UNIT-III

❖ User Interface (UI) Screen Elements

In Android development, the **User Interface (UI) Screen Elements** are the building blocks that help design interactive layouts for apps. These elements are primarily composed of **View** and **ViewGroup** components. Below are the key UI elements:

I. Basic UI Elements

- ✓ **TextView** → Displays text to the user.
- ✓ **EditText** → Input field for user text input (e.g., username, password).
- ✓ **Button** → Clickable element that performs actions.
- ✓ **ImageView** → Displays images.
- ✓ **ProgressBar** → Shows loading progress.
- ✓ **CheckBox** → Allows selection of multiple options.
- ✓ **RadioButton & RadioGroup** → Allows selection of a single option from a group.
- ✓ **Switch** → A toggle button for ON/OFF states.

II. Layouts (ViewGroups)

These define how UI elements are arranged:

- ✓ **LinearLayout** → Arranges elements in a single row or column.
- ✓ **RelativeLayout** → Positions elements relative to each other.
- ✓ **ConstraintLayout** → A flexible layout that uses constraints for positioning.
- ✓ **FrameLayout** → Holds a single child view, useful for overlays.
- ✓ **TableLayout** → Organizes elements in rows and columns.
- ✓ **ScrollView** → Enables vertical scrolling.

III. UI Containers

- ✓ **Dialog** → Shows pop-up messages or input prompts.
- ✓ **PopupMenu** → Displays a small menu when a button is clicked.
- ✓ **DrawerLayout (Navigation Drawer)** → A side menu for navigation.

- ✓ **BottomSheet** → A modal or persistent UI component that slides up from the bottom.

Basic UI Elements

View Components (Basic UI Elements) in Android

View components are the essential building blocks of an Android app's UI. They are responsible for displaying content and handling user interactions. Below are some of the most commonly used **View** components in Android development:

1. TextView 📄

- Used to display **static text** to the user.
- Cannot be edited by the user (unlike EditText).
- Supports text formatting, styling, and linking.

Example:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, World!"
    android:textSize="18sp"
    android:textColor="@android:color/black"/>
```

2. EditText ✍

- Allows the user to enter text (input field).
- Can be used for passwords, numbers, emails, etc.

Example:

```
<EditText
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter your name"
    android:inputType="text"/>
```

3. Button 🔘

- Used to perform an action when clicked.
- Supports different styles (e.g., default, elevated, outlined).

Example:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click Me"
    android:onClick="handleClick"/>
```

(In the Java/Kotlin file, define handleClick(View view) to handle button clicks.)

4. ImageView 🖼

- Displays images or icons.
- Can load images from resources or URLs.

Example:

```
<ImageView
    android:layout_width="100dp"
    android:layout_height="100dp"
```

```
   android:src="@drawable/sample_image"/>
```

5. CheckBox ✅

- Allows users to select multiple options independently.

Example:

```
<CheckBox
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="I agree to the terms"/>
```

6. RadioButton & RadioGroup 🎯

- RadioButton allows users to choose only **one option** from a group.
- RadioGroup ensures that only one option is selected at a time.

Example:

```
<RadioGroup
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">
   <RadioButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Option 1"/>

   <RadioButton
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Option 2"/>
</RadioGroup>
```

7. ProgressBar ⏳

- Used to indicate loading or processing progress.

Example (Indeterminate Circular Progress):

```
<ProgressBar
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"/>
```

Example (Determinate Horizontal Progress):

```
<ProgressBar
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  style="@android:style/Widget.ProgressBar.Horizontal"
  android:progress="50"
  android:max="100"/>
```

8. SeekBar 🎚

- Allows users to **select a value** by sliding.

Example:

```
<SeekBar
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"/>
```

Layouts (ViewGroups)

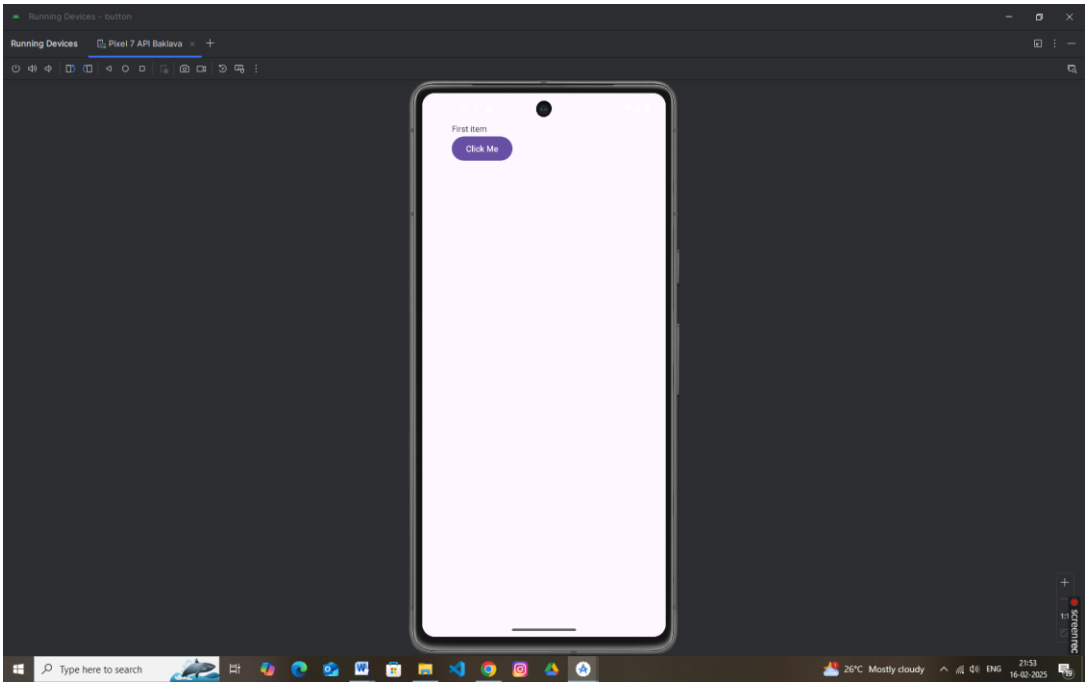
These **layout containers** help you structure the user interface of your Android app. Here's an overview of how each one works and how to use them in **Java**:

1. LinearLayout

LinearLayout arranges child elements either in a row (horizontal) or in a column (vertical).

- **XML (LinearLayout Example)**:

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="First item" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />
</LinearLayout>
```



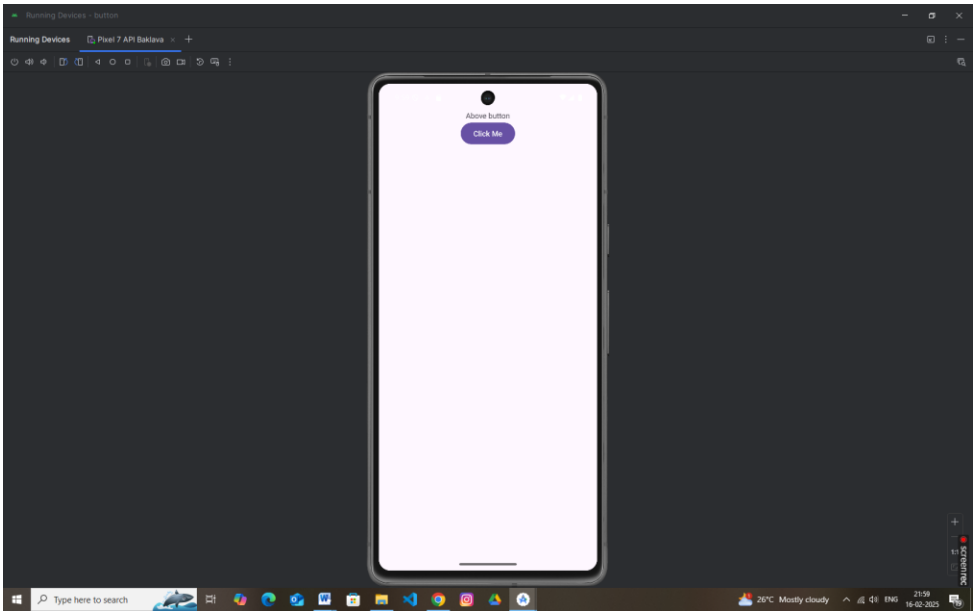
2. RelativeLayout

RelativeLayout positions elements relative to each other. You can position elements by specifying attributes like android:layout_toEndOf, android:layout_below, etc.

- **XML (RelativeLayout Example)**:

```
<RelativeLayout
   android:id="@+id/relativeLayout"
   android:layout_width="match_parent"
   android:layout_height="wrap_content">

   <TextView
      android:id="@+id/textView"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Above button"
      android:layout_centerHorizontal="true" />

   <Button
      android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Click Me"
      android:layout_below="@id/textView"
```

```
        android:layout_centerHorizontal="true" />
</RelativeLayout>
```



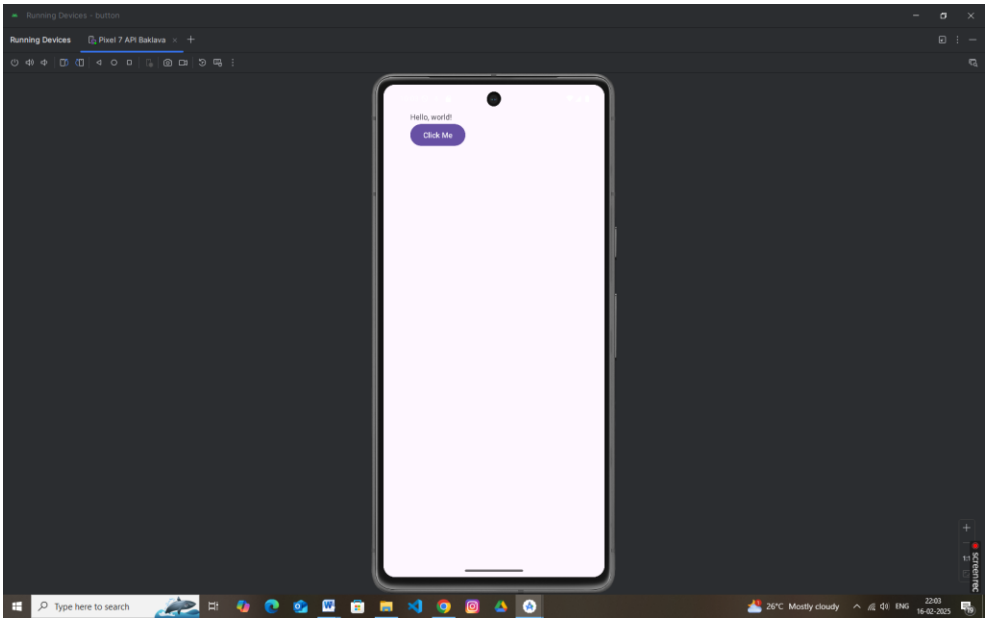
3. ConstraintLayout

ConstraintLayout is the most flexible layout container, allowing you to define complex layouts with flexible positioning and relationships between elements. It's recommended for modern UI design.

- **XML (ConstraintLayout Example)**:

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:id="@+id/constraintLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, world!"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
```

```
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me"
        app:layout_constraintTop_toBottomOf="@id/textView"
        app:layout_constraintStart_toStartOf="parent" />

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



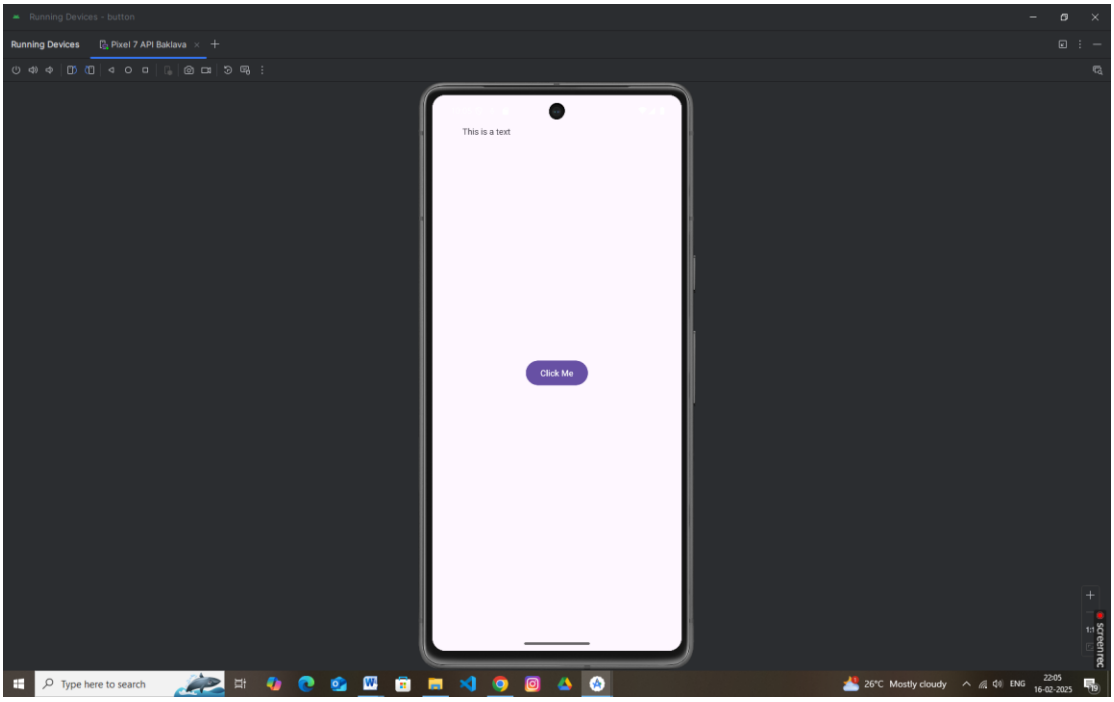
4. FrameLayout

FrameLayout is used to stack child elements on top of each other. This layout is commonly used for displaying one element at a time (like showing a fragment or image).

- **XML (FrameLayout Example)**:

```
<FrameLayout
    android:id="@+id/frameLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

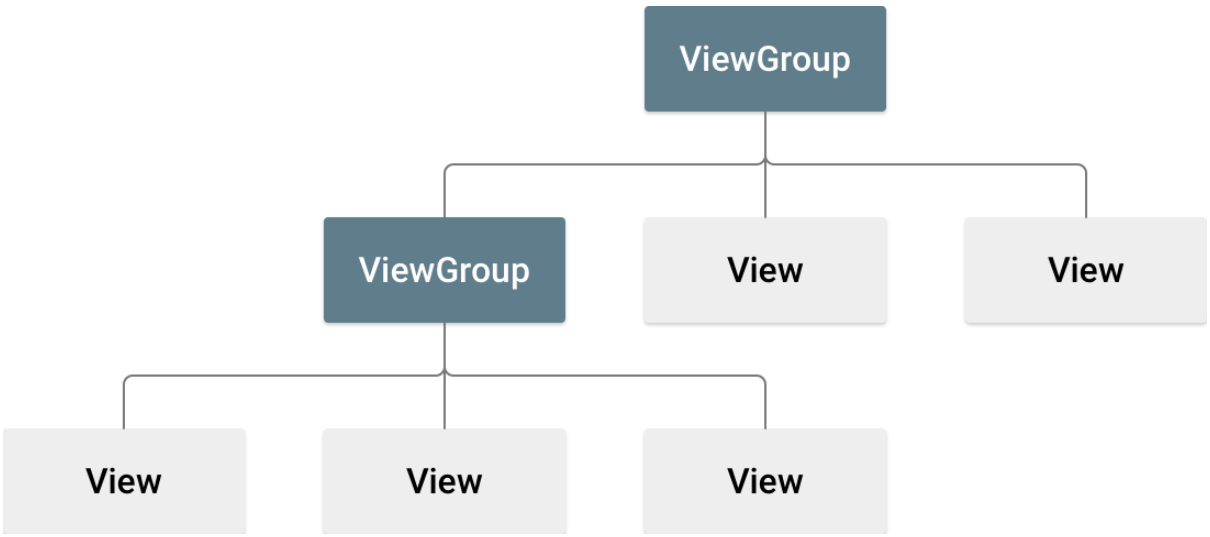
```
        android:text="This is a text" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:layout_gravity="center" />
</FrameLayout>
```



❖ Designing User Interfaces with Layouts

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. A ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects



Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

Sr.No	Layout & Description
1	**Linear Layout** LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	**Relative Layout** RelativeLayout is a view group that displays child views in relative positions.
3	**Table Layout** TableLayout is a view that groups views into rows and columns.
4	**Absolute Layout** AbsoluteLayout enables you to specify the exact location of its children.
5	**Frame Layout** The FrameLayout is a placeholder on screen that you can use to display a single view.
6	**List View** ListView is a view group that displays a list of scrollable items.

