Unit 1:
Introduction to Android

@Mukesh Singh Bhandari

Introduction to Android

Android is a mobile operating system that powers a wide range of devices, including smartphones, tablets, smart TVs, and wearable devices.

It is developed by the Open Handset Alliance (OHA), a consortium of companies led by Google.

Android provides a flexible and customizable platform for developers to create applications, and it has become one of the most widely used operating systems in the world

Open Handset Alliance (OHA)

- The Open Handset Alliance (OHA) was formed in 2007.
- It is a consortium of technology and mobile companies.

Members:

OHA comprises a diverse group of companies from various sectors of the mobile industry, including handset manufacturers, software developers, chipmakers, and wireless carriers.

Some notable members of the Open Handset Alliance include:

1. **Google:** The leading force behind the formation of OHA, and the developer of the Android operating system.
2. **Samsung:** A major global electronics and mobile device manufacturer.
3. **HTC (High Tech Computer Corporation):** A Taiwanese consumer electronics company, known for manufacturing smartphones.
4. **LG (LG Electronics):** A South Korean multinational electronics company with a focus on consumer electronics and mobile communications.
5. **Motorola Mobility (a subsidiary of Lenovo):** Motorola has been a historic player in the mobile phone industry.
6. **Sony:** A Japanese multinational conglomerate known for its electronics, entertainment, and gaming products.
7. **Qualcomm:** An American semiconductor and telecommunications equipment company, known for its contributions to mobile technologies.
8. **Intel:** An American multinational corporation that produces semiconductor chips.
9. **NVIDIA:** A technology company known for its graphics processing units (GPUs) and mobile processors.
10. **Telecom Operators:** Various telecommunications operators may also be part of the alliance, though individual operators may vary by region.

History of Android:

1. **Foundation of Android Inc. (2003):** Android's story begins with the foundation of Android Inc. in October 2003 by Andy Rubin, Rich Miner, Nick Sears, and Chris White. The company aimed to develop an advanced operating system for digital cameras but later shifted its focus to mobile phones.
2. **Acquisition by Google (2005):** In August 2005, Google acquired Android Inc., and Andy Rubin continued to lead the Android project within Google. The acquisition marked Google's entry into the mobile operating system space.
3. **Open Handset Alliance Formation (2007):** In November 2007, Google announced the formation of the Open Handset Alliance (OHA), a consortium of companies collaborating to develop open standards for mobile devices. The OHA aimed to foster the development of the Android platform.
4. **Android Beta Release (2007):** The first beta version of the Android operating system was released in November 2007. This initial release was part of the early development phase and aimed at garnering feedback from developers.
5. **Commercial Launch of Android (2008):** The first commercially available Android device, the HTC Dream, also known as the T-Mobile G1, was released in September 2008. This marked the official launch of Android as a mobile operating system for consumers.
6. **Early Android Versions (2008-2009):** The early versions of Android included Cupcake (1.5), Donut (1.6), and Eclair (2.0/2.1). Each version introduced new features and improvements, laying the foundation for the platform's growth.
7. **Expansion and Market Dominance (2010s):** Android experienced rapid growth and became the dominant mobile operating system globally. Various device manufacturers adopted Android, contributing to its widespread popularity.

1. **Naming Conventions (2011 Onward):** Android versions were named alphabetically after desserts, starting with Cupcake. Notable versions included Gingerbread, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat, Oreo, and Pie. Android 10 (2019) marked a shift away from dessert-themed names.
2. **Introduction of Nexus and Pixel Devices:** Google introduced Nexus devices (2010-2015) and later Pixel devices (2016 onward), showcasing a pure Android experience and serving as reference devices for app developers.
3. **Focus on Security and Privacy (2010s):** With each iteration, Android placed increased emphasis on security and user privacy, introducing features like Google Play Protect and permission controls.
4. **Project Treble and Project Mainline (2017):** Google introduced Project Treble to streamline the Android update process and Project Mainline to modularize system components for easier updates.
5. **Foldable and 5G Era (2019 Onward):** Android adapted to emerging technologies, including the introduction of support for foldable devices and the integration of 5G capabilities.
6. **Numerical Versioning (2020 Onward):** Android versions transitioned to numerical naming conventions, with Android 10, Android 11, and subsequent versions, providing a simpler and more universal approach.

Features of Android:

- **Open Source:** Android is an open-source OS, allowing developers to access and modify its source code.
- **Customization:** Users can personalize their devices through customizable interfaces, widgets, and themes.
- **Multitasking:** Supports running multiple applications simultaneously.
- **Application Ecosystem:** The Google Play Store hosts a diverse range of applications for users to download.
- **Integration with Google Services:** Seamless integration with Google's suite of services.
- **Cloud Integration:** Android seamlessly integrates with cloud services, enabling users to sync and access their data across multiple devices.
- **Navigation and Location Services:** Android devices come equipped with GPS capabilities for location-based services

Android versions

1. Android 1.0 (No codename) - September 23, 2008
2. Android 1.1 (Petit Four) - February 9, 2009
3. Android 1.5 (Cupcake) - April 27, 2009
4. Android 1.6 (Donut) - September 15, 2009
5. Android 2.0/2.1 (Eclair) - October 26, 2009
6. Android 2.2 (Froyo) - May 20, 2010
7. Android 2.3 (Gingerbread) - December 6, 2010
8. Android 3.0/3.1/3.2 (Honeycomb) - February 22, 2011
9. Android 4.0 (Ice Cream Sandwich) - October 18, 2011
10. Android 4.1/4.2/4.3 (Jelly Bean) - July 9, 2012
11. Android 4.4 (KitKat) - October 31, 2013
12. Android 5.0/5.1 (Lollipop) - November 12, 2014
13. Android 6.0 (Marshmallow) - October 5, 2015
14. Android 7.0/7.1 (Nougat) - August 22, 2016
15. Android 8.0/8.1 (Oreo) - August 21, 2017
16. Android 9 (Pie) - August 6, 2018
17. Android 10 - September 3, 2019
18. Android 11 - September 8, 2020
19. Android 12 - October 4, 2021
20. Android 13 (Tiramisu): Released: August 2023
21. Android 14 (Upside Down Cake): Released: October 2023

