İçindekiler

[UI widgets 3](#_Toc191835177)

[**1. Layout Widgets** 3](#_Toc191835178)

[Relative Layout 3](#_Toc191835179)

[Constraint Layout 3](#_Toc191835180)

[Required Parameters 4](#_Toc191835181)

[Optional Parameters 4](#_Toc191835182)

[Example XML Code 6](#_Toc191835183)

[Explanation of the Example 7](#_Toc191835184)

[Table Layout 7](#_Toc191835185)

[Frame Layout 8](#_Toc191835186)

[android:layout\_gravity in FrameLayout 8](#_Toc191835187)

[Grid Layout 8](#_Toc191835188)

[Key Features of GridLayout 8](#_Toc191835189)

[**Key Attributes for GridLayout** 9](#_Toc191835190)

[**2. Advanced Widgets** 9](#_Toc191835191)

[**3. Dialog Widgets** 9](#_Toc191835192)

[**4. Navigation Widgets** 10](#_Toc191835193)

[**5. Input Widgets** 10](#_Toc191835194)

[**6. Material Design Widgets** 10](#_Toc191835195)

[**7. Animation and Transition Widgets** 10](#_Toc191835196)

[**8. Custom Widgets** 10](#_Toc191835197)

[**9. Other Widgets** 11](#_Toc191835198)

[Basic Widgets 12](#_Toc191835199)

[TextView: Displays text to the user. 12](#_Toc191835200)

[Required Parameters 12](#_Toc191835201)

[Optional Parameters 12](#_Toc191835202)

[Example of a Fully Customized TextView 14](#_Toc191835203)

[Paste screen capture of Fully Customized Text View 15](#_Toc191835204)

[**EditText**: Allows users to input text. 15](#_Toc191835205)

[**Button**: A clickable button that performs an action. 15](#_Toc191835206)

[**ImageButton**: A button with an image instead of text. 15](#_Toc191835207)

[**ImageView**: Displays an image or drawable. 15](#_Toc191835208)

[**CheckBox**: A checkbox for binary choices. 15](#_Toc191835209)

[**RadioButton**: Used in groups for single-selection choices. 15](#_Toc191835210)

[**ToggleButton**: A button with an on/off state. 15](#_Toc191835211)

[**Switch**: A modern on/off toggle switch. 15](#_Toc191835212)

# UI widgets

**UI widgets** (or components) available in Android for building user interfaces. These widgets are part of the Android framework and can be used in XML layout files or programmatically in Java/Kotlin code. Below is a list of commonly used UI widgets in Android:

**1. Layout Widgets**

## Relative Layout

Positions child views relative to each other or the parent.

|  |  |
| --- | --- |
| 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\_alignWithParentIf Missing | 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. |

<RelativeLayout 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">  
  
  
   
</RelativeLayout>

## Constraint Layout

A flexible layout for creating complex UIs with constraints.

In a **ConstraintLayout**, UI elements (views) are positioned and sized based on **constraints** that define their relationship to other views or the parent layout. Unlike other layouts (e.g., LinearLayout or RelativeLayout), ConstraintLayout requires you to define constraints for each view to ensure proper placement and behavior.

<?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"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:padding="16dp">

</androidx.constraintlayout.widget.ConstraintLayout>

Below are the **required** and **optional** parameters for UI elements in a ConstraintLayout:

### ****Required Parameters****

1. **android:layout\_width**:
   * Specifies the width of the view.
   * Common values: wrap\_content, match\_constraint (previously 0dp), or a specific dimension (e.g., 100dp).
2. **android:layout\_height**:
   * Specifies the height of the view.
   * Common values: wrap\_content, match\_constraint (previously 0dp), or a specific dimension (e.g., 50dp).
3. **Constraints**:
   * At least one horizontal and one vertical constraint must be defined for each view.
   * Constraints are defined using attributes like:
     + app:layout\_constraintTop\_toTopOf
     + app:layout\_constraintBottom\_toBottomOf
     + app:layout\_constraintStart\_toStartOf
     + app:layout\_constraintEnd\_toEndOf
     + app:layout\_constraintTop\_toBottomOf
     + app:layout\_constraintStart\_toEndOf, etc.

