MOBILE APPLICATION DEVELOPMENT

UNIT-1

Mobile App: A mobile application, commonly referred to as a mobile app, is a software application designed to run on mobile device such as smartphones and tablets. These applications are developed specifically for mobile platform and are typically downloaded and installed from app stores.

Example: Instagram is most popular mobile application.

Mobile application development: Mobile application development is the process of creating software applications that run on mobile devices like smartphones and tablets.

Who is the father of Android:

The father of android is considered to be Andy Rubin. He co-founded android Inc. in 2003, which was later acquired by google in 2025

Web App: A web app or Web application is a software program that run on webserver and accessed through a web browser over the internet.

Brief History of Mobile Technologies:

Mobile Technologies have consistently progressed, driven by innovation and the increasing needs of users.

- **Early Beginnings:** The journey of mobile technology began in the early 20th Century with the invention of radio communication. The first real step towards mobile communication came with the development of mobile radio telephones in the 1940s. It was primarily used in vehicles with limited coverage and functionality.

- **First Generation(1G):** The introduction of the first-generation mobile networks in the 1980s marked the true beginning of the mobile telephony.

  1G networks were analog and allowed for voice communication through mobile phones.

  The launch of the Motorola DynaTAC 800X in 1983, the first commercially available mobile phone, was a landmark event despite its large size and limited battery life.

- **Second Generation(2G):** The 1990s saw the advent of the 2G mobile networks, which were digital and offered significantly improvements in voice quality and capacity.

  2G network introduced the global system for mobile communications and it became the standard for mobile communication worldwide.

  In this 2G era SMS (Short Message Service) was introduced by enabling text messages between mobile devices.

- **Third Generation(3G):** The early 2000s brought the 3G mobile networks enable faster data transmission and improved internet access on the mobile devices.

  3G networks supported multimedia messaging, video calls, and mobile internet browsing. It has shown the way for the development of more advanced mobile applications and services.

  Technologies like universal Mobile Telecommunication System and CDMA2000 expanded mobile capabilities beyond voice calls.

- **Fourth Generation(4G):** The introduction of 4G mobile networks in the late 2000s revolutionized mobile technology with high-speed internet access to enable seamless streaming of the high-definition video, faster downloads, and enhanced mobile gaming experience

  This period also witnessed the rise of smartphones, integrating powerful processors, high-resolution display, and with versatile tools for communication, entertainment, and productivity.

- **Fifth Generation(5G) and Beyond:** 5G represents the latest advancement in mobile technology, promising ultra-fast speeds, low latency, and massive connectivity to support emerging technologies like internet of things (IOT). Augmented reality (AR), and autonomous vehicles.

Mobile technology: Mobile technology is a type of technology in which a user utilizes a mobile phone to perform communications-related tasks, such as communicating with friends, relatives, and others. It is used to send data from one system to another. Portable two-way communications systems, computing devices, and accompanying networking equipment make up mobile technology.

Different mobile technologies:

Cellular Networks: Cellular networks are the foundation of modern mobile communication. It enabled us to make phone calls, send text messages and access the internet from our mobile devices.

Cellular networks operate through radio networks distributed via cell towers, which allows mobile devices to automatically switch frequencies to their nearest geographical tower without interruption.

WiFi: The next type of wireless mobile technology is Wi-Fi, which is highly popular. It allows devices to connect to the Internet wirelessly, allowing users to access online resources without needing a physical connection. WiFi technology uses radio waves to transmit data between devices and a wireless router, enabling seamless communication and internet access.

5G: The latest generation of communication, 5G, has brought with it a plethora of exciting benefits. With faster speeds, reduced latency, and more reliable connectivity, it has revolutionized how we communicate. We can now enjoy lightning-fast data transmission, seamless connectivity for multiple devices simultaneously, and enhanced coverage, ensuring a more reliable connection.

4G: 4G networking is the fourth generation of mobile technology, providing faster data speeds and improved network performance compared to its predecessor, 3G. With 4G, users can experience download speeds of up to 100 megabits per second (Mbps), making activities such as streaming high-definition videos and downloading large files much quicker and more efficient.

Bluetooth: Rather than connect devices to the internet, Bluetooth networks connect devices to other devices via short-wavelength radio waves. With Bluetooth technology, users can quickly pair devices such as headsets and speakers with desktops, laptops, and phones.

SMS: SMS stands for Short Message Service; it is a text messaging service that allows the exchange of short text messages between mobile devices. SMS messages typically can have a maximum length of 160 characters and can be sent and received on various mobile networks.

MMS: MMS stands for Multimedia Messaging Service, it is a standard way to send multimedia content such as images, videos, audio files, and contact cards between mobile devices using a cellular network. MMS is an extension of the Short Message Service (SMS), which is used for sending text messages.

Mobile Operating Systems:

Mobile operating systems are the software platforms that manage the operations of mobile devices such as smartphones, tablets, and smartwatch.

Mobile operating systems serve as the backbone of smartphones by providing the essential software framework that enables devices to function, run applications and connect to services.

These OS provide the basic functions necessary for the device to operate and for users to interact with the device and its applications.

Artificial Intelligence in Mobile:

AI integrated into mobile devices in various forms, from virtual assistants to machine learning algorithms to enhance the functionality and efficiency of mobile applications.

1. Virtual Assistant: Virtual assistant are AI-powered applications that perform tasks and provide information based on voice commands. These assistants use natural language processing and machine learning to understand and responds to user queries, offering a hands-free way to interact with mobile device.

   Example: Siri, Google Assistant.

2. Machine Learning: Machine learning involves implementing algorithms that allow machine apps to learn from data and improve their performance over time. By analyzing user behavior and data patterns, machine learning models can make predictions, automate tasks, and enhance app functionalities.

   Example: Predictive Text: mobile keyboards like Gboard and swiftkey.

Key Mobile Application Services: Mobile Application Services are essential components that support the functionality, performance and user experience of mobile apps. These services provide the necessary infrastructure, tools and features to build, deploy and manage mobile applications efficiently.

Some of the Mobile application services:

- **Push Notifications:** Push notifications are essential for engaging users and delivering timely information. They allow apps to send messages to users even when the app is not actively in use.

- **In-App Messaging:** In app messaging enables users to communicate within the app to enhance engagement and interaction among users.

  Example: Messaging App's like WhatsApp and Messenger provide in app messaging services.

- **Analytics and Tracking:** Analytics services helps developers track user behavior, app usage, and performance metrices to optimize the app's performance and user experience.

