UNIT – 2

Working with the User Interface using views:

Understanding the Components of a Screen-Adapting to Display Orientation-Managing Changes to Screen Orientation- Utilizing the Action Bar-Creating the User Interface Programmatically Listening for UI Notification.

Understanding the Components of a Screen:

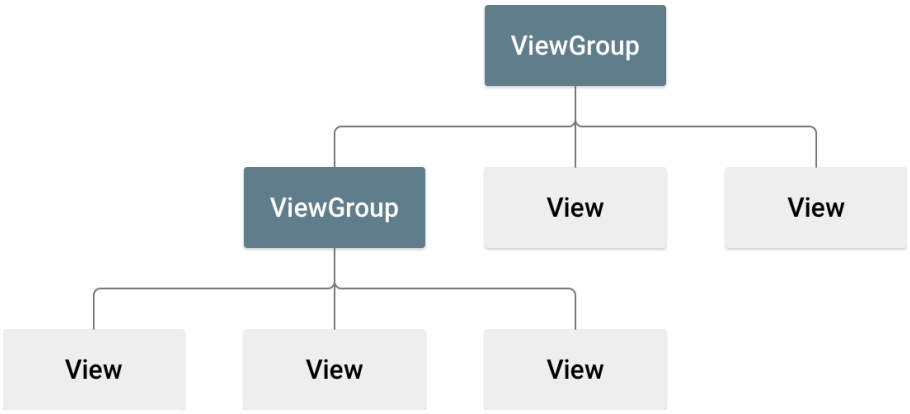
- The basic unit of an Android application is an *activity*, which displays the UI of your application.
- you define your UI using an XML file (for example, the activity_main.xml file located in the res/layout folder of your project)
- During runtime, you load the XML UI in the onCreate() method handler in your Activity class, using the setContentView() method of the Activity class:

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

Views and View groups:

- An activity contains *views* and *ViewGroups*. A View usually draws something the user can see and interact with. A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and text boxes.
- A view derives from the base class **android.view.View.**
- In ViewGroup, one or more Views can be grouped together. A ViewGroup provides the layout, in which you can set the order of the appearance and sequence of the Views. Some examples of ViewGroups are LinearLayout and FrameLayout.

- It is derived from the base class **android.view.ViewGroup**.



Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the view.
2	**android:layout_width** This is the width of the layout.
3	**android:layout_height** This is the height of the layout
4	**android:layout_marginTop** This is the extra space on the top side of the layout.
5	**android:layout_marginBottom** This is the extra space on the bottom side of the layout.
6	**android:layout_marginLeft** This is the extra space on the left side of the layout.
7	**android:layout_marginRight** This is the extra space on the right side of the layout.

8	**android:layout_gravity** This specifies how child Views are positioned from top bottom left and write from the window
9	**android:layout_weight** This specifies how much of the extra space in the layout should be allocated to the View.It distributes the available space among the child views
10	**android:layout_x** This specifies the x-coordinate of the layout.
11	**android:layout_y** This specifies the y-coordinate of the layout.
12	**android:layout_width** This is the width of the layout.
13	**android:paddingLeft** This is the left padding filled for the layout.
14	**android:paddingRight** This is the right padding filled for the layout.
15	**android:paddingTop** This is the top padding filled for the layout.
16	**android:paddingBottom** This is the bottom padding filled for the layout.

Android supports the following ViewGroups.

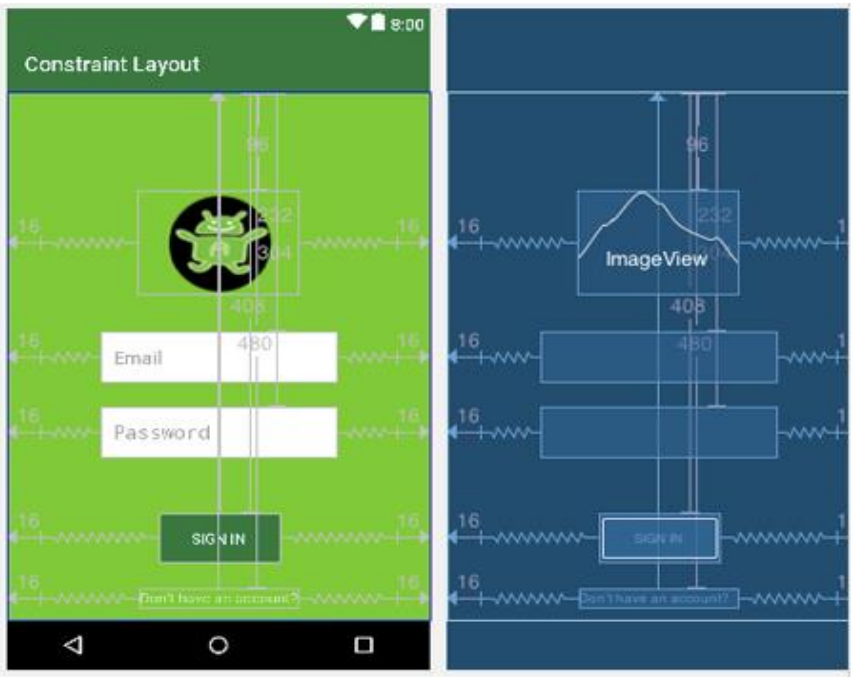
- Layout is used to arrange view and group of views visually on the screen.

1. ConstriantLayout
2. LinearLayout

3. AbsoluteLayout
4. TableLayout
5. RelativeLayout
6. FrameLayout
7. Scroll View
8. GridView
9. ListView

ConstriantLayout:

- ConstraintLayout is a ViewGroup subclass; whenever you open the android studio framework the application will be present in the constraint layout. In this we have to set the constraint in all four sides.
- This is the default layout. It's typically used to manage the positioning of individual views rather than entire groups of views.
- Ex: Designing a login screen for a mobile app using constraint layout in Android.

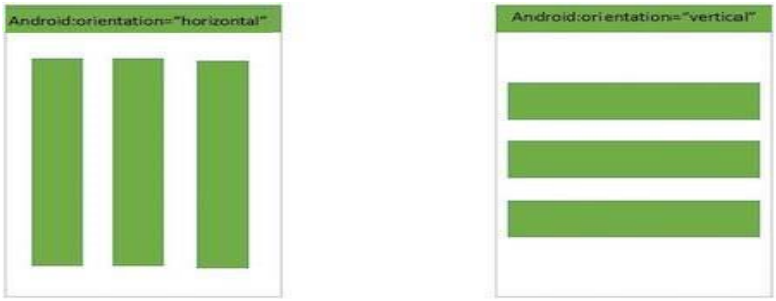


For example, you can specify that the logo is centered horizontally in the parent, the username input field is centered below the logo, the password input field is centered below the username field, and so on.
This way, no matter what device the app is running on, whether it's a small phone or a large tablet, the login screen will adjust dynamically to fit the screen size while maintaining the specified layout constraints.

LinearLayout:

- The LinearLayout arranges views in a single column or a single row. Child views can be arranged either horizontally or vertically in a single direction,

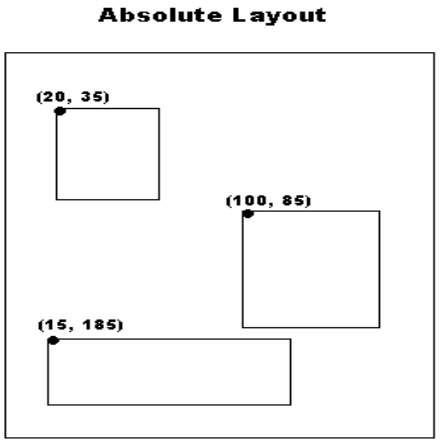
which explains the need for two different layouts—one for horizontal rows of views and one for vertical columns of views. You can specify the layout direction with the android:orientation attribute.



- The root layout is a LinearLayout with **android:orientation="horizontal"**, which means child views, will be placed side by side. Suitable for toolbars, menus, and other horizontally aligned components.
- The root layout is a Linear Layout with **android:orientation="vertical",** which means child views will be placed one below the other. Suitable for forms, lists, and other vertically aligned components.

AbsoluteLayout:

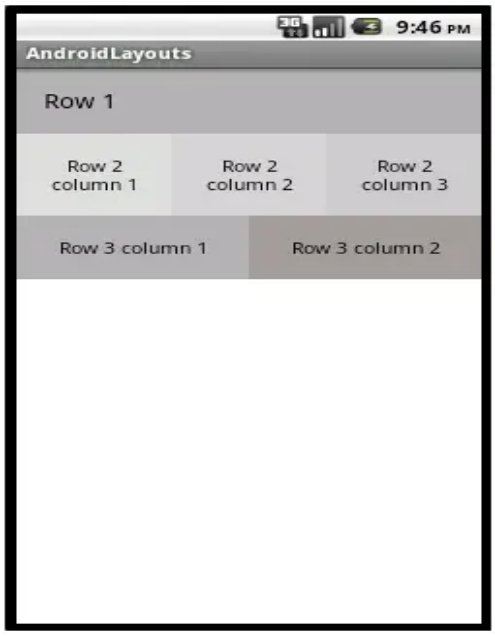
- An Absolute layout allows you to specify the exact location. i.e., X and Y coordinate of its children with respect to the origin at the top left corner of the layout.
- android:layout_x
  - This specifies the x-coordinate of the view.
- android:layout_y
  - This specifies the y-coordinate of the view.



android:layout_x="50px"
android:layout_y="361px"

TableLayout :

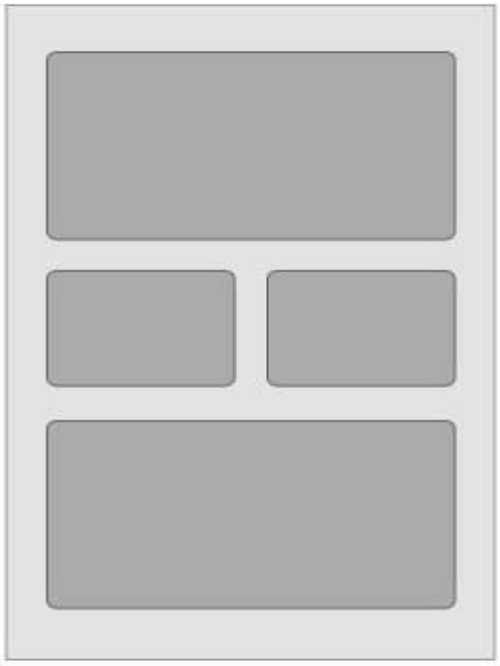
- TableLayout is a view that groups views into rows and columns. You use the <TableRow> element to designate a row in the table. Each row can contain one or more views.
- In android, TableLayout will position its children elements into rows and columns and it won't display any border lines for rows, columns or cell.



