UNIT-3

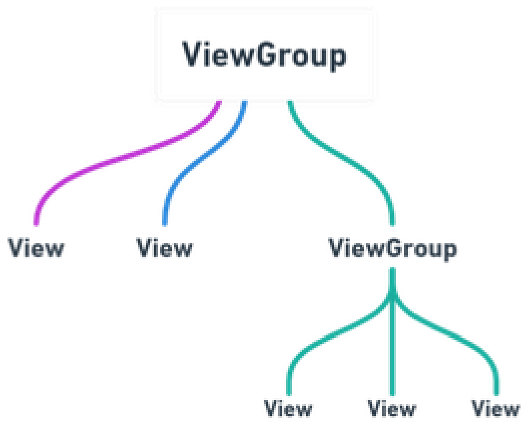
Working with Views

3.1 Working with View Groups

A ViewGroup is a special view that can contain other views. The ViewGroup is the base class for Layouts in android, like LinearLayout , RelativeLayout , FrameLayout etc. In other words, ViewGroup is generally used to define the layout in which views(widgets) will be set/arranged/listed on the android screen.

ViewGroups acts as an invisible container in which other Views and Layouts are placed. Yes, a layout can hold another layout in it, or in other words a **ViewGroup** can have another **ViewGroup** in it.**View:** A View is defined as the user interface which is used to create interactive UI components such as TextView, ImageView, EditText, RadioButton, etc., and is responsible for event handling and drawing. They are Generally Called Widgets.

ViewGroup: A ViewGroup act as a base class for layouts and layouts parameters that hold other Views or ViewGroups and to define the layout properties. They are Generally Called layouts.



The Android framework will allow us to use UI elements or widgets in two ways:

- Use UI elements in the XML file
- Create elements in the Kotlin file dynamically

3.2 Designing different types of Views

- **Android WebView:** WebView is a browser that is used to display the web pages in our activity layout.
- **Android ListView:** ListView is a ViewGroup, used to display scrollable lists of items in a single column.
- **Android GridView:** GridView is a ViewGroup that is used to display a scrollable list of items in a grid view of rows and columns.

- **Android RecyclerView:** A RecyclerView is an advanced version of ListView with improved performance.
- **Android CardView:** CardView is a new widget in Android that can be used to display any sort of data by providing a rounded corner layout along with a specific elevation.
- **Android TextView:** Android TextView is simply a view that are used to display the text to the user and optionally allow us to modify or edit it.
- **Android Button:** A Button is a user interface that is used to perform some action when clicked or tapped.
- **Android RadioGroup:** RadioGroup class of Kotlin programming language is used to create a container which holds multiple RadioButtons.
- **Android ToggleButton:** ToggleButton is just like a switch containing two states either ON or OFF which is represented using boolean values true and false respectively.
- **Android CheckBox:** A CheckBox is a special kind of button in Android which has two states either checked or unchecked.

ListView:

Android ListView is a ViewGroup which is used to display the list of items in multiple rows and contains an adapter which automatically inserts the items into the list.

The main purpose of the adapter is to fetch data from an array or database and insert each item that placed into the list for the desired result. So, it is main source to pull data from strings.xml file which contains all the required strings in Kotlin or xml files.

Android Adapter:

Adapter holds the data fetched from an array and iterates through each item in data set and generates the respective views for each item of the list. So, we can say it act as an intermediate between the data sources and adapter views such as ListView, Gridview.

Different Types of Adapter:

- **ArrayAdapter:** It always accepts an Array or List as input. We can store the list items in the strings.xml file also.
- **CursorAdapter:** It always accepts an instance of cursor.
- **SimpleAdapter:** It mainly accepts a static data defined in the resources like array or database.

- **BaseAdapter:** It is a generic implementation for all three adapter types and it can be used in the views according to our requirements.

activity_main.xml file

```
<?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="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <ListView
                android:id="@+id/userlist"
                android:layout_width="match_parent"
                android:layout_height="wrap_content" >
        </ListView>
</LinearLayout>
```

GridView:

A Grid View is a type of adapter view that is used to display the data in the grid layout format. For setting the data to the grid view adapter is used with the help of the setAdapter() method. This adapter helps to set the data from the database or array list to the items of the grid view. In this article, we will take a look at How to implement Grid View in Android using Kotlin language.

