Parul® University Vadodara, Gujarat | NAAC GRADE A++

Mobile App Development

ANDROID USER INTERFACE ELEMENTS & LAYOUTS

Prof. Nilesh B. Ghavate

Assistant Professor

CSE IEP Department

Lesson Plan

Subject/Course	**Mobile App Development**
Lesson Title	**Android U.I. Elements & Layouts**

Lesson Objectives
Introduction of Material Design
Type’s of Layouts
UI Widget’s with Properties & Menus

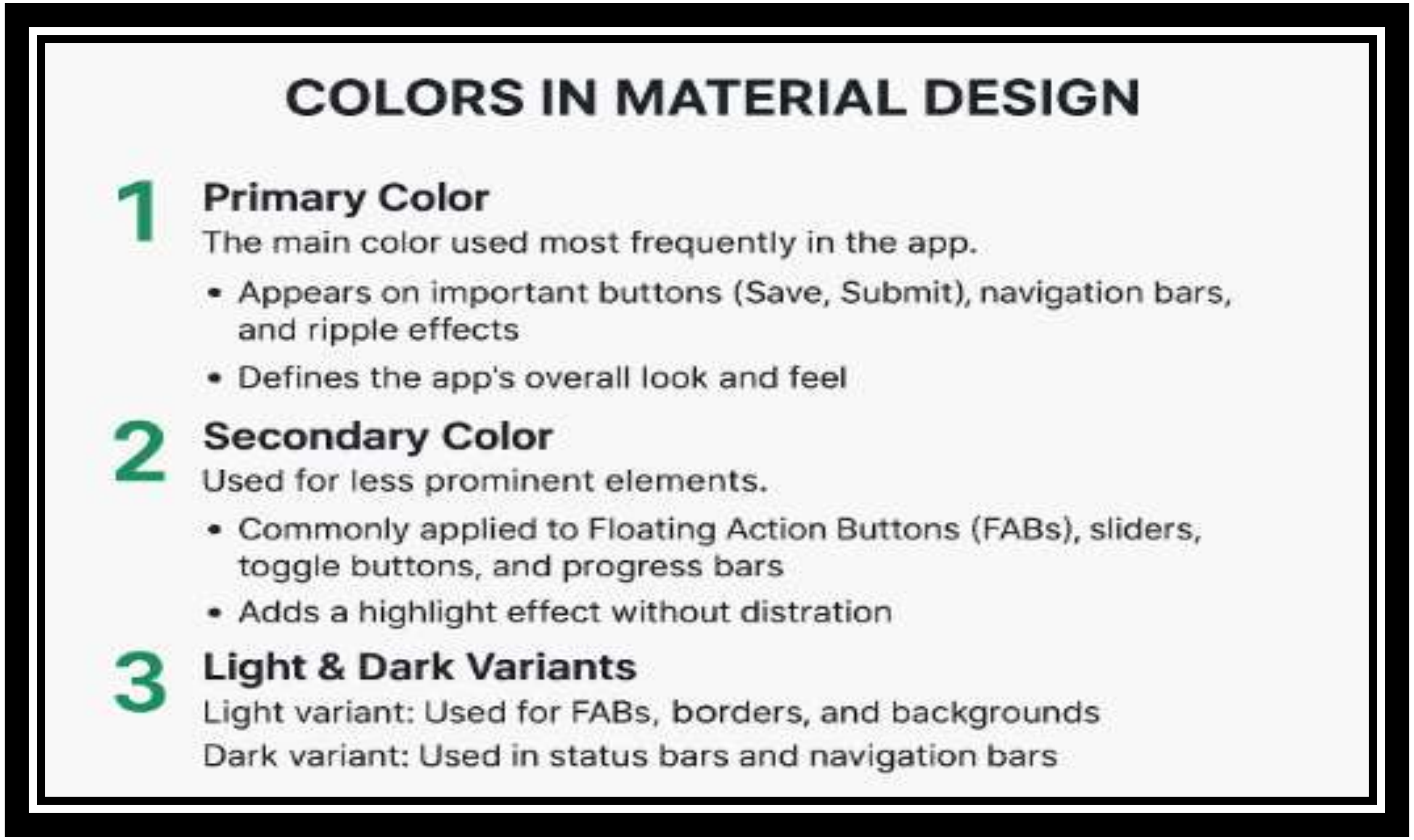
Introduction of Material Design

- **Material Design Components (MDC Android)** help developers create apps with a **modern, clean, and consistent look**.
- Developed by **Google**, it ensures apps are **visually appealing**, **smoothly animated**, and **easy to use**.
- Used widely to give Android apps a **polished and professional** appearance.
- Provides **ready-made UI elements** (buttons, cards, toolbars, etc.) for faster and uniform design.

Material Design Principles include:

- **Colors and Theming** (Choosing the right colors and styles for your app)
- **Typography** (Selecting appropriate fonts)
- **Material Design Components** (Using pre-built elements like buttons, cards, etc.)
- **Shaping Material Design Components** (Customizing the look of these components to match your app's design)

1. Colors and Theming

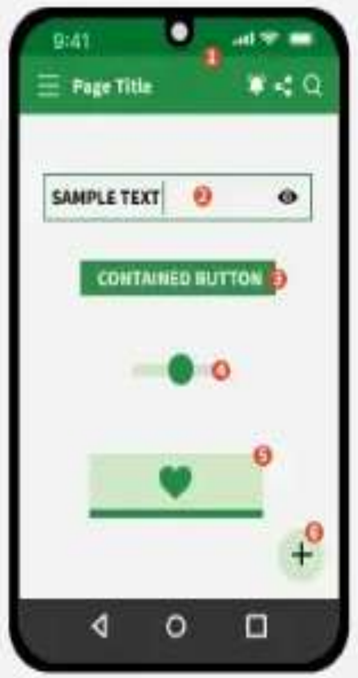


COLORS IN MATERIAL DESIGN

1 **Primary Color**
The main color used most frequently in the app.
- Appears on important buttons (Save, Submit), navigation bars, and ripple effects
- Defines the app's overall look and feel

2 **Secondary Color**
Used for less prominent elements.
- Commonly applied to Floating Action Buttons (FABs), sliders, toggle buttons, and progress bars
- Adds a highlight effect without distration

3 **Light & Dark Variants**
Light variant: Used for FABs, borders, and backgrounds
Dark variant: Used in status bars and navigation bars



1. The status bar is set for dark varient of primary color
2. The elements of edit text are set for primary color
3. The background color for Contained button Is set for primary color
4. The switch button is set for primary color and the background of the switch is set for light varient of primary color
5. In tab opened button the icon is set for primary color, and the background is set for light varient of primary color
6. The background color of the Floating action button is set for light varient of primary color.

