विशाल जाधव सरांचे

VJTech Academy

Inspiring Your Success...

UNIT-III Components and Layouts

Control Flow.

- Create first project

Android Project Folder Structure.

- Android Studio is the official IDE (Integrated Development Environment) developed by the by Google for android app development.
- After completing the setup of Android Architecture, we can create an android application in the studio.
- We need to create a new project for each sample application and we should understand the folder structure.
- The android project contains different types of app modules, source code files, and resource files. We will explore all the folders and files in the android app.

1. **Manifests Folder**
2. **Java Folder**
3. **res (Resources) Folder**
    - **Drawable Folder**
    - **Layout Folder**
    - **Mipmap Folder**
    - **Values Folder**
4. **Gradle Scripts**

Manifests Folder

- Manifests folder contains **AndroidManifest.xml** for creating our android application.
- This file contains information about our application such as the Android version, metadata, states package for Kotlin file, and other application components.
- It acts as an intermediator between android OS and our application.

Java folder

- The Java folder contains all the java and Kotlin source code (.java) files that we create during the app development, including other Test files.
- If we create any new project using Kotlin, by default the class file MainActivity.java file will create automatically under the package name

Resource (res) folder

- The resource folder is the most important folder because it contains all the non-code sources like images, XML layouts, and UI strings for our android application.

res/drawable folder

- It contains the different types of images used for the development of the application. We need to add all the images in a drawable folder for the application development.

res/layout folder

- The layout folder contains all XML layout files which we used to define the user interface of our application. It contains the **activity_main.xml** file.

res/mipmap folder

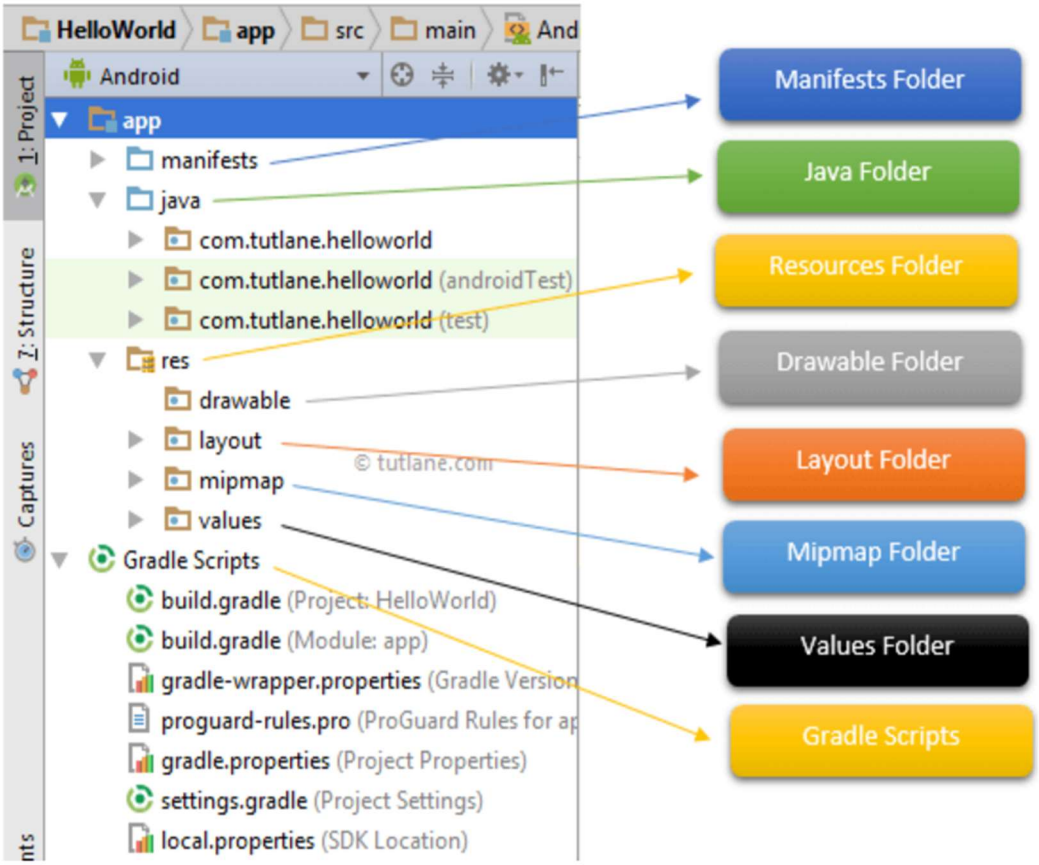
- This folder contains launcher.xml files to define icons that are used to show on the home screen.
- It contains different density types of icons depending upon the size of the device such as hdpi, mdpi, xhdpi.

