5.1. ANDROID OS ARCHITECTURE

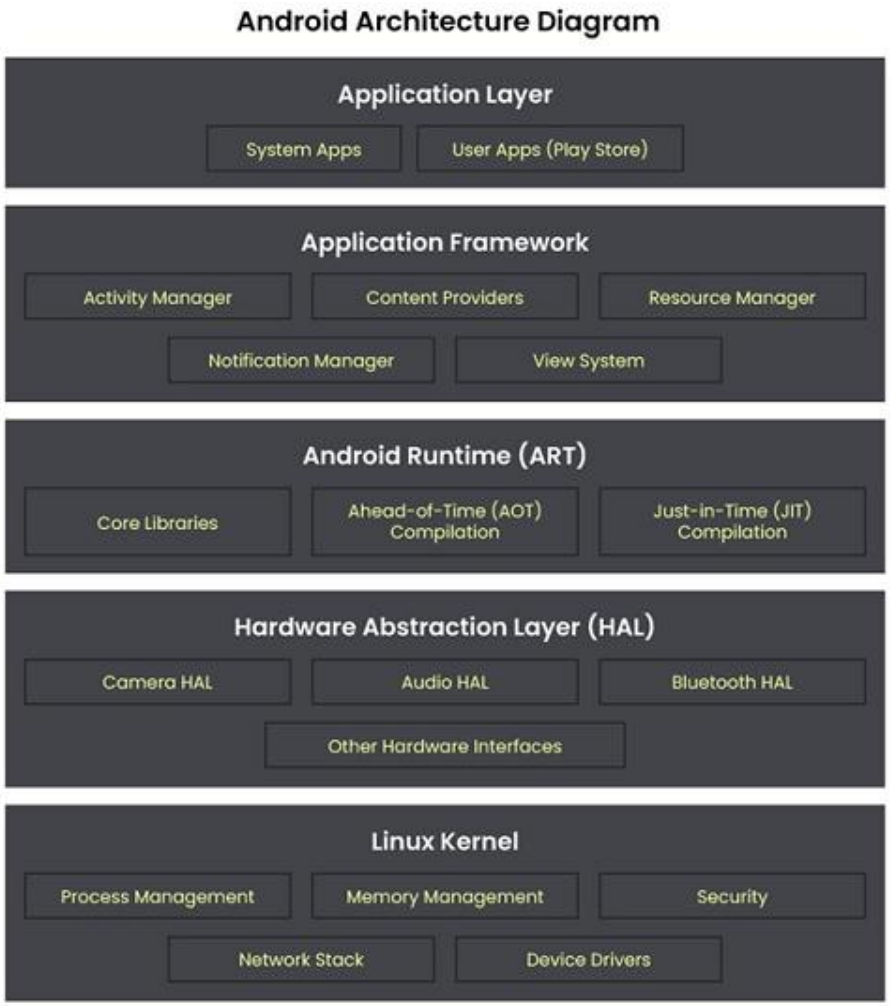
Introduction:

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

Components of Android Architecture:

The main components of Android architecture are the following:-

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

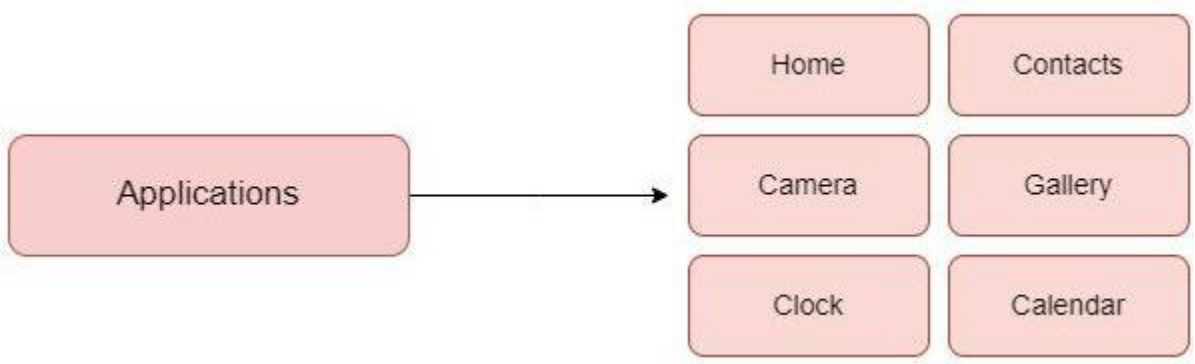


Pictorial representation of Android architecture with several main components and their sub-components.

