Lab 2
User Interface Layout

1. Linear Layout

Linear layout is a simple layout used in android for layout designing. In the Linear layout all the elements are displayed in linear fashion means all the childs/elements of a linear layout are displayed according to its orientation. The value for orientation property can be either horizontal or vertical.

Types Of Linear Layout Orientation
There are two types of linear layout orientation:

1. Vertical
2. Horizontal

As the name specified these two orientations are used to arrange there child one after the other, in a line, either vertically or horizontally. Let’s we describe these in detail.

1.Vertical:

In this all the child are arranged vertically in a line one after the other. In below code snippets we have specified orientation “vertical” so the childs/views of this layout are displayed vertically.

2. Horizontal:

In this all the child are arranged horizontally in a line one after the other. In below code snippets we have specified orientation “horizontal” so the childs/views of this layout are displayed horizontally.

Main Attributes In Linear Layout:

Now let’s we discuss about the attributes that helps us to configure a linear layout and its child controls. Some of the most important attributes you will use with linear layout include:

1. orientation: The orientation attribute used to set the childs/views horizontally or vertically. In Linear layout default orientation is vertical.

Example: Orientation vertical:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"> <!-- Vertical Orientation set -->

    <!-- Put Child Views like Button here -->
</LinearLayout>
```

Example: Orientation Horizontal:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"> <!-- Horizontal Orientation set -->

    <!-- Child Views are here -->

</LinearLayout>
```

3. layout_weight: The layout weight attribute specify each child control’s relative importance within the parent linear layout.

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


<!--Add Child View Here--->

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Weight 2"
        android:background="#761212"
        android:layout_margin="5dp"
        android:id="@+id/button"
        android:layout_weight="2" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#761212"
        android:layout_margin="5dp"
        android:layout_weight="1"
        android:text="Weight 1" />
</LinearLayout>
```

4. weightSum: weightSum is the sum up of all the child attributes weight. This attribute is required if we define weight property of the childs.
Example: In the same above example of weight, we can define weightSum value 3.

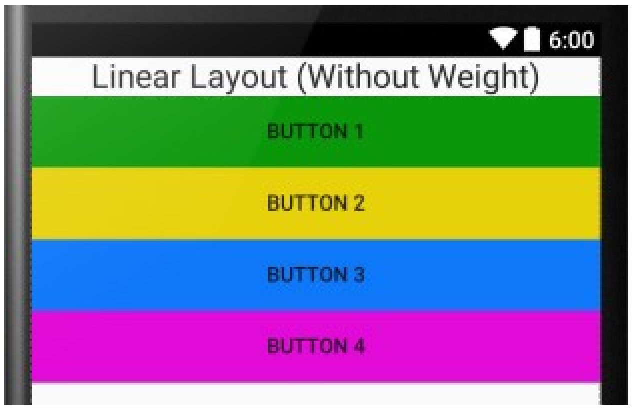
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="3"
    android:orientation="horizontal">
    <!--Add Child View Here--->

</LinearLayout>
```

Example 1: First we will design Android Linear Layout without using weight property
In this example we have used one TextView and 4 Button. The orientation is set to vertical.

Below is the code of activity_main.xml

```
<!-- Vertical Orientation is set -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <!-- Text Displayed At Top -->

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Linear Layout (Without Weight)"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal" />

    <!-- Button Used -->

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:background="#009300" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:background="#e6cf00" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:background="#0472f9" />

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 4"
        android:background="#e100d5" />
</LinearLayout>
```

