**Point 1:**

**what is widget in android development ?**

In Android development, a widget is a small application view that can be embedded in other applications (like the home screen) and receive periodic updates. Widgets provide a quick view of important information and provide users with easy access to specific app functionalities without having to open the app itself.

**Key Components of Android Widgets**

1. **AppWidgetProvider**:
   * This is a BroadcastReceiver that handles the widget's lifecycle events such as updates, enabling, disabling, and deleting.
   * You define this in your manifest and specify the intents it should respond to.
2. **AppWidgetProviderInfo**:
   * This is an XML file where you specify metadata about your widget, such as its initial layout, update frequency, and resize options.
3. **RemoteViews**:
   * This class is used to define the layout for the widget. RemoteViews are used because widgets run in another process (the home screen's process), so you cannot use standard View objects directly.
   * You create a layout XML file and then use RemoteViews to manipulate the UI elements.
4. **Layouts**:
   * Layouts for widgets are defined in XML, similar to standard Android views, but they have limitations due to the RemoteViews model. Not all view types and attributes are supported.
5. **Update Mechanism**:
   * Widgets can be updated periodically using the updatePeriodMillis attribute in the AppWidgetProviderInfo XML file.
   * You can also trigger updates using alarms or other methods to refresh the widget content.

**Basic Steps to Create a Widget**

1. **Define the Widget Layout**:
   * Create an XML layout file in the res/layout directory for the widget.
2. <!-- res/layout/widget\_layout.xml -->
3. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4. android:layout\_width="match\_parent"
5. android:layout\_height="match\_parent"
6. android:orientation="vertical"
7. android:padding="8dp">
8. <TextView
9. android:id="@+id/widget\_text"
10. android:layout\_width="wrap\_content"
11. android:layout\_height="wrap\_content"
12. android:text="Hello, Widget!"
13. android:textSize="18sp" />
14. </LinearLayout>
15. **Create the AppWidgetProviderInfo XML File**:
    * Define the widget's metadata in a file in the res/xml directory.
16. <!-- res/xml/widget\_info.xml -->
17. <appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
18. android:initialLayout="@layout/widget\_layout"
19. android:minWidth="180dp"
20. android:minHeight="90dp"
21. android:updatePeriodMillis="86400000"
22. android:widgetCategory="home\_screen" />
23. **Implement the AppWidgetProvider**:
    * Create a class that extends AppWidgetProvider to manage the widget's behavior.
24. public class MyAppWidgetProvider extends AppWidgetProvider {
25. @Override
26. public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
27. // Update all widgets
28. for (int appWidgetId : appWidgetIds) {
29. RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget\_layout);
30. // Perform updates on views
31. appWidgetManager.updateAppWidget(appWidgetId, views);
32. }
33. }
34. }
35. **Declare the Widget in the Manifest**:
    * Add the widget provider and receiver to the AndroidManifest.xml file.
36. <receiver android:name=".MyAppWidgetProvider">
37. <intent-filter>
38. <action android:name="android.appwidget.action.APPWIDGET\_UPDATE" />
39. </intent-filter>
40. <meta-data
41. android:name="android.appwidget.provider"
42. android:resource="@xml/widget\_info" />
43. </receiver>

**Updating Widgets**

To update a widget's content, you can use:

* **Periodic updates** via the updatePeriodMillis attribute.
* **Explicit updates** via the AppWidgetManager.updateAppWidget method, often triggered by an AlarmManager, JobScheduler, or WorkManager.

