# PART-B ESSAY QUESTIONS WITH SOLUTIONS

## 2.1 ANDROID USER INTERFACE

**Q9. Explain the following terminology,**

(a) Views

(b) ViewGroup

(c) Fragments

(d) Activities.

**Answer :**

**(a) Views**

Views are considered as the base (or) root class for the visual interface elements i.e., controls or widgets. The layout classes and UI controls are derived from view.

**(b) ViewGroup**

ViewGroup is a combination of multiple views. It is an extension of view class carrying multiple child views. The ViewGroup class must be extended to create combined controls from Childviews and to help the layout Managers in designing the controls.

**(c) Fragment**

Fragments can be defined as the sub-activities present within an existing activity. They carry their own collection of views. They can be considered as an alternative form of a specific activity.

**(d) Activity**

For answer refer Unit-I, Q34(i).

**Q10. How user interfaces are created? Explain.**

Model Paper-II, Q4(a)

**Answer :**

A user interface in android can be created in three ways which are as follows,

1. Through user interface in Java code
2. Through user interface in XML.

However, creation usually involves both Java code and XML.

**Activity**

For answer refer Unit-I, Q34(i).

**Example**

```
<?xml version = "1.0" encoding = "utf-8"?>
<LinearLayout xmlns: android = "http://userInterface.android.com/apk/result/android"
android: orientation = "horizontal"
android: layout_height = "fill_parent"
android: layout_width = "fill_parent"
<TextView
    android: layout_height = "wrap_content"
    android: layout_width = "fill_parent"
    android: text = "@string/Hello. This is an example program"/>
</LinearLayout>
```

The user can load the above XML user interface file at runtime in the onCreate( ) method handler of Activity class by using the setContentView( ) method. The code for onCreate( ) is illustrated below;

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

The elements present in the above XML user interface file can be compiled into respective Android GUI class along with attributes of methods (at compilation time). The user interface of an activity gets created when the file is loaded.

The above XML code with be saved in res/layout folder. The controls used in it will be laid in some order either horizontally or vertically. The layout used above is linear layout where, even other layouts can be used to change the layout.

## 2.1.1 Measurements : Device and Pixel Density Independent Measuring Units

**Q11. Write in brief about device and pixel density independent measuring units.**

**Answer :** Model Paper-III, Q4(a)

The pixel density of screen display is defined as the ratio of number of pixels on a display and the size of the display. It is measured in terms of dots per inch (dpi). There are various Android devices available in market with different sizes and resolutions. But based on dpi of screen, devices of different sizes are developed with same number of pixels. With this, it is difficult to design the layouts with specified pixels. So, android uses density-independent pixels which is also referred as device-independent pixels. It is used to specify the dimensions of screen to be displayed on screens of same size but with multiple pixel densities.

The density-independent pixels can be used in the application inorder to avoid the specifications such as ViewSizes, layout dimensions etc.

The Scale-independent Pixel (SP) is used for font sizes. It shares same base unit similar to that of density independent pixel but scaled independently depending upon user's preferred size of text. The users are allowed to increase the font size for all the applications on the device.

Android 5.0 lollipop (API level 21) version provides support for device independent vector drawables. These Vector Drawables are defined in XML and they can be scaled for supporting any type of display densities. Other than this, there are assets which cannot be defined as vector graphics. The Android resource system then down scales the graphics or allows to provide the multiple resources in several folders. Making the designs with density independent pixels allow the user to focus on optimizing as well as adapting designs for various screen sizes.

## 2.2 LAYOUTS – LINEAR, RELATIVE, GRID AND TABLE LAYOUTS

**Q12. Discuss about the following,**

**(i) Linear layout**

**(ii) Relative layout.**

**Answer :** Model Paper-I, Q4(a)

**(i) Linear Layout**

Linear Layout is considered as a basic layout which organizes the views in a row/column format. It arranges the child views either in horizontal or vertical form.

**Attributes of LinearLayout**

The various attributes of linear layout are as follows,

**1. android : orientation**

This attribute is used to arrange the controls in the container either horizontally or vertically. The values to be assigned to it are vertical or horizontal. The method setOrientation( ) is used to modify the orientation at runtime.

**2. android : layout_width**

The default width depends on the text or content to be displayed. This attribute is used to specify the width of the layout. The values for this attribute can be defined in three ways,

(a) The value can be specified as px (Pixels, dip/dp(device independent pixels), sp(scaled pixels), pts(points), in(inches) and mm(millimeters).

**Example:** android : layout_width = "50px".

(b) The value can be specified as wrap_content. It makes the control resizable.

(c) The value can be specified as match_parent. It makes the control to expand upto the container space.

**3. android : layout_height**

The default height depends on the text or content to be displayed. This attribute is used to specify the height of the layout. The values for this attribute can be defined in three ways,

(a) The value can be specified as px, dip/dp(device independent pixels), sp(scaled pixels), pts(points), in(inches) and mm(millimeters).

**Example:** android : layout_height = "50px".

(b) The value can be specified as wrap_content. It makes the control resizable.

(c) The value can be specified as match_parent. It makes the content to expand upto the container space.

**4. android : padding**

This attribute is used for increasing the whitespace in between the boundaries of actual content and control. The same padding or spacing is allowed to be specified on all the sides of control. Individual spacing on four sides of control is also allowed by specifying android : paddingLeft, android : paddingRight, android : paddingTop and android : paddingBottom attributes.

**Example**

```
android:padding="10dip"
android:paddingLeft="10dip"
```

**android : layout_weight**

This attribute makes the control to grow or shrink as per the space. The value to be assigned to it ranges from 0.0 to 1.0.

6. **android:gravity**

This attribute is used to align the content of the control. The various options that can be used for it are as follows,

(a) **center_vertical**

It will place the object in vertical center of the container without affecting the size.

(b) **center_horizontal**

It will place the object in horizontal center of the container without affecting the size.

(c) **center**

It will place the object in center of the container either horizontally or vertically without affecting the size.

(d) **fill_vertical**

It will increase the vertical size of object to fill the container.

(e) **fill_horizontal**

It will increase the horizontal size of object to fill the container.

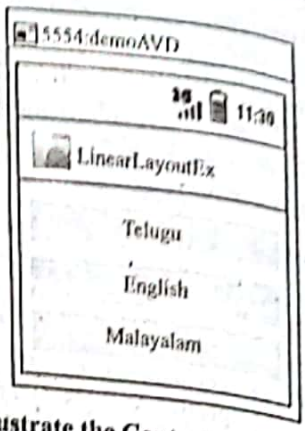
7. **android : layout_gravity**

This attribute is used by the container. It aligns the control in the container. The values for it are left, right and center.

**Example to Illustrate the Controls in Vertical Form**

```
<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/Telugu"
    android:text="Telugu"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
  <Button
    android:id="@+id/English"
    android:text="English"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
  <Button
    android:id="@+id/Malayalam"
    android:text="Malayalam"
    android;layout_width="fill_parent"
    android:layout_height="wrap_content" />
</LinearLayout>
```

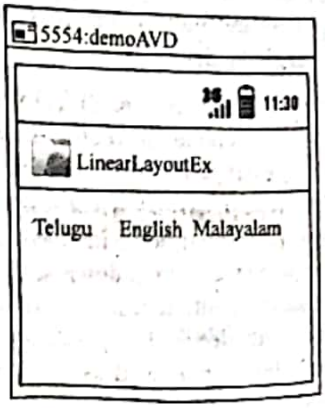
**Output**




**Example to Illustrate the Controls in Horizontal Form**

```
<LinearLayout xmlns:android="http:
                    //schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="horizontal" >
  <Button
    android:id="@+id/Telugu"
    android:text="Telugu"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <Button
    android:id="@+id/English"
    android:text="English"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <Button
    android:id="@+id/Malayalam"
    android:text="Malayalam"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

**Output**



**(ii) Relative Layout**

Relative layout allows users to define the positioning of child views (inherited from view group) relative to each other. Every element is related to other child element as well as the parent element.

**Attributes of Relative Layout**

The attributes to set the location of the control related to the container are as follows,

1. **android : layout_alignParentTop**
   It is used to set the top of control to align with the container top.
2. **android : layout_alignParentBottom**
   It is used to set the bottom of control to align with container bottom.
3. **android : layout_alignParentLeft**
   It is used to set the left side of the control to align with the containers left side.
4. **android : layout_alignParentRight**
   It is used to set the right side of control to align with the containers right side.
5. **android ; layout_centerHorizontal**
   It is used to place the control horizontally at center.
6. **android : layout_centerVertical**
   It is used to place the control vertically at center.
7. **android : layout_centerInParent**
   It is used to place the control horizontally and vertically at the center.

The attributes to set the control position related to other controls are as follows,

1. **android : layout_above**
   It is used to place the control above the referenced control.
2. **android : layout_below**
   It is used to place the control below the referenced control.
3. **android : layout_toLeftOf**
   It is used to place the control at the left of referenced control.
4. **android : layout_toRightOf**
   It is used to place the control at the right of the referenced control.

The attributes to control the alignment of the control related to other controls are as follows,

1. **android : layout_alignTop**
   It is used to set the top of the control to align with referenced control's top.
2. **android : layout_alignBottom**
   It is used to set the bottom of control to align with referenced controls bottom.
3. **android : layout_alignLeft**
   It is used to set the left side of control to align at referenced controls left.
4. **android : layout_alignRight**
   It is used to set the right side of control to align at referenced controls right.
5. **android : layout_alignBaseline**
   It is used to align the baseline of two controls.

The attribute android : padding will define the spacing for the view. The various options of this attribute are as follows,

1. **android : paddingTop**
   It defines the space in between the top of control and its content.
2. **android : paddingBottom**
   It defines the space in between the bottom of control and its content.
3. **android : paddingLeft**
   It defines the space in between the left of control and its content.
4. **android : paddingRight**
   It defines the space in between the right of the control and its content.

The attribute android : layout_margin will define the space for the container. The various options of this attribute is as follows,

1. **android : layout_marginTop**
   It specifies the space between the top of control and related control or container.
2. **android : layout_marginBottom**
   It specifies the space between the bottom of control and related control or container.
3. **android : layout_marginRight**
   It specifies the space between the right side of control and related control or container.
4. **android : layout_marginLeft**
   It specifies the space between the left side of control and related control or container.

**Program**

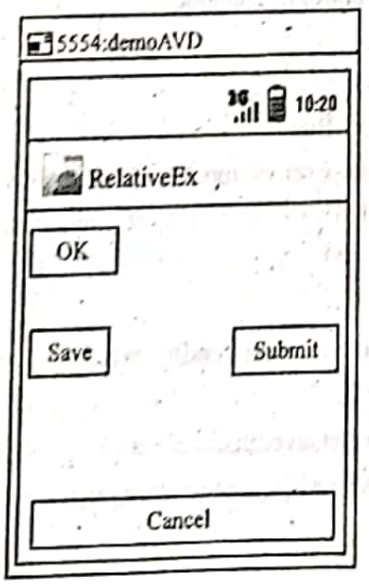
**RelativeEx.java**