RelativeLayout:

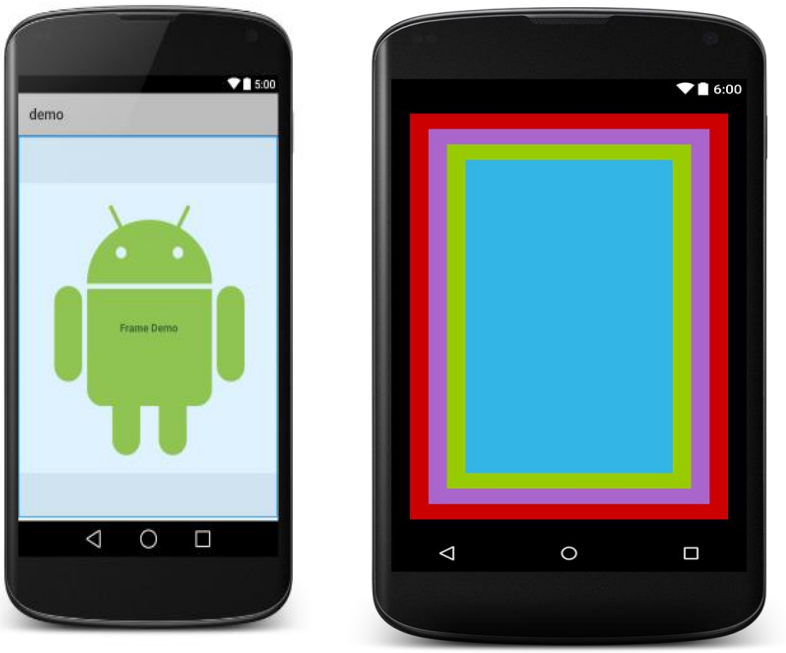
- RelativeLayout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements or relative to the parent.
- Each view embedded within the RelativeLayout has attributes that enable it to align with another view.
- These attributes are as follows:
  - **layout_alignParentTop:** Aligns the top edge of the view with the top edge of the parent.
  - **layout_alignParentStart:** Aligns the start edge of the view with the start edge of the parent.
  - **layout_alignStart:** Aligns the start edge of the view with the start edge of another specified view.
  - **layout_alignEnd:** Aligns the end edge of the view with the end edge of another specified view.
  - **layout_below:** Positions the view directly below another specified view.

- **layout_centerHorizontal:** Centers the view horizontally within the parent.



FrameLayout:

- The FrameLayout is a placeholder on screen that you can use to display a single view. Views that you add to a FrameLayout are always anchored to the top left of the layout.
- FrameLayout is a ViewGroup subclass, The FrameLayout is the most basic of the Android layouts. FrameLayouts are built to hold one view.
- You can add multiple views to a FrameLayout, but each is stacked on top of the previous one. This is when you want to animate a series of images, with only one visible at a time.
- You can, however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the android:layout_gravity attribute. android:gravity="center".



Example: Image Carousel with FrameLayout
Imagine you want to create an image carousel where multiple images are displayed one after another with a fade-in animation, while a caption is shown on top of each image. FrameLayout is perfect for this because it allows stacking views and handling animations easily.

ScrollView

- A ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display.
- The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout. Android supports vertical scroll view as default scroll view. Vertical scrollview scrolls element s vertically. Android uses Horizontal ScrollView for scrolls element horizontally.

GridView

- GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

ListView

- ListView is a view group that displays a list of scrollable items. **wrap_content**: Tells the view to size itself to the dimensions required by its content.

- **Example**: If you have a TextView with **android:layout_width="wrap_content",** the TextView will expand just enough to fit the text it contains.

match_parent (previously fill_parent):

- **Definition**: Tells the view to become as big as its parent view, filling the entire available space.
- **Example**: If you have a Button with android:layout_width="match_parent", the Button will stretch to fill the width of its parent layout.

These attributes are important for controlling the layout and appearance of views within your Android app's user interface.

Common Attributes Used in Views and ViewGroups

attribute	Description
layout_width	Specifies the width of the view or ViewGroup
layout_height	Specifies the height of the view or ViewGroup
layout_marginTop	Specifies extra space on the top side of the view or ViewGroup
layout_marginBottom	Specifies extra space on the bottom side of the view or ViewGroup
layout_marginLeft	Specifies extra space on the left side of the view or ViewGroup
layout_marginRight	Specifies extra space on the right side of the view or ViewGroup
layout_gravity	Specifies how child views are positioned
layout_weight	Specifies how much of the extra space in the layout should be allocated to the view
layout_x	Specifies the x-coordinate of the view or ViewGroup
layout_y	Specifies the y-coordinate of the view or ViewGroup

Adapting to Display Orientation :

- As with almost all smartphones, Android supports two screen orientations: portrait and landscape.
- When the screen orientation of an Android device is changed, the current activity being displayed is destroyed and re-created automatically to redraw its content in the new orientation.
- In other words, the onCreate() method of the activity is fired whenever there is a change in screen orientation.
- Portrait mode is longer in height and smaller in width, whereas landscape mode is wider but smaller in height.
- Being wider, landscape mode has more empty space on the right side of the screen.
- At the same time, some of the controls don't appear because of the smaller height.
- Thus, controls need to be laid out differently in the two screen orientations because of the difference in the height and width of the two orientations.

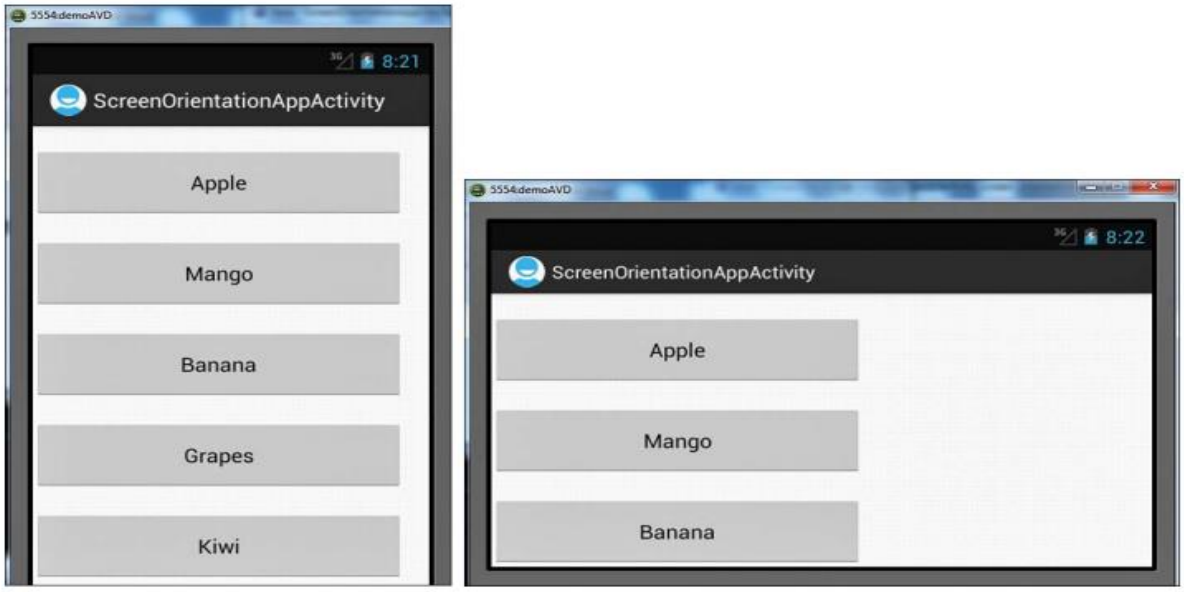
There are two ways to handle changes in screen orientation:

Anchoring Views/Controls—

1. Anchoring involves setting the position of views (or controls) relative to the four edges of the screen. When the screen orientation changes, the controls/views do not disappear but are rearranged (the views adjust their positions accordingly, maintaining a consistent layout and avoiding disappearing off-screen) relative to the four edges.
2. For anchoring controls relative to the four edges of the screen, we use a RelativeLayout container. The controls are aligned relative to the edges of the container.

Defining layout for each mode—

- This approach involves creating separate XML layout files for different screen orientations (portrait and landscape). Each layout file is tailored to suit the specific orientation, ensuring the UI is optimized for both modes.



•This vertical arrangement makes a few of the Button controls disappear when the screen is in landscape mode. To use the blank space on the right side of the screen in landscape mode, we need to define another layout file.