2. Typography (Choosing the Right Font)

- **Typography** plays a key role in defining the **look, feel, and readability** of an Android app.
- In Android, the **Roboto font** is the default and most popular choice — it’s **modern, versatile, and highly legible**.
- Developers may also select other fonts to **enhance brand identity** or **create a unique visual style**.

Font Selection and Usage in Android

- **Roboto** meets the design needs of most apps with multiple **weight variants**:

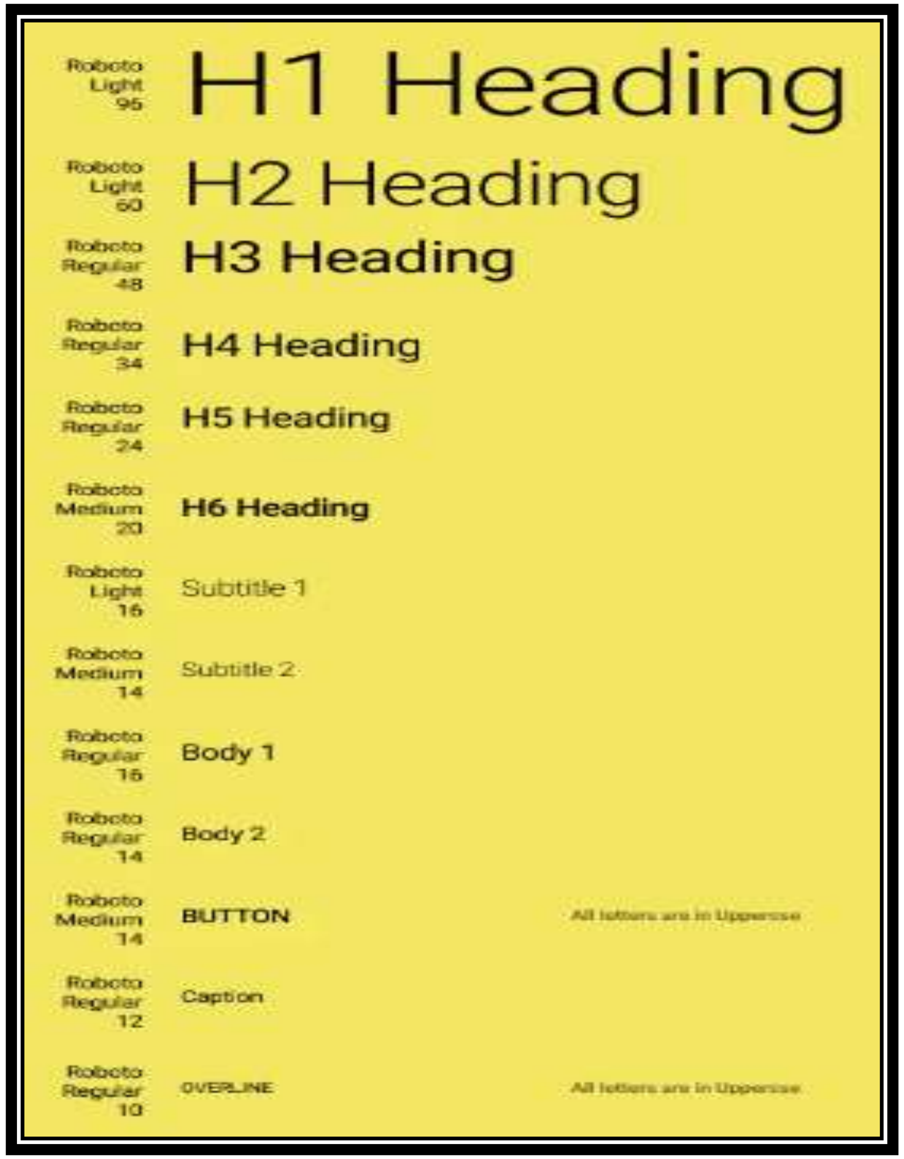
➤ Light ➤ Regular ➤ Medium ➤ Bold

- For additional customization, developers can choose fonts from **Google Fonts**, which offer:

- A **wide variety** of modern, web-safe fonts
- **Consistent styling** across devices
- **Complete variants** for flexibility in design

Guidelines for Font Selection

- Choose fonts that are **clear, professional, and readable** on all screen sizes.
- Maintain **consistency** — limit your app to **one or two font families**.
- Ensure fonts include **multiple weights** for hierarchy (titles, subtitles, body text, buttons).
- Test fonts for **contrast** and **legibility** in both **Light and Dark modes**.
- Match typography with your **app’s theme and personality** (formal, playful, minimal, etc.).



Roboto Light 96	H1 Heading	
Roboto Light 60	H2 Heading	
Roboto Regular 48	H3 Heading	
Roboto Regular 34	H4 Heading	
Roboto Regular 24	H5 Heading	
Roboto Medium 20	H6 Heading	
Roboto Light 16	Subtitle 1	
Roboto Medium 14	Subtitle 2	
Roboto Regular 16	Body 1	
Roboto Regular 14	Body 2	
Roboto Medium 14	BUTTON	All letters are in Uppercase
Roboto Regular 12	Caption	
Roboto Regular 10	OVERLINE	All letters are in Uppercase

UI & UX

❖ User Interface (UI)

- It is defined in an XML File.
- Visual design of the app.
- Layouts: LinearLayout, ConstraintLayout, etc.
- Widgets: Button, TextView, EditText.
- Colors, Fonts, Icons, Themes.
- Animation & Transitions.

❖ User Experience (UX)

- How the app feels to the user.
- Easy navigation & logical flow.
- Responsiveness & feedback.
- Accessibility & Consistency.
- Improves user satisfication.

