Mobile application development

- **What is mobile application development?**
- Mobile application development is the process of making software for smartphones, tablets and digital assistants, most commonly for the Android and iOS operating systems.
- The software can be **preinstalled on the device**, **downloaded from a mobile app store** or **accessed** through a **mobile web browser**.
- The **programming and markup languages** used for this kind of software development include **Java, Kotlin, Swift, C# and HTML5**.

Three main types of Mobile Applications

- **Progressive Web Apps-web-based applications** that work on different platforms and **devices-combine the best features of both web and mobile app**s -**HTML, CSS, and JavaScript**-**Twitter Lite, Flipkart, Pinterest, Uber etc.**
- **Native Applications**-developed for a specific platform, such as Android or iOS, using the platform's native programming languages and development tools, such as Swift or Objective-C for iOS and Java or Kotlin for Android
- **Hybrid Applications-**Hybrid apps combine elements of both web and native apps- **run on multiple platforms-**single codebase that can be deployed across different operating systems, such as iOS and Android.

- **What is Android OS?**
- Android OS is a Linux-based mobile operating system that primarily runs on smartphones and tablets.
- Android is an operating system developed by Google. It is a modified version of Linux kernel and other open-source software.
- Today android used in many electronic devices like touchscreen mobile, tablets, Android TV, Android Auto for cars, and Android watches, each with a specialized user interface.
- Android is also used on the game, digital cameras other electronics.
- The Android platform includes an operating system based upon the **Linux kernel, a GUI, a web browser and end-user applications** that can be downloaded.

- Android is based on **open source software**, but **most Android devices come preinstalled** with a suite of **proprietary software**, such as **Google Maps, YouTube, Google Chrome and Gmail.**

- **Android OS versions**
- Google makes **incremental changes to the OS** with each release.
- This often includes **security patches and performance improvements.**

Evolution of Android

- **Android 1.0.** Released Sept. 23, 2008. Included a suite of Google apps, including Gmail, Maps, Calendar and YouTube.
- **Android 1.5 (Cupcake).** Released April 27, 2009. Introduced an onscreen virtual keyboard and the framework for third-party app widgets.
- **Android 1.6 (Donut).** Released Sept. 15, 2009. Introduced the ability for the OS to run on different screen sizes and resolutions; added support for CDMA networks.
- **Android 2.0 (Eclair).** Released Oct. 26, 2009. Added turn-by-turn voice navigation, real-time traffic information, pinch-to-zoom capability.
- **Android 2.2 (Froyo).** Released May 20, 2010. Added dock at the bottom of the home screen and voice actions, which allows users to tap an icon and speak a command. Also introduced support for Flash to the web browser.

- **Android 2.3 (Gingerbread).** Released Dec. 6, 2010. Introduced black and green into the UI.
- **Android 3.0 to 3.2 (Honeycomb).** Released Feb. 22, 2011. This release was exclusive to tablets and introduced a blue, space-themed holographic design.
- **Android 4.0 (Ice Cream Sandwich).** Released Oct. 18, 2011. Introduced a unified UI to both tablets and smartphones; emphasized swiping as a navigational method.
- **Android 4.1 to 4.3 (Jelly Bean).** Released July 9, 2012, Nov. 13, 2012, and July 24, 2013, respectively. Introduced Google Now, a day planner service. Added interactive notifications and improved voice search system.
- **Android 4.4 (KitKat).** Released Oct. 31, 2013. Introduced lighter colors into the UI, along with a transparent status bar and white icons.

- **Android 5.0 (Lollipop).** Released Nov. 12, 2014. Incorporated a card-based appearance in the design with elements such as notifications and Recent Apps list. Introduced hands-free voice control with the spoken "OK, Google" command.
- **Android 6.0 (Marshmallow).** Released Oct. 5, 2015. This release marked Google's adoption of an annual release schedule. Introduced more granular app permissions and support for USB-C and fingerprint readers.
- **Android 7.0 and 7.1 (Nougat).** Released Aug. 22, 2016 and Oct. 4, 2016, respectively. Introduced a native split-screen mode and the ability to bundle notifications by app.

- **Android 8.0 and 8.1 (Oreo).** Released Aug. 21, 2017 and Dec. 5, 2017, respectively. These versions introduced a native picture-in-picture (PIP) mode and the ability to snooze notifications. Oreo was the first version to incorporate Project Treble, an effort by OEMs to provide more standardized software updates.
- **Android 9.0 (Pie).** Released Aug. 6, 2018. This version replaced Back, Home and Overview buttons for a multifunctional Home button and a smaller Back button. Introduced productivity features, including suggested replies for messages and brightness management capabilities.

