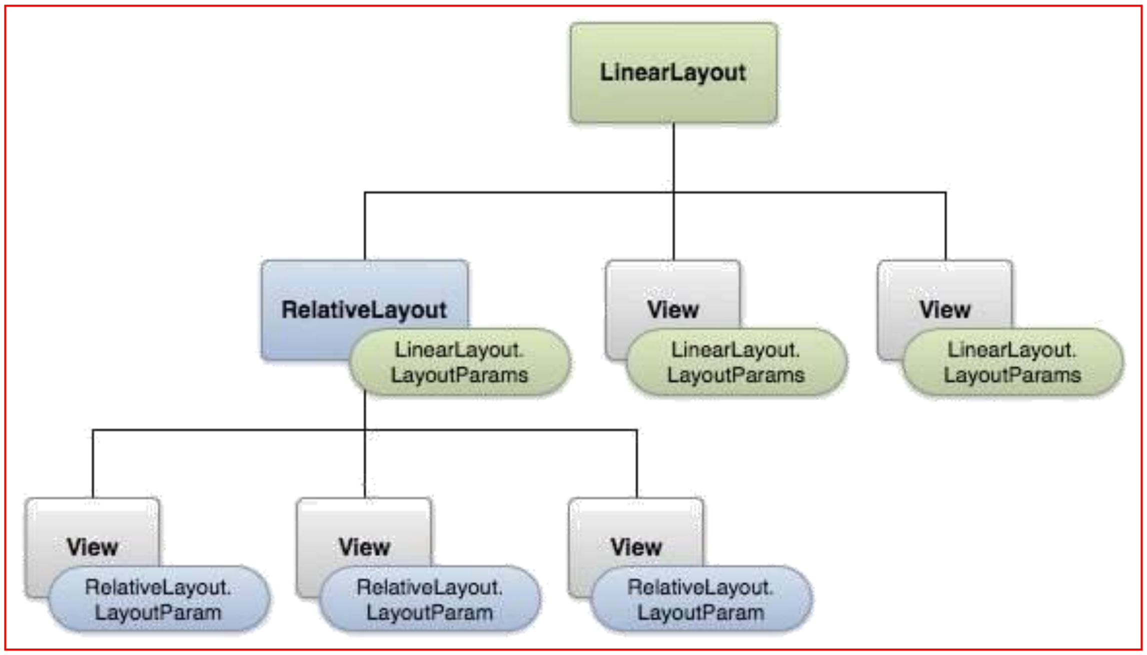
The action bar is an important design element, usually at the top of each
screen in an app, that provides a consistent familiar look between Android
apps. It is used to provide better user interaction and experience by

supporting easy navigation through tabs and drop-down lists. It also

provides a space for the app or activity's identity, thus enabling the action bar was introduced in Android 3.0, although support for older versions can be achieved by using the Android Support Library. Before its release, the Options Menu was usually used to provide the actions and functionality that are now put on the action bar. The action bar is included by default in all activities for apps with a `minSdkVersion` of 11. You can disable it and opt to only use the options menu, but for better user experiences it's better to use the action bar as it is visible to the user, while the options menu needs the user to request it and the user might not be aware of its existence in the first place

LAYOUT IN ANDROID:-

Android UI Layouts. Android Layout is used to define the user interface which holds the UI controls or widgets that will appear on the screen of an android application or activity. ... A View is defined as the user interface which is used to create an interactive UI components such as TextView, EditText, Radio Button, etc.


LinearLayout
RelativeLayout
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
LinearLayout. LayoutParams
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam
View
RelativeLayout. LayoutParam

DISPLAY ORIENTATION:-

The screen orientation attribute is provided by the activity element in the Android Manifest.Xml file. The orientations provided by the activity are Portrait, Landscape, Sensor, Unspecified and so on. To perform a screen orientation activity you define the properties in the Android Manifest.Xml file.

Example

```
<activity
        android:name="com.screenorientation.MainActivity"
        android:label="@string/app_name"
        android:screenOrientation="portrait">
```

Portrait: In a portrait mode the screen will be taller not wider.

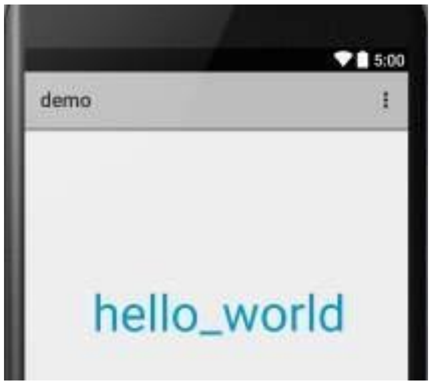
LandScape: In a landscape mode the screen will be wider not taller.

Value	Description
unspecified	It is the default value. In such case, system chooses the orientation.
portrait	taller not wider
landscape	wider not taller
sensor	Orientation is determined by the device orientation sensor.

BASIC VIEWS

TEXTVIEW:-

A **TextView** displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.



ANDROID BUTTON:-

Android Button represents a push-button. The android.widget.Button is subclass of TextView class and CompoundButton is the subclass of Button class.

There are different types of buttons in android such as RadioButton, ToggleButton, CompoundButton etc.