Pre-dessert era (2008-2009):

Android 1.0 (no codename)

Android 1.1 (no codename)

Android 1.5 Cupcake

Android 1.6 Donut

Dessert era (2009-2019):

Android 2.0/2.1 Eclair

Android 2.2 Froyo

Android 2.3 Gingerbread

Android 3.0-3.2 Honeycomb (tablet-focused)

Android 4.0 Ice Cream Sandwich

Android 4.1-4.3 Jelly Bean

Android 4.4 KitKat

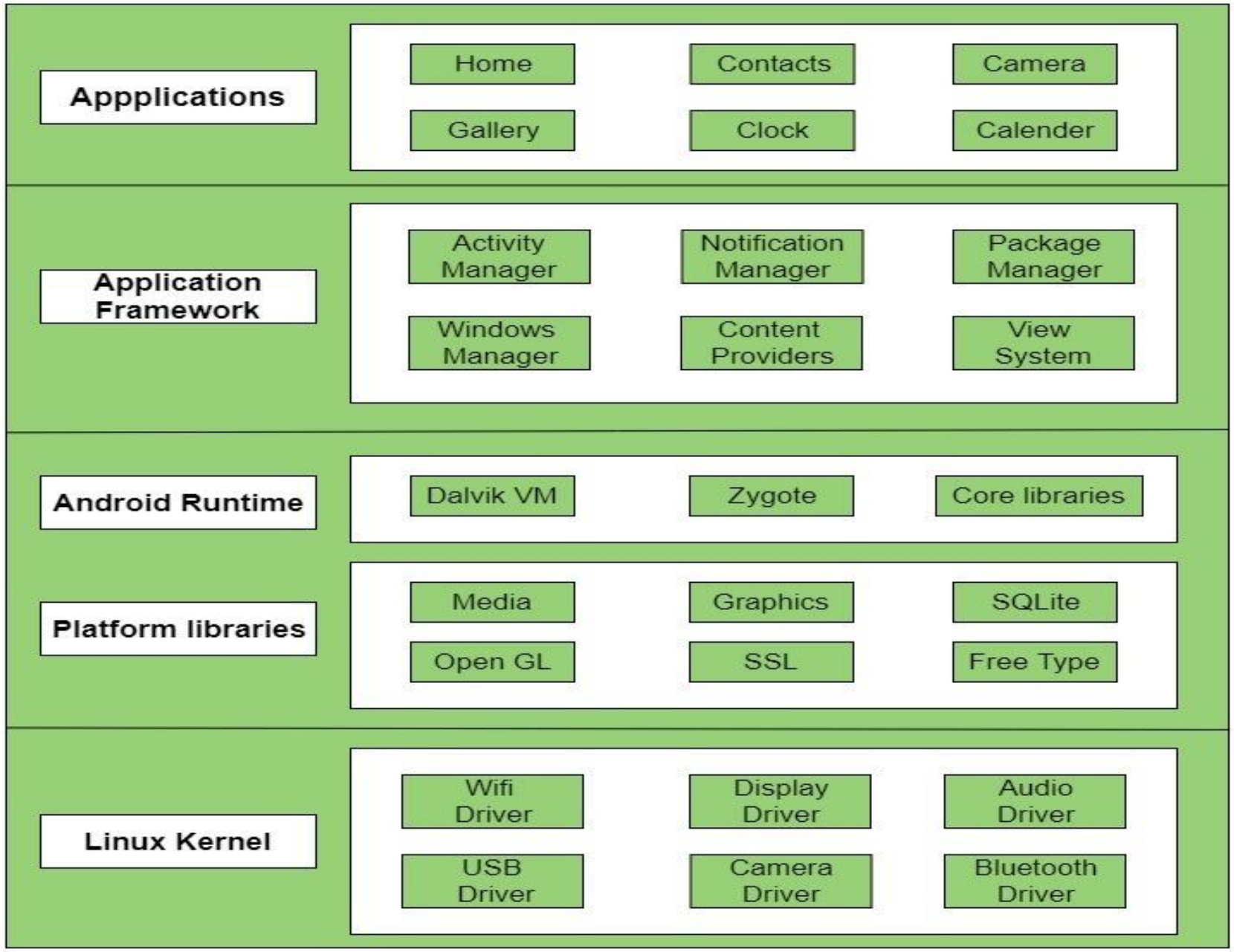
Android 5.0 Lollipop

Android 5.1 Lollipop

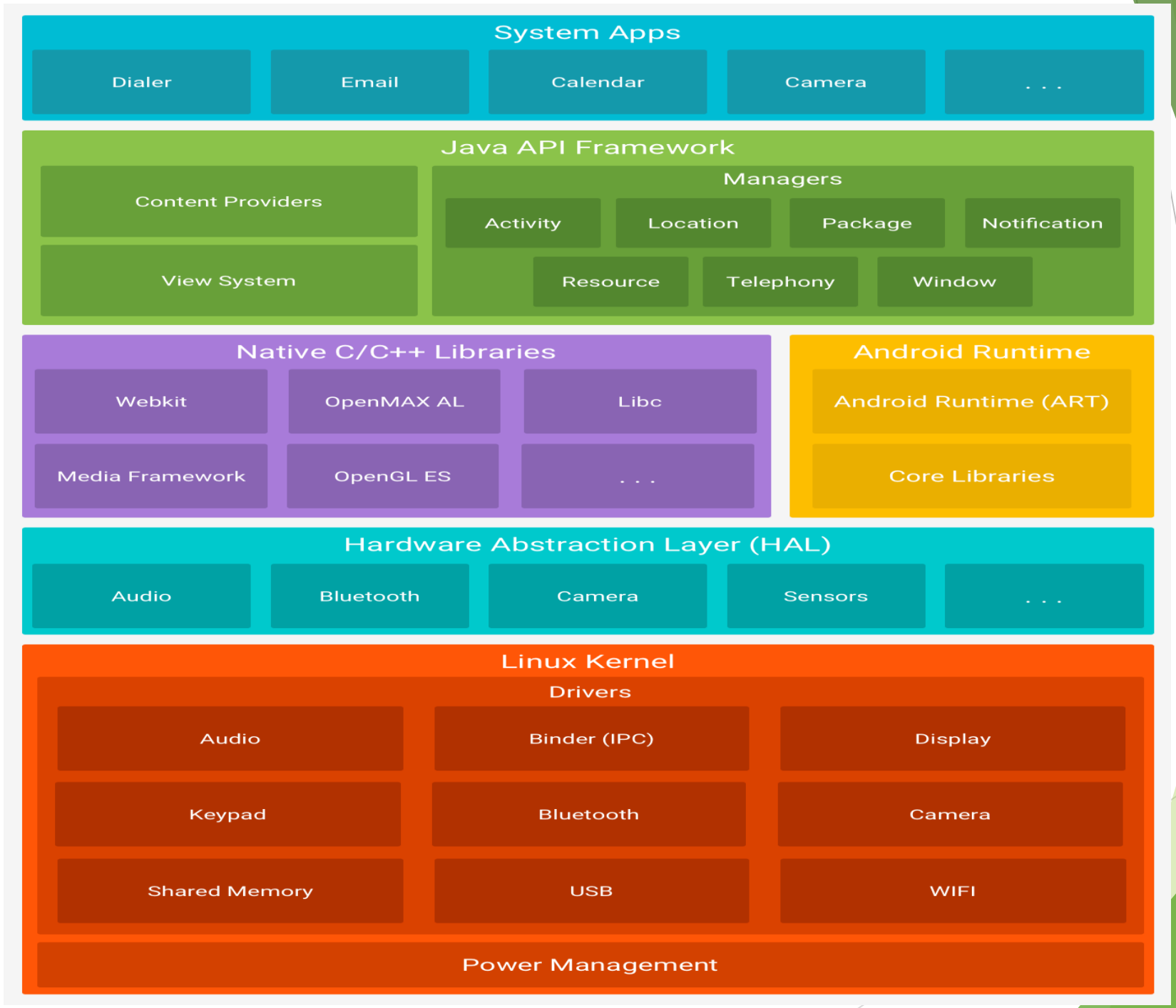
Post-dessert era (2016-present):

- Android 6.0 Marshmallow
- Android 7.0-7.1 Nougat
- Android 8.0-8.1 Oreo
- Android 9 Pie
- Android 10 (no codename)
- Android 11 (no codename)
- Android 12 (no codename)
- Android 13 (Tiramisu)
- Android 14 (Upside Down Cake)

Platform architecture



Hardware abstraction layer (HAL)


System Apps
Dialer
Email
Calendar
Camera
. . .
Java API Framework
Content Providers
View System
Managers
Activity
Location
Package
Notification
Resource
Telephony
Window
Native C/C++ Libraries
Webkit
OpenMAX AL
Libc
Media Framework
OpenGL ES
. . .
Android Runtime
Android Runtime (ART)
Core Libraries
Hardware Abstraction Layer (HAL)
Audio
Bluetooth
Camera
Sensors
. . .
Linux Kernel
Drivers
Audio
Binder (IPC)
Display
Keypad
Bluetooth
Camera
Shared Memory
USB
WIFI
Power Management

Linux kernel

- Linux Kernel is heart of the android architecture.
- 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.
- 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.

hardware abstraction layer (HAL)

- **The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework.**
- **The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth module.**

Android 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).
- 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.
- For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART).

Native C/C++ libraries

- Many core Android system components and services, such as ART and HAL, are built from native code that requires native libraries written in C and C++.
- They provide core functionalities for the system and are often used by applications.

Android Framework

- On the top of Native libraries and android runtime, there is android framework.
- Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers.
- It provides a lot of classes and interfaces for android application development.

Applications

