Unit 1 : Introduction to Android Operating System

Android operating system is the largest installed base among various mobile platforms across the globe. Hundreds of millions of mobile devices are powered by **Android** in more than 190 countries of the world. It conquered around **71%** of the global market share by the end of 2021, and this trend is growing bigger every other day. The company named **Open Handset Alliance** developed Android for the first time that is based on the modified version of the Linux kernel and other open-source software. **Google** sponsored the project at initial stages and in the year 2005, it acquired the whole company. Android dominates the mobile OS industry because of the long list of features it provides. It's user-friendly, has huge community support, provides a greater extent of customization, and a large number of companies build Android-compatible smartphones.

What is Android Development?

Making the "**apps**" or applications that you use on your phone or tablet is basically what Android Development is.

Prerequisites to Android Development:

1. Basic Understanding of Programming

Programming is the foundation of Android development, and it's essential to understand the basics before diving into app creation.

Key concepts to focus on:

- Variables**:** Data storage containers that hold text, integers, and other values.
- Control structures**:** They are the tools that let your program make decisions and carry out actions repeatedly, such as loops and conditionals.
- Functions**:** Reusable code segments that carry out particular operations, which keep your programs efficient and well-organized.
- Error Handling**:** Techniques for handling unforeseen issues without causing the application to crash.

2. Proficiency in Java/Kotlin

Why Kotlin? In order to avoid frequent issues, Kotlin minimizes boilerplate code, and has built-in safety features like null safety. It is easy to use even for beginners and works well with Android Studio.

Note: Boilerplate code refers to sections of code that are repeated in multiple places with little to no modification.

Why Java? Java is among the most used programming languages in the world. It serves as the basis for a lot of Android apps and aids in your comprehension of the conventional Android ecosystem.

3. Understanding of OOP

Code is arranged into objects using the object-oriented programming (OOP) technique, which makes it modular, reusable, and simpler to administer. Because Android apps frequently incorporate intricate structures and functionalities, OOP is particularly crucial to the development process.

Important OOP concepts:

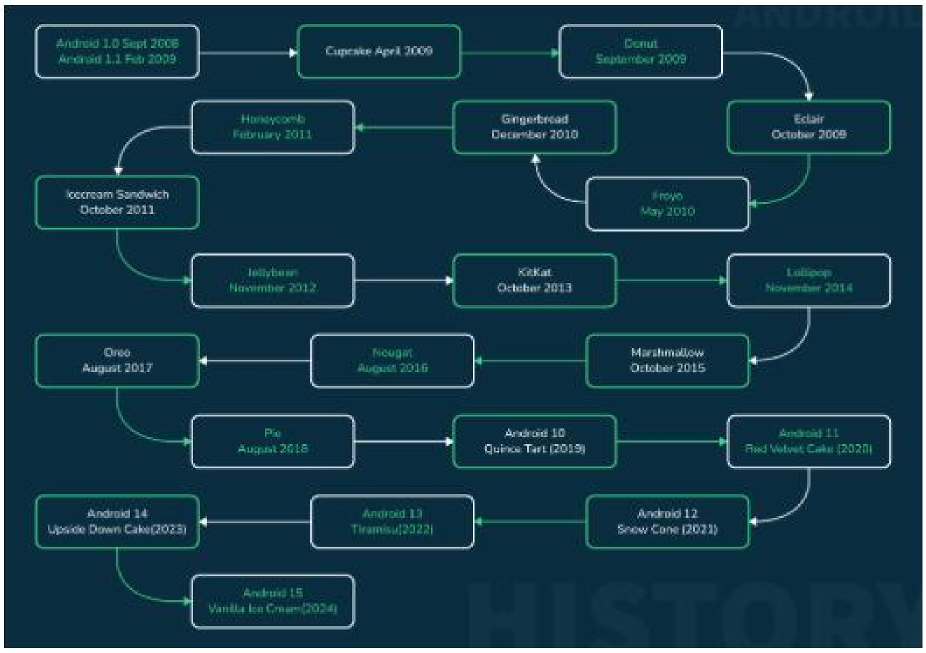
- Encapsulation: is the process of enclosing procedures and data inside objects to shield them from external intervention.
- Inheritance: Reusing code and adding new features by building new classes from preexisting ones.
- Polymorphism: Code that is flexible and dynamic is made possible by polymorphism, which permits objects to be regarded as instances of their parent class.
- Abstraction: Hiding superfluous complexity while concentrating on important details.