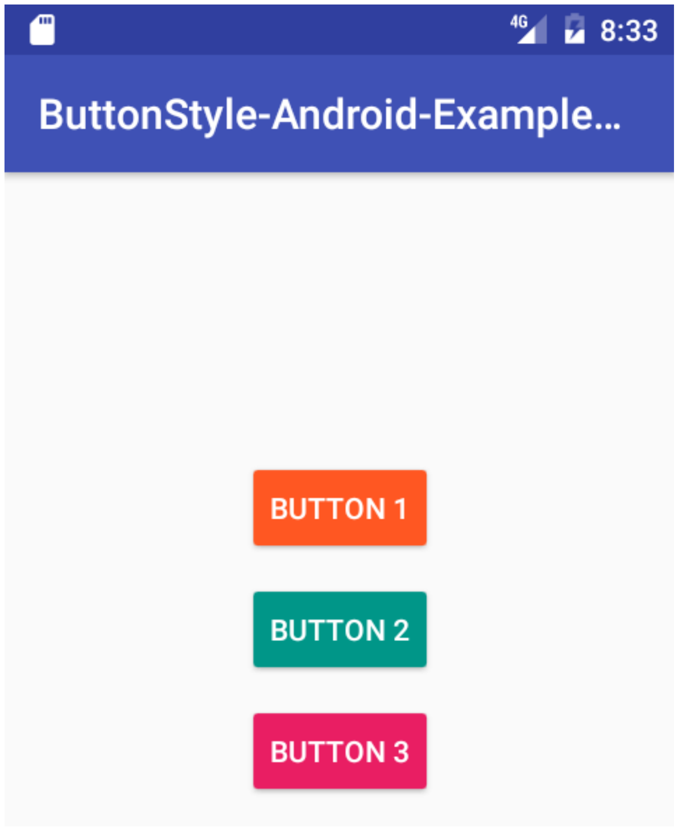
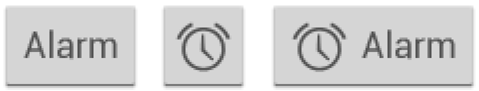


IMAGE BUTTON:-

In android, **Image Button** is a user interface control that is used to display a button with an image and to perform an action when a user clicks or taps on it.

By default, the ImageButton looks same as normal button and it performs an action when a user clicks or touches it, but the only difference is we will add a custom image to the button instead of text.

Following is the pictorial representation of using **Image Buttons** in android applications.



In android, we have different types of buttons available to use based on our requirements, those are Button, ImageButton, ToggleButton, and RadioButton.

In android, we can add an image to the button by using **<ImageButton>** attribute **android:src** in XML layout file or by using the **setImageResource()** method.

In android, we can create ImageButton control in two ways either in the XML layout file or create it in the Activity file programmatically.

EDIT TEXT:-

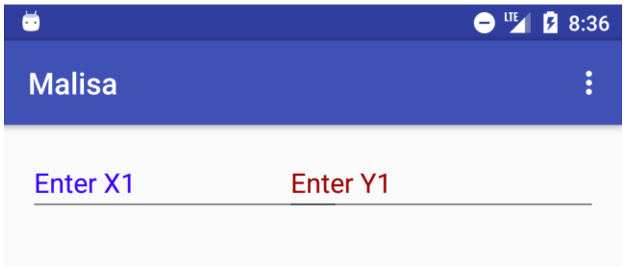
A user interface element for entering and modifying text. When you define an edit text widget, you must specify the R.styleable.TextView_inputType attribute. For example, for plain text input set inputType to "text":

```
<EditText
   android:id="@+id/plain_text_input"
   android:layout_height="wrap_content"
   android:layout_width="match_parent"
   android:inputType="text"/>
```

Choosing the input type configures the keyboard type that is shown, acceptable characters, and appearance of the edit text. For example, if you want to accept a secret number, like a unique pin or serial number, you can set inputType to "numericPassword". An inputType of "numericPassword" results in an edit text that accepts numbers only, shows a numeric keyboard when focused, and masks the text that is entered for privacy.

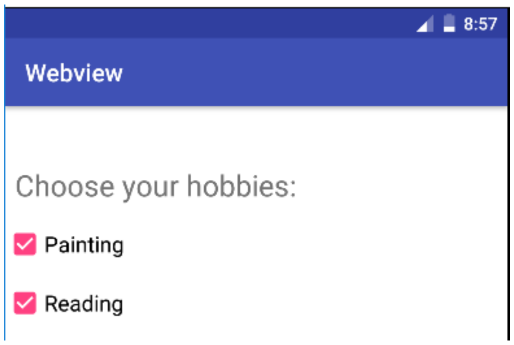
See the Text Fields guide for examples of other R.styleable.TextView_inputType settings.

This widget does not support auto-sizing text.