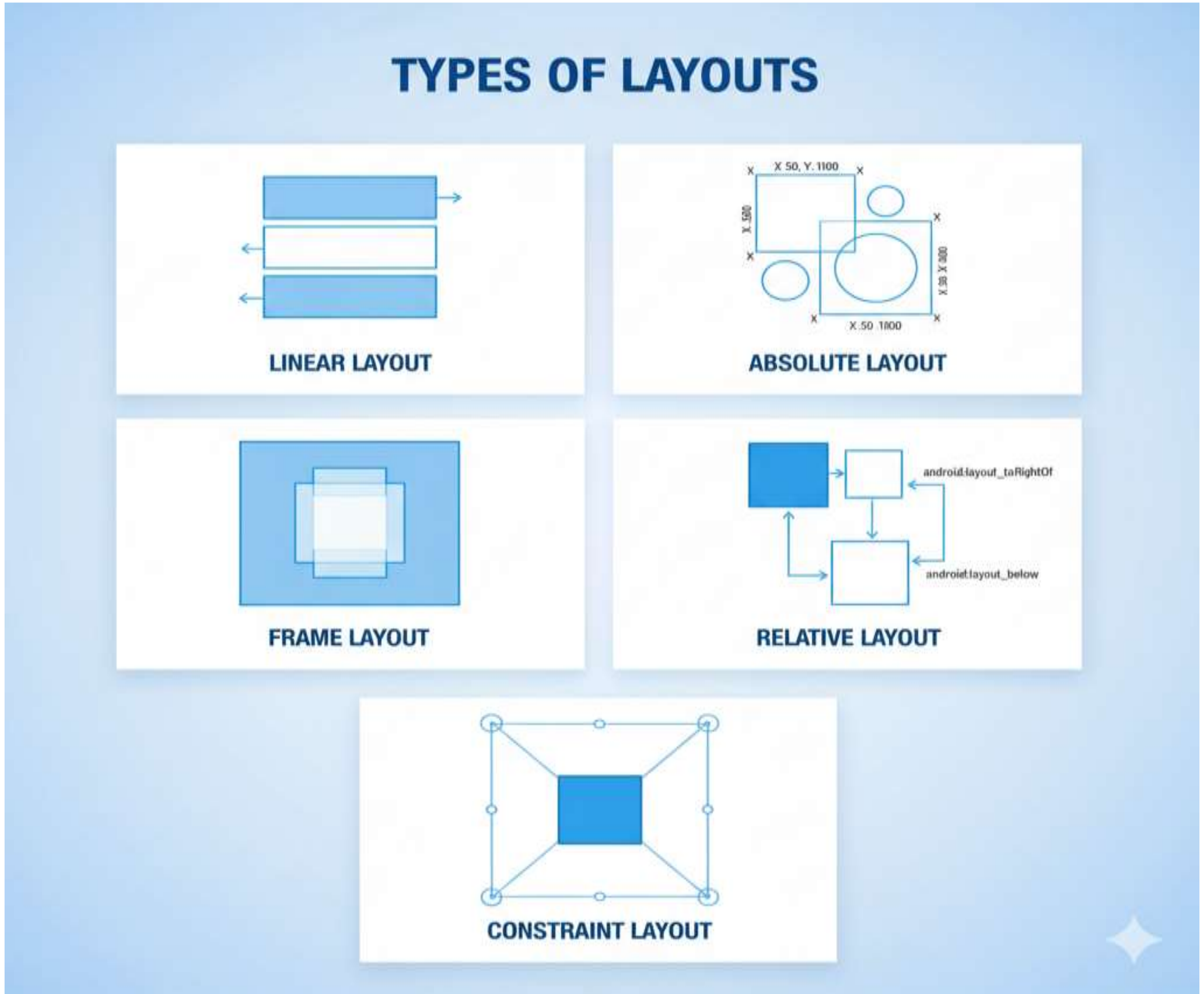
❖ Relationship Between UI & UX

UI = LOOKS , UX = FEEL

GOOD UI + GOOD UX =*SUCCESSFUL APP*

LAYOUTS

- Layouts define how UI elements are arranged on the screen.
- They control position, size & alignment of components.


TYPES OF LAYOUTS
LINEAR LAYOUT
X 50, Y 1100
X 500
X 30 X 300
X 50 1100
ABSOLUTE LAYOUT
FRAME LAYOUT
android:layout_toRightOf
android:layout_below
RELATIVE LAYOUT
CONSTRAINT LAYOUT