7	Grid View GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and their are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the view.
2	android:layout_width This is the width of the layout.
3	android:layout_height This is the height of the layout
4	android:layout_marginTop This is the extra space on the top side of the layout.
5	android:layout_marginBottom This is the extra space on the bottom side of the layout.
6	android:layout_marginLeft This is the extra space on the left side of the layout.
7	android:layout_marginRight This is the extra space on the right side of the layout.
8	android:layout_gravity This specifies how child Views are positioned.
9	android:layout_weight This specifies how much of the extra space in the layout should be allocated to the View.
10	android:layout_x This specifies the x-coordinate of the layout.
11	android:layout_y This specifies the y-coordinate of the layout.
12	android:layout_width This is the width of the layout.
13	android:paddingLeft This is the left padding filled for the layout.

14	**android:paddingRight** This is the right padding filled for the layout.
15	**android:paddingTop** This is the top padding filled for the layout.
16	**android:paddingBottom** This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp (Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).

You can specify width and height with exact measurements but more often, you will use one of these constants to set the width or height −

- **android:layout_width=wrap_content** tells your view to size itself to the dimensions required by its content.
- **android:layout_width=fill_parent** tells your view to become as big as its parent view.

View Identification

A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is −

android:id="@+id/my_button"

Following is a brief description of @ and + signs −

- The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.
- The plus-symbol (+) means that this is a new resource name that must be created and added to our resources.

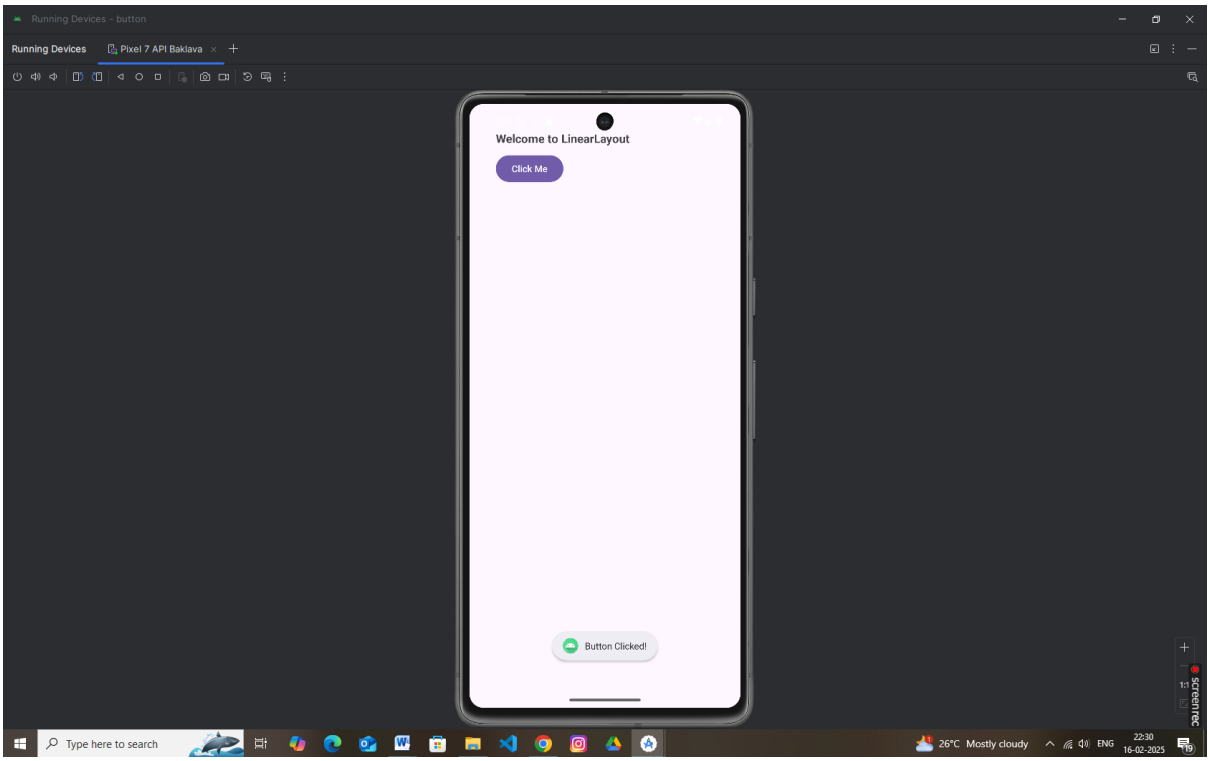
MainActivity.java

```
package com.example.button;

import android.os.Bundle;
```

```
import android.widget.Button;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button button = findViewById(R.id.button);
        button.setOnClickListener(v ->
                Toast.makeText(this, "Button Clicked!",
Toast.LENGTH_SHORT).show()
        );
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="40dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to LinearLayout"
        android:textSize="18sp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me"
        android:layout_marginTop="10dp"/>

</LinearLayout>
```



❖ Animation in Android with Example

Animation is the process of adding a motion effect to any view, image, or text. With the help of an animation, you can add motion or can change the shape of a specific view. Animation in Android is generally used to give your UI a rich look and feel.

Here's example—this time using a **slide-in animation** that makes a view slide in from the left.

Step 1: Create a Slide-In Animation Resource

Create a new XML file called slide_in_left.xml in your project's res/anim folder:

```
<?xml version="1.0" encoding="utf-8"?>
<translate xmlns:android="http://schemas.android.com/apk/res/android"
    android:fromXDelta="-100%"
    android:toXDelta="0%"
    android:duration="1000" />
```

Explanation:

- **fromXDelta="-100%"**: The view starts off-screen to the left.
- **toXDelta="0%"**: The view ends at its natural position.
- **duration="100"**: The animation lasts for 500 milliseconds.

Step 2: Create a Layout with a View

For this example, let's create a layout with a Button that will slide in from the left.

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16sp"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/slideButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Slide In" />

</LinearLayout>
```

Step 3: Load and Start the Animation in Your Activity

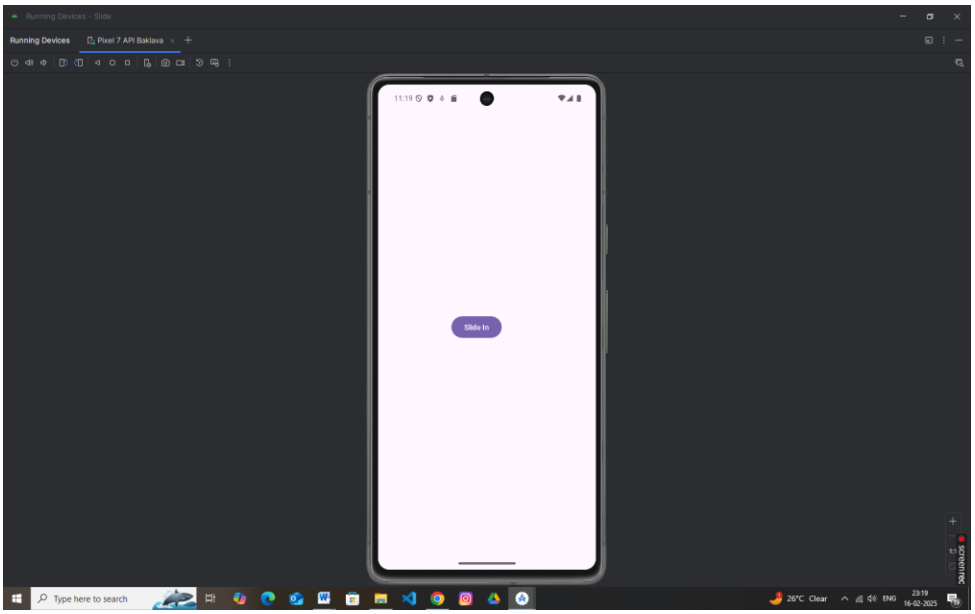
In your MainActivity.java, load the slide-in animation using AnimationUtils and apply it to the button.

MainActivity.java