```
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
public class RelativeEx extends AppCompatActivity
{
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.relative_ex);
  }
}
```

relative_ex.xml

```
<RelativeLayout xmlns:android="http:
                    //schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:paddingLeft="10dp"
  android:paddingRight="10dp">
  <Button
    android:id="@+id/b1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:text=" OK " />
  <Button
    android:id="@+id/b2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_centerVertical="true"
    android:text=" Save " />
  <Button
    android:id="@+id/b3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentRight="true"
    android:layout_centerVertical="true"
    android:text="Submit" />
  <Button
    android:id="@+id/b4"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text=" Cancel " />
</RelativeLayout>
```

**Output**




**Q13. Write short notes on the following.**

**(I) Grid layout**

**(II) Table layout.**

**Answer :**

**(I) Grid Layout**

The Grid Layout will arrange the views in the form of two-dimensional grid pattern. It will be in the form of rows and columns. The intersection of row and column is referred as a cell. The child views can be placed in a cell.

**Specifying Row and Column Position**

The row and column position can be specified within the cell by using the android : layout_row and android : layout_column attributes.

**Spanning Rows and Columns**

Rows or columns can be spanned by the views by using the android : layout_rowSpan and android : layout_columnSpan attributes.

**Inserting Spaces in Grid Layout**

A spacing view known as space is used for inserting the space. It is inserted as child view.

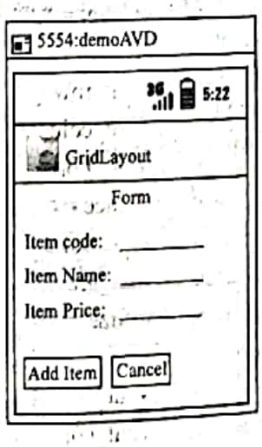
```
<Space
    android : layout_row = "3"
    android : layout_column = "0"
    android : layout_columnSpan = "3"
    android : layout_gravity = "fill"/>
```

**Example**

```
<GridLayout 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 : orientation = "horizontal"
    android : rowCount = "5"
    android : columnCount = "2">
    <TextView
        android : layout_row = "0"
        android : layout_column = "0"
        android : text = "Form"
        android : typeface = "Serif"
        android : layout_columnSpan = "3"
        android : layout_gravity = center_horizontal"
        android : textSize = "30dip"/>
    <Space
        android : layout_row = "1"
        android : layout_column = "0"
        android : layout_width = "30dp"
        android : layout_height = "15dp"/>
    <TextView
        android : layout_row = "2"
        android : layout_column = "0"
        android : text = "Item Code:"/>
```

```
<EditText
    android : id = "@+id/item_code"
    android : layout_width = "100dip"/>
<TextView
    android : text "Item Name:"/>
<EditText
    android : layout_row = "3"
    android : layout_column = "1"
    android : id = "@+id/i_name"
    android : layout_width = "200dip"/>
<TextView
    android : layout_row = "4"
    android : layout_column = "0"
    android : text = "Item Price:"/>
<EditText
    android : layout_row = "4"
    android : layout_column = "1"
    android : id = "@+id/i_price"
    android : layout_width = "80dip"/>
<Space
    android : layout_row = "5"
    android : layout_column = "0"
    android : layout_width = "30dp"
    android : layout_height = "10dp"/>
<Button
    android : layout_row = "6"
    android : layout_column = "0"
    android : id = "@+id/add_button"
    android : text= "Add Item"/>
<Button
    android : id = "@+id/cancel_button"
    android : text = "Cancel"/>
</GridLayout>
```

Output




(ii) **Table Layout**

The Table Layout arranges the views in the form of rows and columns. Every row is defined using TableRow object. A row can contain any number of columns and each column is called as cell. A cell can even embed other Table Layouts in it. Various operations that can be performed on the TableLayout are as follows,

1. **Stretching the Columns**

   The width of column will be width of widest column by default. They can be stretched using android : stretchColumns attribute in Table Layout. The value of this attribute can be either a single column number or comma-delimited column numbers.

   **Example:** android : stretchColumns = "1"

   It stretches the second column to use the available row space.

2. **Shrinking Columns**

   The columns can be shrinked or reduced by using android : shrinkColumns attribute. The value of this attribute can be either single column or comma-delimited column numbers.

   **Example:** android : shrinkColumns = "0"

   It shrinks or reduces the first columns width by word wrapping the content.

3. **Collapsing Columns**

   The columns can be made to collapse or invisible using android : collapseColumns attribute. It accepts one or more than one comma delimited columns as the value to this attribute. The columns can be made visible or invisible by passing the true or false boolean values for setColumnCollapsed( ) method in TableLayout.

   **Example**

   android : collapseColumns = "0"

   It collapses the first column of the layout.

4. **Spanning Columns**

   The columns can be made to span or use the space of other columns by using the android:layout_span attribute. The value that is greater than or equal to 1 i.e., > = 1 must be passed to this attribute.

   **Example**

   android : layout_span = "3"

**Program**

```
<TableLayout xmlns : android = "http:
                //schemas.android.com/apk/res/android"
    android : orientation = "vertical"
    android : layout_width = "match_parent"
    android : layout_height = "match_parent"
    android : stretchColumns = "1">
<TableRow android : padding = "5dip">
<TextView
    android : layout_height = "wrap_content"
    android : text = "Sign In"
    android : typeface = "Serif"
    android : layout_span = "2"
    android : gravity = "center_horizontal"
    android : textsize = "20dip"/>
</TableRow>
<TableRow>
```

```
<TextView
    android : layout_height = "wrap_content"
    android : text = "Email:"
    android : layout_column = "0"/>
<EditText
    android : id = "@ + id/em1"
    android : layout_height = "wrap_content"
    android : layout_column = "2"/>
</TableRow>
<TableRow>
    <TextView
        android : layout_height = "wrap_content"
        android : text = "password"
        android : layout_column = "0"/>
    <EditText
        android : id = "@ + id/P1"
        android : layout_height = "wrap_content"
        android : scrollHorizontally = "true"/>
</TableRow>
<TableRow>
    <Button
        android : id = "@ + id/button"
        android : text = "SIGN IN"
        android : layout_height = "wrap_content"/>
    <Button
        android : id = "@ + id/f_button"
        android : text = "FORGOT PASSWORD"
        android : layout_height = "wrap_content"/>
</TableRow>
</TableLayout>
```

**Output**


## 2.3 USER INTERFACE (UI) COMPONENTS – EDITABLE AND NON-EDITABLE TEXTVIEWS, BUTTONS, RADIO AND TOGGLE BUTTONS, CHECKBOXES, SPINNERS, DIALOG AND PICKERS

**Q14. Explain about TextView Control. How it can be made editable and non-editable.**

**Answer :** Model Paper-I, Q4(b)

**TextView Control**

TextView control allows users to remove the existing default text and enter the new text. There are two different ways for assigning the text to the textview.

(i) Assigning the text directly to text view control.

(ii) Assigning text indirectly to textview control by using the Java Activity file.

**Text Assigned Directly In Layout File**

The text to be displayed is assigned to the textview control in the xml layout file. Double click the res/layout folder in the package explorer dialogbox for opening the XML file. The following code is written in xml layout file.

```
<RelativeLayout>
    xmlns: android1 = "http:
            //schemas.android.com/apk/res/android1"
    xmlns: tools1 = "http:
                    //schemas.android.com/tools1"
    android1: layout_width = "match_parent"
    android1: layout_height = "match_parent">
<TextView>
    android1: layout_width = "wrap_content"
    android1: layout_height = "wrap_content"
    android1: layout_centerHorizontal = "true"
    android1: layout_centerVertical = "true"
    android1: text = "Hello!"
    tools1: context = ".HelloWorldAppActivity"/>
</RelativeLayout>
```

In the above code, the text is assigned to the Textview control using an attribute android1:text = "Hello!". When this application is run then the following output is generated.

**Output**

**Text Assigned in Activity File**

The text is assigned to the TextView control in the java activity file i.e., Helloworld.java by using the two ways as given below;

(i) **Removing/Deleting Text from an xml File**

Initially, the text assigned to Textview control is removed from the xml file.

(ii) **Assigning an ID to Textview**

The text view control can be accessed in activity file by assigning it with an unique ID.

The android : id is the attribute used for assigning an id to a textview control. This attribute is added in the activity file i.e., activity_hello_world_app.xml file. The android : id = "@ + id/message" is the new text in which, message is a constant assigned to textview control as an ID. The + sign present in the new text defines that if ID, message are not available then they are required to be created.

The following is the code of layout file activity_hello_world.xml in which one text is removed and other text is added.

