Unit - 1: Android OS Architecture & Features

Introduction

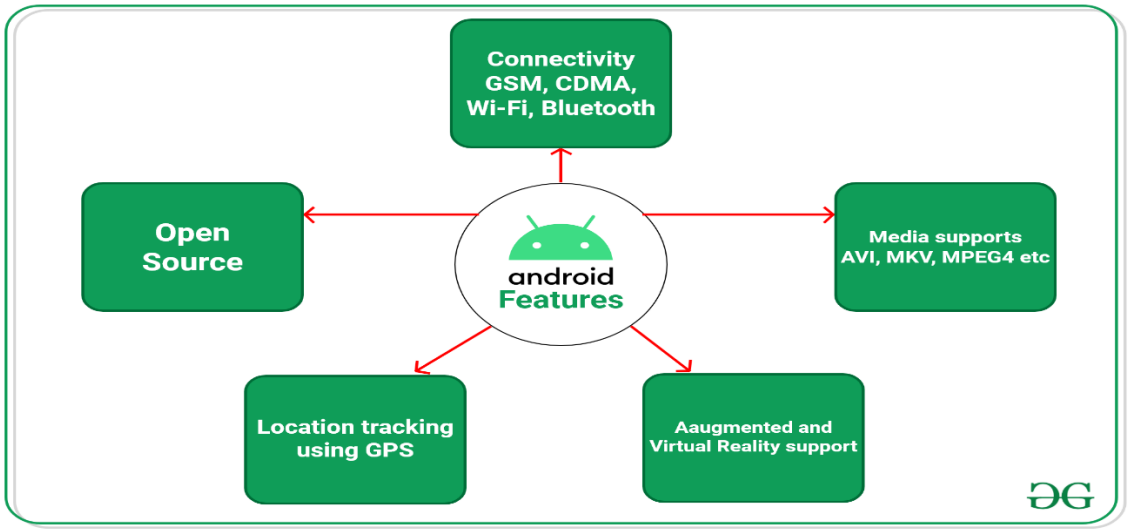
- Android is an open-source, Linux-based operating system designed mainly for smartphones, tablets, smart TVs, smartwatches, and other smart devices.

In simple words:

- Android is the software that controls a mobile device and allows users to run applications (apps) like WhatsApp, YouTube, Instagram, Google Maps, etc.
- Just like Windows is an operating system for computers, Android is an operating system for mobile devices.

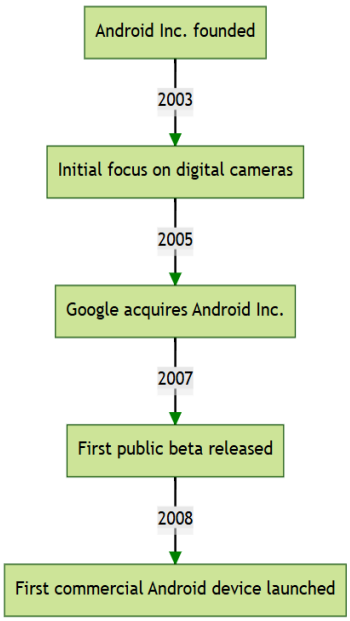
Features of Android

Android is a powerful open-source operating system that open-source provides immense features and some of these are listed below.



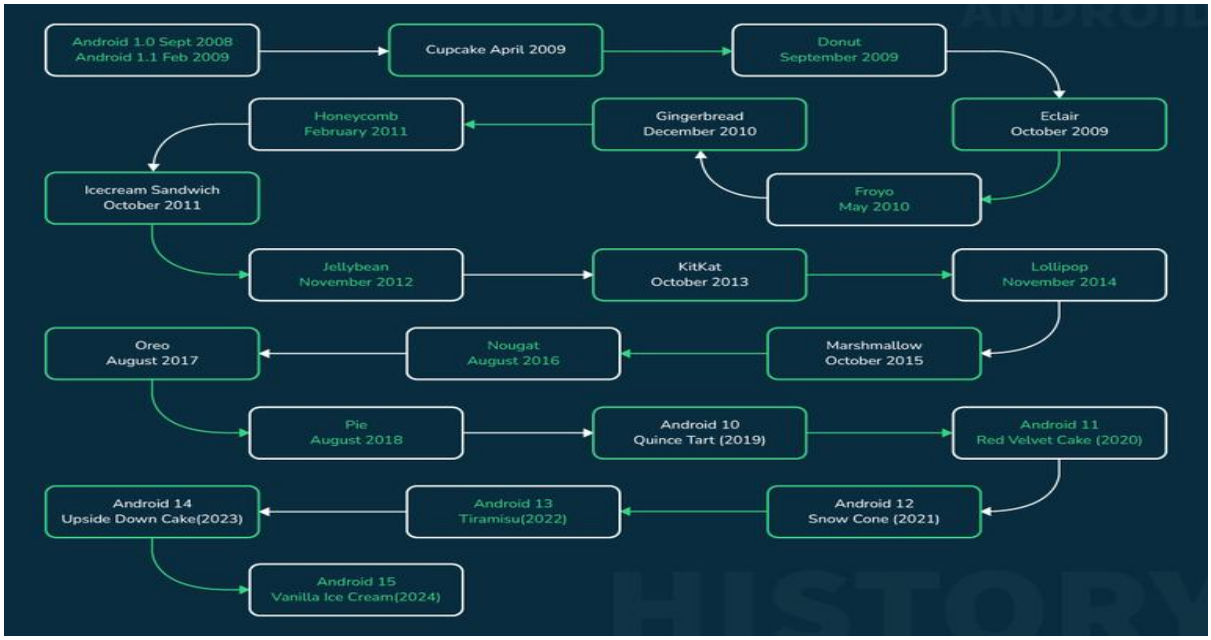
- Android is an Open Source Project so we can customize the OS based on our requirements.
- Android supports different types of connectivity for GSM, CDMA, Wi-Fi, Bluetooth, etc. for telephonic conversation or data transfer.
- Using wifi technology we can pair with other devices while playing games or using other applications.
- It contains multiple APIs to support location-tracking services such as GPS.
- We can manage all data storage-related activities by using the file manager.
- It contains a wide range of media supports like AVI, MKV, FLV, MPEG4, etc. to play or record a variety of audio/video.
- It also supports different image formats like JPEG, PNG, GIF, BMP, MP3, etc.
- It supports multimedia hardware control to perform playback or recording using a camera and microphone.
- Android has an integrated open-source WebKit layout-based web browser to support User Interfaces like HTML5, and CSS3.
- Android supports multi-tasking means we can run multiple applications at a time and can switch between them.
- It provides support for virtual reality or 2D/3D Graphics.

History of Android



Android Versions

Google first publicly announced Android in November 2007 but was released on 23 SEPTEMBER 2008 to be exact. The first device to bring Android into the market was the HTC Dream with the version Android 1.0. Since then, Google released a lot of android versions such as Apple Pie, Banana Bread, Cupcake, Donut, Éclair, Froyo, Gingerbread, Jellybeans, Kitkat, Lollipop, marshmallow, Nougat, Oreo, etc. with extra functionalities and new features.



Programming Languages used in Developing Android Applications

1. **Java**

2. **Kotlin**

Developing the Android Application using Kotlin is preferred by Google, as Kotlin is made an official language for Android Development, which is developed and maintained by JetBrains. Previously before Java is considered the official language for Android Development. Kotlin is made official for Android Development in Google I/O 2017.

Advantages of Android Development

- The Android is an open-source Operating system and hence possesses a vast community for support.
- The design of the Android Application has guidelines from Google, which becomes easier for developers to produce more intuitive user applications.
- Fragmentation gives more power to Android Applications. This means the application can run two activities on a single screen.
- Releasing the Android application in the Google play store is easier when it is compared to other platforms.

Disadvantages of Android Development

- Fragmentation provides a very intuitive approach to user experience but it has some drawbacks, where the development team needs time to adjust to the various screen sizes of mobile smartphones that are now available in the market and invoke the particular features in the application.
- The Android devices might vary broadly. So the testing of the application becomes more difficult.
- As the development and testing consume more time, the cost of the application may increase, depending on the application's complexity and features.

Android Architecture

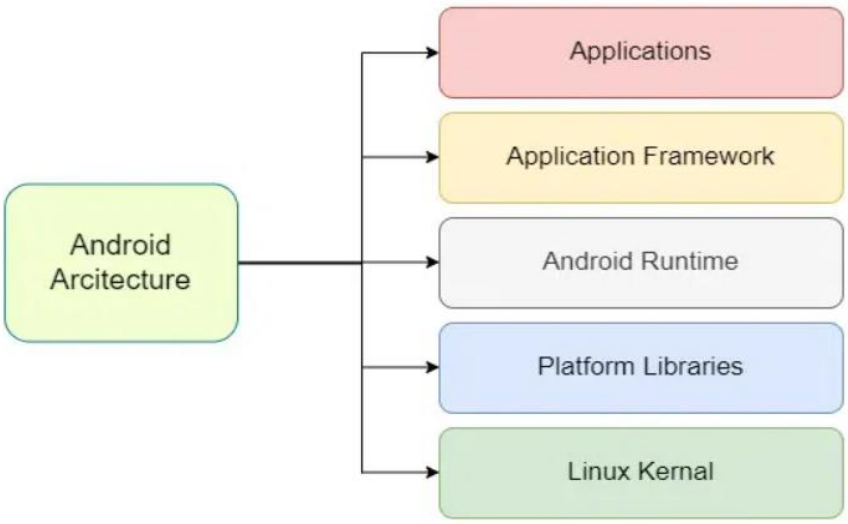
Android architecture contains a different number of components to support any Android device's needs. Android software contains an open-source Linux Kernel having a collection of a number of C/C++ libraries which are exposed through application framework services. Among all the components Linux Kernel provides the main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide a platform for running an Android application.

Components of Android Architecture

The main components of Android architecture are the following:-

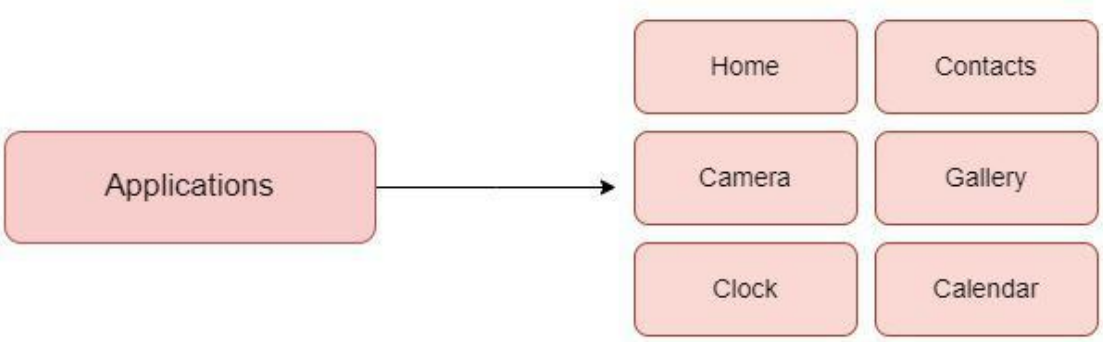
- Applications
- Application Framework
- Android Runtime
- Platform Libraries
- Linux Kernel

Pictorial representation of Android architecture with several main components and their sub-components



1. Applications

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only. It runs within the Android run time with the help of the classes and services provided by the application framework.