res/values folder

- Values folder contains a number of XML files like strings, dimensions, colors, and style definitions.
- One of the most important files is the **strings.xml** file which contains the resources.

Gradle Scripts folder

- Gradle means automated build system and it contains a number of files that are used to define a build configuratiWon that can be applied to all modules in our application.
- In build.gradle (Project) there are buildscripts and in build.gradle (Module) plugins and implementations are used to build configurations that can be applied to all our application modules.



Components of Screen:

- User interface of android application consists of action bar and the application content area.
- Main Action Bar
- View Control
- Content Area
- Split Action Barclipse (IDE)

Understanding the screen components:

- The basic unit of android application is the activity.
- A UI is defined in an xml file.
- During compilation, each element in xml file is compiled into equivalent Android UI class with attributes represented by methods.

View and ViewGroup:

- An activity is consist of view.
- View is just widget that appears on screen.
- It could be button etc. One or more views can be grouped together into one ViewGroup.
- Example of ViewGroup includes layouts.
- An activity provides the window in which the app draws its UI.
- An activity displays the user interface of our application which may contain widgets like buttons, Image buttons, text boxes, labels.

Fundamental of UI Design:

- Its important of creation of user interface in any android application. Android presents some new terminology for programming.

1) View:

- View is a simple rectangle box which responds to the user's actions.
- The View is a base class for all UI components in android.
- It is used to create interactive UI components.
- For example, the **EditText** class is used to accept the input from users in android apps, which is a subclass of View.
- Following are the some of common View subclasses that will be used in android applications.

  - TextView
  - EditText
  - Button
  - CheckBox
  - RadioButton
  - ImageButton
  - Progress Bar

2) ViewGroup:

- The ViewGroup is a subclass of View and it will act as a base class for **layouts** and **layouts parameters**.
- The ViewGroup will provide an invisible container to hold other **Views** or **ViewGroups** and to define the layout properties.

- For example, Linear Layout is the ViewGroup that contains a UI controls like button, textview, etc. and other layouts also.
- Following are the commonly used ViewGroup subclasses in android applications.

  - Linear Layout
  - Relative Layout
  - Table Layout
  - Frame Layout
  - Web View
  - List View
  - Grid View

- Both View and ViewGroup subclasses together will play a key role to create a layout in android applications.

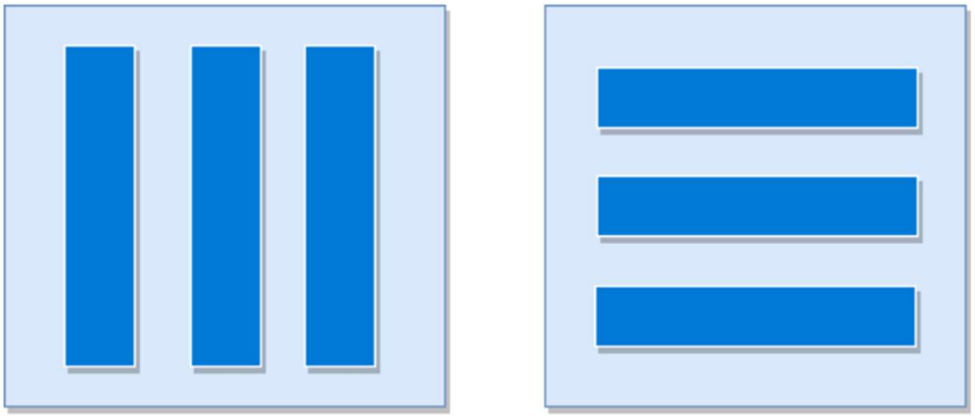
List any four attributes of checkbox:

- Checkbox has two states i.e ON / OFF
- Checkbox is a rectangular box which has associated label.
- We can checked or unchecked the box.
- Following are the attributes of checkboxes:

1) **id:** The id is an attribute used to uniquely identity a checkbox.
2) **Checked:** The checked is an attribute of checkbox used to set the current state of checkbox. The value should be true or false where true shows the checked state and false shows unchecked state of a checkbox. The default value of checked attributes is false.
3) **gravity:** The gravity attributes is an optional attribute which is used to control the alignment of text in checkbox like left, right, center, top, bottom, center-vertical, center-horizontal.
4) **textcolor:** The textcolor attribute is used to set the text color of a checkbox. Color value is in form of “#rgb”, “#rrggbb”, etc.
5) **textsize:** The textsize attribute is used to set the size of text of a checkbox. We can set the text size in sp or dp (density pixel).

LinearLayout:

- In android, **LinearLayout** is a **ViewGroup** subclass which is used to render all child **View** instances one by one either in **Horizontal** direction or **Vertical** direction based on the **orientation** property.
- In android, we can specify the linear layout orientation using **android:orientation** attribute.
- Following is the pictorial representation of linear layout in android applications.

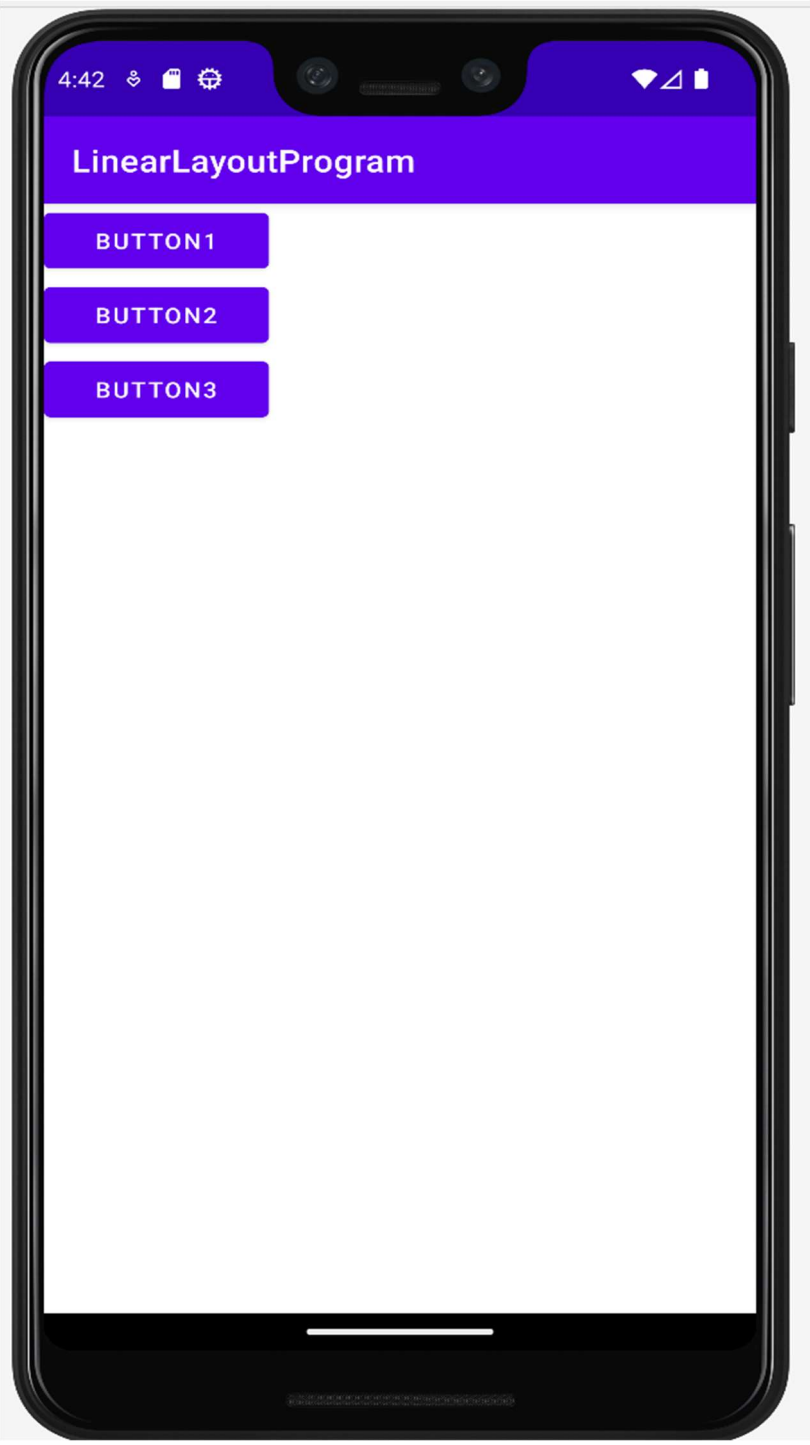


Horizontal Vertical

- In **LinearLayout**, the child **View** instances arranged one by one, so the horizontal list will have only one row of multiple columns and vertical list will have one column of multiple rows.
- Android LinearLayout Example(**activity_main.xml)**