```
<RelativeLayout xmlns: android1 = "http:
        //schemas.android.com/apk/res/android1"
    xmlns: tools1 = "http:
        //schemas.android.com/tools1"
    android1: layout_width = "match_parent"
    android1: layout_height = "match_parent">
<TextView
    android1: id = "@ + id/message"
    android1: layout_width = "wrap_content"
    android1: layout_height = "wrap_content" .
    tools1: context = ".HelloWorldAppActivity1"/>
</RelativeLayout>
```

An ID that is assigned to the textview control can be accessed in activity file. The text can be assigned to the text view control in the java activity file.

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
public class HelloWorld extends Activity
{
@Override
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    TextView msg = (TextView) findViewById
                        (R.id.message);
    msg.setText("Hello!")'
}
}
```

**Output**


**Editable and Non-editable TextViews**

TextViews can be made editable or read only using EditText and Label classes respectively.

## Q15. Write and explain about the edit text control.

**Answer :**

**Edit Text Control**

This control is used to extract the input from the user. It is the subclass of TextView. It has various attributes to configure as per the requirements. By default, it has some restrictions. It displays text in one line and moves to the next line when text goes beyond the width. Such constraints can be modified by the users.

Attributes of EditText control are as follows,

1. **android:layout_width**

   It is used to set the width of the EditText control. Examples of these values are wrap_content and match_parent. The former will set the width to only one character. Whereas the latter one sets width equal to the container width.

2. **android:layout_height**

   It is used to set the height of the EditText. Some values of it are wrap_content and match_parent. The former will get increased when input text is beyond the width. Whereas latter one will increase height equal to the container height.

3. **android:singleLine**

   It makes the control to allow only single character when set to true. Otherwise, the cursor moves to next line.

4. **android:hint**

   It shows certain information that is helpful to user in entering the text into it. User needs to type the data in it.

5. **android:textSize**

   It is sets the text size entered in EditText control. It specifies size in px, in, mm, dip, pts and sp units of measurement.

6. **android:autoText**
   It allows the EditText control to correct the spelling mistakes, when it is set to true.
7. **android:capitalize**
   It converts the text entered in it to capital letters.
8. **android:password**
   It converts the characters typed in it to dots in order to hide the data.
9. **android:minWidth**
   It specifies the minimum width of the control.
10. **android:minHeight**
    It specifies the minimum height of the control.
11. **android:maxWidth**
    It specifies the maximum width of the control.
12. **android:maxHeight**
    It specifies the maximum height of the control.
13. **android:scrollHorizontally**
    It makes the text to scroll horizontally when set to true.
14. **andriod:inputType**
    It specifies the type of data that is typed in EditText control.

**Q16. Write in brief about Button Control.**

**Answer :**

**Button**

Button is one of the most widely used controls. It is used to perform a specific action when the user clicks (or) presses the button. The various attributes of Button control are as follows,

(i) **android : id**
This attribute is used to define the id for the view.

(ii) **android : gravity**
This attribute is used to align the text as specified i.e., right, left, top, bottom etc.

(iii) **android : padding**
This attribute is used to set the padding i.e., right, left, top, bottom.

(iv) **android : text**
This attribute is used to specify the text.

(v) **android : textSize**
This attribute is used to set size of the text.

(vi) **android : textColor**
This attribute is used to modify color of the text.

(vii) **android : textStyle**
This attribute is used to modify style of the text.

(viii) **android : visibility**
This attribute is used to specify visibility of view.

**Example**

For answer refer Unit-II, Q17, Topic: Example (Radio Buttons).

**Q17. Write short notes on the following.**
(i) **Radio buttons**
(ii) **Toggle buttons.**

Model Paper II, Q.No.

**Answer :**

(i) **Radio Buttons**

Radio button control is a form of two state widgets i.e., checked or unchecked state. They are mutually exclusive thereby allowing only one button to be selected at a time in a group of radio buttons. A set of radio buttons are grouped into RadioGroup element to make them mutually exclusive. Initially a RadioButton group is created and filled with RadioButton controls.

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
<RadioButton android:id="@+id/rd_1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="single"/>
<RadioButton android:id="@+id+rd_2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Married"/>
</RadioGroup>
```

The methods that can be applied to RadioButton are as follows,

1. **IsChecked ( )**
   It checks whether the RadioButton is selected.
2. **toggle( )**
   It toggles the RadioButton state from selected to unselected or vice versa.
3. **check( )**
   It locates the matching RadioButton with the ID that is passed to the method.
4. **getCheckedRadioButtonId( )**
   It locates and gets the currently selected RadioButton control. If checked, it returns –1 otherwise it returns 0.

**Example**

**RadioButton.xml**

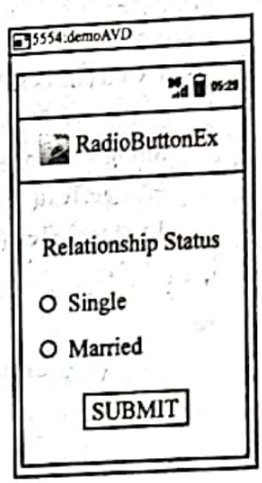
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation = "vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<Textview
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Relationship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

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation = "vertical">
<RadioButton
    android:id="@+id/rd_1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Single"/>
<RadioButton
    android:id="@+id/rd_2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Married"/>
</RadioGroup>
<Button
    android: id = "@+id/b1"
    android: layout_width = "wrap_content"
    android: layout_height = "wrap_content"
    android: text = "SUBMIT"
    androjd: onClick = "onclickbuttonMethod"
    android: layout_gravity = "center horizontal"/>
</LinearLayout>
```

**RadioButtonEx.java**

```
import android.app.activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.RadioButton;
import android.view.View;
import android.view.View.onClickListener;
public class RadioButtonEx extends Activity
{
    String str;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.RadioButton);
        RadioButton rd_1 = (RadioButton)
                           findViewById(R.id.rd_1);
        RadioButton rd_2 = (RadioButton)
                           findViewById(R.id.rd_2);
        Button b1 = (Button)findViewById(R.id.b1)
                    b1.SetOnClicklistener
                    (new view.OnClickListener( ))
        {
            @ Override
            public void onClick(view vi)
            {
                if(rd_1.isChecked( ))
                {
                    str = rd_1.getText( ).toString( );
                }elseif(rd_2.isChecked( ))
                {
                    str = rd_2.getText( ).toString( );
                }
                Toast.makeText(getApplication(Context( ),
                str, Toast.LENGTH_LONG).Show( );
            }
        });
    }
}
```

**Output**




**(ii) Toggle Buttons**

Toggle button in Android is used to display the checked or unchecked state of a button. It is a type of on/off button with a light indicator that depicts the current state of a toggle button. The users are allowed to change the setting in between two states on and off. Various methods of ToggleButton are as follows,

1. **CharSequence getTextOff( )**
   This method is used to return the text when button is in unchecked state.
2. **CharSequence getTextOn( )**
   This method is used to return the text when button is in checked state.
3. **void setChecked(boolean checked)**
   This method is used to change the checked state of the button.

*Example*

**Activity1.java**

```
import android.support.v7.pp.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Button;
import android.widget.Toast;
import android.widget.ToggleButton;
public class Activity1 extends AppCompatActivity
{
    ToggleButton tb1, tb2;
    Button ClickHere;
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tb1 = (ToggleButton)findViewById(R.id.tb1);
        tb2 = (ToggleButton)findViewById(R.id.tb2);
        Button ClickHere = (Button)findViewById(R.id.Submit)(Button);
        ClickHere.setOnClickListener(new View.onClickListener( ))
        {
            @Override
            public void onClick(View v)
            {
                String s = "Toggle_B1:"
                +tb1.getText( ) + "\n" + "Toggle_B2:" + tb2.getText( );
                Toast.makeText(getApplicationContext( ), s, Toast.LENGTH_SHORT).show( );
            }
        });
    }
}
```

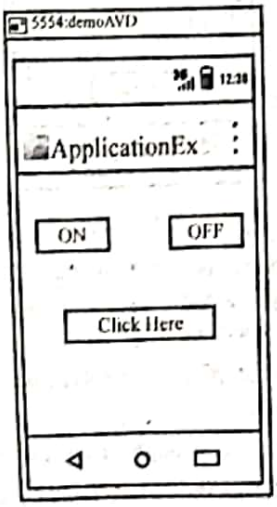
**Example**

**Main.xml**

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:orientation="vertical"
  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=".Main">
  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:orientation="horizontal">
```