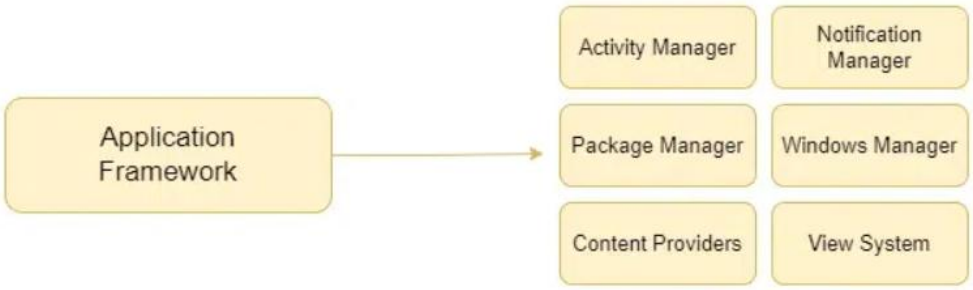


2. Application framework

The Application Framework is a core part of Android that gives developers the tools and services they need to build apps. It provides access to device features like hardware, screen display, and system resources. It includes several important services that make it easier to build powerful and

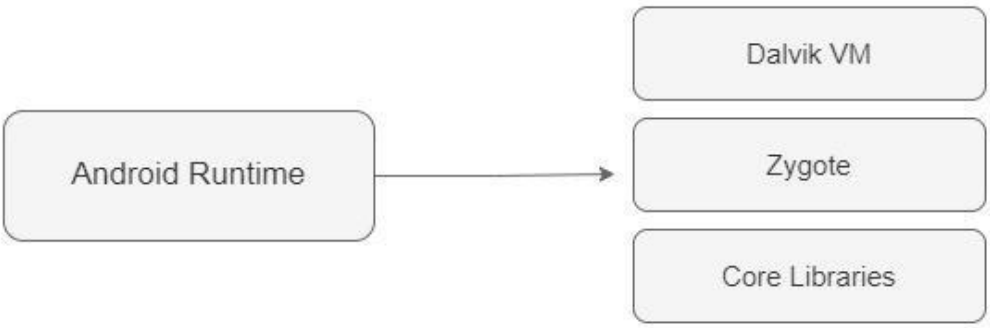
consistent Android apps without having to create everything from scratch. The services are as follows:

1. Activity Manager - Manages the app's activities and their life cycle (like opening, pausing, or closing screens).
2. Notification Manager - Allows apps to show alerts or updates to the user.
3. Package Manager - Keeps track of all the apps installed on the device.
4. Window Manager - Handles the placement and appearance of windows on the screen.
5. Content Providers - Help apps share data with other apps (like contacts or photos).
6. View System - Controls how things (like buttons or text) appear on the screen.



3. Application runtime

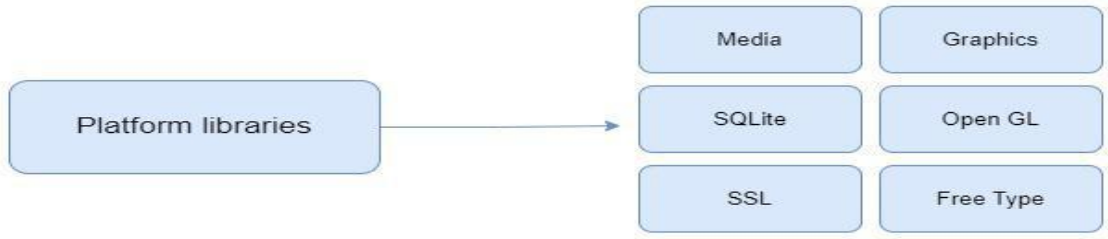
Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries. Like Java Virtual Machine (JVM), **Dalvik Virtual Machine (DVM)** is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.



4. Platform libraries

The Platform Libraries includes various C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL etc. to provide a support for android development.

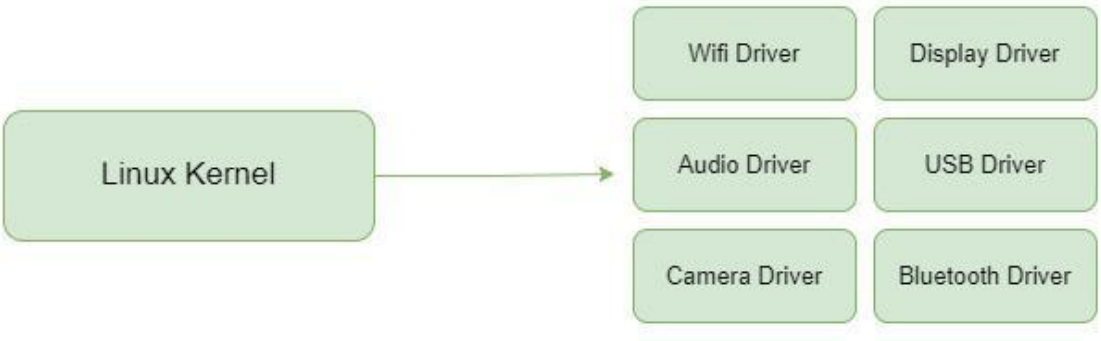
- **Media** library provides support to play and record an audio and video formats.
- **Surface manager** responsible for managing access to the display subsystem.
- **SGL** and **OpenGL** both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- **SQLite** provides database support and **FreeType** provides font support.
- **Web-Kit** This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- **SSL (Secure Sockets Layer)** is security technology to establish an encrypted link between a web server and a web browser.



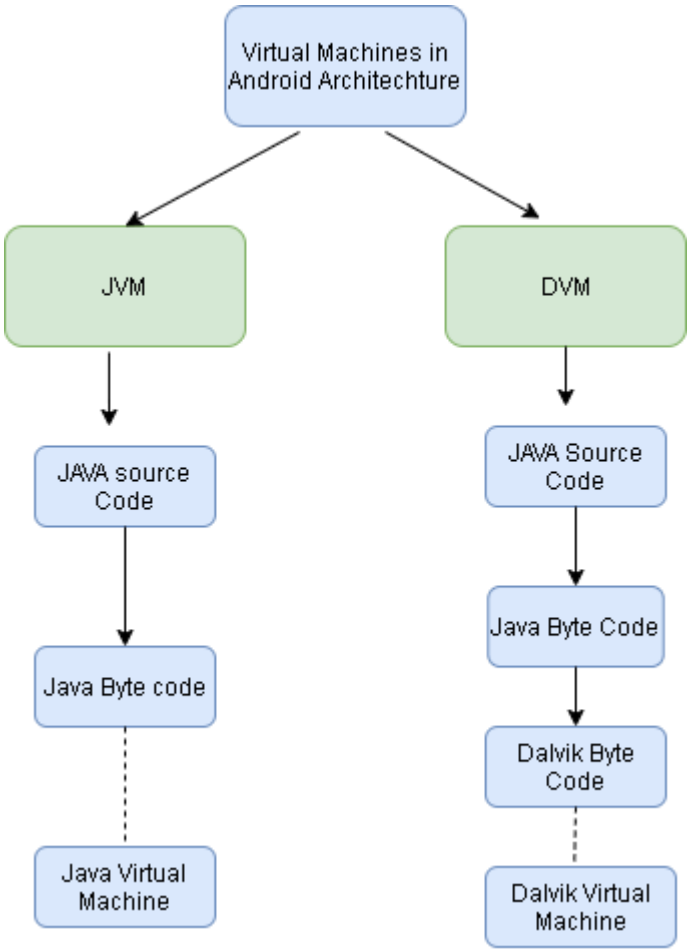
5. Linux Kernel

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime. The Linux Kernel will provide an abstraction layer between the device hardware and the other components of android architecture. It is responsible for management of memory, power, devices etc. The features of Linux kernel are:

- **Security:** The Linux kernel handles the security between the application and the system.
- **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.
- **Process Management:** It manages the process well, allocates resources to processes whenever they need them.
- **Network Stack:** It effectively handles the network communication.
- **Driver Model:** It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.



DVM (Dalvik Virtual Machine)



Introduction to API (Application Programming Interface)

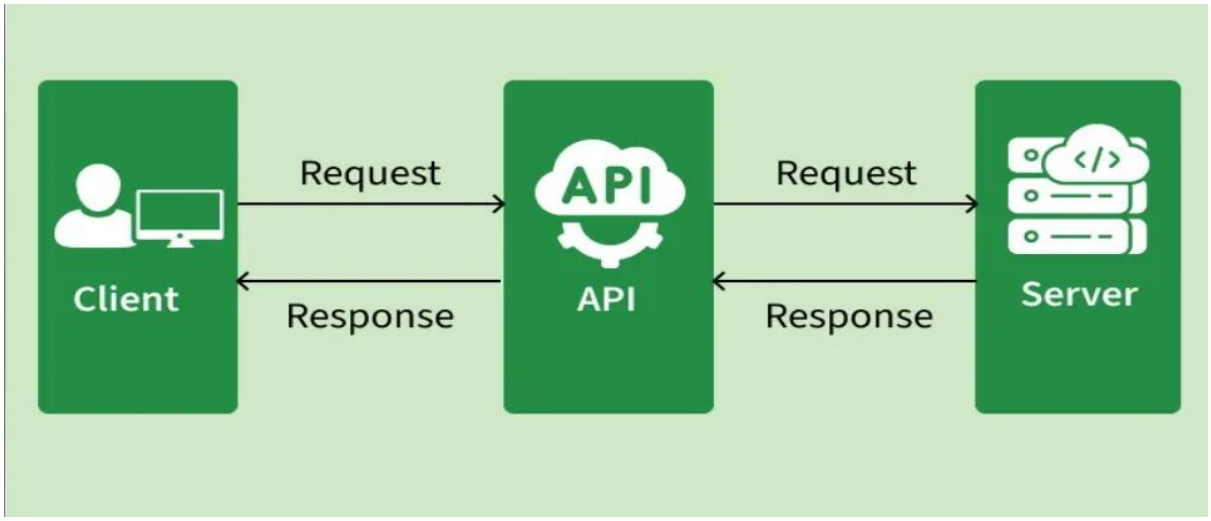
An API (Application Programming Interface) is a set of rules that allows different software applications to communicate and exchange data with each other. It acts as a bridge between systems, enabling one application to request services or information from another in a structured way.

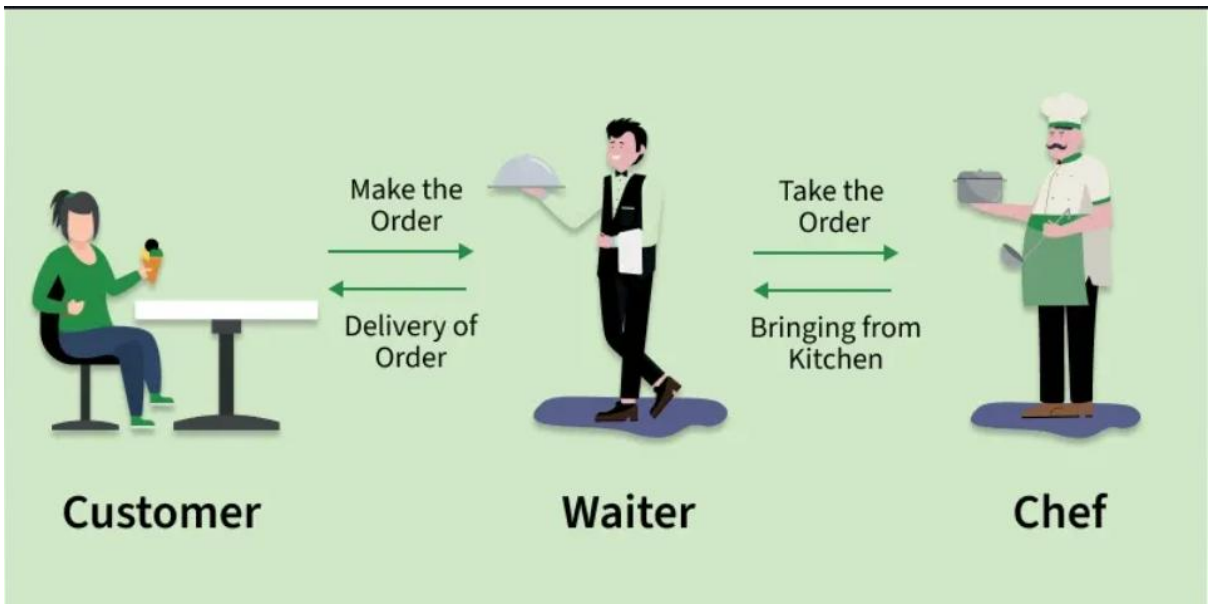
- APIs help different applications connect and work together smoothly.

- They allow data sharing and communication without exposing internal system logic.
- APIs are widely used in web applications, mobile apps, payment systems, and cloud services.

Example: *A mobile banking app requests account details through an API. The API verifies the request, fetches data from the banking server, and returns the account information to the app.*

This is exactly how APIs work in software the API takes a request, sends it to the server, retrieves the data, and returns the response.





How an API Works

An API acts as a communication layer between a client and server, handling requests and returning responses to enable data exchange between applications.