- **Android 10 (Android Q).** Released Sept. 3, 2019. Abandoned the Back button in favor of a swipe-based approach to navigation. Introduced a dark theme and Focus Mode, which enables users to limit distractions from certain apps.
- **Android 11 (Red Velvet Cake).** Released Sept. 8, 2020. Added built-in screen recording. Created a single location to view and respond to conversations across multiple messaging apps. This version also updated the chat bubbles so users can pin conversations to the top of apps and screens.

- **Android 12 (Snow Cone).** Released Oct. 4, 2021. Added customization options for the user interface. The conversation widget let users store preferred contacts on their home screens. Added more privacy options, including sharing when apps access information such as camera, photos or microphone.
- **Android 12L.** Released March 7, 2022.The L stands for larger screens. This update aimed to improve the user interface and optimize for the larger screen of a tablet, foldable or Chromebook. This update added a dual-panel notification center for tablets and foldables.

- **Android 13 (Tiramisu).** Released Aug. 15, 2022. Included more customizable options including color, theme, language and music. Security updates included control over information apps can access, notification permission required for all apps and clearing of personal information on clipboard. This update enables multitasking by sharing of messages, chats, links and photos across multiple Android devices – including phones, tablets and Chromebooks.
- **Android 14 (Upside Down Cake).** Released Oct. 4, 2023. Included more customization options for the lock screen and wallpapers. OS efficiency was improved to decrease the strain on a phone's battery. For accessibility, Google added larger scalable fonts for vision-impaired users and camera flashes to give hearing-impaired users another visual cue when they get a notification. Security updates include notifications of changes in data-sharing policies for third-party applications, enhanced PIN security features and better support for passkey authentication across more applications.

Building Blocks of Android

- The building blocks of Android are the fundamental components that make up the structure of an Android application.
- These building blocks work together to create a functional and interactive user experience.
- There are four building blocks to an Android application:
- **Activity**
- **Intent Receiver**
- **Service**
- **Content Provider**
- Not every application needs to have all four, but your application will be written with some combination of these.
- Once you have decided what components you need for your application, you should list them in a file called AndroidManifest.xml. This is an XML file where you declare the components of your application and what their capabilities and requirements are.

- **Activity** :
- Activities are the most common of the four Android building blocks.
- An activity is usually a **single screen in your application.**
- Each activity is implemented as a **single class that extends the Activity base class.**
- When a new screen opens, the previous screen is paused and put onto a history stack.
- The user can navigate backward through previously opened screens in the history.
- Activities are defined in Java or Kotlin classes and are crucial for handling user interactions.

- **Intent and Intent Filters :**
- Android uses a special class called Intent to move from screen to screen. Intent describe what an application wants done.
- **Intents are messaging objects used to communicate between different components within an application or between different applications**.
- They facilitate **the transfer of data and trigger actions**.
- The **two most important parts of the intent** data structure are the **action and the data to act upon.**
- Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI).
- For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI.
- **new Intent(android.content.Intent.VIEW_ACTION, ContentURI.create("http://anddev.org"));**
- Java files use Intents to facilitate communication between different components of the application, such as starting new activities or services.

Types of Intents

- **Explicit Intents-Launching an activity within the same application**
- An Explicit Intent is used to launch a specific component within the same application, typically by providing the target component's class name.
- **Intent explicitIntent = new Intent(CurrentActivity.this, TargetActivity.class);**
- **startActivity(explicitIntent);**

Implicit

- An Implicit Intent is used when the **system is asked to find a suitable component to handle the specified action, such as opening a web page, sending an email, or capturing a photo).**
- **Action or Action and Data Specified:**
  - Instead of specifying a particular component, an Implicit **Intent includes an action (e.g., "ACTION_VIEW," "ACTION_SEND")** and, optionally, **data (e.g., a URI**).
  - The system resolves the Intent and launches the appropriate component capable of handling the specified action.
- **Example: Opening a web page:**
  - **Intent implicitIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("https://www.example.com")); startActivity(implicitIntent);**

Key Differences:

- **Explicit Intent:**
  - Target component is explicitly specified.
  - Used for intra-application communication.
  - The class name or component identifier is included.
  - Commonly used for navigating between activities within the same application.
- **Implicit Intent:**
  - Target component is not explicitly specified.
  - Used for inter-application communication.
  - Specifies an action (and optionally data).
  - The system resolves the Intent and launches the appropriate external component.

- **Services:**
- Services are components that **run in the background, independent of any user interface**, to perform long-running operations or handle tasks such as **playing music or downloading data.**
- the media player activity could start a service using **Context.startService() to run in the background to keep the music going.**
- Services don't have a UI but can communicate with other application components.