```
<ToggleButton
  android:id="@+id/tb1"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="center_horizontal"
  android:checked="false"
  android:drawablePadding="30dp"
  android:drawableRight="@drawable/ic_launcher"
  android:textColor="#000" />
<ToggleButton
  android:id="@+id/tb2"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="center_horizontal"
  android:layout_marginLeft="40dp"
  android:checked="true"
  android:drawableLeft="@drawable/ic_launcher"
  android:drawablePadding="30dp"
  android:textColor="#000" />
</LinearLayout>
<Button
  android:id="@+id/submitButton"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="center"
  android:layout_marginTop="40dp"
  android:background="#0f0"
  android:padding="20dp"
  android:text="Click Here"
  android:textColor="#fff"
  android:textSize="30sp"
  android:textStyle="bold" />
</LinearLayout>
```

**Output**




**Q18. Discuss in brief about the following,**

**(i) Check boxes**

**(ii) Spinners.**

**Answer :** *Model Paper-III, Q4(b)*

**(i) CheckBoxes**

Checkbox is a special type of button. It has two states namely, checked and unchecked. It must be selected to check or uncheck itself. It can be created as shown below.

android.widget.checkbox

The methods of checkbox are as follows,

**1. isChecked( )**

This method returns true if checkbox is checked. Otherwise, it returns false.

**2. setChecked( )**

This method changes the state of the checkbox.

An event listener can be added by using the onCheckedchangeListener interface. It involves the callback method onCheckedchanged( ) whenever the checkbox state changes. An alternative is to use onClickListener interface that involves onClick( ) method whenever the checkbox is clicked.

**Example**

**Fruits.xml**

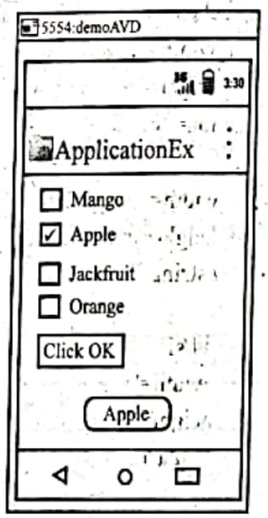
```
<LinearLayout xmlns:android="http:
                    //schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <CheckBox
     android:id="@+id/MangoCB"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:padding="10dp"
     android:layout_marginTop="150dp"
     android:layout_marginLeft="100dp"
     android:text="Mango"
     android:onClick="onCheckboxClicked"/>
   <CheckBox
     android:id="@+id/AppleCB"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:padding="10dp"
     android:layout_marginLeft="100dp"
     android:text="Apple"
     android:onClick="onCheckboxClicked"/>
```

```
<CheckBox
    android:id="@+id/JackfruitCB
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:layout_marginLeft="100dp"
    android:text="Jackfruit"
    android:onClick="onCheckboxClicked"/>
<CheckBox
    android:id="@+id/OrangeCB"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="10dp"
    android:layout_marginLeft="100dp"
    android:text="Orange"
    android:onClick="onCheckboxClicked"/>
<Button
    android:id="@+id/getB1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="100dp"
    android:text="Click Ok" />
</LinearLayout>
```

**CBExample.java**

```
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;
public class CBExample extends AppCompatActivity
{
  CheckBox mango, apple, jackfruit,orange;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // initiate views
    mango = (CheckBox) findViewById(R.id.MangoCB);
    apple = (CheckBox) findViewById(R.id.AppleCB);
    jackfruit = (CheckBox) findViewById(R.id.JackfruitCB);
    orange = (CheckBox) findViewById(R.id.OrangeCB);
        Button b1 = (Button)findViewById(R.id.getB1);
    b1.setOnClickListener(new View.OnClickListener()
  }
  @Override
  public void onClick(View vi) {
    switch (vi.getId()) {
      case R.id.MangoCB:
        if (mango.isChecked())
          Toast.makeText(getApplicationContext(),
              "Mango", Toast.LENGTH_LONG).show();
        break;
      case R.id.AppleCB:
        if (apple.isChecked())
          Toast.makeText(getApplicationContext(),
              "Apple", Toast.LENGTH_LONG).show();
        break;
      case R.id.JackfruitCB:
        if (Jackfruit.isChecked())
          Toast.makeText(getApplicationContext(),
              "Jackfruit", Toast.LENGTH_LONG).show();
        break;
      case R.id.OrangeCB:
        if (Orange.isChecked())
          Toast.makeText(getApplicationContext(),
              "Orange", Toast.LENGTH_LONG).show();
        break;
    }
  }
}
```

**Output**




**(ii) Spinners**

Spinner control is used to display long list of items in an activity. It displays one item at a single instance from the list and allows to select an item.

Populating the spinner control can be done in two ways. They are as follows,

1. Populating through string resources
2. Populating through ArrayAdapter.

**Populating Through String Resources**

This can be done by defining two resources, one is "string resource" to show the prompt in the spinner control and another is "string array" to show the list of options to the user.

Initially, a string resource is defined and then string array is defined for which a string resource is required which is shown as follows,

**District_List.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="application">SAppExample</string>
<string name="app_settings">Settings</string>
<string name="District">Select Your District</string>
<string-array name="district_List">
  <item>Hyderabad</item>
  <item>Nizamabad</item>
  <item>Warangal</item>
  <item>Adilabad</item>
  <item>Karimnagar</item>
</string-array>
</resources>
```

In above code, a string array "district_List" is defined with five items which are displayed as options in the spinner control. Now it is required to define a layout file to display the spinner control in the application which is shown as follows,

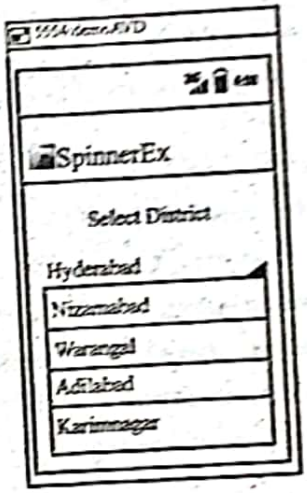
**spinnerapp.xml**

```
<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:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/District" />
    <Spinner
    android:id="@+id/s1"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
      android:entries="@array/district_list"/>
</LinearLayout>
```

**SpinnerEx.java**

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.Spinner;
public class SpinnerEx extends Activity
{
  Spinner s;
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.spinnerapp);
     s = (Spinner)findViewById(R.id.s1);
   }
   @Override
   public boolean onCreateOptionsMenu(Menu m)
   {
     getMenuInflater().inflate(R.m.spinnerex1, m);
      return true;
   }
}
```

**Output**




**2. Populating Through ArrayAdapter**

Populating spinner through ArrayAdapter can be done by removing the entries i.e., removing android : entries = "@array/district_List"/> attribute from the main XML definition. Therefore, the elements in the district_list array do not exist for longer time in the spinner control. The definition of main xml file after removing the entries is as follows,

**Program**

**spinnerapp.xml**

```
<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:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/District" />
    <Spinner
    android:id="@+id/s1"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
</LinearLayout>
```

**District_List.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="application">SAppExample</string>
  <string name="app_settings">Settings</string>
  <string name="District">Select Your District</string>
</resources>
```

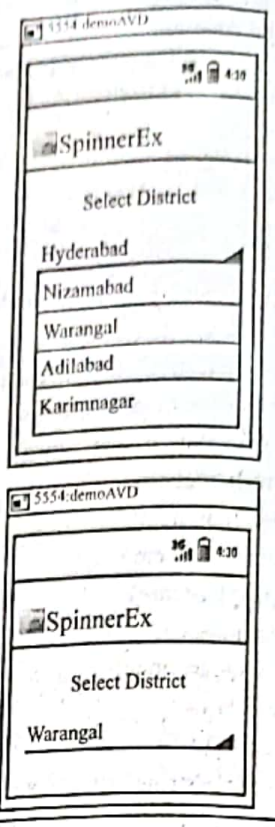
In the above District_list.xml file the string array is no longer required.

To populate the spinner through array adapter ,it is required to create array adapter in the code which is shown as follows,

**SpinnerEx.java**

```
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Spinner;
import android.widget.ArrayAdapter;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemSelectedListener;
public class SpinnerEx extends Activity
{
  Spinner s;
  @Override
  protected void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.spinnerapp);
    final String[] district_List={"Hyderabad","Nizamabad",
                    "Warangal","Adilabad","Karimnagar"}
    s = (Spinner)findViewById(R.id.s1);
    ArrayAdapter<String> ad=new ArrayAdapter<String>
        (this, android.R.layout.simple_spinner_item,
                                    district_List);
    s.setAdapter(ad)
  }
  @Override
  public boolean onCreateOptionsMenu(Menu m)
  {
    getMenuInflater().inflate(R.m.spinnerex1, m);
    return true;
  }
}
```

**Output**




## Q19. Write in short about the dialogs.

**Answer :** Model Paper-I, Q5(a)

**Dialog**

A dialog is a special purpose window which is displayed during the execution process only for a small duration of time and then disappears. When simple questions are to be asked or when certain events are to be informed to the user, dialogs are used. It is required to attach the dialog to the Frame's instance to make it automatically disappear when the frame will be hidden.

Android SDK provides various dialog window types. They are as follows,

1. **Dialog**

   It is a basic dialog class for various types of dialog windows.

2. **AlertDialog**

   It is a dialog consists of one, two or three buttons.

3. **CharacterPickerDialog**

   It is a dialog that enables user to choose a highlighted character that is connected to a regular character source.

4. **DatePickerDialog**

   It is a dialog that allows user to set/change the system date by using DatePicker control.

5. **TimePickerDialog**

   It is a dialog that enables user to set/change the system time by using TimePicker control.

6. **Progress**

It is a dialog that is used to show the progress of a task using progress dialog control.

A dialog is created by defining an instance of Dialog class. A dialog window has a feature that it can be created once and displayed many times. It is also possible update it dynamically.

