Unit -3
Designing the User Interface [5 Hrs]

Android Layout Types

A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with. Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects, as shown in figure.

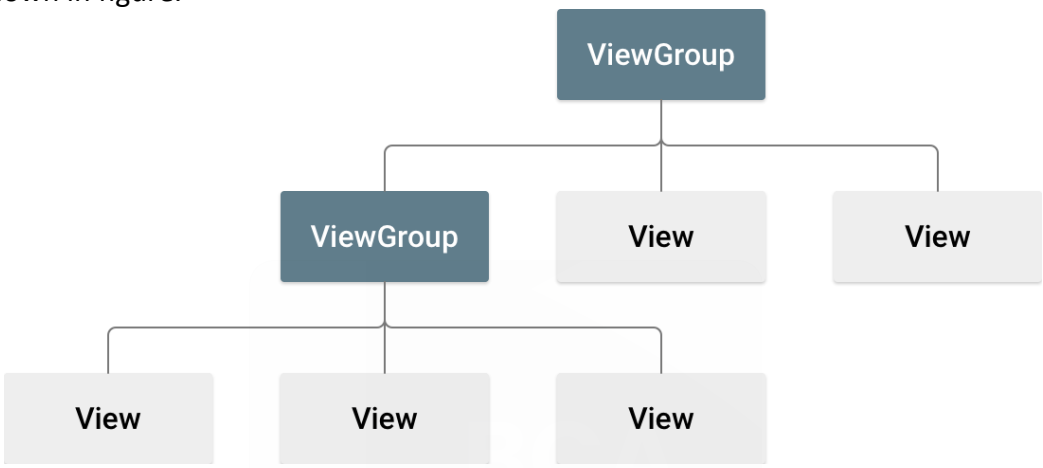

Figure 3-1. Illustration of a view hierarchy, which defines a UI layout

The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView. The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout .

You can declare a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts. You can also use Android Studio's Layout Editor to build your XML layout using a drag-and-drop interface.
- **Instantiate layout elements at runtime**. Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

Following are the different types of layout used for designing the User Interface.

LinearLayout

LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the android:orientation attribute.

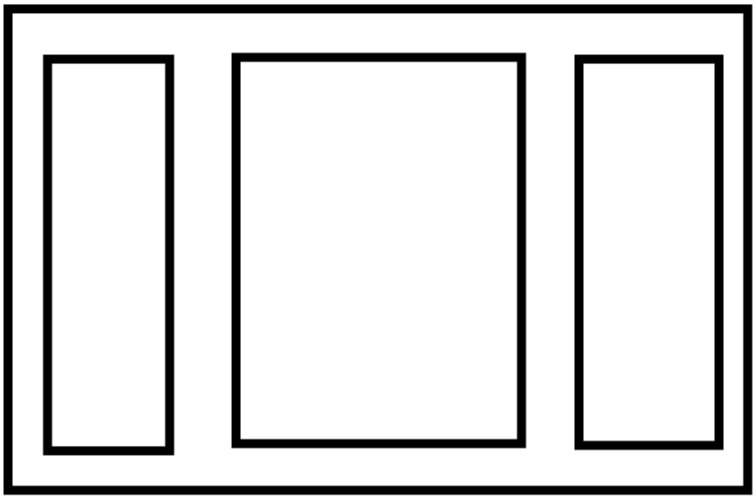

Figure 3-2. Representation of LinearLayout

All children of a LinearLayout are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding). A LinearLayout respects margins between children and the gravity (right, center, or left alignment) of each child.

Following code snippet will demonstrate LinearLayout.

```
<?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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is label 1"
        android:textSize="20sp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is label 2"
        android:textSize="20sp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="This is label 3"
        android:textSize="20sp"/>

</LinearLayout>
```

Above code will produce following output.

Figure 3-3. Output produced for LinearLayout

Following are some attributes used with LinearLayout:

android:baselineAligned	When set to false, prevents the layout from aligning its children's baselines.
android:baselineAlignedChildIndex	When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align to (that is, which child TextView).
android:divider	Drawable to use as a vertical divider between buttons.
android:gravity	Specifies how an object should position its content, on both the X and Y axes, within its own bounds.
android:measureWithLargestChild	When set to true, all children with a weight will be considered having the minimum size of the largest child.
android:orientation	Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.
android:weightSum	Defines the maximum weight sum.

Layout Weight

LinearLayout also supports assigning a weight to individual children with the **android:layout_weight** attribute. This attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen. A larger weight value allows it to expand to fill any remaining space in the parent view. Child views can specify a weight

value, and then any remaining space in the view group is assigned to children in the proportion of their declared weight. Default weight is zero.

Equal distribution

To create a linear layout in which each child uses the same amount of space on the screen, set the android:layout_height of each view to "0dp" (for a vertical layout) or the android:layout_width of each view to "0dp" (for a horizontal layout). Then set the android:layout_weight of each view to "1".

Unequal distribution

You can also create linear layouts where the child elements use different amounts of space on the screen:

- If there are three text fields and two of them declare a weight of 1, while the other is given no weight, the third text field without weight doesn't grow. Instead, this third text field occupies only the area required by its content. The other two text fields, on the other hand, expand equally to fill the space remaining after all three fields are measured.
- If there are three text fields and two of them declare a weight of 1, while the third field is then given a weight of 2 (instead of 0), then it's now declared more important than both the others, so it gets half the total remaining space, while the first two share the rest equally.

Following code snippet demonstrate layout_weight attribute. Here we are creating two TextViews. First TextView have weight equal to 2 and Second TextView have weight equal to 1. Total weight of LinearLayout is 3.

```
<?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:weightSum="3"
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 1"
        android:layout_weight="2"
        android:textSize="20sp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label2"
        android:layout_weight="1"
        android:textSize="20sp" />

</LinearLayout>
```

Above code will produce following output:

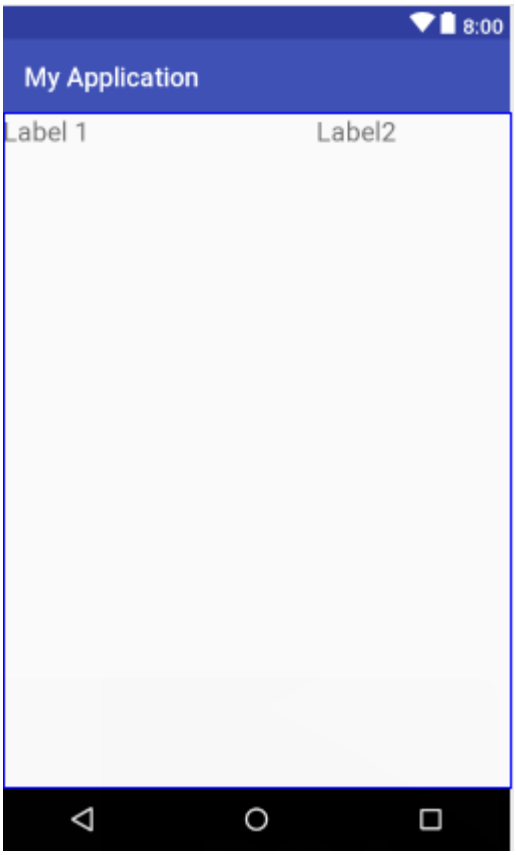

Figure 3-4. Output demonstrating layout_weight attribute

RelativeLayout

RelativeLayout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left or center).

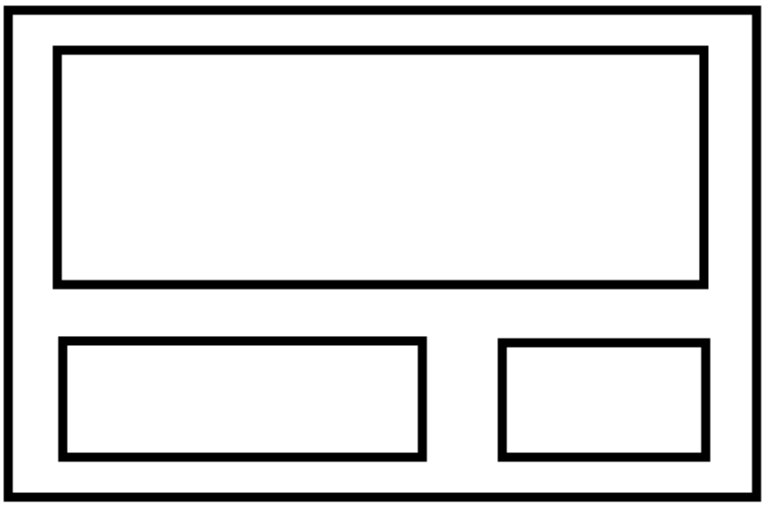

Figure 3-5. Representation of RelativeLayout

A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance. If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout.