### ****Optional Parameters****

These parameters are not required but are commonly used to customize the behavior and appearance of views in a ConstraintLayout.

#### **Positioning and Constraints**

1. **Bias**:
   * Adjusts the position of a view within the available space.
   * Attributes:
     + app:layout\_constraintHorizontal\_bias (e.g., 0.5 for center, 0.0 for start, 1.0 for end).
     + app:layout\_constraintVertical\_bias (e.g., 0.5 for center, 0.0 for top, 1.0 for bottom).
2. **Chains**:
   * Groups views together to distribute space horizontally or vertically.
   * Attributes:
     + app:layout\_constraintHorizontal\_chainStyle (e.g., spread, spread\_inside, packed).
     + app:layout\_constraintVertical\_chainStyle.
3. **Aspect Ratio**:
   * Maintains a specific aspect ratio for the view.
   * Attribute: app:layout\_constraintDimensionRatio (e.g., 1:1 for square, 16:9 for widescreen).
4. **Margin**:
   * Adds space between the view and its constraints.
   * Attributes:
     + android:layout\_marginStart
     + android:layout\_marginEnd
     + android:layout\_marginTop
     + android:layout\_marginBottom
     + android:layout\_margin
5. **Gone Margin**:
   * Adjusts margins when a referenced view is GONE.
   * Attributes:
     + app:layout\_goneMarginStart
     + app:layout\_goneMarginEnd
     + app:layout\_goneMarginTop
     + app:layout\_goneMarginBottom

#### **Sizing**

1. **match\_constraint Behavior**:
   * When layout\_width or layout\_height is set to match\_constraint (previously 0dp), the view expands to fill the available space based on constraints.
   * Additional attributes:
     + app:layout\_constraintWidth\_min (minimum width).
     + app:layout\_constraintWidth\_max (maximum width).
     + app:layout\_constraintHeight\_min (minimum height).
     + app:layout\_constraintHeight\_max (maximum height).
2. **Percentage Width/Height**:
   * Sets the width or height as a percentage of the parent or another view.
   * Attributes:
     + app:layout\_constraintWidth\_percent (e.g., 0.5 for 50%).
     + app:layout\_constraintHeight\_percent.

#### **Alignment**

1. **Baseline Alignment**:
   * Aligns the baseline of a TextView with another view.
   * Attribute: app:layout\_constraintBaseline\_toBaselineOf.
2. **Circular Positioning**:
   * Positions a view relative to another view in a circular manner.
   * Attributes:
     + app:layout\_constraintCircle
     + app:layout\_constraintCircleRadius
     + app:layout\_constraintCircleAngle

### ****Example XML Code****

Here’s an example of a ConstraintLayout with a TextView and two Button widgets:

<?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"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:padding="16dp">

<!-- TextView -->

<TextView

android:id="@+id/myTextView"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:text="Hello, World!"

android:textSize="24sp"

android:textColor="#000000"

android:textStyle="bold"

android:gravity="center"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

android:layout\_marginTop="16dp"/>

<!-- Button 1 -->

<Button

android:id="@+id/button1"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:text="Button 1"

android:textSize="18sp"

android:backgroundTint="#6200EE"

android:textColor="#FFFFFF"

app:layout\_constraintTop\_toBottomOf="@id/myTextView"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintEnd\_toStartOf="@id/button2"

android:layout\_marginTop="24dp"

android:layout\_marginEnd="8dp"/>

<!-- Button 2 -->

<Button

android:id="@+id/button2"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:text="Button 2"

android:textSize="18sp"

android:backgroundTint="#03DAC5"

android:textColor="#000000"

app:layout\_constraintTop\_toBottomOf="@id/myTextView"

