

# 1. What is Android?

- Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance*, led by Google, and other companies.
- Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.
- The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.
- On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 **Jelly Bean**. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.
- The source code for Android is available under free and open source software licenses. Google publishes most of the code under the Apache License version 2.0 and the rest, Linux kernel changes, under the GNU General Public License version 2.

## 2. Android 4.1 Jelly Bean SDK features/ *Features of Android*

The Android 4.1 Jelly Bean SDK was released with new features for developers in July 2012. It improves the beauty and simplicity of Android 4.0 and is a major platform release that adds a variety of new features for users and app developers. A few of the big features of this release include the following:

### 1. **Project Butter - UI Performance Improvement**

- Makes the Jelly Bean UI faster and more responsive.
- Adds CPU Touch Responsiveness, increasing CPU performance when the screen is touched.
- Predicts finger movement for faster navigation.

### 2. **Faster Speech Recognition**

- Speech recognition is now faster and offline, eliminating the need for a network connection to convert voice to text.
- Users can dictate to the device without an Internet connection.

### 3. **Improved Notification System**

- Notifications include pictures and lists along with text.
- Users can expand or collapse notifications using gestures.
- Action buttons in notifications enable users to call directly from the notification menu.

### 4. **Supports New Languages**

- Jelly Bean includes support for several languages, including Arabic, Hebrew, Hindi, and Thai.
- Supports bidirectional text.

### 5. **Predictive Keyboard**

- Predicts the next word in the message based on the current context.

### 6. **Auto-Arranging Home Screen**

- Icons and widgets automatically resize and realign to fit the available space.

### 7. **Accessibility for Visually Impaired Users**

- Gesture Mode combined with voice assists visually impaired users in navigating the user interface.

### 8. **Improved Camera App**

- New review mode for captured photos, accessed by swiping from the right of the screen.
- Users can pinch to switch to a film strip view and swipe to delete photos.

### 9. **Better Communication in Jelly Bean**

- Two devices can communicate with Near Field Communication (NFC).
- Android devices can be paired to Bluetooth devices supporting Simple Secure Pairing by tapping them together.

### 10. **Improved Google Voice Search**

- Jelly Bean features a question and answer search method similar to Apple's Siri.

### 11. **Face Unlock**

- Unlocks the device when the user looks at it, and prevents the screen from blacking out.
- Can use a "blink" to confirm a live person is unlocking the device instead of a photo.

### 12. **Google Now**

- Provides users with personalized information through cards.
- Displays relevant information automatically, such as nearby places, transit details, sports scores, weather conditions, etc.

### 13. **Google Play Widgets**

- Provides quick and easy access to movies, games, magazines, and media on the device.
- Suggests new purchases on Google Play.