```
package com.example.slide;

import android.os.Bundle;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.Button;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

public class MainActivity extends AppCompatActivity {

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

        // Find the button by its ID
        Button slideButton = findViewById(R.id.slideButton);

        // Load the slide-in animation from the resource
        Animation slideAnimation =
AnimationUtils.loadAnimation(this,R.anim.slide_in_left);

        // Start the animation when the activity starts
        slideButton.startAnimation(slideAnimation);
    }
}
```

Explanation:

- **AnimationUtils.loadAnimation(this, R.anim.slide_in_left)**: Loads the slide-in animation from the XML resource.
- **slideButton.startAnimation(slideInAnimation)**: Applies the animation to the Button.



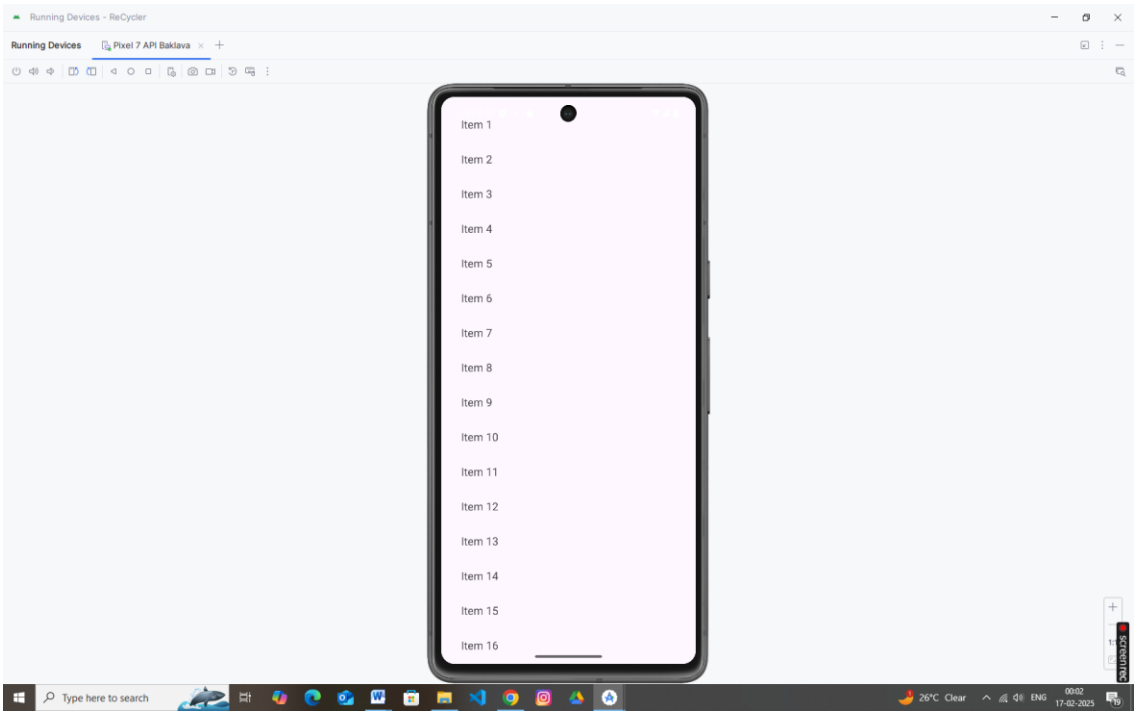
❖ RecyclerView

RecyclerView makes it easy to efficiently display large sets of data. You supply the data and define how each item looks, and the RecyclerView library dynamically creates the elements when they're needed.

As the name implies, RecyclerView *recycles* those individual elements. When an item scrolls off the screen, RecyclerView doesn't destroy its view. Instead, RecyclerView reuses the view for new items that have scrolled onscreen. RecyclerView improves performance and your app's responsiveness, and it reduces power consumption.

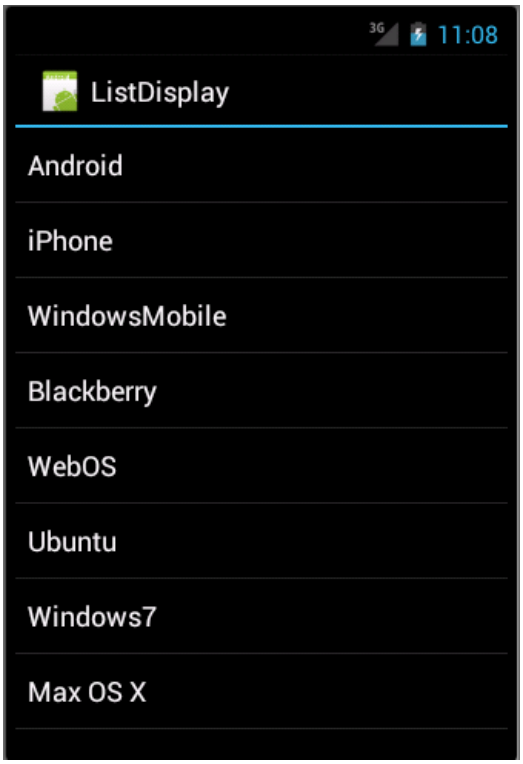
- **RecyclerView** is the ViewGroup that contains the views corresponding to your data. It's a view itself, so you add RecyclerView to your layout the way you would add any other UI element.
- Each individual element in the list is defined by a *view holder* object. When the view holder is created, it doesn't have any data associated with it. After the view holder is created, the RecyclerView *binds* it to its data. You define the view holder by extending RecyclerView.ViewHolder.
- The RecyclerView requests views, and binds the views to their data, by calling methods in the *adapter.* You define the adapter by extending RecyclerView.Adapter.

- The *layout manager* arranges the individual elements in your list. You can use one of the layout managers provided by the RecyclerView library, or you can define your own. Layout managers are all based on the library's LayoutManager abstract class.



❖ Android **ListView**

*Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.*



List View

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (i.e. ListView or GridView). The common adapters are **ArrayAdapter**,**Base Adapter**, **CursorAdapter**, **SimpleCursorAdapter**,**SpinnerAdapter** and **WrapperListAdapter**. We will see separate examples for both the adapters.

ArrayAdapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**. Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array −

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);
```

Here are arguments for this constructor −

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

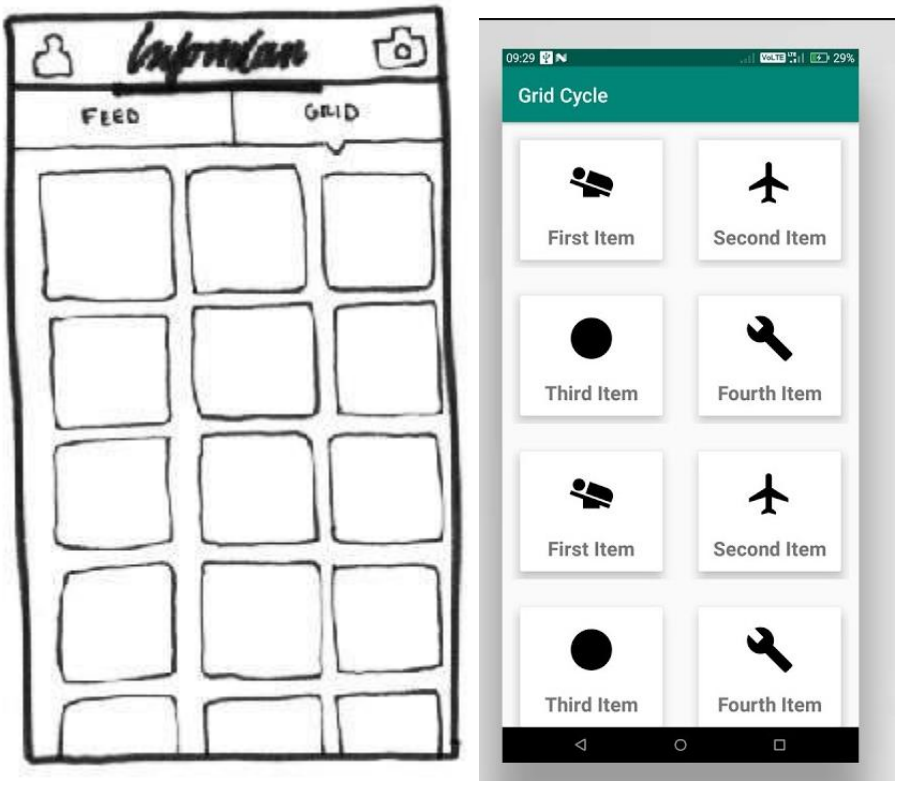
Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows −

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

You will define your list view under res/layout directory in an XML file. For our example we are going to using activity_main.xml file.