  Example: Google Analytics for Firebase is a powerful tool that track user engagement, retention, demographics and in-app events.

- **User Authentication:** User authentication services verify user identities to provide secure access to app features and data.

  Example: Google Sign-In and Facebook Login are popular authentication services.

- **Payment Processing:** Payment processing services enable secure in-app purchases, subscriptions, and financial transactions.

  Example: Apple pay and Google pay widely used payment processing services.

- **Cloud Storage:** Cloud storage services enable apps to store data on remote servers to facilitate data synchronization and backup across multiple devices.

  Example: Google Drive and iCloud Offers cloud Services.

- **Location Services:** Location Services empower apps to provide location-based functionalities such as navigation, geotagging and location-based notifications.

  Example: Apps like Google Map and Uber utilize location services to offers navigation.

- **Social Media Integration:** Social media integration services enable apps to connect with social platforms, allowing users to share content and interact with their social networks seamlessly.

  Example: Social Media Apps like Instagram, Facebook etc.

- **Voice Recognition:** Voice Recognition services allow apps to perform tasks and provide information based on user voice commands.

  Example: Siri and Gemini.

- **Augmented Reality Integration:** AR Integration services enable apps to overlay digital content and interactive experiences onto the real world, enhancing user engagement and interactivity.

  Example: Apps like Pokemon Go leverage and Amazon AR View
  use AR Technology

Android: Android is an open-source operating system developed by google, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is a Linux-based operating system, it helps handles essential tasks like memory and CPU management.

Key Features of Android:

- **Operating system**: Android operating system is a software platform that manages a device's hardware and software resources, providing an interface for users to interact with their device.

- **Based on Linux**: It is built on a modified version of the Linux kernel, that handles essential tasks like memory and CPU management.

- **Open-source**: The core of the Android operating system is open-source, though many devices include proprietary components from Google.

- **Customizable User Interface:** Android allows user to customize their home screens, widgets, and themes to personalize their experience.

- **Multi-Device Support:** Android powers a variety of devices beyond smartphones and tablets, including smartwatches, smart TV's, car infotainment systems and home appliances.

- **Multi-Tasking and Split-Screen Mode:** Android supports multitasking and split-screen mode, allowing users to run view two apps simultaneously.

- **Advanced Connectivity options:** Android supports various connectivity options such as NFC, Bluetooth, Wi-Fi direct, and USB OTG.

- **Messaging:** Android Supports both SMS and MMS for text and multimedia communication.

- **Google Service Integration:** Android devices are tightly integrated with google services such as Gmail, Google Maps, Google Drive, and Google Assistant.

- **Storge:** Android uses SQLite, a lightweight relational database for local data storage. This allows applications to store and manage complex data structures efficiently on the device.

Mobile Application Development using Android:

Mobile application development using android refers to the process of creating software applications that run on mobile devices by using Linux operating system.

Key aspects of Android application development include:

1. **Programming Languages:** Java and Kotlin has traditionally been the primary language for android app development.

2. **Development Tools:** Android studio is the official Integrated Development Environment for android app development. It provides comprehensive set of tools for coding, debugging, testing and deploying android applications.

3. **Android SDK:** The Android Software Development Kit (SDK) includes libraries, tools, and API's necessary for building Android Applications.

4. **User Interface Design:** Android app developers make sure the app looks nice and is easy to use. They follow Google's design rules (called Material Design) so that buttons, colors, and layouts feel familiar and comfortable for users.

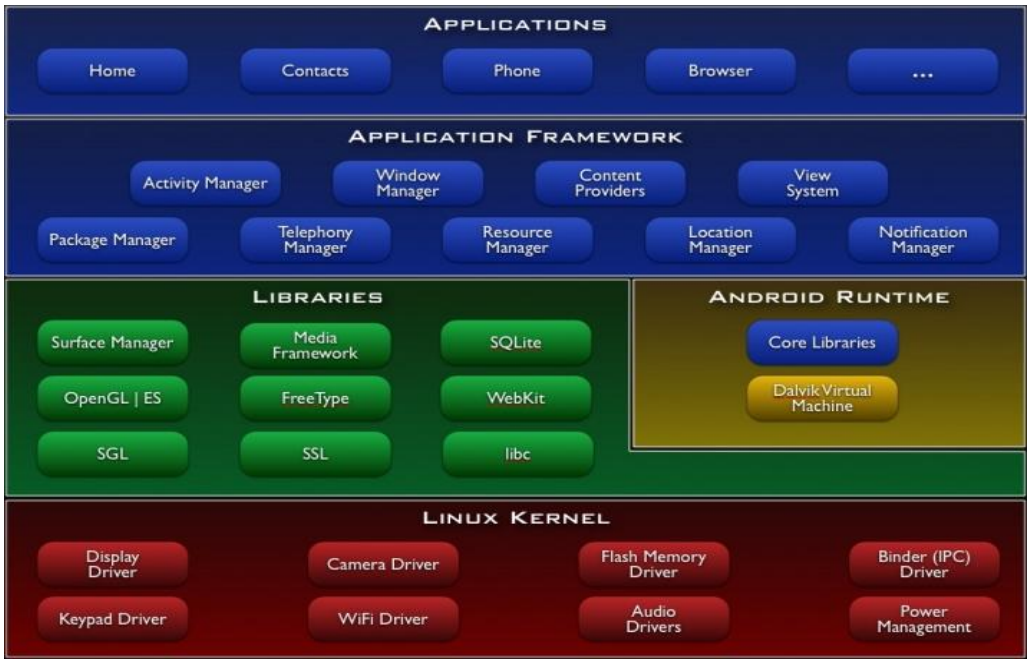
5. **Testing and Debugging:** Testing is a critical phase in android app development to ensure the application functions correctly across different devices and screen sizes. Developers use tools like android debug bridge (ADB), android Emulator, and third-party testing framework.

6. **Publishing:** Once the app is developed and tested, developers can publish it on the Google Play Store or other app distribution platforms.

7. **Continuous Updates and Maintenance:** Android developers need to regularly update the applications to add new features, fix bugs, enhance performance, and ensure compatibility with the latest android versions.

Architecture of Android: 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**.**

Components of Android Architecture:



The Android operating System is structured into five sections across four main layers:

- **Linux Kernel:** At the base of the Android OS is the Linux Kernel, which serves as the foundation for the system. This layer includes low-level device drivers that facilitate communication between the hardware components of an Android device and the software.

- **Libraries:** The libraries layers contain code that provides essential functionalities of the android OS.

- **Android Runtime:** Android Runtime environment is one of the most important parts 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.