app:layout\_constraintStart\_toEndOf="@id/button1"

app:layout\_constraintEnd\_toEndOf="parent"

android:layout\_marginTop="24dp"

android:layout\_marginStart="8dp"/>

</androidx.constraintlayout.widget.ConstraintLayout>

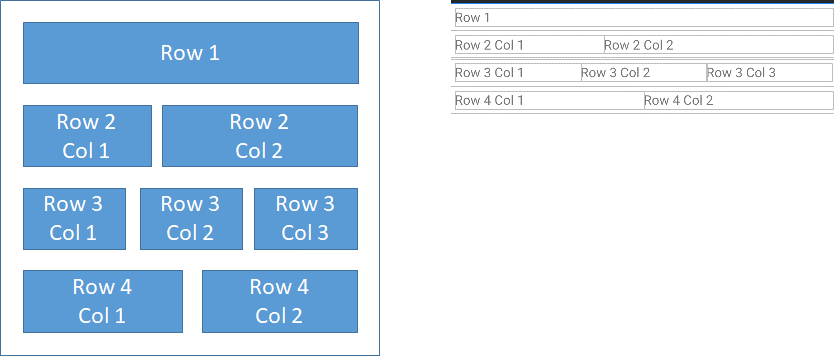
### ****Explanation of the Example****

1. **TextView**:
   * Width is set to 0dp (match constraints), so it spans the entire width of the parent.
   * Centered horizontally using app:layout\_constraintStart\_toStartOf and app:layout\_constraintEnd\_toEndOf.
   * Positioned at the top of the parent with app:layout\_constraintTop\_toTopOf.
2. **Button 1**:
   * Width is set to 0dp (match constraints), so it shares space with Button 2.
   * Positioned below the TextView with app:layout\_constraintTop\_toBottomOf.
   * Aligned to the start of the parent and end of Button 2.
3. **Button 2**:
   * Width is set to 0dp (match constraints), so it shares space with Button 1.
   * Positioned below the TextView with app:layout\_constraintTop\_toBottomOf.
   * Aligned to the end of the parent and start of Button 1.

## Table Layout

Arranges child views in rows and columns (like a table).

TableLayout is used to group views into rows and columns. Each row may have different number of columns.  A Table will have as many columns as the row with the most cells. As an example, let’s design a table with four rows as follows



TableLayout headers are given as follows: Each row is defined as a table row with the TableRow command.

<?xml version="1.0" encoding="utf-8"?>  
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 >  
 <TableRow android:background="@color/white" android:padding="5dp">

</TableRow>

</TableLayout>

## Frame Layout

Stacks child views on top of each other.

Android **Framelayout** is a ViewGroup subclass that is used to specify the position of multiple views placed on top of each other to represent a single view screen. FrameLayout simply blocks a particular area on the screen to display a single view. Here, all the child views or elements are added in stack format means the most recently added child will be shown on the top of the screen. But, we can add multiple children’s views and control their positions only by using gravity attributes in FrameLayout.

### android:layout\_gravity****in****FrameLayout

* **Purpose**: Aligns the **child views** within the FrameLayout.
* **Usage**: This is the primary parameter used in FrameLayout to position child views.
* **Values**:
  + top: Aligns the child view to the top of the FrameLayout.
  + bottom: Aligns the child view to the bottom of the FrameLayout.
  + left or start: Aligns the child view to the left (or start) of the FrameLayout.
  + right or end: Aligns the child view to the right (or end) of the FrameLayout.
  + center: Centers the child view both horizontally and vertically.
  + center\_vertical: Centers the child view vertically.
  + center\_horizontal: Centers the child view horizontally.
  + Combinations: You can combine values like top|left, bottom|right, etc.

For example, the following is added to widgets.

* android:layout\_gravity="top|left" />

<?xml version="1.0" encoding="utf-8"?>  
<FrameLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_marginTop="64dp"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent">  
  
   
</FrameLayout>

## Grid Layout

Arranges child views in a grid, allows you to arrange views in a grid of rows and columns. Unlike TableLayout, which requires the use of TableRow,  GridLayout lets you place views directly into specific cells, making it more flexible and easier to use for grid-based designs.