Following are some attributes used with RelativeLayout:

android:layout_above	Positions the bottom edge of this view above the given anchor view ID.
android:layout_alignBaseline	Positions the baseline of this view on the baseline of the given anchor view ID.
android:layout_alignBottom	Makes the bottom edge of this view match the bottom edge of the given anchor view ID.
android:layout_alignEnd	Makes the end edge of this view match the end edge of the given anchor view ID.
android:layout_alignLeft	Makes the left edge of this view match the left edge of the given anchor view ID.
android:layout_alignParentBottom	If true, makes the bottom edge of this view match the bottom edge of the parent.
android:layout_alignParentEnd	If true, makes the end edge of this view match the end edge of the parent.
android:layout_alignParentLeft	If true, makes the left edge of this view match the left edge of the parent.
android:layout_alignParentRight	If true, makes the right edge of this view match the right edge of the parent.
android:layout_alignParentStart	If true, makes the start edge of this view match the start edge of the parent.
android:layout_alignParentTop	If true, makes the top edge of this view match the top edge of the parent.
android:layout_alignRight	Makes the right edge of this view match the right edge of the given anchor view ID.
android:layout_alignStart	Makes the start edge of this view match the start edge of the given anchor view ID.
android:layout_alignTop	Makes the top edge of this view match the top edge of the given anchor view ID.
android:layout_alignWithParentIfMissing	If set to true, the parent will be used as the anchor when the anchor cannot be be found for layout_toLeftOf, layout_toRightOf, etc.
android:layout_below	Positions the top edge of this view below the given anchor view ID.
android:layout_centerHorizontal	If true, centers this child horizontally within its parent.
android:layout_centerInParent	If true, centers this child horizontally and vertically within its parent.

android:layout_centerVertical	If true, centers this child vertically within its parent.
android:layout_toEndOf	Positions the start edge of this view to the end of the given anchor view ID.
android:layout_toLeftOf	Positions the right edge of this view to the left of the given anchor view ID.
android:layout_toRightOf	Positions the left edge of this view to the right of the given anchor view ID.
android:layout_toStartOf	Positions the end edge of this view to the start of the given anchor view ID.

Following code snippet will demonstrate RelativeLayout:

```
<?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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 1"
        android:textSize="20sp"
        android:id="@+id/label1" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 2"
        android:textSize="20sp"
        android:id="@+id/label2"
        android:layout_toRightOf="@+id/label1"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 3"
        android:textSize="20sp"
        android:id="@+id/label3"
        android:layout_below="@+id/label1" />

</RelativeLayout>
```

Above code will produce following output:

Figure 3-6. Output demonstrating RelativeLayout

TableLayout

TableLayout is a layout that arranges its children into rows and columns. It consists of a number of TableRow objects, each defining a row (actually, you can have other children, which will be explained below). TableLayout containers do not display border lines for their rows, columns, or cells. Each row has zero or more cells; each cell can hold one View object. The table has as many columns as the row with the most cells. A table can leave cells empty. Cells can span columns, as they can in HTML.

The children of a TableLayout cannot specify the layout_width attribute. Width is always MATCH_PARENT. However, the layout_height attribute can be defined by a child; default value is ViewGroup.LayoutParams.WRAP_CONTENT. If the child is a TableRow, then the height is always ViewGroup.LayoutParams.WRAP_CONTENT.

Cells must be added to a row in increasing column order, both in code and XML. Column numbers are zero-based. If you don't specify a column number for a child cell, it will auto increment to the next available column. If you skip a column number, it will be considered an empty cell in that row.

Although the typical child of a TableLayout is a TableRow, you can actually use any View subclass as a direct child of TableLayout. The View will be displayed as a single row that spans all the table columns.

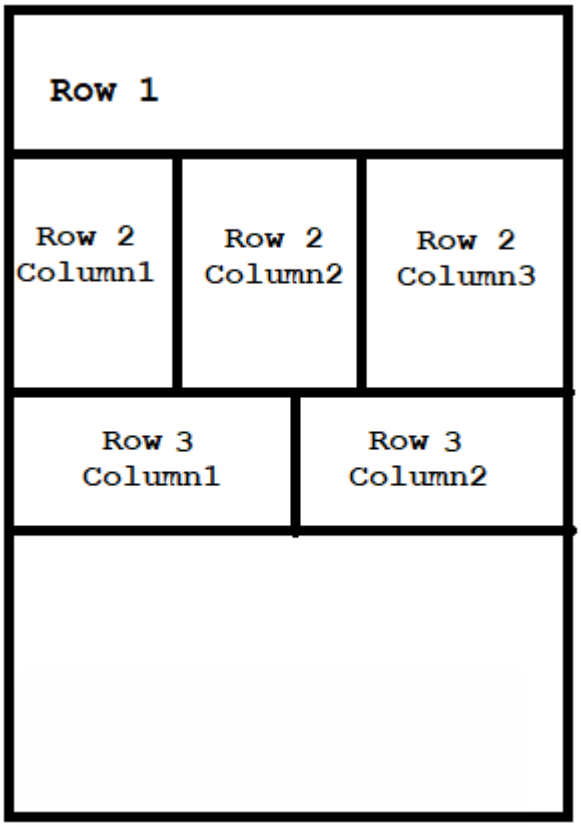

Figure 3-7. Representation of TableLayout

Following are the attributes associated with TableLayout.

android:collapseColumns	The zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5. Illegal and duplicate indices are ignored.
android:shrinkColumns	The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5. Illegal and duplicate indices are ignored. You can shrink all columns by using the value "*" instead. Note that a column can be marked stretchable and shrinkable at the same time.
android:stretchColumns	The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5. Illegal and duplicate indices are ignored. You can stretch all columns by using the value "*" instead. Note that a column can be marked stretchable and shrinkable at the same time.

Following code snippet will demonstrate TableLayout.

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:stretchColumns="*"
    android:collapseColumns="1"
    android:layout_height="match_parent">

    <TableRow
        android:layout_height="match_parent"
        android:layout_width="match_parent">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Label 1"
            android:textSize="20sp"
            android:layout_column="1"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Label 2"
            android:textSize="20sp"
            android:layout_column="2"/>
    </TableRow>

    <TableRow
        android:layout_height="match_parent"
        android:layout_width="match_parent">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Label 3"
            android:textSize="20sp"
            android:layout_column="1"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Label 4"
            android:textSize="20sp"
            android:layout_column="2"/>

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Label 5"
            android:textSize="20sp"
            android:layout_column="3"/>
    </TableRow>
</TableLayout>
```

Above code will display following output:

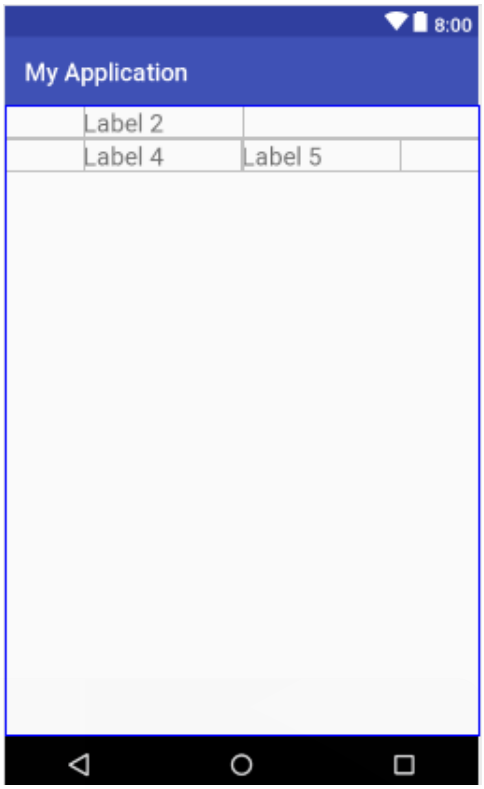

Figure 3-8. Output demonstrating TableLayout

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.

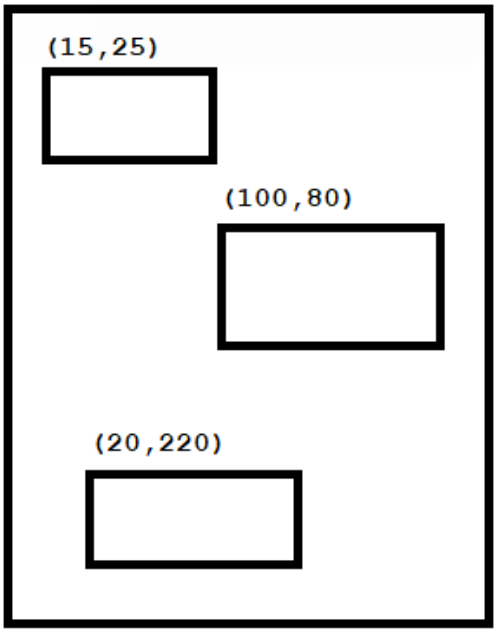

Figure 3-9. Representation of AbsoluteLayout

Following are the important attributes specific to AbsoluteLayout.

android:layout_x	This specifies the x-coordinate of the view.
android:layout_y	This specifies the y-coordinate of the view.

Following code snippet will demonstrate AbsoluteLayout:

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 1"
        android:textSize="20sp"
        android:layout_x="100dp"
        android:layout_y="50dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 2"
        android:textSize="20sp"
        android:layout_x="150dp"
        android:layout_y="100dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 3"
        android:textSize="20sp"
        android:layout_x="180dp"
        android:layout_y="150dp" />

</AbsoluteLayout>
```

Since AbsoluteLayout was deprecated in API level 3. So, we can use RelativeLayout, LinearLayout or any other layout for better performance.