❖ Android GridView

*Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter***



An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

GridView Attributes

Following are the important attributes specific to GridView –

Sr.No	Attribute & Description
1	**android:id** This is the ID which uniquely identifies the layout.
2	**android:columnWidth** This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
3	**android:gravity** Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:horizontalSpacing** Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
5	**android:numColumns** Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.

- ❖ Input Controls

Below is a comprehensive guide to various input controls and UI components in Android. Each section includes an explanation and a simple code example. You can mix and match these concepts to build robust and interactive user interfaces.

1. Buttons

In Android, Button represents a push button. A Push buttons can be clicked, or pressed by the user to perform an action. There are different types of buttons used in android such as CompoundButton, ToggleButton, RadioButton.

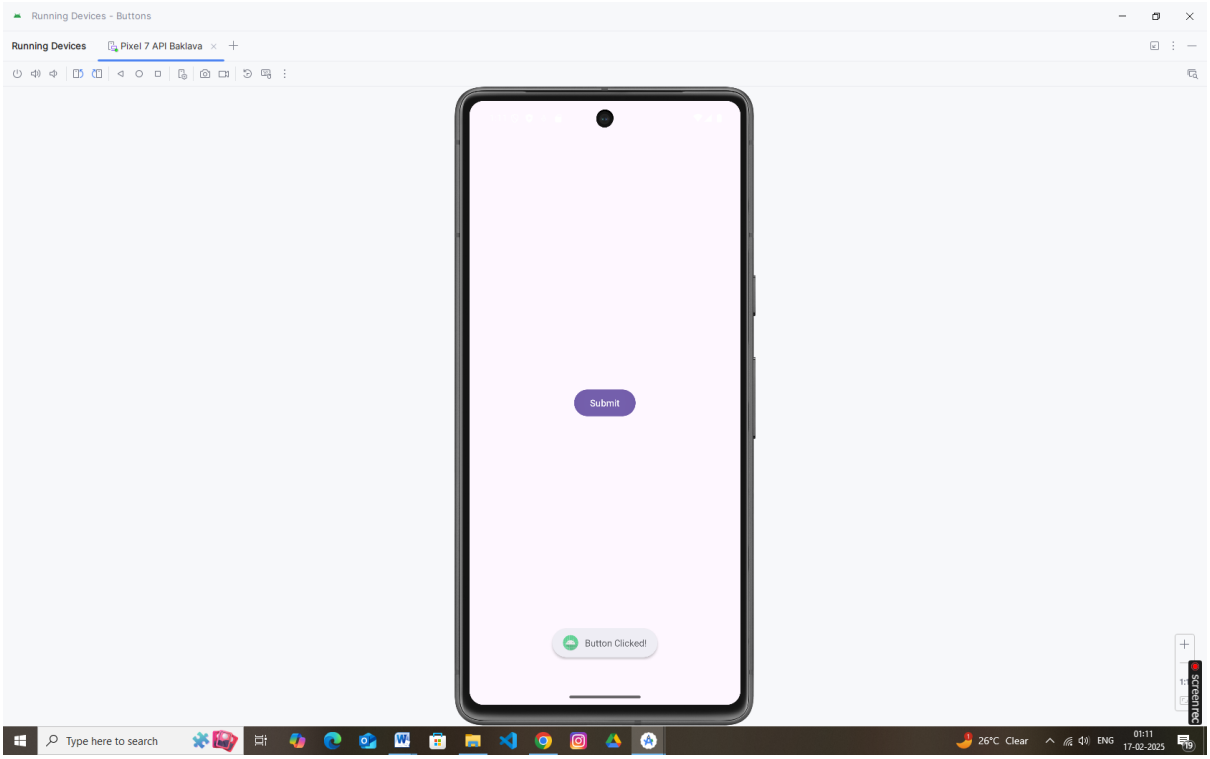
Button is a subclass of TextView class and compound button is the subclass of Button class. On a button we can perform different actions or events like click event, pressed event, touch event etc.

XML Example:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <Button
        android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Submit" />


</LinearLayout>
```

Java Example:

```
package com.example.buttons;

import android.os.Bundle;
import android.widget.Button;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnSubmit = findViewById(R.id.btnSubmit);
        btnSubmit.setOnClickListener(v ->
                Toast.makeText(this, "Button Clicked!",
Toast.LENGTH_SHORT).show()
        );

    }
}
```



2. Checkboxes

In Android, CheckBox is a type of two state button either unchecked or checked in Android. Or you can say it is a type of on/off switch that can be toggled by the users. You should use checkbox when presenting a group of selectable options to users that are not mutually exclusive. CompoundButton is the parent class of CheckBox class.

In android there is a lot of usage of check box. For example, to take survey in Android app we can list few options and allow user to choose using CheckBox. The user will simply checked these checkboxes rather than type their own option in EditText. Another very common use of CheckBox is as remember me option in Login form.

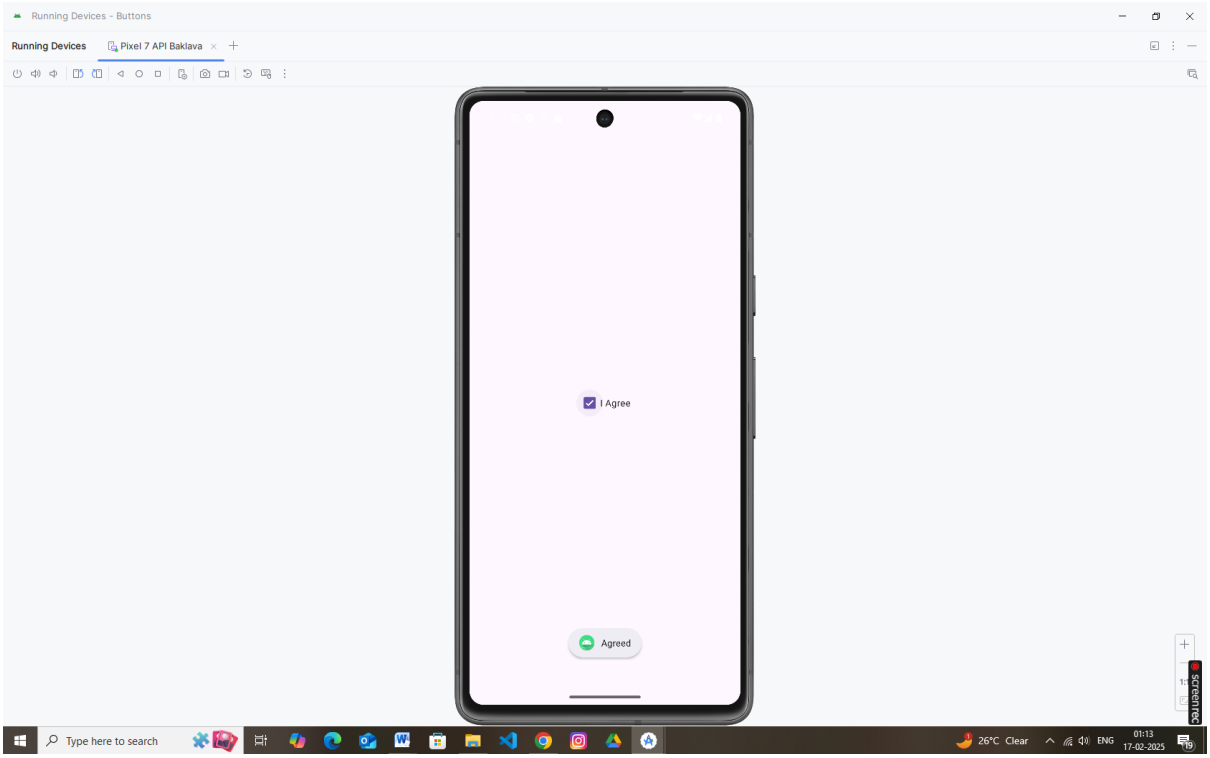
XML Example:

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