- **Client -> API (Request)**: The client application sends a request to the API containing required data, parameters, headers, and authentication details.
- **API -> Server (Request Forwarding)**:The API receives the request, validates the input, applies business logic, and forwards the request to the appropriate server or database.

- **Server -> API (Response)**: The server processes the request, retrieves or updates data, and sends the result back to the API.
- **API -> Client (Response)**: The API formats the response (commonly in JSON or XML format) and sends it back to the client application.

Components of an API

An API consists of several key components that define how applications communicate and exchange data.

- **API Endpoint:** An API endpoint is a specific URL through which clients access API resources or services.
- **Request:** A request is sent by the client to the API to perform an operation or retrieve data from the server.
- **HTTP Methods:** HTTP methods define the type of action performed by an API request, such as retrieving, creating, updating, or deleting data.
- **Headers:** Headers provide additional information about the request or response, such as authentication details, content type, and authorization tokens.
- **Request Body (Payload):** The request body contains the data sent from the client to the server, usually in JSON or XML format.
- **Response:** A response is the data returned by the server after processing the API request.
- **HTTP Status Codes:** HTTP status codes indicate the result of an API request, such as success, client error, or server error.
- **Authentication and Authorization:** Authentication verifies the identity of a user or application, while authorization controls access to specific resources.
- **API Documentation:** API documentation provides details about endpoints, request formats, parameters, and response structures to help developers use the API effectively.

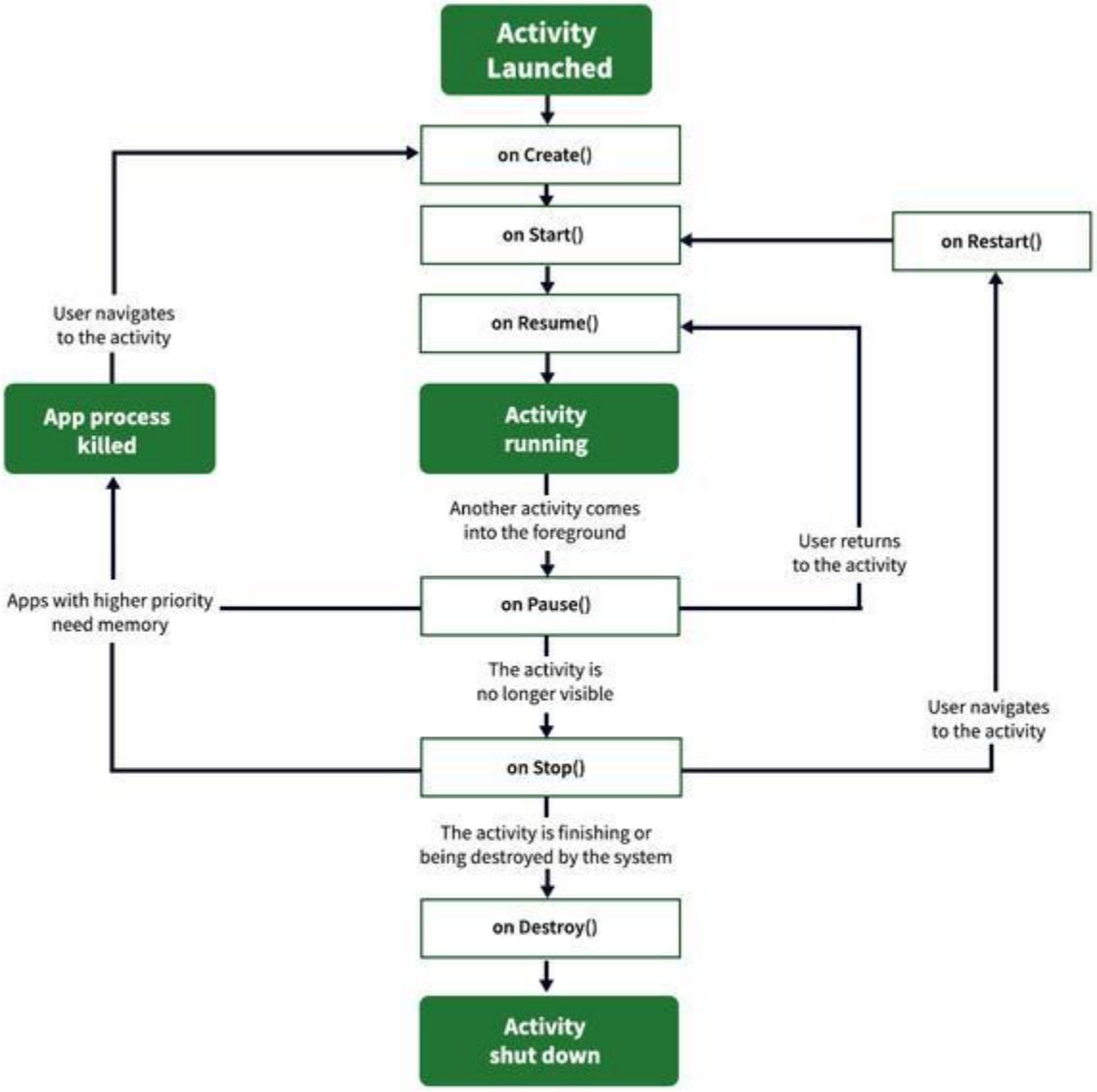
Activity Lifecycle in Android

In Android, an **activity** is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities. Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it. There are four stages of an activity.

1. If an activity is in the foreground of the screen i.e at the top of the stack, then it is said to be active or running. This is usually the activity that the user is currently interacting with.
2. If an activity has lost focus and a non-full-sized or transparent activity has focused on top of your activity. In such a case either another activity has a higher position in multi-window mode or the activity itself is not focusable in the current window mode. Such activity is completely alive.
3. If an activity is completely hidden by another activity, it is stopped or hidden. It still retains all the information, and as its window is hidden thus it will often be killed by the system when memory is needed elsewhere.

4. The system can destroy the activity from memory by either asking it to finish or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

For each stage, android provides us with a set of 7 methods that have their own significance for each stage in the life cycle. The image shows a path of migration whenever an app switches from one state to another.



Activity Lifecycle in Android

Detailed introduction on each of the method is stated as follows:

1. onCreate()

It is called when the activity is first created. This is where all the static work is done like creating views, binding data to lists, etc. This method also provides a Bundle containing its previous frozen state, if there was one.

2. onStart()

It is invoked when the activity is visible to the user. It is followed by onResume() if the activity is invoked from the background. It is also invoked after onCreate() when the activity is first started.

3. onRestart()

It is invoked after the activity has been stopped and prior to its starting stage and thus is always followed by onStart() when any activity is revived from background to on-screen.

4. onResume()

It is invoked when the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, with a user interacting with it. Always followed by onPause() when the activity goes into the background or is closed by the user.

5. onPause()

It is invoked when an activity is going into the background but has not yet been killed. It is a counterpart to onResume(). When an activity is launched in front of another activity, this callback will be invoked on the top activity (currently on screen). The activity, under the active activity, will not be created until the active activity's onPause() returns, so it is recommended that heavy processing should not be done in this part.

6. onStop()

It is invoked when the activity is not visible to the user. It is followed by **onRestart()** when the activity is revoked from the background, followed by onDestroy() when the activity is closed or finished, and nothing when the activity remains on the background only. Note that this method may never be called, in low memory situations where the system does not have enough memory to keep the activity's process running after its onPause() method is called.

7. onDestroy()

The final call received before the activity is destroyed. This can happen either because the activity is finishing (when finish() is invoked) or because the system is temporarily destroying this instance of the activity to save space. To distinguish between these scenarios, check it with **isFinishing()** method.

Basic Components of Android

Android applications are built using four main components:

1. Activity

An **Activity** represents a single screen of an application.

Example: Login screen, Home screen, Settings screen.

---

2. Service

A **Service** performs tasks in the background without interacting with the user.

Example: Music player running while using another app.

---

3. Broadcast Receiver

A **Broadcast Receiver** receives system-wide messages or events.

Example: Battery low notification, incoming SMS, Wi-Fi connected.

---

4. Content Provider

A **Content Provider** allows applications to share data securely.

Example: Accessing contacts, images, or calendar data.

Unit – 2: XML Introduction

What is XML?

- **XML** stands for **eXtensible Markup Language**.
- It is a **markup language** used to **store, organize, and transfer data** between different applications and systems.
- Unlike HTML, XML does **not display data**. It only describes and stores data.

Definition

XML (eXtensible Markup Language) is a text-based language that allows users to create their own tags to store and transport data in a structured format.

Why was XML developed?

Before XML, different applications stored data in different formats.

- Example:
    - Software A stores:
        - Name: Rahul
        - Age: 20
    - Software B stores:
        - StudentName = Rahul
        - StudentAge = 20

Since formats are different, exchanging data becomes difficult.

XML provides a **common format** that every application can understand.

Real-Life Example

Imagine sending student details from one college software to another.

Instead of writing:

Rahul

20

BCA

XML stores data like this:

<Student>

<Name>Rahul</Name>

<Age>20</Age>

<Course>BCA</Course>

</Student>

Now every software knows:

- Rahul is Name
- 20 is Age
- BCA is Course

Uses of XML

XML is widely used for:

- Data storage
- Data transfer
- Web services
- Configuration files

- Android development
- Banking systems
- E-commerce
- APIs
- Office documents

Advantages of XML

- Easy to read
- Human-readable
- Machine-readable
- Platform independent
- Language independent
- Extensible
- Stores structured data
- Easy to transfer over the Internet

Difference Between HTML and XML

HTML	**XML**
Displays data	Stores data
Fixed tags	User-defined tags
Focus on presentation	Focus on information
Tags are predefined	Tags are created by users

2.1 Characteristics of XML

1. Extensible

Users can create their own tags.

Example

<Student>

or

<Book>

or

<Employee>

No fixed tags like HTML.

---

2. Self-descriptive

The tags describe the data.

Example

<Price>500</Price>

Anyone can understand that 500 is the price.

---

3. Human Readable

People can easily read XML.

Example

<Name>Priya</Name>

---

4. Machine Readable

Computers can also process XML easily.

Software can automatically understand the tags.

---

5. Platform Independent

XML works on:

- Windows
- Linux
- Mac
- Android

Same XML file can be used everywhere.

---

6. Language Independent

XML works with

- Java
- Python
- PHP
- C#
- JavaScript
- Android

---

7. Hierarchical Structure

XML stores data like a tree.

Example

```
<Student>
   <Name>Rahul</Name>
   <Age>20</Age>
</Student>
```

Tree Structure

2.2 XML Declaration, Tags and Elements

XML Declaration

Every XML document generally starts with an XML declaration.

Syntax

<?xml version="1.0" encoding="UTF-8"?>

Part	**Meaning**
version="1.0"	XML version
encoding="UTF-8"	Character encoding

Example

<?xml version="1.0" encoding="UTF-8"?>

<Student>

<Name>Rahul</Name>

</Student>

XML Tags

Tags define the data.

Example

<Name>

and

</Name>

There are two types of tags.

Opening Tag

<Name>

Starts the element.

Closing Tag

</Name>

XML Elements

An XML element consists of:

- Opening tag
- Content
- Closing tag

Nested Elements

Elements can contain other elements.

Example

<Student>

<Name>Rahul</Name>

<Age>20</Age>

<City>Surat</City>

</Student>

Here,

Student contains three child elements.

2.3 Root Element

Every XML document must have **only one root element**.

The root element is the **top-most (parent) element** that contains all other elements.

Think of it like the trunk of a tree.

Example

<Student>

<Name>Rahul</Name>

<Age>20</Age>

<Course>BCA</Course>

</Student>

Here,

Student

is the root element.

Another Example

<Library>

<Book>

<Title>Java</Title>

</Book>

<Book>

<Title>Python</Title>

</Book>

</Library>

Root element is

Library

Rules of Root Element

There must be only **one** root element.

Two root elements are not allowed.

2.4 XML Documents

An XML document is simply a file that contains XML data.