Advantages of Android Development

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

Android Versions:

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



What is mobile application

A mobile application, commonly refferred to as a mobile app, is a software application designed to run on mobile devices such as smartphones and tablets. These applications re developed specifically for mobile platforms and are typically downloaded and installed from app stores or marketplaces like apple app store for ios devices and googe play store for android devices.

Mobile application services:

1. **User Sign-up/Sign-in and Management:** • This service involves creating a seamless experience for users to register, sign in, and manage their accounts within the mobile app. It includes features like email-based registration, social login (such as Facebook or Twitter), and password recovery.
2. **Social Login:** • Social login allows users to sign in to your app using their existing social media credentials (e.g., Face book, Google, Twitter). It simplifies the authentication process and enhances user convenience.
3. **Analytics and User Engagement:** • Analytics services help track user behavior within the app. By analyzing data such as user interactions, session duration, and conversion rates, you can make informed decisions to improve the app's performance and engagement.
4**. Push Notifications:** • Push notifications keep users informed and engaged by sending timely updates, reminders, or personalized messages directly to their devices. • Effective push notification strategies can enhance user retention and drive app usage.
5.**Real Device Testing** • Real device testing ensures that the app works flawlessly across various devices and operating systems. • Testing on actual devices, rather than just simulators, helps identify any issues related to performance, compatibility, or usability, ensuring a smooth user experience.

Introduction to Android OS Design

Define Android: Android is an open-source operating system based on Linux, designed primarily for touchscreen mobile devices such as smartphones and tablets. It is developed by Google and provides a rich application framework.

1. Android OS Design Principles

- **Open-Source & Customizable**: Manufacturers can modify and customize Android to suit their devices.
- **Linux-Based Architecture**: Uses the **Linux kernel** for process management, security, and hardware interaction.
- **Multi-User & Multi-Tasking**: Supports multiple user profiles and background task execution.
- **Modular & Scalable**: Android is designed to run on various screen sizes, resolutions, and hardware configurations.
- **Secure & Sandboxed**: Each app runs in its **own process and memory space** to prevent security breaches.

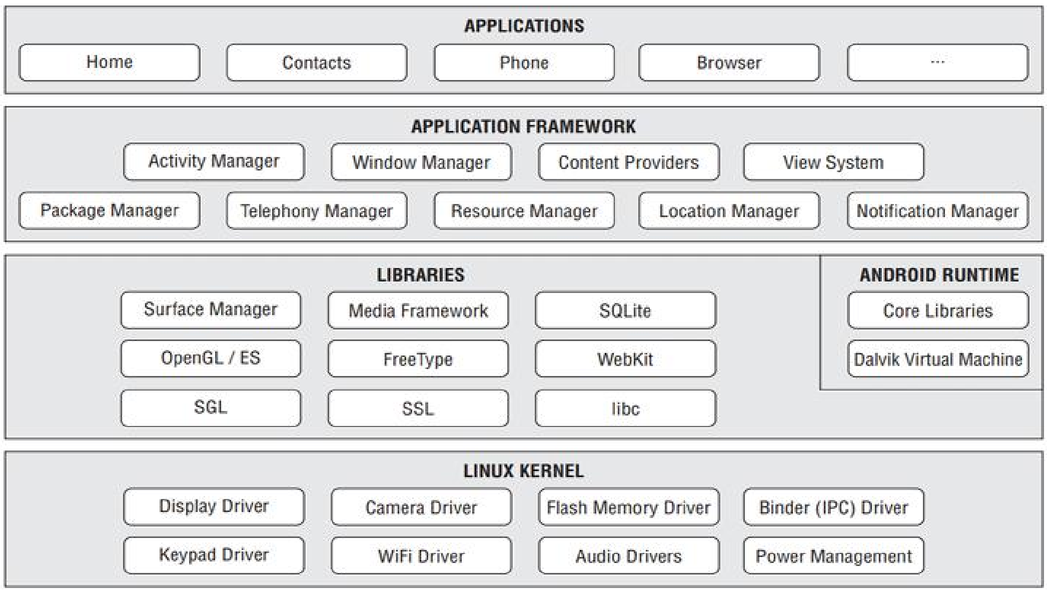
2. Android OS Architecture (Design Layers)/Android development framework:

An Android app development framework is a collection of tools that help developers create Android apps. It includes templates, components, and pre-written code. The framework helps maintain a consistent architecture across Android apps and devices.