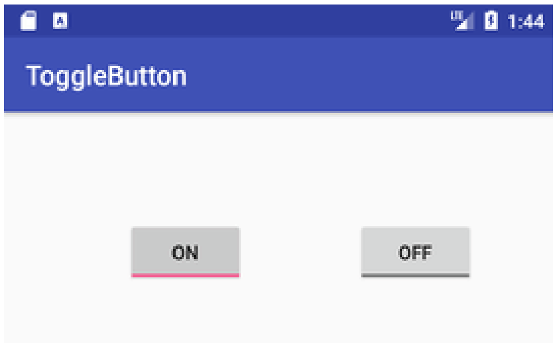
CHECKBOX:-

CheckBox belongs to android.widget.CheckBox class. Android CheckBox class is the subclass of CompoundButton class. It is generally used in a place where user can select one or more than choices from a given list of choices. For example, selecting hobbies.



TOGGLE-BUTTON:-

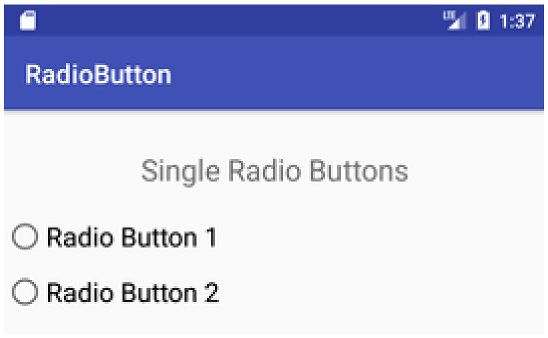
Android Toggle Button with Examples. In android, Toggle Button is a user interface control that is used to display ON (Checked) or OFF (Unchecked) states as a button with a light indicator.
The ToggleButton is useful for the users to change the settings between two states either ON or OFF.



RADIO BUTTON:-

RadioButton is a two states button which is either checked or unchecked. If a single radio button is unchecked, we can click it to make checked radio button. Once a radio button is checked, it cannot be marked as unchecked by user.

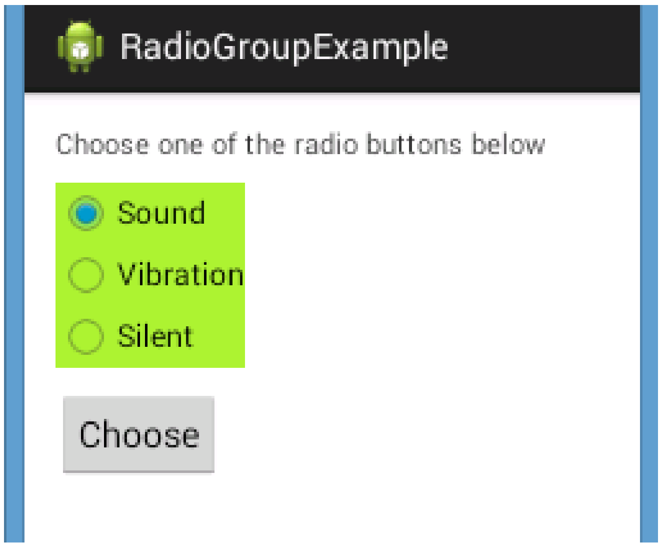
RadioButton is generally used with *RadioGroup*. RadioGroup contains several radio buttons, marking one radio button as checked makes all other radio buttons as unchecked.



RADIO GROUP:-

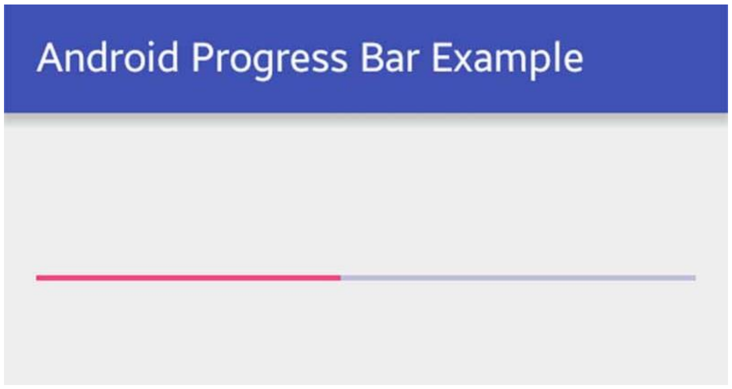
This class is used to create a multiple-exclusion scope for a set of radio buttons. Checking one radio button that belongs to a radio group unchecks any previously checked radio button within the same group.

Intially, all of the radio buttons are unchecked. While it is not possible to uncheck a particular radio button, the radio group can be cleared to remove the checked state.



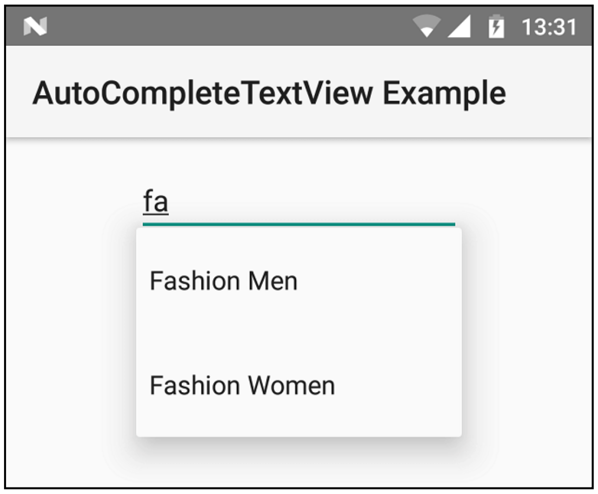
PROGRESS BAR:-

Android ProgressBar is a graphical view indicator that shows some progress. Android progress bar displays a bar representing the completing of the task. Progress bar in android is useful since it gives the user an idea of time to finish its task.