Its extension is

.xml

Example

student.xml

Structure of an XML Document

```
<?xml version="1.0" encoding="UTF-8"?>

<Student>

   <Name>Rahul</Name>

   <Age>20</Age>

   <Course>BCA</Course>

</Student>
```

Parts of XML Document

1. XML Declaration

```
<?xml version="1.0" encoding="UTF-8"?>
```

2. Root Element

```
<Student>
```

3. Child Elements

```
<Name>

<Age>

<Course>
```

4. Data

Rahul

20

BCA

Rules of XML Documents

1. Only One Root Element

2. Every Opening Tag Must Have a Closing Tag

3. Tags are Case Sensitive

4. Tags Must Be Properly Nested

5. Attribute Values Must Be in Quotes

6. File Extension Should Be .xml

Complete XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<Student>
   <RollNo>101</RollNo>
   <Name>Rahul Sharma</Name>
   <Age>20</Age>
   <Course>BCA</Course>
   <Semester>5</Semester>
   <City>Surat</City>
</Student>
```

Unit 3: Android Widgets and Their Implementation

Introduction to Android Widgets

Android widgets are user-interface (UI) components used to create interactive Android applications. A widget allows the user to **see information, enter data, select options, press buttons, and interact with an application**.
Examples of common Android widgets are:

- Text Field
- Button
- CheckBox
- RadioButton
- ToggleButton
- Toast

Android applications also use **layouts** to decide how these widgets are arranged on the screen.

3.1 First Android Project

A **First Android Project** is the basic Android application created to understand the structure and working of Android development.
When creating an Android project, we generally provide:

1. Application Name

This is the name given to the Android application. It is the name that users may see on their device.
For example, an application could be named:

- Student Information
- Calculator
- College App
- Library Management

2. Package Name

A package name uniquely identifies an Android application. It helps Android and application stores distinguish one application from another.

3. Project Location

This specifies where the project files are stored on the computer.

4. Minimum SDK

The **Minimum SDK** specifies the oldest Android version on which the application is intended to run.
Choosing a lower minimum SDK can allow the application to run on more devices, while newer Android features may require a higher SDK version.

5. Android Activity

An **Activity** represents a screen or a particular interaction area of an Android application.
For example, an application may have:

- Login Activity
- Home Activity
- Profile Activity
- Settings Activity

The first screen displayed when an application starts is generally associated with its starting activity.

6. User Interface

The user interface is the visible part of the application with which the user interacts.
It can contain:

- Text
- Buttons
- Input fields
- Checkboxes
- Radio buttons
- Images
- Lists
- Other widgets

7. Running the Project

The application can be tested using:

- An Android emulator
- A physical Android device

The project is built and then installed on the selected device or emulator.

Purpose of the First Android Project

The first project helps students understand:

- Android project structure
- Activities
- User interfaces
- Layouts
- Widgets
- Application execution
- Basic user interaction

3.2 Title Bar

A **Title Bar** is the area at the top of an application screen that can display the title of the current screen or application.
For example, in a student application, the title bar might display:
Student Information
The title bar helps the user understand **which screen or section of the application is currently open**.

Functions of a Title Bar

1. **Displays the screen title**
   It identifies the current activity or screen.
2. **Provides navigation**
   In many Android applications, the top bar can contain navigation controls such as a back button.
3. **Provides actions**
   Some applications place important actions in the top area, such as search, save, or settings.
4. **Improves user experience**
   It gives the application a consistent structure and makes navigation easier.

Example

Imagine a college application with different screens:

- Home
- Student Details
- Attendance
- Examination
- Settings

When the Student Details screen is open, the title bar can display **Student Details**.

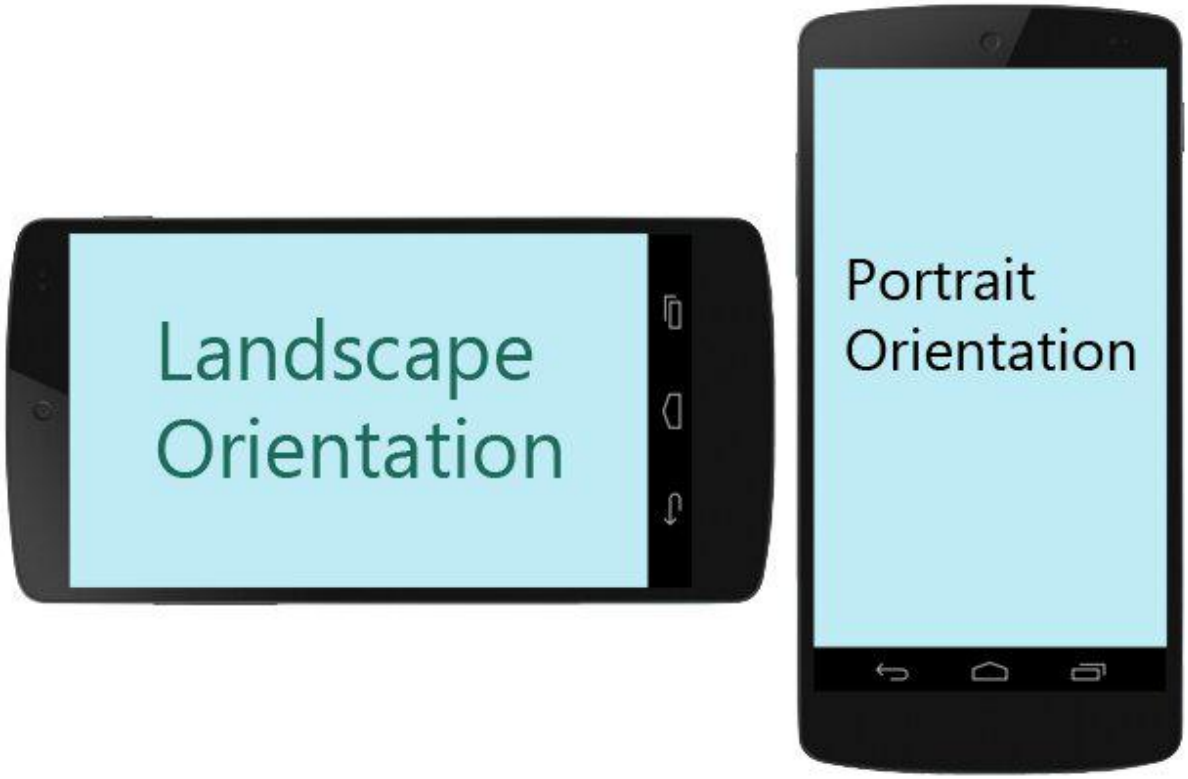
Modern Android

In modern Android applications, the traditional title bar concept is commonly implemented using an **app bar** or **toolbar**. It can contain the screen title, navigation controls, and action buttons.

3.3 Screen Orientation

Screen orientation refers to the direction in which an Android device displays its screen.
There are two main orientations:

1. **Portrait**
2. **Landscape**



Portrait Orientation

In **portrait orientation**, the screen is taller than it is wide.
It is the normal orientation when a phone is held vertically.
Example:

Uses of Portrait Orientation

Portrait mode is commonly used for:

- Reading
- Forms
- Login screens
- Messaging
- Contact lists
- Social media applications

Landscape Orientation

In **landscape orientation**, the screen is wider than it is tall.
It occurs when the device is held horizontally.

Uses of Landscape Orientation

Landscape mode is useful for:

- Games

- Video playback
- Presentations
- Tables
- Graphs
- Applications requiring more horizontal space

Automatic Orientation

Android devices can automatically change the screen orientation when the user rotates the device, provided that the application and device settings allow orientation changes.

Orientation and User Interface

An application should be designed so that its interface remains usable when orientation changes.
For example, a form that looks good in portrait mode should not become unusable when the screen changes to landscape mode.

3.4 Android Layouts

An **Android layout** defines how UI elements are arranged on the screen.
A layout determines:

- Position of widgets
- Size of widgets
- Relationship between widgets
- Spacing between widgets
- Alignment of widgets
- Overall screen structure