<?xml version="1.0" encoding="utf-8"?>  
<GridLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:rowCount="3"  
 android:columnCount="3"  
 android:padding="16dp"> <!-- Padding around the GridLayout -->  
  
  
   
</GridLayout>

### Key Features of GridLayout

1. **Rows and Columns**:
   * Define the number of rows and columns using android:rowCount and android:columnCount.
   * Example: android:rowCount="3" and android:columnCount="3" creates a 3x3 grid.
2. **Cell Placement**:
   * Each view is placed in a specific cell using android:layout\_row and android:layout\_column (0-based indexing).
   * Example: android:layout\_row="1" and android:layout\_column="2" places the view in the second row and third column.
3. **Spanning Cells**:
   * A view can span multiple rows or columns using android:layout\_rowSpan and android:layout\_columnSpan.
   * Example: android:layout\_columnSpan="2" makes the view span two columns.
4. **Equal Distribution**:
   * You can distribute space evenly among rows or columns using android:layout\_weight or android:layout\_gravity.
5. **No Nesting Required**:
   * Unlike LinearLayout or RelativeLayout, you don’t need to nest layouts to create complex grids.

**Key Attributes for GridLayout**

| **Attribute** | **Description** |
| --- | --- |
| android:rowCount | Number of rows in the grid. |
| android:columnCount | Number of columns in the grid. |
| android:layout\_row | Specifies the row index for a view (0-based). |
| android:layout\_column | Specifies the column index for a view (0-based). |
| android:layout\_rowSpan | Specifies how many rows the view should span. |
| android:layout\_columnSpan | Specifies how many columns the view should span. |
| android:layout\_gravity | Specifies the alignment of the view within its cell (e.g., center, fill). |

## Linear Layout

LinearLayout aligns all views in a single direction, vertically or horizontally. The layout direction can be specified with the android:orientation attribute.

All children of a LinearLayout are stacked one after the other. Accordingly, a vertical list has **one** child per row, no matter how wide they are. Similarly, 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.

|  |  |
| --- | --- |
| **android:id** | This is the ID which uniquely identifies the layout. |
| **android:baselineAligned** | A boolean value, either "true" or "false" that 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. |
| **android:divider** | This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb". |
| **android:gravity** | Specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center\_vertical, center\_horizontal etc. |
| **android:orientation** | Specifies the direction of arrangement: "horizontal" for a row, "vertical" for a column. |
| **android:weightSum** | Sum up of child weight |

The linear layout xml file has the following text.

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="fill\_parent"  
 android:layout\_height="fill\_parent"  
 android:orientation="vertical" >  
…  
   
</LinearLayout>

**2. Advanced Widgets**

* **RecyclerView**: Displays a scrollable list of items efficiently.
* **ListView**: Displays a scrollable list of items (older alternative to RecyclerView).
* **Spinner**: A dropdown menu for selecting one item from a list.
* **ProgressBar**: Shows progress (determinate or indeterminate).
* **SeekBar**: A draggable slider for selecting a value within a range.
* **RatingBar**: Allows users to rate something with stars.
* **WebView**: Displays web pages or HTML content.
* **VideoView**: Plays video files.
* **MapView**: Displays Google Maps (requires Google Play Services).

**3. Dialog Widgets**

* **AlertDialog**: A popup dialog for alerts, confirmations, or inputs.
* **DatePickerDialog**: A dialog for selecting a date.
* **TimePickerDialog**: A dialog for selecting a time.
* **ProgressDialog**: A dialog showing progress (deprecated in favor of ProgressBar).

**4. Navigation Widgets**

* **BottomNavigationView**: A bottom navigation bar for switching between fragments or activities.
* **NavigationView**: A side navigation drawer.
* **TabLayout**: Tabs for switching between views or fragments.
* **ViewPager**: Allows swiping between views or fragments.
* **ViewPager2**: An improved version of ViewPager.