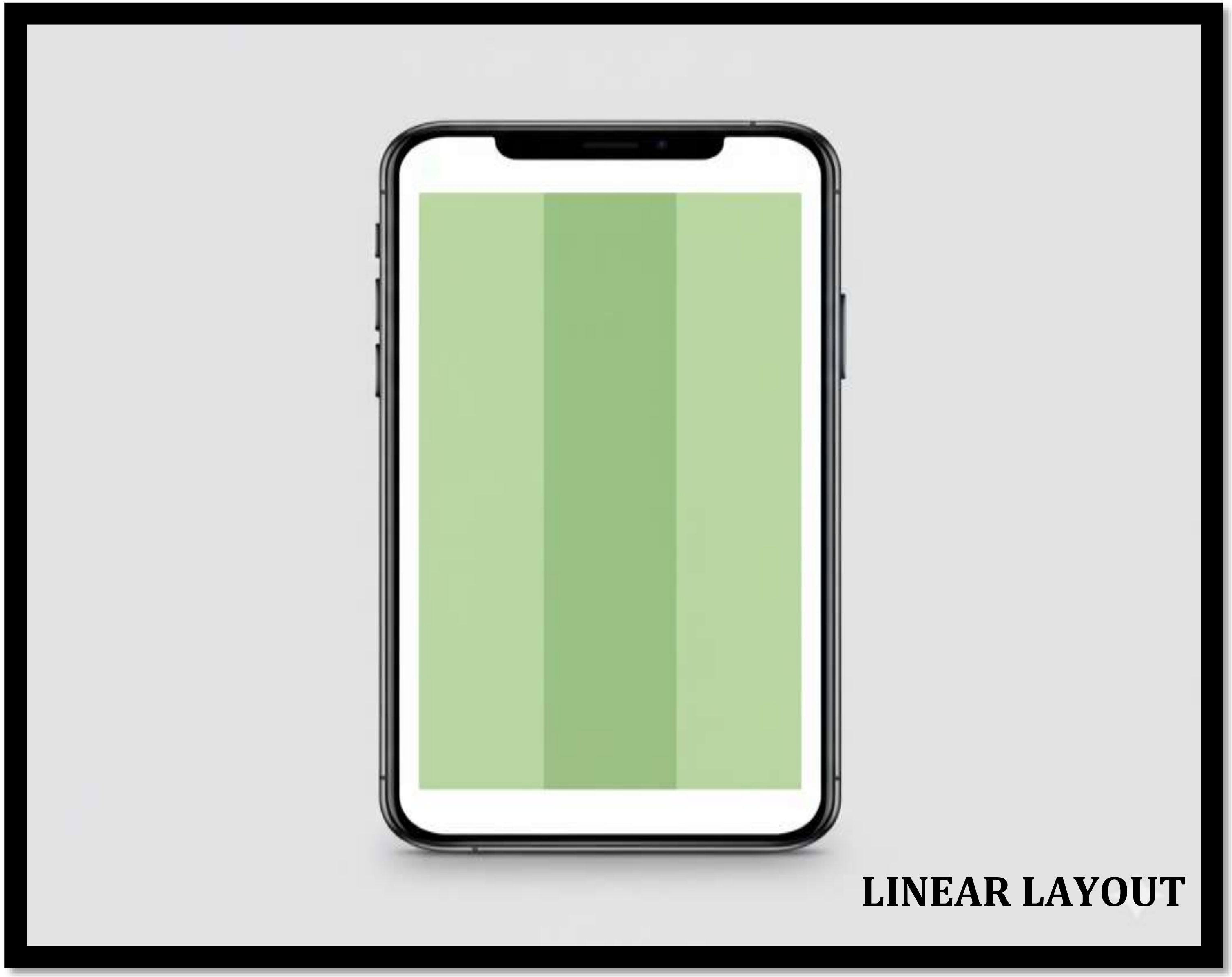
1.Linear Layout

- Arrange views in a single direction
- ❑ Vertical : single column of views
- ❑ Horizantal : single row of views
- It Supports weight attribute to control relative size of views.
- It is best for simple layouts with few elements.
- Avoid excessive nesting to prevent performance issues.

EXAMPLE:

```
<LinearLayout

xmlns:android="http://schemas.android.com/apk/r
es/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="horizontal"
   android:padding="16dp"
   android:background="@android:color/white">
<View
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:layout_weight="1"
      android:background="#A5D6A7"/>
<View
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:layout_weight="1"
      android:background="#81C784"/>

<View
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:layout_weight="1"
      android:background="#A5D6A7"/>

</LinearLayout>
```

OUTPUT:



2. ABSOLUTE LAYOUT

- Positions views using **exact X and Y coordinates** on the screen.
- Provides full control over where each view appears.
- Not recommended for different screen sizes (poor responsiveness.)
- Best for simple , fixed layouts with few elements.

- **EXAMPLE:**

```
<?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">
<View android:layout_width="100dp"
android:layout_height="80dp"
android:layout_x="20dp"
android:layout_y="30dp"
android:background="#FF5722" />
<View android:layout_width="120dp"
android:layout_height="100dp"
android:layout_x="80dp"
android:layout_y="100dp"
android:background="#2196F3" />
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_x="150dp"
android:layout_y="50dp"
android:text="Absolute Text"
android:textColor="#FFFFFF"
android:textSize="18sp"
android:background="#4CAF50"
android:padding="8dp"/>
</AbsoluteLayout>
```

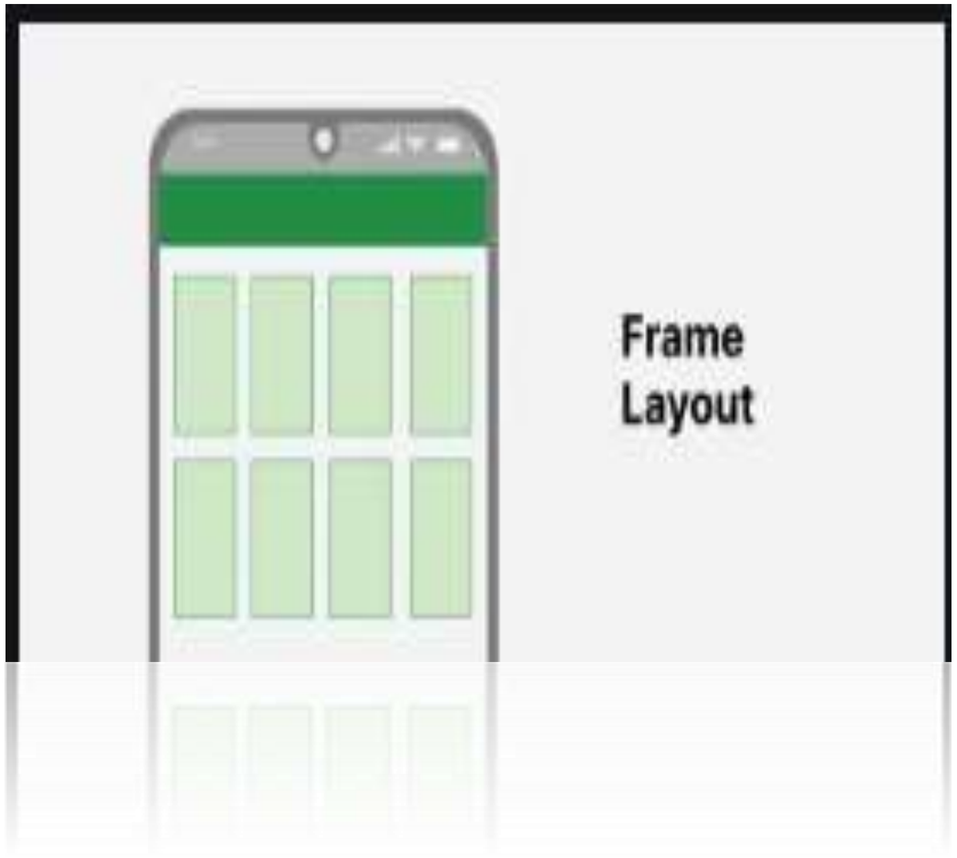
OUTPUT:

#FF57-22
100p
20
#2196F3
18sp
Absolute Text
16p
18sp

ABSOLUTE LAYOUT

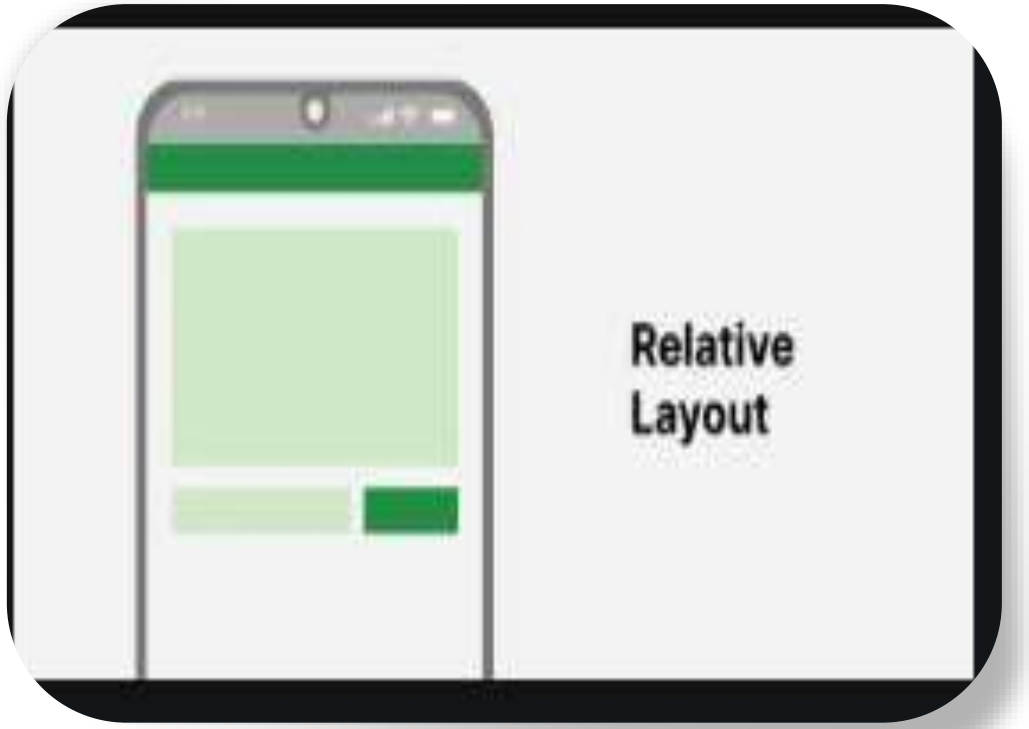
3. FRAME LAYOUT

- A simple layout designed to hold **one child view**.
- Multiple views can be **stacked on top of each other**.
- Commonly used for **overlays, fragments, and image views**.
- Ideal for **simple UI components** where layering is required.



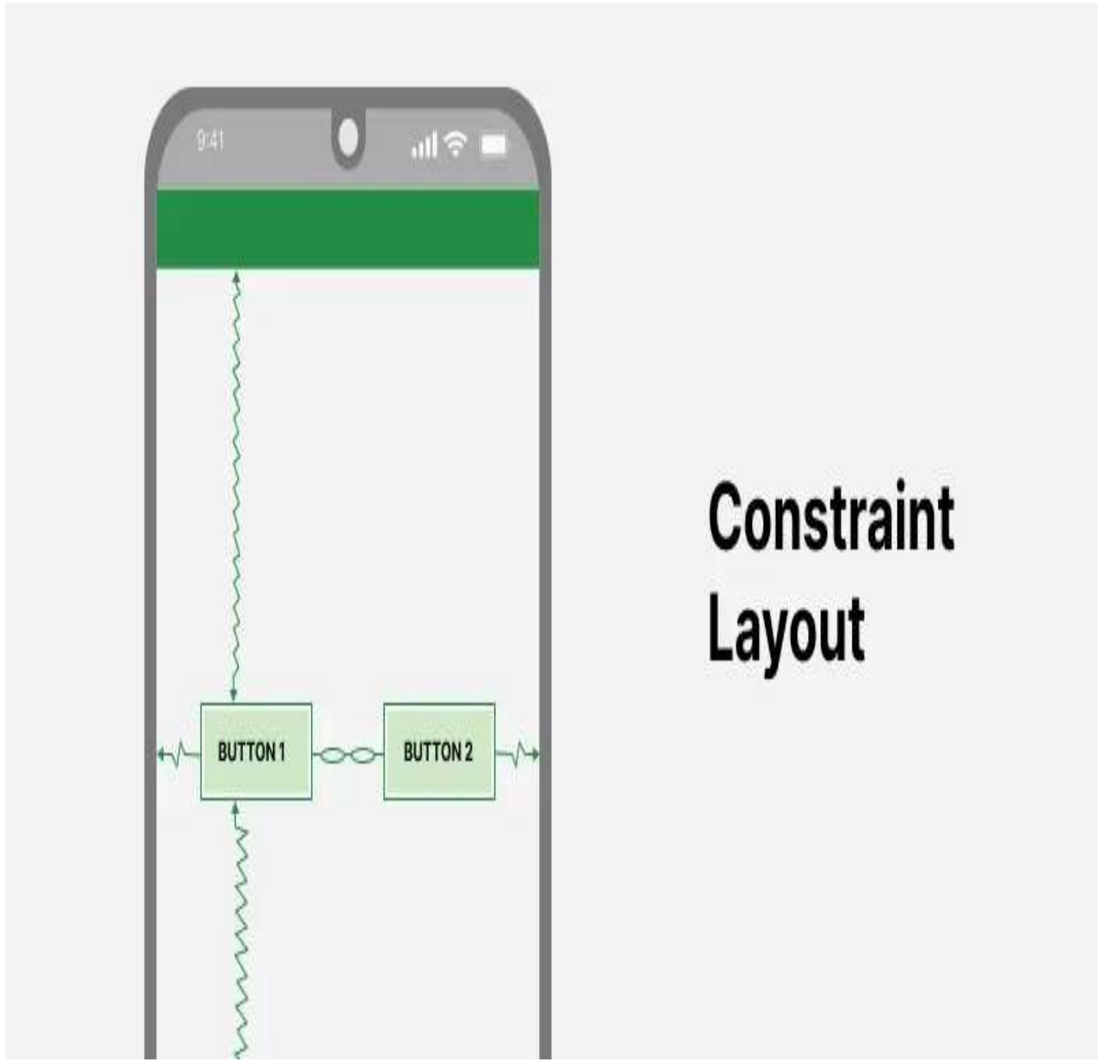
4. RELATIVE LAYOUT

- It Arranges **child views relative to each other** or the parent.
- It Views can be positioned **above, below, to the left, or right** of other views.
- It Useful for **flexible and dynamic UI designs**.
- It Allows **precise control over view placement** without nesting multiple layouts.