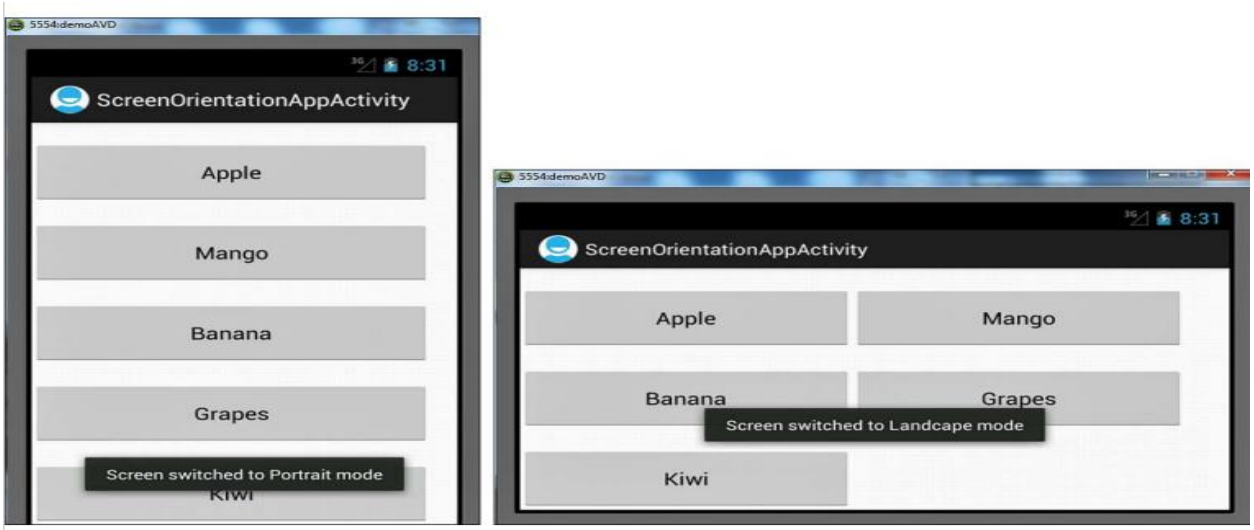


Figure 3.18. (left) Controls in `portrait` mode, and (right) all controls are visible in `landscape` mode.

Resizing and Repositioning Views:

- This approach involves dynamically adjusting the size and position of views within a single layout file to respond to changes in screen orientation. Instead of switching layouts, you modify view properties programmatically.

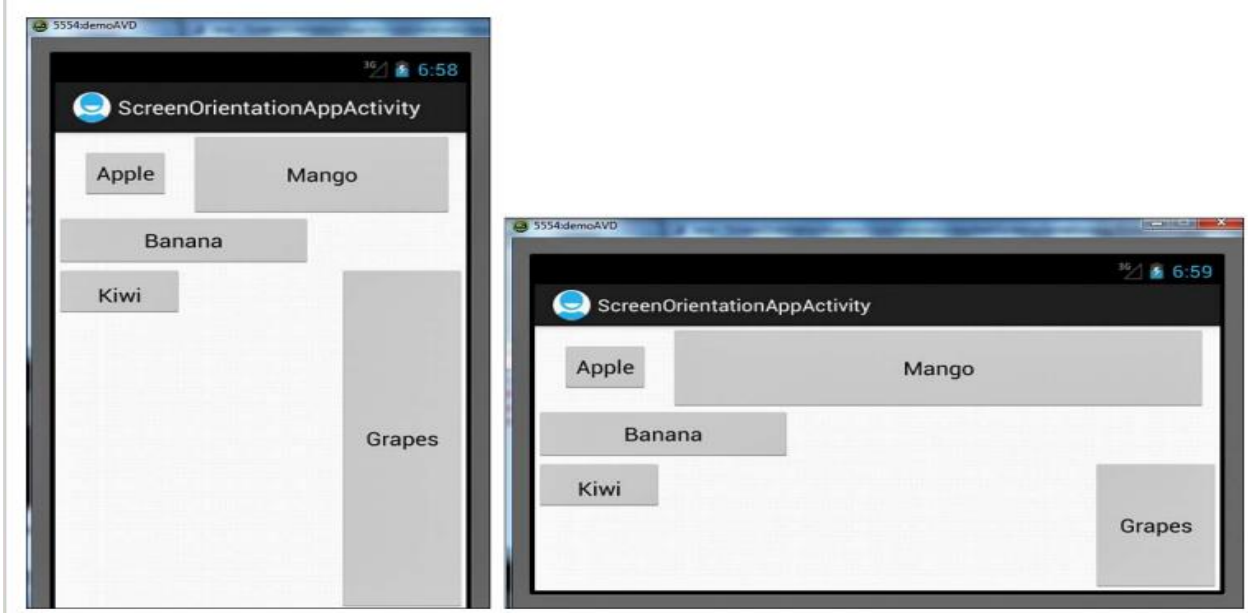


Figure 3.16. (left) Controls in `portrait` mode, and (right) the controls in `landscape` mode

- Anchoring Views Anchoring can be easily achieved by using RelativeLayout. Consider the following main.xml file, which contains five Button views embedded within the element:

CODE:

```
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">

   <!-- Button anchored to the top-left corner -->
   <Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Top Left"
      android:layout_alignParentStart="true"
      android:layout_alignParentTop="true" />

   <!-- Button anchored to the top-right corner -->
   <Button
      android:id="@+id/button2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Top Right"
      android:layout_alignParentTop="true"
      android:layout_alignParentEnd="true" />

   <!-- Button anchored to the bottom-left corner -->
   <Button
      android:id="@+id/button3"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
    android:text="Bottom Left"
    android:layout_alignParentStart="true"
    android:layout_alignParentBottom="true" />

  <!-- Button anchored to the bottom-right corner -->
  <Button
    android:id="@+id/button4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bottom Right"
    android:layout_alignParentEnd="true"
    android:layout_alignParentBottom="true" />



  <!-- Button centered in the middle of the parent -->
  <Button
    android:id="@+id/button5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Middle"
    android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

</RelativeLayout>
```

Button 1 (button1): Anchored to the top-left corner of the parent.
Button 2 (button2): Anchored to the top-right corner of the parent.
Button 3 (button3): Anchored to the bottom-left corner of the parent.
Button 4 (button4): Anchored to the bottom-right corner of the parent.
Button 5 (button5): Centered both vertically and horizontally in the parent.

Note the following attributes found in the various Button views:

➢ layout_alignParentStart—Aligns the view to the left of the parent view

- layout_alignParentEnd—Aligns the view to the right of the parent view
- layout_alignParentTop—Aligns the view to the top of the parent view
- layout_alignParentBottom—Aligns the view to the bottom of the parent view
- layout_centerVertical—Centers the view vertically within its parent view
- layout_centerHorizontal—Centers the view horizontally within its parent view

FIGURE 4-18

Managing Changes to Screen Orientation:

- What happens to an activity's state (data) when the device changes orientation Understanding Activity Behavior when Orientation Changes.
- Activity behavior Changes when the screen orientation Changes in Android,so when the device orientation changes,frist the activity will disappear for a millisecond when the onPause( ),onStop( ),onDestroy( ) methods are called.
- After a few milliseconds, the activity will be restarted when the onCreate( ), onStart( ),onResume( ) methods are called.

1. **Using Android Studio, create a new Android project and name it Orientations.**

2. Add the bolded statements in the following code to the 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" >
<EditText
 android:id="@+id/txtField1"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content" />
<EditText
# android:id="@+id/txtField2"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content" />
</LinearLayout>
```

3. Add the bolded statements in in the following code to the MainActivity.java file:

```
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends Activity {
 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 Log.d("StateInfo", "onCreate");
  }
 @Override
 public void onStart() {
```