Benefits of using an Android development framework:

1. **Speed up development**: Developers can reuse code and recycle components
2. **Maintain quality**: The framework helps ensure that apps have a consistent architecture across devices
3. **Build cross-platform apps**: Some frameworks allow developers to write code once and use it on multiple platforms (react native, flutter, xamarin).

Android follows a **layered architecture**, consisting of five key layers:



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.

Application framework

Application Framework provides several important classes which are used to create an Android application. It provides a generic abstraction for hardware access and also helps in managing the user interface with application resources.

- Includes components like:
  - **Activity Manager**: Manages app lifecycle.
  - **Notification Manager**: Handles system notifications.
  - **Resource Manager**: Manages UI elements like layouts, strings, themes.

Application runtime

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.

Note: ART (Android Runtime) is the default runtime environment for Android applications, introduced in **Android 4.4 (KitKat)** and made the default in **Android 5.0 (Lollipop)**. It replaced the older **Dalvik Virtual Machine (DVM)** to improve **performance, efficiency, and battery life**.

· When an app is installed, ART **compiles it into native machine code** using **AOT Compilation**.
· This makes app execution **faster** compared to **JIT(just in time) Compilation** in Dalvik, which compiles code **at runtime**.

Platform libraries

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

- **Media** library provides support to play and record an audio and video formats.
- **Surface manager** responsible for managing access to the display subsystem.
- **SGL** and **OpenGL** both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- **SQLite** provides database support and **FreeType** provides font support.

- **Web-Kit** This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- **SSL (Secure Sockets Layer)** is security technology to establish an encrypted link between a web server and a web browser.

Linux Kernel(Core Layer)

Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc.

- Acts as the core foundation of Android OS.
- Handles **memory management, process scheduling, power management, and hardware drivers** (camera, Bluetooth, Wi-Fi).

The features of Linux kernel are:

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

3. Key Features of Android OS

A. Open-Source & Customizable

- Android is based on **AOSP (Android Open Source Project)**, allowing manufacturers (Samsung, Xiaomi, OnePlus) to modify and add custom UI skins like **One UI, MIUI, OxygenOS**.[MIUI is a custom Android firmware developed by Xiaomi, built on top of the Android Open Source Project, and used on Xiaomi smartphones and devices, though it is being replaced by Xiaomi HyperOS.]

B. Multi-Tasking & Multi-Touch Support

- Users can run **multiple apps at the same time** (e.g., split-screen mode).
- Supports **multi-touch gestures** (pinch to zoom, swipe, tap).

C. Extensive App Ecosystem

- **Google Play Store** offers **millions of apps** across various categories (Games, Productivity, Entertainment).
- Supports **third-party app stores** (Samsung Galaxy Store, Huawei AppGallery).

D. Security & Privacy

- **Google Play Protect** scans apps for malware.
- **Sandboxing**: Apps run in **isolated environments** to prevent data leaks.
- **App Permissions**: Users can grant or revoke access to camera, location, and storage.
- **Biometric Security**: Supports **fingerprint scanning, face recognition, and iris scanning**.

E. Voice Assistance & AI Integration

- **Google Assistant** allows voice commands for hands-free control.
- AI-based features like:
    - **Adaptive Battery** (optimizes power usage).
    - **Live Captions** (real-time subtitles for videos).

F. Connectivity Features

- Supports multiple connectivity options:
    - **5G, Wi-Fi, Bluetooth, NFC (Near Field Communication), USB-C, Hotspot & Tethering**.

G. Storage & File Management

- Supports **internal storage, microSD cards, and cloud storage (Google Drive, OneDrive)**.
- File management system allows easy access and organization of files.

H. Device Compatibility & Scalability

- Runs on **various screen sizes** (smartphones, tablets, foldable devices, TVs).
- Supports **Android Auto (cars), Wear OS (smartwatches), and Android Things (IoT devices)**.

I. Background Services & Notifications

- Apps can run background services (e.g., **music players, fitness trackers**).
- Push notifications system for real-time alerts (WhatsApp messages, news updates).

What is android studio

The Integrated DevelopmentEnvironment (IDE) is a software application that provides comprehensive facilities to programmers for softwaredevelopment. AndroidStudio is the official Integrated Development Environment(IDE) for Android app development. It is designed to streaming 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 facilitate app development.