Above code will produce following output:

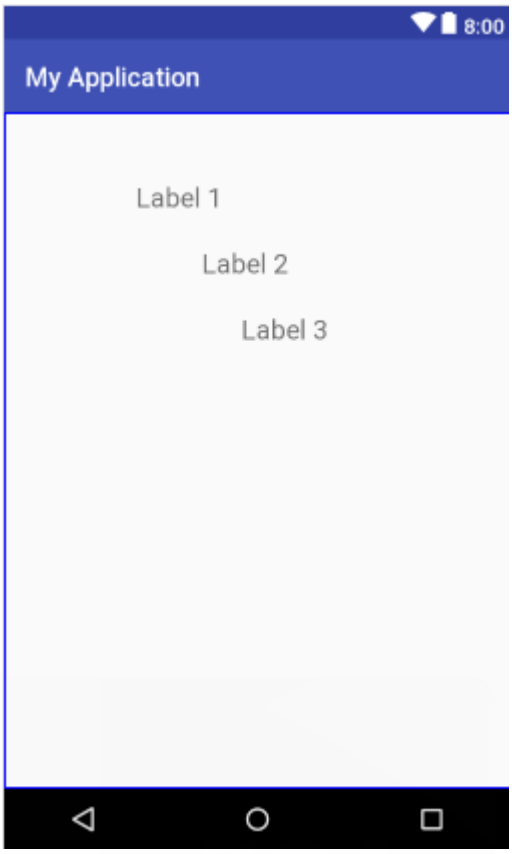

Figure 3-10. Output Demonstrating AbsoluteLayout

ConstraintLayout

Constraint layout is an advanced version of a Relative layout. It is used to reduce the child view hierarchies and improve the performance. It is used to define a layout by assigning constraints for every child view/widget relative to other views present.

A ConstraintLayout is similar to a RelativeLayout, but with more power. The aim of ConstraintLayout is to improve the performance of the applications by removing the nested views with a flat and flexible design.

The aim of the ConstraintLayout is to help reduce the number of nested views, which will improve the performance of our layout files. The layout class also makes it easier for us to define layouts than when using a RelativeLayout as we can now anchor any side of a view with any side of another, rather than having to place a whole view to any side of another.

For example, the attributes of a relative layout allow us to position a view using:

- layout_toRightOf
- layout_toLeftOf
- layout_toTopOf
- layout_toBottomOf

However, the ConstraintLayout features several more attributes:

- **layout_constraintTop_toTopOf** — Align the **top** of the desired view to the **top** of another.
- **layout_constraintTop_toBottomOf** — Align the **top** of the desired view to the **bottom** of another.
- **layout_constraintBottom_toTopOf** — Align the **bottom** of the desired view to the **top** of another.
- **layout_constraintBottom_toBottomOf** — Align the **bottom** of the desired view to the **bottom** of another.
- **layout_constraintLeft_toTopOf** — Align the **left** of the desired view to the **top** of another.
- **layout_constraintLeft_toBottomOf** — Align the **left** of the desired view to the **bottom** of another.
- **layout_constraintLeft_toLeftOf** — Align the **left** of the desired view to the **left** of another.
- **layout_constraintLeft_toRightOf** — Align the **left** of the desired view to the **right** of another.
- **layout_constraintRight_toTopOf** — Align the **right** of the desired view to the **top** of another.
- **layout_constraintRight_toBottomOf** — Align the **right** of the desired view to the **bottom** of another.
- **layout_constraintRight_toLeftOf** — Align the **right** of the desired view to the **left** of another.
- **layout_constraintRight_toRightOf** — Align the **right** of the desired view to the **right** of another.
- If desired, attributes supporting **start** and **end** are also available in place of **left** and **right** alignment.

Following code snippet will demonstrate constraint layout.

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 1"
        android:id="@+id/label1"
        android:textSize="20sp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 2"
        android:textSize="20sp"
        android:id="@+id/label2"
        app:layout_constraintLeft_toRightOf="@+id/label1"
        app:layout_constraintTop_toBottomOf="@+id/label1" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Label 3"
        android:textSize="20sp"
        android:id="@+id/label3"
        app:layout_constraintBottom_toTopOf="@+id/label2"
        app:layout_constraintLeft_toRightOf="@+id/label2" />

</android.support.constraint.ConstraintLayout>
```

Note: Please add following dependency in your build.gradle file.

```
implementation 'com.android.support.constraint:constraint-layout:1.1.3'
```

Above code will produce following output:

Figure 3-11. Output Demonstrating ConstraintLayout

Android Widgets

Widgets are the building blocks you use to compose a user interface. A widget can show text or graphics, interact with the user, or arrange other widgets on the screen. Buttons, text input controls, and check boxes are all types of widgets. Some of the common widgets used in android are discussed below:

TextView

It is a user interface element that displays text to the user. Following are the attributes associated with TextView.

android:allowUndo	Whether undo should be allowed for editable text.
android:autoLink	Controls whether links such as urls and email addresses are automatically found and converted to clickable links.
android:autoSizeMaxTextSize	The maximum text size constraint to be used when auto-sizing text.
android:autoSizeMinTextSize	The minimum text size constraint to be used when auto-sizing text.
android:autoSizePresetSizes	Resource array of dimensions to be used in conjunction with autoSizeTextType set to uniform.
android:autoSizeStepGranularity	Specify the auto-size step size if autoSizeTextType is set to uniform.
android:autoSizeTextType	Specify the type of auto-size.
android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:breakStrategy	Break strategy (control over paragraph layout).
android:bufferType	Determines the minimum type that getText() will return.
android:capitalize	If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types.
android:cursorVisible	Makes the cursor visible (the default) or invisible.
android:digits	If set, specifies that this TextView has a numeric input method and that these specific characters are the ones that it will accept.
android:drawableBottom	The drawable to be drawn below the text.
android:drawableEnd	The drawable to be drawn to the end of the text.
android:drawableLeft	The drawable to be drawn to the left of the text.
android:drawablePadding	The padding between the drawables and the text.
android:drawableRight	The drawable to be drawn to the right of the text.
android:drawableStart	The drawable to be drawn to the start of the text.
android:drawableTint	Tint to apply to the compound (left, top, etc.) drawables.
android:drawableTintMode	Blending mode used to apply the compound (left, top, etc.) drawables tint.
android:drawableTop	The drawable to be drawn above the text.

android:editable	If set, specifies that this TextView has an input method.
android:editorExtras	Reference to an <input-extras> XML resource containing additional data to supply to an input method, which is private to the implementation of the input method.
android:elegantTextHeight	Elegant text height, especially for less compacted complex script text.
android:ellipsize	If set, causes words that are longer than the view is wide to be ellipsized instead of broken in the middle.
android:ems	Makes the TextView be exactly this many ems wide.
android:enabled	Specifies whether the widget is enabled.
android:fallbackLineSpacing	Whether to respect the ascent and descent of the fallback fonts that are used in displaying the text.
android:firstBaselineToTopHeight	Distance from the top of the TextView to the first text baseline.
android:fontFamily	Font family (named by string or as a font resource reference) for the text.
android:fontFeatureSettings	Font feature settings.
android:fontVariationSettings	Font variation settings.
android:freezesText	If set, the text view will include its current complete text inside of its frozen icicle in addition to meta-data such as the current cursor position.
android:gravity	Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
android:height	Makes the TextView be exactly this tall.
android:hint	Hint text to display when the text is empty.
android:hyphenationFrequency	Frequency of automatic hyphenation.
android:imeActionId	Supply a value for EditorInfo.actionId used when an input method is connected to the text view.
android:imeActionLabel	Supply a value for EditorInfo.actionLabel used when an input method is connected to the text view.
android:imeOptions	Additional features you can enable in an IME associated with an editor to improve the integration with your application.
android:includeFontPadding	Leave enough room for ascenders and descenders instead of using the font ascent and descent strictly.
android:inputMethod	If set, specifies that this TextView should use the specified input method (specified by fully-qualified class name).
android:inputType	The type of data being placed in a text field, used to help an input method decide how to let the user enter text.
android:justificationMode	Mode for justification.
android:lastBaselineToBottomHeight	Distance from the bottom of the TextView to the last text baseline.
android:letterSpacing	Text letter-spacing.
android:lineHeight	Explicit height between lines of text.

android:lineSpacingExtra	Extra spacing between lines of text.
android:lineSpacingMultiplier	Extra spacing between lines of text, as a multiplier.
android:lines	Makes the TextView be exactly this many lines tall.
android:linksClickable	If set to false, keeps the movement method from being set to the link movement method even if autoLink causes links to be found.
android:marqueeRepeatLimit	The number of times to repeat the marquee animation.
android:maxEms	Makes the TextView be at most this many ems wide.
android:maxHeight	Makes the TextView be at most this many pixels tall.
android:maxLength	Set an input filter to constrain the text length to the specified number.
android:maxLines	Makes the TextView be at most this many lines tall.
android:maxWidth	Makes the TextView be at most this many pixels wide.
android:minEms	Makes the TextView be at least this many ems wide.
android:minHeight	Makes the TextView be at least this many pixels tall.
android:minLines	Makes the TextView be at least this many lines tall.
android:minWidth	Makes the TextView be at least this many pixels wide.
android:numeric	If set, specifies that this TextView has a numeric input method.
android:password	Whether the characters of the field are displayed as password dots instead of themselves.
android:phoneNumber	If set, specifies that this TextView has a phone number input method.
android:privateImeOptions	An addition content type description to supply to the input method attached to the text view, which is private to the implementation of the input method.
android:scrollHorizontally	Whether the text is allowed to be wider than the view (and therefore can be scrolled horizontally).
android:selectAllOnFocus	If the text is selectable, select it all when the view takes focus.
android:shadowColor	Place a blurred shadow of text underneath the text, drawn with the specified color.
android:shadowDx	Horizontal offset of the text shadow.
android:shadowDy	Vertical offset of the text shadow.
android:shadowRadius	Blur radius of the text shadow.
android:singleLine	Constrains the text to a single horizontally scrolling line instead of letting it wrap onto multiple lines, and advances focus instead of inserting a newline when you press the enter key.
android:text	Text to display.
android:textAllCaps	Present the text in ALL CAPS.
android:textAppearance	Base text color, typeface, size, and style.
android:textColor	Text color.
android:textColorHighlight	Color of the text selection highlight.
android:textColorHint	Color of the hint text.

android:textColorLink	Text color for links.
android:textCursorDrawable	Reference to a drawable that will be drawn under the insertion cursor.
android:textFontWeight	Weight for the font used in the TextView.
android:textIsSelectable	Indicates that the content of a non-editable text can be selected.
android:textScaleX	Sets the horizontal scaling factor for the text.
android:textSelectHandle	Reference to a drawable that will be used to display a text selection anchor for positioning the cursor within text.
android:textSelectHandleLeft	Reference to a drawable that will be used to display a text selection anchor on the left side of a selection region.
android:textSelectHandleRight	Reference to a drawable that will be used to display a text selection anchor on the right side of a selection region.
android:textSize	Size of the text.
android:textStyle	Style (normal, bold, italic, bold|italic) for the text.
android:typeface	Typeface (normal, sans, serif, monospace) for the text.
android:width	Makes the TextView be exactly this wide.

Following code snippet will demonstrate TextView:

```
<?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">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="This is a textview"
        android:id="@+id/label1"
        android:textSize="25sp"
        android:textStyle="bold"
        android:textAllCaps="true"
        android:textAlignment="center"
        android:layout_centerVertical="true"
        android:textColor="#FF4081" />

</RelativeLayout>
```

Above code will produce following output:

Figure 3-12. Output Demonstrating TextView

EditText

A user interface element for entering and modifying text. When you define an edit text widget, you must specify the **inputType** attribute. For example, for plain text input set inputType to "text".

Choosing the input type configures the keyboard type that is shown, acceptable characters, and appearance of the edit text. For example, if you want to accept a secret number, like a unique pin or serial number, you can set inputType to "numericPassword". An inputType of "numericPassword" results in an edit text that accepts numbers only, shows a numeric keyboard when focused, and masks the text that is entered for privacy.

We can use same set of attributes that we have used for TextView.

Following code snipped will demonstrate EditText:

```
<?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">

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="49dp"
        android:hint="Enter Text Here"
        android:inputType="text"
        android:textColor="#3F51B5"
        android:textColorHint="#3F51B5"
        android:textSize="20sp"
        android:textAlignment="center"
        android:tooltipText="Enter Text" />

</RelativeLayout>
```

Above code will produce following output:

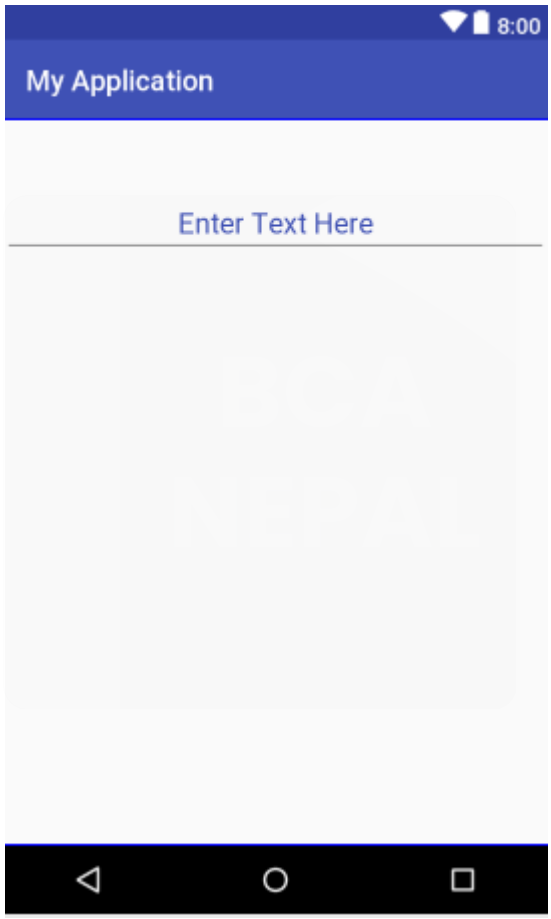

Figure 3-13. Output Demonstrating EditText

Button

It is a user interface element the user can tap or click to perform an action. Following are the attributes associated with Button.

Attributes Inherited from TextView – Almost all attributes of TextView can be used for Button.

Other Attributes:

android:accessibilityHeading	Whether or not this view is a heading for accessibility purposes.
android:accessibilityLiveRegion	Indicates to accessibility services whether the user should be notified when this view changes.
android:accessibilityPaneTitle	The title this view should present to accessibility as a pane title.
android:accessibilityTraversalAfter	Sets the id of a view after which this one is visited in accessibility traversal.
android:accessibilityTraversalBefore	Sets the id of a view before which this one is visited in accessibility traversal.
android:alpha	alpha property of the view, as a value between 0 (completely transparent) and 1 (completely opaque).
android:autofillHints	Describes the content of a view so that a autofill service can fill in the appropriate data.
android:autofilledHighlight	Drawable to be drawn over the view to mark it as autofilled May be a reference to another resource, in the form "@[+][*package*:]*type*/*name*" or a theme attribute in the form "?[*package*:]*type*/*name*".
android:background	A drawable to use as the background.
android:backgroundTint	Tint to apply to the background.
android:backgroundTintMode	Blending mode used to apply the background tint.
android:clickable	Defines whether this view reacts to click events.
android:contentDescription	Defines text that briefly describes content of the view.
android:contextClickable	Defines whether this view reacts to context click events.
android:defaultFocusHighlightEnabled	Whether this View should use a default focus highlight when it gets focused but doesn't have R.attr.state_focused defined in its background.
android:drawingCacheQuality	Defines the quality of translucent drawing caches.
android:duplicateParentState	When this attribute is set to true, the view gets its drawable state (focused, pressed, etc.) from its direct parent rather than from itself.

android:elevation	base z depth of the view.
android:fadeScrollbars	Defines whether to fade out scrollbars when they are not in use.
android:fadingEdgeLength	Defines the length of the fading edges.
android:filterTouchesWhenObscured	Specifies whether to filter touches when the view's window is obscured by another visible window.
android:fitsSystemWindows	Boolean internal attribute to adjust view layout based on system windows such as the status bar.
android:focusable	Controls whether a view can take focus.
android:focusableInTouchMode	Boolean that controls whether a view can take focus while in touch mode.
android:focusedByDefault	Whether this view is a default-focus view.
android:forceHasOverlappingRendering	Whether this view has elements that may overlap when drawn.
android:foreground	Defines the drawable to draw over the content.
android:foregroundGravity	Defines the gravity to apply to the foreground drawable.
android:foregroundTint	Tint to apply to the foreground.
android:foregroundTintMode	Blending mode used to apply the foreground tint.
android:hapticFeedbackEnabled	Boolean that controls whether a view should have haptic feedback enabled for events such as long presses.
android:id	Supply an identifier name for this view, to later retrieve it with View.findViewById() or Activity.findViewById().
android:importantForAccessibility	Describes whether or not this view is important for accessibility.
android:importantForAutofill	Hints the Android System whether the view node associated with this View should be included in a view structure used for autofill purposes.
android:importantForContentCapture	Hints the Android System whether the view node associated with this View should be use for content capture purposes.
android:isScrollContainer	Set this if the view will serve as a scrolling container, meaning that it can be resized to shrink its overall window so that there will be space for an input method.
android:keepScreenOn	Controls whether the view's window should keep the screen on while visible.
android:keyboardNavigationCluster	Whether this view is a root of a keyboard navigation cluster.
android:layerType	Specifies the type of layer backing this view.
android:layoutDirection	Defines the direction of layout drawing.
android:longClickable	Defines whether this view reacts to long click events.

android:minHeight	Defines the minimum height of the view.
android:minWidth	Defines the minimum width of the view.
android:nextClusterForward	Defines the next keyboard navigation cluster.
android:nextFocusDown	Defines the next view to give focus to when the next focus is View.FOCUS_DOWN If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusForward	Defines the next view to give focus to when the next focus is View.FOCUS_FORWARD If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusLeft	Defines the next view to give focus to when the next focus is View.FOCUS_LEFT.
android:nextFocusRight	Defines the next view to give focus to when the next focus is View.FOCUS_RIGHT If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusUp	Defines the next view to give focus to when the next focus is View.FOCUS_UP If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:onClick	Name of the method in this View's context to invoke when the view is clicked.
android:outlineAmbientShadowColor	Sets the color of the ambient shadow that is drawn when the view has a positive Z or elevation value.
android:outlineSpotShadowColor	Sets the color of the spot shadow that is drawn when the view has a positive Z or elevation value.
android:padding	Sets the padding, in pixels, of all four edges.
android:paddingBottom	Sets the padding, in pixels, of the bottom edge; see R.attr.padding.
android:paddingEnd	Sets the padding, in pixels, of the end edge; see R.attr.padding.
android:paddingHorizontal	Sets the padding, in pixels, of the left and right edges; see R.attr.padding.
android:paddingLeft	Sets the padding, in pixels, of the left edge; see R.attr.padding.
android:paddingRight	Sets the padding, in pixels, of the right edge; see R.attr.padding.
android:paddingStart	Sets the padding, in pixels, of the start edge; see R.attr.padding.

android:paddingTop	Sets the padding, in pixels, of the top edge; see R.attr.padding.
android:paddingVertical	Sets the padding, in pixels, of the top and bottom edges; see R.attr.padding.
android:requiresFadingEdge	Defines which edges should be faded on scrolling.
android:rotation	rotation of the view, in degrees.
android:rotationX	rotation of the view around the x axis, in degrees.
android:rotationY	rotation of the view around the y axis, in degrees.
android:saveEnabled	If false, no state will be saved for this view when it is being frozen.
android:scaleX	scale of the view in the x direction.
android:scaleY	scale of the view in the y direction.
android:screenReaderFocusable	Whether this view should be treated as a focusable unit by screen reader accessibility tools.
android:scrollIndicators	Defines which scroll indicators should be displayed when the view can be scrolled.
android:scrollX	The initial horizontal scroll offset, in pixels.
android:scrollY	The initial vertical scroll offset, in pixels.
android:scrollbarAlwaysDrawHorizontalTrack	Defines whether the horizontal scrollbar track should always be drawn.
android:scrollbarAlwaysDrawVerticalTrack	Defines whether the vertical scrollbar track should always be drawn.
android:scrollbarDefaultDelayBeforeFade	Defines the delay in milliseconds that a scrollbar waits before fade out.
android:scrollbarFadeDuration	Defines the delay in milliseconds that a scrollbar takes to fade out.
android:scrollbarSize	Sets the width of vertical scrollbars and height of horizontal scrollbars.
android:scrollbarStyle	Controls the scrollbar style and position.
android:scrollbarThumbHorizontal	Defines the horizontal scrollbar thumb drawable.
android:scrollbarThumbVertical	Defines the vertical scrollbar thumb drawable.
android:scrollbarTrackHorizontal	Defines the horizontal scrollbar track drawable.
android:scrollbarTrackVertical	Defines the vertical scrollbar track drawable.
android:scrollbars	Defines which scrollbars should be displayed on scrolling or not.
android:soundEffectsEnabled	Boolean that controls whether a view should have sound effects enabled for events such as clicking and touching.
android:stateListAnimator	Sets the state-based animator for the View.
android:tag	Supply a tag for this view containing a String, to be retrieved later with View.getTag() or searched for with View.findViewWithTag().
android:textAlignment	Defines the alignment of the text.
android:textDirection	Defines the direction of the text.
android:theme	Specifies a theme override for a view.

android:tooltipText	Defines text displayed in a small popup window on hover or long press.
android:transformPivotX	x location of the pivot point around which the view will rotate and scale.
android:transformPivotY	y location of the pivot point around which the view will rotate and scale.
android:transitionName	Names a View such that it can be identified for Transitions.
android:translationX	translation in x of the view.
android:translationY	translation in y of the view.
android:translationZ	translation in z of the view.
android:visibility	Controls the initial visibility of the view.

Following code snipped will demonstrate Button.