**Activity Class Dialog Methods**

The various activity class dialog methods are as follows,

1. **onCreateDialog( )**

This call back method executes when a dialog is created for the first time and returns a dialog of the given type. This method can be called only once.

2. **onPrepareDialog( )**

This callback method is used to update the specified dialog.

3. **showDialog( )**

This method is used to display a dialog if it is already exists otherwise it creates and displays. Every dialog consists of a special dialog identifier which can be passed as a parameter to this method.

4. **dismissDialog( )**

This method is used to close the dialog of specified dialog identifier which is passed to method. It cannot be dismissed permanently. So, user can redisplay the dialog using showDialog( ) method.

5. **removeDialog( )**

This method is used to remove a dialog permanently even from the cache.

## Q20. Explain about AlertDialog.

**Answer :**

**AlertDialog**

AlertDialog method provides the feedback collected from the users. This dialog will remain same until and unless it is closed by a user. It is used to display some important messages that requires a fast response or to get essential feedback before proceeding further.

Generally, AlertDialog can be created by using a static inner class AlertDialog.Builder which provides many series of methods to configure an AlertDialog.

**Example**

AlertDialog.BuilderalertDialog = new AlertDialog. Builder(this);

AlertDialog.Builder subclass provides various methods to configure the AlertDialog box. They are as follows,

1. **setMessage( )**

It is used to display a message in the dialog box.

2. **setTitle( ) and setIcon( )**

It is used to set a Title and icon to be displayed on the top of the dialog box.

3. **setPositiveButton( )**

It is used to configure the positive button that represents OK button.

4. **setNegativeButton( )**

It is used to configure the negative button that represents Cancel button.

5. **setNeutralButton( )**

It is used to configure the neutral button that represents another button excluding OK, cancel.

**Example**

**ad_ex.xml**

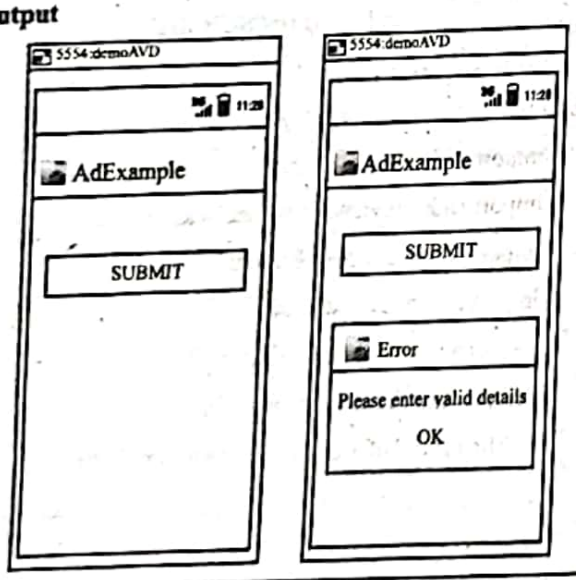
```
<LinearLayout xmlns : android = "http:
        //schemas.android.com/apk/res/android"
    xmlns : tools = "http://schemas.android.com/tools"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <Button>
            android:layout_width="fill_parent"
            android:layout_heigth="wrap_content"
            android:id= "@+id/click"
            android:text="SUBMIT" />
</LinearLayout>
```

**AdEx.java**

```
import android.os.Bundle;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.view.View;
import android.app.AlertDialog;
import android.content.DialogInterface;
public class AdEx extends Activity implements
                                    OnClickListener
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.ad_ex);
        Button btn = (Button) this.findViewById
                                    (R.id.click);
        btn.setOnClickListener(this);
    }
```

```
@Override
public void onClick(View vi)
{
    AlertDialog.Builder alertDialog = new
                        AlertDialog.Builder(this);
    alertDialog.setTitle("Error");
    alertDialog.setIcon(R.drawable.ic_launcher);
    alertDialog.setMessage("Plaase enter valid
                                        details");
    alertDialog.setPositiveButton("OK",
       new DialogInterface.OnClickListener( ))
    {
       public void OnClick(DialogInterface d,
                                        intb_Id)
       {
          return;
       }
    });
    alertDialog.show( );
  }
}
```

**Output**




**Q21. Explain how a AlertDialog used to get the input from user.**

**Answer :**

The AlertDialog can be used to get the input from the user by doing the following modifications to the application,

- Create an EditText control dynamically and make it as part of the AlertDialog to request the user for input.
- Define a TextView control in .xml layout file to show the input given by user in the AlertDialog.

**Example**

**ad_ex.xml**

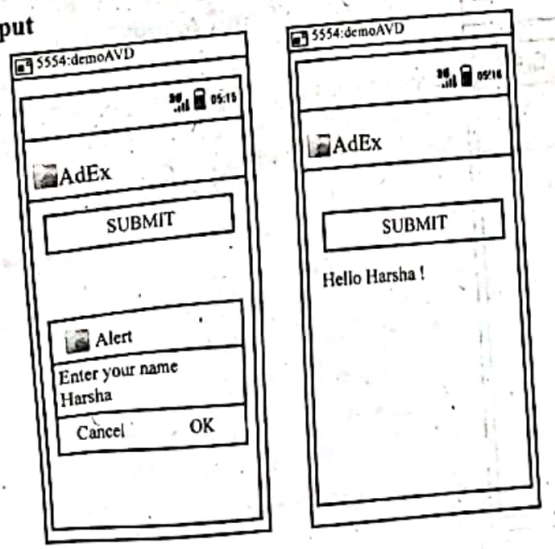
```
<LinearLayout xmlns : android = "http://schemas.
android.com/apk/res/android"
    xmlns : tools = "http://schemas.android.com.tools"
    android : orientation = "vertical"
    android : layout_width = "fill_parent"
    android : layout_height = "fill_parent">
    <Button
        android : layout_width = "fill_parent"
        android : layout_height = "wrap_content"
        android : id = "@+id/click"
        android : text = "SUBMIT"/>
    <TextView
        android : layout _width = "fill_parent"
        android : layout_height = "wrap_content"
        android : id = "@+id/rep"/>
</LinearLayout>
```

**AdEx.java**

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.view.View;
import android.content.DialogInterface;
public class AdEx extends Activity implements
                                    OnClickListener
{
    TextView rep;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.OnCreate(savedInstanceState);
        setContentView(R.layout.ad_ex);
        r = (TextView)this.findViewById(R.id.rep);
        Button btn = (Button)this.findViewById
                                    (R.id.click);
        btn.setOnClick.Listener(this);
    };
```

```
    @Override
    public void onClick(View vi)
    {
        AlertDialog.Builder alertDialog = new
                        AlertDialog.Builder(this);
        alertDialog.setTitle("Details");
        alertDialog.setIcon(R.drawable.ic_launcher);
        alertDialog.setMessage("Enter your name");
        final EditText u_name = new EditText(this);
        alertDialog.setView(u_name);
        alertDialog.setPositiveButton("OK",
            new DialogInterface.OnClickListener( ))
        {
            public void onClick(DialogInterface
                                dialog, int b_Id)
            {
                String s = name.getText( ).toString( );
                r.setText("Hello" + "s" + "!");
                return;
                }
        });
        alertDialog.setNegativeButton("Cancel",
            new DialogInterface.OnClickListener( )
        {
            public void onClick(DialogInterface d,
                                int b_Id)
            {
                return;
            }
        });
        alertDialog.show( );
    }
}
```

**Output**




**Q22. Explain how system date can be changed and set through DatePickerDialog.**

**Answer :**

DatePickerDialog allows the user to set,view and change the system date. The values of year, month and date are passed to the constructor inorder to display the initial date through this dialog. A 'calendar' instance is used to initialize the values. The constructor consists of a callback listener that provides information when the date has been changed.

**Example**

**main.xml**

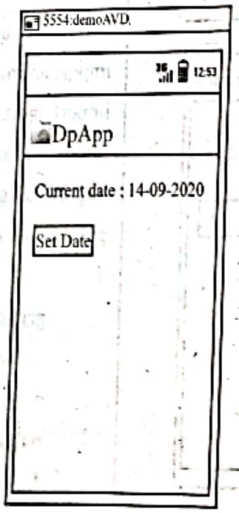
```
<LinearLayout xmlns: android = "http:
            //schemas.android.com/apk/res/android"
xmlns: android = "http://schemas.android.com/tools"
        android : orientation = "horizontal"
        android : layout_width = "fill_parent"
        android : layout_height = "fill_parent">
    <TextView
        android : id = "@+id/view_date"
        android : layout_width = "wrap_content"
        android : layout_height = "wrap_content"/>
    <Button
        android.id = "@+id/button_d"
        android : layout_width = "wrap_content"
        android : layout_height = "wrap_content"
        android : text = "Set Date"/>
</LinearLayout>
```

**DpApp.java**

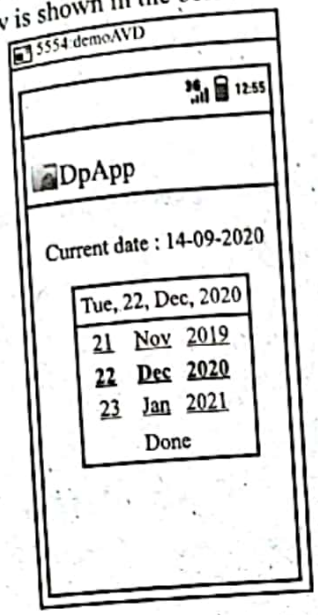
```
import android.app.Activity;
import android.os.Bundle;
import android.TextView;
import android.widget.Button;
import java.util.Calendar;
import android.app.DatePickerDialog;
import android.view.View.OnClickListener;
import android.viewView;
import android.widget.DataPicker;
public class DpApp extends Activity
{
    private TextView show_Date;
    private int year, month, day;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
```