Features :

CodeEditor: Android Studio provides a powerful code editor with features like highlighting, code completion, refactoring, and code navigation to enhance productivity and code quality.
LayoutEditor: The Layout Editor enables developers to create visually appealing interfaces with drag and drop functionality, real-timepreviews, and support for different screen sizes and orientations.
SDKManager: The Android SDK Manager is a tool within Android Studio that allows developers to download, install, and manage different versions of the AndroidSDK, as well as other essential tools and components.
DeviceManager: The Device Manager is a tool within Android Studio that allows developers to create and manage virtual devices that simulate physical Android devices These virtual devices are used to test and debug applications on various configurations and Android versions without needing physicaldevices.
APKAnalyzer: Helps analyze APKsize, contents, and dependencies to optimize app performance and reduce file size
Built-inEmulator: The built-in Android Emulator allows developers to test their apps on virtual devices with various configurations, screensizes, and Android versions for comprehensive testing
DeviceTesting: Developers can test their apps on physical devices connected via USB for real-world testing scenarios, in addition to the Android Emulator, to ensure app compatibility and performance
VersionControlIntegration: Android Studio supports version control systems like Git, enabling developers to manage sourcecode, trackchanges, and collaborate with team members efficiently within the IDE
Extensive PluginEcosystem: Android Studio offers a wide range of plugins and extensions from the Plugin Marketplace to extend functionality, integrate with external tools, and customize the development environment according to specific requirements. **PerformanceProfilingTools:**Android Studio offers performance profiling tools to analyze app performance, identify bottlenecks, and optimize CPU, memory, and network usage for better user experience.

Android SDK features:

The Android SDK (Software Development Kit) provides developers with tools, libraries, and APIs to build and debug Android applications, including tools like the Android Debug Bridge (adb), build tools, and APIs for accessing platform features and Google services.

1. Android API Libraries

- Provides pre-built libraries to access Android functionalities like UI design, networking, storage, and more.

2. Android Emulator

- A virtual device that allows developers to test apps without needing a physical phone.
- Supports different screen sizes, Android versions, and hardware configurations.

3. Android Debug Bridge (ADB)

- A command-line tool for communicating with an emulator or connected device.
- Helps in installing APKs, debugging apps, and accessing logs.

4. Gradle-Based Build System

- Automates the build process, making it easier to manage dependencies.
- Helps in generating APK and AAB (Android App Bundle) files for different device configurations.

5. Android Studio (IDE Support)

- Official IDE for Android development with built-in support for coding, testing, and debugging.
- Features:
    - **Code Editor** with syntax highlighting and auto-completion
    - **Layout Editor** for designing UI visually
    - **Device Emulator** for testing on virtual devices

6. UI Components & Layouts

- Supports different UI components such as Buttons, TextViews, RecyclerView, and Fragments.
- Uses XML and Jetpack Compose for UI design.

Installing and Running Applications on Android Studio

Android Studio is the official Integrated Development Environment (IDE) for Android app development

WhatisAndroid Emulator?

Android emulators are virtual 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 configurations without needing physical devices they provide a virtual testing environmental ensure app compatibility performance and functionality across different device configurations.

What is an AVD (Android Virtual Device)?

An **Android Virtual Device (AVD)** is an emulator configuration in Android Studio that allows developers to test Android applications on different virtual devices without needing physical hardware. It simulates real Android devices with customizable hardware profiles, system images, storage, and sensors.

It simulates real Android devices with customizable hardware profiles, system images, storage, and sensors. The **AVD Manager** in Android Studio enables developers to create, configure, and manage virtual devices for testing apps across various Android versions and screen sizes. AVD supports features like GPS, cameras, network simulation, and hardware acceleration, making it a valuable tool for debugging and performance testing. It helps developers ensure their applications run smoothly across different device configurations before deployment.

Advantages of AVD

- No need for a physical device.
- Allows testing on multiple Android versions and screen sizes.
- Provides debugging and performance testing features.
- Supports various hardware configurations like different CPUs, memory sizes, and sensors.

Why Use AVD?

- Test apps on various screen sizes, resolutions, and Android versions.
- Simulate real-world scenarios like incoming calls, SMS, GPS location changes, and battery levels.
- Useful for debugging and app optimization.