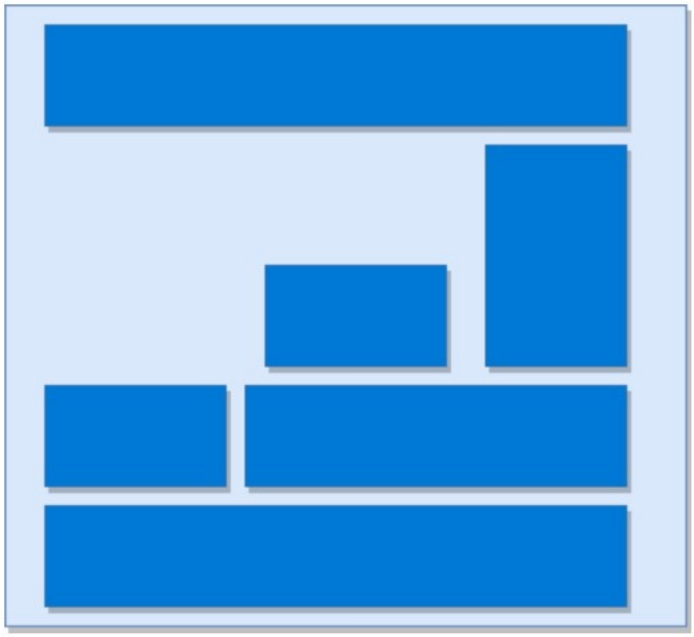
```
<?xml version="1.0" encoding="utf-8"?>
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
        <Button
                android:layout_width="130dp"
                android:layout_height="wrap_content"
                android:text="Button1"/>
        <Button
            android:layout_width="130dp"
            android:layout_height="wrap_content"
            android:text="Button2"/>
        <Button
            android:layout_width="130dp"
            android:layout_height="wrap_content"
            android:text="Button3"/>
    </LinearLayout>
```

- **Output of Android LinearLayout Example:**



RelativeLayout:

- In android, **RelativeLayout** is a **ViewGroup** which is used to specify the position of child **View** instances relative to each other (Child **A** to the left of Child **B**) or relative to the parent (Aligned to the top of parent).
- Following is the pictorial representation of relative layout in android applications.



- In android, **RelativeLayout** is very useful to design user interface because by using relative layout we can eliminate the nested view groups and keep our layout hierarchy flat, which improves the performance of application.
- In **RelativeLayout** we need to specify the position of child views relative to each other or relative to the parent. In case if we didn't specify the position of child views, by default all child views are positioned to top-left of the layout.
-
- Following are the some of most useful layout properties available to views in RelativeLayout.

Attribute	Description
layout_alignParentTop	If it specified "true", the top edge of view will match the top edge of the parent.