```
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <CheckBox
        android:id="@+id/checkboxAgree"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I Agree" />


</LinearLayout>
```

Java Example:

```
package com.example.buttons;

import android.os.Bundle;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        CheckBox checkboxAgree = findViewById(R.id.checkboxAgree);
        checkboxAgree.setOnCheckedChangeListener((buttonView, isChecked) -> {
            String msg = isChecked ? "Agreed" : "Not Agreed";
            Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();
        });


    }
}
```



3. Radio Buttons

In Android, RadioButton are mainly used together in a RadioGroup.
In RadioGroup checking the one radio button out of several radio button added in it will automatically unchecked all the others. It means at one time we can checked only one radio button from a group of radio buttons which belong to same radio group. The most common use of radio button is in Quiz Android App code.

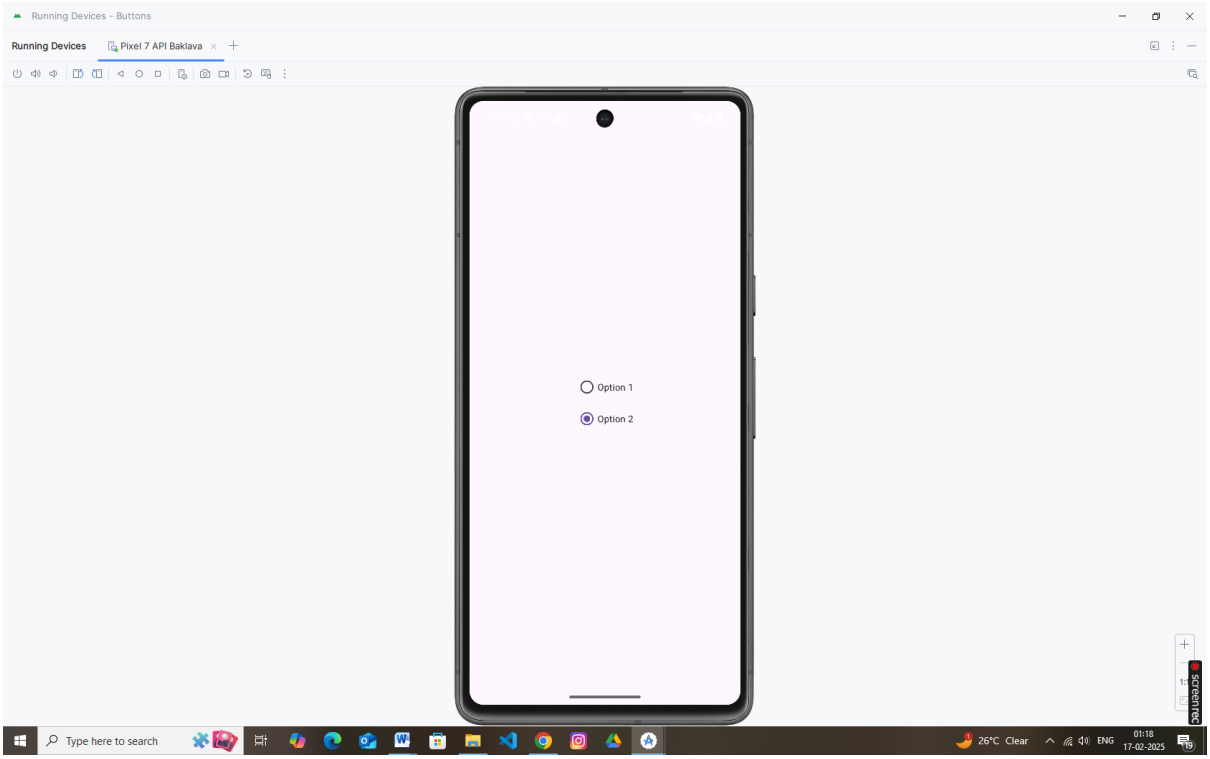
XML Example (with RadioGroup):

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <RadioGroup
        android:id="@+id/radioGroup"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
```

```
        <RadioButton
            android:id="@+id/radioOption1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Option 1" />

        <RadioButton
            android:id="@+id/radioOption2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Option 2" />
    </RadioGroup>



</LinearLayout>
```

Java Example:

```
package com.example.buttons;

import android.os.Bundle;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        RadioGroup radioGroup = findViewById(R.id.radioGroup);
        radioGroup.setOnCheckedChangeListener((group, checkedId) -> {
            RadioButton radioButton = findViewById(checkedId);
            Toast.makeText(this, "Selected: " + radioButton.getText(),
Toast.LENGTH_SHORT).show();
        });


    }
}
```



4. Toggle Buttons

In Android, ToggleButton is used to display checked and unchecked state of a button. ToggleButton basically an off/on button with a light indicator which indicate the current state of toggle button. The most simple example of ToggleButton is doing on/off in sound, Bluetooth, wifi, hotspot etc. It is a subclass of compoundButton.

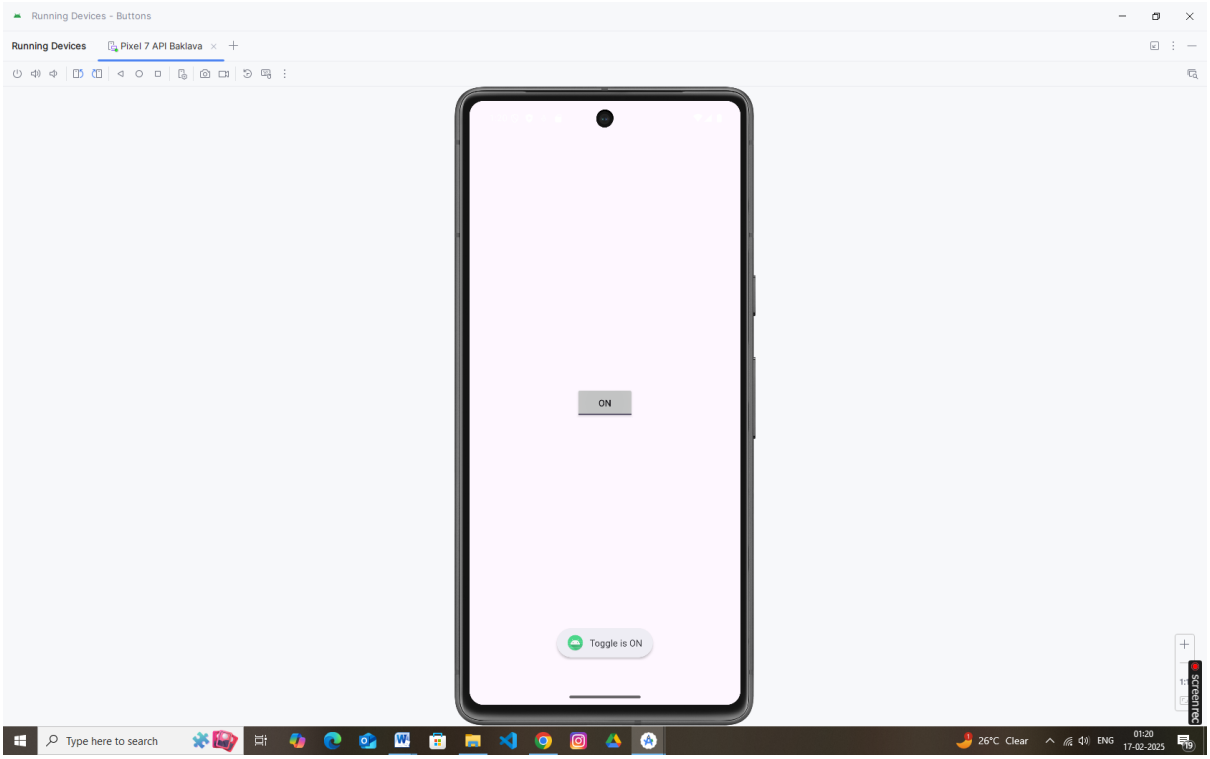
XML Example:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="ON"
        android:textOff="OFF" />
```