```
Log.d("StateInfo", "onStart");
super.onStart();
  }
@Override
public void onResume() {
Log.d("StateInfo", "onResume");
super.onResume();
  }
@Override
public void onPause() {
Log.d("StateInfo", "onPause");
super.onPause();
  }
@Override
public void onStop() {
Log.d("StateInfo", "onStop");
super.onStop();
  }
@Override
public void onDestroy() {
Log.d("StateInfo", "onDestroy");
super.onDestroy();
  }
@Override
public void onRestart() {
Log.d("StateInfo", "onRestart");
super.onRestart();
  }
}
```

- Press F11 to debug

- Enter some text into the two EditText views
- Change the orientation of the Android Emulator by pressing Ctrl+F11.
- Note that the text in the first EditText view is still visible, while the second EditText view is now empty.

How It Works

- From the output shown in the logcat console, it is apparent that when the device changes orientation.

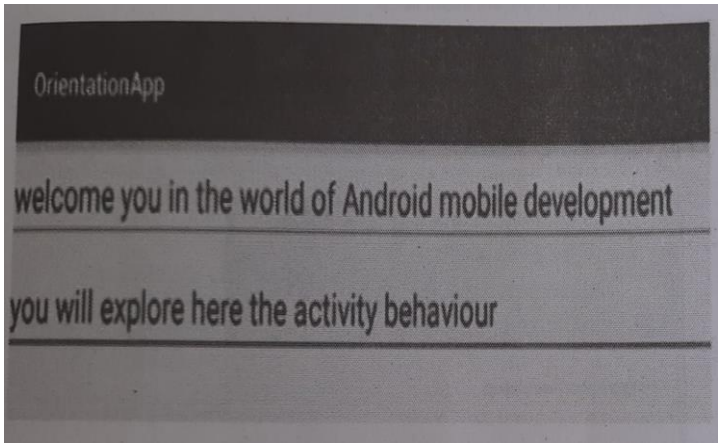
The activity is destroyed:

- 12-15 12:39:37.846: D/StateInfo(557): onPause
- 12-15 12:39:37.846: D/StateInfo(557): onStop
- 12-15 12:39:37.866: D/StateInfo(557): onDestroy

It is then re-created:

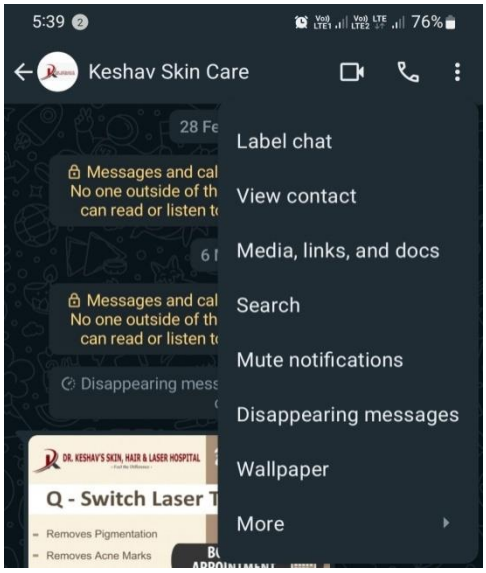
- 12-15 12:39:38.206: D/StateInfo(557): onCreate
- 12-15 12:39:38.216: D/StateInfo(557): onStart
- 12-15 12:39:38.257: D/StateInfo(557): onResume

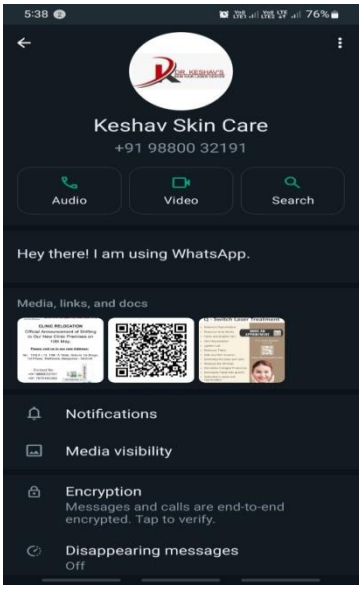
- For example, the user might change orientation while entering some text into an EditText view. When this happens, any text inside the EditText view is persisted and restored automatically when the activity is re-created. Conversely, if you do not name the EditText view using the **android:id** attribute, the activity isn't able to persist the text currently contained within it.

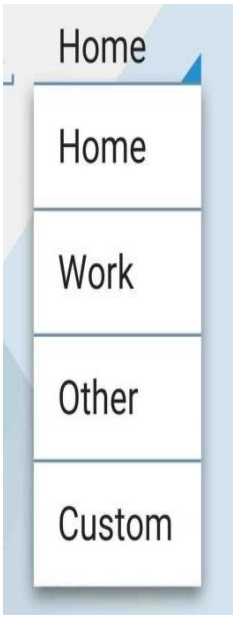


Utilizing the Action Bar:

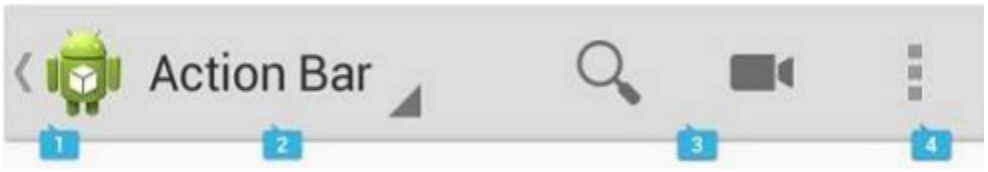
- Android ActionBar is a menu bar that runs across the top of the activity screen in android. Android ActionBar can contain menu items which become visible when the user clicks the "menu" button.
- Action bar is a combination of on-screen action items and overflow options. **Toolbar/Action Bar**: This is a bar at the top of the screen in Android apps, which contains the application logo, title, and navigation/menu items. It often hosts contextual actions or options relevant to the current screen.
- **In general an ActionBar consists of the following four components:**
  - **App Icon**: App branding logo or icon will be displayed here.
  - **View Control**: A dedicated space to display Application title. Also provides option to switch between views by adding spinner or tabbed navigation.
  - **Action Buttons**: Some important actions of the app can be added here
  - **Action Overflow**: All unimportant action will be shown as a menu.







Spinner



Spinner: A spinner typically appears as a small rectangular box with an arrow icon at the right side. When the user taps on the spinner, a dropdown list of items appears below it, allowing the user to select one.
Dropdown Menu: A dropdown menu, on the other hand, is a pop-up menu that appears below the anchor view (such as a button or text field) when the user interacts with it. It doesn't have a visible box or arrow icon associated with it.

How to set and Changing the Action Bar title:
Setting Toolbar as An ActionBar

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar); // get the reference
of Toolbar
setSupportActionBar(toolbar); // Setting/replace toolbar as the ActionBar
```

This example demonstrates How to set title for action bar in android.

```
if (getSupportActionBar() != null) {
        getSupportActionBar().setTitle("Home");
        getSupportActionBar().setSubtitle("sairam");
        getSupportActionBar().setDisplayShowHomeEnabled(true);
        getSupportActionBar().setIcon(R.drawable.ic_launcher); // Replace with
your icon
    }
    textView.setText("Title is Home");
```

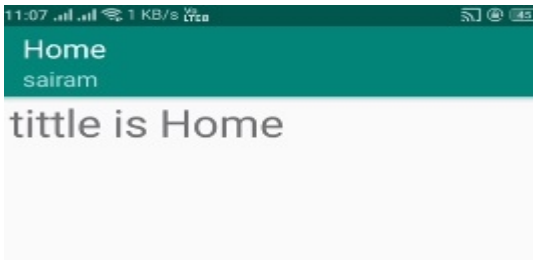
```
}
```

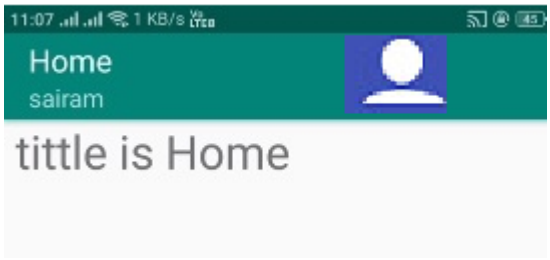
Changing the Action Bar title:

```
if (actionBar != null) {
        actionBar.setTitle(newTitle);
    }
```

Add items to the Action bar:

```
<item
 android:id="@+id/miProfile"
 android:icon="@drawable/ic_profile"
 app:showAsAction="ifRoom|withText"
 android:title="Profile">
 </item>
```





Creating the user interface programmatically:

1. In Android you can create the user interface programmatically by instantiating and configuring UI components in your java or kotlin code rather than using XML layout files.
2. So far, we have seen all the UIs which have created, using XML file but we can create the user interface programmatically. This is useful; When UI needs to be dynamically generated during runtime.

Ex: If you are creating an app for the air ticket reservation system and your app is supposed to display the seats for each way's travel, using the Buttons. In this case, you will have to dynamically generate the UI code, which is based on the air travel selected by the user.

To create a user interface programmatically in Android Studio, you typically follow these steps:

1. **Create a Layout Container:** You'll need a layout container to hold your UI Elements programmatically. This can be a LinearLayout, RelativeLayout, ConstraintLayout, etc. You can create it directly in your activity's XML layout file or Programmatically in your Java/Kotlin code.
2. **Instantiate UI Elements:** Create instances of the UI elements you want to Include in your layout. For example, TextView, Button, EditText, etc.
3. **Set Layout Parameters:** For each UI element, create layout parameters Specifying how it should be positioned and sized within the layout container.
4. **Add UI Elements to Layout:** Add the UI elements to the layout container Using the addView() method.
5. **Set Content View:** If you created the layout container programmatically, set it as the content view of your activity using the setContentView () method.

Creating the UI via Code:

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.LinearLayoutCompat;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
LinearLayoutCompat.LayoutParams params =
newLinearLayoutCompat.LayoutParams (
LinearLayoutCompat.LayoutParams.WRAP_CONTENT,
LinearLayoutCompat.LayoutParams.WRAP_CONTENT);
LinearLayout layout = new LinearLayout(this);
layout.setOrientation(LinearLayout.VERTICAL);
```

```
        TextView  tv = new TextView(this);
        tv.setText("This is a TextView");
        tv.setLayoutParams(params);
        Button btn = new Button(this);
        btn.setText("This is a Button");
        btn.setLayoutParams(params);

        layout.addView(tv);
        layout.addView(btn);
LinearLayoutCompat.LayoutParams layoutParam = new
LinearLayoutCompat.LayoutParams(
LinearLayoutCompat.LayoutParams.WRAP_CONTENT,
LinearLayoutCompat.LayoutParams.WRAP_CONTENT );
this.addContentView(layout, layoutParam);
```

Listening for UI notifications:

- Users interact with your UI at two levels:
  - Activity level
  - View level

Activity level :

- At the activity level, the Activity class exposes methods that you can override. Some common methods that you can override in your activities include the following:
  - **On Key Up():** This is called when a key was released. This is not handled by any of the views inside the activity.
  - **On Key Down():** This is called when a key was pressed. This is not handled by any of the views inside the activity.
  - **On Menu Item Selected():** This is called when any item of the menu panel is pressed by user.
  - **On Menu Opened():** This method is called when user opens the panel's menu.

View Level:

- When any user interacts with a view, the corresponding view fires event. When a user touches a button or an image button or any such view we have to service the related service so that appropriate action can be performed. For this, events need to be registered.

Chapter -2 unit-2

Using Basic Views-Using Picker Views -Using List Views to Display Long ListsUnderstanding Specialized Fragments - Using Image Views to Display Pictures -Using Menus with Views Using WebView- Saving and Loading User Preferences-Persisting Data to Files-Creating and Using Databases.

2.1 Designing Your User Interface with Views:

In an android application, an activity contains Various Views such as text view, button, radio button, and Checkbox, which are used to design UI elements. Each view represents visual elements that users can interact with.

Generally there are 3 types of Views namely:

a) **Basic views** – Commonly used views such as the Textview, EditText, and Button views.
b) **Picker views** – Views that enable users to select from a list, such as the TimePicker and DatePicker views.
c) **List views** – Views that display a long list of items, such as the ListView and the Spinner View views.

Basic Views:

Some of the basic views that can be used to design the UI components of the android applications.

1. TextView
2. EditText
3. Button
4. ImageButton
5. CheckBox
6. ToggleButton
7. RadioButton
8. RadioGroup

TextView:

This is a view that displays text. It can be used to show a single line or multi-line text. It's one of the most basic and frequently used views in Android. Android TextView Attributes:
The following are some of the commonly used attributes related to TextView control in android applications.

Code:

```
<TextView
   android:id="@+id/textView"
   android:layout_width="fill_content"
   android:layout_height="wrap_content"
   android:text="hello BCA students"/>
```

Basic Attributes in textview:

android:id - Unique identifier for the view.
android:layout_width - Width of the view (e.g., match_parent, wrap_content, or a specific dimension).

android:layout_height - Height of the view (e.g., match_parent, wrap_content, or a specific dimension).
android:text - Text to display.
android:gravity - Alignment of the text within the view (e.g., center, left, right)
android : hint - The text to display when TextView is empty.

EditText:

An EditText is a subclass of the TextView that is configured to allow the user to edit the text inside it.