AUTOCOMPLETE TEXTVIEWS:-

Autocomplete TextView in Android. AutocompleteTextView is an editable text view that shows completion suggestions automatically while the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box with.



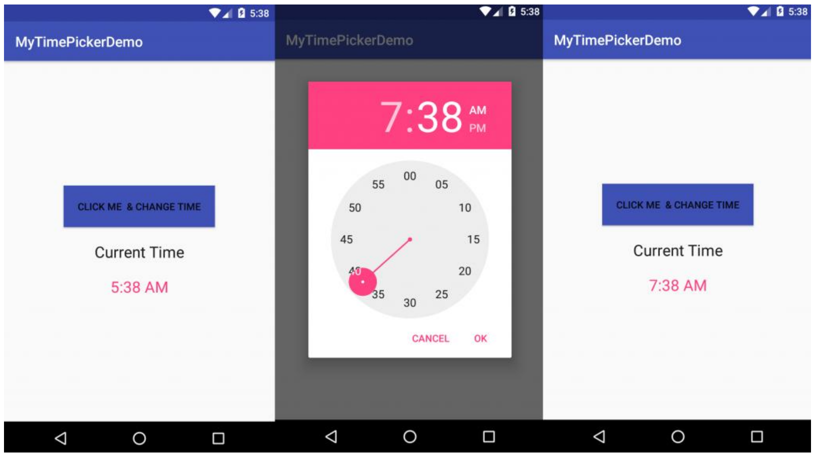
ADVANCED VIEWS

TIMEPICKER VIEW:-

Time Picker in Android is a widget that lets users to select the time of the day in either AM/PM or 24 hours mode.

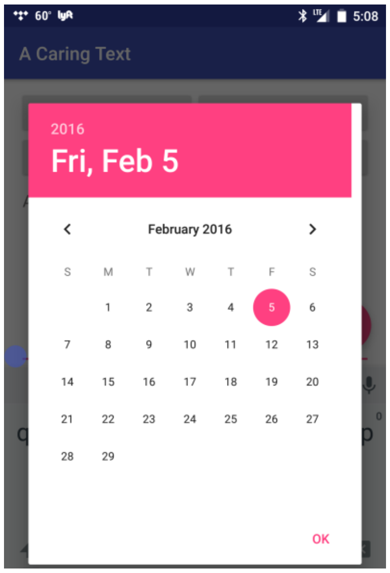
The displayed time in the widget consists of hours, minutes, and the clock format. And, in order to show add this widget in your mobile app, you'll have to use a TimePickerDialog class.

In today's Android app tutorial, we'll see how to add time picker in Android app to allow users pick time by hour, minute, and clock format.



DATEPICKER VIEW:-

In Android, DatePicker is a widget used to select a date. It allows to select date by day, month and year in your custom UI (user interface). If we need to show this view as a dialog then we have to use a DatePickerDialog class. For selecting time Android also provides timepicker to select time.



LIST VIEW:-

Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

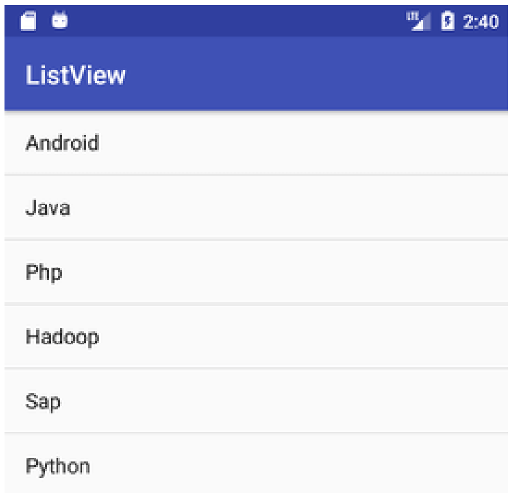
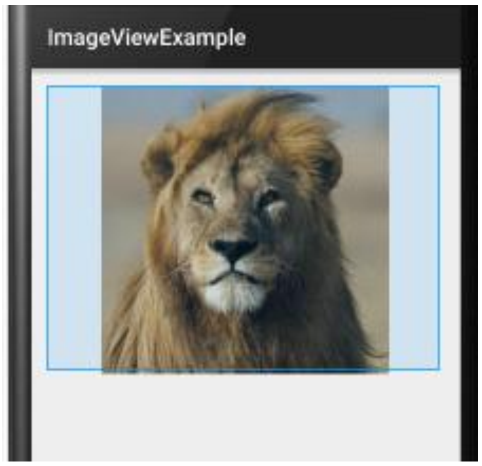