- **Content Provider :**
- **Applications can store their data in files, a SQLite database, preferences** or any other mechanism that makes sense.
- Content Providers **manage the application's data** and enable **sharing data between applications**.
- A content provider, however, is useful if you want your application's data to be shared with other applications.
- A **content provider is a class that implements a standard set of methods** to let other applications store and retrieve the type of data that is handled by that content provider.

- **Broadcast Receivers:**
- Broadcast Receivers respond to system-wide broadcast announcements or messages.
- They allow the **application to receive and respond to events like the completion of a download or a change in network connectivity.**
- Broadcast receivers can initiate other components or services.

- **Fragments:**
- Fragments represent a portion of a user interface and can be combined to create a complete UI.
- Fragments are often used within activities to build more modular and flexible user interfaces, especially for tablet layouts or large-screen devices.

- **Layouts and Views:**
- Layouts define the **structure of the user interface**
- **Views are the UI elements such as buttons, text fields, and images.**
- Android provides a variety of layouts and views that can be combined to create the desired user interface.

- **Manifest File:**
- The **AndroidManifest.xml file** is a fundamental configuration file in an Android application.
- It contains essential metadata and declarations required by the **Android Operating System to understand the structure and behavior of the app.**
- Below are the key components typically found in the **AndroidManifest.xml file:**
- It includes details like the **app's package name, the components it consists of, required permissions, and more.**

- Explanation of key elements:
- **Package Declaration (package attribute):**
  - Specifies the unique identifier for the application. It's typically in reverse domain format.
- **Permissions (<uses-permission> element):**
  - Declares the permissions required by the application. For example, the INTERNET permission is declared to allow internet access.
- **Application Information (<application> element):**
  - Contains general information about the application, such as its icon, label, theme, and whether it supports right-to-left (RTL) languages.
- **Launcher Activity (<activity> element with <intent-filter>):**
  - Declares the main activity of the application, which is the entry point. The <intent-filter> specifies that it should respond to the MAIN action and be launched as the main activity.
- **Other Components (<activity>, <service>, <receiver>, <provider>):**
  - Additional components of the application, such as activities, services, broadcast receivers, and content providers, are declared within the <application> element.
- **Minimum and Target SDK Versions (<uses-sdk> element):**
  - Specifies the minimum and target SDK versions required by the application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <!-- Permissions -->
    <uses-permission android:name="android.permission.INTERNET" />

    <!-- Application Information -->
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <!-- Launcher Activity -->
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action
android:name="android.intent.action.MAIN" />
 <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Other Activities, Services, Broadcast Receivers, and Content Providers -->

    </application>

    <!-- Minimum and Target SDK Versions -->
    <uses-sdk
        android:minSdkVersion="16"
        android:targetSdkVersion="30" />