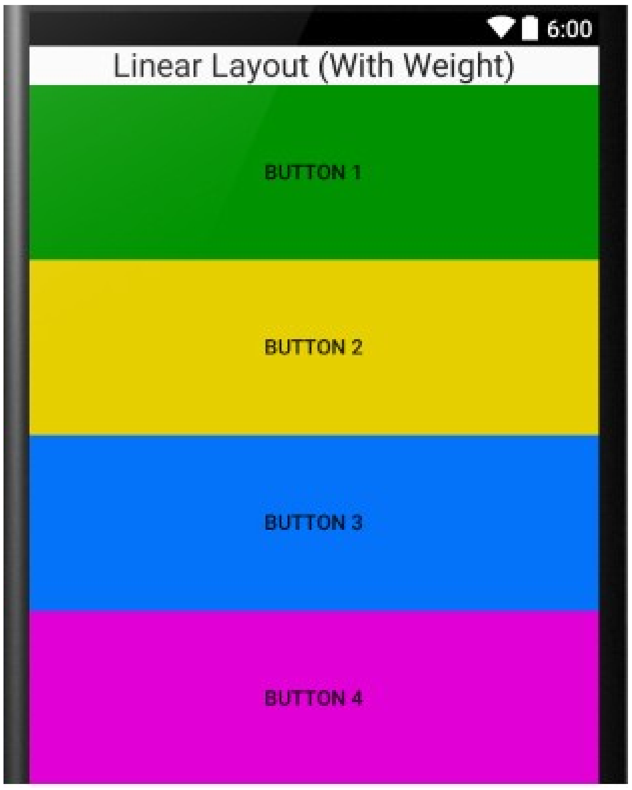
Output Screen:



Example 2: In this example of linear layout we have used weight property.
Below is the code of activity_main.xml with explanation included

```
<!-- Vertical Orientation is set  with weightSum-->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:weightSum="5"
    android:orientation="vertical">

    <!-- Text Displayed At Top -->

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Linear Layout (With Weight)"
        android:id="@+id/textView"
        android:layout_gravity="center_horizontal"
        android:layout_weight="0"/>

    <!-- Button Used -->

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 1"
        android:background="#009300"
        android:layout_weight="1"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 2"
        android:background="#e6cf00"
        android:layout_weight="1"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 3"
        android:background="#0472f9"
        android:layout_weight="1"/>

    <Button
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Button 4"
        android:background="#e100d5"
        android:layout_weight="1"/>
</LinearLayout>
```

Output Screen:



2. Relative Layout

The Relative Layout is very flexible layout used in android for custom layout designing. It gives us the flexibility to position our component/view based on the relative or sibling component's position. Just because it allows us to position the component anywhere we want so it is considered as most flexible layout. For the same reason Relative layout is the most used layout after the Linear Layout in Android. It allow its child view to position relative to each other or relative to the container or another container.

Attributes of Relative layout:

Lets see different properties of Relative Layout which will be used while designing Android App UI:

1.above: Position the bottom edge of the view above the given anchor view ID and must be a reference of the another resource in the form of id. Example, android:layout_above="@+id/textView" .

For example, suppose a view with id textview2 is what we want to place above another view with id textview. Below is the code and layout image.

```
<!-- textView2 is placed above textView-->
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Text2"
        android:id="@+id/textView2"
        android:layout_above="@+id/textView"
        android:layout_marginBottom="100dp"
        android:layout_centerHorizontal="true"/>
```

2. alignBottom: alignBottom is used to makes the bottom edge of the view match the bottom edge of the given anchor view ID and it must be a reference to another resource, in the form of id. Example: android:layout_ alignBottom ="@+id/button1″
In the below example we have aligned a view with id textView2 Bottom of another view with id textView.

```
<!-- textView2 alignBottom of textView -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_centerHorizontal="true"
    android:id="@+id/textView2"
    android:layout_alignBottom="@+id/textView"
    android:text="Text2 alignBottom of Text1"
    android:layout_marginBottom="90dp"
/>
```

3. alignLeft: alignLeft is used to make the left edge of the view match the left edge of the given anchor view ID and must be a reference to another resource, in the form of Example: android:layout_ alignLeft ="@+id/button1″.

Below is the code and layout image in which we have aligned a view with id textView2 left of another view with id textView.

```
<!-- textView2 alignLeft of textView -->
<TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView2"
    android:layout_alignLeft="@+id/textView"
    android:text="Text2 alignLeft of Text1"
    android:layout_below="@+id/textView"
    android:layout_marginTop="20dp"/>
```

4. alignRight: alignRight property is used to make the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form like this example: android:layout_alignRight="@+id/button1″

Below is the code and layout image in which we have aligned a view with id textView2 right of another view with id textView.

```
<!-- textView2 alignRight of textView-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView2"
    android:layout_alignRight="@+id/textView"
    android:text="Text2 alignRight of Text1"
    android:layout_below="@+id/textView"
    android:layout_marginTop="20dp"/>
```

5.alignStart: alignStart property is used to makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form of like this example: android:layout_alignStart="@+id/button1″

Below is the alignStart code and layout image in which we have aligned a view with id textView2 start of another view with id textView.

```
<!-- Text2 alignStart-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView2"
    android:text="Text2 align start of Text1"
    android:layout_alignStart="@+id/textView"
 />
```

6. alignTop: alignTop property is used to makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form like this example: android:layout_alignTop="@+id/button1″.