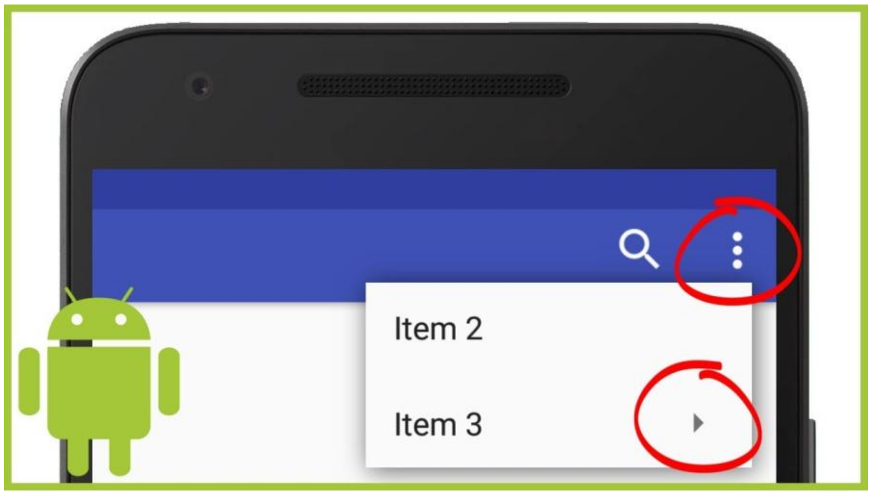
IMAGE VIEW:-

In Android, ImageView class is used to display an image file in application. Image file is easy to use but hard to master in Android, because of the various screen sizes in Android devices. An android is enriched with some of the best UI design widgets that allows us to build good looking and attractive UI based application.



MENU:-

Defining a Menu in XML. For all menu types, Android provides a standard XML format to define menu items. ... You can then inflate the menu resource (load it as a Menu object) in your activity or fragment. Using a menu resource is a good practice for a few reasons: It's easier to visualize the menu structure in XML.



DISPLAYING PICTURES & MENU WITH VIEWS

IMAGE VIEW:-

Android ImageView is used to display an image file. Android also has an Image button. As the name suggests,
the ImangeButton component is a button with an image on. The ImageButton is represented by the Android
class android.widget.ImageButton**.** In this tutorial we're going to implement both android ImageView and ImageButton in our application.

GALLERY VIEW:-

In Android, Gallery is a view used to show items in a center locked, horizontal scrolling list and user will select a view and then user selected view will be shown in the center of the Horizontal list. The items in Gallery are added using Adapter just like in ListView or GridView.

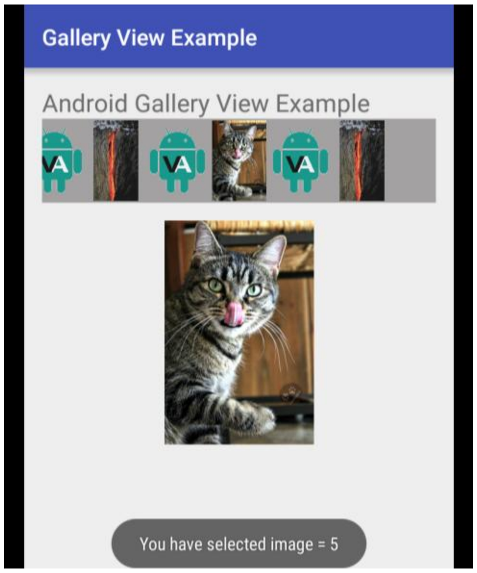
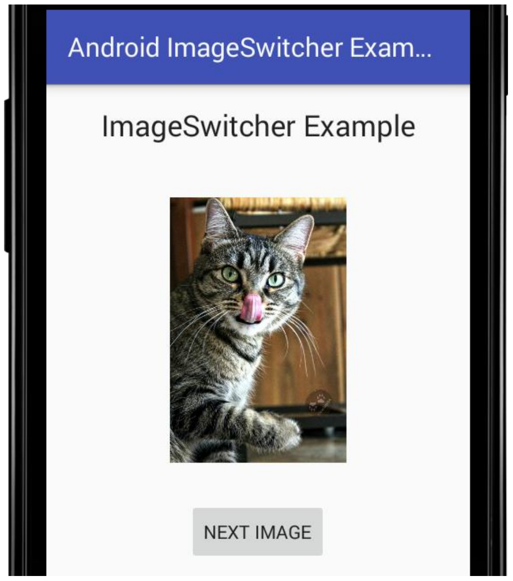


IMAGE SWITCHER:-

Sometimes you don't want an image to appear abruptly on the screen, rather you want to apply some kind of animation to the image when it transitions from one image to another. This is supported by android in the form of ImageSwitcher.