```
</LinearLayout>
```

Java Example:

```
package com.example.buttons;

import android.os.Bundle;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Toast;
import android.widget.ToggleButton;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        ToggleButton toggleButton = findViewById(R.id.toggleButton);
        toggleButton.setOnCheckedChangeListener((buttonView, isChecked) -> {
            String status = isChecked ? "Toggle is ON" : "Toggle is OFF";
            Toast.makeText(this, status, Toast.LENGTH_SHORT).show();
        });


    }
}
```



5. Spinners

In Android, Spinner provides a quick way to select one value from a set of values. Android spinners are nothing but the drop down-list seen in other programming languages. In a default state, a spinner shows its currently selected value. It provides a easy way to select a value from a list of values.

In Simple Words we can say that a spinner is like a combo box of AWT or swing where we can select a particular item from a list of items. Spinner is a sub class of AsbSpinner class.

Important Note: Spinner is associated with Adapter view so to fill the data in spinner we need to use one of the Adapter class.

XML Example:

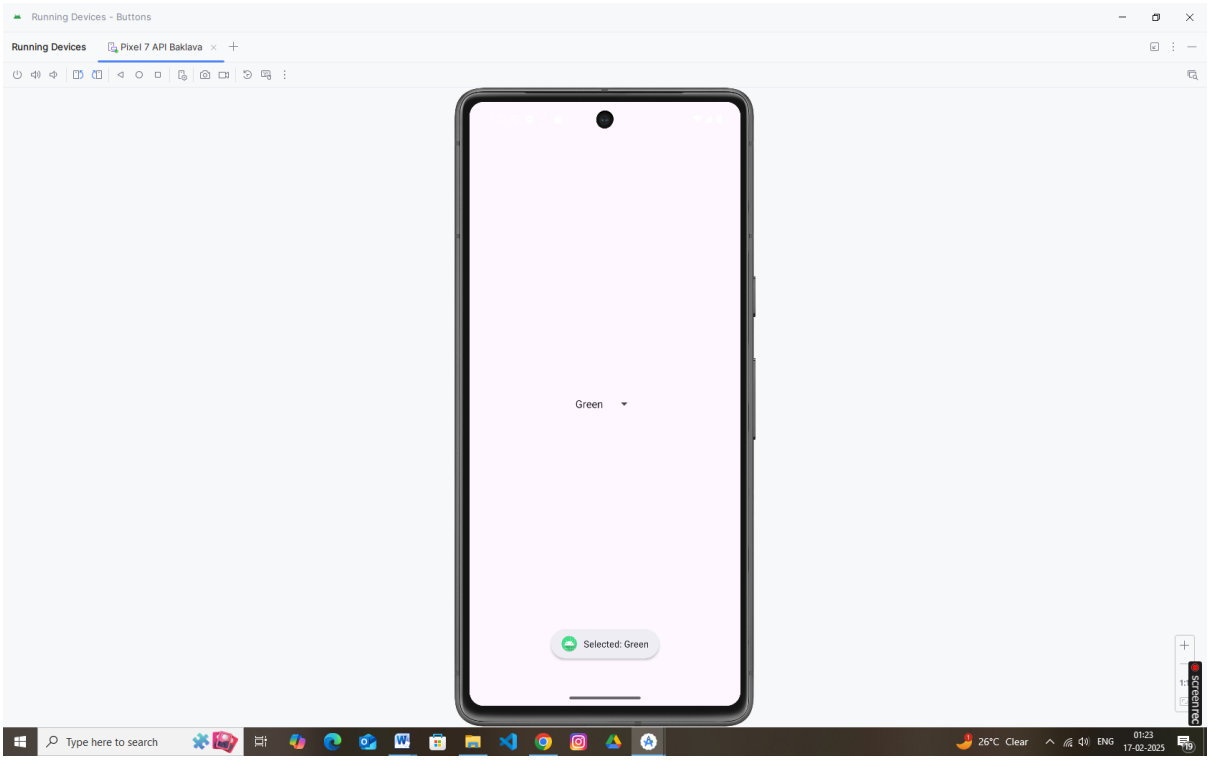
```
<?xml version="1.0" encoding="utf-8"?>
<!-- res/layout/activity_main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <Spinner
        android:id="@+id/spinnerOptions"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />



</LinearLayout>
```

Java Example:

```
package com.example.buttons;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Spinner;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

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

        Spinner spinner = findViewById(R.id.spinnerOptions);
        String[] options = {"Red", "Green", "Blue"};
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, options);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item
);
        spinner.setAdapter(adapter);

        spinner.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int
position, long id) {
                Toast.makeText(getApplicationContext(), "Selected: " +
options[position], Toast.LENGTH_SHORT).show();
```

```
            }
            @Override
            public void onNothingSelected(AdapterView<?> parent) {}
        });


    }
}
```



6. Input Events

In android, **Input Events** are used to capture the events, such as button clicks, edittext touch, etc. from the View objects that defined in a user interface of our application, when the user interacts with it.

To handle input events in android, the views must have in place an **event listener**. The View class, from which all UI components are derived contains a wide range event listener interfaces and each listener interface contains an abstract declaration for a

callback method. To respond to an event of a particular type, the view must register an appropriate event listener and implement the corresponding callback method.

For example, if a button is to respond to a click event it must register to **View.onClickListener** event listener and implement the corresponding **onClick()** callback method. In application when a button click event is detected, the Android framework will call the **onClick()** method of that particular view.

Generally, to handle input events we use **Event Listeners** and **Event Handling** in android applications to listen for user interactions and to extend a View class, in order to build a custom component.

Android Event Listeners

In android, **Event Listener** is an interface in the View class that contains a single call-back method. These methods will be called by the Android framework when the View which is registered with the listener is triggered by user interaction with the item in UI.

The following are the call-back methods included in the event listener interface.

Method	Description
onClick()	This method is called when the user touches or focuses on the item using navigation-keys or trackball or presses on the "enter" key or presses down on the trackball.
onLongClick()	This method is called when the user touches and holds the item or focuses on the item using navigation-keys or trackball and presses and holds "enter" key or presses and holds down on the trackball (for one second).
onFocusChange()	This method is called when the user navigates onto or away from the item, using the navigation-keys or trackball.

Method	Description
onKey()	This method is called when the user is focused on the item and presses or releases a hardware key on the device.
onTouch()	This method is called when the user performs a touch event, including a press, a release, or any movement gesture on the screen.
onCreateContextMenu()	This method is called when a Context Menu is being built.

Example (Button Click):