```
<?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">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit Button"
        android:textAllCaps="false"
        android:textSize="20sp"
        android:layout_centerInParent="true"
        android:textColor="#3F51B5" />

</RelativeLayout>
```

Above code will produce following output:

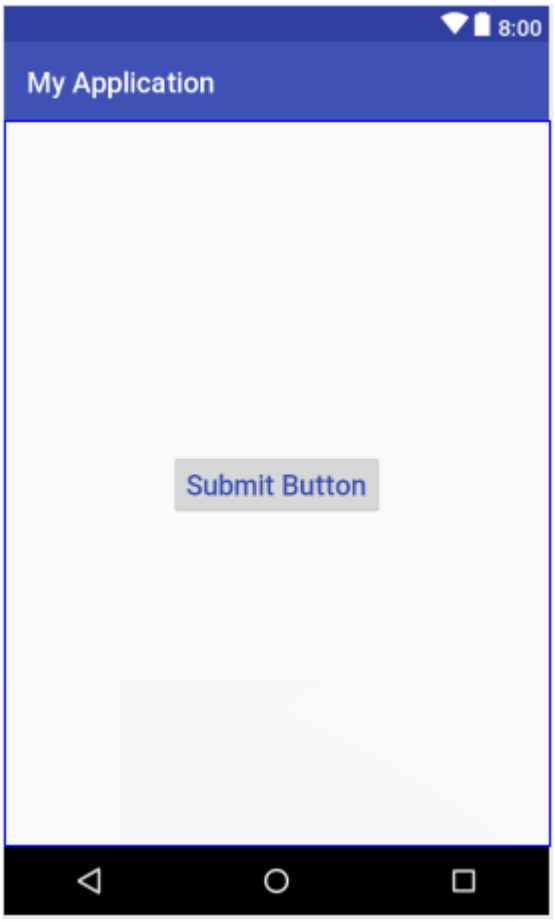

Figure 3-14. Output Demonstrating Button

CheckBox

A checkbox is a specific type of two-states button that can be either checked or unchecked. Following are the attributes associated with CheckBox.

Attributes Inherited from TextView – Almost all attributes of TextView can be used for Button.

Other Attributes:

android:accessibilityHeading	Whether or not this view is a heading for accessibility purposes.
android:accessibilityLiveRegion	Indicates to accessibility services whether the user should be notified when this view changes.
android:accessibilityPaneTitle	The title this view should present to accessibility as a pane title.
android:accessibilityTraversalAfter	Sets the id of a view after which this one is visited in accessibility traversal.
android:accessibilityTraversalBefore	Sets the id of a view before which this one is visited in accessibility traversal.

android:alpha	alpha property of the view, as a value between 0 (completely transparent) and 1 (completely opaque).
android:autofillHints	Describes the content of a view so that a autofill service can fill in the appropriate data.
android:autofilledHighlight	Drawable to be drawn over the view to mark it as autofilled May be a reference to another resource, in the form "@[+][*package*:]*type*/*name*" or a theme attribute in the form "?[*package*:]*type*/*name*".
android:background	A drawable to use as the background.
android:backgroundTint	Tint to apply to the background.
android:backgroundTintMode	Blending mode used to apply the background tint.
android:clickable	Defines whether this view reacts to click events.
android:contentDescription	Defines text that briefly describes content of the view.
android:contextClickable	Defines whether this view reacts to context click events.
android:defaultFocusHighlightEnabled	Whether this View should use a default focus highlight when it gets focused but doesn't have R.attr.state_focused defined in its background.
android:drawingCacheQuality	Defines the quality of translucent drawing caches.
android:duplicateParentState	When this attribute is set to true, the view gets its drawable state (focused, pressed, etc.) from its direct parent rather than from itself.
android:elevation	base z depth of the view.
android:fadeScrollbars	Defines whether to fade out scrollbars when they are not in use.
android:fadingEdgeLength	Defines the length of the fading edges.
android:filterTouchesWhenObscured	Specifies whether to filter touches when the view's window is obscured by another visible window.
android:fitsSystemWindows	Boolean internal attribute to adjust view layout based on system windows such as the status bar.
android:focusable	Controls whether a view can take focus.
android:focusableInTouchMode	Boolean that controls whether a view can take focus while in touch mode.
android:focusedByDefault	Whether this view is a default-focus view.
android:forceHasOverlappingRendering	Whether this view has elements that may overlap when drawn.
android:foreground	Defines the drawable to draw over the content.
android:foregroundGravity	Defines the gravity to apply to the foreground drawable.
android:foregroundTint	Tint to apply to the foreground.
android:foregroundTintMode	Blending mode used to apply the foreground tint.

android:hapticFeedbackEnabled	Boolean that controls whether a view should have haptic feedback enabled for events such as long presses.
android:id	Supply an identifier name for this view, to later retrieve it with View.findViewById() or Activity.findViewById().
android:importantForAccessibility	Describes whether or not this view is important for accessibility.
android:importantForAutofill	Hints the Android System whether the view node associated with this View should be included in a view structure used for autofill purposes.
android:importantForContentCapture	Hints the Android System whether the view node associated with this View should be use for content capture purposes.
android:isScrollContainer	Set this if the view will serve as a scrolling container, meaning that it can be resized to shrink its overall window so that there will be space for an input method.
android:keepScreenOn	Controls whether the view's window should keep the screen on while visible.
android:keyboardNavigationCluster	Whether this view is a root of a keyboard navigation cluster.
android:layerType	Specifies the type of layer backing this view.
android:layoutDirection	Defines the direction of layout drawing.
android:longClickable	Defines whether this view reacts to long click events.
android:minHeight	Defines the minimum height of the view.
android:minWidth	Defines the minimum width of the view.
android:nextClusterForward	Defines the next keyboard navigation cluster.
android:nextFocusDown	Defines the next view to give focus to when the next focus is View.FOCUS_DOWN If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusForward	Defines the next view to give focus to when the next focus is View.FOCUS_FORWARD If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusLeft	Defines the next view to give focus to when the next focus is View.FOCUS_LEFT.
android:nextFocusRight	Defines the next view to give focus to when the next focus is View.FOCUS_RIGHT If the reference refers to a view that does not exist or is part of a hierarchy

	that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusUp	Defines the next view to give focus to when the next focus is View.FOCUS_UP If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:onClick	Name of the method in this View's context to invoke when the view is clicked.
android:outlineAmbientShadowColor	Sets the color of the ambient shadow that is drawn when the view has a positive Z or elevation value.
android:outlineSpotShadowColor	Sets the color of the spot shadow that is drawn when the view has a positive Z or elevation value.
android:padding	Sets the padding, in pixels, of all four edges.
android:paddingBottom	Sets the padding, in pixels, of the bottom edge; see R.attr.padding.
android:paddingEnd	Sets the padding, in pixels, of the end edge; see R.attr.padding.
android:paddingHorizontal	Sets the padding, in pixels, of the left and right edges; see R.attr.padding.
android:paddingLeft	Sets the padding, in pixels, of the left edge; see R.attr.padding.
android:paddingRight	Sets the padding, in pixels, of the right edge; see R.attr.padding.
android:paddingStart	Sets the padding, in pixels, of the start edge; see R.attr.padding.
android:paddingTop	Sets the padding, in pixels, of the top edge; see R.attr.padding.
android:paddingVertical	Sets the padding, in pixels, of the top and bottom edges; see R.attr.padding.
android:requiresFadingEdge	Defines which edges should be faded on scrolling.
android:rotation	rotation of the view, in degrees.
android:rotationX	rotation of the view around the x axis, in degrees.
android:rotationY	rotation of the view around the y axis, in degrees.
android:saveEnabled	If false, no state will be saved for this view when it is being frozen.
android:scaleX	scale of the view in the x direction.
android:scaleY	scale of the view in the y direction.
android:screenReaderFocusable	Whether this view should be treated as a focusable unit by screen reader accessibility tools.
android:scrollIndicators	Defines which scroll indicators should be displayed when the view can be scrolled.
android:scrollX	The initial horizontal scroll offset, in pixels.

android:scrollY	The initial vertical scroll offset, in pixels.
android:scrollbarAlwaysDrawHorizontalTrack	Defines whether the horizontal scrollbar track should always be drawn.
android:scrollbarAlwaysDrawVerticalTrack	Defines whether the vertical scrollbar track should always be drawn.
android:scrollbarDefaultDelayBeforeFade	Defines the delay in milliseconds that a scrollbar waits before fade out.
android:scrollbarFadeDuration	Defines the delay in milliseconds that a scrollbar takes to fade out.
android:scrollbarSize	Sets the width of vertical scrollbars and height of horizontal scrollbars.
android:scrollbarStyle	Controls the scrollbar style and position.
android:scrollbarThumbHorizontal	Defines the horizontal scrollbar thumb drawable.
android:scrollbarThumbVertical	Defines the vertical scrollbar thumb drawable.
android:scrollbarTrackHorizontal	Defines the horizontal scrollbar track drawable.
android:scrollbarTrackVertical	Defines the vertical scrollbar track drawable.
android:scrollbars	Defines which scrollbars should be displayed on scrolling or not.
android:soundEffectsEnabled	Boolean that controls whether a view should have sound effects enabled for events such as clicking and touching.
android:stateListAnimator	Sets the state-based animator for the View.
android:tag	Supply a tag for this view containing a String, to be retrieved later with View.getTag() or searched for with View.findViewWithTag().
android:textAlignment	Defines the alignment of the text.
android:textDirection	Defines the direction of the text.
android:theme	Specifies a theme override for a view.
android:tooltipText	Defines text displayed in a small popup window on hover or long press.
android:transformPivotX	x location of the pivot point around which the view will rotate and scale.
android:transformPivotY	y location of the pivot point around which the view will rotate and scale.
android:transitionName	Names a View such that it can be identified for Transitions.
android:translationX	translation in x of the view.
android:translationY	translation in y of the view.
android:translationZ	translation in z of the view.
android:visibility	Controls the initial visibility of the view.

Following code snippet will demonstrate CheckBox:

```
<?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">

    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Option 1"
        android:textSize="20sp" />

    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Option 2"
        android:checked="true"
        android:textSize="20sp" />