An image switcher allows you to add some transitions on the images through the way they appear on screen. In order to use image Switcher, you need to define its XML component first. Its syntax is given below −

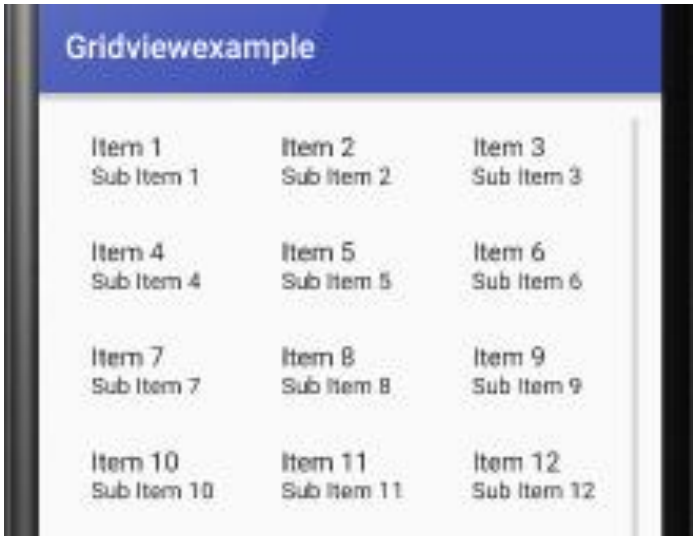
```
<ImageSwitcher
  android:id="@+id/imageSwitcher1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_centerHorizontal="true"
  android:layout_centerVertical="true" >
</ImageSwitcher>
```



GRID VIEW:-

In android GridView is a view group that display items in two dimensional scrolling grid (rows and columns), the grid items are not necessarily predetermined but they are automatically inserted to the layout using a ListAdapter. Users can then select any grid item by clicking on it. GridView is default scrollable so we don't need to use ScrollView or anything else with GridView.

GridView is widely used in android applications. An example of GridView is your default Gallery, where you have number of images displayed using grid.
Adapter Is Used To Fill Data In Gridview: To fill the data in a GridView we simply use adapter and grid items are automatically inserted to a GridView using an Adapter which pulls the content from a source such as an arraylist, array or database. You can read full Adapter tutorial here.

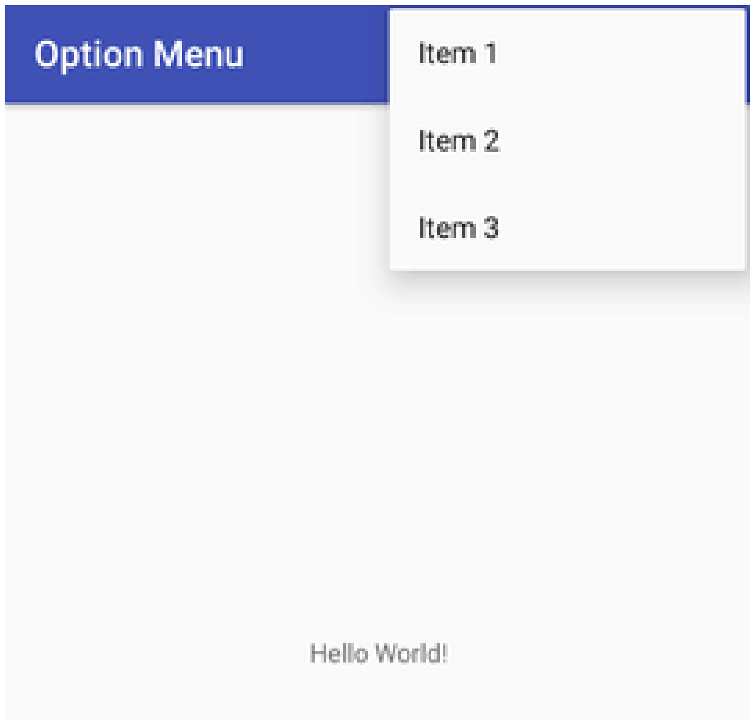


OPTIONS MENU:-

Android Option Menus are the primary menus of android. They can be used for settings, search, delete item etc.