AVDs are a powerful way to test Android apps across different devices and configurations. By setting up multiple virtual devices, developers can ensure their apps run smoothly on different screen sizes and Android versions.

Types of Mobile Applications

There are three main types of mobile applications:

a) Native Apps

- Developed specifically for a single platform (Android or iOS).
- Uses platform-specific languages and frameworks.
    - o **Android**: Java, Kotlin
    - o **iOS**: Swift, Objective-C

- Offers the best performance and access to device features like GPS, Camera, and Sensors.
- Examples: WhatsApp, Instagram (Android/iOS versions)

b) Web Apps

- Mobile-optimized websites that function like apps.
- Built using web technologies: **HTML, CSS, JavaScript**.
- Runs in a mobile browser and does not require installation.
- Examples: Twitter Lite, Gmail Web

c) Hybrid Apps

- A combination of native and web apps.
- Developed using frameworks like **React Native, Flutter, Ionic, Xamarin**.
- Runs on multiple platforms using a single codebase.
- Examples: Instagram, Uber

Based on Usage:

A. Social Media Apps

- Allow users to connect, share content, and communicate.
  Examples: Facebook, Instagram, TikTok, Snapchat

B. Entertainment & Media Apps

- Provide **streaming services, gaming, and digital content**.
  Examples: Netflix, YouTube, Spotify, PUBG Mobile

C. E-Commerce Apps

- Enable users to **buy/sell products** online.
  Examples: Amazon, Flipkart, eBay

D. Utility & Productivity Apps

- Help users perform daily tasks and improve efficiency.
  Examples: Google Drive, Microsoft Office, Evernote

E. Financial & Banking Apps

- Allow users to manage finances, transactions, and investments.
  Examples: Google Pay, Paytm, PhonePe, SBI YONO

F. Health & Fitness Apps

- Monitor physical activities and provide health-related features.
  Examples: Google Fit, MyFitnessPal, Fitbit

G. Educational Apps

- Provide **e-learning, courses, and skill development**.
  Examples: Udemy, Coursera, Byju's, Khan Academy

H. On-Demand Apps

- Offer **instant services** like food delivery, cab booking, or house services.
  Examples: Swiggy, Zomato, Uber, UrbanClap

Best practices in android programming:

1. Defining Mobile App Objectives

- Identify **target audience**, and clearly define objectives and goals before starting mobile application development.
- Conduct **market research and competitor analysis**, gather user preferences, and before making a final decision.
- Consider **market trends** based on that include features in the app to attract a large user base.

2.Identify Platforms (Native Or Cross-platform) For Development

- Identify advantages and limitations of Cross-platforms(Flutter, Xamarin) versus **native development.**
- Consider performance(Choose native for better performance), platform-specific requirements

3.User experience(UI) and User interface(UI) play vital roles in the success of mobile applications. Each mobile platform comes with its own UI standards such as Material design for Android and **HR guidelines in iOS.** Following these guidelines helps developers to keep end-users engaged with applications.

4. Focus on App Responsiveness

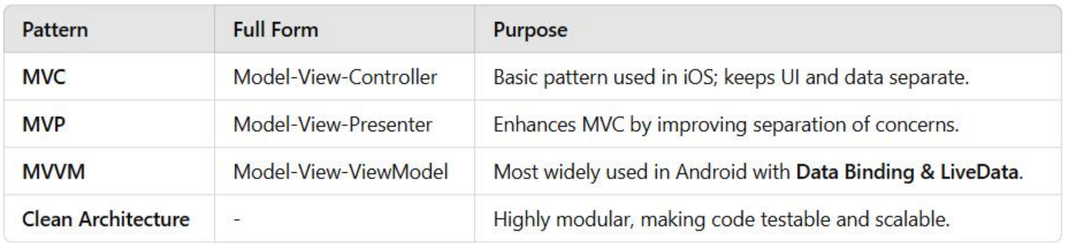
Mobile devices come in different screen sizes and resolutions specifically Android devices as large number of companies manufacture physical devices that run **Android OS** and each device comes in different screen sizes, an application must look the same on all devices with different screen sizes.

- Use flexible layouts(constraint layout for Android and Auto layout for iOS), and scalable assets to ensure consistent user experience in different device sizes.

5.Adopt The Best Codebase Architecture

Adopting efficient **codebase architecture** helps developers maintain code easily in the future mainly for large enterprise projects where multiple developers are working.