**Example of Updating Widget on Button Click**

1. **Add a Button to the Widget Layout**:
2. <!-- res/layout/widget\_layout.xml -->
3. <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4. android:layout\_width="match\_parent"
5. android:layout\_height="match\_parent"
6. android:orientation="vertical"
7. android:padding="8dp">
8. <TextView
9. android:id="@+id/widget\_text"
10. android:layout\_width="wrap\_content"
11. android:layout\_height="wrap\_content"
12. android:text="Hello, Widget!"
13. android:textSize="18sp" />
14. <Button
15. android:id="@+id/widget\_button"
16. android:layout\_width="wrap\_content"
17. android:layout\_height="wrap\_content"
18. android:text="Update" />
19. </LinearLayout>
20. **Handle Button Clicks in AppWidgetProvider**:
21. public class MyAppWidgetProvider extends AppWidgetProvider {
22. private static final String ACTION\_BUTTON\_CLICK = "ACTION\_BUTTON\_CLICK";
23. @Override
24. public void onReceive(Context context, Intent intent) {
25. super.onReceive(context, intent);
26. if (ACTION\_BUTTON\_CLICK.equals(intent.getAction())) {
27. AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);
28. ComponentName thisWidget = new ComponentName(context, MyAppWidgetProvider.class);
29. int[] appWidgetIds = appWidgetManager.getAppWidgetIds(thisWidget);
30. // Update all widgets
31. for (int appWidgetId : appWidgetIds) {
32. RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget\_layout);
33. views.setTextViewText(R.id.widget\_text, "Updated!");
34. appWidgetManager.updateAppWidget(appWidgetId, views);
35. }
36. }
37. }
38. @Override
39. public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
40. // Set up the button click pending intent
41. for (int appWidgetId : appWidgetIds) {
42. Intent intent = new Intent(context, MyAppWidgetProvider.class);
43. intent.setAction(ACTION\_BUTTON\_CLICK);
44. PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, intent, PendingIntent.FLAG\_UPDATE\_CURRENT);
45. RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget\_layout);
46. views.setOnClickPendingIntent(R.id.widget\_button, pendingIntent);
47. appWidgetManager.updateAppWidget(appWidgetId, views);
48. }
49. }
50. }

**Conclusion**

Widgets are a powerful feature in Android that allow users to access key information and functionalities directly from their home screens. Creating and managing widgets involves defining their layout, handling updates through AppWidgetProvider, and ensuring a smooth user experience through timely and efficient updates.

**Point 2:**

**Is button an widget ?**

In the context of Android development, a button is not considered a widget in the specific sense that widgets are small app components that can be placed on the home screen or lock screen. However, a button is a UI element (a view) that can be part of a widget's layout.

To clarify:

**Definitions**

1. **Widget (App Widget)**:
   * A small, interactive component that users can add to their home screen.
   * Examples include weather widgets, calendar widgets, and music control widgets.
   * Implemented using AppWidgetProvider, RemoteViews, and defined in an XML file with metadata.
2. **Button (UI Element)**:
   * A basic interactive element in Android's UI toolkit.
   * Used within activities, fragments, and app widgets to perform actions when clicked.
   * Part of the Android view hierarchy, can be included in any layout XML.

**Usage of Button in a Widget**

While a button itself is not an app widget, you can include a button within the layout of a widget to add interactivity. This allows users to interact with the widget by clicking the button, which can trigger certain actions defined in your AppWidgetProvider.

**Example**

Let's consider an example where a button is included in a widget's layout:

**Layout File**

<!-- res/layout/widget\_layout.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:padding="8dp">

<TextView

android:id="@+id/widget\_text"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, Widget!"

android:textSize="18sp" />

<Button

android:id="@+id/widget\_button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Update" />

</LinearLayout>

**AppWidgetProvider Implementation**

public class MyAppWidgetProvider extends AppWidgetProvider {

private static final String ACTION\_BUTTON\_CLICK = "ACTION\_BUTTON\_CLICK";

@Override

public void onReceive(Context context, Intent intent) {

super.onReceive(context, intent);

if (ACTION\_BUTTON\_CLICK.equals(intent.getAction())) {

AppWidgetManager appWidgetManager = AppWidgetManager.getInstance(context);

ComponentName thisWidget = new ComponentName(context, MyAppWidgetProvider.class);

int[] appWidgetIds = appWidgetManager.getAppWidgetIds(thisWidget);

// Update all widgets

for (int appWidgetId : appWidgetIds) {

RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget\_layout);

views.setTextViewText(R.id.widget\_text, "Updated!");

appWidgetManager.updateAppWidget(appWidgetId, views);

}

}

}

@Override

public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {

// Set up the button click pending intent

for (int appWidgetId : appWidgetIds) {

Intent intent = new Intent(context, MyAppWidgetProvider.class);

intent.setAction(ACTION\_BUTTON\_CLICK);

PendingIntent pendingIntent = PendingIntent.getBroadcast(context, 0, intent, PendingIntent.FLAG\_UPDATE\_CURRENT);

RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.widget\_layout);

views.setOnClickPendingIntent(R.id.widget\_button, pendingIntent);

appWidgetManager.updateAppWidget(appWidgetId, views);

}

}

}

**Manifest Declaration**

<receiver android:name=".MyAppWidgetProvider">

<intent-filter>

<action android:name="android.appwidget.action.APPWIDGET\_UPDATE" />

</intent-filter>

<meta-data

android:name="android.appwidget.provider"

android:resource="@xml/widget\_info" />

</receiver>

In this example, the button within the widget layout allows the user to update the text displayed in the widget. This demonstrates how interactive elements like buttons can be embedded in widgets to enhance their functionality.