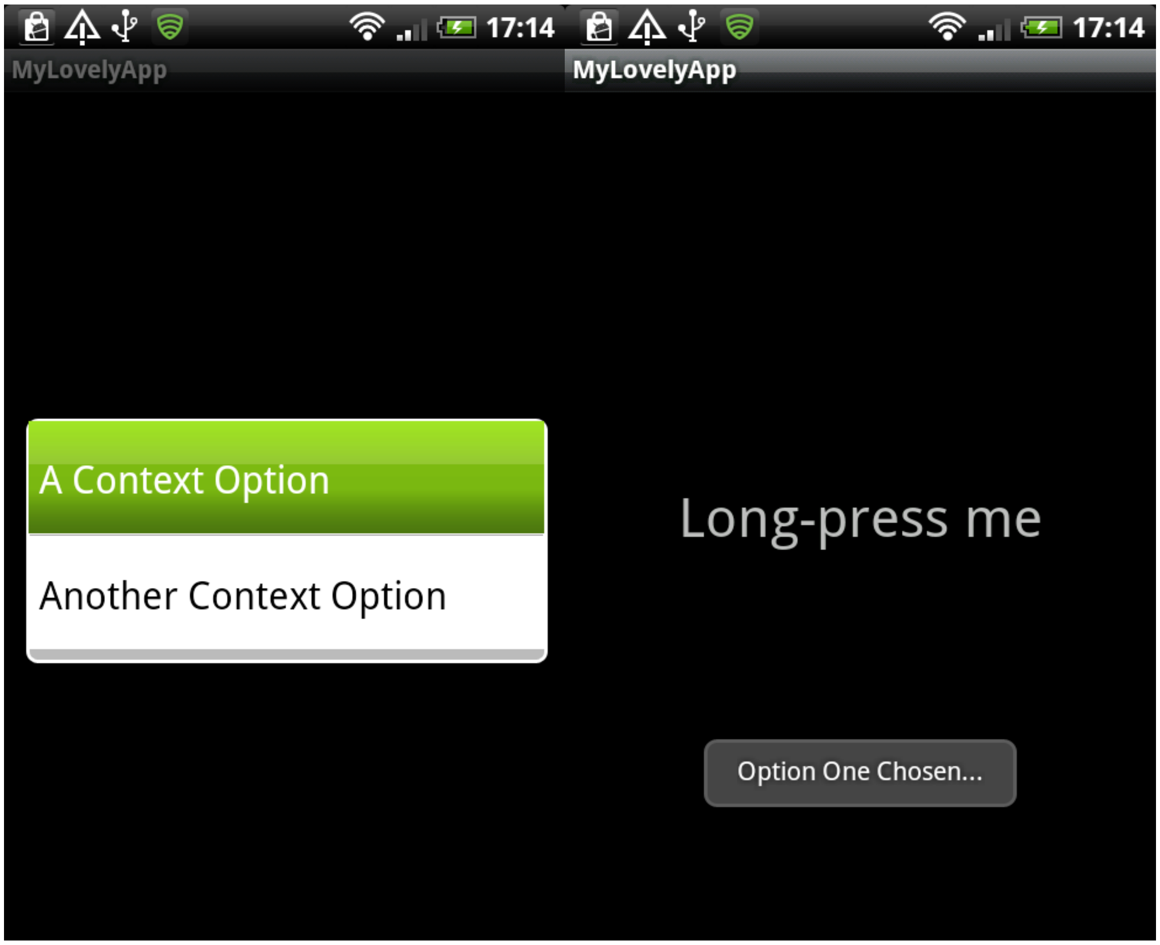
Here, we are going to see two examples of option menus. First, the simple option menus and second, options menus with images.

Here, we are inflating the menu by calling the inflate() method of MenuInflater class. To perform event handling on menu items, you need to override onOptionsItemSelected() method of Activity class.



CONTEXT MENU:-

The Android platform provides a few standard menus you can use in your apps. The context menu appears when users long-press user interface items, pressing the item and holding it until the menu appears. Android users are typically accustomed to this type of interaction, as it is standard for system functions such as altering home screen icons. The Android developer guide compares the context menu to right-clicking on a computer. Implementing the context menu is straightforward, and is a key ingredient in many applications.



SMS, PHONE

SEND-SMS:-

In this final section, I'll show you how to send SMS messages directly from your app, using Android's SmsManager class.

Text messaging can be a handy way of encouraging users to invite friends and family to sign up for your application, for example you could generate bonuses or discount codes that can then be shared over SMS. Messaging is also a convenient way of sharing information that the user has discovered in your app, or for sending notifications, for example if you've developed a calendar app then your users might appreciate the ability to send event reminders over SMS.

I'm going to implement this functionality in a separate SMSActivity, so let's start by creating a new layout resource file:

Select 'File > New > Layout resource file' from the Android Studio toolbar.

Name this file 'activity_sms.xml.'

Open your new layout file, and then add the following:

<u>XML_FILE:-</u>

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

<LinearLayout

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

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

android:id="@+id/activity_main"

android:orientation="vertical"

android:layout_width="match_parent"

android:layout_height="match_parent"

tools:context="com.jessicathornsby.telephonesms.SMSActivity
">



<EditText

android:id="@+id/phoneNumber"

android:layout_width="match_parent"

android:layout_height="wrap_content"

android:hint="Enter phone number"

android:inputType="phone" />
```

```
<EditText
android:id="@+id/message"
android:hint="Message"
android:layout_width="match_parent"
android:layout_height="wrap_content"/>
<Button
android:id="@+id/sendSMS"
android:text="Send SMS"
android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
</LinearLayout>
```

This is a simple UI where the user can send SMS messages by entering any valid mobile number, some message text, and then giving the ‘Send SMS’ button a tap.

Our ‘activity_sms.xml’ needs a corresponding Java class, so select ‘File > New > Java class’ from the Android Studio toolbar, and name this class ‘SMSActivity.’

JAVA

```
import android.support.v4.app.ActivityCompat;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.os.Build;

import android.widget.Button;

import android.support.v4.content.ContextCompat;

import android.widget.EditText;

import android.util.Log;

import android.Manifest;

import android.content.pm.PackageManager;

import android.telephony.SmsManager;

import android.text.TextUtils;

import android.widget.Toast;

import android.view.View;


public class SMSActivity extends AppCompatActivity {


  private static final int PERMISSION_REQUEST_CODE = 1;

  private Button sendSMS;
```

```
 @Override

  protected void onCreate(Bundle savedInstanceState) {

     super.onCreate(savedInstanceState);

     setContentView(R.layout.activity_sms);



     final EditText phoneNumber = (EditText)
findViewById(R.id.phoneNumber);

     if (Build.VERSION.SDK_INT >= 23) {

      if (checkPermission()) {

          Log.e("permission", "Permission already granted.");