Working with the activity_main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="5dp"
        app:cardCornerRadius="5dp"
        app:cardElevation="5dp">
        <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
        android:orientation="vertical">

        <ImageView
            android:id="@+id/idIVCourse"
            android:layout_width="80dp"
            android:layout_height="80dp"
            android:layout_gravity="center"
            android:layout_margin="5dp"
            android:padding="4dp"
            android:src="@mipmap/ic_launcher" />

        <TextView
            android:id="@+id/idTVCourse"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="5dp"
            android:padding="4dp"
            android:text="@string/app_name"
            android:textAlignment="center"
            android:textColor="@color/black" />

    </LinearLayout>

</androidx.cardview.widget.CardView>
```

RecyclerView:

A RecyclerView is an advanced version of ListView with improved performance. When you have a long list of items to show you can use RecyclerView. It has the ability to reuse its views. In RecyclerView when the View goes out of the screen or not visible to the user it won't destroy it, it will reuse these views. This feature helps in reducing power consumption and providing more responsiveness to the application.

- **onCreateViewHolder():** This function sets the views to display the items.

- **onBindViewHolder():** This function is used to bind the list items to our widgets such as TextView, ImageView, etc.
- **getItemCount():** It returns the count of items present in the list.

Working with 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="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        tools:context=".MainActivity">

        <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/recyclerview"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                tools:itemCount="5"
                tools:listitem="@layout/card_view_design" />

</LinearLayout>
```

CardView:

CardView is a new widget in Android that can be used to display any sort of data by providing a rounded corner layout along with a specific elevation. CardView is the view that can display views on top of each other. The main usage of CardView is that it helps to give a rich feel and look to the UI design. This widget can be easily seen in many different Android Apps. CardView can be used for creating items in listview or inside Recycler View. The best part about CardView is that it extends Framelayout and it can be displayed on all platforms of Android.

Some important attributes of Cardview are :

1.cardBackgroundColor : Used for setting up the background-color of the card.

2.cardElevation : Defines the elevation (the process of moving to a higher place or more important position) of the card. It’s value should not be quite large else the design may not look good.

3.cardCornerRadius : It sets radius around the corners of the card. More the value of this attribute more circular the edges would appear in the card.

4.cardUseCompactPadding : It has two values true and false. By default, the cardview is set to (0,0) top left corner of the screen. And if this attribute is set to true then the card will set padding for itself so that our UI looks good. This case is helpful in the scenarios when our gravity is not set to center or any other parameters.

Working with XML:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        tools:context=".MainActivity">

<androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:cardElevation="10dp"
        app:cardCornerRadius="20dp"
        android:layout_margin="10dp"
        app:cardBackgroundColor="@color/white"
        app:cardMaxElevation="12dp"
        app:cardPreventCornerOverlap="true"
        app:cardUseCompatPadding="true"
        >