**5. Input Widgets**

* **AutoCompleteTextView**: A text input with auto-complete suggestions.
* **MultiAutoCompleteTextView**: Similar to AutoCompleteTextView but supports multiple entries.
* **NumberPicker**: A widget for selecting a number.
* **TextInputLayout**: A wrapper for EditText that adds floating labels and error messages (part of Material Design).

**6. Material Design Widgets**

* **FloatingActionButton (FAB)**: A circular button for primary actions.
* **Snackbar**: A lightweight feedback message at the bottom of the screen.
* **CardView**: A container with rounded corners and elevation for a card-like appearance.
* **Chip**: A compact element for input, attribute, or action.
* **BottomAppBar**: A bottom app bar with a FAB.
* **MaterialButton**: A customizable button with Material Design styling.
* **TextInputEditText**: A subclass of EditText for use with TextInputLayout.

**7. Animation and Transition Widgets**

* **MotionLayout**: A layout for complex animations and transitions.
* **TransitionManager**: Manages transitions between layouts.

**8. Custom Widgets**

* Developers can create custom widgets by extending existing widgets or creating entirely new ones.

**9. Other Widgets**

* **SurfaceView**: A view for rendering graphics or video.
* **TextureView**: A view for displaying a content stream (e.g., video or camera feed).
* **SearchView**: A widget for implementing search functionality.
* **Toolbar**: A replacement for the ActionBar with more customization options.
* **Space**: A lightweight view for adding gaps between widgets.
* **ViewStub**: A placeholder for lazily inflating views.

## Basic Widgets

* **TextView**: Displays text to the user.
* **EditText**: Allows users to input text.
* **Button**: A clickable button that performs an action.
* **ImageButton**: A button with an image instead of text.
* **ImageView**: Displays an image or drawable.
* **CheckBox**: A checkbox for binary choices.
* **RadioButton**: Used in groups for single-selection choices.
* **ToggleButton**: A button with an on/off state.
* **Switch**: A modern on/off toggle switch.

## TextView: Displays text to the user.

**Prompt**: What are the parameters that have to be defined as well as optional parameters for textView

The TextView widget in Android is one of the most commonly used UI components for displaying text. When defining a TextView in XML or programmatically, there are several parameters (attributes) you can set to customize its appearance and behavior. These parameters can be categorized into **required** and **optional** attributes.

### ****Required Parameters****

In most cases, you **must** define the following parameters for a TextView to display properly:

1. **android:layout\_width**:
   * Specifies the width of the TextView.
   * Common values: wrap\_content, match\_parent, or a specific dimension (e.g., 100dp).
2. **android:layout\_height**:
   * Specifies the height of the TextView.
   * Common values: wrap\_content, match\_parent, or a specific dimension (e.g., 50dp).

<TextView

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

### ****Optional Parameters****

These parameters are not required but are commonly used to customize the TextView:

#### **Text and Appearance**

1. **android:text**:
   * Sets the text to display.
   * Example: android:text="Hello, World!".
2. **android:textSize**:
   * Sets the size of the text.
   * Example: android:textSize="18sp".
3. **android:textColor**:
   * Sets the color of the text.
   * Example: android:textColor="#FF0000" (red).
4. **android:textStyle**:
   * Sets the style of the text (e.g., bold, italic).
   * Example: android:textStyle="bold|italic".
5. **android:fontFamily**:
   * Sets the font family for the text.
   * Example: android:fontFamily="sans-serif".
6. **android:gravity**:
   * Aligns the text within the TextView (e.g., center, left, right).
   * Example: android:gravity="center".
7. **android:textAlignment**:
   * Aligns the text relative to the TextView's layout.
   * Example: android:textAlignment="center".
8. **android:lineSpacingExtra**:
   * Adds extra spacing between lines of text.
   * Example: android:lineSpacingExtra="5dp".
9. **android:lineSpacingMultiplier**:
   * Multiplies the line spacing by a factor.
   * Example: android:lineSpacingMultiplier="1.5".
10. **android:maxLines**:
    * Limits the number of lines the text can occupy.
    * Example: android:maxLines="2".
11. **android:ellipsize**:
    * Adds an ellipsis (...) when the text overflows.
    * Example: android:ellipsize="end".
12. **android:autoLink**:
    * Automatically detects and links URLs, emails, etc.
    * Example: android:autoLink="web".
13. **android:inputType**:
    * Specifies the type of input (useful for EditText, but can be used with TextView).
    * Example: android:inputType="textEmailAddress".