### 14. **Faster Google Search**

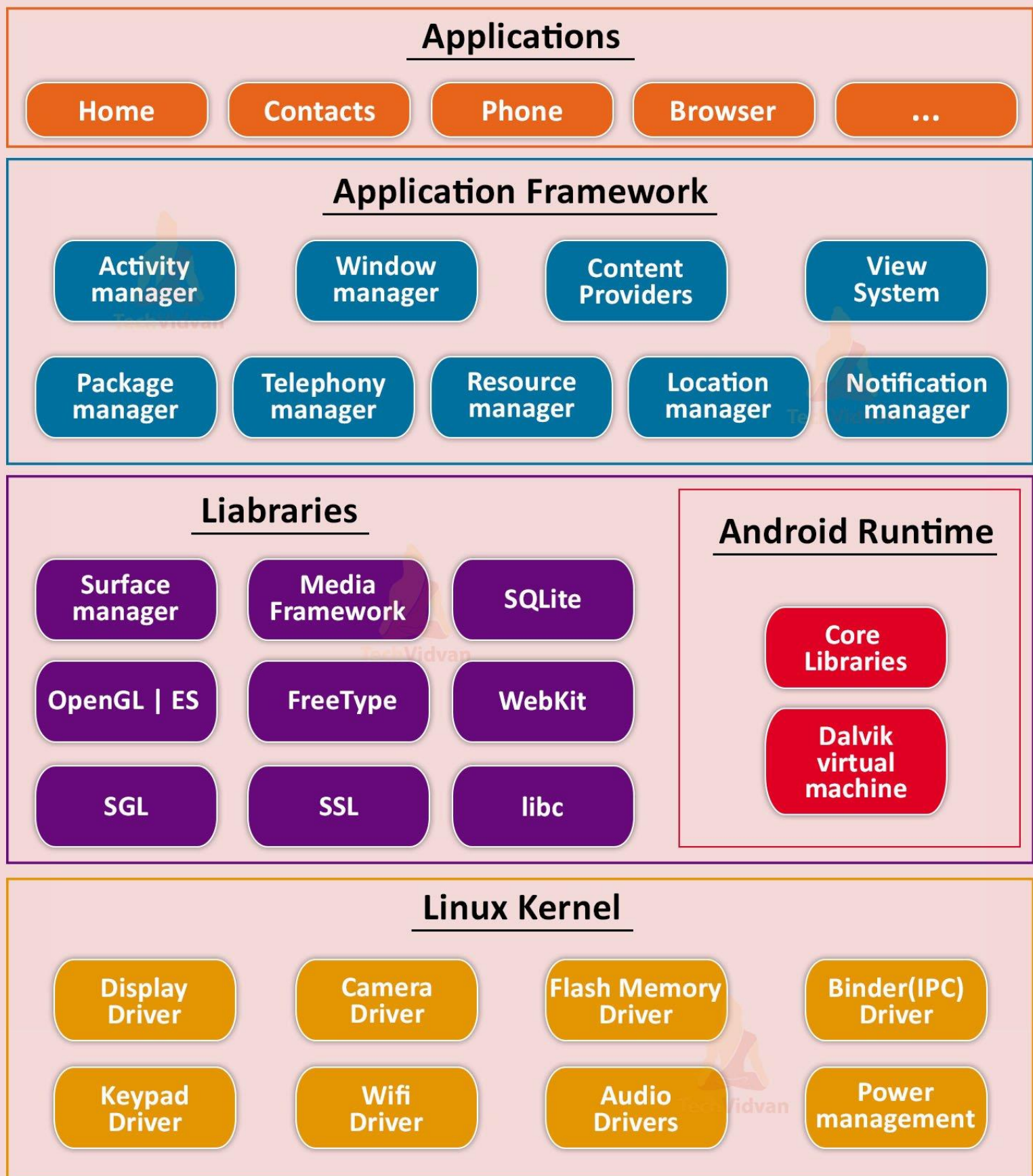
- Google Search can be opened quickly from the lock screen and system bar through gestures or hardware search key.

### 15. **Supports Antipiracy**

- Applications are encrypted with a device-specific key, making it difficult to copy and distribute them illegally.

### 3. Android Architecture:

The Android architecture is based on a layered approach, with each layer responsible for specific functionalities. It follows the principles of the Linux kernel and utilizes various components to provide a rich and seamless user experience. The main components of the Android architecture are as follows:



Android Architecture Diagram:

## **I. Linux Kernel:**

At the core of Android is the Linux kernel, which handles essential tasks like device drivers, memory management, process management, security, and power management. The Linux kernel acts as an abstraction layer between the hardware and the higher-level software layers of the Android operating system.

## **II. Libraries:**

On top of the Linux kernel, Android includes a set of C/C++ libraries that provide core functionalities for the system. These libraries include:

- **LIBC:** The C library that provides basic system calls and standard C functions.
- **SURFACE MANAGER:** Manages the display subsystem.
- **MEDIA LIBRARIES:** Provide support for various multimedia formats and codecs.
- **SQLITE:** A lightweight relational database management system used for data storage.
- **WEBKIT:** The browser engine used for web rendering in Android.
- **OPENGL ES:** A graphics rendering API for 2D and 3D graphics.
- **WIDGET** – A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc

## **III. Android Runtime (ART/Dalvik):**

The Android Runtime (ART) or the older Dalvik is responsible for executing and managing Android applications. It is a virtual machine that runs bytecode compiled from Java source code. ART replaced Dalvik with Android 4.4 (KitKat) as the default runtime. ART provides improved application performance and reduces memory usage through Ahead-of-Time (AOT) compilation.