</manifest>
```

XML Files:

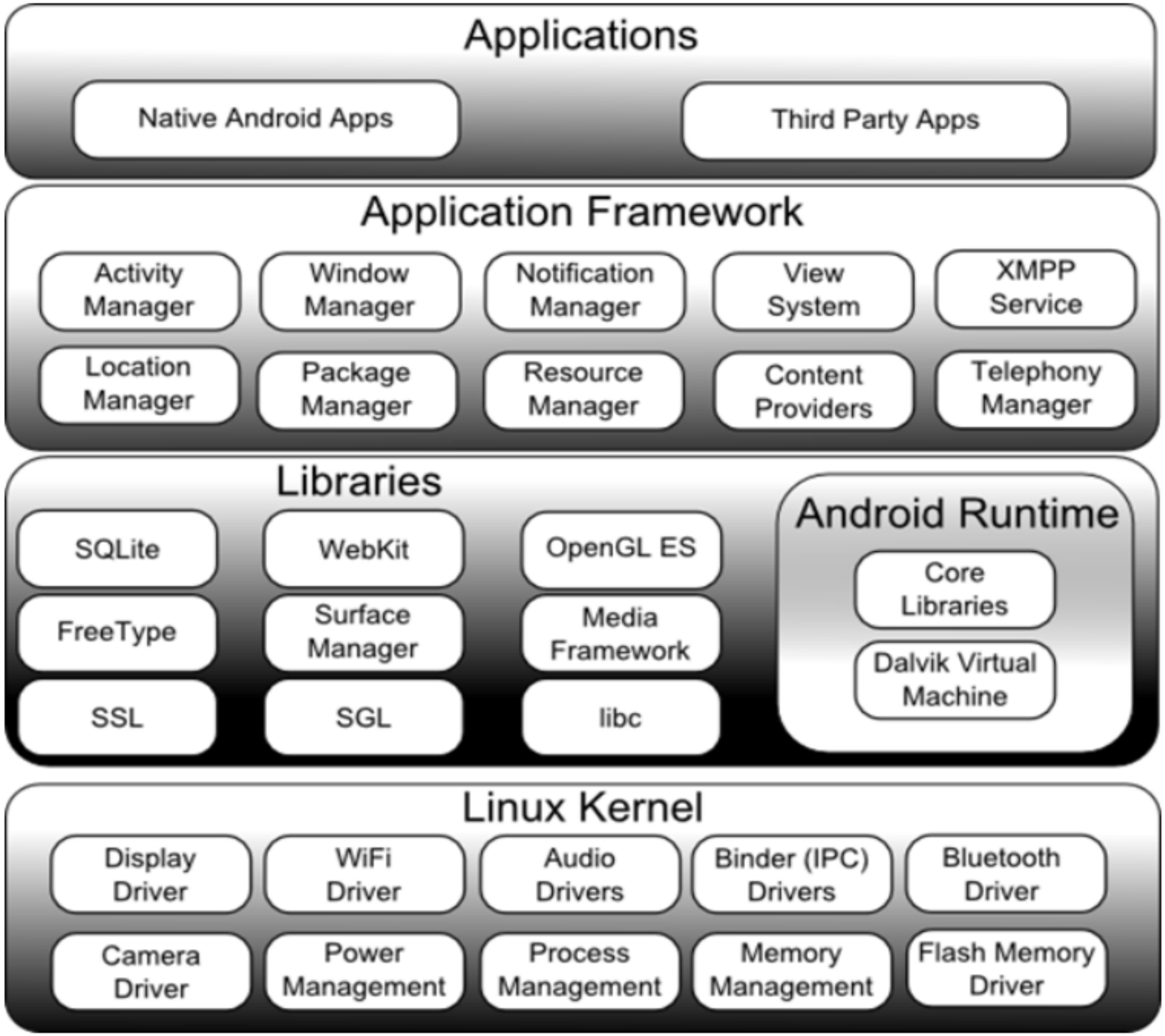
1. **User Interface (UI) Definition:**
   - XML files, located in the `**res/layout**` directory, are commonly used to define the structure and appearance of the user interface. They specify the arrangement of UI elements such as buttons, text fields, images, and layouts.
2. **Layout Configuration:**
   - XML files are used to configure layouts and define the arrangement and appearance of UI components. Different layout types (e.g., LinearLayout, RelativeLayout) and attributes (e.g., width, height, padding) are specified in XML to control the visual presentation.
3. **Resource Declarations:**
   - XML is used to declare various resources like strings, colors, dimensions, drawables, and styles. These resources can be referenced in both XML and Java files, promoting consistency and easy maintenance.
4. **Menus and Drawables:**
   - XML files are used for defining menus (res/menu) and drawable resources (res/drawable). Menus specify the options in an app's action bar or overflow menu, while drawables define images, shapes, and other graphic resources.
5. **Animation and Transition:**
   - Animation and transition effects can be defined using XML in the `**res/anim**` and `**res/transition**` directories. These XML files describe the behavior of animations and transitions.

Java (or Kotlin) Files:

1. **Application Logic:**
   - Java (or Kotlin) files, often found in the `src` directory, contain the code that defines the behavior and logic of the Android application. This includes handling user interactions, processing data, and implementing the application's functionality.
2. **Activity and Fragment Code:**
   - Java files define the behavior of activities and fragments. They handle lifecycle events, respond to user input, and interact with the Android framework. Each activity typically has a corresponding Java (or Kotlin) file that extends the `Activity` class.
3. **Event Handling:**
   - Java files contain event-handling code, responding to user interactions such as button clicks, item selections, or touch gestures. Event listeners are often implemented in Java to handle these interactions.