- Use MVVM, MVC, MVP, and clean architecture are commonly used architecture in Android and iOS native which helps developers to isolate UI data binding with business logic.
- Include the use of dependency Injection libraries to avoid code redundancy in applications.



Pattern	Full Form	Purpose
MVC	Model-View-Controller	Basic pattern used in iOS; keeps UI and data separate.
MVP	Model-View-Presenter	Enhances MVC by improving separation of concerns.
MVVM	Model-View-ViewModel	Most widely used in Android with **Data Binding & LiveData**.
Clean Architecture	-	Highly modular, making code testable and scalable.

MVC :

- **Model:** Manages data (Database, API, etc.).
- **View:** UI elements that display data to the user.
- **Controller:** Handles user interactions and updates the model/view.

MVP : Used in Android & iOS when better separation is needed.

- **Model:** Business logic and data handling.
- **View:** Displays UI, contains only UI-related code.
- **Presenter:** Acts as an **intermediary** between Model & View.

MVVM :

- **Model:** Handles business logic (API, Database).
- **View:** Displays UI but does NOT handle logic.
- **ViewModel:** Manages UI-related data & communicates with Model.

Removes tight coupling between UI & business logic.
Works well with **LiveData & Data Binding**.
Easier unit testing compared to MVP.

· **Write clean, readable code** with good naming conventions.

· **Conduct unit and integration tests** using JUnit and Espresso.

· **Secure sensitive data** with encryption and the Android Keystore.

· **Use version control** with Git and follow branching strategies.

· **Maintain documentation** for your code and project.

Android tools:

Android Studio

- **Primary IDE** for Android development, officially supported by Google.
- Provides an **intelligent code editor, emulator, debugging tools, and performance profilers**.
- Supports Java, Kotlin, and C++ for Android development.

2. Android SDK (Software Development Kit)

- Collection of tools and libraries needed for **building Android applications**.
- Includes **APIs, emulators, compilers, and debugging utilities**.

- **SDK Platform**: Required for targeting specific Android versions.
- **SDK Tools**: Includes emulator, adb, lint, and more.
- **Android Debug Bridge (ADB)**: A command-line tool for communicating with devices/emulators.
- **Android Emulator**: Allows testing apps without a physical device.

3. ADB (Android Debug Bridge)

- A **command-line tool** that lets developers communicate with an Android device or emulator.

4. Android Emulator

- Simulates Android devices on a PC for testing apps.
- Supports various device configurations (phones, tablets, wearables, TVs).
- Allows testing **networking, sensors, GPS, battery, and screen sizes**.

Android application components:

Android application components are the essential building blocks of an Android app. These components work together to create a fully functional Android app. Here are the main components:

1.Activities: An **Activity** represents a single screen in an Android application. It acts as the **entry point** for user interaction.

Eg: A **Login screen** is one activity.

A **Dashboard screen** is another activity.

Lifecycle:

Activities go through different **states** managed by the **Activity Lifecycle**:

1. onCreate() – Called when the activity is created.
2. onStart() – Activity is visible but not interactive.
3. onResume() – Activity is fully interactive.
4. onPause() – Activity is partially visible.
5. onStop() – Activity is no longer visible.
6. onDestroy() – Activity is destroyed.

2.Services:

A **Service** runs in the **background** to perform long-running tasks **without a user interface**.

Examples:

- **Playing music** in the background.
- **Downloading files** in the background.

Types of Services:

1. **Foreground Service** – Runs continuously (e.g., Music Player).
2. **Background Service** – Runs in the background (e.g., Syncing data).
3. **Bound Service** – Allows other components to bind to it (e.g., Bluetooth Service).

3.Broadcast Receivers

A **BroadcastReceiver** listens for system-wide or app-specific **broadcast messages**.

Examples:

- **Detecting low battery** (BATTERY_LOW).
- **Detecting network changes** (CONNECTIVITY_CHANGE).

4. Content Providers

A **ContentProvider** manages and shares data **between applications,** allowing access to databases, files, or other resources securely.

Examples:

- **Contacts App** provides access to stored contacts.
- **Media App** provides access to images & videos.