- On the top of android framework, there are applications.
- All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using linux kernal.

ANDROID – Applications Component

- Application components are the essential building blocks of an Android application.
- These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact.
- These components define the different parts of an application's behavior and contribute to the overall user experience. There are following four main components that can be used within an Android application:

In the context of Android development, the Applications Components refer to the building blocks that developers use to create Android applications. These components define the different parts of an application's behavior and contribute to the overall user experience. The main application components in Android are:

1. **Activities:**

   An Activity represents a single screen with a user interface. It is the basic building block for user interactions. Each screen in an Android app is implemented as an activity. Activities can be launched individually or work together to form a cohesive user experience.

2. **Services:**

   A Service is a background process that performs long-running operations or works to handle tasks without a user interface. Services can continue to run even if the user switches to another application. They are often used for tasks such as playing music in the background, fetching data from the internet, or performing other background tasks.

3. **Broadcast Receivers:**

   Broadcast Receivers respond to system-wide broadcast announcements, allowing the application to respond to events even when it's not running. For example, an app might register a broadcast receiver to listen for a network connectivity change or the device being connected to a power source.

4. **Intent:**

   While not a traditional component, an Intent is a fundamental part of the Android system. It's a messaging object used to request an action from another component. For example, you can use an intent to start an activity, initiate a service, or broadcast an event.

5. **Content Providers:**

   Content Providers manage a shared set of application data. They allow data to be shared between different applications or components in a standardized way. Content providers are often used to manage structured data, such as a database of contacts.