```
        super.onCreate(SavedInstanceState);
        setContentView(R.layout.main);
        show_Date = (TextView) findViewById(R.id.view_date);
        Button d_Button = (Button) findViewById(R.id.button_d);
        final Calendar cal = Calendar.getInstance( );
        year = cal.get(Calendar.YEAR);
        month = cal.get(Calendar.MONTH);
        day = cal.get(Calendar.DAY_OF_MONTH);
        show_Date.setText("Current date:"+(day) + "-" + (month + 1) + "-" + year);
        d_Button.setOnClickListener(new OnClickListener( )
        {
            public void onClick(View v)
            {
                new DatePickerDialog(DpApp.this, dateListener, year, month, day).show( );
            }
        });
    }
    private DatePickerDialog.OnDateSetListener dl = new DatePickerDialog.OnDateSetListener( )
    {
        public void onDateSet(DatePicker view, int y, int, int)
        {
            year = y;
            month = m;
            day = d;
            show_Date.setText("Current date :"+ day + "-" + (month + 1) + "-" + year);
        }
    };
}
```

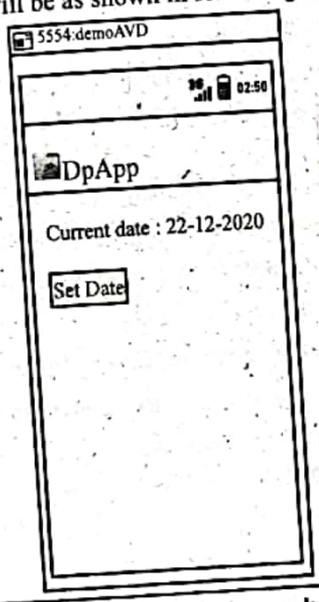
When the above application begins, the initial view will be as shown in below figure,



When the 'Set Date' button is clicked, it allows to modify the value of day, month, year. At the time of setting the date, the view is shown in the before as follows,




After changing the date, click Done button then the android output will be as shown in following figure,




**Q23. Explain how system time can be changed and set through a TimePickerDialog.**

**Answer :**

TimePickerDialog control allows user to set or select time via Timepickerview that allows user to perform a task of setting the time of the day. The format of the time can be either 24 hours mode or AM/PM mode.

**Example**

**main.xml**

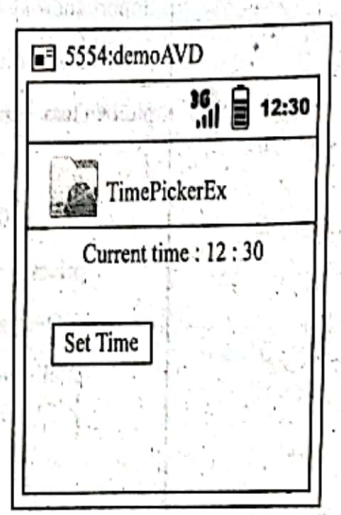
```
<LinearLayout xmlns : android = "http:
        //schemas.android.com/apk/result/android"
xmls : tools = "http: //schemas.android.com/tools"
android : layout_height = "fill_parent"
android : layout_width = "fill_parent"
android : orientation = "vertical">
<TimePicker android: id = "@ +id/tp"
    android : layout_height = "wrap_content"
    android : layout_width = "wrap_content"/>
<Button android : id = "@ + id/bs"
    android : layout_height = "wrap_content"
    android : layout_width = "wrap_content"
    android : text = "Set Time"/>
</LinearLayout>
```

**TimePickerEx.java**

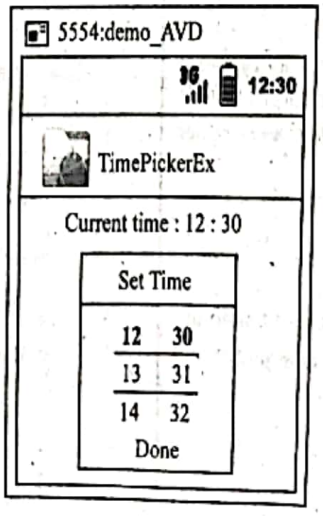
```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Button;
import java.util.Calendar;
import android.app.TimePickerDialog;
import android.view.View.OnClickListener;
import android.view.View;
import android.widget.TimePicker;
public class TimePickerEx extends Activity
{
    private TextView show_Time;
    private int h1, m1;
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        show_Time = (TextView)findViewById
                                (R.id.view_time);
        Button t_button = (Button)findViewById
                                (R.id.button_t);
        final Calendar c = Calendar.getInstance();
        h1 = c.get(Calendar.HOUR_OF_DAY);
        m1 = c.get(Calendar.MINUTE);
        show_Time.setText("Current time : "+h1+":"+m1);
        timeButton.setOnClickListener
                            (new OnClickListener())
        {
            public void onClick(View v)
```

```
            {
                new TimePickerDialog(TimePickerEx.this, timeListener,h1,m1,true).show();
            }
        });
    }
    private TimePickerDialog.OnTimeSetListener timeListener = new TimePickerDialog.OnTimeSetListener()
    {
        public void onTimeSet(TimePicker view, int hour, int minute)
        {
            h1 =hour;
            m1 = minute;
            show_Time.setText("Current time : "+h1+":"+m1);
        }
    };
}
```

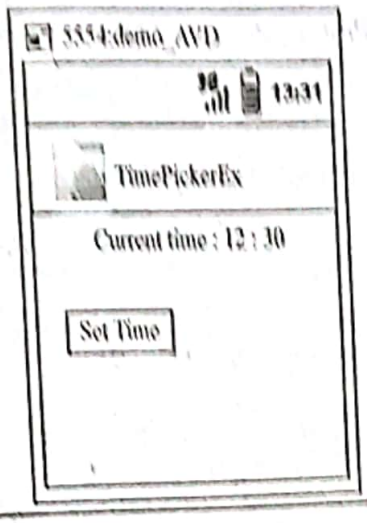
**Output**




After clicking on Set Time Button, the user is allowed to modify the time as follows,



After changing the time the android output is as follows,




## 2.4 EVENT HANDLING

**Q24. Give a brief introduction on event handling. How anonymous inner class is created?** Model Paper-II, Q5(a)

**Answer :**

**Event Handling**

In general, events are the responses made by the browser on account of user's interaction. For example, playing audio clip as soon as the page is loaded, generation of informative text as the mouse pointer is moved through a certain region of the web page, submission of user entered data to the server upon clicking the submit button etc., are the normally observed events.

Once the event is generated, there is often requirement of code to process these events. Such mechanism is known as event handler or event handling.

For example pressing or clicking a button after entering text in editText control is referred as event. Events are handled by the listeners. Event handling is done in three ways, they are as follows,

1. By creating an anonymous inner class
2. By implementing the onClickListener interface
3. By declaring event handler in XML control definition.

**Creation of an Anonymous Inner Class**

In this method a listener is implemented inline. An anonymous class is created by defining with OnClickListener interface and OnClick( ) method. The setOnClickListener( ) method is used to pass the anonymous inner class to the listener.

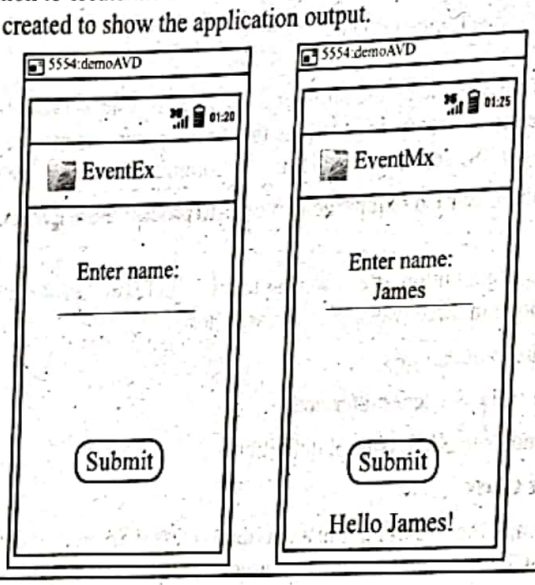
**Example**

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.EditText;
import android.widget.Button;
import android.view.View;
public class EventEx extends Activity
{
    @Override
    public void onCreate (Bundle saveInstanceState)
    {
        super.onCreate (savedInstanceState);
        setContentView(R.layout.event_msg);
        Button btn=(Button)this.findViewByld(R.id.btn_e1);     //anonymous class is associated with obj 'btn'
        btn.setOnClickListener(new Button.OnClickListener( )
```

```
        {
            public void onClick (View vi)          //invoking onClick( ) method
            {
                TextView r = (TextView) findViewById (R.id.e_resp);
                EditText n = (EditText) findViewById (R.id.u_name);
                String s = "Hello" + n.getText( ).toString( )+ "!";
                r.setText(s);
            }
        });
    }
}
```

**Running the Application**

The Eclipse Launch application must be created to run this application. Otherwise the Run As dialog box is shown on the screen. Select Android Application to create the launch configuration. The application is compiled by ADT and deployed to emulator. The Android emulator is created to show the application output.