5.CONSTRAINT LAYOUT

- It is Flexible layout for Android UI design.
- Positions views using **constraints**, not nested layouts.
- It Reduces **view hierarchy**, improving performance.
- It Supports **complex and responsive designs**.
- It is most **recommended layout** for modern applications.



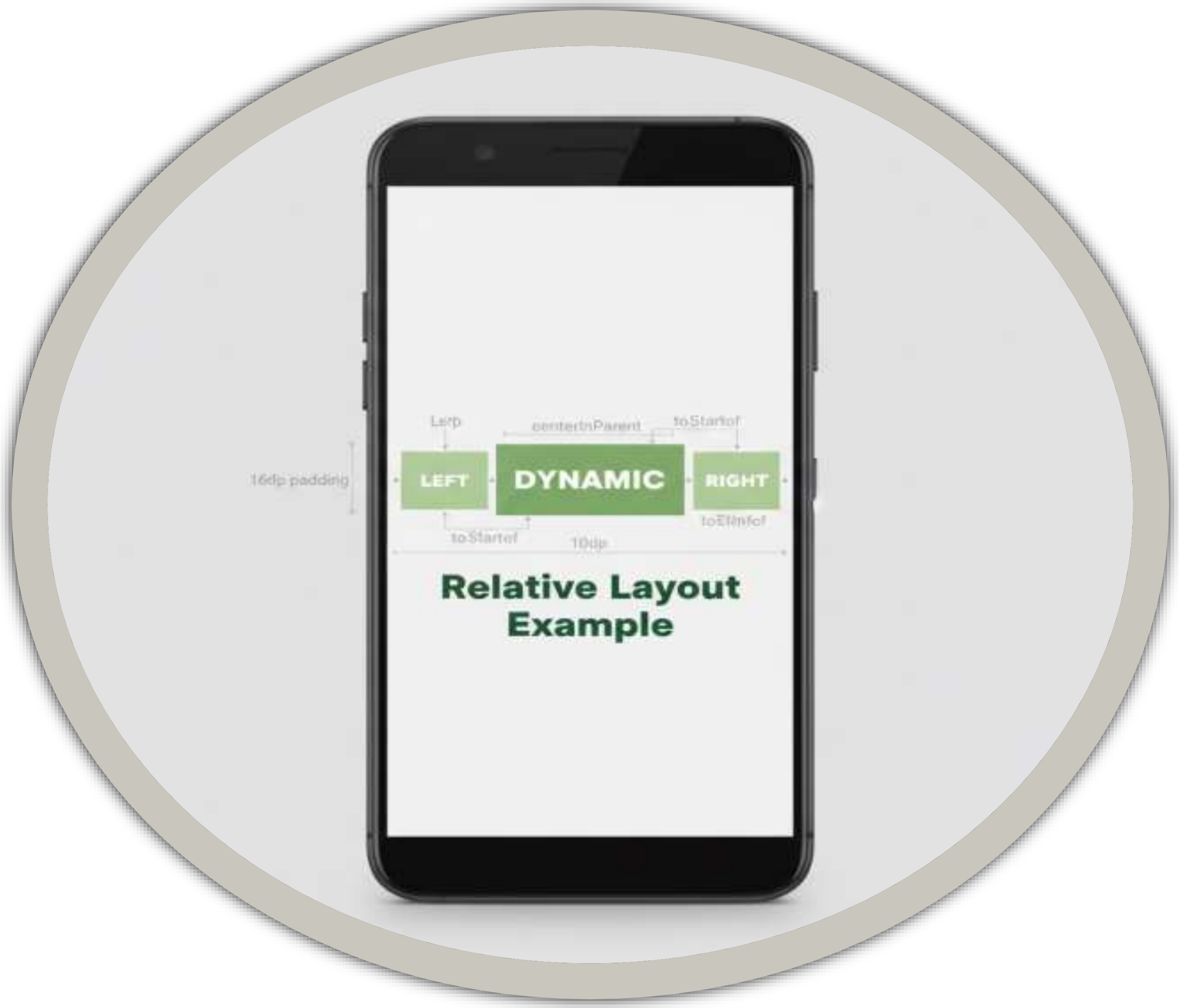
DYNAMIC IMPLEMENTATION OF LAYOUT

- UI created or modified **programmatically at runtime**.
- Flexible; can change based on user input, data, or events.
- Requires more code and logic to implement.
- Can be slightly heavier if many views are added dynamically.
- Dynamic content like lists, forms, or user-driven UI.