        <ImageView
                android:layout_width="200dp"
                android:layout_height="200dp"
                android:layout_gravity="center"
                android:src="@drawable/gfgimage"
                android:layout_margin="10dp"
                android:id="@+id/img"
                android:contentDescription="@string/app_name" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        android:layout_gravity="bottom|center_horizontal"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp"
        android:textSize="20sp"
        android:textStyle="bold"
        />
</androidx.cardview.widget.CardView>
</RelativeLayout>
```

TextView:

Android TextView is simply a view that are used to display the text to the user and optionally allow us to modify or edit it.

Different attributes of TextView:

Attributes	Description
ndroid:text	Sets text of the Textview
android:id	Gives a unique ID to the Textview
android:cursorVisible	Use this attribute to make cursor visible or invisible. Default value is visible.
android:drawableBottom	Sets images or other graphic assets to below of the Textview.
android:drawableEnd	Sets images or other graphic assets to end of Textview.
android:drawableLeft	Sets images or other graphic assets to left of Textview.
android:drawablePadding	Sets padding to the drawable(images or other graphic assets) in the Textview.
android:autoLink	This attribute is used to automatically detect url or emails and show it as clickable link.
android:autoText	Automatically correct spelling errors in text of the Textview.
android:capitalize	It automatically capitalize whatever the user types in the Textview.
android:drawableRight	Sets drawables to right of text in the Textview.
android:drawableStart	Sets drawables to start of text in the Textview.
android:drawableTop	Sets drawables to top of text in the Textview.

android:ellipsize	Use this attribute when you want text to be ellipsized if it is longer than the Textview width.
android:ems	Sets width of the Textview in ems.
android:gravity	We can align text of the Textview vertically or horizontally or both.
android:height	Use to set height of the Textview.
android:hint	Use to show hint when there is no text.
android:inputType	Use to set input type of the Textview. It can be Number, Password, Phone etc.
android:lines	Use to set height of the Textview by number of lines.
android:maxHeight	Sets maximum height of the Textview.
android:minHeight	Sets minimum height of the Textview.
android:maxLength	Sets maximum character length of the Textview.
android:maxLines	Sets maximum lines Textview can have.
android:minLines	Sets minimum lines Textview can have.
android:maxWidth	Sets maximum width Textview can have.
android:minWidth	Sets minimum lines Textview can have.
android:textAllCaps	Show all texts of the Textview in capital letters.
android:textColor	Sets color of the text.
android:textSize	Sets font size of the text.
android:textStyle	Sets style of the text. For example, bold, italic, bolditalic.
android:typeface	Sets typeface or font of the text. For example, normal, sans, serif etc
android:width	Sets width of the TextView.

activity_main.xml file;

```
<?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:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

        <TextView
```

```
                android:id="@+id/text_view_id"
                android:layout_height="wrap_content"
                android:layout_width="wrap_content"
                android:text="@string/text_view"
                android:textColor="#008000"
                android:textSize="40dp"
                android:textStyle="bold"/>
</LinearLayout>
```

Button:

A **Button** is a user interface that is used to perform some action when clicked or tapped. It is a very common widget in Android and developers often use it.

XML Attributes of Button:

XML Attributes	Description
android:id	Used to specify the id of the view.
android:text	Used to the display text of the button.
android:textColor	Used to the display color of the text.
android:textSize	Used to the display size of the text.
android:textStyle	Used to the display style of the text like Bold, Italic, etc.
android:textAllCaps	Used to display text in Capital letters.
android:background	Used to set the background of the view.
android:padding	Used to set the padding of the view.
android:visibility	Used to set the visibility of the view.
android:gravity	Used to specify the gravity of the view like center, top, bottom, etc

activity_main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:background="#168BC34A"
        tools:context=".MainActivity">
        <Button
                android:id="@+id/button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="#4CAF50"
                android:paddingStart="10dp"
                android:paddingEnd="10dp"
                android:text="@string/btn"
                android:textColor="@android:color/background_light"
                android:textSize="24sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

RadioGroup:

RadioGroup class of Kotlin programming language is used to create a container which holds multiple RadioButtons. The RadioGroup class is beneficial for placing a set of radio buttons inside it because this class adds multiple-exclusion scope feature to the radio buttons. This feature assures that the user will be able to check only one of the radio buttons among all which belongs to a RadioGroup class. If the user checks another radio button, the RadioGroup class unchecks the previously checked radio button.

XML attributes of RadioGroup:

XML Attributes	Description
android:id	To uniquely identify the RadioGroup
android:background	To set a background colour
android:onClick	A method to perform certain action when RadioGroup is clicked
android:onClick	It’s a name of the method to invoke when the radio button clicked.
android:visibility	Used to control the visibility i.e., visible, invisible or gone
android:layout_width	To set the width
android:layout_height	To set the height
android:contentDescription	To give a brief description of the view
android:checkedButton	Stores id of child radio button that needs to be checked by default within this radio group
android:orientation	To fix the orientation constant of the view

activity_main.xml file:

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

```
<androidx.constraintlayout.widget.ConstraintLayout

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

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

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

        android:layout_width="match_parent"

        android:layout_height="match_parent"

        android:background="#168BC34A"

        tools:context=".MainActivity">

        <TextView

                android:id="@+id/textView"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:fontFamily="@font/roboto"

                android:text="@string/Heading"

                android:textAlignment="center"

                android:textColor="@android:color/holo_green_dark"

                android:textSize="36sp"

                android:textStyle="bold"

                app:layout_constraintBottom_toBottomOf="parent"

                app:layout_constraintEnd_toEndOf="parent"