**Conclusion**

While a button itself is not an app widget, it can be used within a widget's layout to add interactive capabilities. Widgets, in the specific sense of Android development, refer to small apps or components that live on the home screen and provide quick access to information and functionality.

**Point 3:**

**Then what is fragment ?**

In Android development, a Fragment is a reusable portion of the user interface in an activity. Fragments allow for modular design, making it easier to manage UI components and logic within an activity. Each fragment has its own lifecycle, which is closely tied to the lifecycle of its host activity.

**Key Characteristics of Fragments**

1. **Modular and Reusable**:
   * Fragments can be reused in multiple activities, allowing for a more modular and maintainable codebase.
2. **Lifecycle**:
   * Fragments have their own lifecycle methods that are parallel to the activity lifecycle, such as onCreate(), onCreateView(), onStart(), onResume(), onPause(), onStop(), and onDestroyView().
3. **Layout and Logic Separation**:
   * Fragments can define their own layout and handle their own logic, which can be dynamically added, removed, or replaced within an activity.
4. **Support for Multiple Screens**:
   * Fragments facilitate building UIs that can adapt to different screen sizes and orientations. For instance, a tablet can display multiple fragments side-by-side, while a phone might display them one at a time.

**Basic Usage of Fragments**

**Creating a Fragment**

To create a fragment, you extend the Fragment class and override its lifecycle methods.

public class ExampleFragment extends Fragment {

@Override

public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {

// Inflate the layout for this fragment

return inflater.inflate(R.layout.fragment\_example, container, false);

}

}

**Defining Fragment Layout**

You define the fragment's layout in an XML file.

<!-- res/layout/fragment\_example.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:padding="16dp">

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, Fragment!"

android:textSize="18sp" />

</LinearLayout>

**Adding a Fragment to an Activity**

Fragments can be added to an activity's layout statically or dynamically.

**Statically Adding a Fragment**

In your activity's layout XML, use the <fragment> tag to statically add a fragment.

<!-- res/layout/activity\_main.xml -->

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

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<fragment

android:id="@+id/fragment\_container"

android:name="com.example.ExampleFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent" />

</FrameLayout>

**Dynamically Adding a Fragment**

You can dynamically add, remove, or replace fragments in your activity code using the FragmentManager.

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Check if the fragment container is present

if (findViewById(R.id.fragment\_container) != null) {

// If we're being restored from a previous state, then don't do anything

if (savedInstanceState != null) {

return;

}

// Create a new Fragment to be placed in the activity layout

ExampleFragment firstFragment = new ExampleFragment();

// Add the fragment to the 'fragment\_container' FrameLayout

getSupportFragmentManager().beginTransaction()

.add(R.id.fragment\_container, firstFragment).commit();

}

}

}

**Fragment Lifecycle**

A fragment's lifecycle is similar to an activity's lifecycle but has additional methods for managing the fragment's UI.

* **onAttach()**: Called when the fragment is first attached to its context.
* **onCreate()**: Called to do initial creation of the fragment.
* **onCreateView()**: Called to create the fragment's view hierarchy.
* **onActivityCreated()**: Called when the activity's onCreate() method has returned.
* **onStart()**: Called when the fragment is visible to the user.
* **onResume()**: Called when the fragment is interacting with the user.
* **onPause()**: Called when the fragment is no longer interacting with the user.
* **onStop()**: Called when the fragment is no longer visible to the user.
* **onDestroyView()**: Called to clean up resources associated with the fragment's view.
* **onDestroy()**: Called to clean up resources associated with the fragment.
* **onDetach()**: Called when the fragment is detached from its context.