```
<Edittext
        android:id =”@+id/myEdittext”
        android:layout_width=”fill_parent”
        android:layout_height=”fill_parent”
        android:hint=”Entera Number”
        android:singleLine=”true”
        android:inputType=”textPassword”/>
```

Basic Attributes:
android:id - Unique identifier for the view.
android:layout_width - Width of the view (e.g., match_parent, wrap_content, or a specific dimension).
android:layout_height - Height of the view (e.g., match_parent, wrap_content, or a specific dimension).
android:gravity - Alignment of the text within the view (e.g., center, left, right)
android:text - The text to be displayed in the EditText.
android:hint - Hint text to be displayed when the EditText is empty.
android:textColor - Color of the text.
android:textSize - Size of the text.
android:textStyle - Style of the text (e.g., bold, italic).

Button:

- In android, Button is a user interface control that is used to perform an action whenever the user clicks or tap on it.

- Generally, Buttons in android will contain a text or an icon or both and perform an action when the user touches it.
- In android we have different types of buttons available to use based on our requirements those are ImageButton, ToggleButton, RadioButton.

Code:

```
<Button
        android:id="@+id/button"
        android:layout_width="fill_content"
        android:layout_height="wrap_content"
        android:text="Click Here!"/>
```

Basic Attributes:

android:id - Unique identifier for the Button.
android:layout_width - Width of the Button (e.g., wrap_content, match_parent, or a specific dimension).
android:layout_height - Height of the Button (e.g., wrap_content, match_parent, or a specific dimension).
android:text- The text to be displayed on the Button.
android:background- Background drawable or color.
android:padding- Padding inside the Button.
android:gravity- Aligns the text inside the Button (e.g., center, left, right).

ImageButton:

- 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, but the only difference is we will add a custom image to the button instead of text.
- In android, we can add an image to the button by using attribute android:src in XML layout file or by using the setImageResource() method.

Code:

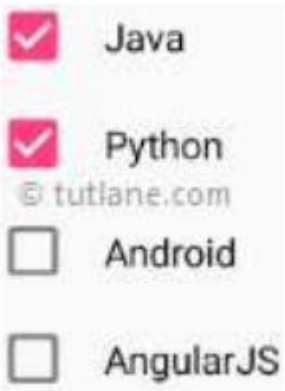
```
<ImageButton
android:id="@+id/addBtn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/add_icon" />
```

Basic Attributes:

android:id - Sets the unique identifier for the ImageButton.
android:layout_width and android:layout_height - Defines the width and height of the ImageButton.
android:src -Sets the image resource to be displayed on the ImageButton.
android:background - Uses a selectable background to give visual feedback when the button is pressed.
android:layout_centerInParent - Centers the ImageButton within its parent layout.

Check Box:

A CheckBox in Android is a type of button that has two states: checked and unchecked. It's often used to represent a binary choice, such as a yes/no option or to toggle a setting on or off. By default, the android checkbox will be in the OFF (unchecked) state.we can change the default state of checkbox by using
android: checked attribute.



Code:

```
<CheckBox
android:id="@+id/simpleCheckBox"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Simple CheckBox"/>
android:checked=”false”
```

Basic attributes:

android:id - it is used to uniquely identify the control
android:checked - It is used to specify the current state of the Checkbox
android:visibility - it is used to control the visibility of Control
android:padding - it is used to set the padding from left,right,top and bottom.

ToggleButton:

- In android, ToggleButton is a UI control that is used to display ON (checked) or OFF(unchecked) states as a button with a light indicator.
- ToggleButton allow the users to change the setting between two states like turn on/off your wifi, Bluetooth etc from your phone's setting menu. You can add a basic toggle button to your layout with the ToggleButton object.

Code:

```
<ToggleButton
android:id="@+id/toggle1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_marginLeft="100dp"
android:layout_marginTop="120dp"
android:checked="true"
android:textOff="OFF"
android:textOn="ON"/>
```

Basic Attributes:

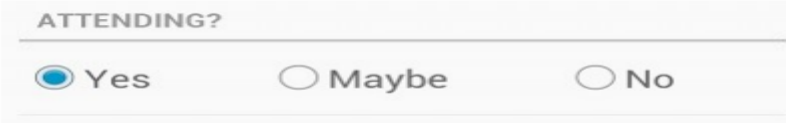
android:id - It is used to uniquely identify the control

android:checked - It is used to specify the current state of toggle button

android:text - It is used to set the text.
android:textOn - It is used to set the text when the toggle button is in the ON / Checked state.
android:textOff - It is used to set the text when the toggle button is in the OFF / Unchecked state.

Radio Button:

- In android, Radio Button is a two-state button that can be either checked or unchecked and it's the same as CheckBox control, except that it will allow only one option to select from the group of options. The user can press or click on the radio button to make it select.
- In android, CheckBox control allows users to change the state of control either Checked or Unchecked but the radio button cannot be unchecked once it is checked. Generally, we can use RadioButton controls in an android application to allow users to select only one option from the set of values.



Code:

```
<RadioButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Java"
android:checked="true"/>
```

Basic attributes:

android:id - It is used to uniquely identify the control
android:checked - It is used to specify the current state of radio button.
android:onClick - It's the name of the method to invoke when the radio button clicked.
android:visibility - It is used to control the visibility of control.

Radio Group:

In android, we use radio buttons with in a RadioGroup to combine multiple radio buttons into one group and it will make sure that users can select only one option from the group of multiple options.

```
<RadioGroup
  android:id="@+id/radioGroup"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_marginTop="16dp"
  android:orientation="vertical">

  <RadioButton
```

```
    android:id="@+id/radioButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 1" />

  <RadioButton
    android:id="@+id/radioButton2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 2" />

  <RadioButton
    android:id="@+id/radioButton3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 3" />

</RadioGroup>
```

In this example:

- RadioGroup contains three RadioButton elements.
- android: orientation="vertical" is used to align the radio buttons vertically. You can use "horizontal" if you want them aligned horizontally.
- Each RadioButton has its own android:id and android:text.

Using picker Views:

- Picker views in Android are specialized UI components that allow users to select a value from a predefined set of values. Selecting a date and time is one of the common tasks you need to perform in a mobile application. Android supports this functionality through the TimePicker and DatePicker views.
  1. Date picker View
  2. Time picker View

Time Picker View:

- In Android, TimePicker is a widget used for selecting the time of the day in either AM/PM mode or 24 hours mode. The displayed time consist of hours, minutes and clock format.
- Generally in android TimePicker available in two modes
  1. Show the time in clock mode
  2. Show the time in Spinner mode

TimePicker with clock mode:

- We can define the Timepicker to show time in clock format by using
- android: timePickerMode attribute.
- android:timePickerMode="clock": This attribute sets the mode of the TimePicker to display a clock-style time picker.

- android:format24Hour="HH:mm": This sets the format of the time displayed. In this case, it's in 24-hour format (e.g., 13:45).

```
<TimePicker
android:id="@+id/simpleTimePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:timePickerMode="Clock"/>
```

TimePicker with Spinner mode:

- The TimePicker displays a standard UI to enable users to set a time. By default, it displays the time in the AM/PM format. If you want to display the time in the 24-hour format, you can use¬ the setIs24HourView () method.
- We can define the Timepicker to show time in Spinner format by using android: timePickerMode attribute

Code:

```
<TimePicker
android:id="@+id/simpleTimePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:timePickerMode="spinner"/>
```

Date picker:

- In android, DatePicker is a control that will allow users to select the date by a day, month and year in our application user interface.
- If we use DatePicker in our application, it will ensure that the users will select a valid date. Following is the pictorial representation of using a datepicker control in android applications.
- Generally, in android DatePicker available in two modes,
  1. Show the complete calendar
  2. Show the dates in spinner view.

Date Picker in Calendar format:

- We can define the Datepicker to show Date in Calender format by using android: DatePickerMode attribute.

  Code:

  ```
  <DatePicker
  android:id="@+id/simpleDatePicker"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:datePickerMode="Calender"
  ```
- The above code will return the DatePicker like as shown below

Date Picker in Spinner format:

- If we want to show the DatePicker in spinner format like showing day, month and year separately to select the
- date, then by using DatePicker android:datePickerMode attribute we can achieve this.
- Following is the example of showing the DatePicker in Spinner mode.

Code:

```
<DatePicker
android:id="@+id/datePicker1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:datePickerMode="spinner"
android:calendarViewShown="false"/>
```

- The above code will return the DatePicker like as shown below

android:id - It is used to uniquely identify the control .

android:datePickerMode - It is used to specify datepicker mode either spinner or calendar.

android:background - It is used to set the background color for the date picker.

android:padding - It is used to set the padding for left, right, top or bottom of the date picker.

ListView :

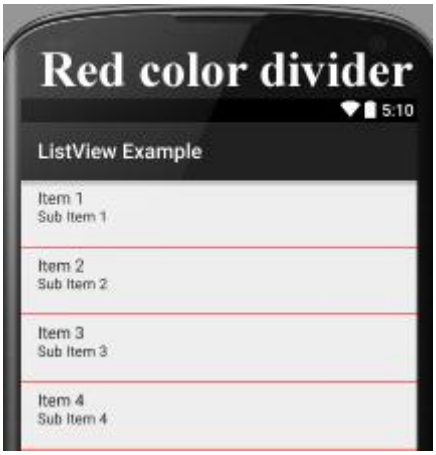
- Android ListView is a view which Contains several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database. ListView is implemented by importing android.widget.ListView class.
- In Android, there are two types of list views: **ListView and SpinnerView.**
- A very common example of ListView is your phone contact book, where you¬ have a list of your contacts displayed in a ListView and if you click on it then user information is displayed.

Code:

```
<ListView
android:id="@+id/simpleListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:divider="#f00"
android:dividerHeight="1dp"
/>
```

Main Activity.java code:

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends AppCompatActivity {
  ListView listView;
  String tutorials[] = { "Algorithms", "Data Structures", "Languages", "Interview Corner",    "GATE"};
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Initialize ListView
    listView = findViewById(R.id.list);
    // Create ArrayAdapter
    ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, tutorials);
    // Set ArrayAdapter to ListView
    listView.setAdapter(adapter);
  }
}
```