                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                app:layout_constraintVertical_bias="0.19" />
        <RadioGroup
                android:id="@+id/radioGroup1"
                android:layout_width="0dp"
                android:layout_height="wrap_content"
                android:background="#024CAF50"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@+id/textView"
                app:layout_constraintVertical_bias="0.24000001">
                <RadioButton
                    android:id="@+id/radioButton1"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:fontFamily="@font/roboto"
                    android:text="@string/RadioButton1"
                    android:textSize="24sp" />
                <RadioButton
                    android:id="@+id/radioButton2"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:fontFamily="@font/roboto"
                    android:text="@string/radioButton2"
                    android:textSize="24sp" />
                <RadioButton
                    android:id="@+id/radioButton3"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:fontFamily="@font/roboto"
                    android:text="@string/radioButton3"
                    android:textSize="24sp" />
                <RadioButton
                    android:id="@+id/radioButton4"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:fontFamily="@font/roboto"
                    android:text="@string/radioButton4"
                    android:textSize="24sp" />
                <RadioButton
                    android:id="@+id/radioButton5"
```

```
                    android:layout_width="match_parent"

                    android:layout_height="wrap_content"

                    android:fontFamily="@font/roboto"

                    android:text="@string/radioButton5"

                    android:textSize="24sp" />

        </RadioGroup>

</androidx.constraintlayout.widget.ConstraintLayout>
```

ToggleButton:

ToggleButton is just like a switch containing two states either ON or OFF which is represented using boolean values true and false respectively. ToggleButton unlike switch does not have a slider interface i.e we cannot slide to change the states.

Android ToggleButton XML Attributes:

Attribute	**Description**
android:id	The id assigned to the toggle button
android:textOff	The text shown on the button when it is not checked
android:textOn	The text shown on the button when it is checked
android:disabledAlpha	The alpha (opacity) to apply to the when disabled.

activity_main.xml:

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

<androidx.constraintlayout.widget.ConstraintLayout

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

```
    xmlns:app="http://schemas.android.com/apk/res-auto"

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

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">



    <ToggleButton

        android:id="@+id/toggleButton"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintEnd_toEndOf="parent"

        app:layout_constraintStart_toStartOf="parent"

        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

CheckBox:

A CheckBox is a special kind of button in Android which has two states either checked or unchecked. The Checkbox is a very common widget to be used in Android and a very good example is the “Remember me” option in any kind of Login activity of an app which asks the user to activate or deactivate that service. There are many other uses of the CheckBox widget like offering a list of options to the user to choose from and the options are mutually exclusive i.e., the user can select more than one option. This feature of the CheckBox makes it a better option to be used in designing multiple-choice questions application or survey application in android.

XML attributes of CheckBox:

XML Attributes	**Description**
android:id	Used to uniquely identify a CheckBox
android:checked	To set the default state of a CheckBox as checked or unchechek
android:background	To set the background color of a CheckBox
android:text	Used to store a text inside the CheckBox
android:fontFamily	To set the font of the text of the CheckBox
android:textSize	To set the CheckBox text size
android:layout_width	To set the CheckBox width
android:layout_height	To set the CheckBox height
android:gravity	Used to adjust the CheckBox text alignment
android:padding	Used to adjust the left, right, top and bottom padding of the CheckBox

activity_main.xml file:

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

<androidx.constraintlayout.widget.ConstraintLayout

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

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

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

        android:layout_width="match_parent"

        android:layout_height="match_parent"
```

```
android:background="#168BC34A"

tools:context=".MainActivity" >

<TextView

        android:id="@+id/textView"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:fontFamily="@font/roboto"

        android:text="@string/Heading"

        android:textAlignment="center"

        android:textColor="@android:color/holo_green_dark"

        android:textSize="36sp"

        android:textStyle="bold"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintEnd_toEndOf="parent"

        app:layout_constraintStart_toStartOf="parent"

        app:layout_constraintTop_toTopOf="parent"

        app:layout_constraintVertical_bias="0.17000002" />

<LinearLayout

        android:id="@+id/checkBox_container"

        android:layout_width="0dp"