**Communication Between Fragments and Activities**

Fragments often need to communicate with their host activity or other fragments. This can be done through interfaces.

public class ExampleFragment extends Fragment {

OnFragmentInteractionListener mListener;

@Override

public void onAttach(Context context) {

super.onAttach(context);

if (context instanceof OnFragmentInteractionListener) {

mListener = (OnFragmentInteractionListener) context;

} else {

throw new RuntimeException(context.toString()

+ " must implement OnFragmentInteractionListener");

}

}

// Interface to communicate with the host activity

public interface OnFragmentInteractionListener {

void onFragmentInteraction(String data);

}

// Method to send data to the activity

public void sendDataToActivity() {

if (mListener != null) {

mListener.onFragmentInteraction("Hello from Fragment");

}

}

}

In the activity:

public class MainActivity extends AppCompatActivity implements ExampleFragment.OnFragmentInteractionListener {

@Override

public void onFragmentInteraction(String data) {

// Handle interaction from the fragment

}

}

**Conclusion**

Fragments are essential for creating flexible and reusable UI components in Android applications. They help in building dynamic and adaptable interfaces that can handle various screen sizes and orientations, contributing to a modular and maintainable codebase.

**Point 4.1:**

**Is textview an widget (tell me yes or no --- intervier asked..) ?**

In Android development terminology, a TextView is not considered a "widget" in the specific sense of home screen widgets. However, a TextView is a type of user interface element, also referred to as a "view" or "UI widget."

**Clarification of Terms**

1. **Home Screen Widget**:
   * These are mini-applications that run on the home screen. Examples include weather widgets, clock widgets, and calendar widgets. They are created using AppWidgetProvider and have their own lifecycle and update mechanisms.
   * They provide quick access to app functionality and information directly on the home screen without opening the app.
2. **UI Widget (View)**:
   * These are the basic building blocks for user interface components in Android. Examples include TextView, Button, ImageView, EditText, etc.
   * They are used within activities, fragments, or home screen widgets to display content and interact with users.
   * TextView is a type of UI widget or view used to display text to the user.

**Understanding TextView**

A TextView is a view that displays text to the user and optionally allows them to edit it. It is one of the most commonly used UI widgets in Android applications.

**Basic Usage**

Here's a basic example of using a TextView in an activity layout:

**XML Layout**

<!-- 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:padding="16dp">

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, World!"

android:textSize="18sp" />

</LinearLayout>

**Activity Code**

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Get the TextView from the layout

TextView textView = findViewById(R.id.textView);

// Set some text programmatically

textView.setText("Welcome to Android Development!");

}

}

**Properties and Methods**

TextView comes with many properties and methods that allow you to customize its appearance and behavior.

**Common Properties**

* android:text: Sets the text to be displayed.
* android:textSize: Sets the size of the text.
* android:textColor: Sets the color of the text.
* android:gravity: Sets the alignment of the text within the view.
* android:padding: Sets the padding inside the view.

**Common Methods**

* setText(String text): Sets the text to be displayed.
* setTextColor(int color): Sets the color of the text.
* setTextSize(float size): Sets the size of the text.
* getText(): Gets the current text displayed in the TextView.

**Example of Customizing TextView**

Here's an example of how you might customize a TextView both in XML and programmatically:

**XML Layout**

<!-- 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:padding="16dp">

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, World!"

android:textSize="18sp"

android:textColor="#FF0000"

android:gravity="center"

android:padding="8dp" />

</LinearLayout>

**Activity Code**

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Get the TextView from the layout

TextView textView = findViewById(R.id.textView);

// Set text programmatically

textView.setText("Welcome to Android Development!");

// Set text color programmatically

textView.setTextColor(Color.BLUE);

// Set text size programmatically

textView.setTextSize(24);

// Set padding programmatically

textView.setPadding(16, 16, 16, 16);

}

}

**Conclusion**

While a TextView is not a home screen widget, it is a fundamental UI widget (or view) in Android used to display text. Understanding the distinction between home screen widgets and UI widgets is essential for Android development. Home screen widgets are special components that live on the home screen and provide interactive content directly to the user, whereas UI widgets like TextView are used within activities and fragments to build the user interface.

**Point 4.2:**

**Is textview an widget (tell me yes or no & explain --- intervier asked..) ?**

Yes, a TextView is considered a widget in Android development.

**Explanation:**

In the context of Android development, the term "widget" can refer to different things based on context:

1. **UI Widgets**:
   * These are basic user interface elements that are part of the Android View system. Examples include TextView, Button, ImageView, EditText, and others.
   * These widgets are the building blocks of an app's user interface and are defined in XML layout files or created programmatically.
2. **App Widgets**:
   * These are mini-applications that can be embedded in other applications (like the home screen) and provide users with quick access to information or functionality.
   * Examples include calendar widgets, weather widgets, and clock widgets.