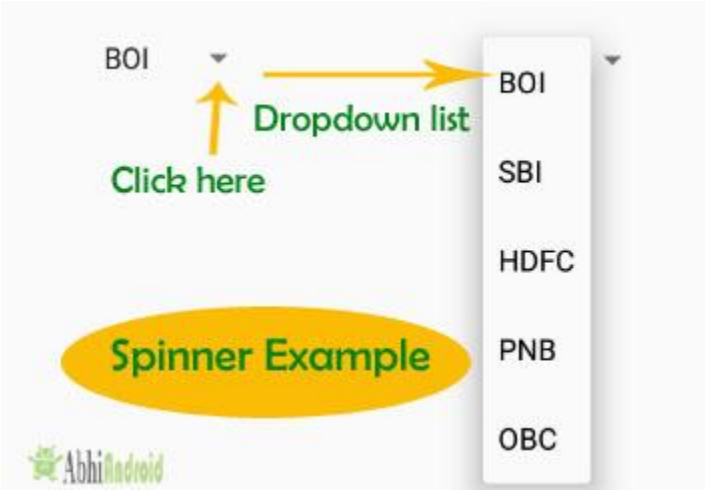
SpinnerView:

In Android development a spinner is a view, A spinner typically appears as a small rectangular box with an arrow icon at the right side. When the user taps on the spinner, a dropdown list of items appears below it, allowing the user to select one.

Code:

```
<Spinner
android:id="@+id/simpleSpinner "
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
```

- To fill the data in a spinner we need to implement an adapter class. A spinner is mainly used to display only text field so we can implement Array Adapter for that. We can also use Base Adapter and other custom adapters to display a spinner with more customize list.
- Suppose if we need to display a textview and a imageview in spinner item list then array adapter is not enough for that. Here we have to implement custom adapter in our class. Below image of Spinner and Custom Spinner will make it more clear.



2.2 Specialized fragments:

- Specialized fragments are fragments that serve specific purposes within an Android application. They are modular components that encapsulate a portion of the user interface and behavior, allowing for better organization and reusability of code. Here's an overview of some common types of specialized fragments and their purposes:
- **ListFragment:**
    - A ListFragment is a specialized fragment that displays a list of items. It typically uses a ListView or RecyclerView to present the data and handles item selection events.
    - ListFragment is useful for implementing master-detail interfaces, navigation drawers, or any screen that requires displaying a list of items.

Master - detail Interfaces :

- Imagine you're building a contacts application.
- The master portion of the interface displays a list of contacts (names or thumbnails).
- The detail portion shows the details of the selected contact, such as their name, phone number, email address, etc.

- **DialogFragment:**
    - DialogFragment is a specialized fragment that displays a dialog window, floating on top of the current activity's window.
    - DialogFragment is commonly used to show dialogs such as alerts, confirmation dialogs, input dialogs, or custom dialogs.
- For example, when users attempt to delete a file or exit the app, an alert dialog can prompt them to confirm their action. DialogFragment simplifies the creation and management of alert dialogs, ensuring a consistent and user-friendly experience.

- **PreferenceFragment:**
    - PreferenceFragment is a specialized fragment that displays a hierarchy of preference items as a list.
    - PreferenceFragment is used to create settings screens or preference screens where users can modify app settings.
    - In the social media app, a Preference Fragment could be used to display the app's settings screen, These XML files define preferences such as notification settings, privacy settings, theme preferences, etc.

2.3 Using Image views to display the pictures:

Adding an ImageView to an Activity:

- Whenever ImageView is added to an activity, it means there is a requirement for an image resource. It can be done by adding an image file that is present in the Android Studio itself or we can add our own image file. Android Studio owns a wide range of drawable resources which are very common in the android application layout.

1. **The following are the steps to add a drawable resource to the ImageView class.**

- Open the activity_main.xml File in which the Image is to be Added
- Switch from the Code View to the Design View of the activity_main.xml File.
- For adding an image from Android Studio, Drag the ImageView widget to the activity area of the application, a pop-up dialogue box will open choose from the wide range of drawable resources and click "OK".

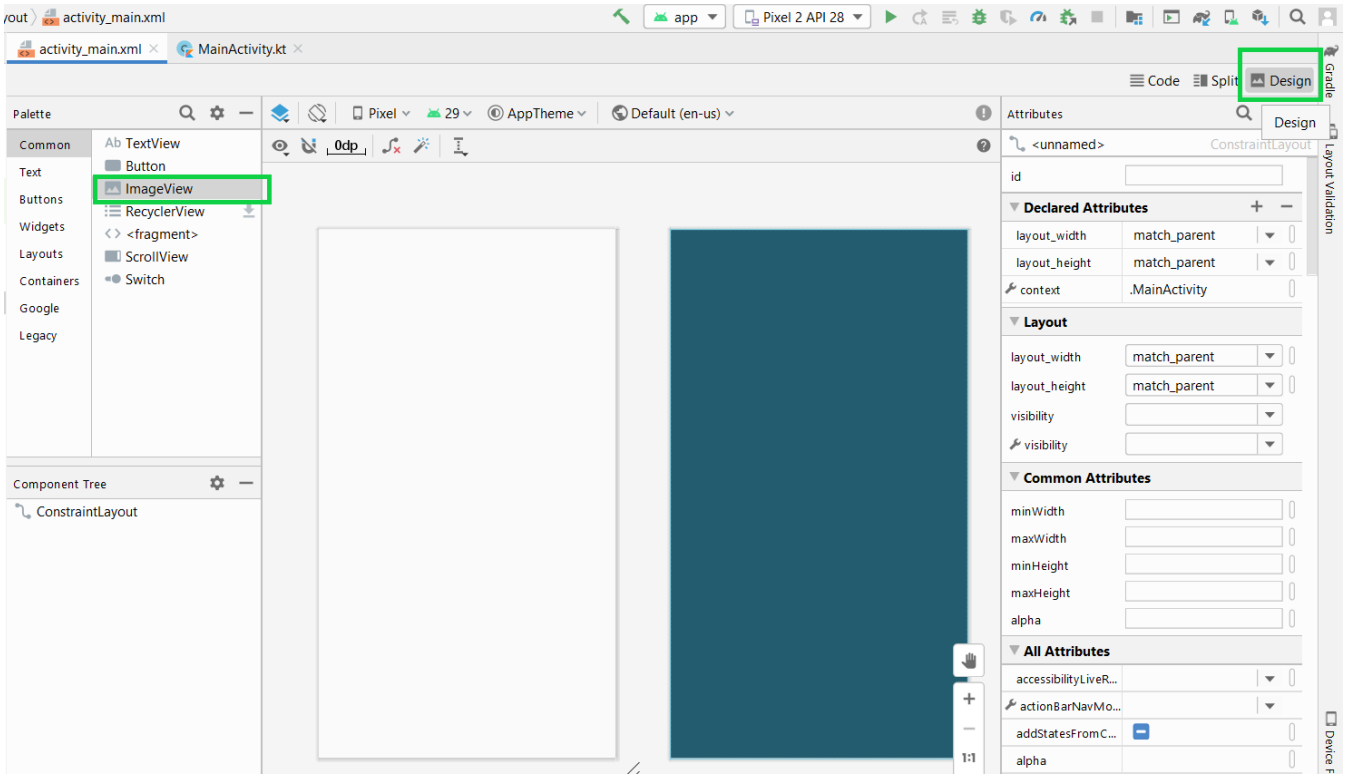
2. **For Adding an Image File other than Android Studio Drawable Resources: (Add a Resource Images.**

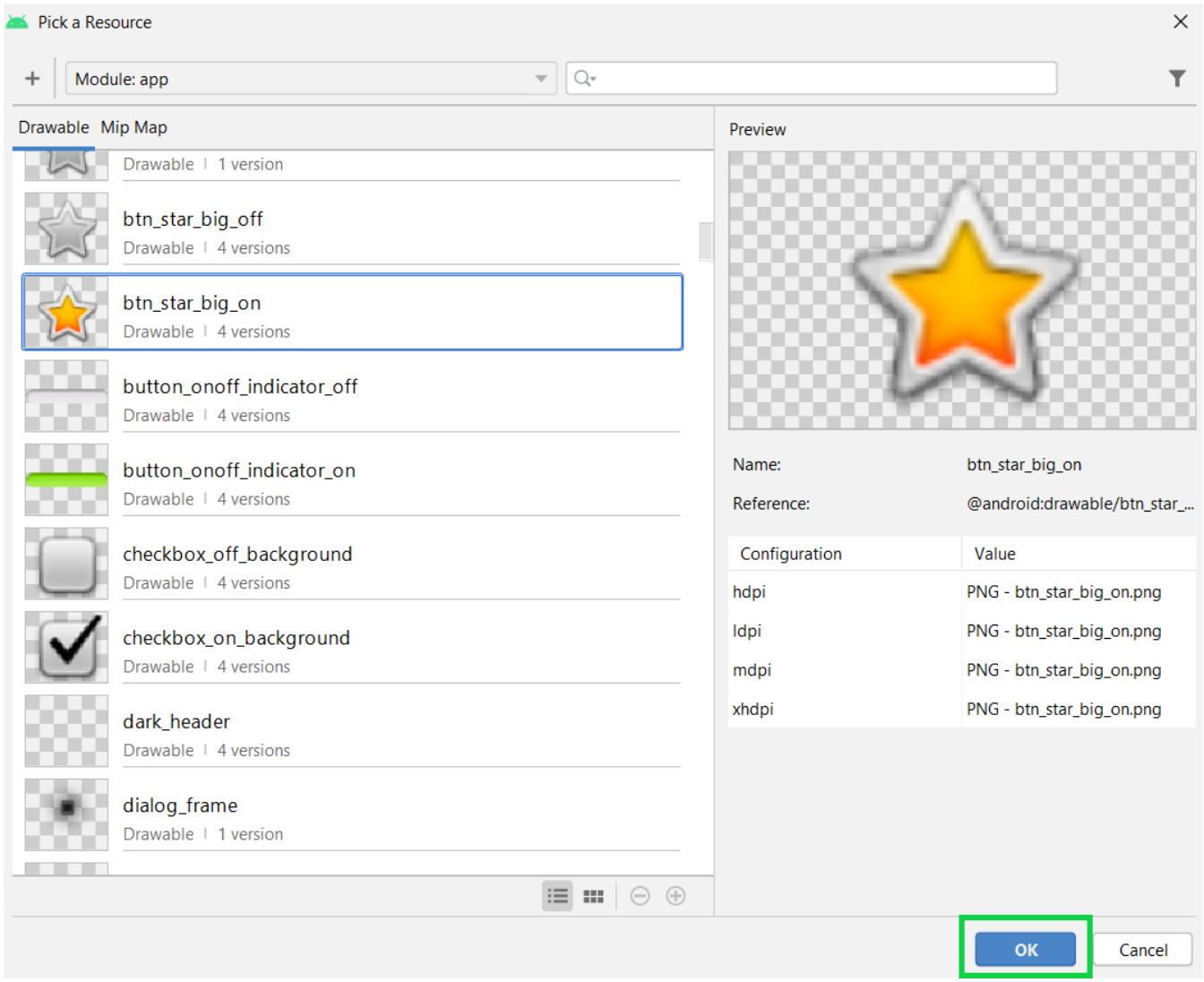
   Step 1: Create a new project and name it ImageViewExample.
   Step 2: Download two images lion and monkey from the web. Now save those images in the drawable folder of your project.
   Step 3: Now open res -> layout -> activity_main.xml (or) main.xml and add following code:
   In this step we add the code for displaying an image view on the screen in a relative layout.