        android:layout_height="wrap_content"
```

```
android:orientation="vertical"

app:layout_constraintBottom_toBottomOf="parent"

app:layout_constraintEnd_toEndOf="parent"

app:layout_constraintStart_toStartOf="parent"

app:layout_constraintTop_toBottomOf="@+id/textView"

app:layout_constraintVertical_bias="0.18">

<CheckBox

        android:id="@+id/checkBox"

        android:layout_width="match_parent"

        android:layout_height="wrap_content"

        android:fontFamily="@font/roboto"

        android:text="@string/checkBox1_text"

        android:textSize="18sp"

        android:padding="7dp"/>

<CheckBox

        android:id="@+id/checkBox2"

        android:layout_width="match_parent"

        android:layout_height="wrap_content"

        android:fontFamily="@font/roboto"

        android:text="@string/checkBox2_text"

        android:textSize="18sp"
        android:padding="7dp"/>

    <CheckBox

        android:id="@+id/checkBox3"

        android:layout_width="match_parent"

        android:layout_height="wrap_content"

        android:fontFamily="@font/roboto"

        android:text="@string/checkBox3_text"

        android:textSize="18sp"

        android:padding="7dp"/>

    <CheckBox

        android:id="@+id/checkBox4"

        android:layout_width="match_parent"

        android:layout_height="wrap_content"

        android:fontFamily="@font/roboto"

        android:text="@string/checkBox4_text"

        android:textSize="18sp"

        android:padding="7dp"/>

    <CheckBox

        android:id="@+id/checkBox5"

        android:layout_width="match_parent"

        android:layout_height="wrap_content"
```

```
                    android:fontFamily="@font/roboto"

                    android:text="@string/checkBox5_text"

                    android:textSize="18sp"

                    android:padding="7dp"/>

        </LinearLayout>



        <Button

                android:id="@+id/submitButton"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:background="#AB4CAF50"

                android:fontFamily="@font/roboto"

                android:text="@string/submitButton"

                android:textSize="18sp"

                android:textStyle="bold"

                app:layout_constraintBottom_toBottomOf="parent"

                app:layout_constraintEnd_toEndOf="parent"

                app:layout_constraintStart_toStartOf="parent"

                app:layout_constraintTop_toBottomOf="@+id/checkBox_container"

                app:layout_constraintVertical_bias="0.23000002" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

3.3 Implementing Screen Orientation

Screen Orientation, also known as **screen rotation**, is the attribute of activity element in android. When screen orientation change from one state to other, it is also known as **configuration change**.

States of Screen Orientation :

There are various possible **screen orientation states** for any android application, such as:

- ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE
- ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
- ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED
- ActivityInfo.SCREEN_ORIENTATION_USER
- ActivityInfo.SCREEN_ORIENTATION_SENSOR
- ActivityInfo.SCREEN_ORIENTATION_BEHIND
- ActivityInfo.SCREEN_ORIENTATION_NOSENSOR
- ActivityInfo.SCREEN_ORIENTATION_SENSOR_LANDSCAPE
- ActivityInfo.SCREEN_ORIENTATION_SENSOR_PORTRAIT
- ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT

The initial orientation of the Screen has to be defined in the AndroidManifest.xml file.

Syntax:

```
<activity android:name="package_name.Your_ActivityName"
          android:screenOrientation="orientation_type">
</activity>
```

Example:

```
   android:screenOrientation="orientation_type">
```

How to change Screen orientation?

We will create **two activities** of different screen orientation.

- The first activity will be as "**portrait**" orientation and
- Second activity as "**landscape**" orientation state.