4. **Data Processing and Business Logic:**
   - Java files implement the core logic of the application, including data processing, business rules, and any backend communication. They may interact with databases, APIs, or other external services.
5. **Asynchronous Tasks:**
   - Background tasks and asynchronous operations are often implemented in Java (or Kotlin) files. These tasks run separately from the main thread to avoid blocking the user interface during resource-intensive operations.
6. **Adapter and RecyclerView Handling:**
   - Adapters for RecyclerViews, ListViews, and other UI components are implemented in Java files. These adapters connect d ↓ ources to UI elements, facilitating dynamic content display.

Anatomy of Android

- The anatomy of Android refers to **the various components(4), layers(5), and concepts(User Interface (UI), Intent System, Manifest File) that make up the Android operating system and its application development framework.**
- Android architecture is also known as the **Android software stack.**
- **It can be categorized into five parts:**
  - **Linux kernel**
  - **Native Libraries (Middleware)**
  - **Android Runtime**
  - **Application Framework**
  - **Applications**


Applications
Native Android Apps
Third Party Apps
Application Framework
Activity Manager
Window Manager
Notification Manager
View System
XMPP Service
Location Manager
Package Manager
Resource Manager
Content Providers
Telephony Manager
Libraries
SQLite
WebKit
OpenGL ES
FreeType
Surface Manager
Media Framework
SSL
SGL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
WiFi Driver
Audio Drivers
Binder (IPC) Drivers
Bluetooth Driver
Camera Driver
Power Management
Process Management
Memory Management
Flash Memory Driver

- ***Linux kernel:***
- **Linux kernel** exists at the **root of android architecture** and is thus also called as the **heart of the android architecture**.
- The **device drivers, power management, memory management, device management, and resource access** comes under the responsibility of the Linux Kernel.