## **IV. Android Framework:**

The Android Framework is a set of Java APIs that developers use to build applications for the Android platform. It provides a vast range of classes and tools for tasks like UI design, resource management, content providers, location services, and more. The framework acts as an abstraction layer between the application and the lower-level system services.

The Android framework includes the following key services –

- **Activity Manager** – Controls all aspects of the application lifecycle and activity stack.
- **Content Providers** – Allows applications to publish and share data with other applications.
- **Resource Manager** – Provides access to non-code embedded resources such as strings, colors, settings and user interface layouts.
- **Notifications Manager** – Allows applications to display alerts and notifications to the user.
- **View System** – An extensible set of views used to create application user interfaces.

## **V. Applications:**

At the top layer, Android runs various applications developed by users and third-party developers. These applications include pre-installed system apps like Contacts, Browser, Camera, etc., as well as user-installed applications downloaded from Google Play Store or other sources.

# 4. ANATOMY OF AN ANDROID APPLICATION

It refers to the various components and files that make up the structure of an Android project. Each component plays a crucial role in defining the behavior, appearance, and functionality of the application. Let's break down the anatomy of an Android application based on the provided information:

## 1. **src (Source Folder):**

- Contains the .java source files for the project.
- In this example, there is one file named MainActivity.java, which is the source file for the main activity.
- You write the code for your application in these .java files.

## 2. **Android 3.0 library:**

- Contains the android.jar file, which includes all the class libraries needed for an Android application.
- These class libraries provide the core functionality of the Android framework.

## 3. **gen (Generated Folder):**

- Contains the R.java file, which is a compiler-generated file.
- R.java references all the resources found in your project, such as layouts, drawables, and strings.
- It is automatically generated by Eclipse when you modify your project, so you should not manually edit it.

## 4. **assets:**

- This folder contains assets used by your application, such as HTML files, text files, databases, etc.
- These assets are not compiled into the application but can be accessed at runtime.

## 5. **res (Resources Folder):**

- Contains all the resources used in your application, including:
  - **drawable-<resolution>:** Contains drawable resources for different screen resolutions.
  - **layout:** Contains XML layout files that define the user interface of activities.
  - **values:** Contains XML files for different value resources, such as strings, dimensions, colors, etc.
- Resources are referenced from the code and are used to provide the appearance and content of the application.

## 6. **main.xml:**

- Defines the user interface for the main activity (MainActivity).
- It contains XML elements that describe the layout of the activity.
- In the example provided, it includes a TextView element displaying the text "@string/hello".

## 7. **strings.xml:**

- Contains string resources used in the application.
- Recommended for storing all string constants used in the application.
- Allows easy localization by referencing strings using the @string identifier.