Below is the alignTop code and layout image in which we have aligned a view with id textView Top of another image with id imageView.

```
<!--text is align top on Image-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:layout_alignTop="@+id/imageView"
    android:text="Text Here is AlignTop on Image"
```

```
        />
```

7.alignParentBottom: If alignParentBottom property is true, makes the bottom edge of this view match the bottom edge of the parent. The value of align parent bottom is either true or false. Example: android:layout_alignParentBottom=”true”

Important Note:alignParentBottom and alignBottom are two different properties. In alignBottom we give the reference of another view in the form of id that the view is aligned at the bottom of referenced view but in alignParentBottom the bottom edge of the view matches the bottom edge of the parent.

Below is the alignParentBottom code and layout image in which textView is simply displayed using the alignParentBottom.

```
<!-- textView is alignParentBottom -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text Here is AlignParentBottom with bottom margin of 120dp"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="120dp" />
```

8. alignParentEnd: If alignParentEnd property is true, then it makes the end edge of this view match the end edge of the parent. The value of align parent End is either true or false. Example: android:layout_alignParentEnd=”true”.

Important Note: In alignParentEnd the bottom edge of the view matches the bottom edge of the parent.

Below is the alignParentEnd code and layout image in which textView is simply displayed on Image in the end.

```
<!-- Text displayed in the end of parent image-->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text in Parent End"
    android:layout_alignBottom="@+id/imageView"
    android:layout_alignParentEnd="true"
 />
```

9. alignParentLeft: If alignParentLeft property is true, makes the left edge of this view match the left edge of the parent. The value of align parent left is either true or false. Example: android:layout_alignParentLeft=”true”.

Important Note: alignParentLeft and alignLeft are two different properties. In alignLeft we give the reference of another view in the form of id that the view is aligned to the left of the referenced view but in alignParentLeft the left edge of the view matches the left edge of the parent.

Below is the alignParentLeft example code and layout image in which textView is simply displayed on parent Image in the left side.

```
<!-- align parent left in Android -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text in Parent Left"
    android:layout_alignBottom="@+id/imageView"
    android:layout_alignParentLeft="true"
     />
```

10. alignParentRight: If alignParentRight property is true, then it makes the right edge of this view match the right edge of the parent. The value of align parent right is either true or false. Example: android:layout_alignParentRight="true".

Important Note: alignParentRight and alignRight are two different properties. In alignRight we give the reference of another view in the form of id that the view is aligned to the right of the referenced view but in alignParentRight the right edge of the view matches the right edge of the parent.

Below is the alignParentRight example code and layout image in which textView is simply displayed on parent Image in the right side.

```
<!-- alignRightParent Example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text in Parent Right"
    android:layout_alignBottom="@+id/imageView"
    android:layout_alignParentRight="true"
  />
```

11.alignParentStart: If alignParentStart is true, then it makes the start edge of this view match the start edge of the parent. The value of align parent start is either true or false. Example: android:layout_alignParentStart="true".

Important Note: alignParentStart and alignStart are two different properties, In alignStart we give the reference of another view in the form of id that the view is aligned at the start of referenced view but in alignParentStart the start edge of the view matches the start edge of the parent(RelativeLayout).

Below is the alignParentStart example code and layout image in which textView is simply displayed on parent Image in the right side.

```
<!-- alignParentStart Example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView"
    android:text="Text in Parent Start"
    android:layout_alignTop="@+id/imageView"
    android:layout_alignParentStart="true"
     />
```

12.alignParentTop: If alignParentTop is true, then it makes the top edge of this view match the top edge of the parent. The value of align parent Top is either true or false. Example: android:layout_alignParenTop=”true”.

Important Note: alignParentTop and alignTop are two different properties, In alignTop we give the reference of another view in the form of id that the view is aligned to the top of the referenced view but in alignParentTop the top edge of the view matches the top edge of the parent(RelativeLayout).