Native Libraries:

Native libraries such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc), etc are on top of the Linux kernel.

Sl.No.	Library	Responsibility
1	WebKit library	Browser support
2	SQLite	Database
3	FreeType	Font support
4	Media	Playing and recording audio and video formats

- ***Android Runtime:***
- The **core libraries and DVM (Dalvik Virtual Machine)** are there in android runtime.
- They are **responsible for running android applications.**
- Originally being like JVM, DVM is optimized for **mobile devices to consume less memory and to facilitate a fast performance**.

- ***Android Framework:***
- The Android framework is on the **top of Native libraries and android runtime.**
- Android APIs like **UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers** are a part of the Android framework.
- For the development of the android application, **the Android framework facilitates a lot of classes and interfaces.**

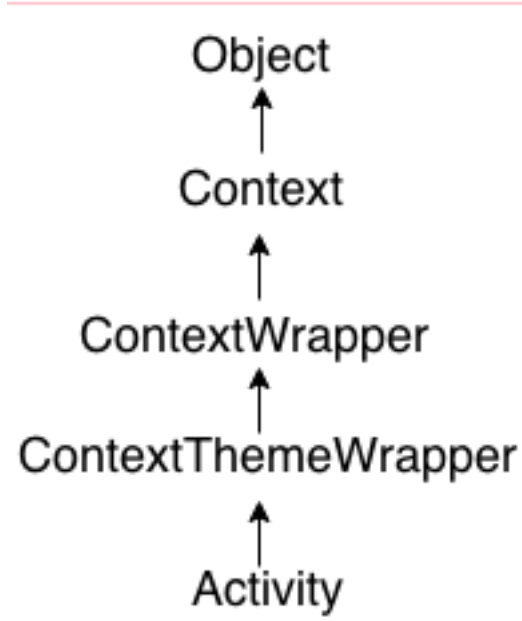
Android Activity Lifecycle

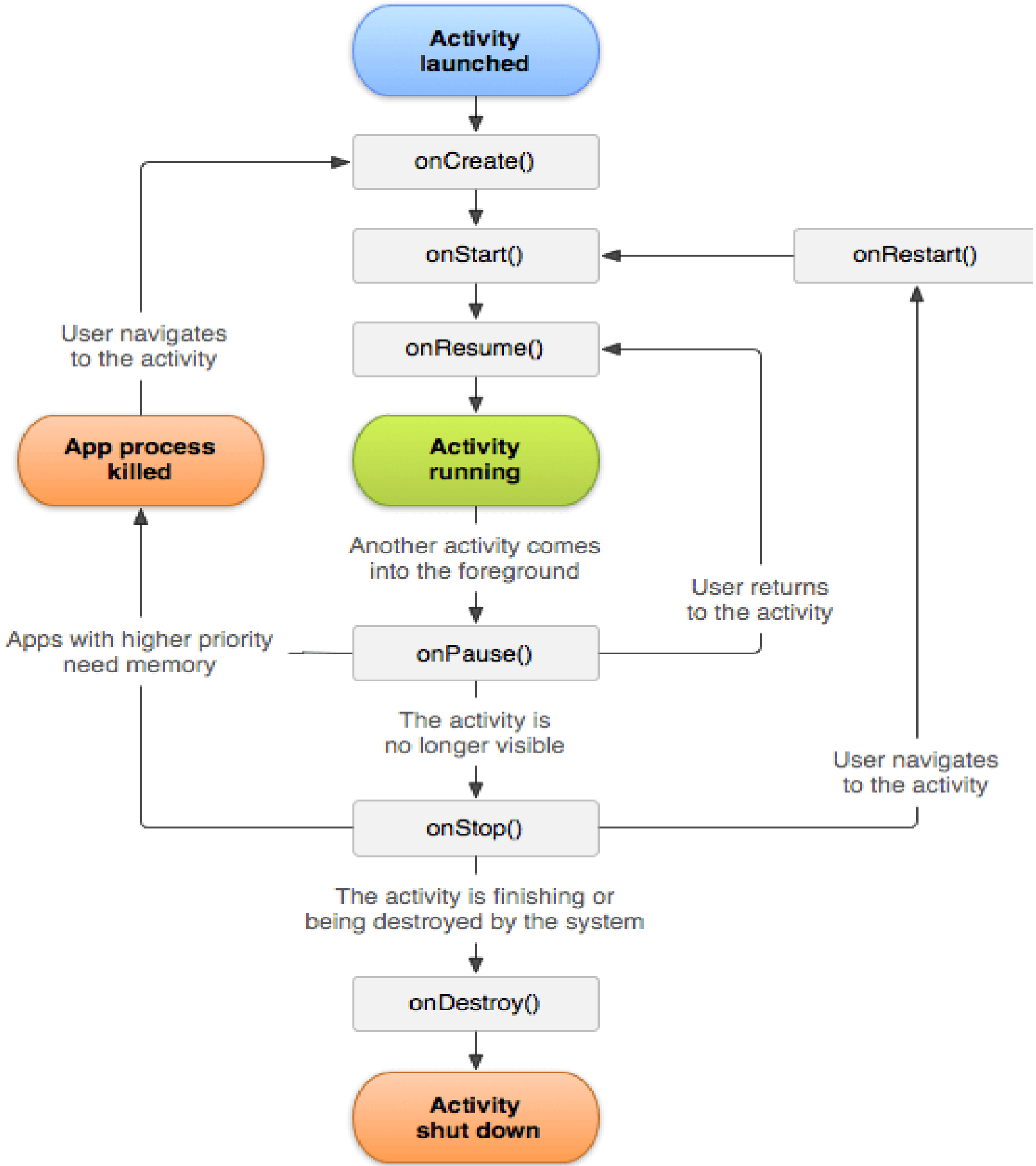
- **Android Activity Lifecycle** is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.
- **An activity is the single screen in android. It is like window or frame of Java.**
- By the **help of activity, you can place all your UI components or widgets in a single screen.**
- The **7 lifecycle method of Activity describes how activity will behave at different states.**

Android Activity Lifecycle methods

Let's see the 7 lifecycle methods of android activity.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.




Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.javatpoint.com.activitylifecycle.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

- **Android Activity Lifecycle Example**
- It provides the details about the invocation of life cycle methods of activity. In this example, we are displaying the content on the logcat.

- **Override a method from the superclass (Activity in this case)**

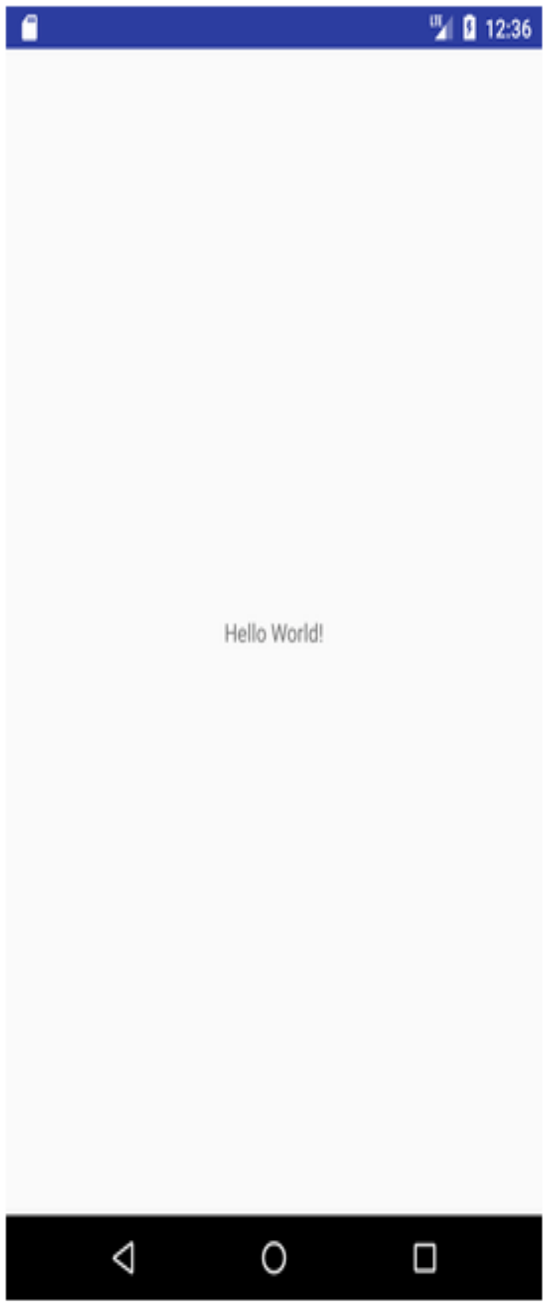
File: MainActivity.java