       } else {

           requestPermission();

      }

    }



     final EditText smsText = (EditText)
findViewById(R.id.message);

     sendSMS = (Button) findViewById(R.id.sendSMS);

     sendSMS.setOnClickListener(new View.OnClickListener() {
        @Override

        public void onClick(View view) {

            String sms = smsText.getText().toString();

            String phoneNum = phoneNumber.getText().toString();

            if(!TextUtils.isEmpty(sms) &&
!TextUtils.isEmpty(phoneNum)) {

              if(checkPermission()) {



//Get the default SmsManager//



                 SmsManager smsManager =
SmsManager.getDefault();



//Send the SMS//



                 smsManager.sendTextMessage(phoneNum, null,
sms, null, null);

              }else {

                 Toast.makeText(SMSActivity.this, "Permission
denied", Toast.LENGTH_SHORT).show();

              }

           }
        }

      });

  }



  private boolean checkPermission() {

      int result =
ContextCompat.checkSelfPermission(SMSActivity.this,
Manifest.permission.SEND_SMS);

      if (result == PackageManager.PERMISSION_GRANTED) {

        return true;

      } else {

         return false;

      }

  }



  private void requestPermission() {

      ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.SEND_SMS},
PERMISSION_REQUEST_CODE);



  }
  @Override

  public void onRequestPermissionsResult(int requestCode,
String permissions[], int[] grantResults) {

     switch (requestCode) {

        case PERMISSION_REQUEST_CODE:

if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {



              Toast.makeText(SMSActivity.this,

                    "Permission accepted",
Toast.LENGTH_LONG).show();



           } else {

               Toast.makeText(SMSActivity.this,

                    "Permission denied",
Toast.LENGTH_LONG).show();

               Button sendSMS = (Button)
findViewById(R.id.sendSMS);

               sendSMS.setEnabled(false);



           }

           break;
```

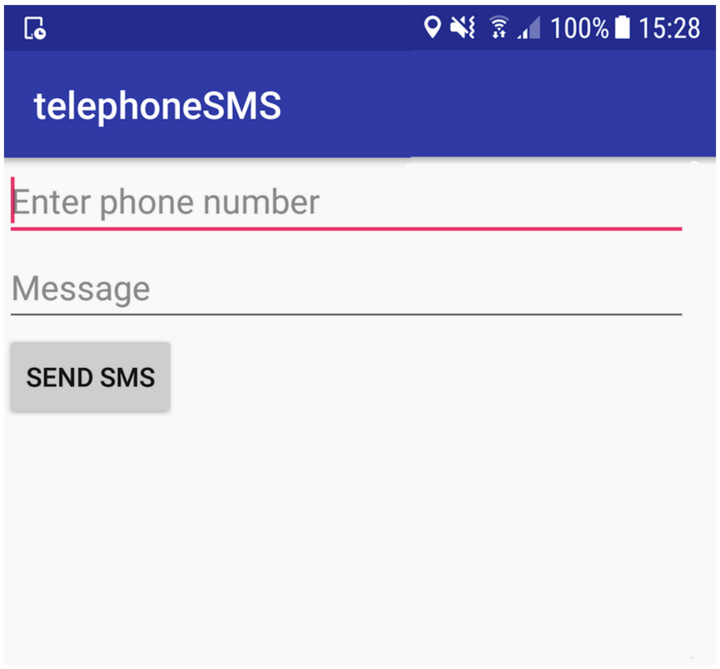
```
    }
  }
}
```

Next, open your project’s Manifest and add the SEND_SMS permission. We also need to declare the SMSActivity component:

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

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

  package="com.jessicathornsby.telephonesms">

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

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

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

  <uses-feature android:name="android.hardware.telephony"
android:required="true" />

  <application

    android:allowBackup="true"

    android:icon="@mipmap/ic_launcher"

    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"

    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>

    <activity android:name=".SMSActivity">

    </activity>

  </application>



</manifest>
```

The last task is ensuring the user can reach SMSActivity from the app's MainActivity. Open your project's activity_main.xml file and add a 'launchSMS' button:

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

<LinearLayout


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

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

  android:id="@+id/activity_main"

  android:orientation="vertical"

  android:layout_width="match_parent"

  android:layout_height="match_parent"


tools:context="com.jessicathornsby.telephonesms.MainActivit
y">



  <EditText

     android:id="@+id/phoneNumber"

     android:layout_width="match_parent"

     android:layout_height="wrap_content"

     android:hint="Enter phone number"

     android:inputType="phone" />
  <Button
     android:id="@+id/call"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:onClick="call"
     android:text="Call" />

  <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:id="@+id/contacts"
     android:text="Load contacts" />



//Add the following//


  <Button
     android:id="@+id/launchSMS"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:layout_gravity="center_horizontal"

     android:onClick="launchSMS"

     android:text="Launch SMS Activity" />


  <ListView

     android:id="@+id/listview"

     android:layout_width="match_parent"

     android:layout_height="match_parent" />

</LinearLayout>
```


100% 15:28
telephoneSMS
Enter phone number
Message
SEND SMS