For example, suppose a registration screen contains:

- Name
- Email
- Password
- Submit button

The layout determines where each of these components appears.

Why Layouts Are Important

Layouts help developers create:

- Organized interfaces
- Attractive screens
- User-friendly applications
- Interfaces that work on different screen sizes

The major layouts in this unit are:

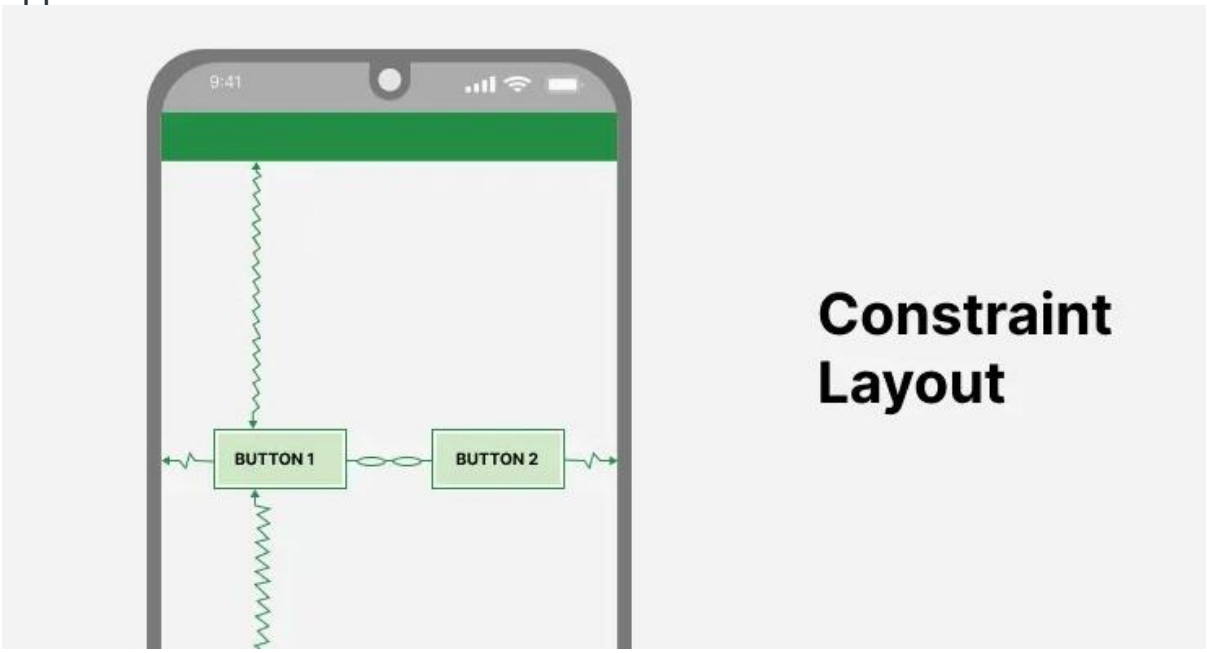
1. Constraint Layout
2. Linear Layout
3. Table Layout
4. Grid View

Layout Managers (or simply layouts) are said to be extensions of the ViewGroup class. They are used to set the position of child Views within the UI we are building. We can nest the layouts, and therefore we can create arbitrarily complex UIs using a combination of layouts.

There is a number of layout classes in the Android SDK. They can be used, modified or can create your own to make the UI for your Views, Fragments and Activities. You can display your contents effectively by using the right combination of layouts.

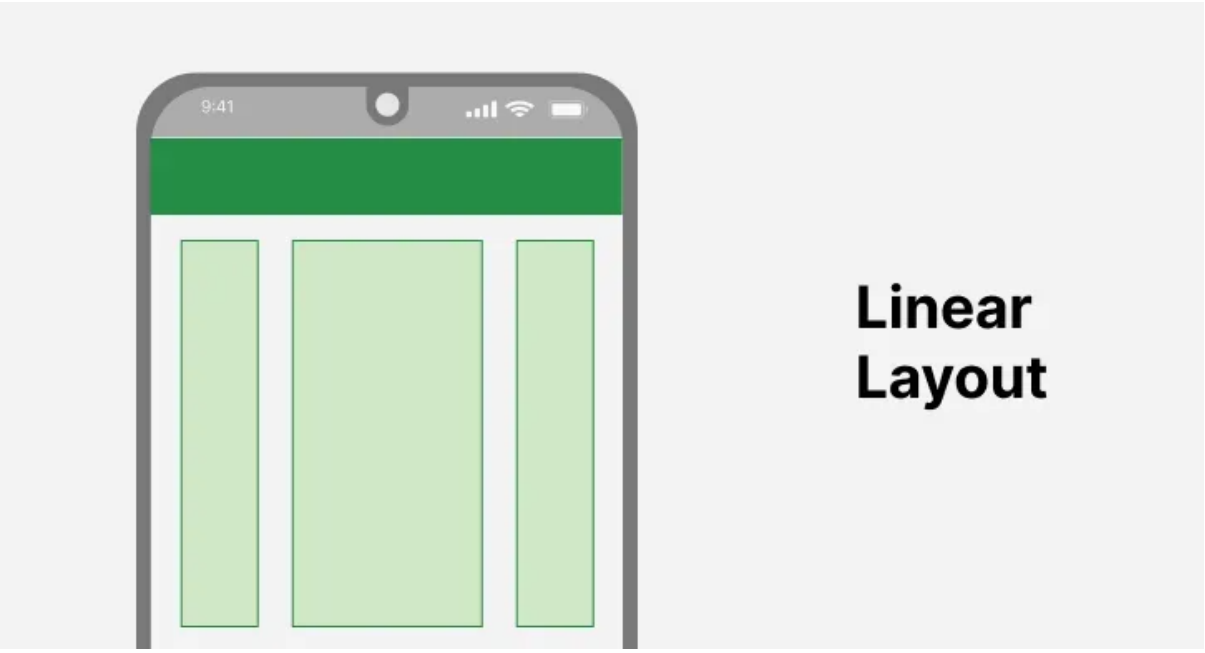
1. ConstraintLayout

Constraint Layout is a **flexible** layout that allows views to be **positioned using constraints** instead of nesting layouts. It helps in reducing the view hierarchy and improves performance by a lot. It is the **most popular** and recommended layout for designing complex interfaces in modern Android applications.



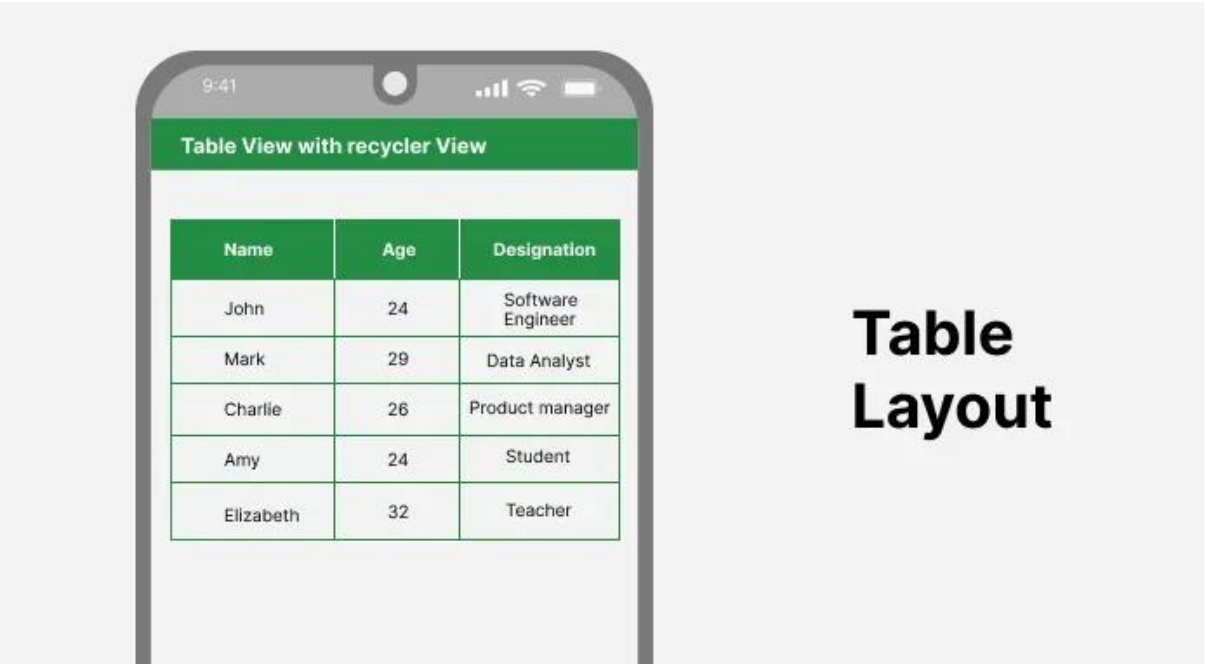
2. LinearLayout

A LinearLayout aligns each of the view in a **single direction** either a **vertical** or a **horizontal** manner. A layout in vertical manner has a single column of views, whereas in a layout in horizontal manner, has a single row of views. It supports a **weight attribute** for each view that can control the relative size of each view within the available space. However, excessive nesting of Linear Layouts can lead to **performance issues**, so it's best for straightforward layouts with a few elements.



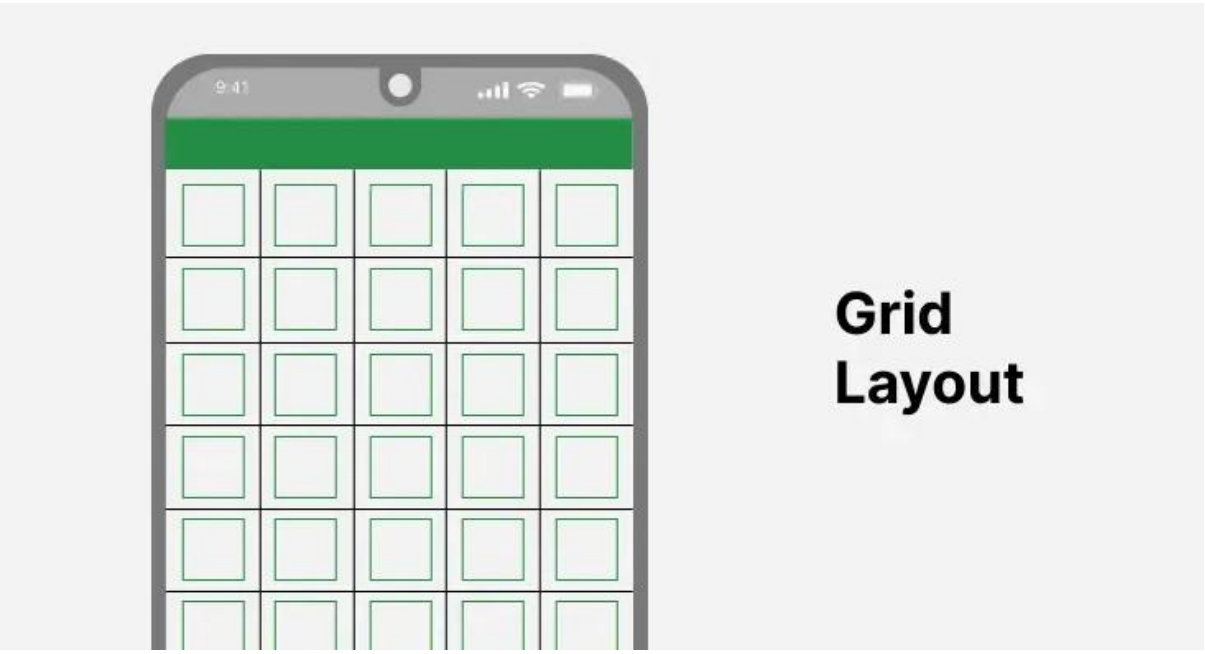
3. TableLayout

TableLayout arranges views in a grid-like format using rows and columns, similar to an HTML table. It is useful for displaying structured data, such as forms or spreadsheets. Each row consists of multiple **TableRow** elements, and views within rows can be assigned specific column spans.



6. GridLayout

Introduced in Android 4.0 (API level 14), the GridLayout is an advanced and more flexible alternative to TableLayout. It divides the screen into a matrix of rows and columns, allowing precise placement of elements without needing of nested layouts. It is efficient for creating grid-based UIs like image galleries or dashboards.

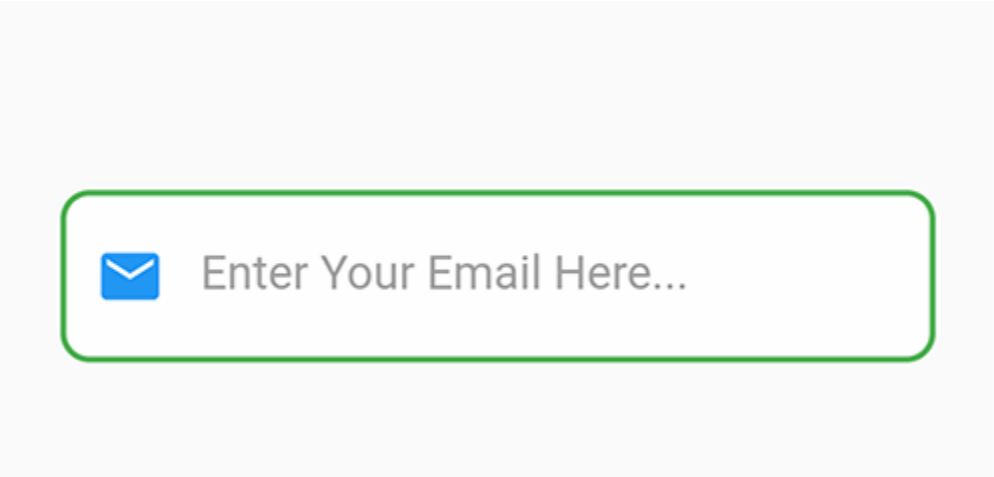


3.5 Text Field, Button, CheckBox, RadioButton

These are important Android widgets used for user interaction.

3.5.1 Text Field

TextField is a UI element that lets users type in text as an input. This input can then be stored and used for various desired functions. In General, there is no hint for a TextField. However, we can customize TextField to display hints to the user.



A **Text Field** allows the user to enter information.
Examples:

- Name
- Email
- Password
- Mobile number
- Address
- Search text

Types of Input

Depending on its configuration, a text field can accept:

- Normal text
- Numbers
- Email addresses
- Passwords
- Phone numbers

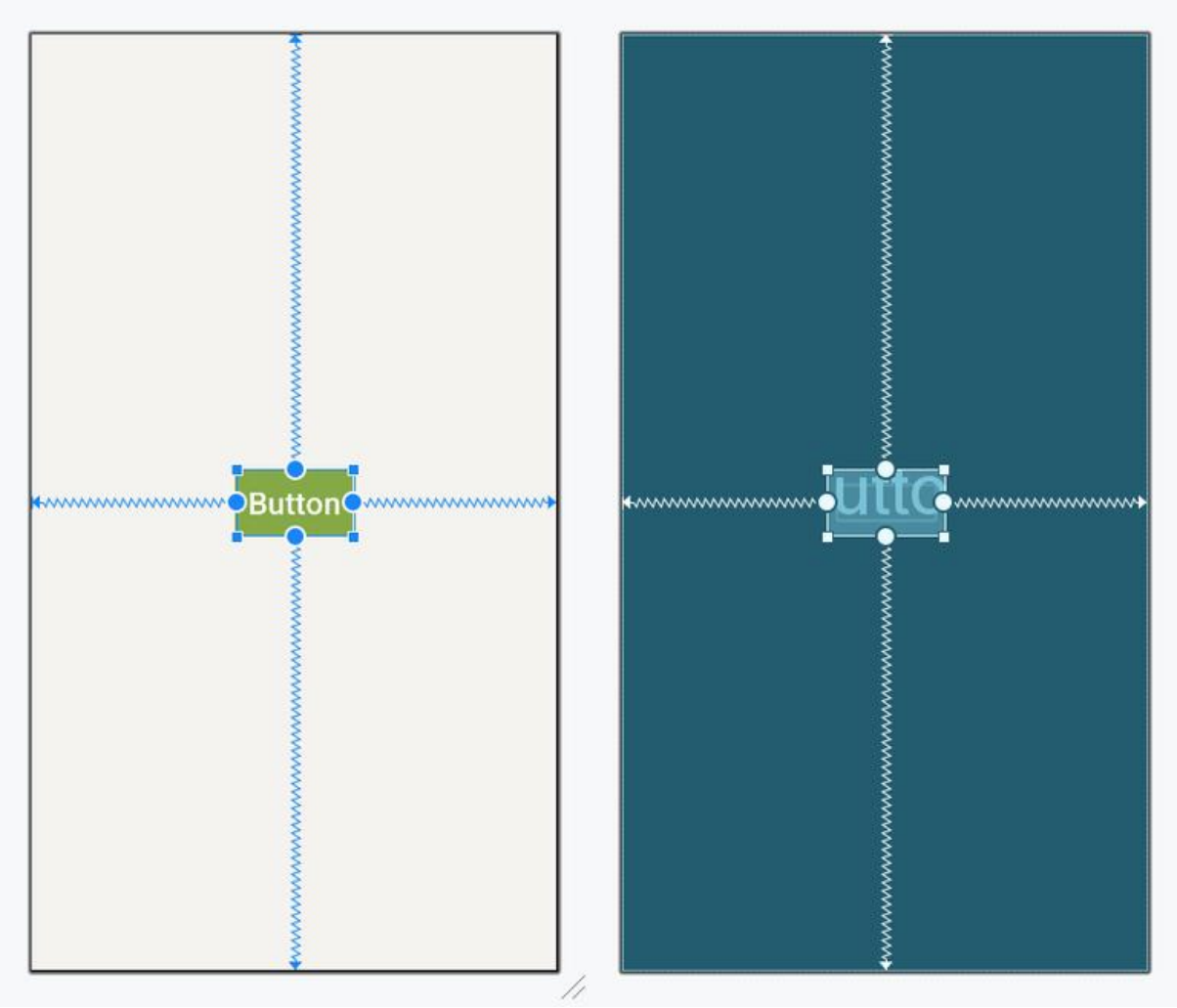
Uses

Text fields are commonly used in:

- Login forms
- Registration forms
- Search boxes
- Feedback forms
- Data-entry applications

Button

In Android applications, a **Button** is a user interface that is used to perform some action when clicked or tapped. It is a very common widget in Android and developers often use it. This article demonstrates how to create a button in Android Studio.



A **Button** is a UI component that allows the user to perform an action by pressing or clicking it.
Examples:

- Login
- Submit
- Save

- Delete
- Cancel
- Search

For example:
LOGIN
When the user presses the Login button, the application can process the entered information and perform the required action.

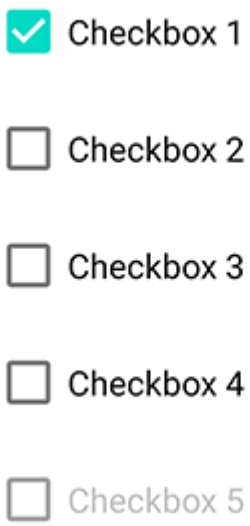
Characteristics

A button usually contains:

- Text
- A visible clickable area
- An associated action

CheckBox

CheckBox is used for adding multiple selections of items from the given set of options. This is seen used in many android applications for adding a feature for multiple selections. In this article, we will take a look at How to implement Checkbox in Android.



A **CheckBox** allows the user to select or deselect an option.
It is generally used when **multiple options can be selected at the same time**.
Example:

☐ `Java`
☐ `Python`
☐ `Android`
☐ `Web Development`

The user can select:

- Java
- Python
- Android

at the same time.

Example

A registration form might ask:

Select your interests:

- Programming
- Sports
- Music
- Photography

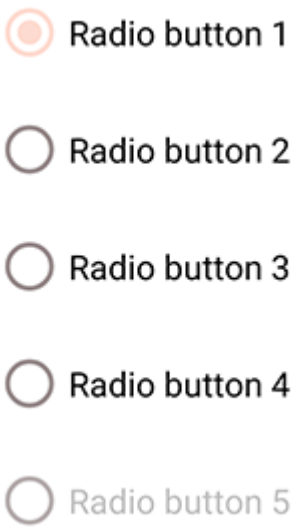
The user can select several interests.

Important Point

CheckBox is suitable when zero, one, or multiple options can be selected.

RadioButton

RadioButton is a widget used in android which provides functionality to choose a single option from multiple sets of options. This is generally used when we want the user to select only one option from the set of given options



RadioGroup in Android

A **RadioGroup** is a container used to organize multiple **RadioButtons** into a single group.

Its main purpose is to make sure that the user can select **only one RadioButton at a time** from that group.

Simple Example

Suppose an application asks:
Select your gender:

- ○ Male
- ○ Female
- ○ Other

All three RadioButtons can be placed inside one **RadioGroup**.
When the user selects **Male**, it becomes selected. If the user then selects **Female**, **Male is automatically unselected**.
So, the RadioGroup provides **mutually exclusive selection**.

Why is RadioGroup Used?

Without a RadioGroup, multiple RadioButtons may not behave as one selection group.
RadioGroup tells Android:
"These RadioButtons belong together, and normally only one of them should be selected."

Examples of RadioGroup