   Code:

```
<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"
   tools:context=".MainActivity">

   <ImageView
      android:id="@+id/simpleImageViewLion"
      android:layout_width="fill_parent"
      android:layout_height="200dp"
      android:src="@drawable/lion" />

   <ImageView
   android:id="@+id/simpleImageViewMonkey"
   android:layout_width="fill_parent"
   android:layout_height="200dp"
   android:layout_below="@+id/simpleImageViewLion"
   android:layout_marginTop="10dp"
   android:src="@drawable/monkey" />

</RelativeLayout>
```


ImageViewExample
Monkey


activity_main.xml
MainActivity.kt
app
Pixel 2 API 28
Code
Split
Design
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch
Component Tree
ConstraintLayout
Attributes
<unnamed>
id
Declared Attributes
layout_width
match_parent
layout_height
match_parent
context
.MainActivity
Layout
visibility
Common Attributes
minWidth
maxWidth
minHeight
maxHeight
alpha
All Attributes
Gradle
Layout Validation


Pick a Resource
Module: app
Drawable
Mip Map
Drawable | 1 version
btn_star_big_off
Drawable | 4 versions
btn_star_big_on
Drawable | 4 versions
button_onoff_indicator_off
Drawable | 4 versions
button_onoff_indicator_on
Drawable | 4 versions
checkbox_off_background
Drawable | 4 versions
checkbox_on_background
Drawable | 4 versions
dark_header
Drawable | 4 versions
dialog_frame
Drawable | 1 version
Preview
Name:
btn_star_big_on
Reference:
@android:drawable/btn_star_...
Configuration
Value
hdpi
PNG - btn_star_big_on.png
ldpi
PNG - btn_star_big_on.png
mdpi
PNG - btn_star_big_on.png
xhdpi
PNG - btn_star_big_on.png
OK
Cancel

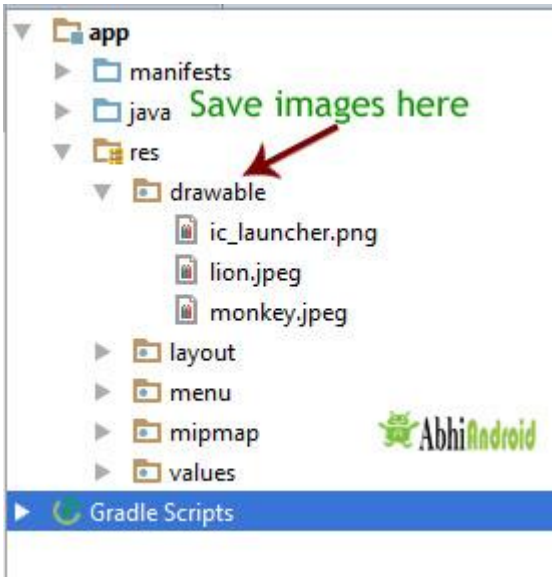
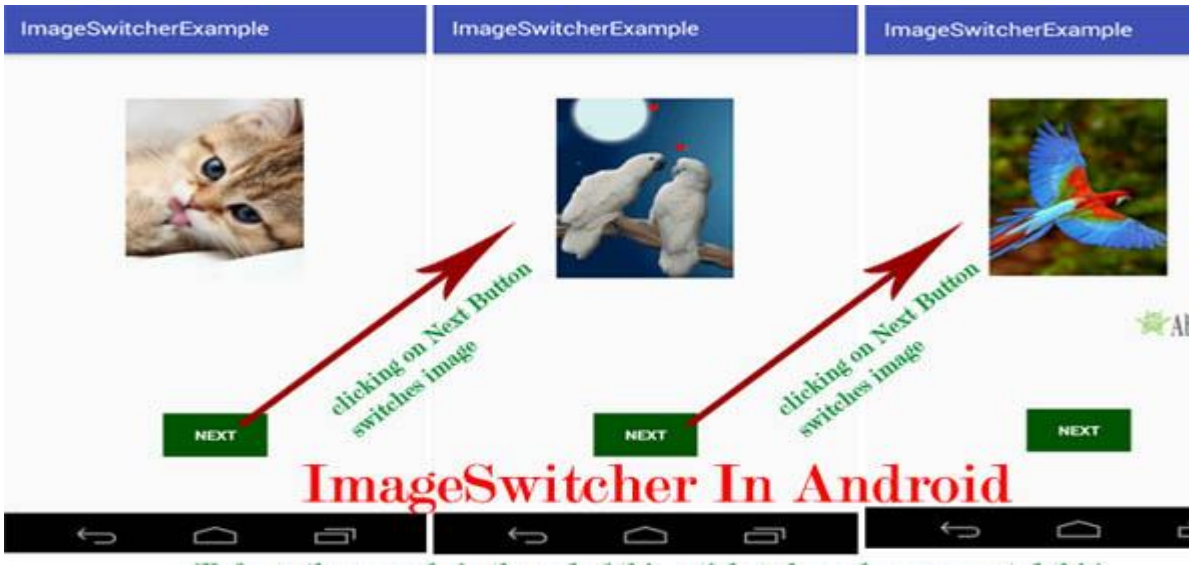


Image switcher:

In Android, ImageSwitcher is a specialized ViewSwitcher , t is mainly useful to animate an image on screen. It will provide a smooth transition animation effect to the images while switching from one image to another.



Basic ImageSwitcher code in XML:

```
<ImageSwitcher
   android:id="@+id/simpleImageSwitcher"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true" />
```

Steps for Implementation of ImageSwitcher:

- Get the reference of ImageSwitcher in class using findViewById () method or you can also create an object dynamically.
- Set a factory using **switcherid.setFactory()**
- Set an in-animation using **switcherid.setInAnimation()**
- Set an out-animation using **switcherid.setOutAnimation()**

o A factory is set using setFactory () to create ImageView objects dynamically. By using this method we create a new ImageView and replace the old view with that.

o In-animation and out-animation are set using setInAnimation () and setOutAnimation () methods.

```
// Set the animations
imageSwitcher.setInAnimation (AnimationUtils.loadAnimation (this, android.R.anim.fade_in));
imageSwitcher.setOutAnimation (AnimationUtils.loadAnimation (this, android.R.anim.fade_out));
// Set the first image
imageSwitcher.setImageResource (R.drawable.image1);
// to switch images
imageSwitcher.setImageResource (R.drawable.image2); // this triggers the animation
```

ImageSwitcher: Manages image transitions with animations.
In Animation: Animation for the image entering the view.
Out Animation: Animation for the image leaving the view.

In Animation (setInAnimation):

- This animation is applied to the new image that is entering the ImageSwitcher.
- In this example, android.R.anim.fade_in makes the new image fade into view.
- It's the animation you see when a new image appears.

Out Animation (setOutAnimation):

- This animation is applied to the current image that is exiting the ImageSwitcher.
- In this example, android.R.anim.fade_out makes the current image fade out of view.
- It's the animation you see when the old image disappears.
- The current image fades out.
- The new image fades in.

2.4 Using menus with Views:

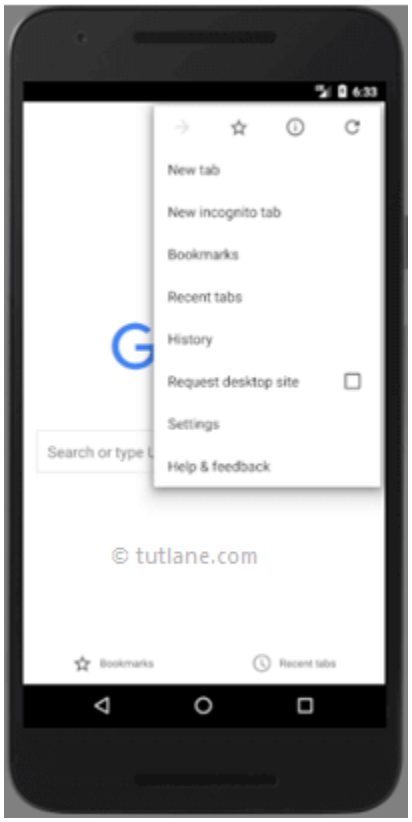
Menus are useful for displaying additional options that are not directly visible on the main UI of an application. There are two types of menus in application.

a) Option Menu
b) Context menu

Option menu: Option menu is a primary collection of menu items for an activity and it is useful to implement actions that have a global impact on the app, such as settings, search etc. It appears when the user presses the menu button on their device or the three-dot overflow button in the app's action bar.