- **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:

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

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.

The Android Application Components: An Android application is composed of several key components that works together to provide the desired functionality.

Android components can be categorized into primary components (Core Components) and Secondary Components (Non-Core Components) based on their fundamental roles.

Primary Components: Core Components are crucial for the basic functionality and structure of an android application. They handle the main processes, user interactions, and essential background tasks that make the app operational.

- **Activities**

Activities are said to be the presentation layer of our applications. The UI of our application is built around one or more extensions of the Activity class. By using Fragments and Views, activities set the layout and display the output and also respond to the user's actions. An activity is implemented as a subclass of class Activity.

Example:

```
class MainActivity extends AppCompatActivity() {

}
```

- **Services:**

Services are like invisible workers of our app that means it handles tasks that need to continue even when the app is not in the foreground. These components run at the backend, updating your data sources and Activities, triggering Notification, and also broadcast Intents. They also perform some tasks when applications are not active.

A service can be used as a subclass of class Service:

```
Example: class ServiceName extends Service() {
        }
```

- **Broadcast Receivers:**

Broadcast Receivers respond to system-wide broadcast announcements. They are used to listen for and respond to broadcast messages from other apps or the system. Broadcast Receivers enables applications to notified of events like battery status charges, network connectivity changes, or custom broadcasts sent by other applications.

- **Content Providers:** Content Providers manage a shared set of app data. They allow apps to securely share data with other apps or access data from other apps. Content providers encapsulate data storage and retrieval mechanisms, providing a standard interface for querying and updating data.

Secondary Components: The Secondary Components provide additional features and optimizations.

- **Fragments:** Fragments represent a behaviour of user interface in an activity. Fragments are modular sections of an activity with their own lifecycle and UI. They can be dynamically added, removed or replaced within an activity.

  Example: Imagine a Fragment as a separate section within the PhonePe apps Payment screen. It's like having a specific area on the screen that shows payment details and options.

- **Intents:** Intent are messaging objects used to request actions from app components. Apps use Intent to communicate and trigger actions within the app. They help to coordinating different parts of app to work together seamlessly.

- **Views and ViewGroup:** Views are the basic building blocks for user interface components, and ViewGroups are container that hold multiple views and define their layout structure. Views includes elements like buttons, text fields, and images, while ViewGroups include layouts like LinearLayout and RelativeLayout, which arrange the views.

- **Adapters:** Adapters act as a bridge between UI components and data sources to facilitate the display of data in UI components. Adapters are like messengers that fetch data from sources like database and show it in UI elements such as lists.

- **Notifications:** Notifications are used to alert users about events or updates even when the app is not in the foreground. They provide a way to keep users informed about important information.

- **Loaders:** Loaders in Android are components that load data asynchronously in the background, helping manage data across configuration changes like screen rotations.

- **SharedPreference:** SharedPreferences in Android is a lightweight mechanism to store and retrieve key-value pairs of primitive data types. It's commonly used for storing user settings, login states, or simple app preferences.

- **Widgets:** In Android, Widgets are miniature application views that can be embedded in other applications (like the home screen) and receive periodic updates. They offer users quick access to app functionality or information without opening the full app.

- **Intent Filters:** Intent Filters in Android define how your app can respond to different types of intents especially implicit ones. They allow your app to declare its ability to handle specific actions, data types, or categories, making it discoverable by other apps and the system.

Exploring The Development Environment:

Exploring the android development Environment involves understanding the tools and resources available for creating android applications. The android development environment requires several components or tools or resources to develop and test android applications efficiently.

Android SDK: The Android SDK is a set of tools, libraries and resources provided by Google to develop android applications. It serves as the foundation for android app development and provides essential components and APIs for building feature-rich apps.

Android Studio: Android Studio is the official Integrated Development Environment (IDE) for Android app development. It is designed to streamline the entire app development process. It serves as a comprehensive platform for designing, coding, testing, and debugging android applications. It provides a range of tools and features to facilities app development.

Android Emulators: Android Emulators are vertical devices that simulate the behaviour of real Android devices for testing and debugging apps. They allow developers to test and debug applications on various device configuration without needing physical devices. They provide a virtual testing environment to ensure app compatibility, performance and functionality across different device Configurations.

ADB (Android Debug Bridge): The Android Debug Bridge is a versatile command-line tool that serves as a bridge between a development machine and an android emulator. It offers a range of debugging and troubleshooting functionalities to help in the development and testing of android applications.

Gradle: The Gradle Build System is a powerful build automation tool integrated into Android Studio to streamline that management of project dependencies, build configurations, and tasks efficiently.

Gradle is the official build automation tool for Android development. It manages how your app is compiled, packaged, and deployed, and it allows you to define dependencies, build variants, and custom build logic.

Obtaining the Required Tools: The primary components required are the Java Development Kit (JDK) and Android Studio. These tools provide all the necessary resources for Android development as android studio includes the Android SDK, Android Emulators, ADB, and Gradle.

- **Java Development Kit (JDK):** The mainly JDK is used to Compiling Java code. The JDK is crucial for developing Java applications, including android apps. It includes the Java Runtime Environment, an interpreter, a complier, and other necessary for Java Development.

- **Android Studio:** Android Studio is the official Integrated Development Environment (IDE) for Android app development. It is designed to streamline the entire app development process. It serves as a comprehensive platform for designing, coding, testing, and debugging android applications. It provides a range of tools and features to facilities app development.

Launching Your First Android Applications:

Step-by-Step: Launching Your First Android Applications:

Step1: **Install Android Studio**

- Download from developer.android.com/studio
- Follow the setup wizard to install the Android SDK, emulator, and tools.

Step2: Create a New Project

- Open Android Studio → **File > New > New Project**
- Choose a template (e.g., Empty Activity)
- Name your app and choose: Demoapp
- Language: Java.
- Minimum SDK: Choose based on your target devices.