RadioGroup is useful when the user must choose **one option from several choices**.

Example 1: Gender

- Male
- Female
- Other

Example 2: Payment Method

- Cash
- Card
- UPI

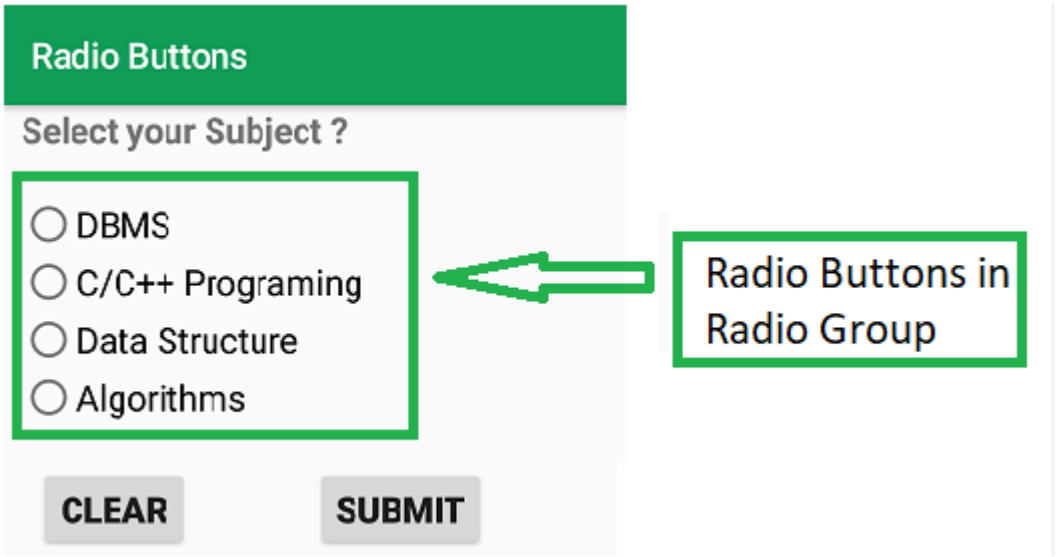
Example 3: Education

- School
- College
- University

Example 4: Size

- Small
- Medium
- Large

In all these situations, the user normally needs to select **one option**.



3.5.1 Event Handling on Button, CheckBox and RadioButton

Event handling in Android means managing the actions performed by a user and making the application respond appropriately.
Examples of events include:

- Pressing a button
- Selecting a checkbox
- Selecting a radio button
- Touching or holding a widget

Android manages events through **event listeners** and **event handlers**.

Event Listener

An **Event Listener** is responsible for detecting when a particular event occurs on a widget. When the event occurs, it calls the appropriate callback method.

Event Handler

An **Event Handler** performs the required action after an event occurs.
For example:

Button pressed → Event Listener detects it → Event Handler performs the required action

Event Listener Registration

Event registration means associating an event listener with a widget so that the required handler is called when the event occurs.

Touch Mode

Touch Mode is used when the user interacts with an Android application through a touchscreen. In touch mode, focus is generally not required for every view because the user directly touches the desired component.

Common Event Listeners

- **OnClickListener** – triggered when the user clicks a widget such as a Button.
- **OnLongClickListener** – triggered when the user presses and holds a widget.
- **OnMenuItemClickListener** – triggered when the user selects a menu item.
- **OnTouchListener** – triggered when the user touches or performs a movement gesture.

Event Handling on Button

A Button generates a **click event** when the user presses it.
There are two common ways to handle a Button click:

1. **Using** android:onClick **in XML** – specifies the method to be called when the button is clicked.
2. **Using** OnClickListener – registers a listener programmatically to respond to the click.

Example flow:
User presses Button → Click event occurs → Listener detects event → Handler performs action
The action could be changing text, changing color, opening another screen, or saving information.

Event Handling on CheckBox

A CheckBox generates an event whenever its state changes.
It has two main states:

- **Checked**
- **Unchecked**

For example, a **Receive Notifications** checkbox can enable notifications when selected and disable them when unselected.
User selects CheckBox → State changes → Event occurs → Application responds

---

Event Handling on RadioButton

A RadioButton generates an event when the user changes the selected option.
For example:
Payment Method:

- ○ Cash
- ○ Card
- ○ UPI

If the user selects **UPI**, the application can display the appropriate payment options.
User selects RadioButton → Selection changes → Event occurs → Application responds

3.6 ToggleButton

ToggleButton is basically a stop/play or on/off button with an indicator light indicating the current state of ToggleButton. ToggleButton is widely used, some examples are on/off audio, Bluetooth, WiFi, hot-spot etc. This is a subclass of **Composite Button**.

ToggleButton allows users to change settings between two states from their phone's Settings menu such as turning their WiFi, Bluetooth, etc. on / off. Since the Android 4.0 version (API level 14), it has another type of toggle button called a **switch** which provides user slider control.

Layout to display the ToggleButton

Operation	XML Attribute
android:textOn	Used to change the text to be displayed when the button is On. By default text is "ON"
android:textOff	Used to change the text to be displayed when the button is Off. By default text is "OFF"
android:disabledAlpha	The alpha to apply to the indicator when disabled.

ToggleButton vs CheckBox

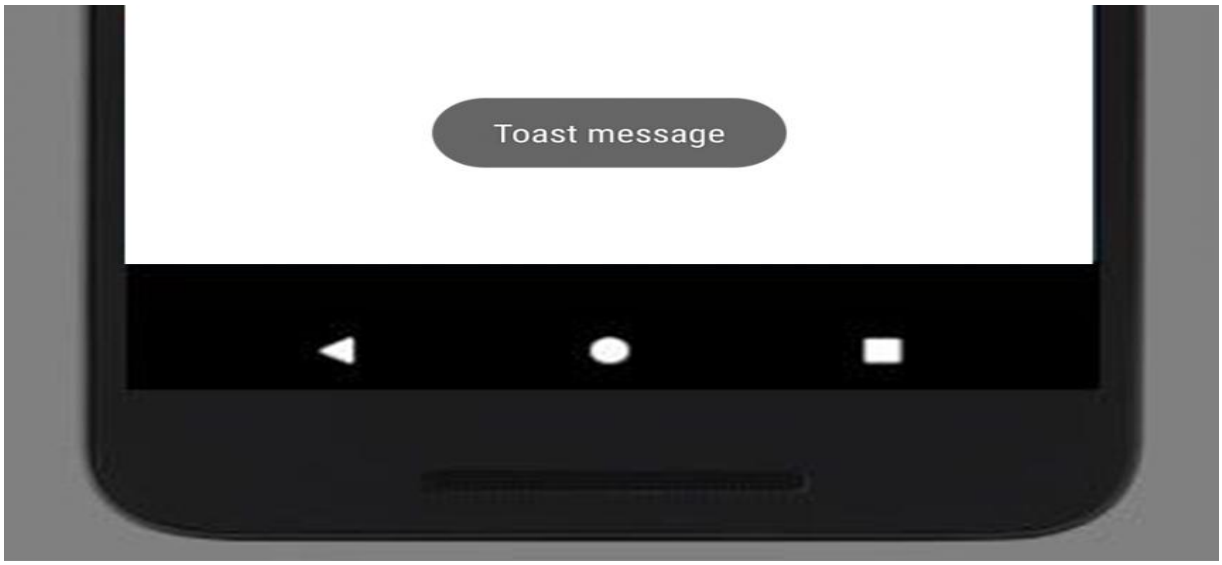
Both can represent a two-state choice, but their typical usage differs.

ToggleButton	CheckBox
Represents an ON/OFF state	Represents selection/deselection
Common for settings	Common for selecting options
Example: Dark Mode ON/OFF	Example: Select hobbies
Visually emphasizes current state	Usually appears as a checkbox

3.7 Toast

What is Toast in Android?

A **Toast** is a feedback message. It takes a very little space for displaying while the overall activity is interactive and visible to the user. It disappears after a few seconds. It disappears automatically. If the user wants a permanently visible message, a **Notification** can be used. Another type of Toast is **custom Toast**, in which images can be used instead of a simple message.



Example: Toast class provides a simple popup message that is displayed on the current activity UI screen (e.g. Main Activity).

Constants of Toast class

Constants	Description
public static final int LENGTH_LONG	displays for a long time
public static final int LENGTH_SHORT	displays for a short time

Methods of Toast class

Methods	Description
public static Toast makeText(Context context, CharSequence text, int duration)	makes the toast message consist of text and time duration
public void show()	displays a toast message
public void setMargin (float horizontalMargin, float verticalMargin)	changes the horizontal and vertical differences

Characteristics of Toast

1. **Short message**
   Toast is generally used for brief information.
2. **Temporary**
   It disappears automatically after a short period.
3. **Does not require user interaction**
   The user does not normally need to press a button to close it.
4. **Provides feedback**
   It informs the user that an action occurred or provides simple status information.

Common Uses

Toast can be used to show:

- Successful operations
- Errors or warnings
- Confirmation messages
- Status information
- User action feedback

Unit 4: Android Widgets

4.1 Alert Dialog Box

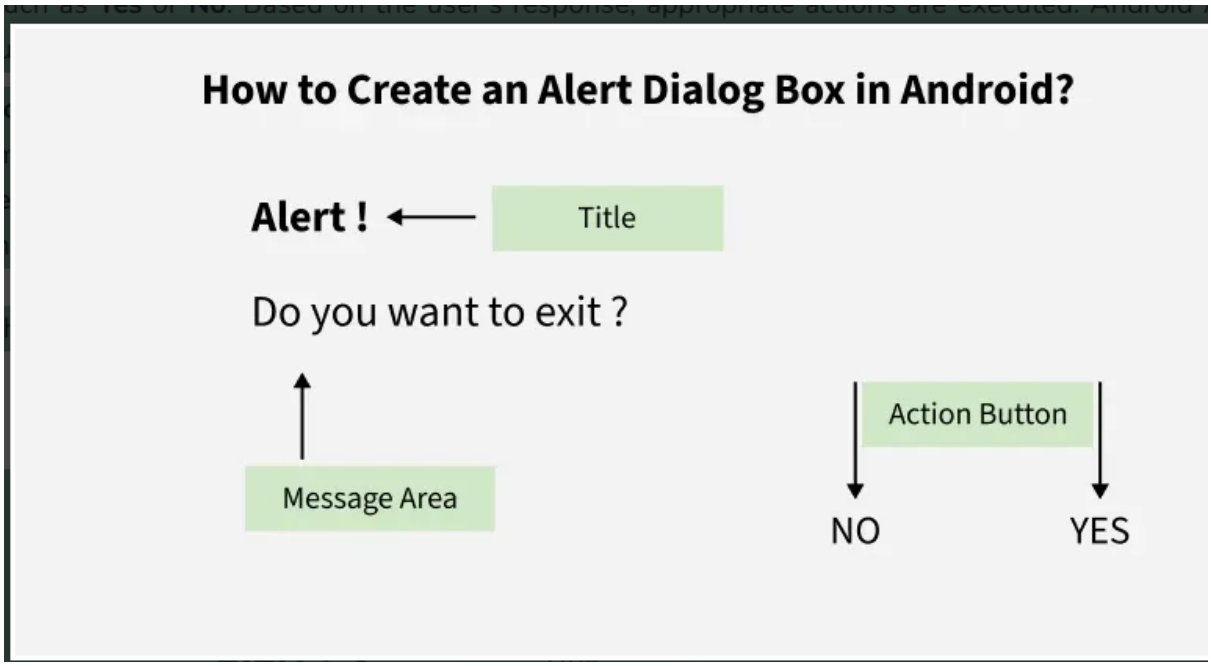