Attribute	Description
layout_alignParentBottom	If it specified “true”, the bottom edge of view will match the bottom edge of parent.
layout_alignParentLeft	If it specified “true”, the left edge of view will match the left edge of parent.
layout_alignParentRight	If it specified “true”, the right edge of view will match the right edge of the parent.
layout_centerInParent	If it specified “true”, the view will be aligned to the centre of parent.
layout_centerHorizontal	If it specified “true”, the view will be horizontally centre aligned within its parent.
layout_centerVertical	If it specified “true”, the view will be vertically centre aligned within its parent.
layout_above	It accepts another sibling view id and places the view above the specified view id.
layout_below	It accepts another sibling view id and places the view below the specified view id.
layout_toLeftOf	It accepts another sibling view id and places the view left of the specified view id.
layout_toRightOf	It accepts another sibling view id and places the view right of the specified view id.
layout_toStartOf	It accepts another sibling view id and places the view to start of the specified view id.

Attribute	Description
layout_toEndOf	It accepts another sibling view id and places the view to the end of the specified view id.

- Android LinearLayout Example(**activity_main.xml)**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="10dp"
    android:paddingRight="10dp">
    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"
        android:text="Button1" />
    <Button
        android:id="@+id/btn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"
        android:text="Button2" />
    <Button
        android:id="@+id/btn3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:text="Button3" />
    <Button
        android:id="@+id/btn4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:text="Button4" />
</RelativeLayout>
```

Output of Android RelativeLayout Example:

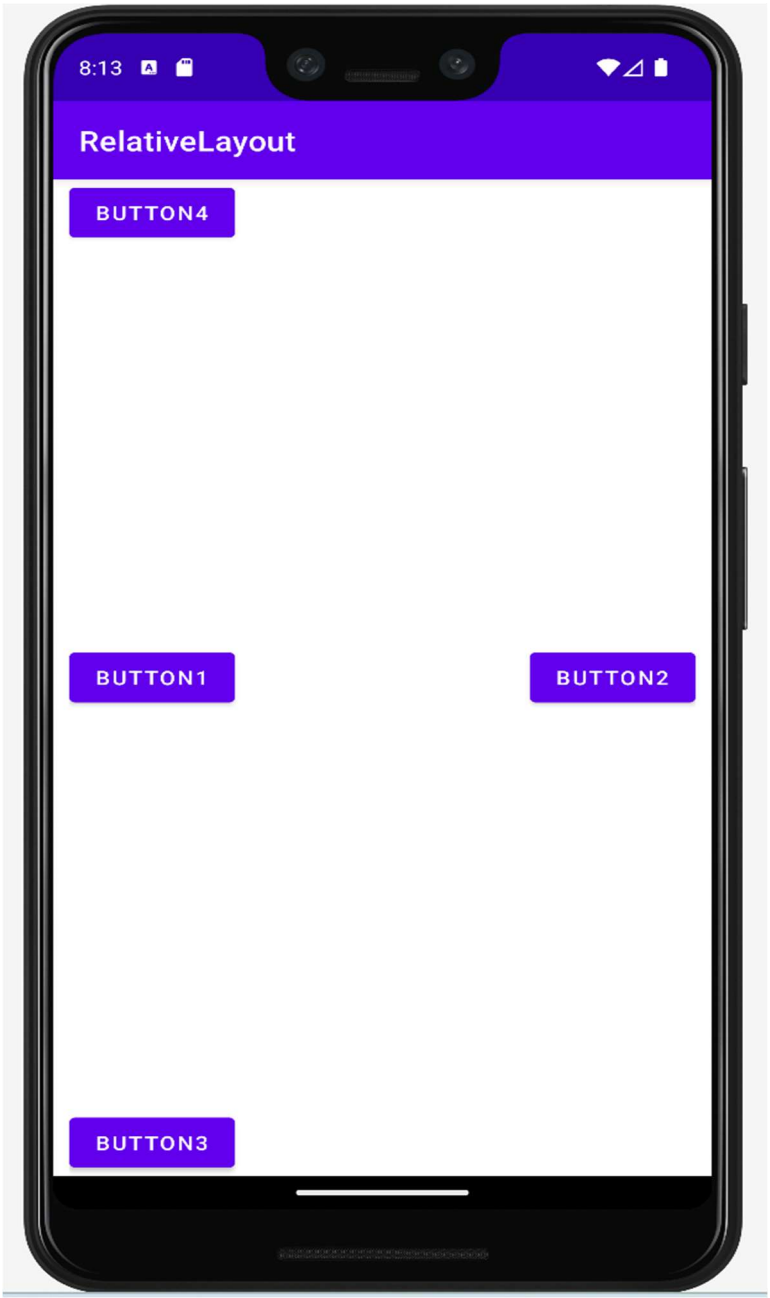


Table Layout:

- In android, **TableLayout** is a **ViewGroup** subclass that is used to display the child View elements in rows and columns.
- Following is the pictorial representation of table layout in android applications.

- In android, TableLayout will position its children elements into rows and columns and it won't display any border lines for rows, columns or cells.
- The TableLayout in android will work same as the HTML table and the table will have as many columns as the row with the most cells. The TableLayout can be explained as **<table>** and TableRow is like **<tr>** element.
- Android TableLayout Example(**activity_main.xml)**

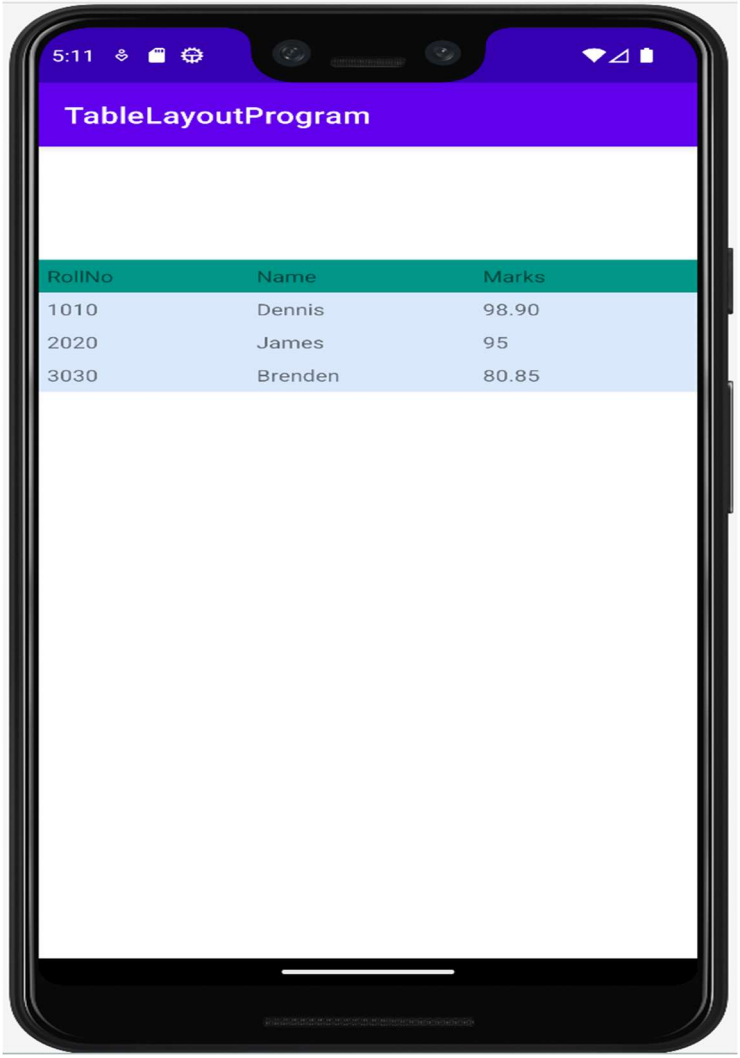
```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="100dp">