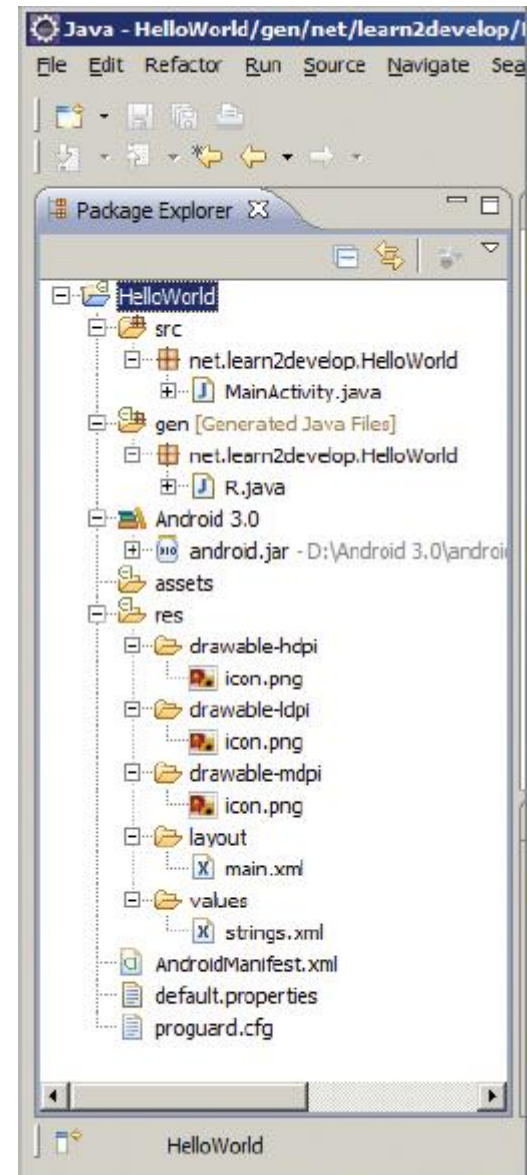
## 8. **AndroidManifest.xml:**

- The manifest file for the Android application.
- Contains detailed information about the application, such as:
  - Package name (e.g., net.learn2develop.HelloWorld).
  - Version code and version name for identifying the application version.
  - Application icon and name.
  - Permissions required by the application.
  - Declaration of activities, services, receivers, etc.
  - Intent filters to specify the entry point of the application.

## 9. **R.java:**

- Automatically generated by Eclipse based on the resources used in the project.
- Contains integer constants that reference resources (e.g., R.layout.main, R.string.app\_name).

The anatomy of an Android application shows how different components, resources, and files come together to create a functional and user-friendly application. Each file and folder serves a specific purpose and contributes to the overall functionality and appearance of the app. Understanding the anatomy is essential for developing and maintaining Android applications effectively.





# 5. Android - Application Components

- Application components are the essential building blocks of an Android application.
- These components are loosely coupled by the application manifest file *AndroidManifest.xml* that describes each component of the application and how they interact.

*There are following four main components that can be used within an Android application*

Sure, here's a more organized arrangement of the Android components and their descriptions:

## 1. Activities:

- Represents a single screen with a user interface.
- Handles user interactions and performs actions on the screen.
- Implemented as a subclass of the Activity class.
- Java code example:

```
java Copy code

public class MainActivity extends Activity {
    // Activity code here
}
```

## 2. Services:

- Runs in the background to perform long-running operations.
- Used for tasks such as playing music, fetching data from the network, etc.
- Implemented as a subclass of the Service class.
- Java code example:

```
java Copy code

public class MyService extends Service {
    // Service code here
}
```

## 3. Broadcast Receivers:

- Respond to broadcast messages from other applications or the system.
- Intercept and handle broadcast messages using the `onReceive()` method.
- Implemented as a subclass of the BroadcastReceiver class.
- Java code example:

```
java Copy code

public class MyReceiver extends BroadcastReceiver {
    public void onReceive(Context context, Intent intent) {
        // BroadcastReceiver code here
    }
}
```

## 4. Content Providers:

- Supplies data from one application to others on request.
- Handles data and database management issues.
- Implemented as a subclass of the ContentProvider class.
- Java code example:

```
java Copy code

public class MyContentProvider extends ContentProvider {
    // ContentProvider code here
}
```

Additionally, other important components used in Android app development are:

## 1. Fragments:

- Represents a portion of the user interface in an Activity.
- Can be combined to build a complete UI.
- Allows modular UI design and reusability.

## 2. Views:

- UI elements that are drawn on-screen (e.g., buttons, lists, forms).

## 3. Layouts:

- View hierarchies that control the screen format and appearance of views.

## 4. Intents:

- Messages used for wiring components together and initiating actions.

## 5. Resources:

- External elements like strings, constants, and drawable pictures.

## 6. Manifest:

- Configuration file for the application that describes each component and their interactions.

Understanding these components and how they interact with each other is essential for building robust and efficient Android applications.

# 6. VIEW GROUPS

- 1) ViewGroup is a special type of View that acts as a container for other views.
- 2) It is used to define the structure and layout of the user interface in an Android app.
- 3) ViewGroup is an abstract class, and various concrete implementations provide different ways to organize and arrange child views within them.
- 4) ViewGroup defines the basic methods and properties needed to manage child views, and its subclasses extend and implement these functionalities.
- 5) ViewGroup is an essential part of the Android UI hierarchy and plays a critical role in designing the user interface. By nesting different ViewGroups and Views, developers can create complex and dynamic layouts that adapt to various screen sizes and orientations.
- 6) Each View and ViewGroup has a set of common attributes as shown in the following table.