Introduction to activities

- An activity represents a single screen with a user interface just like window or frame of Java.
- It represents a single task or action that a user can take within an Android application.

Here are some key points about Android activities:

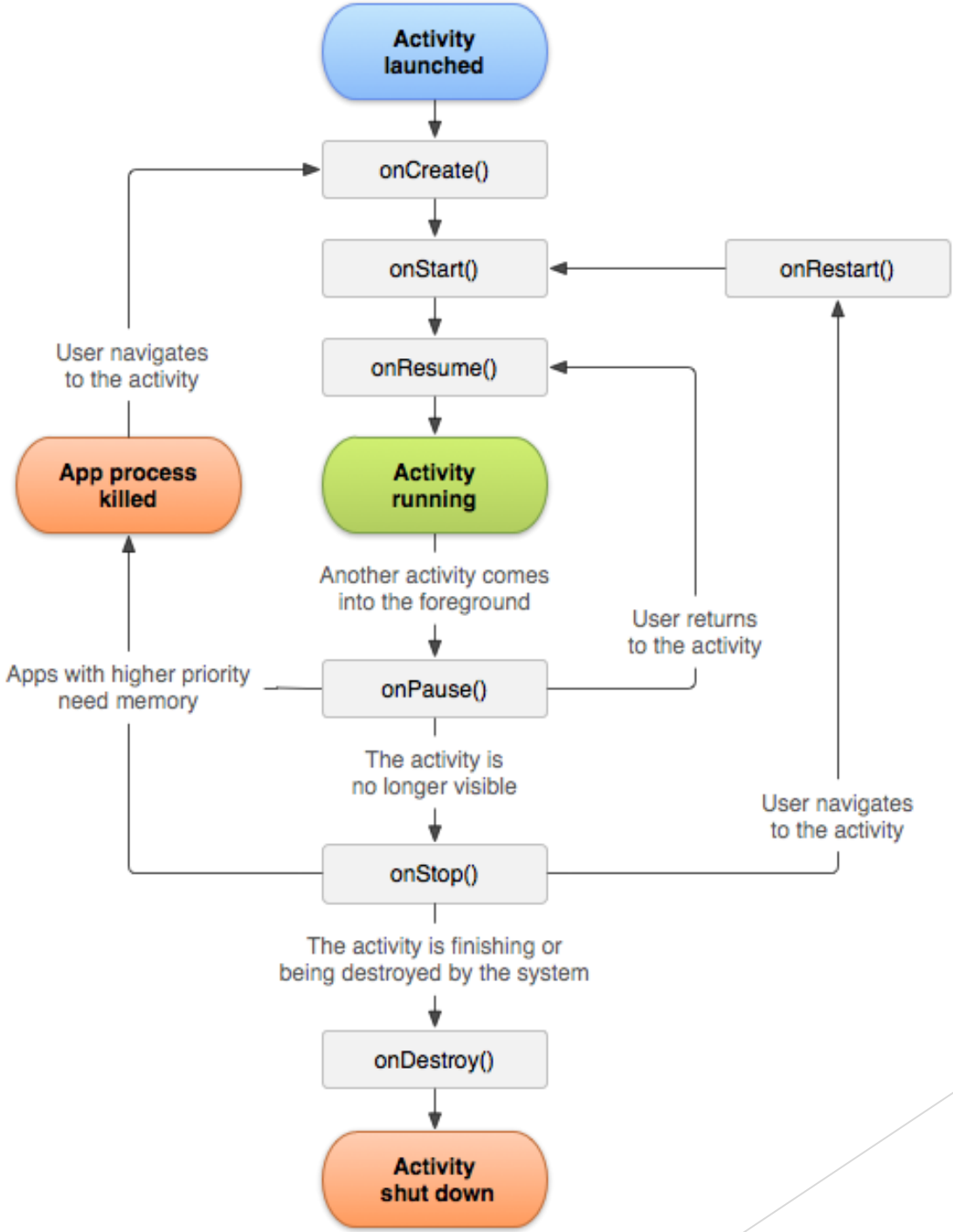
1. **Lifecycle**: Activities have a well-defined lifecycle consisting of several callback methods, such as onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(). Developers can override these methods to perform specific actions at different points in the activity's lifecycle.
2. **UI Components**: Activities typically contain UI components such as buttons, text fields, images, and more. These UI components are defined in XML layout files and can be accessed and manipulated programmatically within the activity code.
3. **Intent and Navigation**: Activities are typically started by sending an "intent" from another activity or component. Intents are objects used to request an action from another component within the Android system, such as starting a new activity, broadcasting an event, or starting a service.
4. **Back Stack**: Activities are organized in a "back stack" maintained by the Android system. When a new activity is started, it's pushed onto the back stack, and the previous activity is paused but remains in the stack. Users can navigate through the stack by pressing the back button, which pops activities off the stack.
5. **Activity States**: Activities can be in various states, such as "running," "paused," "stopped," or "destroyed," depending on their position in the activity lifecycle and user interactions.
6. **Activity Communication**: Activities can communicate with each other using intents, which can carry data between activities. Additionally, activities can share data using shared preferences, files, databases, or other mechanisms.

Activity-lifecycle

- Activities in the system are managed as activity stacks. When a new activity is started, it is usually placed on the top of the current stack and becomes the running activity -- the previous activity always remains below it in the stack and will not come to the foreground again until the new activity exits.
- To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(). The system invokes each of these callbacks as the activity enters a new state.

An activity has essentially four states:

- If an activity is in the foreground of the screen (at the highest position of the topmost stack), it is *active* or *running*. This is usually the activity that the user is currently interacting with.
- If an activity has lost focus but is still presented to the user, it is *visible*. It is possible if a new non-full-sized or transparent activity has focus on top of your activity, another activity has higher position in multi-window mode, or the activity itself is not focusable in current windowing mode. Such activity is completely alive (it maintains all state and member information and remains attached to the window manager).
- If an activity is completely obscured by another activity, it is *stopped* or *hidden*. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden, and it will often be killed by the system when memory is needed elsewhere.
- The system can drop the activity from memory by either asking it to finish, or simply killing its process, making it *destroyed*. When it is displayed again to the user, it must be completely restarted and restored to its previous state.