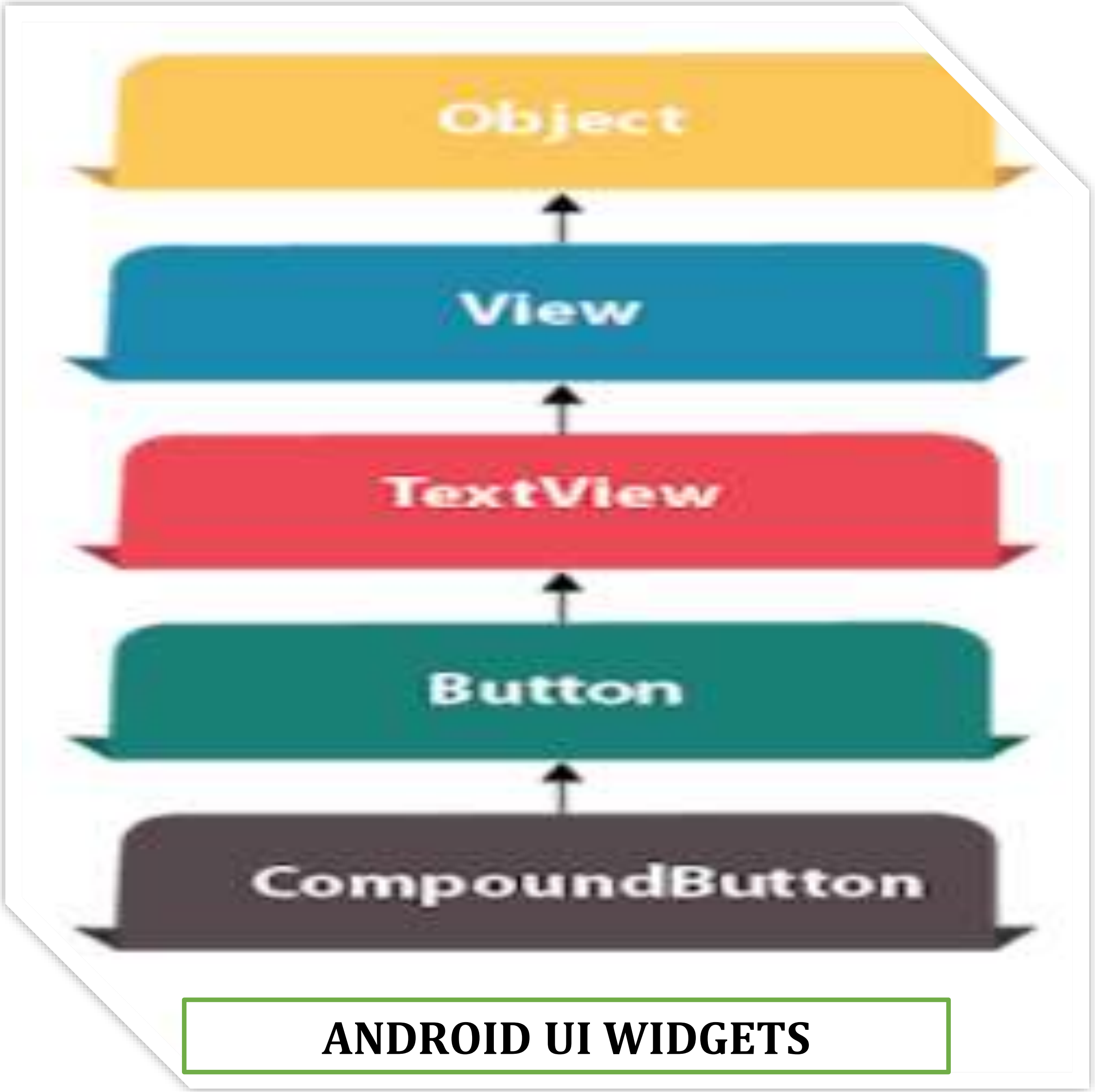
EXAMPLE:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk
/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp"
android:background="@android:color/white"
tools:context=".MainActivity">

<TextView android:id="@+id/tv_dynamic_center"
android:layout_width="120dp"
android:layout_height="80dp"
android:layout_centerInParent="true"
android:background="#66BB6A"
android:text="DYNAMIC"
android:textColor="#FFFFFF"
android:textSize="18sp" android:textStyle="bold"
android:gravity="center" />

<TextView android:id="@+id/tv_left"
android:layout_width="80dp" android:layout_height="80dp"
android:layout_toStartOf="@id/tv_dynamic_center"
android:layout_centerVertical="true"
android:layout_marginEnd="10dp"
android:background="#9CCC65" android:text="LEFT"
android:textColor="#FFFFFF" android:textSize="16sp"
android:textStyle="bold" android:gravity="center" />

<TextView android:id="@+id/tv_right"
android:layout_width="80dp" android:layout_height="80dp"
android:layout_toEndOf="@id/tv_dynamic_center"
android:layout_centerVertical="true"
android:layout_marginStart="10dp"
android:background="#9CCC65" android:text="RIGHT"
android:textColor="#FFFFFF" android:textSize="16sp"
android:textStyle="bold" android:gravity="center" />