**TextView as a UI Widget**

A TextView is a standard UI widget in Android. It is a subclass of View and is used to display text to the user. You can customize a TextView by setting various properties like text size, color, and style. It can also handle user interactions, such as being clicked, although typically a TextView is used for display purposes rather than for interaction.

**Example**

Here is an example of a TextView defined in an XML layout file:

<!-- 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:padding="16dp">

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, TextView!"

android:textSize="18sp"

android:textColor="#000000" />

</LinearLayout>

In this example, TextView is used to display the text "Hello, TextView!" on the screen.

**Conclusion**

In summary, a TextView is indeed a widget in the context of Android UI elements. It is a fundamental component used to display text in Android applications. The term "widget" in Android can refer to both basic UI elements like TextView and specialized components like app widgets on the home screen.

**Point 5.1:**

**Is there any difference between textview element & textview widget ?**

In the context of Android development, there is no difference between a "TextView element" and a "TextView widget." Both terms refer to the same component, which is the TextView class that is used to display text in an Android user interface.

**TextView**

* **Definition**: TextView is a UI component (or widget) that displays text to the user.
* **Class**: It is a part of the Android View hierarchy and is defined by the android.widget.TextView class.
* **Usage**: It is commonly used in XML layout files or created programmatically in an Android app to show text on the screen.

**Example of TextView in XML**

Here’s an example of how a TextView is defined in an XML layout file:

<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="16dp">

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, TextView!"

android:textSize="18sp"

android:textColor="#000000" />

</LinearLayout>

**Example of TextView in Java**

Here’s how a TextView can be created programmatically in Java:

TextView textView = new TextView(this);

textView.setText("Hello, TextView!");

textView.setTextSize(18);

textView.setTextColor(Color.BLACK);

LinearLayout layout = findViewById(R.id.layout);

layout.addView(textView);

**Conclusion**

Whether you refer to it as a "TextView element" or a "TextView widget," you are talking about the same UI component in Android development. It is used to display text within an app's interface and is an essential part of the Android SDK's widget toolkit. The terms are interchangeable and both describe the TextView class that provides this functionality.

**Point 5.2:**

**Is there any difference between textview element & textview widget ?**

In the context of Android development, the terms "TextView element" and "TextView widget" generally refer to the same thing. However, it is important to understand the nuances that might be implied by each term.

**TextView Element**

* **TextView element** typically refers to the TextView as it is used within an XML layout file. It is an XML element that defines a TextView in the user interface layout.
* Example in XML:
* <TextView
* android:id="@+id/textView"
* android:layout\_width="wrap\_content"
* android:layout\_height="wrap\_content"
* android:text="Hello, TextView!"
* android:textSize="18sp"
* android:textColor="#000000" />

**TextView Widget**

* **TextView widget** is a more general term that refers to the TextView component as part of the Android UI toolkit. It emphasizes that TextView is a widget, which is a fundamental building block for creating user interfaces in Android.
* In this context, "widget" refers to any of the basic UI components (views) provided by Android, such as Button, ImageView, EditText, etc.

**Understanding the Terms**

1. **TextView Element**:
   * Focuses on the TextView as it is defined in the layout XML file.
   * Example: When discussing layout design and XML structure, the term "TextView element" might be used.
2. **TextView Widget**:
   * Refers to the TextView as a UI component that can be used in both XML and programmatically in code.
   * Example: When discussing the functionality and properties of UI components, the term "TextView widget" might be used.

**Practical Example**

Using TextView in an activity, both terms could apply, but from different perspectives:

**XML Layout (TextView Element)**

<!-- 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:padding="16dp">

<TextView

android:id="@+id/textView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello, TextView!"

android:textSize="18sp"

android:textColor="#000000" />

</LinearLayout>

**Java Code (TextView Widget)**

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Referencing the TextView widget in the code

TextView textView = findViewById(R.id.textView);

textView.setText("Updated TextView Text");

}

}

**Conclusion**

* There is no functional difference between a "TextView element" and a "TextView widget"; they refer to the same component in Android.
* The distinction lies in the context: "element" is often used when discussing XML layout, while "widget" is used more broadly to refer to UI components in general, whether in XML or code.