Step3: Understand the Project Structure

- MainActivity.java: Entry point of your app
- activity_main.xml: UI layout for the main screen
- AndroidManifest.xml: App configuration and permissions

Explain All the Folder available in android Project

Step4: Write Your First Code

In MainActivity.Java, you'll see something like:

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

In activity_main.xml, you can add a simple TextView tag:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Hello World!"
   android:textSize="30dp"
   app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintStart_toStartOf="parent"
   app:layout_constraintTop_toTopOf="parent" />
```

Step5: Run the App

- Click the Run button in Android Studio
- Choose a device:
    - Emulator: Use AVD Manager to create a virtual device
    - Physical Device: Enable Developer Mode and USB Debugging

Step6: **Output of Action**

- The app will compile, install, and launch on the selected device.
- we should see your "Hello, World!" message on screen.

Exploring The IDE-Debugging Your Application in Android Studio:

We can debug android applications in android studio to identify and resolve issues efficiently and to ensure the smooth functioning of their apps before releasing them to users.

The steps in debugging the Android application are shown below:

1. **Step Breakpoints:** One of the most powerful debugging features in android studio is the ability to set breakpoints in code. Breakpoints allow the app to pause at specific points to inspect the state of the app.

2. **Start an Emulator:** Before starting debugging process, developers need start an android Emulator from Android Studio's device manger. The connected emulator will be used to run and debug the application.

3. **Run App in Debug Mode:** Developers can run the android application in debug mode by clicking on the "Debug" icon in top toolbar of android studio or shortcut Key is (Shift+F9). The emulator deploys the application for debugging.

4. **Navigate Through the App:** Once the application is running in debug mode on the emulator, developers can interact with the app to trigger the execution of the code. The application will pause at the set breakpoints to analyze the state of variables and understand the program flow.

5. **Inspect Variable and Evaluate Expression:** Developer can utilize the debug panel in android studio to inspect variables values, evaluate expressions, and monitor the application's state during runtime. This feature helps in understanding how data changes as the program executes.

6. **Step Through the Code:** Android studio provides various debugging controls such as Step Over, Step Into, and Step Out to navigate through the code line by line. By stepping through the code, developers can trace the execution path and identify any anomalies.
   Step Over (F8): Moves to the next line of code.
   Step Into (F7): Enters the method on the current line.
   Step Out (Shift+F8): Exits the current method and returns to the caller.

7. **Evaluate Logs and Exceptions:** Developers should check the logcat tab in android studio while debugging to view system logs, error messages and exceptions thrown during the application's execution.

   To open Logcat tool window (View>Tool Windows> Logcat) to see log messages from the running app.

8. **User Profiling Tools:** For advanced debugging and optimization, developers can leverage android profiler in android studio. This tool provides insights into app performance, CPU usage, memory allocation, and network activity, helping developers optimize their application for better efficiency and user experience.

   Use the android Profiler tool (View>ToolWindow>Profiler).

9. **Modifying Code and Re-Test:** Based on the insight gained from debugging, developer can make necessary code modifications to address issues. After making changes, developers should remove breakpoints, re-run the application in debug mode, and verify that the fixes have resolved the identified problem.

Publishing Android Application

To publish an Android application, we need to prepare our app, set up our developer account, and follow a structured release process on the Google Play Console.

Step1: **Prepare Your App for Release**

- Remove debug code: Disable logging and debugging tools.
- Minify and optimize: Use ProGuard or R8 to shrink and obfuscate code.
- Versioning: Update versionCode and versionName in build.gradle.

  Every Android app has two version values you set in the app/build.gradle file:

  versionCode: A number that increases every time you release a new update.

  Example: 1, 2, 3 …

  versionName: A text string that users see as the app version.

  Example: "1.0", "1.1", "2.0" …
- Sign the app: Generate a signed Android App Bundle (AAB) using your keystore.

Step2: **Create a Google Play Developer Account**

- Visit the Google Play Console.
- Pay a one-time registration fee of **$25 USD**.
- Fill in your developer profile (name, email, contact details).

Step3: **Create a New App in Play Console**

- Click **“**Create App”.
- Enter:
  - App name
  - Default language
  - App or game type
  - Free or paid
- Accept the developer policies and click Create.

Step4: **Prepare Store Listing**

Fill in:

- App title and short/long description
- Screenshots (min 2 for phones, 1 for tablets/TVs if applicable)
- App icon (512x512 px)
- Feature graphic (1024x500 px)
- Category (e.g., Education, Tools)
- Contact details (email, website, phone)
- Privacy policy URL

Write the Policy: Use a template or generator (e.g., FreePrivacyPolicy.com).

Step5: **Upload the App Bundle**

- Navigate to “Release > Production”.
- Click “Create new release”.
- Upload your .aab file.
- Add release notes (what’s new in this version).

Step6: **Set Up App Content**

Complete mandatory declarations:

- Privacy policy
- App access
- Ads (if your app contains ads)
- Content rating questionnaire
- Target audience and age
- News app declaration (if applicable)
- Data safety section

Step7: **Set Pricing and Distribution**

- Choose whether your app is Free or Paid.
- Select countries where your app will be available.
- Opt-in for devices and programs (e.g., Android TV, Wear OS).

Step8: **Review and Publish**

- Review all sections for completeness.
- Fix any warnings or errors.
- Click **"**Publish" to submit your app for review.

Activity In Android: An Activity in Android is a single screen with user interface where users can interact, it represents a window in an application that displays the UI elements. Each screen in an android app is typically implemented as an activity.

Creating an Activity:

Step1: Creating New Java class for an Activity

In Android Studio, right-click on app's package name in the Project pane, select "New" then "Activity" and choose the activity template to use like Empty Activity, Empty Views Activity etc.

Example: package com.example.helloapp;