Below is the example code of alignParentTop property and also layout image.

```
<!-- alignParentTop example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text1 align parent top"
    android:layout_alignParentTop="true"
    android:layout_margin="20dp"
    android:textSize="20sp"
    android:textColor="#000"/>
```

13.centerInParent: If center in parent is true, makes the view in the center of the screen vertically and horizontally. The value of center in parent is either true or false. Example: android:layout_centerInParent=”true”.
Below is the example code of centerInParent property and also layout image.

```
<!-- centerInParent example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text1 center in parent"
    android:layout_centerInParent="true"
    android:textSize="20sp"
    android:textColor="#000"
/>
```

14.centerHorizontal: If centerHorizontal property is true, makes the view horizontally center. The value of centerHorizontal is either true or false.Example: android:layout_centerHorizontal=”true”.
Below is the example code of centerHorizontal property and also layout image.

```
<!-- centerHorizontal example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text1 center Horizontal"
    android:layout_centerHorizontal="true"
    android:textSize="20sp"
    android:textColor="#000"
/>
```

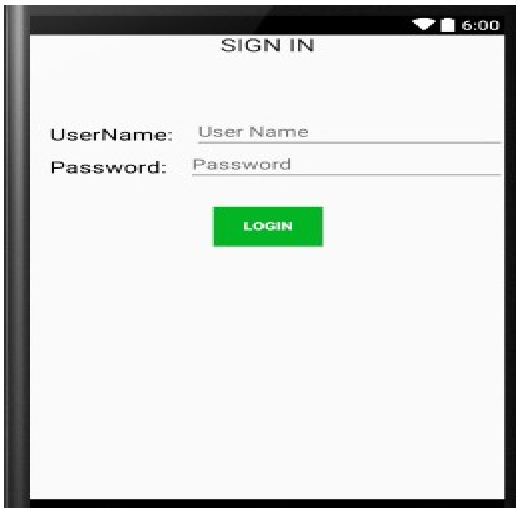
15.centerVertical: If centerVertical property is true, make the view vertically center. The value of align parent bottom is either true or false. Example: android:layout_centerVertical=”true”.

Below is the example code of centerVertical property and also layout image.

```
<!-- centerVertical example -->
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Text1 center vertical"
    android:layout_centerVertical="true"
    android:textSize="20sp"
    android:textColor="#000"
/>
```

Example 1: Here we are designing a simple log in screen in Android Studio using Relative Layout. Below is the code of activity_main.xml for designing UI with explanation included in it:

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

<!--Relative Layout Is Used-->

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

    <!--Text View for Displaying SIGN IN Text At Top of UI-->

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="SIGN IN"
        android:id="@+id/textView3"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />

    <!--Text View for Displaying Username-->

    <TextView
        android:id="@+id/userName"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="@dimen/activity_horizontal_margin"
        android:layout_marginTop="110dp"
        android:text="UserName:"
        android:textColor="#000000"
        android:textSize="20sp" />

    <!--Text View for Displaying Password-->

    <TextView
        android:id="@+id/password"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/userName"
        android:layout_margin="@dimen/activity_horizontal_margin"
        android:text="Password:"
        android:textColor="#000000"
        android:textSize="20sp" />

    <!--Edit Text for Filling Username-->

    <EditText
        android:id="@+id/edt_userName"
        android:layout_width="fill_parent"
        android:layout_height="40dp"
        android:layout_marginLeft="@dimen/activity_horizontal_margin"
        android:layout_marginTop="100dp"
        android:layout_toRightOf="@+id/userName"
        android:hint="User Name" />
    <!--Edit Text for Filling Password-->

    <EditText
        android:layout_width="fill_parent"
        android:layout_height="40dp"
        android:layout_below="@+id/edt_userName"
        android:layout_centerVertical="true"
        android:layout_toRightOf="@+id/password"
        android:hint="Password" />

    <!--Button for Clicking after filling details-->

    <Button
        android:id="@+id/btnLogin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/password"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="20dp"
        android:background="#03B424"
        android:text="Login"
        android:textColor="#ffffff"
        android:textStyle="bold" />


</RelativeLayout>
```



3. Frame Layout

Frame Layout is one of the simplest layout to organize view controls. They are designed to block an area on the screen. Frame Layout should be used to hold child view, because it can be difficult to display single views at a specific area on the screen without overlapping each other.

We can add multiple children to a FrameLayout and control their position by assigning gravity to each child, using the **android:layout_gravity** attribute.

Attributes of Frame Layout:

Lets see different properties of Frame Layout which will be used while designing Android App UI:

1. android:id

This is the unique id which identifies the layout in the R.java file.

Below is the id attribute’s example with explanation included in which we define the id for Frame Layout.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/frameLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"/>
```

2. android:foreground

Foreground defines the drawable to draw over the content and this may be a color value. Possible color values can be in the form of “#rgb”, “#argb”, “#rrggbb”, or “#aarrggbb”. This all are different color code model used.