Step-by-Step demonstration:

- **Creating the activities:** There will be two activities and hence two XML files, one for each activity.
    1. **activity_main.xml**: XML file for first activity consist of constraint layout with Button and Text View in it. This activity is in Landscape state.
    2. **activity_next.xml**: XML file for second activity consist of constraint layout with Text View in it. This activity is in Landscape state.

activity_main.xml:

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

```
<android.support.constraint.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context="com.screenlayout.screenorientation.MainActivity">
        <Button
                android:id="@+id/b1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Next Activity"
                android:layout_marginTop="100dp"
                android:onClick="onClick"
                android:layout_marginBottom="10dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintVertical_bias="0.613"
                app:layout_constraintHorizontal_bias="0.612"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@+id/tv1"
                />
        <TextView
                android:text="Potrait orientation"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_marginTop="124dp"
                android:textSize="25dp"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.502"
                app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Activity_next.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout


    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    tools:context="com.screenlayout.screenorientation.NextActivity">



    <TextView
        android:id="@+id/tv"
        android:text="Landscape orientation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="170dp"
        android:textSize="22dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

- **Creating the Java file:** There will be two activities and hence two Java files, one for each activity.
    1. **MainActivity.java**: Java file for Main Activity, in which *setOnClick() listener* is attached to the button to launch next activity with different orientation.
    2. **NextActivity.java**: Java file for Next Activity which is in Landscape mode.

Mainactivity.java

```
package com.geeksforgeeks.screenorientation;

import android.support.v7.app.AppCompatActivity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

        // declare button variable
        Button button;

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

                // initialise button with id
                button = findViewById(R.id.b1);
        }

        // onClickListener attached to button
        // to send intent to next activity
        public void onClick(View v)
        {
                // Create instance of intent and
                // startActivity with intent object
                Intent intent
                        = new Intent(
                                MainActivity.this,
                                        NextActivity.class);
                startActivity(intent);
        }
}
```

Nextactivity.java:

```
package com.geeksforgeeks.screenorientation;

import android.support.v7.app.AppCompatActivity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

        // declare button variable
        Button button;

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

                // initialise button with id
                button = findViewById(R.id.b1);
        }

        // onClickListener attached to button
        // to send intent to next activity
        public void onClick(View v)
        {
                // Create instance of intent and
                // startActivity with intent object
                Intent intent
                        = new Intent(
                                MainActivity.this,
                                NextActivity.class);
                startActivity(intent);
        }
}
```

Updating the AndroidManifest file: In AndroidManifest.xml file, add the **screenOrientation state** in activity along with its orientation. Here, we provide "**portrait**" orientation for MainActivity and "**landscape**" for NextActivity.

AndroidManifest.xml:

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


        <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">
                <!-Define potrait orientation for Main activity-->
                <activity
                        android:name="com.geeksforgeeks.screenorientation.MainActivity"
                        android:screenOrientation="portrait">
                        <intent-filter>
                                <action android:name="android.intent.action.MAIN" />
                                <category android:name="android.intent.category.LAUNCHER" />
                        </intent-filter>
                </activity>
                <!--Define landscape orientation for NextActivity-->
        <activity android:name=".NextActivity"
            android:screenOrientation="landscape">
        </activity>
    </application>
</manifest>
```

3.4. Designing the Views Programmatically

We can create or instantiate UI elements or widgets during runtime by using the custom View and ViewGroup objects programmatically in the Kotlin file. Below is the example of creating a layout using LinearLayout to hold an EditText and a Button in an activity programmatically.

Kotlin

```
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.LinearLayout
import android.widget.Toast
import android.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // create the button
        val showButton = Button(this)
        showButton.setText("Submit")

        // create the editText
        val editText = EditText(this)

        val linearLayout = findViewById<LinearLayout>(R.id.l_layout)
        linearLayout.addView(editText)
        linearLayout.addView(showButton)
        // Setting On Click Listener
        showButton.setOnClickListener
        {
            // Getting the user input
            val text = editText.text

            // Showing the user input
            Toast.makeText(this, text, Toast.LENGTH_SHORT).show()
        }
    }
}
```

Different Attribute of the Layouts

XML attributes	Description
android:id	Used to specify the id of the view.
android:layout_width	Used to declare the width of View and ViewGroup elements in the layout.
android:layout_height	Used to declare the height of View and ViewGroup elements in the layout.
android:layout_marginLeft	Used to declare the extra space used on the left side of View and ViewGroup elements.
android:layout_marginRight	Used to declare the extra space used on the right side of View and ViewGroup elements.
android:layout_marginTop	Used to declare the extra space used in the top side of View and ViewGroup elements.

XML attributes	Description
android:layout_marginBottom	Used to declare the extra space used in the bottom side of View and ViewGroup elements.
android:layout_gravity	Used to define how child Views are positioned in the layout.