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

Sr. No	Callback & Description
1	**onCreate()** This is the first callback and called when the activity is first created.
2	**onStart()** This callback is called when the activity becomes visible to the user.
3	**onResume()** This is called when the user starts interacting with the application.
4	**onPause()** The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	**onStop()** This callback is called when the activity is no longer visible.
6	**onDestroy()** This callback is called before the activity is destroyed by the system.
7	**onRestart()** This callback is called when the activity restarts after stopping it.

Basic Empty Activity

```
package com.example.myapplication;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override

    protected void onCreate(Bundle savedInstanceState)


    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="My Program"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Fragment

In Android, the fragment is the part of Activity which represents a portion of User Interface(UI) on the screen.

A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events.

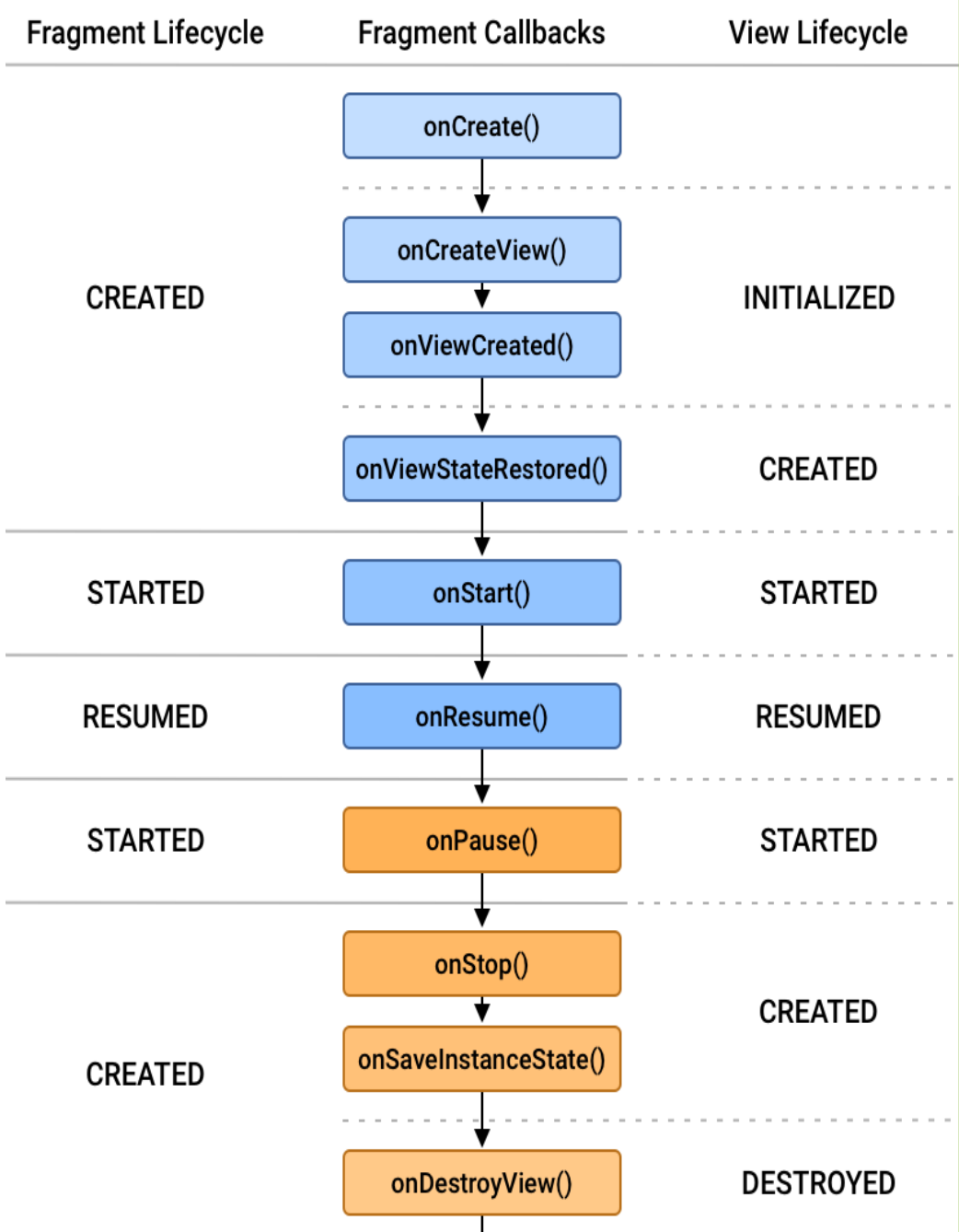
It is the modular section of the android activity that is very helpful in creating UI designs that are flexible in nature and auto-adjustable based on the device screen size.

Modularity

Fragments introduce modularity and reusability into your activity's UI by letting you divide the UI into discrete chunks.

Fragment lifecycle

No.	Method	Description
1)	onAttach()	it is called only once when it is attached with activity.
2)	onCreate( )	It is used to initialize the fragment.
3)	onCreateView( )	creates and returns view hierarchy.
4)	onActivityCreated( )	It is invoked after the completion of onCreate() method.
5)	onViewStateRestored( )	It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6)	onStart( )	makes the fragment visible.
7)	onResume( )	makes the fragment interactive.
8)	onPause( )	is called when fragment is no longer interactive.
9)	onStop( )	is called when fragment is no longer visible.
10)	onDestroyView( )	allows the fragment to clean up resources.
11)	onDestroy( )	allows the fragment to do final clean up of fragment state.



The needs fragments:

1. **Adaptability and Responsiveness:** Fragments enable developers to create responsive and adaptable user interfaces that can adjust to different screen sizes, resolutions, and orientations. This is crucial for providing a consistent user experience across a wide range of devices.
2. **Code Reusability:** Fragments promote code reusability by encapsulating specific UI components and functionalities. They can be reused across multiple activities within an application, reducing redundancy and improving maintainability.
3. **UI Composition:** Fragments facilitate the composition of complex user interfaces by allowing developers to break down the UI into smaller, manageable components. This modular approach makes it easier to design, develop, and maintain sophisticated UIs.
4. **Parallel Execution:** Fragments can run in parallel within the same activity, allowing for improved performance and responsiveness. This is particularly useful when multiple tasks or features need to be handled simultaneously, such as downloading data while displaying UI elements.
5. **Support for Tablets and Large Screens:** Fragments are especially useful for designing applications that target tablets and devices with larger screens. They enable developers to create multi-pane layouts and take advantage of the additional screen real estate available on such devices.
6. **Separation of Concerns:** Fragments promote the separation of concerns by allowing developers to separate different aspects of an application's UI and functionality into distinct fragments. This makes it easier to manage and maintain the codebase, as each fragment can focus on a specific feature or task.
7. **Fragment Lifecycle Management:** Fragments have their own lifecycle, which allows developers to manage their state and behavior independently of the hosting activity. This gives developers greater control over the lifecycle of UI components and helps ensure a smooth user experience.