```
package example.javatpoint.com.activitylifecycle;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
    }
}
```

1. **`@Override` Annotation:**
   - This annotation indicates that the method is intended to override a method from the superclass (`Activity` in this case). It is a good practice to include this annotation for clarity and to ensure that you are indeed overriding a method.
2. **`protected void onCreate(Bundle savedInstanceState)`:**
   - This is the method signature. It declares that the method is `protected`, returns `void`, and takes a single parameter of type `Bundle` named `savedInstanceState`.
3. **`super.onCreate(savedInstanceState)`:**
   - This calls the `onCreate` method of the superclass (`Activity`). It is crucial to call the superclass method to ensure that the default setup tasks are performed before any additional initialization in the derived class.
4. **`// setContentView and other initialization tasks here`:**
   - This is where you perform specific initialization tasks for your activity. Common tasks include setting the content view (using `setContentView` to define the UI layout), initializing UI elements, binding data, and other setup steps.
5. **`Bundle savedInstanceState`:**
   - The `savedInstanceState` parameter is used to retrieve the previous state of the activity if it was destroyed and recreated. For example, if the device is rotated, the activity is recreated, and any data stored in the `savedInstanceState` bundle can be used to restore the previous state.

Output:

You will not see any output on the emulator or device. You need to open logcat.



- **Bundle parameter, which is used to restore the activity's previous state if it was previously destroyed and recreated (e.g., due to a configuration change).**

Notifications and Toast messages

- Notifications and Toast messages are distinct features in Android, each serving a different purpose.
- **Notifications are typically used to alert users about events or updates even when the app is not currently active**, while
- **Toast messages are used for displaying short-lived messages within the app's UI**.
- If you want to create a notification in Android, you would typically use the NotificationManager along with a Notification object.

Q. Name the widget used to display a short message. List the members of that object and give an example.

- **Case Study: Improving User Feedback with Android Toasts**
- **Background:** You are a developer working on a mobile application for a social networking platform. **The app allows users to share posts, comment on content, and interact with other users.** Users have provided feedback indicating that they want more informative and non-intrusive feedback for certain actions, especially when posting comments or liking posts.

- **Implementing Toasts for Comment Posting:**When a user successfully posts a comment, a short-duration Toast is displayed at the bottom of the screen, confirming the action.
- If there is an issue with posting the comment (e.g., network error), an error Toast is displayed, alerting the user to try again.

- **// Inside the code for posting a comment**
- if (commentPostedSuccessfully) {
- **Toast.makeText(context, "Comment posted successfully", Toast.LENGTH_SHORT).show();**
- **}**
- else {
- **Toast.makeText(context, "Failed to post comment. Please try again.", Toast.LENGTH_SHORT).show();**
- }

Q. Name the widget used to display a short message. List the members of that object and give an example.

- The **widget used to display a short-duration message in Android is called Toast.**
- The **Toast class is part of the android.widget package** and is commonly used to show brief messages to the user.
- **Members of the Toast Object:**
- **makeText() Method:**
  - Static method to **create a new Toast object.**
  - Takes parameters such as the **application context, the text message to be displayed, and the duration of the toast.**
- **show() Method:**
  - **Displays the Toast message on the screen.**

Example: Toast widget in Android

- **// Inside an Activity or a Context**
- Context context = getApplicationContext();
- CharSequence message = "This is a short message!";
- int duration = Toast.LENGTH_SHORT;

- **Toast toast = Toast.makeText(context, message, duration);**
- toast.show();

- In this example:
- **context: The application context or activity context where the Toast is being displayed.**
- **message: The text message to be shown in the Toast**.
- **duration: The duration for which the Toast should be displayed. It can be either Toast.LENGTH_SHORT or Toast.LENGTH_LONG.**
- **makeText(): Creates a new Toast object with the specified parameters.**
- **show(): Displays the Toast on the screen.**

Android Widgets and Attributes

- In Android, **a widget is a user interface component that users can interact with on the screen.**
- Widgets are an essential part of the Android development framework and are used to build the graphical user interface (GUI) of an Android application.
- Widgets can include a variety of elements such as **buttons, text fields, images, checkboxes, radio buttons, toasts, and more.**

1. **TextView:** Displays text on the screen.
2. **EditText:** Allows users to input text.
3. **Button:** Represents a clickable button.
4. **ImageView:** Displays images.
5. **CheckBox:** A box that can be checked or unchecked.
6. **RadioButton:** A button that can be either checked or unchecked, within a group of radio buttons.
7. **ToggleButton:** A button that can be in either an "on" or "off" state.
8. **SeekBar:** Allows users to select a value from a range by sliding a thumb.
9. **ProgressBar:** Displays a visual indication of the progress of an operation.
10. **Spinner:** A drop-down menu for selecting an item from a list.
11. **Switch:** A two-state toggle switch.
12. **RatingBar:** Allows users to rate something by selecting a number of stars.
13. **DatePicker:** A widget for selecting a date.
14. **TimePicker:** A widget for selecting a time.
15. **ListView:** Displays a scrollable list of items.
16. **RecyclerView:** A more flexible and advanced version of ListView for displaying large sets of data.
17. **GridLayout:** A layout manager that arranges its children in a grid.
18. **LinearLayout:** A layout manager that arranges its children in a single column or row.
19. **FrameLayout:** A simple layout manager that stacks its children on top of each other.
20. **RelativeLayout:** A layout manager that arranges its children relative to each other or to the parent.