ATTRIBUTE	DESCRIPTION
layout_width	Specifies the width of the View or ViewGroup
layout_height	Specifies the height of the View or ViewGroup
layout_marginTop	Specifies extra space on the top side of the View or ViewGroup
layout_marginBottom	Specifies extra space on the bottom side of the View or ViewGroup
layout_marginLeft	Specifies extra space on the left side of the View or ViewGroup
layout_marginRight	Specifies extra space on the right side of the View or ViewGroup
layout_gravity	Specifies how child Views are positioned
layout_weight	Specifies how much of the extra space in the layout should be allocated to the View
layout_x	Specifies the x-coordinate of the View or ViewGroup
layout_y	Specifies the y-coordinate of the View or ViewGroup


# Some commonly used ViewGroup subclasses include:

As of my last update in September 2021, Android indeed supports the following ViewGroups:  

## 1. LinearLayout:

- Definition: LinearLayout is a flexible layout manager that arranges its children either horizontally or vertically, based on the orientation set.

### • Code Example:

```
xml  Copy code

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">


    <!-- Child views go here -->

</LinearLayout>
```

## 2. AbsoluteLayout (Deprecated):

- Definition: AbsoluteLayout is a layout manager that allows you to specify exact positions (x, y coordinates) for child views. However, this layout has been deprecated, and it's not recommended for use anymore. ConstraintLayout or RelativeLayout are better alternatives.

### • Code Example:

```
xml  Copy code

<AbsoluteLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Child views with absolute positions go here -->

</AbsoluteLayout>
```

## 3. TableLayout:

- Definition: TableLayout is a layout manager that arranges its children in rows and columns, similar to an HTML table.

### • Code Example:

```
xml  Copy code

<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TableRow>
        <!-- Row 1, Column 1 -->
        <TextView
            android:text="Row 1, Column 1" />

        <!-- Row 1, Column 2 -->
        <TextView
            android:text="Row 1, Column 2" />
    </TableRow>

    <TableRow>
        <!-- Row 2, Column 1 -->
        <TextView
            android:text="Row 2, Column 1" />

        <!-- Row 2, Column 2 -->
        <TextView
            android:text="Row 2, Column 2" />
    </TableRow>


    <!-- Add more rows as needed -->

</TableLayout>
```

## 4. RelativeLayout:

- Definition: RelativeLayout is a flexible layout manager that arranges its children relative to each other or to the parent view.

### • Code Example:

```
xml  Copy code

<RelativeLayout

    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <!-- Child views with relative positions go here -->

</RelativeLayout>
```

## 5. FrameLayout:

- Definition: FrameLayout is a simple layout manager that displays its children on top of each other. It's often used for single-child containers or overlapping views.

### • Code Example:

```
xml  Copy code

<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <!-- Child views go here -->

</FrameLayout>
```

## 6. ScrollView:

- Definition: ScrollView is a layout manager that allows its content to be scrollable if it exceeds the visible area of the screen.

### • Code Example:

```
xml  Copy code

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <!-- Scrollable content goes here -->

</ScrollView>
```

# 7. TextView

In Android, TextView is a fundamental UI widget used to display text content on the screen. It is one of the most commonly used views for showing static or dynamic text in an Android app's user interface. TextViews can display a single line or multiple lines of text, and they support various text styling options, such as different font sizes, colors, styles (bold, italic, etc.), and alignment.