**Q25. Write an activity to implement OnClickListener Interface.**

**Answer :**

The activity class implements the OnClickListener Interface in the concept of event handling. In addition to this, it is also required to implement the OnClick method in the class. A reference to the class is passed to set the listener. The implementation of onClickListener interface in activity is shown in below example.

```
btn.setOnClickListener(this);
```

**Example**

```
import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.EditText;
import android.widget.Button;
import android.view.View;
import android.view.View.OnClickListener;
public class EventEx extends Activity implements OnClickListener
```

```
{
    @Override
    public void OnCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.event_msg1);
        Button btn = (Button) this.findViewById (R.id.btn_e1);
        btn.setOnClickListener(this);
    }
    public void onClick (View vi)
    {
        TextView r = (TextView) this.findViewById
            (R.id.e_resp);
        EditText n = (EditText) this.findViewById
            (R.id.u_name);
        String s = "Hello"+n.getText( ).toString( ) + "!";
        r.setText(s);
    }
}
```

**Q26. Explain how to declare an event handler in XML control definition with example.**

**Answer :**

**EventEx.java**

In this event handling method, an event handler needs to be defined in XML control definition. It can be done by adding android:onClick attribute in Button definition of layout file. It is used to represent the method that is to be executed when a click event occurs.

**Example**

**event_msg.xml**

```
<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="wrap_content"
    android:text="Enter name:"/>
<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/u_name"/>
<Button
    android:id="@+id/btn_e1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Submit"
    android:onClick="show_Message"/>
```

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/r"/>
</LinearLayout>
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.EditText;
public class EventEx extends Activity
@Override
{
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setcontentview(R.layout.event_msg);
    }
    public void show_Message(View vi)
    {
        TextView r = (TextView) findViewById (R.id.e_resp);
        EditText n = (EditText) findViewById (R.id.u_name);
        String s = "Hello" +n.getText( ).toString( ) + "!";
        r.setText(s);
    }
}
```

## 2.4.1 Handling Clicks or Changes of Various UI Components

**Q27. How clicks or changes of various UI components are handled?**

**Answer :**

Model Paper-III, Q5(a)

A view becomes interactive when it responds to user-initiated events like key presses, screen touches and button clicks. The Android provides various event handlers which are used to respond to the input.

1. **onKeyDown**

   It is called when a device key is pressed. It includes D-pad, keyboard, hang-up, call, back and camera buttons.

2. **onKeyUp**

   It is called when a pressed key is released by user.

3. **onTrackballEvent**

   It is called when a device trackball is moved.

4. **onTouchEvent**

   It is called when the user presses or releases the touchscreen or when a movement is detected.

The below code depicts handling of clicks or changes to UI components.

```
@Override
public boolean onKeyDown(int kc, KeyEvent ke)
{
    retrun true;
}
@Override
public boolean onKeyUp(int kc, KeyEvent ke)
{
    return true;
}
@Override
public boolean onTrackballEvent(MotionEvent e)
{
    int actionPerformed = e.getAction( );
    return true;
}
@Override
public boolean onTouchEvent(MotionEvent e)
{
    int actionPerformed = e.getAction( );
    return true;
}
```

## 2.5 FRAGMENTS

### 2.5.1 Creating Fragments, Lifecycle of Fragments, Fragment States

**Q28. What is a fragment? Explain briefly about creating a fragment.** Model Paper-I, Q5(b)

**Answer :**

**Fragment**

Fragments can be defined as the mini-activities that are present within an existing activity. They contain their own collection of views. It can be considered as an alternative form of activity.

**Structure of a Fragment**

A fragment is the combination of both the layout and an activity. It consists of a set of views that create an independent and atomic user interface. For example, consider an activity with one or more fragments to cover the remaining space that is visible when a user is switching from the portrait to landscape. In the same way, the fragments manages the different views depending on the screen size.

For example, consider that there are two fragments $f_1$ and $f_2$ with different set of views. Suppose, if the size of a screen is small then a user can create two activities each with a single fragment, displaying them simultaneously. If the targeted screen device is small to manage two fragments, then these can be embedded into a single activity to cover the complete screen.

**Creating a Fragment**

A new fragment can be created by extending the fragment class. It is usually created by defining a UI. However, it can also be created without defining UI. It helps in background behaviour of the Activity.

The onCreateView handler is overridden if the fragment requires a UI. It helps in inflating and returning the view hierarchy. The below code snippet illustrates this statement.

```
package com.paad.fragments
    import android.app.Fragment;
    import android.os.Bundle;
    import android.view.LayoutInflater;
    import android.view.View;
    import android.view.ViewGroup;
    public class FirstFragment extends Fragment
    {
        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle SavedInstanceState)
        {
            return inflater.inflate(R.Layout.Frag_Sample, container, false);
        }
    }
```

Layout view groups are used to create (or) design the layouts in fragment code. As the fragment lifecycle is dependent on the corresponding activity lifecycle, it is not required to register in the manifest.

**Example**

**F1.xml**

```
<?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 : layout_width = "fill_parent"
    android : layout_height = "fill_parent"
    android : orientation = "#0000FF">
    <ListView
        android : id ="@+id/mobiles_List"
        android : layout_width = "fill_parent"
        android : layout_height = "fill_parent"
        android : drawSelectorOnTop = "false"
    </LinearLayout>
```

**F2.xml**

```
<?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">
    <TextView
    android : id = "@+id/selectedOpt"
    android : layout_width = "fill_parent"
    android : layout_height= "wrap_content"
    android : text = "Select a mobile"/>
</LinearLayout>
```

**F1Activity.java**

```
package com.androidunleashed.fragmentsapp;
import android.app.Fragment;
import android.os.Bundle;
import android.view.ViewGroup;
import android.view.View;
import android.view.LayoutInflater;
import android.widget.ListView;
import android.widget.ArrayAdapter;
import android.content.Context;
import android.widget.AdaterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.TextView;
public class F1Activity extends Fragment
{
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
{
Context cntxt = getActivity( ).getApplicationContext( );
View v = inflater.inflate(R.layout.f1, container,false)
String[ ] mobiles = ("Redmi", "Oppo", "Vivo", "iPhone", "Nokia");
ListView mobilesList = (ListView) v.findViewById(R.id.mobiles_list);
ArrayAdapter<String> a = new ArrayAdapter<String> (cntxt, android.R.layout.list_items, mobiles);
mobilesList.setAdapter(arrayAdpt);
fruitsList.setOnItemClickListener(new OnItemClickListener( )
{
    @Override
    public void onItemClick(AdapterView<?> p_View v, int pos, long id)
    {
        TextView s_Opt = (TextView) getActivity( ). findViewById(R, id.s_Opt);
        s_Opt.setText("The mobile"+((TextView v). getText( ). toString( ) + " is selected");
    }
});
return v;
}
```

**F2Activity.java**

```
package com.androidunleashed.fragmentsapp;
import android.app.Fragment;
import android.os.Bundle;
import android.view.ViewGroup;
import android.view.View;
import android.view.LayoutInflater;
public classF2Activity extends Fragment
{
@Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState {
    return Inflater.inflate(R.layout. f2, container, false);
    }
}
```

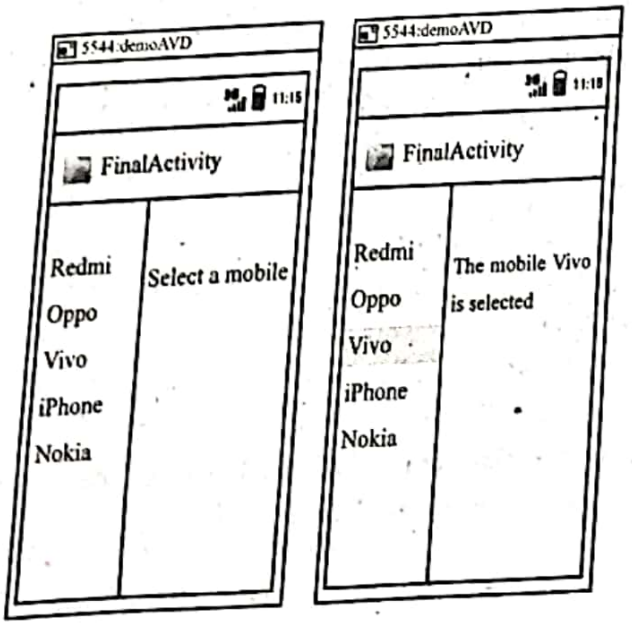
**main.xml**

```
<LinearLayout xmlns : android = "http://schemas.android.com/apk/res/android"
xmlns : tools = "http://schemas.android.com/tools"
android : layout_width = "fill_parent"
android : layout_height= "fill_parent"
android : orientation= "horizontal">
<fragment
    android : name = "com.androidunleashed.fragmentsapp.F1Activity"
    android : id = "@ + id/f1"
    android : layout_weight = "1"
    android : layout_width = "wrap_content"
    android : layout_height= "fill_parent"/>
<fragment
    android : name = "com.androidunleashed.fragmentsapp.F2Activity"
    android : id = "@ + id/f2"
    android : layout_weight = "0"
    android : layout_width = "wrap_content"
    android : layout_height= "fill_parent"/>
</LinearLayout>
```

**FinalActivity.java**