```
<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">

    <Button
        android:id="@+id/myButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />

    <TextView
        android:id="@+id/myTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, World!" />
</LinearLayout>
```

Widgets and Attributes

- Widgets have attributes that describe **how they should behave.**
- Attributes of widgets in Android are properties that **define the appearance, behavior, and characteristics of user interface elements** within an Android application.
- These **attributes are specified in XML layout files** and are used to customize the widgets according to the application's design.
- **Those attributes can be modified by a "Properties" view in the graphical layout editor of the IDE.**

Common Widget Attributes:

- **Common Attributes for Views:**
  - **android:id:** Unique identifier for the view.
  - **android:layout_width** and **android:layout_height:** Specify the width and height of the view.
  - **android:layout_margin and android:layout_padding**: Set the margin and padding for the view.

TextView Attributes:

android:text: Sets the text content of the TextView.

android:textSize: Specifies the text size.

android:textColor: Sets the text color.

```
<TextView
    android:id="@+id/myTextView"
    android:text="Hello, World!"
    android:textSize="16sp"
    android:textColor="#000000" />
```

- **EditText Attributes:**
- **android:hint:** Sets the hint text (displayed when the EditText is empty).
- **android:inputType:** Specifies the input type (e.g., text, number, password).

```
<EditText
    android:id="@+id/myEditText"
    android:hint="Enter text here"
    android:inputType="text" />
```

- **Button Attributes:**
- **android:text:** Sets the text displayed on the button.
- **android:onClick:** Specifies the method to be called when the button is clicked.

```
<Button
    android:id="@+id/myButton"
    android:text="Click Me"
    android:onClick="onButtonClick" />
```

```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">

    <Button
        android:id="@+id/myButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me" />

    <TextView
        android:id="@+id/myTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, World!" />
</LinearLayout>
```

Simple Calculator using Widgets for Addition

- To create a simple calculator in Android for addition, you can use widgets such as **EditText for input, Button for numeric and operator buttons, and a TextView to display the result**. Here's a basic example:
- **1. Layout (activity_main.xml):**
- **Create the layout file** (res/layout/activity_main.xml) with EditText, Buttons, and a TextView:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/andro
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextNumber1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:hint="Enter number 1"
        android:inputType="numberDecimal" />

    <EditText
        android:id="@+id/editTextNumber2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/editTextNumber1"
        android:layout_marginBottom="16dp"
        android:hint="Enter number 2"
        android:inputType="numberDecimal" />
```

```
    <Button
        android:id="@+id/btnAdd"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/editTextNumber2"
        android:text="Add" />

    <TextView
        android:id="@+id/textViewResult"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/btnAdd"
        android:layout_marginTop="16dp"
        android:text="Result: "
        android:textSize="20sp" />

</RelativeLayout>
```

Activity (MainActivity.java):

Handle the button click and perform addition in the activity (MainActivity.java):

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private EditText editTextNumber1, editTextNumber2;
    private Button btnAdd;
    private TextView textViewResult;

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

```
        editTextNumber1 = findViewById(R.id.editTextNumber1);
        editTextNumber2 = findViewById(R.id.editTextNumber2);
        btnAdd = findViewById(R.id.btnAdd);
        textViewResult = findViewById(R.id.textViewResult);

        btnAdd.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // Perform addition when the button is clicked
                calculateAndDisplayResult();
            }
        });
    }
```

```
    private void calculateAndDisplayResult() {
        // Get input numbers from EditText fields
        String num1Str = editTextNumber1.getText().toString();
        String num2Str = editTextNumber2.getText().toString();

        if (!num1Str.isEmpty() && !num2Str.isEmpty()) {
            // Parse input numbers
            double num1 = Double.parseDouble(num1Str);
            double num2 = Double.parseDouble(num2Str);

            // Perform addition
            double result = num1 + num2;

            // Display the result
            textViewResult.setText("Result: " + result);
        } else {
            // Handle empty input fields
            textViewResult.setText("Result: Please enter valid numbers"
        }
    }
}
```