</LinearLayout>
```

Above code will produce following output:

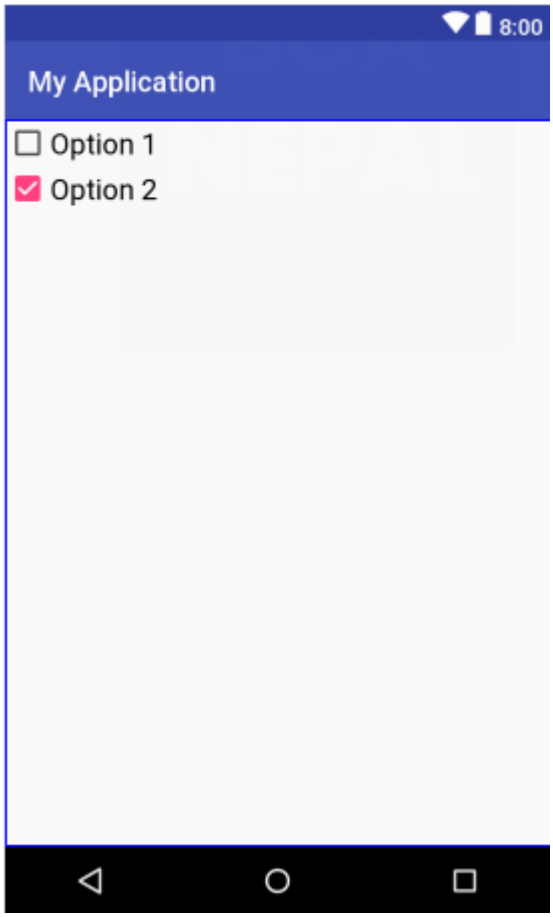

Figure 3-15. Output Demonstrating CheckBox

RadioButton

A radio button is a two-states button that can be either checked or unchecked. When the radio button is unchecked, the user can press or click it to check it. However, contrary to a CheckBox, a radio button cannot be unchecked by the user once checked.
Radio buttons are normally used together in a RadioGroup. When several radio buttons live inside a radio group, checking one radio button unchecks all the others.

Following are the attributes of RadioButton:

Attributes Inherited from TextView – Almost all attributes of TextView can be used for Button.

Other Attributes:

android:accessibilityHeading	Whether or not this view is a heading for accessibility purposes.
android:accessibilityLiveRegion	Indicates to accessibility services whether the user should be notified when this view changes.
android:accessibilityPaneTitle	The title this view should present to accessibility as a pane title.
android:accessibilityTraversalAfter	Sets the id of a view after which this one is visited in accessibility traversal.
android:accessibilityTraversalBefore	Sets the id of a view before which this one is visited in accessibility traversal.
android:alpha	alpha property of the view, as a value between 0 (completely transparent) and 1 (completely opaque).
android:autofillHints	Describes the content of a view so that a autofill service can fill in the appropriate data.
android:autofilledHighlight	Drawable to be drawn over the view to mark it as autofilled May be a reference to another resource, in the form "@[+][*package*:]*type*/*name*" or a theme attribute in the form "?[*package*:]*type*/*name*".
android:background	A drawable to use as the background.
android:backgroundTint	Tint to apply to the background.
android:backgroundTintMode	Blending mode used to apply the background tint.
android:clickable	Defines whether this view reacts to click events.
android:contentDescription	Defines text that briefly describes content of the view.
android:contextClickable	Defines whether this view reacts to context click events.
android:defaultFocusHighlightEnabled	Whether this View should use a default focus highlight when it gets focused but doesn't

	have R.attr.state_focused defined in its background.
android:drawingCacheQuality	Defines the quality of translucent drawing caches.
android:duplicateParentState	When this attribute is set to true, the view gets its drawable state (focused, pressed, etc.) from its direct parent rather than from itself.
android:elevation	base z depth of the view.
android:fadeScrollbars	Defines whether to fade out scrollbars when they are not in use.
android:fadingEdgeLength	Defines the length of the fading edges.
android:filterTouchesWhenObscured	Specifies whether to filter touches when the view's window is obscured by another visible window.
android:fitsSystemWindows	Boolean internal attribute to adjust view layout based on system windows such as the status bar.
android:focusable	Controls whether a view can take focus.
android:focusableInTouchMode	Boolean that controls whether a view can take focus while in touch mode.
android:focusedByDefault	Whether this view is a default-focus view.
android:forceHasOverlappingRendering	Whether this view has elements that may overlap when drawn.
android:foreground	Defines the drawable to draw over the content.
android:foregroundGravity	Defines the gravity to apply to the foreground drawable.
android:foregroundTint	Tint to apply to the foreground.
android:foregroundTintMode	Blending mode used to apply the foreground tint.
android:hapticFeedbackEnabled	Boolean that controls whether a view should have haptic feedback enabled for events such as long presses.
android:id	Supply an identifier name for this view, to later retrieve it with View.findViewById() or Activity.findViewById().
android:importantForAccessibility	Describes whether or not this view is important for accessibility.
android:importantForAutofill	Hints the Android System whether the view node associated with this View should be included in a view structure used for autofill purposes.
android:importantForContentCapture	Hints the Android System whether the view node associated with this View should be use for content capture purposes.
android:isScrollContainer	Set this if the view will serve as a scrolling container, meaning that it can be resized to shrink its overall window so that there will be space for an input method.

android:keepScreenOn	Controls whether the view's window should keep the screen on while visible.
android:keyboardNavigationCluster	Whether this view is a root of a keyboard navigation cluster.
android:layerType	Specifies the type of layer backing this view.
android:layoutDirection	Defines the direction of layout drawing.
android:longClickable	Defines whether this view reacts to long click events.
android:minHeight	Defines the minimum height of the view.
android:minWidth	Defines the minimum width of the view.
android:nextClusterForward	Defines the next keyboard navigation cluster.
android:nextFocusDown	Defines the next view to give focus to when the next focus is View.FOCUS_DOWN If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusForward	Defines the next view to give focus to when the next focus is View.FOCUS_FORWARD If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusLeft	Defines the next view to give focus to when the next focus is View.FOCUS_LEFT.
android:nextFocusRight	Defines the next view to give focus to when the next focus is View.FOCUS_RIGHT If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:nextFocusUp	Defines the next view to give focus to when the next focus is View.FOCUS_UP If the reference refers to a view that does not exist or is part of a hierarchy that is invisible, a RuntimeException will result when the reference is accessed.