Example: In Below example of foreground we set the green color for foreground of frameLayout so the ImageView and other child views of this layout will not be shown.

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

android:id="@+id/framelayout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_gravity="center"
android:foregroundGravity="fill"
android:foreground="#0f0"><!--foreground color for a FrameLayout-->

<LinearLayout
android:orientation="vertical"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerInParent="true"
>

<!-- Imageview will not be shown because of foreground color which is drawn over
it-->

<ImageView
android:layout_width="200dp"
android:layout_height="200dp"
android:layout_marginBottom="10dp"
android:src="@mipmap/ic_launcher"
android:scaleType="centerCrop"
/>

<!--Textview will not be shown because of foreground color is drawn over it-->

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:gravity="center_horizontal"
android:text="Android"/>

</LinearLayout>

</FrameLayout>
```

Important Note: On applying android:foregroud feature, all the views taken in the framelayout will goes to the background and the framelayout comes in the foreground.

3. android:foregroundGravity

This defines the gravity to apply to the foreground drawable. Default value of gravity is fill. We can set values in the form of “top”, ”center_vertical” , ”fill_vertical”, ”center_horizontal”, ”fill_horizontal”, ”center”, ”fill”, ”clip_vertical”, ”clip_horizontal”, ”bottom”, ”left” or ”right” .

It is used to set the gravity of foreground. We can also set multiple values by using “|”. Ex: fill_horizontal|top .Both the fill_horizontal and top gravity are set to framelayout.

In the above same example of foreground we also set the foregroundGravity of FrameLayout to fill.

4. android:visibility

This determine whether to make the view visible, invisible or gone.

visible – the view is present and also visible

invisible – The view is present but not visible

gone – The view is neither present nor visible

Example 1: Example of A Frame Layout without using layout gravity. Here we will use a image to fill complete screen and then we will write “android” text in center of the image. We will do all this inside Frame Layout.

Step 1: Create a new project and name it FrameWithoutGravity.

Select File -> New -> New Project in Android Studio. Fill the forms and click “Finish” button.

Step 2: Now open res -> layout -> activity_main.xml and add following code. Here we are designing ImageView and TextView inside Frame Layout.

Important Note: Make sure you have an image saved in drawable folder name img_name. If you have saved other image with different name then please change it in the code.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frameLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY"
        android:src="@drawable/img_name" /><!--Change image as per your name of
image saved in drawable-->
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center"
        android:text="Android"
        android:textSize="30sp"
        android:textColor="#f3f3f3"
        android:textStyle="bold" />
</FrameLayout>
```

Step 3: Let the MainActivity.java has default java code or add the below code:

```
package android.com.framewithoutgravity;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

4. Grid View

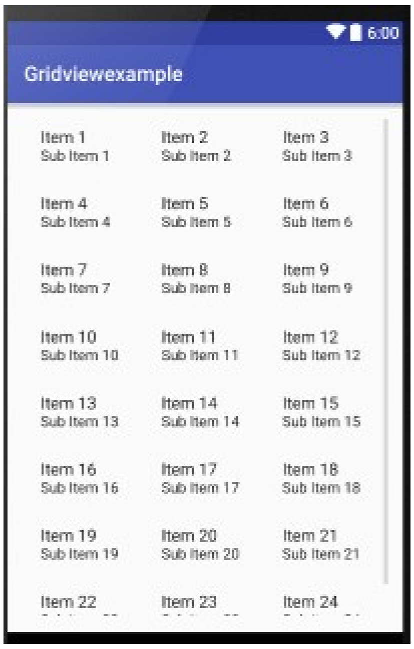
In android GridView is a view group that display items in two dimensional scrolling grid (rows and columns), the grid items are not necessarily predetermined but they are automatically inserted to the layout using a ListAdapter. Users can then select any grid item by clicking on it. GridView is default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

GridView is widely used in android applications. An example of GridView is your default Gallery, where you have number of images displayed using grid.

Adapter Is Used To Fill Data In Gridview: To fill the data in a GridView we simply use adapter and grid items are automatically inserted to a GridView using an Adapter which pulls the content from a source such as an arraylist, array or database.

Basic GridView code in XML:

```
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3"/>
```



```
Item 1        Item 2        Item 3
Sub Item 1    Sub Item 2    Sub Item 3

Item 4        Item 5        Item 6
Sub Item 4    Sub Item 5    Sub Item 6

Item 7        Item 8        Item 9
Sub Item 7    Sub Item 8    Sub Item 9

Item 10       Item 11       Item 12
Sub Item 10   Sub Item 11   Sub Item 12

Item 13       Item 14       Item 15
Sub Item 13   Sub Item 14   Sub Item 15

Item 16       Item 17       Item 18
Sub Item 16   Sub Item 17   Sub Item 18

Item 19       Item 20       Item 21
Sub Item 19   Sub Item 20   Sub Item 21

Item 22       Item 23       Item 24
```

Important Note: numColumns property has to be specified otherwise GridView behaves like a ListView with just singleChoice. In the above image `numColumns` property specified that there is 3 columns to show, if we set it to `auto_fit` then it automatically display as many column as possible to fill the available space of the screen. Even if the phone is in portrait mode or landscape mode it automatically fill the whole space.

Attributes of GridView:

1.id: id is used to uniquely identify a GridView.
Below is the id attribute's example code with explanation included in which we don't specify the number of columns in a row that's why the GridView behaves like a ListView.

Below is the id attribute example code for Gridview:

```
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
```

2.numColumns: numColumn define how many columns to show. It may be a integer value, such as "5" or `auto_fit`.

`auto_fit` is used to display as many columns as possible to fill the available space on the screen.

Important Note: If we don't specify numColumn property in GridView it behaves like a ListView with singleChoice.

Below is the numColumns example code where we define 4 columns to show in the screen.

```
<!-- numColumns example code -->
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="4"/> <!-- numColumns set to 4-->
```

3. horizontalSpacing: horizontalSpacing property is used to define the default horizontal spacing between columns. This could be in pixel(px),density pixel(dp) or scale independent pixel(sp).

Below is the horizontalSpacing example code with explanation included where horizontal spacing between grid items is 50 dp.

```
<!--Horizontal space example code in grid view-->>
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3"
android:horizontalSpacing="50dp"/><!--50dp horizontal space between grid items-->
```

4.verticalSpacing: verticalSpacing property used to define the default vertical spacing between rows. This should be in px, dp or sp.

Below is the verticalSpacing example code with explanation included, where vertical spacing between grid items is 50dp.

```
<!-- Vertical space between grid items code -->
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3"
android:verticalSpacing="50dp"/><!--50dp vertical space set between grid items-->
```

5.columnWidth: columnWidth property specifies the fixed width of each column. This could be in px, dp or sp.

Below is the columnWidth example code. Here column width is 80dp and selected item's background color is green which shows the actual width of a grid item.

Important Note: In the below code we also used listSelector property which define color for selected item. Also to see the output of columnWidth using listSelector we need to use Adapter which is our next topic. The below code is not sufficient to show you the output.

```
<!--columnWidth in Grid view code-->
<GridView
android:id="@+id/simpleGridView"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:numColumns="3"
android:columnWidth="80dp"
android:listSelector="#0f0"/><!--define green color for selected item-->
```

Example:

GridView Using Base Adapter In Android:

Base Adapter is a common base class of a general implementation of an Adapter that can be used in GridView. Whenever you need a customized grid view you create your own adapter and extend base adapter in that. Base Adapter can be extended to create a custom Adapter for displaying custom grid items.

Following is the simple example showing user details using **GridView** and showing the position of a particular image when clicking on it in android applications.

Create a new android application using android studio and give names as **GridView**.

Once we create an application, add some sample images to project **/res/drawable** directory to show the images in GridView.

Now open an **activity_main.xml** file from **/res/layout** path and write the code like as shown below

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="110dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center" />
```

Once we are done with creation of layout, we need to create a custom adapter (**ImageAdapter.java**) by extending it using **BaseExtender** to show all the items in the grid, for that right click on **java** folder → Give name as **ImageAdapter.java** and click **OK**.