```
Button btnAction = findViewById(R.id.btnSubmit);
btnAction.setOnClickListener(v -> {
    // Handle click event here
    Toast.makeText(this, "Button was clicked!", Toast.LENGTH_SHORT).show();
});
```

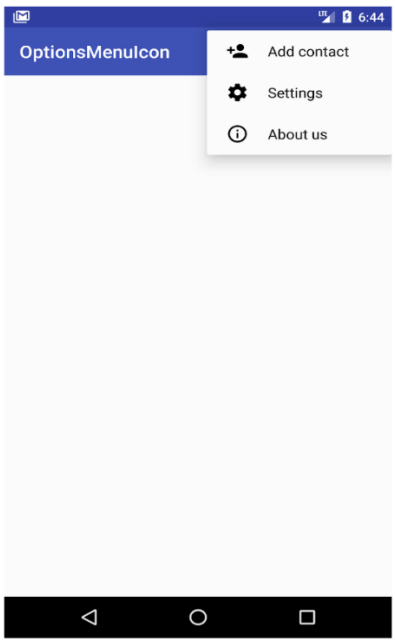
You can also override other event methods (like onTouchEvent or onKeyDown) in your activity for more granular control.

7. Menus

The menu is a part of the User Interface (UI) component, used to handle some common functionality around the app. To utilize the menu, you should define it in a separate XML file and use that file in your app based on your requirements. You can also use menu APIs to represent user actions and other options in your app activities.
Different types of menus
Android provides three types of menus. They are as follows:

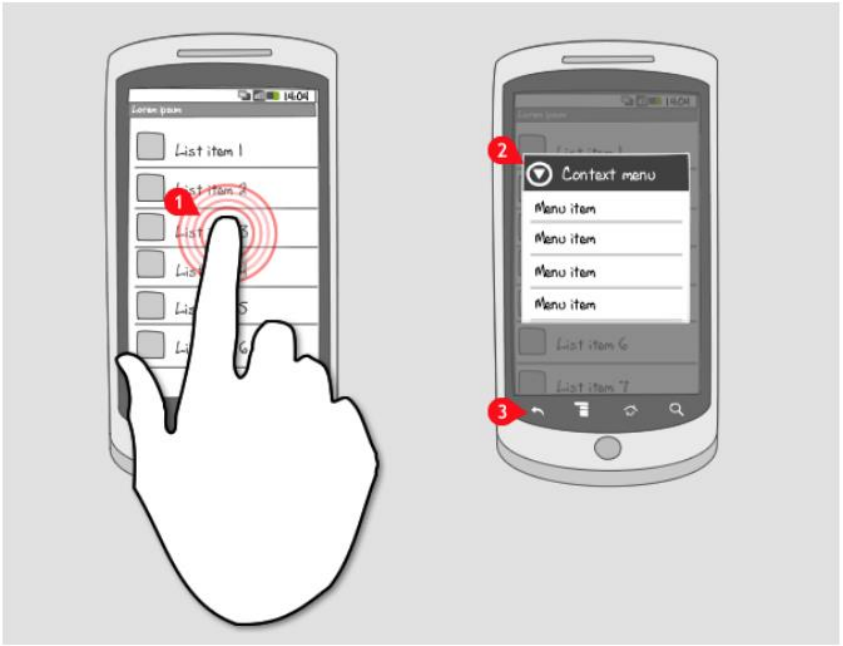
i. Option Menu

This type of menu is a primary collection of menu items in an app and is useful for actions that have a global impact on the searching app. The Option Menu can be used for settings, searching, deleting items, sharing, etc.



ii. Context Menu

This type of menu is a floating menu that only appears when a user presses for a long time on an element and is useful for elements that affect the selected content or context frame.



iii. Popup Menu

Using Popup Menu we can display a list of items in a vertical list which presents the view that invokes the menu. Popup Menu is useful since it can provide an overflow of actions which are related to any specific content.

To define a menu in the XML file

In this step, we will write the menu's code in an XML format to define the type of menu and its items. First, you should create a new menu folder inside of your project resource folder (res/menu) to define the menu. Add a new XML file (res/menu/file.xml) to build your menu. This XML (res/menu/file.xml) file can be given any name that you provide. There are the following important elements of a menu:

1. **<menu>**: A **<menu>** element defines a menu, which is a container for menu items that holds one or more elements. It must be the root of a file.

2. **<items>**: This element is used to create items in the menu. An **<items>** element can contain a nested **<menu>** element to create a submenu.

3. **<group>**: This element is an optional, invisible container for **<item>** elements. **<group>** allows categorizing menu items so they share properties such as active state and visibility.

8. Toast

In android, Toast is a small popup notification that is used to display an information about the operation which we performed in our app. The Toast will show the message for a small period of time and it will disappear automatically after a timeout.

Generally, the size of Toast will be adjusted based on the space required for the message and it will be displayed on the top of the main content of activity for a short period of time.

For example, some of the apps will show a message like "**Press again to exit**" in toast, when we pressed a back button on the home page or showing a message like "**saved successfully**" toast when we click on the button to save the details.

If you observe above syntax, we defined a Toast **notification** using **makeText()** method with three parameters, those are

Parameter	Description
context	It's our application context.
message	It's our custom message which we want to show in Toast notification.
duration	It is used to define the duration for notification to display on the screen.

We have two ways to define the Toast **duration**, either in **LENGTH_SHORT** or **LENGTH_LONG** to display the toast notification for a short or longer period of time.

```
Toast.makeText(getApplicationContext(), "This is a Toast message",
Toast.LENGTH_SHORT).show();
```

9. Dialogs

Android AlertDialog can be used to display the dialog message with OK and Cancel buttons. It can be used to interrupt and ask the user about his/her choice to continue or discontinue.

Android AlertDialog is composed of three regions: title, content area and action buttons.

Android AlertDialog is the subclass of Dialog class.

Example: AlertDialog

```
new AlertDialog.Builder(this)
        .setTitle("Exit")
        .setMessage("Are you sure you want to exit?")
        .setPositiveButton("Yes", (dialog, which) -> finish())
        .setNegativeButton("No", null)
        .show();
```

10. Styles and Themes

Styles and themes on Android let you separate the details of your app design from the UI structure and behavior, similar to stylesheets in web design.

A *style* is a collection of attributes that specifies the appearance for a single View. A style can specify attributes such as font color, font size, background color, and much more.

A *theme* is a collection of attributes that's applied to an entire app, activity, or view hierarchy—not just an individual view. When you apply a theme, every view in the app or activity applies each of the theme's attributes that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

Define a Style (res/values/styles.xml):

```
<resources>
    <style name="CustomButtonStyle">
        <item name="android:background">#FF5722</item>
        <item name="android:textColor">#FFFFFF</item>
        <item name="android:padding">10dp</item>
    </style>
</resources>
```

Apply Style in XML:

```
<Button
    style="@style/CustomButtonStyle"
    android:id="@+id/styledButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Styled Button" />
```

Themes:
Themes are applied at the app or activity level (see the AndroidManifest.xml or your styles.xml for your app theme).

11. Creating Lists (Using ListView)

Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.

XML Layout (res/layout/activity_list.xml):

```
<ListView
    android:id="@+id/listView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Java Example:

```
ListView listView = findViewById(R.id.listView);
String[] items = {"Apple", "Banana", "Cherry", "Date", "Elderberry"};
ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
        android.R.layout.simple_list_item_1, items);
listView.setAdapter(adapter);
```

12. Custom Lists (Using Custom Adapter for ListView)

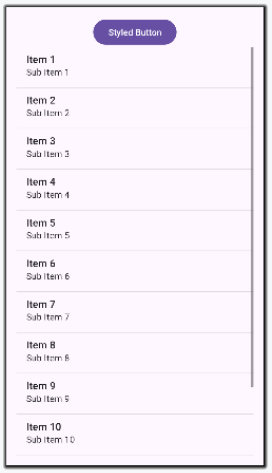
In this tutorial we'll use a **CustomAdapter** that populates the custom rows of the Android ListView with an ArrayList. Also to enhance the user experience, we'll animate the ListView while scrolling.

Android ListView Custom Adapter Overview

The simplest Adapter to populate a view from an ArrayList is the ArrayAdapter. That's what we'll implement in this tutorial. There are other adapters as well, such as the CursorAdapter which binds directly to a result set from a Local SQLite Database and it uses a Cursor as it's data source.

Custom Row Layout (res/layout/custom_row.xml):

```
<Button
    style="@style/CustomButtonStyle"
    android:id="@+id/styledButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Styled Button" />


<ListView
    android:id="@+id/listView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>


<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp">

    <ImageView
        android:id="@+id/imgIcon"
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:src="@mipmap/ic_launcher" />

    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="10dp"
        android:textSize="18sp" />
</LinearLayout>
```


Styled Button
Item 1
Sub Item 1
Item 2
Sub Item 2
Item 3
Sub Item 3
Item 4
Sub Item 4
Item 5
Sub Item 5
Item 6
Sub Item 6
Item 7
Sub Item 7
Item 8
Sub Item 8
Item 9
Sub Item 9
Item 10
Sub Item 10