```
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
public class Main_Activity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello);
    }
}
```

Line By Line Explination:

- **package com.example.helloapp**

Defines the package name of your app.

- **import android.os.Bundle**

Imports the Bundle class. A Bundle is like a backpack that carries data between Android components.

- **import androidx.appcompat.app.AppCompatActivity**

Imports the AppCompatActivity class from AndroidX. This is the base class for modern Android Activities.

- **public class HelloActivity extends AppCompatActivity {**

Declares a new Activity class named Main_Activity. The extends means it inherits from AppCompatActivity, gaining all its behavior.

- **protected void onCreate(Bundle savedInstanceState) {**

Overrides the onCreate() method the first lifecycle method called when the Activity is created.savedInstanceState holds previous state data if the Activity is being recreated.

- **super.onCreate(savedInstanceState)**

Calls the parent class's onCreate() to ensure the Activity is properly initialized. Always required when overriding lifecycle methods.

- **setContentView(R.layout.activity_hello)**

Sets the UI layout for this Activity.

- R.layout.activity_hello refers to the XML file activity_hello.xml in the res/layout folder.

Step2: Create an XML Layout: main_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">
    <TextView
        android:text="Hello, Android!"
        android:textSize="24sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Line By Line Explination:

- **<?xml version="1.0" encoding="utf-8"?>**

Declares the XML version and character encoding.

- **<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"**

Starts a LinearLayout, which arranges child views in a straight line (vertically or horizontally). xmlns:android defines the Android XML namespace, required for Android to understand layout attributes.

- **android:layout_width="match_parent"**

Sets the layout's **width** to match the parent container (e.g., full screen width).

- **android:layout_height="match_parent"**

Sets the layout's height to match the parent container (e.g., full screen height).

- **android:orientation="vertical"**

Arranges child views top to bottom.

- **android:gravity="center"**

Aligns child views **to the center** of the layout.

- **android:padding="16dp">**

Adds space inside the layout around its edges.

- **<TextView>**

Declares a TextView, which displays text on the screen.

- **android:text="Hello, Android!"**

Sets the text content to be shown.

- **android:textSize="24sp"**

Sets the font size to 24 scale-independent pixels.

- **android:layout_width="wrap_content"**

Width adjusts to fit the text like wrapping paper around a gift.

- **android:layout_height="wrap_content" />**

Height adjusts to fit the text no extra space

Activity Life Cycle: The lifecycle of an android activity represents the series of stages and activity goes through from to creation to its destruction.

Activity Lifecycle Stages:

1. **onCreate():** The onCreate Called when the activity is first created. Initialize UI and resources here.

Code: protected void onCreate(Bundle saveInstanceStace){
Super.onCreate(savedInstanceStage);
setContentView(R.layout.activity_main);
}

2. **onStart():** This Method Called when the activity becomes visible to the user.

Code: Protected void onStart(){
Super.onstart();

}

3. **onResume():** This Method is called when the activity starts interacting with user.

Code: protected void onResume(){
Super.onResume();
}

4. **onPause():** This Method is called when the system is about to start resuming another activity.

Code: protected void onPause(){
Super.onPause();
}

5. **onStop():** This method is called when the activity is no longer visible to the user.

Code: protected void onStop(){
super. onStop();
}

6. **onDestroy():** This method is called before an activity is destroyed.

```
Code: protected void onDestroy (){
      Super.onDestroy();
}
```

7. **onRestart():** This method is called when activity has been stopped and is restarting again.

```
Code: protected void onRestart(){
       super.onRestart();
}
```

Applying Styles and Thems to an Activity:

Style: A Style in android is a collection of properties that specify the look and format for a View. It can be used to share common attributes across multiple elements to ensure a consistent look and feel throughout the application.

In other words, it is Like a dress code for individual UI elements (e.g., TextView, Button).

Example of a Style:

1. Define the Style in res/values/styles.xml

```
<style name="MyTextStyle" parent="TextAppearance.AppCompat.Large">
   <item name="android:textColor">#4CAF50</item>
   <item name="android:textStyle">italic</item>
   <item name="android:background">#E8F5E9</item>
</style>
```

Apply the Style in activity_main.xml:

```
<TextView
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   style="@style/MyTextStyle"
   android:text="Welcome to Android!" />
```

Themes: A Theme in Android is a style applied to an entire Activity rather than an individual View. In Other words, it's like a school uniform applies a consistent look across the entire Activity or app.

Example of a Themes:

Define the Theme in res/values/themes.xml:

```
<style name="MyAppTheme"
parent="Theme.MaterialComponents.DayNight.DarkActionBar">
    <item name="colorPrimary">#3F51B5</item>
    <item name="colorPrimaryVariant">#303F9F</item>
    <item name="colorOnPrimary">#FFFFFF</item>
    <item name="android:windowBackground">#F5F5F5</item>
</style>
```

Apply the Theme in AndroidManifest.xml:

```
<application
    android:theme="@style/MyAppTheme">
    <activity android:name=".MainActivity" />
</application>
```

Displaying a Dialog Window Using an Activity

A Dialog is a small window that prompts the user to make a decision or enter information. It doesn't fill the screen and is often used for alerts, confirmations, or input.

Types of Dialogs in Android:

AlertDialog: AlertDialog is a versatile dialog that can show a title, message, and buttons. It is used for prompting user actions, displaying information or gathering input.

Example: Code (Inside an Define a Button in activity_main.xml)

Step 1: Define a Button in activity_main.xml

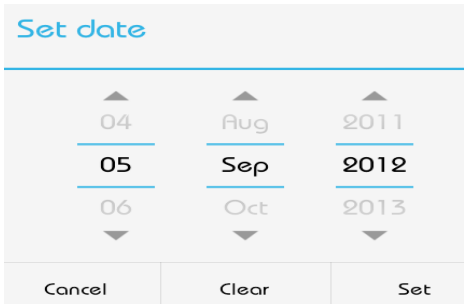
```
<Button
    android:id="@+id/btnShowDialog"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show Dialog"
    android:layout_gravity="center"
    android:layout_marginTop="50dp" />
```

Step 2: Code (Inside an Activity like MainActivity.java)

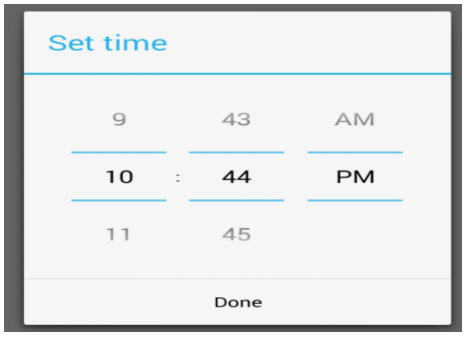
```
Button btnShowDialog = findViewById(R.id.btnShowDialog);

// Set click listener
btnShowDialog.setOnClickListener(v -> {
    AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
    builder.setTitle("Hello!");
    builder.setMessage("This AlertDialog appears when you click the button.");


    builder.setPositiveButton("OK", (dialog, which) -> {
        Toast.makeText(MainActivity.this, "You clicked OK",
Toast.LENGTH_SHORT).show();
        dialog.dismiss();
    });
    builder.setNegativeButton("Cancel", (dialog, which) -> {
        Toast.makeText(MainActivity.this, "You clicked Cancel",
Toast.LENGTH_SHORT).show();
        dialog.dismiss();
    });
```