Open **ImageAdapter.java** file and write the code like as shown below

ImageAdapter.java

```
import android.content.Context;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

public class ImageAdapter extends BaseAdapter {
//Context provides the connection to the Android system and the resources of

//the project.

    private Context mContext;


    public ImageAdapter(Context c) {
        mContext = c;
    }

//he getCount() function returns the total number of items to be displayed in
//list.
    public int getCount() {
        return thumbImages.length;
    }

/This function is used to Get the data item associated with the specified position
in the data set to obtain the corresponding data of the specific location in the
collection of data items. */


    public Object getItem(int position) {
        return null;
    }

/*As for the getItemId (int position), it returns the corresponding to the
position item ID. */


    public long getItemId(int position) {
        return 0;
    }
/*The adapters are built to reuse Views, when a View is scrolled so that is no
longer visible, it can be used for one of the new Views appearing. This reused
View is the convertView.

The parent is provided so you can inflate your view into that for proper layout
parameters. */
    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
            ImageView imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(200, 200));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
            imageView.setImageResource(thumbImages[position]);
            return imageView;
    }
    // Add all our images to arraylist
    public Integer[] thumbImages = {
            R.drawable.img1, R.drawable.img2,
            R.drawable.img3, R.drawable.img4,
            R.drawable.img5, R.drawable.img6,
            R.drawable.img7, R.drawable.img8,
            R.drawable.img1, R.drawable.img2,
            R.drawable.img3, R.drawable.img4,
            R.drawable.img5, R.drawable.img6,
            R.drawable.img7, R.drawable.img8,
            R.drawable.img1, R.drawable.img2,
            R.drawable.img3, R.drawable.img4,
            R.drawable.img5
    };
}
```

If you observe above code we referred some images, actually those are the sample images which we added in **/res/drawable** directory.

Now we will bind images to our **GridView** using our custom adapter (**ImageAdapter.java**), for that open main activity file **MainActivity.java** from **\java\com.tutlane.gridview** path and write the code like as shown below.

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.GridView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        GridView gv = (GridView) findViewById(R.id.gvDetails);
        gv.setAdapter(new ImageAdapter(this));


        gv.setOnItemClickListener(new AdapterView.OnItemClickListener()
        {
            public void onItemClick(AdapterView<?> parent, View v, int position, long
id) {
            Toast.makeText(MainActivity.this, "Image Position: " + position,
Toast.LENGTH_SHORT).show();
                }
            });
        }
}
```

```
If you observe above code, we are binding image details to **GridView** using our
custom adapter (**ImageAdapter.java**) and calling our layout using **setContentView**
method in the form of **R.layout.layout_file_name**. Here our xml file name is
**activity_main.xml** so we used file name **activity_main**.
```

Android GridView Details Activity Example

In above example, we implemented an image gallery using gridview in android application. Now we will extend the functionality of above example to show the selected grid image in full screen.

Now we need to create a new layout (**image_details.xml**) file in project **/res/layout** directory to show the image details, for that right click on the layouts folder → select New → Layout resource file → Give name as **image_details.xml** and click **OK**. Now open newly created file (**image_details.xml**) and write the code like as shown below.

image_details.xml

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

Now we need to create a custom activity (**FullImageActivity.java**) to show the image details in our newly created layout (**image_details.xml**) file, for that right click on java folder → select New → Java Class → Give name as **FullImageActivity.java** and click **OK**.

Open **FullImageActivity.java** file and write the code like as shown below

FullImageActivity.java

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.ImageView;


public class FullImageActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.image_details);
        // Get intent data
        Intent i = getIntent();
        // Get Selected Image Id
        int position = i.getExtras().getInt("id");
        ImageAdapter imageAdapter = new ImageAdapter(this);
        ImageView imageView = (ImageView) findViewById(R.id.full_image_view);
        imageView.setImageResource(imageAdapter.thumbImages[position]);
    }
}
```

Now we need to include our newly created activity file (**FullImageActivity.java**) in **AndroidManifest.xml** file like as shown below. For that, open **AndroidManifest.xml** file and write the code like as shown below