Common Content Providers:

- **Contacts**: content://contacts/people/
- **Media Store**: content://media/external/images/media/

Android Manifest File

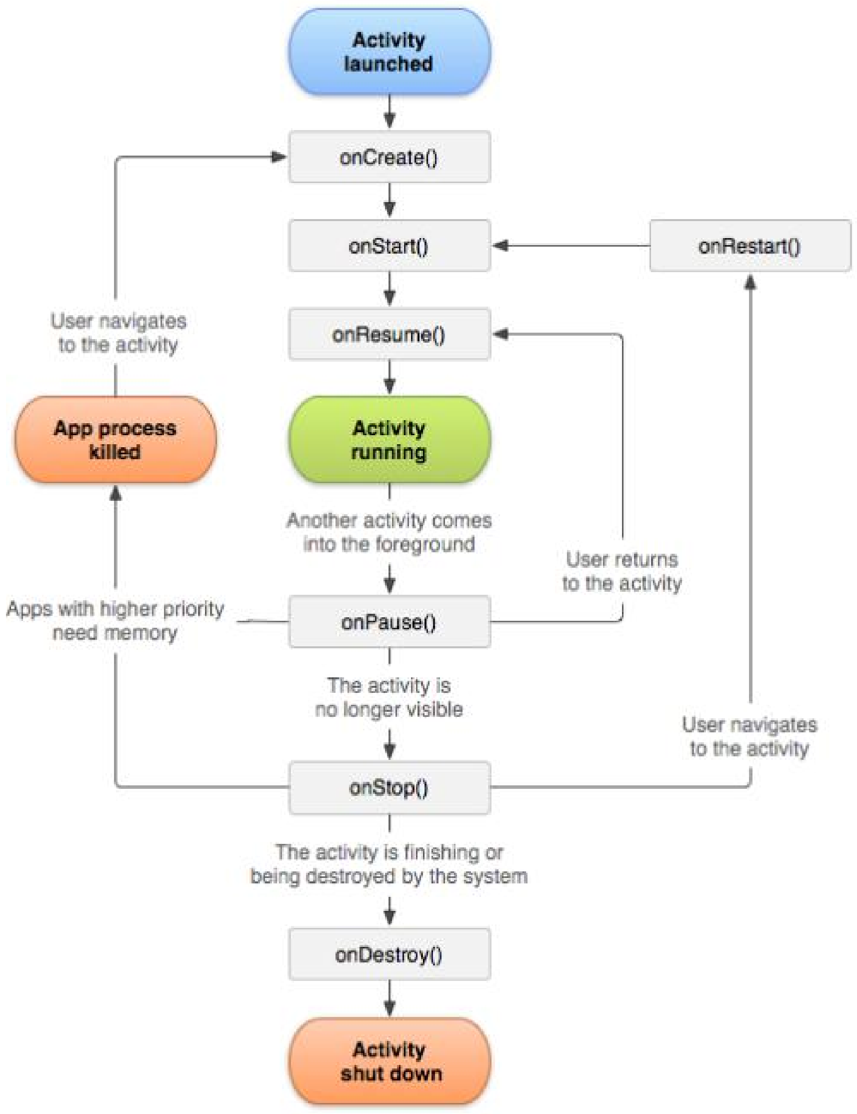
The **AndroidManifest.xml** file declares all components and permissions required by the application.

Why is the Manifest file important / features :

· **Declares Application Components** – Registers Activities, Services, Broadcast Receivers, and Content Providers.

· **Specifies App Permissions** – Requests necessary permissions (e.g., INTERNET, CAMERA, LOCATION).
· **Defines Application Metadata** – Includes version code, version name, app theme, and other configurations.
· **Sets Intent Filters** – Determines how the app interacts with the system and other apps (e.g., handling specific file types or URLs).
· **Declares Hardware & Software Features** – Specifies device requirements like camera, Bluetooth, GPS, or touchscreen.
· **Defines App Entry Point** – Sets the **Launcher Activity** that starts when the app runs.
· **Controls App Components' Visibility & Security** – Restricts component access using permissions and export settings.
· **Manages App Process & Task Settings** – Defines background process behavior and app launch modes.
Sets **app metadata** like version, theme, and launcher activity.