An **Android Alert Dialog** is a UI element that displays a warning or notification message and asks the user to respond with options such as **Yes** or **No**. Based on the user's response, appropriate actions are executed. Android Alert Dialog is built with the use of three fields: **Title, Message area, and Action Button.**

Alert Dialog code has three methods :

- **setTitle():** method for displaying the Alert Dialog box Title
- **setMessage():** method for displaying the message
- **setIcon():** method is used to set the icon on the Alert dialog box.

Then we add the two Buttons, **setPositiveButton** and **setNegativeButton** to our Alert Dialog Box as shown below.

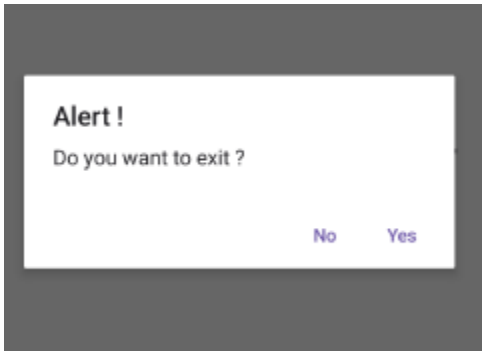
Example:



An **Alert Dialog Box** is a small pop-up window that appears on top of the current screen to provide information, ask for confirmation, or get a response from the user. It temporarily interrupts the user's activity and requires the user to read or interact with it.

Example

Suppose the user wants to exit from activity. The application can display:



Uses of Alert Dialog

Alert dialogs are commonly used for:

- Confirmation messages
- Warning messages
- Error messages
- Important information
- Asking the user to make a choice
- Confirming deletion or logout

Main Parts of an Alert Dialog

An Alert Dialog can contain:

1. **Title** – describes the purpose of the dialog.
2. **Message** – provides additional information.
3. **Buttons** – allow the user to perform an action.
4. **Optional content** – such as a list, input field, or other UI component.

Types of Buttons

Common buttons include:

- **Positive button** – performs the main/confirm action, such as OK, Save, or Yes.
- **Negative button** – cancels or rejects an action, such as Cancel or No.
- **Neutral button** – performs an alternative action.

Example Flow

User selects Delete → Alert Dialog appears → User selects Yes → File is deleted

Advantages

- Gives immediate feedback.
- Helps prevent accidental actions.
- Useful for important decisions.
- Easy for users to understand.

4.2 ListView

A ListView in Android is a type of AdapterView that displays a vertically scrollable list of items, with each item positioned one below the other. Using an adapter, items are inserted into the list from an array or database efficiently. For displaying the items in the list method `setAdaptor()` is used. The `setAdaptor()` method binds the adapter with the ListView.
ListView in Android is a ViewGroup that is used to display the list of items in multiple rows and contains an adapter that automatically inserts the items into the list dynamically. The main purpose of the adapter is to retrieve data from an array or database and dynamically insert each item into the list for the desired result. Adapters can fetch data from various sources including resources like the *strings.xml* file.

Some Important XML Attributes of ListView

Attribute	Description
android:divider	A color or drawable to separate list items.
android:dividerHeight	Divider's height.
android:entries	Reference to an array resource that will populate the ListView.
android:footerDividersEnabled	When set to false, the ListView will not draw the divider before each footer view.
android:headerDividersEnabled	When set to false, the ListView will not draw the divider before each header view.

A **ListView** is an Android UI component used to display a collection of items in a **vertically scrollable list**.

Uses of ListView

ListView can be used for:

- Contact lists
- Student lists
- Menu items
- Messages
- Product lists
- Country lists
- News headlines

Main Components

A ListView generally involves:

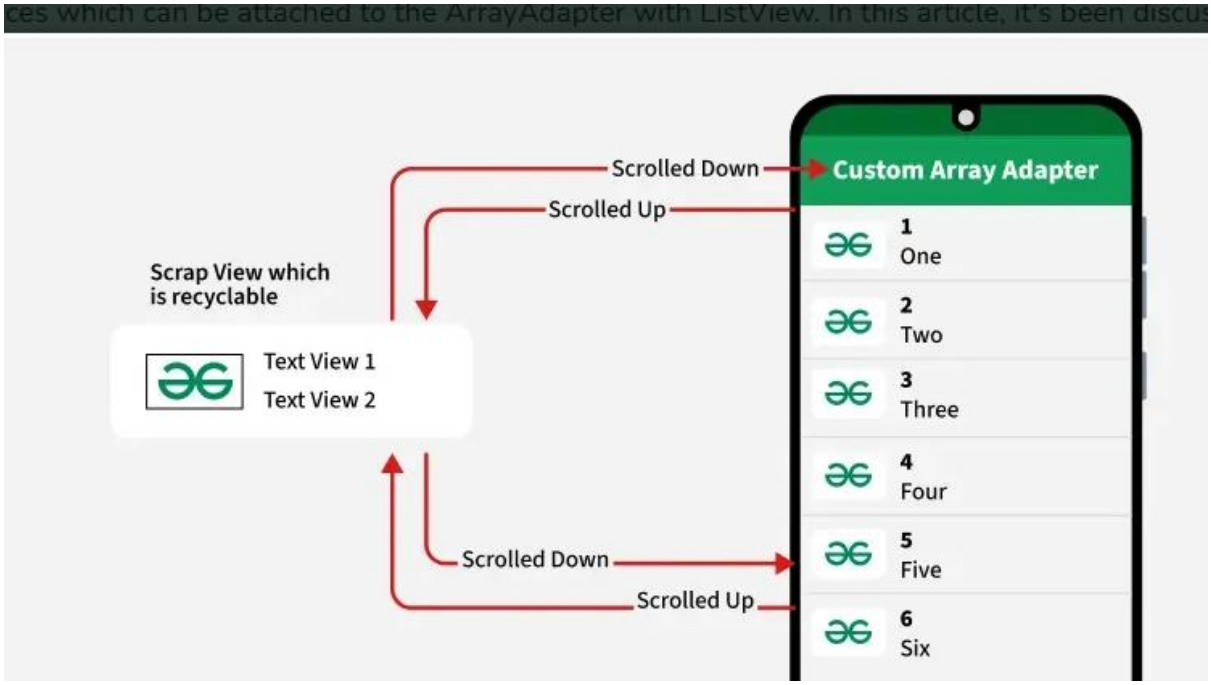
1. **ListView** – displays the list.
2. **Data source** – contains the information to be displayed.
3. **Adapter** – connects the data to the ListView.

Adapter

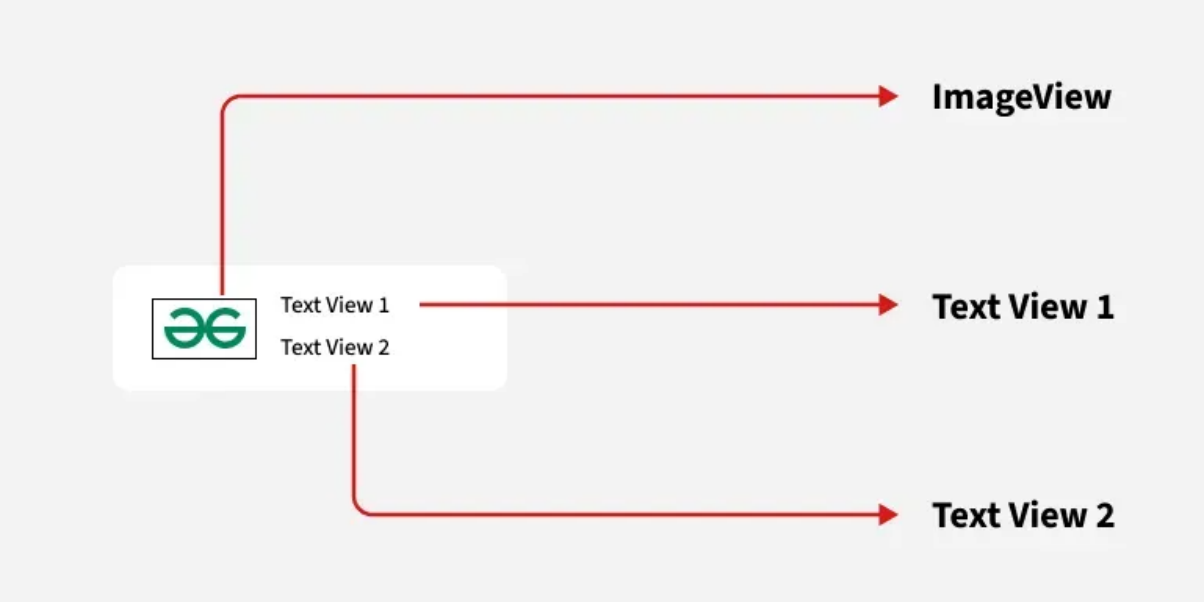
An **Adapter** acts as a bridge between the data and the ListView.

4.3 Custom ListView

- The Adapter acts as a bridge between the UI Component and the Data Source. It converts data from the data sources into view items that can be displayed into the UI Component. Data Source can be Arrays, HashMap, Database, etc. and UI Components can be ListView, GridView, Spinner, etc.
- **ArrayAdapter** is the most commonly used adapter in android. When you have a list of single type items which are stored in an array you can use ArrayAdapter. Likewise, if you have a list of phone numbers, names, or cities. ArrayAdapter has a layout with a single TextView. If you want to have a more complex layout instead of ArrayAdapter use **CustomArrayAdapter**
- The ArrayAdapter works and what are the data sources which can be attached to the ArrayAdapter with ListView. In this article, it's been discussed how to implement custom ArrayAdapter with the ListView. Have a look at the following image in which a single view in the ArrayAdapter can be customized.



For every single item in the ListView this layout creates the following view for every single item in the array adapter.



Difference Between ListView and Custom ListView

ListView	Custom ListView
Usually displays simple list items	Displays specially designed list items
Simple appearance	More attractive and flexible
Suitable for basic lists	Suitable for complex information
Example: List of names	Example: Photo + name + description

4.4 ImageView

- **ImageView** is an Android widget used to **display images** in an application.
- Images can be used to improve the appearance and usability of an application.
- **ImageView** class is used to display any kind of image resource in the android application either it can be **android.graphics.Bitmap** or **android.graphics.drawable.Drawable** (It is a general abstraction for anything that can be drawn in Android). ImageView class or **android.widget.ImageView** inherits the **android.view.View** class which is the subclass of Kotlin. **AnyClass.Application** of ImageView is also in applying tints to an image in order to reuse a drawable resource and create overlays on background images. Moreover, ImageView is also used to control the size and movement of an image.