Intents

What: Intents are like messengers in Android development. They describe an action an app wants to perform, and any data needed for that action.

Why: They enable communication between different parts of your app (activities, services, etc.) and even other apps on the device. This fosters flexibility and modularity.

Types:

Explicit: Specifies the exact component (activity, service) to target. Example: startActivity(new Intent(MainActivity.this, SecondActivity.class));

Implicit: Specifies an action and data, letting the system choose the best app to handle it. Example: startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse("https://www.example.com")))

Layouts

Layouts define the visual structure of your app's screens. They tell the Android system how to arrange different UI components (views) on the screen.

Types:

Linear: Views arranged horizontally or vertically, in a single row or column.

Relative: Views positioned relative to each other or the parent layout.

Tabular: Similar to a table, with rows and columns of views.

Frame: One child view occupies the entire layout area.

Absolute: Views positioned with exact pixel coordinates (rarely used, less flexible).

Views:

Views are the basic building blocks of your app's user interface. They represent interactive elements like buttons, text fields, images, etc.

Examples:

Button: A clickable button to trigger actions.

TextView: Displays text, can be single-line or multi-line.

EditText: Allows users to input text.

RadioButton: One of a group of mutually exclusive choices.

ImageView: Displays an image.

Toast: A temporary pop-up message.

Adapter: Provides data to views in lists and grids.

Spinner: A drop-down menu for selecting a single item from a list.

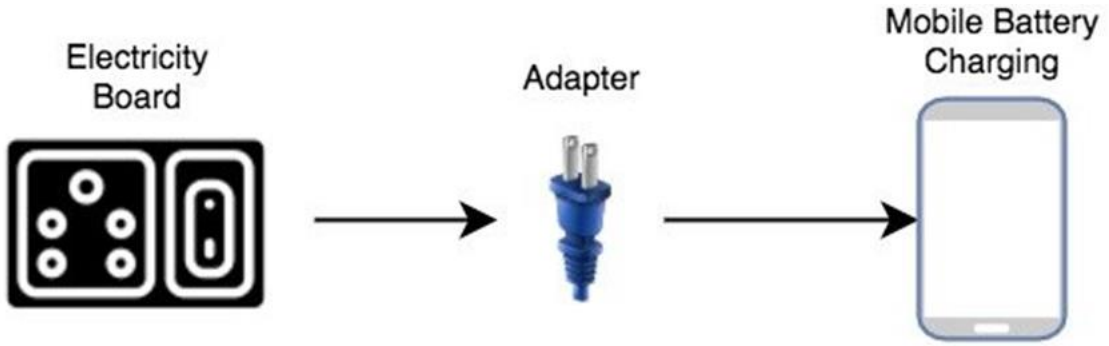
ListView: A vertically scrolling list of items.

GridView: A grid-like arrangement of items.

Adaptor

An adapter acts like a bridge between a data source and the user interface.

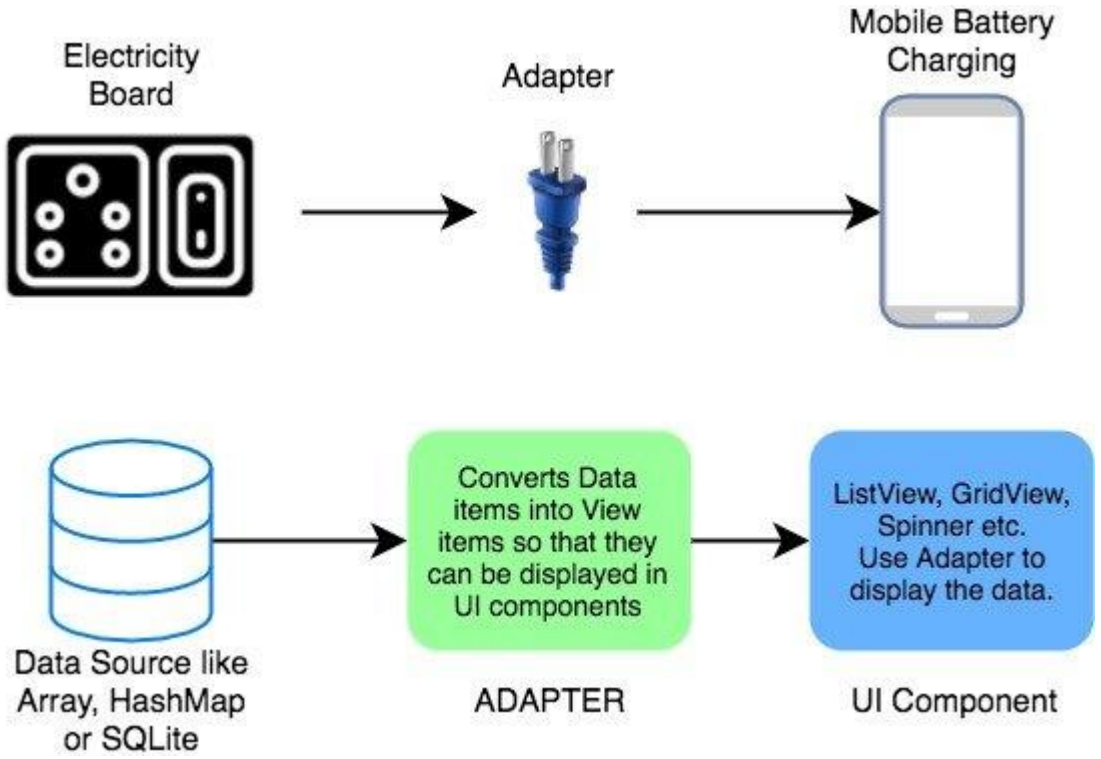
It reads data from various data sources, coverts it into View objects and provide it to the linked Adapter view to create UI components.



It holds the data and send the data to an Adapter view then view can takes the data from the adapter view and shows the data on different views like as ListView, GridView, Spinner etc. For more customization in Views we uses the base adapter or custom adapters.

Adapter:

- In Android, an adapter is a bridge between an AdapterView (such as ListView or RecyclerView) and the underlying data source.
- The main purpose of an adapter is to take the data from a data source (like an array, list, or database) and present it in a format that can be easily consumed and displayed by the UI components.
- There are different types of adapters, such as BaseAdapter, ArrayAdapter, CursorAdapter, and custom adapters that developers can create to meet specific needs.
- Adapters typically provide methods to determine the number of items in the data set (getCount()), retrieve a specific item (getItem()), and create a view for each item (getView()).


Electricity Board
Adapter
Mobile Battery Charging
Converts Data items into View items so that they can be displayed in UI components
ListView, GridView, Spinner etc. Use Adapter to display the data.
Data Source like Array, HashMap or SQLite
ADAPTER
UI Component

There are the some commonly used Adapter in Android used to fill the data in the UI components.

1. **BaseAdapter – It is parent adapter for all other adapters**
2. **ArrayAdapter – It is used whenever we have a list of single items which is backed by an array**
3. **Custom ArrayAdapter – It is used whenever we need to display a custom list**
4. **SimpleAdapter – It is an easy adapter to map static data to views defined in your XML file**
5. **Custom SimpleAdapter – It is used whenever we need to display a customized list and needed to access the child items of the list or grid**

AdapterView:

- An AdapterView is a UI component in Android that is capable of displaying a collection of data items, usually obtained through an adapter.