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item
      android:id="@+id/action_search"
      android:title="Search"
      android:icon="@drawable/ic_search"
      android:showAsAction="ifRoom"/>
   <item
      android:id="@+id/action_settings"
      android:title="Settings"
      android:showAsAction="never"/>
</menu>
```



Context Menu: The context menu is a floating menu that appears when the user performs a long-click (tap and hold) on a view. It's typically used for actions that affect the selected item or context in which the menu was activated.

Options Menu:

- Used for global actions within the application.
- Typically accessed via the action bar or overflow menu.
- Common actions include settings, search, and other app-wide actions.

Context Menu:

- Used for context-specific actions related to a particular view or item.
- Accessed via a long-click on a view.
- Common actions include edit, delete, and other item-specific actions.

2.5 WEB View:

In Android, WebView is a view used to display the web pages in application. In Android, a WebView is a view that displays web pages, allowing you to integrate web content into your app. It's essentially a mini-browser embedded within your application.
You can use it to show web pages, render HTML content, execute JavaScript, handle user interactions like clicking links, and more.

1. Add the WebView to your layout XML file:

```
<WebView
  android:id="@+id/webview"
  android:layout_width="match_parent"
  android:layout_height="match_parent"/>
```

2. In your activity or fragment, find the WebView and load a web page:

Main Activity Java Code:

```
WebView webView = findViewById(R.id.webview);
webView.loadUrl("https://www.google.com");
```

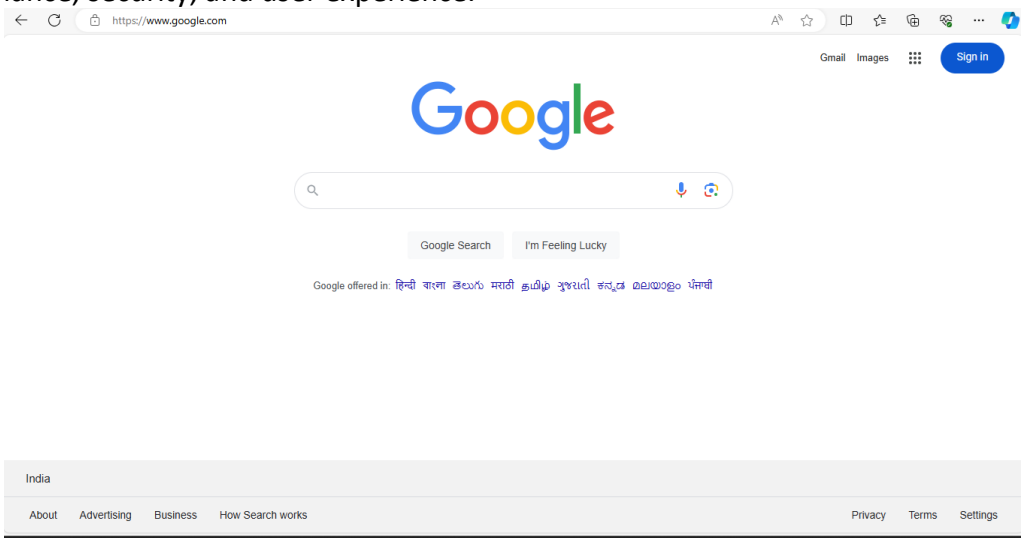
3. Optionally, you can enable JavaScript, handle page navigation events, add a WebViewClient for custom handling of page loading, etc.

```
webView.getSettings().setJavaScriptEnabled(true);
webView.setWebViewClient(new WebViewClient() {
   @Override
   public boolean shouldOverrideUrlLoading(WebView view, String url) {
      // Handle when a new URL is about to be loaded
      view.loadUrl(url);
      return true;
   }
});
```

4. Remember to add the necessary permissions in your AndroidManifest.xml file if your app requires internet access:

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

5. WebView is quite powerful and versatile, allowing you to create various types of content-rich applications, such as web browsers, hybrid apps combining native and web elements, or apps that display dynamic web content. However, it's essential to use it responsibly, considering performance, security, and user experience.



2.6 Saving and Loading User Preferences:

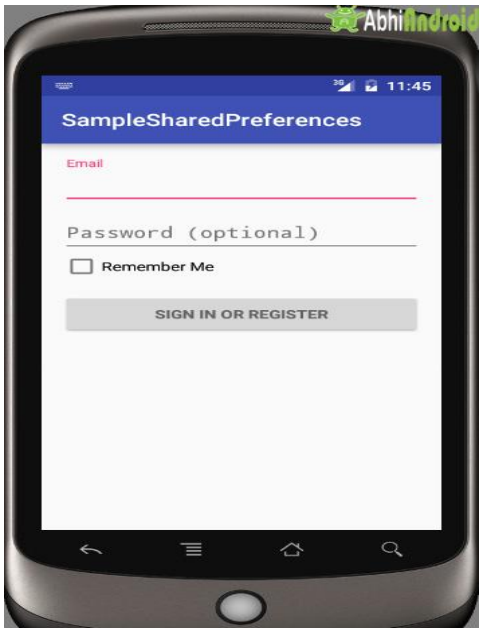
- Shared Preference in Android are used to save data and restore based on key-value pair. SharedPreferences data is shared and preferred data. Shared Preference can be used to save primitive data type: string, long, int, float and Boolean.
- Shared Preferences are suitable for different situations. For example, when the user's settings need to be saved or to store data that can be used in different activities within the app. As you know, onPause() will always be called before your activity is placed in the background or destroyed, So for the data to be saved persistently, it's preferred to save it in onPause(), which could be restored in onCreate() of the activity. The data stored using shared preferences are kept private within the scope of the application.
- Before we begin explaining shared preference, it is important to understand preference file.
- A preference file is actually a xml file saved in internal memory of device. Every application has some data stored in memory in a directory data/data/application package name. In order to use shared preferences, you have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

```
editor.putString("username", "JohnDoe");     // Save a String
editor.putInt("userAge", 25);                 // Save an int
editor.putLong("lastLogin", System.currentTimeMillis()); // Save a long
editor.putFloat("userHeight", 5.9f);     // Save a float
editor.putBoolean("notificationsEnabled", true); // Save a boolean
```

There are three types of Mode in Shared Preference:

- **Context.MODE_PRIVATE** – default value (Not accessible outside of your application)
- **Context.MODE_WORLD_READABLE** – readable to other apps
- **Context.MODE_WORLD_WRITEABLE** – read/write to other apps
- **MODE_PRIVATE –** It is a default mode. MODE_PRIVATE means that when any preference file is created with private mode then it will not be accessible outside of your application. This is the most common mode which is used.
- **MODE_WORLD_READABLE –** If developer creates a shared preference file using mode world readable then it can be read by anyone who knows it's name, so any other outside application can easily read data of your app. This mode is very rarely used in App.
- **MODE_WORLD_WRITEABLE –** It's similar to mode world readable but with both kind of accesses i.e read and write. This mode is never used in App by Developer. It grants both read and write access to all other applications on the device. This means any app installed on the device can potentially modify the data saved by your app, which could lead to various security and stability issues.
- Let's use Shared Preference for a very basic purpose of saving login details of a person i.e. email and password on his device. So he don't have to re-enter his login details every time he opens the App.

- Below is the final output we will create and use Shared Preference to save Signin Details:




```
public void saveLoginDetails(String email, String password) {
    SharedPreferences sharedPreferences = context.getSharedPreferences("LoginDetails", Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString("Email", email);
    editor.putString("Password", password);
    editor.commit();
  }
  public String getEmail() {
    SharedPreferences sharedPreferences = context.getSharedPreferences("LoginDetails", Context.MODE_PRIVATE);
    return sharedPreferences.getString("Email", "");
```

2.7 Persisting Data to files:

In android you can persist data to files using various methods. One common approach is to use internal or external storage to save files. Persisting data to files in Android involves storing data in the device's storage so that it can be retrieved and used later. Android provides several ways to persist data, including shared preferences, internal storage, external storage, and databases.

Internal Storage in Android

- The stored data in memory is allowed to read and write files.
- When files are stored in internal storage these file can only be accessed by the application itself not by other applications.
- These files in storage exist till the application stays over the device, as you uninstall associated files get removed automatically.
- The files are stored in directory data/data which is followed by the application package name.
- User can explicitly grant the permission to other apps to access files.
- To read and write in the android internal storage we have two methods

  OpenFileOutput(): used for creating and saving a file. This method returns a FileOutputStream instance.

  Syntax: *OpenFileOutput(String filename,int mode)*

  - *Context.MODE_PRIVATE: If the file exists then it is overridden else a new file is created.*
  - *Context.MODE_APPEND: if the file exists then the data is appended at the end of the file.*

  OpenFileInput(): Used to read data from a file, this returns an FileInputStream instance.

  Syntax: *OpenFileInput(String filename)*

```
FileOutputStream fileobj = openFileOutput( File_Name, Context.MODE_PRIVATE);
byte[] ByteArray = Data.getBytes(); //Converts into bytes stream
fileobj.write(ByteArray); //writing to file
fileobj.close(); //File closed
```

External Storage

- The data is stored in a file specified by the user itself and user can access these file. These files are only accessible till the application exits or you have SD card mounted on your device.
- *It is necessary to add external storage the permission to read and write. For that you need to add permission in android Manifest file.*
- *Open AndroidManifest.xml file and add permissions to it just after the package name.*
- ***<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />***
- ***<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />***

Methods to Store Data In Android:

- **getExternalStorageDirectory()** – Older way to access external storage in API Level less than 7. It is absolute now and not recommended. It directly get the reference to the root directory of your external storage or SD Card.
- **getExternalFilesDir(String type) –** It is recommended way to enable us to create private files specific to app and files are removed as app is uninstalled. Example is app private data.

- **getExternalStoragePublicDirectory() :** This is current recommended way that enable us to keep files public and are not deleted with the app uninstallation. Example images clicked by the camera exists even we uninstall the camera app.

2.8 Creating and Using Databases :