AndroidManifest.xml

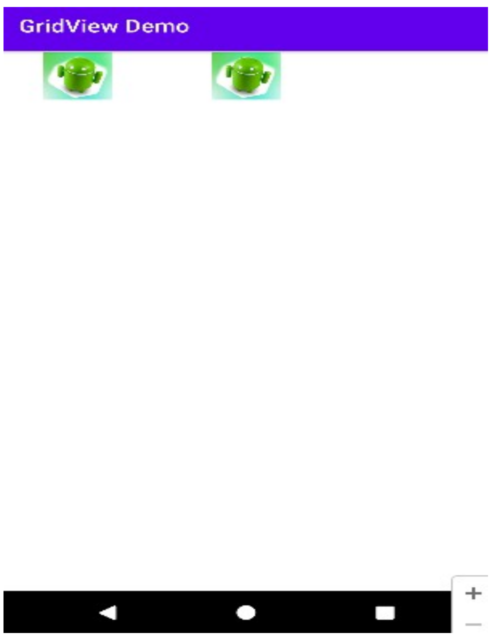
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tutlane.gridview">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!-- FullImageActivity -->
        <activity android:name=".FullImageActivity"></activity>
    </application>
</manifest>
```

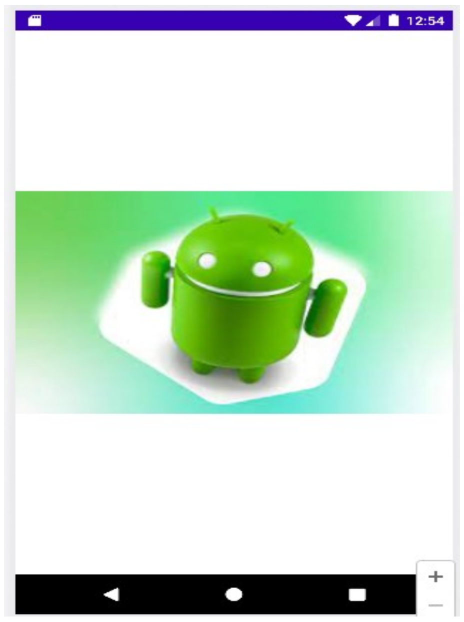
Now we need to modify gridview image click function in main activity file (**MainActivity.java**) to get image details and show it in new activity.

Open main activity file (**MainActivity.java**) and write the code like as shown below.

MainActivity.java

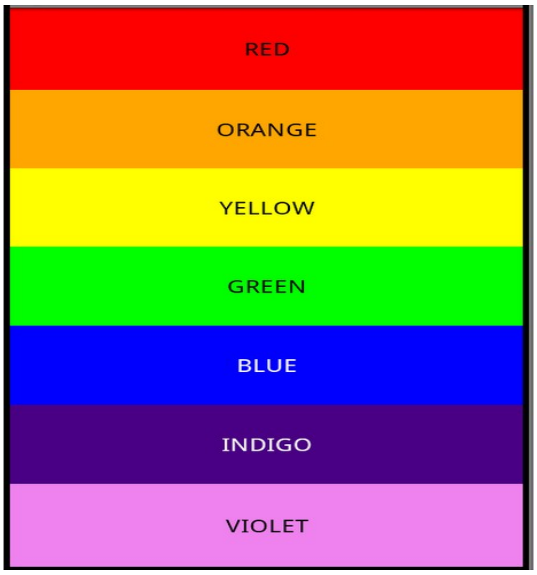
```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.GridView;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        GridView gv = (GridView) findViewById(R.id.gvDetails);
        gv.setAdapter(new ImageAdapter(this));


 gv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View v, int position,
long id) {
                // Sending image id to FullScreenActivity
          Intent i = new Intent(getApplicationContext(), FullImageActivity.class);
                // passing array index
                i.putExtra("id", position);
                startActivity(i);
            }
        });
    }
}
```





EXERCISE

1. creating a new layout resource file called rainbow.xml. Within this XML file, add a vertically-oriented linear layout control that fills the entire screen. Next, add seven TextView controls as child controls within the linear layout: one for each color in the spectrum. Set the text of each TextView control to the appropriate color string in the spectrum and the background color to the appropriate color value. Also, set each control's layout_width attribute to fill_parent and its layout_height attribute to wrap_content.



2. Create application to view the fruits images in GridView in android application. When you click on image it should display that image in fullscreen.