- Examples of AdapterView include ListView, GridView, Spinner, and RecyclerView.
- AdapterView is responsible for managing the layout and display of the items provided by the adapter. It relies on the adapter to provide data and views to be shown.
- When an item in an AdapterView is clicked, selected, or otherwise interacted with, the AdapterView communicates with the associated adapter to handle the underlying data.

In summary, the adapter acts as a bridge between the data source and the UI component (AdapterView), while the AdapterView is the UI component responsible for displaying the data items provided by the adapter. They work together to facilitate the presentation of data in a user interface.

Custom Adapter code which extends the BaseAdapter in that

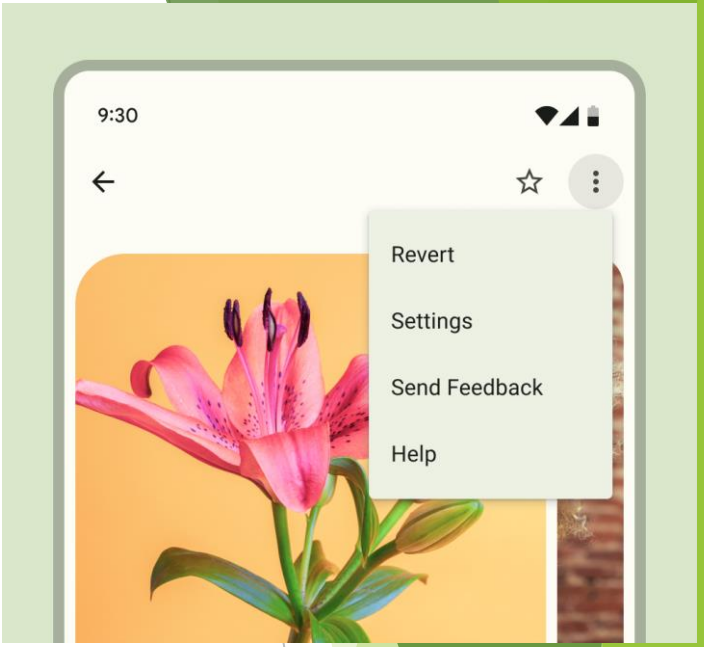
```
public class CustomAdapter extends BaseAdapter {
@Override
public int getCount() {
return 0;
}
@Override
public Object getItem(int i) {
return null;
}
@Override
public long getItemId(int i) {
return 0;
}
@Override
public View getView(int i, View view, ViewGroup viewGroup) {
return null;
}
```

Android Menu

In Android, menus are a fundamental part of the user interface, providing a way to present actions or options to users. There are three main types of menus in Android: Option Menu, Context Menu, and Popup Menu.

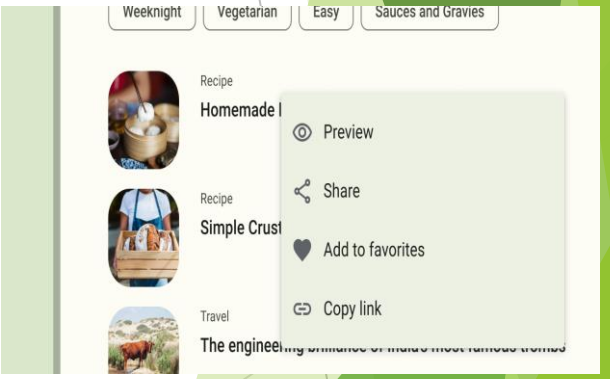
Option Menu:

Description: The Option Menu, also known as the Overflow Menu, is typically displayed as three vertical dots in the top-right corner of the screen. It contains a list of actions or options related to the current activity.



Context Menu:

Description: Context Menus are associated with specific views or elements on the screen. They are triggered by a long press on the view, providing context-specific actions.



Popup Menu:

Description: Popup Menus are used to display a list of actions anchored to a specific view. They are typically shown in response to a user's interaction, such as a click.

Events: Event Handling, and Listeners.

In Android development, events are user actions or occurrences that happen during the execution of an application, such as button clicks, screen touches, or sensor inputs.

Event Handling:

refers to the process of capturing, processing, and responding to events triggered by user interactions or system events. In Android, event handling is implemented through the use of event listeners. When an event occurs, the associated listener is notified, and the appropriate callback method is executed to handle the event.

1. **Event Sources:**
   - **User Interface Components: Buttons, EditText, CheckBox, etc.**
   - **System Events: Lifecycle events, sensor events, etc.**
2. **Event Types:**
   - **Click Events: User taps, clicks, or touches a UI element.**
   - **Text Change Events: User inputs text into an EditText.**
   - **Selection Events: User selects an item from a list or spinner.**
   - **Gesture Events: User gestures like swipes or pinches.**

Event handling in Android is often achieved through the use of listeners.

1. **Listeners**:
    1. **Listeners** are interfaces or classes that contain callback methods, which are invoked when a specific event occurs. They act as bridges between the event source (such as a button or a touch gesture) and the application logic.
    2. Android provides various types of listeners to handle different types of events. Some common listeners include:
        1. **OnClickListener**: Handles click events, such as button clicks.
        2. **OnLongClickListener**: Handles long-click events.
        3. **OnTouchListener**: Handles touch events, allowing for more granular control over touch interactions.
        4. **TextWatcher**: Monitors changes to the text in an EditText view.
        5. **SeekBar.OnSeekBarChangeListener**: Listens for changes in a SeekBar's progress.
        6. **DatePickerDialog.OnDateSetListener**: Listens for the date selection in a DatePickerDialog.

Event Handling Example:

Let's take an example of handling a button click event using an OnClickListener:

```
Button myButton = findViewById(R.id.myButton);
myButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Code to be executed when the button is clicked
        Toast.makeText(MainActivity.this, “HI MCA!", Toast.LENGTH_SHORT).show();
    }
});
```

In this example, a button with the ID myButton is identified, and an OnClickListener is set on it. The onClick method contains the code that will be executed when the button is clicked.

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

    <!-- Your other layout elements go here -->

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

    <!-- Your other layout elements go here -->

</RelativeLayout>
```

Hello MCA ()

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

    <TextView
        android:id="@+id/messageTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello MCA!"
        android:textSize="24sp"
        android:layout_centerInParent="true" />

</RelativeLayout>
```

MainActivity.java

```
package com.example.hellomca;
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Get reference to TextView
        TextView  messageTextView = findViewById(R.id.messageTextView);

        // Set the message
        messageTextView.setText("Hello MCA!");
    }
}
```

Intent

In Android, an Intent is a fundamental component that facilitates communication between different parts of an application or between different applications.

Here are key aspects of Intent:

Action:

An Intent typically carries information about the operation to be performed, known as the "action." Examples of actions include opening a new activity, sending a broadcast, or starting a service.

Data:

An Intent can also carry data along with it. This data might be a URI specifying the location of a resource or extra information to be used by the target component.

Target Component:

An Intent can be explicit or implicit.

Explicit Intent

Specifies the exact component (activity, service, broadcast receiver) to be invoked.

```
Intent explicitIntent = new Intent(CurrentActivity.this, TargetActivity.class);
startActivity(explicitIntent);
```

```
// MainActivity.java
import android.content.Intent;
public class MainActivity extends AppCompatActivity {
    @Override
public void onClick(View view) {
                // Create an explicit intent to open SecondActivity
                Intent explicitIntent = new Intent(MainActivity.this,
SecondActivity.class);
                startActivity(explicitIntent);
            }
        });
    }
}
```

SecondActivity.java

```
// SecondActivity.java


import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
public class SecondActivity extends AppCompatActivity {


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

XML

```
<!-- activity_main.xml -->

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/navigateButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Navigate to SecondActivity"
        android:layout_centerInParent="true"/>
</RelativeLayout>
```

activity_second.xml

```
<!-- activity_second.xml -->

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity">

    <!-- Content for SecondActivity goes here -->

</RelativeLayout>
```

Implicit Intent Example:

```
Intent implicitIntent = new Intent(Intent.ACTION_VIEW, Uri.parse("https://www.example.com"));
startActivity(implicitIntent);
```

```
// MainActivity.java
import android.content.Intent;
import android.net.Uri;
public class MainActivity extends AppCompatActivity {
    @Override
public void onClick(View view) {
                // Create an implicit intent to view the specified URL
                Intent implicitIntent = new Intent(Intent.ACTION_VIEW,
Uri.parse("https://www.example.com"));
                startActivity(implicitIntent);
            }
        });
    }
}
```

```
<!-- activity_main.xml -->

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/openBrowserButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Open Browser to Example.com"
        android:layout_centerInParent="true"/>
</RelativeLayout>
```