```
    builder.show();
});
```

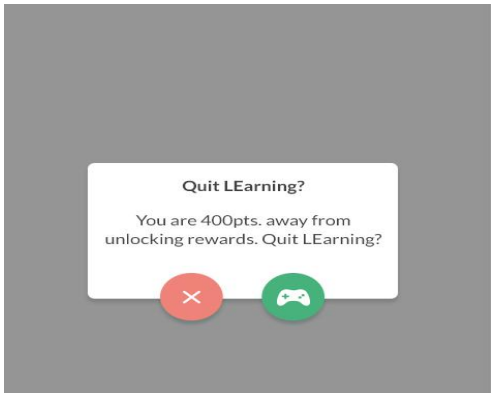
DatePickerDialog: A dialog that allows users to select a date. It provides a user-friendly interface for date selection, showing a calendar view where users can pick a date.



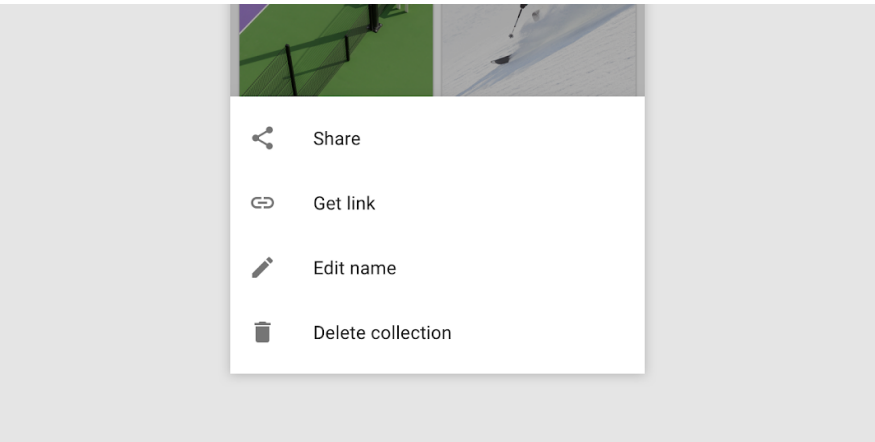
TimePickerDialog: A dialog that allows users to select a time. It can be configured to use either a 24-hours format or an AM/PM format, depending on the applications requirements.



Custom Dialogs: Fully customized dialogs using custom layouts. Custom dialogs provide complete flexibility, allowing developers to design and implement any user interface within the dialog.



BottomSheetDialog: A BottomSheetDialog is a dialog that slides up from the bottom of the screen. BottomSheetDialog provides a modern way to present actions to users, often used for actions related to the current content.



Intent in Android: An Intent in Android is a messaging object that can be used to request an action from another app component, such as activities, services and broadcast receivers. It serves as a bridge to enable communication between different components within an application or between different applications on the device.

Intents can be used to:

- Start an activity.
- Start a service.
- Deliver a broadcast.

Features and Functionalities of Intent:

Intent-Component Communication: Intents facilitate communication between different components of an android application such as activities, services, and broadcast receivers.

Action Specification: Intent can specify the action to be performed such as sending message, opening a web page, or sharing content.

Services Invocation: Intents can Trigger background services to perform tasks independently of the user interface such as downloading files or processing data.

Broadcasting Events: Broadcast intents allow components to send system-wide messages or notifications in case of inter-app communication and event handling.

Creating Intent: Creating an Intent in Android is the process of defining a message that requests an action from another component such as launching an activity, starting a service, or sending a broadcast.

How to Create an Intent Object

Syntax: Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);

Intent: It is a class in the Android framework used to define an action to be performed , such as starting another activity.

CurrentActivity.this: This parameter specifies the current context, usually the activity from which the intent is being created. It provides the necessary context for the intent to start the new activity.

TargetActivity.class: This parameter specifies the class of the activity that you want to navigate to it tells the android system which activity to lunch.

Types of Intent:

1. **Explicit Intent:** Directly specifies the target component (activity, service, etc.) to handle the intent. **Usage**: Used within your own app when you know the exact class to launch.

Example: Intent intent = new Intent(this, SecondActivity.class);

startActivity(intent);

2. **Implicit Intent:** Declares a general action to be performed, allowing the system to find the appropriate component (even from other apps). Usage: Used when you want the system to choose the best app to handle the action.

Example: Intent intent = new Intent(Intent.ACTION_VIEW);

intent.setData(Uri.parse("https://www.example.com"));

startActivity(intent);

An **Intent** is a message object in Android that tells the system you want to do something.

Here, we saying: “I want to VIEW some data.”

ACTION_VIEW is a predefined constant that means open something for the user to look at.

Uri.parse(...) converts the string "https://www.example.com" into a Uri object.

A Uri is Android’s way of representing resource addresses (like web links, file paths, or content provider references).

setData(...) attaches this Uri to the intent, so now the intent knows what you want to view.

This tells Android: “Find an app that can handle this intent and start it.”

Since the intent is ACTION_VIEW with a web URL, Android looks for apps that can open web pages (like Chrome, Firefox, or the default browser).

Intent Class Methods:

Some of the methods of the Intent Class their description and example:

- **Intent():** Constructor to create a new empty Intent.

Example: Intent intent = new Intent();

- **Intent (Context package Context, Class<?> cls):** Constructor to create an intent specifying the context and target activity class.

Example: Intent intent = new Intent(MainActivity.this , SecondActivity.class)

- **putExtra(String name, String value):** Adds extended data to the intent.

Example: intent.putExtra(“USER_NAME”, “TOM”);

- **getStringExtra(String name):** Retrieves extended data from the intent.

Example: String username = intent.getStringExtra(“USER_NAME”);

- **setAction(String action):** Sets a specified action for the intent.

Example: Intent.setAction(Intent.ACTION_VIEW);

- **getAction():** Retrieves the action associated with the Intent.

Example: String action = intent.getAction ();

- **setData(Uri data):** Sets the data URI for the Intent.

Example: intent .setData(Uri.parse("https://www.google.com"));

- **setFlags(int flags):** sets special flags controlling how the intent is handled.

Example: intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

- **getFlags():** Retrieves the special flags associated with the intent.

Example: int flags = intent.getFlags();

Intent Class Attributes: These attributes help in defining the behaviour and purpose of the intent.

- **Intent.ACTION_VIEW:** Intent.ACTION_VIEW in Android is a built-in constant that tells the system user want to *view* some data.

Example: intent.setAction(Intent.ACTION_VIEW);

- **Intent.ACTION_SEND:** This Attribute is used to send data from one activity to another.

Example: intent.setAction(Intent.ACTION_SEND);

- **Intent.ACTION_MAIN:** Indicates the main entry point of an application.

Example: intent.setAction(Intent.ACTION_MAIN);

- **Intent.FLAG_ACTIVITY_NEW_TASK:** This flag tells Android: "Start this activity in a new task, separate from the current one.

Example: intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);

- **Intent.FLAG_ACTIVITY_CLEAR_TOP:** It Tell If the activity we are starting already exists in the current task's back stack, then destroy all activities that are on top of it, and bring that existing activity to the front instead of creating a new one.

Example: intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);

Intent Filters: Intent filter are component of an Android application manifest file that specify the types of intents an activity, service, or broadcast receiver can respond to. Intent filters consist of actions, categories, data, and MIME type that help Android system identify the appropriate component to handle a specific intent.

Components of an Intent Filter

1. **Action:** The action element represents the general action to be performed by the intent. Actions are typically predefined constants in the Intent class.

Example: android.intent.action.VIEW

android.intent.action.SEND

Custom actions like com.example.MY_ACTION

2. **Category:** The category element provides additional information about the action being performed. Categories help to refine the type of component that should handle the intent.

Example:

- android.intent.category.DEFAULT → Required for implicit intents.
- android.intent.category.BROWSABLE → Allows the activity to be started by a web browser.

3. **Data:** The data element specifies the type of data that the intent is interested in. This can include a URI scheme, a MIME type, or both.

   URI Schema: Specifies the protocol for accessing the data (e.g., 'http', 'https'). This indicates the type of data source the component can handle.
   MIME Type: Specifies the format of the data the component can handle (e.g, 'image/*', 'text/plain'). This allows components to specify the exact type of data they can process.

   Example:

```
<intent-filter>
    <data android:scheme="http" android:host="www.example.com" />
</intent-filter>
```

Returning Result from an Intent: In Android, returning a result from an Intent refers to the process where one activity starts another activity and expects some data back when that second activity finishes.

Example: **main_activity.XML**

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

    <Button
        android:id="@+id/openSecondActivity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Open Second Activity" />


<TextView
        android:id="@+id/resultText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:text="Result will appear here"
    android:layout_marginTop="20dp"
    android:textSize="18sp" />

</LinearLayout>
```

Main_Activity.Java

```
public class MainActivity extends AppCompatActivity {

    private static final int REQUEST_CODE = 1;
    private TextView resultText;

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

        resultText = findViewById(R.id.resultText);
        Button button = findViewById(R.id.openSecondActivity);

        button.setOnClickListener(v -> {
            Intent intent = new Intent(MainActivity.this, SecondActivity.class);
            startActivityForResult(intent, REQUEST_CODE);
        });
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == REQUEST_CODE && resultCode == RESULT_OK) {
            String value = data.getStringExtra("name");
            resultText.setText("Received: " + value);
        }
    }
}
```

open: activity_second.XML

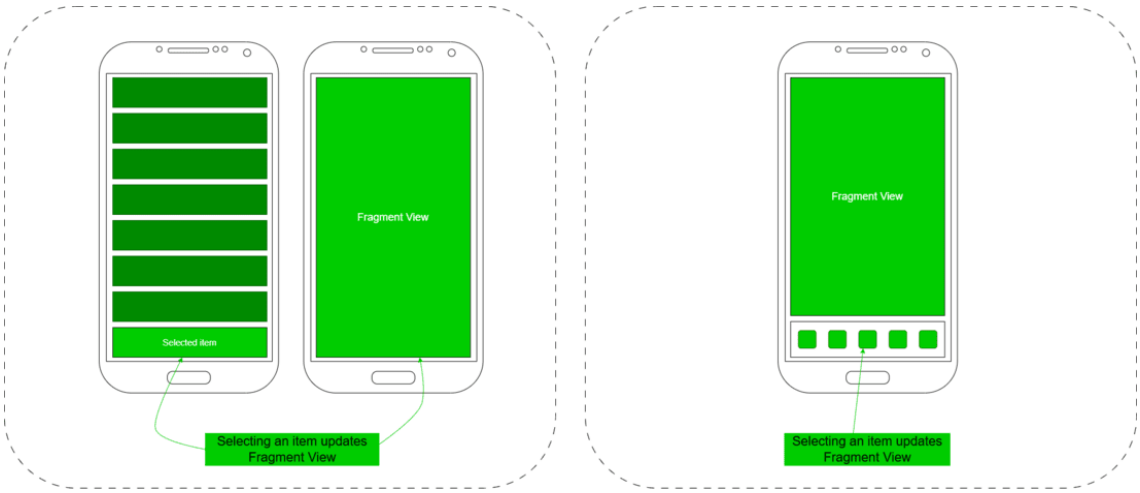
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">
    <Button
        android:id="@+id/openSecondActivity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Open Second Activity" />
    <TextView
        android:id="@+id/resultText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Result will appear here" />
</LinearLayout>
```

Open: SecondActivity

```
public class SecondActivity extends AppCompatActivity {

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

```
        Button button = findViewById(R.id.returnResult);
        button.setOnClickListener(v -> {
            Intent resultIntent = new Intent();
            resultIntent.putExtra("name", "Chandan");
            setResult(RESULT_OK, resultIntent);
            finish(); // closes SecondActivity and returns result
        });
    }
}
```

Fragment: Fragment in Android are modular and reusable UI components that represent a portion of user interface within an activity. Fragments are like mini-activities that can be embedded, within an activity.



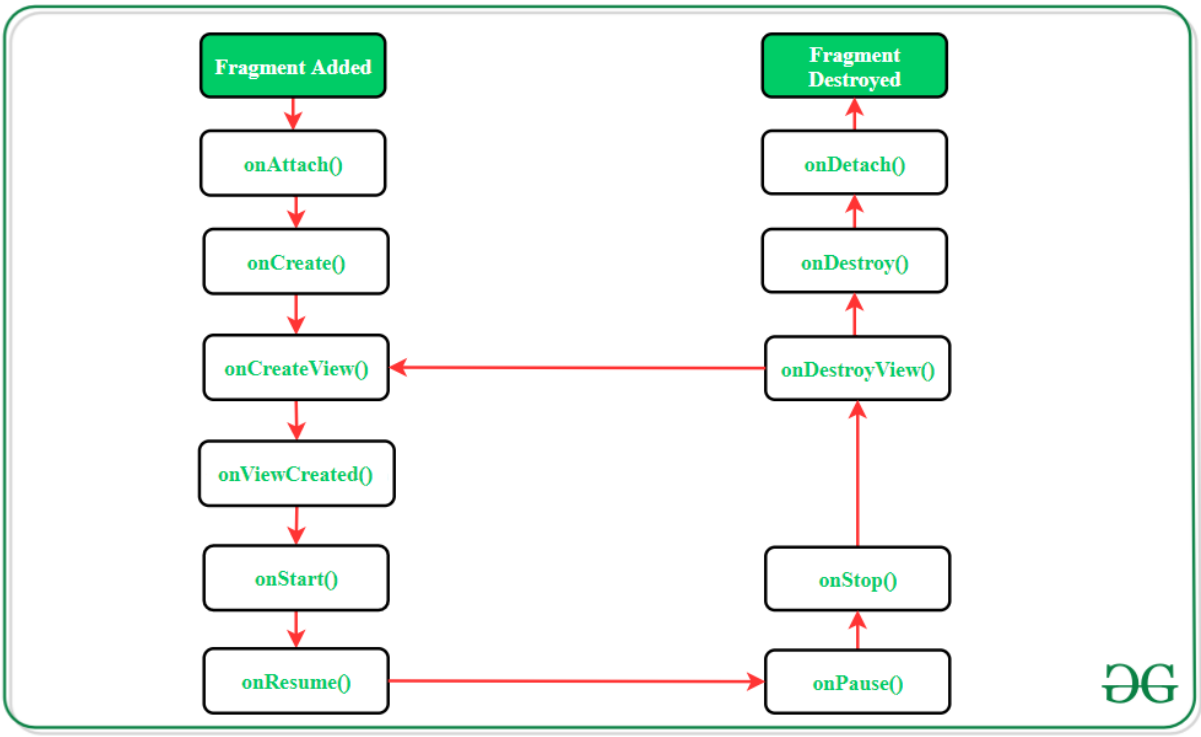
Example: How Fragments Are Used in the PhonePe App