    <TableRow android:background="#009688" android:padding="5dp">>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="RollNo" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:layout_weight="1"
            android:text="Name" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Marks" />
    </TableRow>
    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="1010" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Dennis" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="98.90" />
    </TableRow>
    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="2020" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="James" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="95" />
    </TableRow>
    <TableRow android:background="#DAE8FC" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="3030" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Brenden" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
```

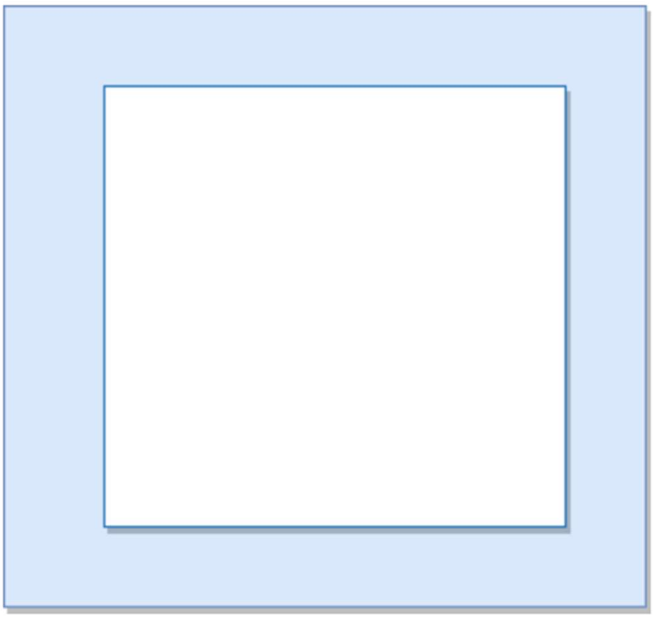
```
            android:text="80.85" />
    </TableRow>
</TableLayout>
```

- **Output of Android TableLayout Example:**



FrameLayout:

- In android, **Framelayout** is a **ViewGroup** subclass that is used to specify the position of **View** instances it contains on the top of each other to display only single **View** inside the FrameLayout.
- In simple manner, we can say FrameLayout is designed to block out an area on the screen to display a single item.
- Following is the pictorial representation of frame layout in android applications.



- In android, **FrameLayout** will act as a placeholder on the screen and it is used to hold a single child view.
- In FrameLayout, the child views are added in a stack and the most recently added child will show on the top. We can add multiple children views to FrameLayout and control their position by using gravity attributes in FrameLayout.
- Android FrameLayout Example(**activity_main.xml)**

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

    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:src="@drawable/vjtech123" />
    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="VJTech Academy"
        android:textColor="#E11515"
        android:textSize="30sp" />
</FrameLayout>
```

- **Output of Android FrameLayout Example:**

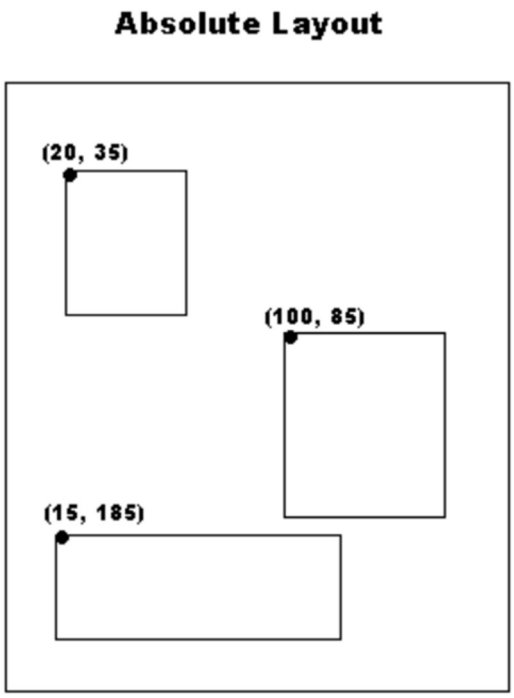
AbsoluteLayout:

- An Absolute Layout lets you specify exact locations (x/y coordinates) of its children.
- Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.

- **AbsoluteLayout Attributes:**

1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:layout_x** This specifies the x-coordinate of the view.
3	**android:layout_y** This specifies the y-coordinate of the view.

- Following is the pictorial representation of Absolute layout in android applications.



- Android Absolute Layout Example(**activity_main.xml)**

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

    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="OK"
        android:layout_x="50px"
        android:layout_y="361px" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:text="Cancel"
        android:layout_x="50px"
        android:layout_y="661px" />
</AbsoluteLayout>
```

- **Output of Android Absolute Layout Example:**