android:onClick	Name of the method in this View's context to invoke when the view is clicked.
android:outlineAmbientShadowColor	Sets the color of the ambient shadow that is drawn when the view has a positive Z or elevation value.
android:outlineSpotShadowColor	Sets the color of the spot shadow that is drawn when the view has a positive Z or elevation value.
android:padding	Sets the padding, in pixels, of all four edges.
android:paddingBottom	Sets the padding, in pixels, of the bottom edge; see R.attr.padding.
android:paddingEnd	Sets the padding, in pixels, of the end edge; see R.attr.padding.

android:paddingHorizontal	Sets the padding, in pixels, of the left and right edges; see R.attr.padding.
android:paddingLeft	Sets the padding, in pixels, of the left edge; see R.attr.padding.
android:paddingRight	Sets the padding, in pixels, of the right edge; see R.attr.padding.
android:paddingStart	Sets the padding, in pixels, of the start edge; see R.attr.padding.
android:paddingTop	Sets the padding, in pixels, of the top edge; see R.attr.padding.
android:paddingVertical	Sets the padding, in pixels, of the top and bottom edges; see R.attr.padding.
android:requiresFadingEdge	Defines which edges should be faded on scrolling.
android:rotation	rotation of the view, in degrees.
android:rotationX	rotation of the view around the x axis, in degrees.
android:rotationY	rotation of the view around the y axis, in degrees.
android:saveEnabled	If false, no state will be saved for this view when it is being frozen.
android:scaleX	scale of the view in the x direction.
android:scaleY	scale of the view in the y direction.
android:screenReaderFocusable	Whether this view should be treated as a focusable unit by screen reader accessibility tools.
android:scrollIndicators	Defines which scroll indicators should be displayed when the view can be scrolled.
android:scrollX	The initial horizontal scroll offset, in pixels.
android:scrollY	The initial vertical scroll offset, in pixels.
android:scrollbarAlwaysDrawHorizontalTrack	Defines whether the horizontal scrollbar track should always be drawn.
android:scrollbarAlwaysDrawVerticalTrack	Defines whether the vertical scrollbar track should always be drawn.
android:scrollbarDefaultDelayBeforeFade	Defines the delay in milliseconds that a scrollbar waits before fade out.
android:scrollbarFadeDuration	Defines the delay in milliseconds that a scrollbar takes to fade out.
android:scrollbarSize	Sets the width of vertical scrollbars and height of horizontal scrollbars.
android:scrollbarStyle	Controls the scrollbar style and position.
android:scrollbarThumbHorizontal	Defines the horizontal scrollbar thumb drawable.
android:scrollbarThumbVertical	Defines the vertical scrollbar thumb drawable.
android:scrollbarTrackHorizontal	Defines the horizontal scrollbar track drawable.
android:scrollbarTrackVertical	Defines the vertical scrollbar track drawable.
android:scrollbars	Defines which scrollbars should be displayed on scrolling or not.

android:soundEffectsEnabled	Boolean that controls whether a view should have sound effects enabled for events such as clicking and touching.
android:stateListAnimator	Sets the state-based animator for the View.
android:tag	Supply a tag for this view containing a String, to be retrieved later with View.getTag() or searched for with View.findViewWithTag().
android:textAlignment	Defines the alignment of the text.
android:textDirection	Defines the direction of the text.
android:theme	Specifies a theme override for a view.
android:tooltipText	Defines text displayed in a small popup window on hover or long press.
android:transformPivotX	x location of the pivot point around which the view will rotate and scale.
android:transformPivotY	y location of the pivot point around which the view will rotate and scale.
android:transitionName	Names a View such that it can be identified for Transitions.
android:translationX	translation in x of the view.
android:translationY	translation in y of the view.
android:translationZ	translation in z of the view.
android:visibility	Controls the initial visibility of the view.

Following code will demonstrate RadioButton.

```
<?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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Gender:"
        android:textSize="20sp"/>

    <RadioGroup
        android:layout_width="match_parent"
        android:orientation="horizontal"
        android:layout_height="wrap_content">

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Male"
        android:textSize="20sp"/>

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Female"
        android:textSize="20sp"/>

    </RadioGroup>

</LinearLayout>
```

Above code will produce following output:

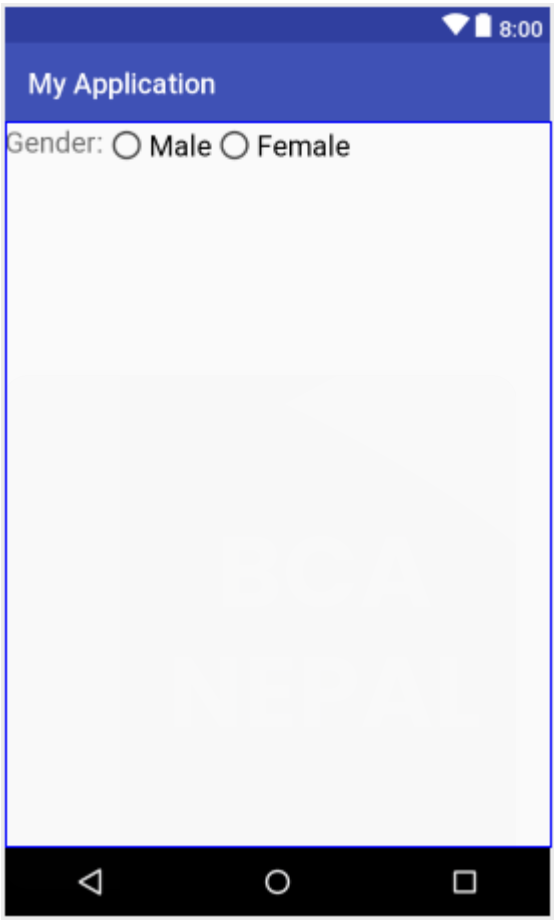

Figure 3-16. Output Demonstrating Button

Spinner

It is a view that displays one child at a time and lets the user pick among them. The items in the Spinner come from the String array or Adapter associated with this view.
Following are some important attributes associated with spinner.

android:dropDownHorizontalOffset	Amount of pixels by which the drop down should be offset horizontally.
android:dropDownSelector	List selector to use for spinnerMode="dropdown" display.
android:dropDownVerticalOffset	Amount of pixels by which the drop down should be offset vertically.
android:dropDownWidth	Width of the dropdown in spinnerMode="dropdown".

android:gravity	Gravity setting for positioning the currently selected item.
android:popupBackground	Background drawable to use for the dropdown in spinnerMode="dropdown".
android:prompt	The prompt to display when the spinner's dialog is shown.
android:spinnerMode	Display mode for spinner options.
android:entries	Used for loading string array.

Following example will demonstrate Spinner. In this example for collection of items to be loaded in Spinner we are creating string array in **strings.xml** file. We can also load items collection programmatically.

strings.xml