PhonePe, like many modern Android apps, leverages fragments to build a scalable and maintainable user interface. Here's how and why:

Modular UI Components

- Each screen or section (e.g., Home, Recharge, Payments, Offers) is typically a fragment.
- This allows PhonePe to reuse UI components across different activities or tabs.

Fragment Lifecycle:



Methods of the Android Fragment

onAttach(): The very first method to be called when the fragment has been associated with the activity. This method executes only once during the lifetime of a fragment. When we attach fragment(child) to Main(parent) activity then it calls first and in simple terms that this method calls only one time.

Example:

```
@Override
public void onAttach(@NonNull Context context) {
    super.onAttach(context);
    // Initialization code here
}
```

onCreate(): This method initializes the fragment by adding all the required attributes and components.

onCreateView(): onCreateView() is a lifecycle callback method in a Fragment. The system calls it when the fragment needs to create its user interface (UI).

onViewCreated(): After the fragment's layout is created in onCreateView(), the system immediately calls onViewCreated().

This method gives you a chance to work with the views safely because the layout is already inflated and ready.

It's the best place to initialize UI components, set listeners, or bind data.

onStart(): The system invokes this method to make the fragment visible on the user's device.

onResume(): This method is called to make the visible fragment interactive.

onPause(): It indicates that the user is leaving the fragment. System calls this method to commit the changes made to the fragment.

onStop(): Method to terminate the functioning and visibility of fragment from the user's screen.

onDestroyView(): When the system decides the fragment's view is no longer needed (for example, you replace the fragment with another one, or the activity goes into a state where the fragment's UI isn't visible), it calls onDestroyView().

onDestroy(): The system calls onDestroy() when the fragment is about to be completely destroyed.

This is where you do the final clean-up of anything the fragment was using (like stopping background tasks, closing resources, or releasing memory).

After this, the fragment object is gone it won't come back unless you recreate it.

onDetach(): When the fragment is no longer connected to that activity (for example, the activity is destroyed, or you replace the fragment with another one), the system calls onDetach().

Creating and Using Fragment:

Step1: Create a Fragment Class

```
public class MyFragment extends Fragment {

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

        View view = inflater.inflate(R.layout.fragment_my, container, false);
```

```
    TextView textView = view.findViewById(R.id.fragmentText);
    textView.setText("Hello from MyFragment!");
    return view;
}
```

Explanation: Declares a new class called MyFragment.

It extends Fragment, meaning it inherits all the behavior of an Android Fragment.

This is the lifecycle method called when the fragment needs to create its UI.

- Parameters:
    - LayoutInflater inflater → Used to convert XML layout files into actual View objects.
    - ViewGroup container → The parent view that will hold the fragment's UI.
    - Bundle savedInstanceState → Holds any saved state if the fragment is being recreated.
- The method must return a View (the root of the fragment's layout).

```
}
```

Step2: Create the Fragment Layout

Path: res/layout/fragment_my.xml

```
<FrameLayout 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=".MyFragment">
    <TextView
        android:gravity="center"
        android:textStyle="bold"
        android:textSize="24sp"
        android:id="@+id/fragmentText"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
```

```
    android:text="Hello from Fragment!"/>

</FrameLayout>
```

3. **Open main_activity.xml**

Path: res/layout/activity_main.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/fragmentContainer"
   android:layout_width="match_parent"
   android:layout_height="match_parent"/>
```

4. **Open Main_Activity.java**

```
        public class MainActivity extends AppCompatActivity {
           @Override
           protected void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.activity_main);
        if (savedInstanceState == null) {
           getSupportFragmentManager().beginTransaction()
                 .replace(R.id.fragmentContainer, new MyFragment())
                 .commit();
            }

     }

}
```

Back Stack: The back stack is a stack-like data structure used by android system to keep track of the sequence of activities that user interacts with. It operates on the principle of Last In, First Out (LIFO) meaning the last activity added to the stack is the first one to be removed when the back button is pressed.

Example for step by step to Implementation of Back Stack:

Project setup with three activities

- Goal: See how Android's default back stack behaves.
- Create activities: MainActivity (A), SecondActivity (B), ThirdActivity (C).
- Navigation: A → B → C via explicit intents; Back should return C → B → A.

- **Get into the AndroidManifest.xml**

```
<!-- AndroidManifest.xml -->
<application ...>
    <activity android:name=".ThirdActivity" />
    <activity android:name=".SecondActivity" />
    <activity android:name=".MainActivity">
   <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>
```

- **Create an MainActivity.java (A)**

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnGoB = findViewById(R.id.btnGoB);
        btnGoB.setOnClickListener(v -> {
            Intent intent = new Intent(MainActivity.this, SecondActivity.class);
            startActivity(intent);
        });
    }
}
```

➢ **Create an SecondActivity.java (B)**

```
public class SecondActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
        Button btnGoC = findViewById(R.id.btnGoC);
        btnGoC.setOnClickListener(v -> {
            Intent intent = new Intent(SecondActivity.this, ThirdActivity.class);
            startActivity(intent);
        });
    }
}
```

➢ **Create an ThirdActivity.java (C)**

```
public class ThirdActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_third);
    }
}
```