#### **Padding and Margins**

1. **android:padding**:
   * Adds padding inside the TextView.
   * Example: android:padding="10dp".
2. **android:paddingLeft**, **android:paddingRight**, **android:paddingTop**, **android:paddingBottom**:
   * Adds padding to specific sides.
   * Example: android:paddingLeft="5dp".
3. **android:layout\_margin**:
   * Adds margin outside the TextView.
   * Example: android:layout\_margin="10dp".
4. **android:layout\_marginLeft**, **android:layout\_marginRight**, **android:layout\_marginTop**, **android:layout\_marginBottom**:
   * Adds margin to specific sides.
   * Example: android:layout\_marginTop="5dp".

#### **Background and Drawables**

1. **android:background**:
   * Sets the background color or drawable.
   * Example: android:background="#FFFFFF" (white).
2. **android:drawableLeft**, **android:drawableRight**, **android:drawableTop**, **android:drawableBottom**:
   * Adds a drawable (image) to the specified side of the text.
   * Example: android:drawableLeft="@drawable/icon".
3. **android:drawablePadding**:
   * Adds padding between the text and drawable.
   * Example: android:drawablePadding="5dp".

#### **Visibility and Behavior**

1. **android:visibility**:
   * Controls the visibility of the TextView.
   * Values: visible, invisible, gone.
   * Example: android:visibility="gone".
2. **android:clickable**:
   * Makes the TextView clickable.
   * Example: android:clickable="true".
3. **android:focusable**:
   * Makes the TextView focusable.
   * Example: android:focusable="true".
4. **android:enabled**:
   * Enables or disables the TextView.
   * Example: android:enabled="false".

#### **Customization**

1. **android:id**:
   * Assigns an ID to the TextView for referencing in code.
   * Example: android:id="@+id/myTextView".
2. **android:hint**:
   * Displays a hint when the text is empty.
   * Example: android:hint="Enter your name".
3. **android:singleLine** (deprecated):
   * Restricts the text to a single line (use maxLines="1" instead).
4. **android:shadowColor**, **android:shadowDx**, **android:shadowDy**, **android:shadowRadius**:
   * Adds a shadow effect to the text.
   * Example:

android:shadowColor="#000000"

android:shadowDx="2"

android:shadowDy="2"

android:shadowRadius="2"

### ****Example of a Fully Customized TextView****

<TextView

android:id="@+id/myTextView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, World!"

android:textSize="18sp"

android:textColor="#FF0000"

android:textStyle="bold"

android:gravity="center"

android:background="#FFFFFF"

android:padding="10dp"

android:layout\_margin="5dp"

android:drawableLeft="@drawable/icon"

android:drawablePadding="5dp"

android:maxLines="2"

android:ellipsize="end"

android:autoLink="web"

android:visibility="visible"

android:clickable="true" />

### Paste screen capture of Fully Customized Text View

## **EditText**: Allows users to input text.

## **Button**: A clickable button that performs an action.

## **ImageButton**: A button with an image instead of text.

## **ImageView**: Displays an image or drawable.

## **CheckBox**: A checkbox for binary choices.

## **RadioButton**: Used in groups for single-selection choices.

## **ToggleButton**: A button with an on/off state.

## **Switch**: A modern on/off toggle switch.