```
package com.androidunleashed.fragmentsapp;
import android.app.Activity;
import android.os.Bundle;
public class FinalActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main);
    }
}
```

**Output**



Q29. Explain in brief about the Lifecycle of fragment.

**Answer :**

**Life Cycle of a Fragment**

A fragment has its own life cycle like an activity. The lifecycle of fragment is dependent on activity life cycle but when a fragment is destroyed, its instance will be saved so that upon recreation its previous state can be restored.

The various callback methods in the fragment life cycle are as follows,

1. **onAttach( )**
   This method is called when a fragment is attached to an activity.
2. **onStart( )**
   This method is called when a fragment is visible to the user.
3. **onPause( )**
   This method is called when a fragment is visible to the user but does not have focus.
4. **onResume( )**
   This method is called when a fragment is visible to the user and in a running stage.
5. **onStop( )**
   This method is called when a fragment is invisible to the user.
6. **onCreate( )**
   This method is called to create a fragment.
7. **onCreateView( )**
   This method is called to create a view for a fragment.
8. **onActivityCreated( )**
   This method is called when the onCreate( ) method of an activity is returned.
9. **onDestroy( )**
   This method is called when a fragment is not in use for long time.
10. **onDestroyView( )**
    This method is called when a view for a fragment is destroyed or it is about to be saved.
11. **onDetach( )**
    This method is called when a fragment is detached from an activity.

**Program**

```
package com.paad.fragments;
import android.app.Activity;
import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
public class Flifecycle extends Fragment
{
    @Override
    public void onAttach(Activity a)
    {
        super.onAttach(a);
    }
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
    }
    @Override
    public View onCreateView(LayoutInflater inflater,ViewGroup container,Bundle savedInstanceState)
```

```
    {
        return inflater.inflate(R.layout.f1, container, false);
    }
    @Override
    public void onActivityCreated(Bundle savedInstanceState)
    {
        super.onActivityCreated(savedInstanceState);
    }
    @Override
    public void onStart()
    {
        super.onStart();
    }
    @Override
    public void onResume()
    {
        super.onResume();
    }
    @Override
    public void onPause()
    {
        super.onPause();
    }
    @Override
    public void onSaveInstanceState(Bundle savedInstanceState)
    {
        super.onSaveInstanceState(savedInstanceState);
    }
    @Override
    public void onStop()
    {
        super.onStop();
    }
    @Override
    public void onDestroyView()
    {
        super.onDestroyView();
    }
    @Override
    public void onDestroy()
    {
        super.onDestroy();
    }
    @Override
    public void onDetach()
    {
        super.onDetach();
    }
}
```

**Q30. Discuss in brief about fragment states.**

**Answer :**

The fragment fate is always confined to the Activity to which it belongs. So, the fragment state transitions are related to the respective Activity states. The various states of fragments are as follows,

(i) **Active**

The fragments are said to be active when their respective activity is focused and active.

(ii) **Paused and Stopped**

The fragments are said to be pause and stop when their respective activity is paused and stopped.

(iii) **Destroyed**

The fragments are said to be destroyed when their respective activity is in inactive state (or) destroyed.

In Android, if memory manager closes all the applications in order to free resources, then the related fragments in the activities are also destroyed.

The designing of UI using fragments provides flexibility of adding or deleting fragments from an active Activity dynamically. By implementing this, each and every fragment of its parent activity can make its growth multiple times through its entire lifecycle. However, managing the state transitions of fragments is important inorder to provide ideal user experience. The fragment which is moving from pause, stop (or) inactive state to active state must not differ and restore the saved state. So, it is required to store all the states of UI and persist the data if a fragment enters into pause (or) stop states.

If fragment transitions through its lifecycle, management of its state transitions becomes difficult in terms of assuring experience of user. The fragment moving from paused, stopped or inactive state back to active must not differ. So, it is significant to save UI state as well as persist data when fragment is stopped or paused when it becomes active, it needs to restore the saved state.

## 2.5.2 Adding Fragments to Activity, Adding, Removing and Replacing Fragments with Fragment Transactions

**Q31. How fragments are added, removed and replaced using Fragment Transactions? Explain.**

*Model Paper-II, Q5(b)*

**Answer :**

Fragment transactions are used for adding, removing and replacing the fragments within an Activity dynamically. These transactions help to design dynamic layouts which can be changed (or) modified based on the criteria i.e., user interactions and application state. The combination of any supported actions such as add, remove and replace can be included in fragment transaction. Moreover, the specifications such as displaying transaction animations, adding the transaction to back stack are also supported by fragment transactions.

A fragment transaction can be created by using beginTransaction( ) method belonging to the fragment manager of Activity. If the layout is modified as per the requirements and is ready to execute, then the commit( ) method is called inorder to add the transaction in the UI queue.

```
FragmentTransaction ft = fragmentManager.beginTransaction( );
    //Specifications
    ft.commit( );
```

A new UI fragment can be added by specifying the fragment instance and container view in which the fragment need to be placed.

Optionally, a tag can be specified to find fragment by using findFragmentByTag method,

```
FragmentTansaction ft = fragmentManager.beginTransaction( );
ft.add(R.id.ui_container, new My_Fragment( ));
ft.commit( );
```

A fragment can be removed by finding a reference of it initially. It can be done by using findFragmentById or findFragmentByTag methods of FragmentManager. Then resultant fragment instance is passed as a parameter to remove( ) method of fragment transaction as follows.

```
FragmentTansaction ft = fragmentManager.beginTransaction( );
Fragment f = fragmentManager.findFragmentById(R.id.fragmentinfo);
ft.remove(f);
ft.commit( );
```

A fragment can be replaced with another fragment by using replace( ) method. It is required to specify the container ID of fragment which is to be replaced and new fragment that replaces the old fragment.

```
FragmentTansaction ft = frgamentManager.beginTransaction( );
ft.replace(R.id.fragmentinfo, new Into_F(S_index));
ft.commit( );
```

## 2.5.3 Interfacing between Fragments and Activities, Multi-screen Activities

**Q32. Explain how interfacing is done between fragments and activities.**

**Answer :**

Model Paper-III, Q5(b)

The getActivity method( ) is used in fragment to return the reference to its respective activity. It is mainly used to find the present context accessing the other fragments with fragment manager and finding views in the hierarchy of activity view.

```
TextView tv = (TextView) getActivity( ).findViewById(R.id.tv);
```

It would be better to use activity as intermediary, eventhough the fragments may interact directly through host activity's fragment manager. With this, the fragment becomes independent and loosely coupled along with the capability of deciding what effect an event of fragment must have on UI falling to host activity. It is again better to create a callback interface in fragment which host activity implements where there is need for fragment to share events with host activity. The below code depicts the code from fragment class which defines public event listener interface. To implement the required interface and to retrieve a reference of host activity, the onAttach handler is overridden.

```
public interface OnSeanSelectedListener
{
    public void OnSeasonSelected(Season s);
}
private OnSeasonSelectedListener ssl;
private Season cs;
public void onAttach(Activity a)
{
    super.onAttach(a);
    try
    {
        ssl = (OnSeasonSelectedListener) a;
    }
    catch(classCastException e)
    {
        throw new ClassCastException(a.toString( ) + "implement OnSeasonSelectedListener");
    }
}
private void setSeason(Season s)
{
    cs = s;
    ossl.ossl(s);
}
```

**Q33. Write in detail about multiscreen activities.**

**Answer :**

Android supports displaying or performing multiple activities at the same time. In handheld devices, multiple applications can be run side by side or one above other in multi screen mode or split screen mode. In android, it is possible to configure the application to handle multi window display. For this the activities minimum dimensions must be specified. It is even possible to enable or disable the multi window display for the application.

User can split the screen for example by viewing the web page on right side and composing an email on left side. The dividing line separating the two applications can be dragged to make one application look large or small than other.

The user is allowed to switch to multi screen mode by following ways,

1. User is allowed to drag the activity to highlighted part of screen to place the activity in multiwindow mode by opening overview screen and then performing long press on activity title.
2. User can place the activity in multiscreen mode and can even open the overview screen to select some other activity to share the screen by long pressing the overview button.

The multiscreen mode does not affect the activity lifecycle. In this mode, recent user interacted activity will be in active state. This activity is considered as top most one and will be in RESUMED state whereas, the other visible activities remain in STARTED state. The system considers the visible but not resumed activities more than the non visible activities. So, if user interacts with any of the STARTED activities then that activity enters into RESUMED state and previous activity enters into STARTED state. If an application is placed in multi-windows mode, then the system will notify the activity about changes in configuration as specified in handling configuration changes. This situation also occurs when the application is resized or placed back into full screen mode.

The activity can handle the configuration change by itself when user resizes the screen and changes the dimensions then system will resize the activity inorder to match with user action and then it issues configuration changes as required.

To make activities support multiple screen displays, the attributes of manifest can be set to control the size as well as layout. The attribute android : resizableActivity in manifest or element can be set to enable or disable the multiscreen display.

android : resizableActivity = ["true" | "false"]

If it is set to true the activity well be launched in split screen and free form modes otherwise it does not support multiscreen mode.

The below code depicts how activity's default size and location and its minimum size can be specified when it is displayed in free form model.

```
<activity android : name = " .Activity1">
<layout   android : defaultHeight = "600dp"
          android : defaultwidth = "300dp"
          android : gravity = "toplend"
          android : minHeight = "350dp"
          android : minWidth = "200dp"/>
</activity>
```

The activity provides some methods to support multiscreen display, which are as follows,

1. **isInMultiWindowMode( )**

   It is used to find out whether the activity is in multi screen mode.

2. **isInPicfureInPictureMode( )**

   It is used to find out whether the activity is in picture in picture mode.

3. **onPictureInPictureModeChanged( )**

   It is used when activity enters in or out of picture-in-picture mode. The true value is passed to method if the activity enters into picture-in-picture mode. Otherwise, false is passed.

4. **onMultiWindowModeChanged( )**

   It is used when activity enters in or out of multiscreen mode .The true value is passed to method if activity enters into multi screen mode. Otherwise, false is passed.