Attribute	Java Method	Description
<b>android:width</b>	<b>setWidth()</b>	Sets the width of the control equal to the specified value. The value can be in any of the units of measurement discussed in Dimensions.
<b>android:height</b>	<b>setHeight()</b>	Sets the height of the control equal to the given value.
<b>android:text</b>	<b>setText()</b>	Sets the text to display in the given control.
<b>android:textColor</b>	<b>setTextColor()</b>	Sets the color of the text.
<b>android:textSize</b>	<b>setTextSize()</b>	Sets the size of the text.
<b>android:ellipsize</b>	<b>setEllipsize()</b>	If set, causes text that is longer than the width of the container to be ellipsized instead of breaking it in the middle.
<b>android:singleLine</b>	<b>setSingleLine()</b>	Decides whether the control is supposed to be a single-line input or multiple-line input.
<b>android:background</b>	<b>setBackgroundResource()</b>	Sets the background color of a control or makes an image appear in the background of the control.
<b>android:textStyle</b>	<b>setTypeface()</b>	Applies a style to the text.
<b>android:typeface</b>	<b>setTypeface()</b>	Sets the text to appear in a given font style.

```
<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 with width and height -->
  <TextView
    android:layout_width="200dp"
    android:layout_height="100dp"
    android:text="Hello World!"
    android:textColor="#0F0"
    android:textSize="25sp"
    android:ellipsize="end"
    android:singleLine="true"
    android:background="#0000FF"
    android:textStyle="bold|italic"
    android:typeface="serif" />

</LinearLayout>
```

```
1  import android.app.Activity;
2  import android.graphics.Color;
3  import android.graphics.Typeface;
4  import android.os.Bundle;
5  import android.text.TextUtils;
6  import android.widget.TextView;
7
8  public class MainActivity extends Activity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.activity_main);
14
15         TextView textView = findViewById(R.id.textView);
16
17         textView.setWidth(500);
18         textView.setHeight(200);
19         textView.setText("Hello World!");
20         textView.setTextColor(Color.GREEN);
21         textView.setTextSize(25);
22         textView.setEllipsize(TextUtils.TruncateAt.END);
23         textView.setSingleLine(true);
24         textView.setBackgroundResource(R.drawable.background_shape);
25
26         Typeface serifTypeface = Typeface.create(Typeface.SERIF, Typeface.NORMAL);
27         textView.setTypeface(serifTypeface);
28     }
29 }
```



## 8. LOGIN FORM

```
<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:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.relativeLayout2.MainActivity" >
```

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="84dp"
    android:layout_marginTop="14dp"
    android:text="Sign In" />
```

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_below="@+id/textView1"
    android:layout_marginLeft="15dp"
    android:layout_marginTop="48dp"
    android:text="User Id" />
```

```
<EditText
    android:id="@+id/editText1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_toRightOf="@+id/textView1"
    android:ems="10" >
```

```
<requestFocus />
```

```
</EditText>
```

```
<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView2"
    android:layout_below="@+id/editText1"
    android:layout_marginTop="39dp"
    android:text="Password" />
```

```
<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView3"
    android:layout_alignBottom="@+id/textView3"
    android:layout_toRightOf="@+id/textView1"
    android:ems="10" />
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="Sign In" />
```

```
<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/button1"
    android:layout_marginTop="52dp"
    android:layout_toRightOf="@+id/textView3"
    android:text="TextView" />
```

```
</RelativeLayout>
```

```
package com.example.relativeLayout2;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.*;
```

```
public class MainActivity extends Activity implements OnClickListener
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button b=(Button)findViewById(R.id.button1);
        b.setOnClickListener(this);
    }

    public void onClick(View v)
    {
        EditText userid=(EditText)findViewById(R.id.editText1);
        EditText password=(EditText)findViewById(R.id.editText2);
        TextView resp=(TextView)findViewById(R.id.textView4);

        String usr=userid.getText().toString();
        String pwd=password.getText().toString();

        if(usr.trim().length()==0 ||pwd.trim().length()==0 )
        {
            String str="left blank";
            resp.setText(str);
        }
        else
        {
            if(usr.equals("cmr")&&pwd.equals("cmr"))
                resp.setText("welcome");
            else
                resp.setText("invalid login");
        }
    }
}
```

## 9. TOGGLING APPLICATION

```
2  <FrameLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent" >
6
7      <ImageView
8          android:id="@+id/imageView1"
9          android:layout_width="wrap_content"
10         android:layout_height="wrap_content"
11         android:src="@drawable/ic_launcher" />
12
13     <ImageView
14         android:id="@+id/imageView2"
15         android:layout_width="wrap_content"
16         android:layout_height="456dp"
17         android:src="@drawable/logo"
18         android:visibility="gone" />
19
20 </FrameLayout>
```

```
package com.example.frameLayoutnew;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageView;

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

        final ImageView i1 = findViewById(R.id.imageView1);
        final ImageView i2 = findViewById(R.id.imageView2);

        i1.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View view){
                i2.setVisibility(View.VISIBLE);
                view.setVisibility(View.GONE);
            }
        });

        i2.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View view){
                i1.setVisibility(View.VISIBLE);
                view.setVisibility(View.GONE);
            }
        });
    }
}
```