<TextView android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_below="@id/tv_dynamic_center"
android:layout_centerHorizontal="true"
android:layout_marginTop="60dp" android:text="Relative Layout
Example" android:textColor="#388E3C" android:textSize="22sp"
android:textStyle="bold" />
</RelativeLayout>
```

OUTPUT:


LEFT
DYNAMIC
RIGHT
16dp padding
centerInParent
toStartof
toStartof
toEndof
10dp
Relative Layout Example

UI WIDGETES WITH PROPERTIES



ANDROID UI WIDGETS

UI Widgets & Properties in Android

UI Widget	Purpose	Key Properties
TextView	Displays text	text, textSize, textColor, gravity, fontFamily
EditText	User input	hint, inputType, textColor, maxLength
Button	Triggers actions	text, background, onClick, enabled
ImageView	Displays images	src, scaleType, contentDescription
CheckBox	Select/deselect options	checked, text, onCheckedChangeListener
RadioButton	Single choice selection	checked, text, onClick
Switch	Toggle ON/OFF	checked, textOn, textOff, onCheckedChangeListener
Spinner	Dropdown selection	entries, onItemSelectedListener, prompt
ProgressBar	Shows loading/progress	max, progress, indeterminate, visibility
SeekBar	Sliding selection	max, progress, onSeekBarChangeListener

Android Events & Methods

- Events are actions triggered by **user interactions** or **system changes** in an app.
- Methods are **functions that handle these events**, like clicks, touches, or text changes.
- Common events include **Click, Touch, Long Click, Key, Focus, Text Change, Item Selection, and Lifecycle events**.
- Handling events properly ensures a **responsive and interactive Android app UI**.

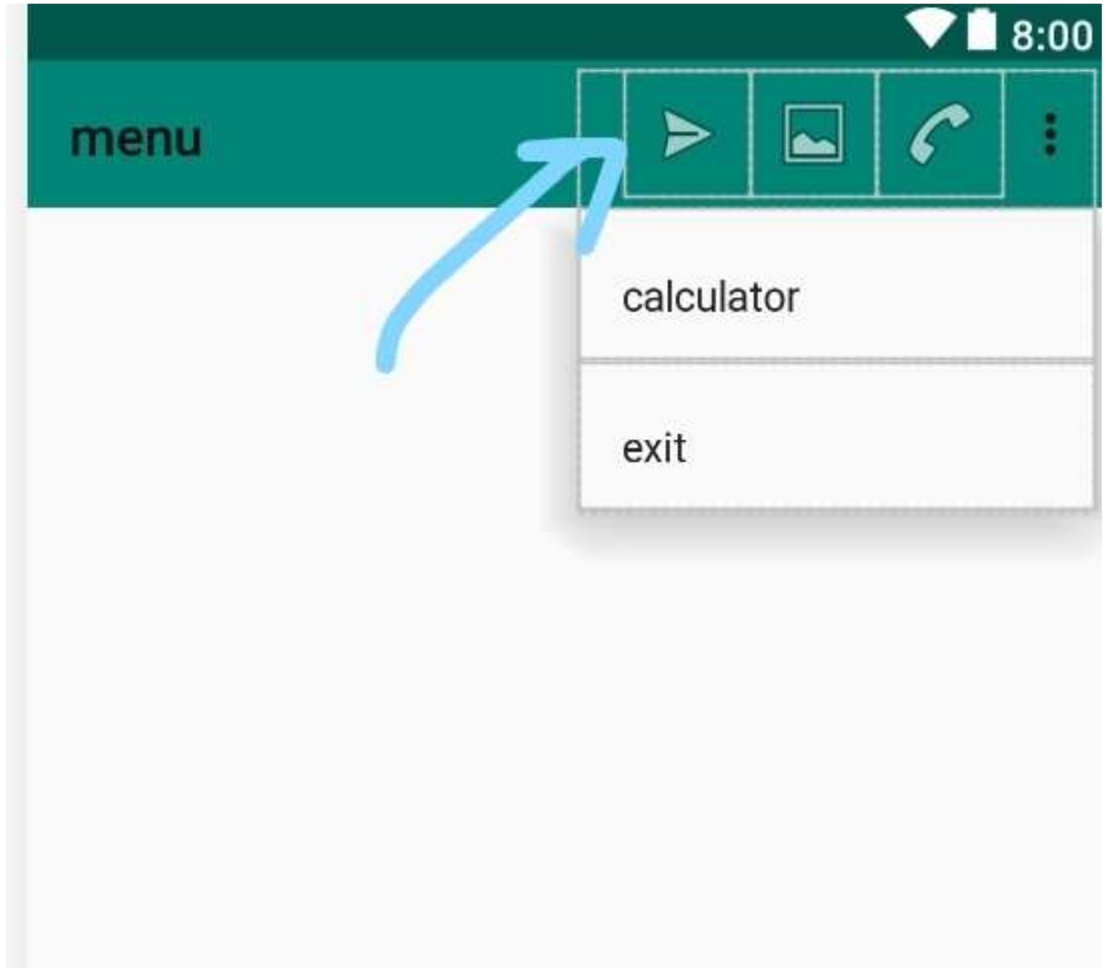
Event Type	Description	Key Methods
Click Events	Triggered when a user taps a view (e.g., Button)	setOnClickListener()
Touch Events	Triggered on touch interactions	onTouch(), MotionEvent (ACTION_DOWN, ACTION_UP)
Long Click Events	Triggered on long press	setOnLongClickListener()
Key Events	Triggered when hardware keys are pressed	onKeyDown(), onKeyUp()
Focus Events	Triggered when a view gains or loses focus	onFocusChange(), setOnFocusChangeListener()
Text Change Events	Triggered when text in EditText changes	addTextChangedListener()
Item Selection Events	Triggered for ListView, Spinner, or RecyclerView item selection	setOnItemSelectedListener(), setOnItemClickListener()
Lifecycle Events	Triggered during Activity/Fragment lifecycle changes	onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy()

Dialog Boxes in Android

- Dialog boxes are **small windows** that prompt the user to make decisions or enter information without leaving the current activity.
- Used for **alerts, confirmations, inputs, or choices**.
- **Common Types:**
- **AlertDialog:** Shows alerts with buttons (OK, Cancel).
- **ProgressDialog:** Displays progress while a task is running. *(Deprecated in newer APIs)*
- **Custom Dialog:** Fully customizable layout for specific needs.
- **Key Methods** : show(), dismisss(), setMessage(), setTitle()

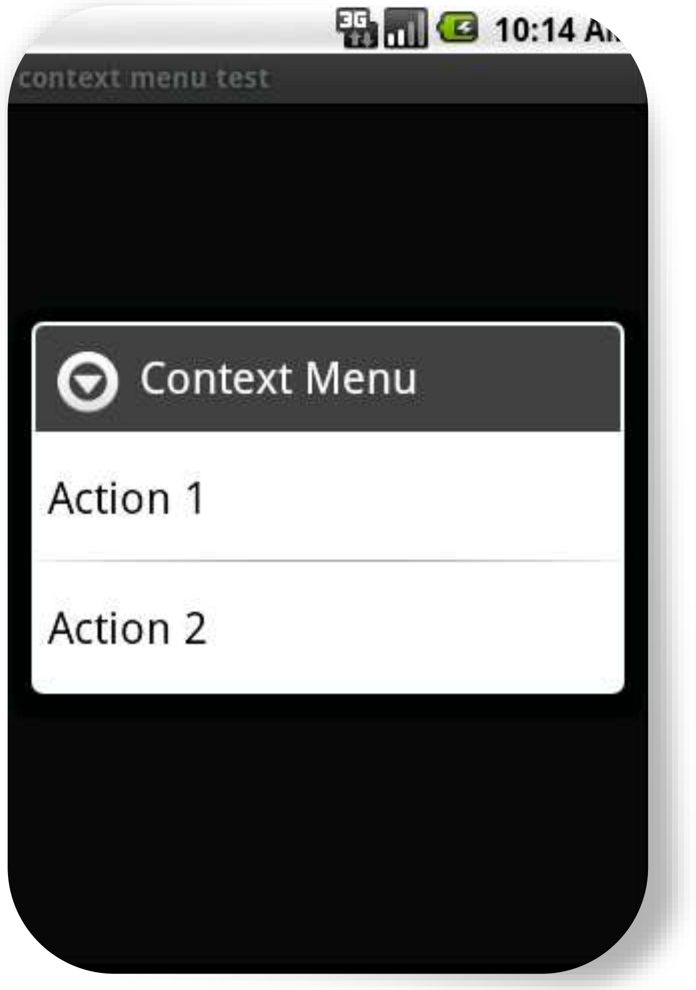
Menus: Option and Context

- Android Option Menus are the primary menus of android. They can be used for settings, search, delete item etc.
- To inflating the menu by calling the inflate() method of MenuInflater class.
- To perform event handling on menu items, you need to override onOptionsItemSelected() method of Activity class.



Android Context Menu

- Android context menu appears when user press long click on the element.
- It is also known as floating menu.
- It affects the selected content while doing action on it.
- It doesn't support item shortcuts and icons.



THANK YOU!