- Adding an ImageView to an Activity
    - Whenever ImageView is added to an activity, it means there is a requirement for an image resource. Thus it is oblivious to provide an Image file to that ImageView class. It can be done by adding an image file that is present in the Android Studio itself

or we can add our own image file. Android Studio owns a wide range of drawable resources which are very common in the android application layout.

Important Functions of ImageView

ImageView can be used to:

- Display an image.
- Control how the image fits inside its area.
- Display different images depending on application requirements.
- Support visual content in the user interface.

Scale Types

Images may need to be adjusted to fit the available space.
Common image scaling concepts include:

- Fit the entire image inside the available area.
- Fill the available area.
- Center the image.
- Crop the image while maintaining its appearance.

Uses

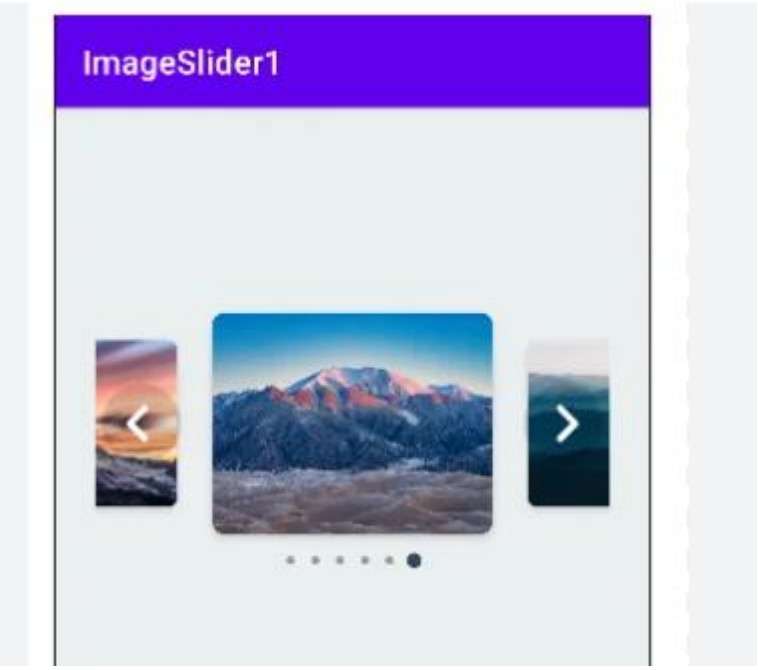
ImageView is commonly used in:

- Profile screens
- Gallery applications
- Product applications
- News applications
- Educational applications
- Image-based menus

4.4 Image Slider

An **Image Slider** is a UI component that displays multiple images one after another, usually allowing the user to move between them.

- Image Slider is one of the most seen UI components in Android.
- This type of slider is mostly seen inside the apps of big E-commerce sites such as Flipkart, Amazon, and many more.
- Auto Image Slider is used to represent data in the form of slides that change after a specific interval of time.



Features of Image Slider

An Image Slider may support:

- Next and previous image navigation
- Swipe gestures
- Automatic image changes
- Page indicators
- Multiple images

Uses

Image Sliders are useful for:

- Product advertisements
- Image galleries
- Banners
- News images
- Promotional content
- App introductions

Advantages

- Displays multiple images in limited space.
- Makes the interface visually attractive.
- Allows users to browse images easily.

4.5 SearchView

- SearchView is a widget provided by the Android framework that allows users to search for specific data within an application. It is commonly used in apps that have large amounts of data or content that can be searched, such as contacts, music, or emails.
- The SearchView widget consists of an input field and a search button. Users can enter a search query into the input field, and then click the search button to initiate the search. The search query can be entered using the device's keyboard, and the search results are displayed in the same activity or fragment as the SearchView.

- ListView
    - ListView is a UI component provided by the Android framework that displays a list of items in a vertical scrollable view. It is commonly used in Android applications to display data that can be scrolled vertically, such as a list of contacts, messages, or news articles.

Working of SearchView

The basic process is:
User enters search text → Application receives the text → Data is searched/filtered → Matching results are displayed

Uses

SearchView can be used in:

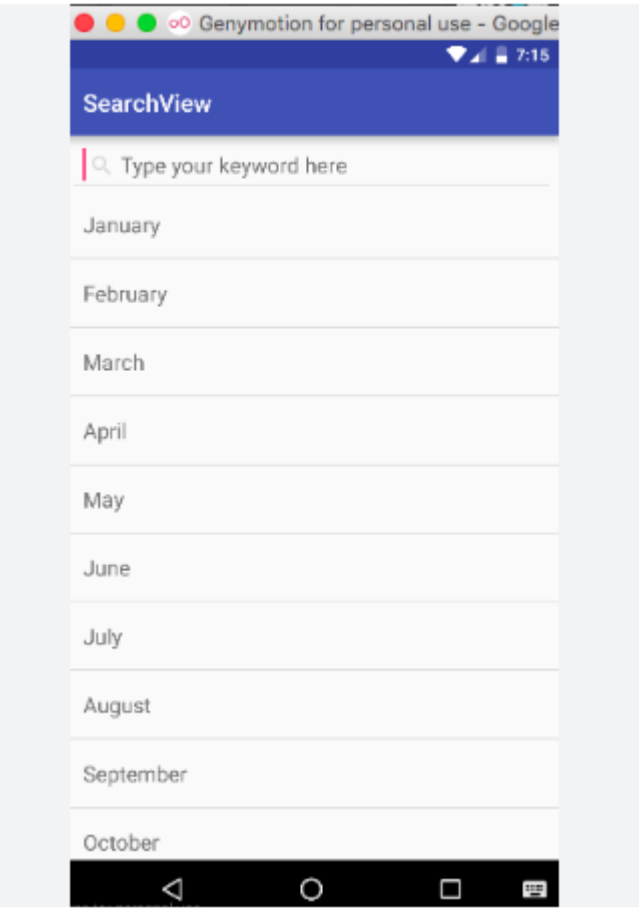
- Contact applications
- Shopping applications
- Music applications
- File managers
- Libraries
- News applications
- Student databases

Advantages

- Makes finding information easier.
- Saves time when there are many items.
- Provides a convenient way to filter information.

Search and ListView

SearchView is often used together with a list.



4.6 Intent

- In Android, it is quite usual for users to witness a jump from one application to another as a part of the whole process, for example, searching for a location on the browser and witnessing a direct jump into Google Maps or receiving payment links in Messages Application (SMS) and on clicking jumping to PayPal or GPay (Google Pay).
- This process of taking users from one application to another is achieved by passing the **Intent** to the system. **Intents** , in general, are used for navigating among various activities within the same application, but note, is not limited to one single application, i.e., they can be utilized from moving from one application to another as well.
- **Intents** could be Implicit, for instance, calling intended actions, and explicit as well, such as opening another activity after some operations like onClick or anything else. Below are some applications of Intents:

1. Sending the User to Another App
2. Getting a Result from an Activity
3. Allowing Other Apps to Start Your Activity

The collaborative nature of Android applications only results in a better user experience. The question here is if the intent is for an application that is not present in the device, what's the next call?

Types of Android Intents

There are two types of intents in android

1. Implicit
2. Explicit

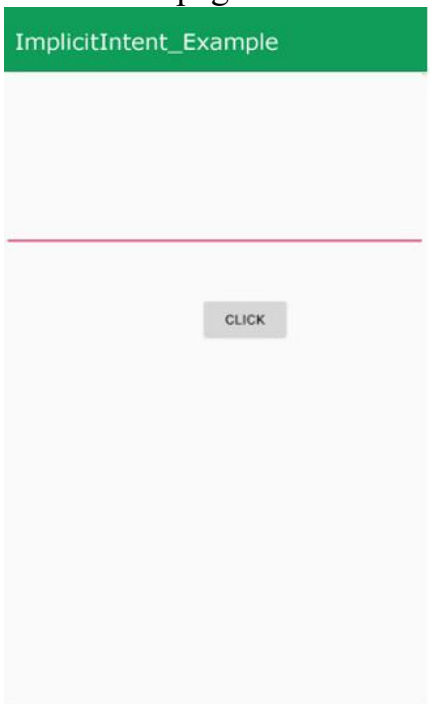
Implicit Intent

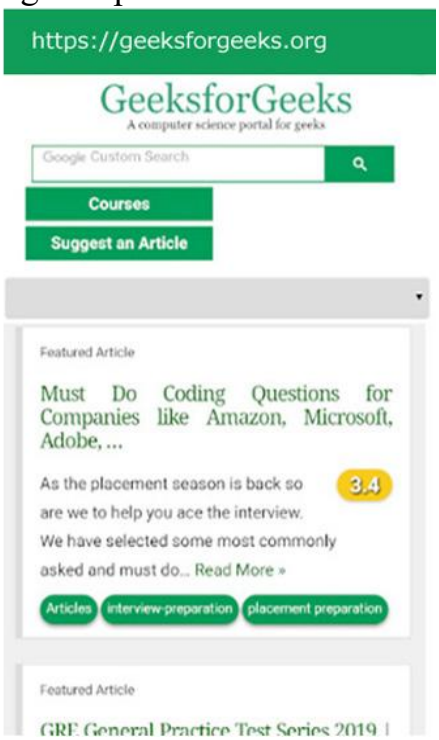
Implicit Intent doesn't specify the component. In such a case, intent provides information on available components provided by the system that is to be invoked. For example, you may write the following code to view the webpage.

Syntax:

```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.geeksforgeeks.org/"));
startActivity(intent);
```

For Example: In the below images, no component is specified, instead, an action is performed i.e. a webpage is going to be opened. As you type the name of your desired webpage and click on the 'CLICK' button. Your webpage is opened.





Explicit Intent

Explicit Intent specifies the component. In such a case, intent provides the external class to be invoked.

Syntax:

```
Intent i = new Intent(getApplicationContext(),
ActivityTwo.class);
startActivity(i);
```

For Example: In the below example, there are two activities (FirstActivity, and SecondActivity). When you click on the 'GO TO OTHER ACTIVITY' Button in the FirstActivity, then you move to the SecondActivity. When you click on the 'GO TO HOME ACTIVITY' button in the SecondActivity, then you move to the first activity. This is getting done through Explicit Intent.