ACTIVITY LIFECYCLE:


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
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

onCreate() :fires when the system first creates the activity. On activity creation, the activity enters the *Created* state. In the onCreate() method, perform basic application startup logic that happens only once for the entire life of the activity.

onStart()

When the activity enters the Started state, the system invokes onStart(). This call makes the activity visible to the user as the app prepares for the activity to enter the foreground and become interactive. For example, this method is where the code that maintains the UI is initialized.

onResume()

When the activity enters the Resumed state, it comes to the foreground, and the system invokes the onResume() callback. This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app, such as the device receiving a phone call, the user navigating to another activity, or the device screen turning off.

onPause()

The system calls this method as the first indication that the user is leaving your activity, though it does not always mean the activity is being destroyed. It indicates that the activity is no longer in the foreground, but it is still visible if the user is in multi-window mode.

onStop()

When your activity is no longer visible to the user, it enters the *Stopped* state, and the system invokes the onStop() callback. This can occur when a newly launched activity covers the entire screen. The system also calls onStop() when the activity finishes running and is about to be terminated.

onDestroy()

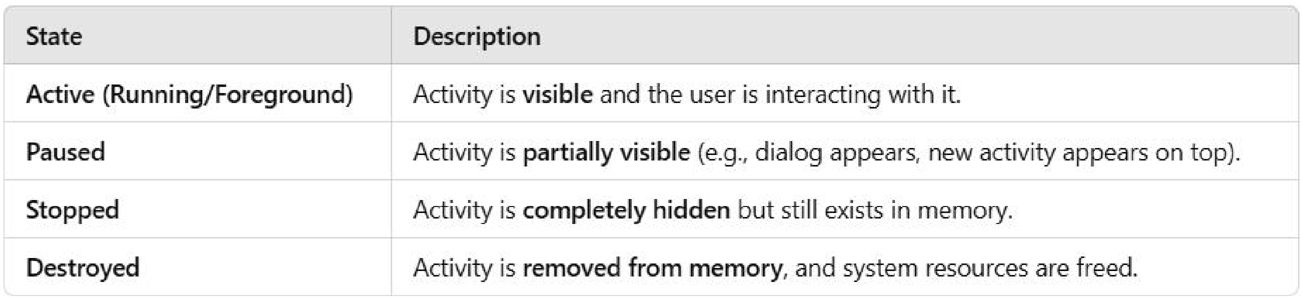
onDestroy() is called before the activity is destroyed. The system invokes this callback for one of two reasons:

1. The activity is finishing, due to the user completely dismissing the activity or due to finish() being called on the activity.

2. The system is temporarily destroying the activity due to a configuration change, such as device rotation or entering multi-window mode.

Activity States in Android

An **Activity State** represents what is happening to an activity at a given time during its lifecycle. Understanding these states is important for **efficient resource management** and **handling user interactions**.



State	Description
Active (Running/Foreground)	Activity is **visible** and the user is interacting with it.
Paused	Activity is **partially visible** (e.g., dialog appears, new activity appears on top).
Stopped	Activity is **completely hidden** but still exists in memory.
Destroyed	Activity is **removed from memory**, and system resources are freed.

Monitoring state changes

Activity Is Killable: This refers to the state where the activity can be terminated by the system to free up resources, usually when it's in the background and not visible to the user.

Activity.onCreate(): This method is called when the activity is first created. It's where initialization tasks such as setting up the user interface or initializing variables are typically performed.

Activity.onRestoreInstanceState(): This method is called after onCreate(), allowing the activity to restore its previously saved state from a bundle.

Activity.onStart(): This method is called when the activity becomes visible to the user but is not yet in the foreground. It's a good place to perform tasks that should happen when the activity becomes visible.

Activity.onResume(): This method is called when the activity is about to become fully visible and interactive to the user. It's where tasks that need to be resumed after pausing can be handled.

Activity.onSaveInstanceState(): This method is called before the activity is paused or stopped, allowing it to save its current state to a bundle that can be restored later.

Activity.onPause(): This method is called when the activity is partially obscured by another activity or when it's about to be paused. It's where tasks that need to be paused can be handled.

Activity.onStop(): This method is called when the activity is no longer visible to the user. It's where tasks that need to be stopped can be handled, such as releasing resources or unregistering listeners.

Activity.onDestroy(): This method is called when the activity is being destroyed, either because the user has finished it or because the system is reclaiming resources. It's where cleanup tasks should be performed.

Activity.onRestart(): This method is called when the activity is being restarted after being stopped. It's where tasks that need to be restarted can be handled.

Why Monitor State Changes?

- To save user data before an app is paused or closed.
- To release or reallocate resources efficiently.
- To track user engagement and analytics.
- To handle system-level events effectively.