```
<string-array name="spinner_items">
    <item>Item 1</item>
    <item>Item 2</item>
    <item>Item 3</item>
    <item>Item 4</item>
</string-array>
```

```
<?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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Items:"
        android:textSize="20sp"/>

    <Spinner
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:entries="@array/spinner_items"
        android:spinnerMode="dropdown"/>

</LinearLayout>
```

Above code will produce following output:

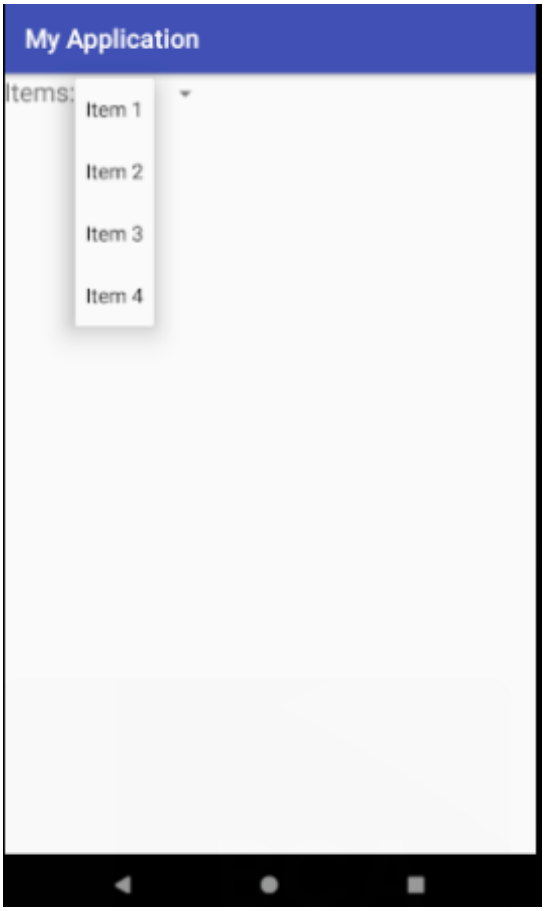

Figure 3-17. Output Demonstrating Spinner

We can also array of elements in Spinner programmatically using adapter as follows:

```
Spinner spinner=findViewById(R.id.spinner);
String items[]={"Item 1","Item 2","Item 3","Item 4"};

ArrayAdapter<String> arrayAdapter=new ArrayAdapter<String>
(this,R.layout.support_simple_spinner_dropdown_item,items);
spinner.setAdapter(arrayAdapter);
```

Event Handling

Events are a useful way to collect data about a user's interaction with interactive components of Applications. Like button presses or screen touch etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.
There are following three concepts related to Android Event Management −

- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	**OnClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	**OnLongClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()** This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	**OnFocusChangeListener()** This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()** This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	**OnMenuItemClickListener()** This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	**onCreateContextMenuItemListener()** This is called when the context menu is being built(as the result of a sustained "long click)

Following example will demonstrate **onClick()** event. For this we are creating a **Button** and while clicking **Button** a **Toast** message will be displayed on screen.

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click Me"
    android:layout_centerInParent="true"
    android:id="@+id/button1"
    />
```

```
Button btn=findViewById(R.id.button1);
//adding click event and listener
btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //displaying message
        Toast.makeText(getApplicationContext(),"Button Clicked !",
              Toast.LENGTH_SHORT).show();
    }
});
```

Above code will produce following output:

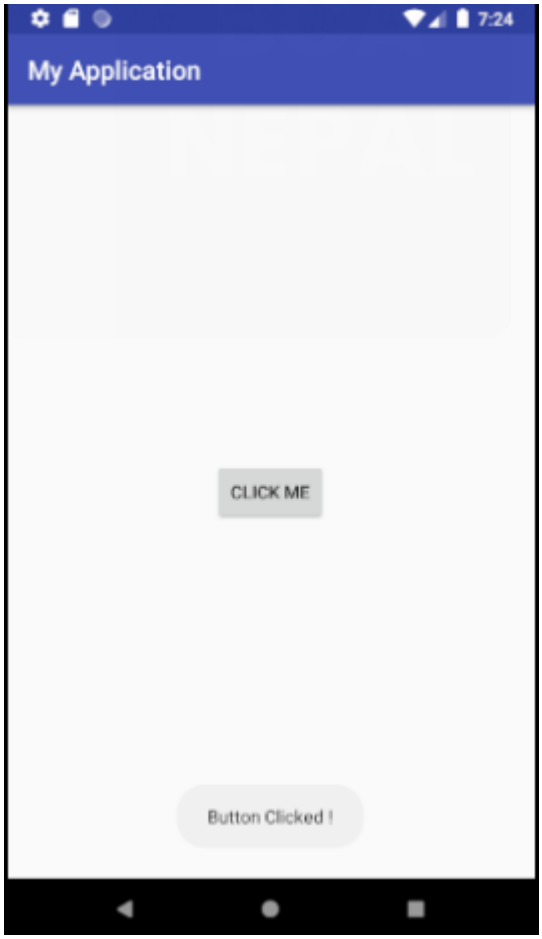

Figure 3-18. Output Demonstrating onCLick() Event

Working with String and String Array

A string resource provides text strings for your application with optional text styling and formatting. **String is a XML resource that provides a single string while String array is a XML resource that provides an array of strings.**

Using String

XML file saved at res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="semester">Sixth </string>
</resources>
```

This layout XML applies a string to a View as follows:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/semester" />
```

This application code retrieves a string as follows:

```
String string = getString(R.string.semester);
```

Using String Array

XML file saved at res/values/strings.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="semesters">
        <item>First</item>
        <item>Second</item>
        <item>Third</item>
        <item>Fourth</item>
    </string-array>
</resources>
```

This layout XML applies a string to a View as follows:

```
<Spinner
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:entries="@array/semesters"/>
```

This application code retrieves a string array as follows:

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.semesters);
```

Working with Colors

A color resource provides set of colors that you can use all over your application.

XML file saved at res/values/colors.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

This layout XML applies a colors to a View as follows:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txt"
    android:textColor="@color/colorPrimary"
    android:background="@color/colorAccent"
   />
```

Working with Drawable

A drawable resource is a general concept for a graphic that can be drawn to the screen and which you can retrieve with APIs such as getDrawable(int) or apply to another XML resource with attributes such as android:drawable and android:icon.
A bitmap file is a .png, .jpg, or .gif file. Android creates a Drawable resource for any of these files when you save them in the res/drawable/ directory.

With an image saved at res/drawable/myimage.png, this layout XML applies the image to a View:

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/myimage" />
```

The following application code retrieves the image as a Drawable:

```
Resources res = getResources();
Drawable drawable = ResourcesCompat.getDrawable(res,
R.drawable.myimage, null);
```

Adding Icon to the Project

With an image saved at res/drawable/myicon.png, you can add this image as icon to the project in **android manifest file**:

```
android:icon="@drawable/myicon"
```

Exercise

1. What do you mean by layout? List and explain different types of layout used in android with their major attributes.
2. Differentiate LinearLayout and RelativeLayout with example.
3. Differentiate RelativeLayout and ConstraintLayout with example.
4. Compare AbsoluteLayout with other types of layout.
5. Explain any five widgets with their attributes in detail.
6. What do you mean by event? Explain event handling in detail.
7. How can you use string and string array in android? Explain.
8. How can you use color in android? Explain.
9. Differentiate string and string array with example.
10. Explain the procedure for adding and displaying image in imageview. Also write code snippet to add icon to your project.
11. Design a signup form using Relative, Linear, Absolute and Constraint Layout.
12. Design a simple UI for Book Entry in library.
13. Design a simple UI for patient registration in hospital.
14. Display the following data using TableLayout.

Student Id	**Name**	**Address**	**Gender**
001	Ram Sharma	Birtamode	Male
002	Gita Sharma	Kathmandu	Female

15. Design a simple calculator UI using TableLayout.