- Understanding Android's architecture is essential for building efficient applications.
- For those looking to master this structure and move from beginner to advanced skills in Kotlin, the **Android Mastery with Kotlin: Beginner to Advanced course** offers a comprehensive guide

1. Applications:

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



2. Application framework:

- The Application Framework is a core part of Android that gives developers the tools and services they need to build apps.
- It provides access to device features like hardware, screen display, and system resources.
- It includes several important services that make it easier to build powerful and consistent Android apps without having to create everything from scratch.
- The services are as follows:

a) **Activity Manager** - Manages the app's activities and their life cycle (like opening, pausing, or closing screens).

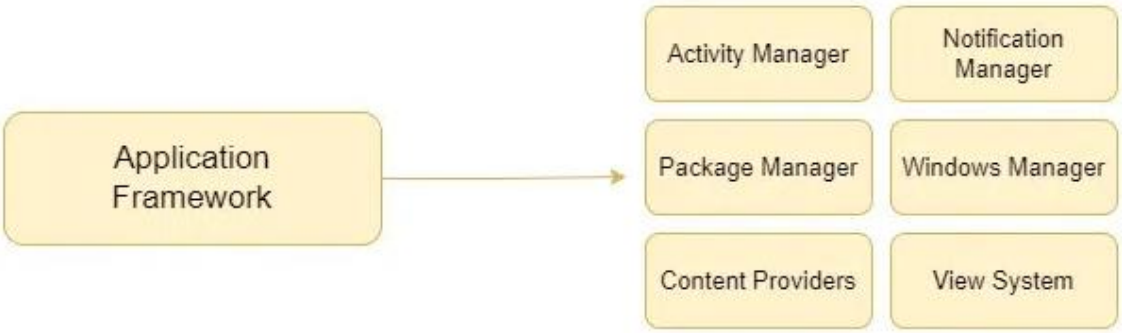
b) **Notification Manager** - Allows apps to show alerts or updates to the user.

c) **Package Manager** - Keeps track of all the apps installed on the device.

d) **Window Manager** - Handles the placement and appearance of windows on the screen.

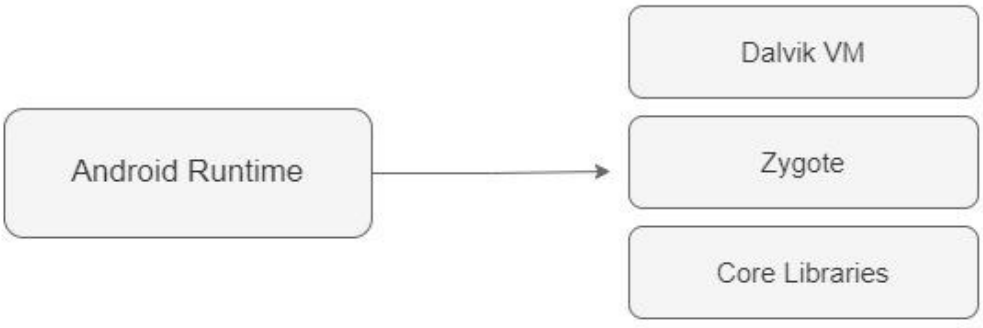
e) **Content Providers** - Help apps share data with other apps (like contacts or photos).

f) **View System** - Controls how things (like buttons or text) appear on the screen.



3. Application runtime:

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



Note: *Now, starting from Android 5.0 and above, we use* ***ART (Android Runtime)*** *to compile bytecode into native code to leverage ahead-of-time compilation.*

4. Platform libraries

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

a) **Media** library provides support to play and record an audio and video formats.

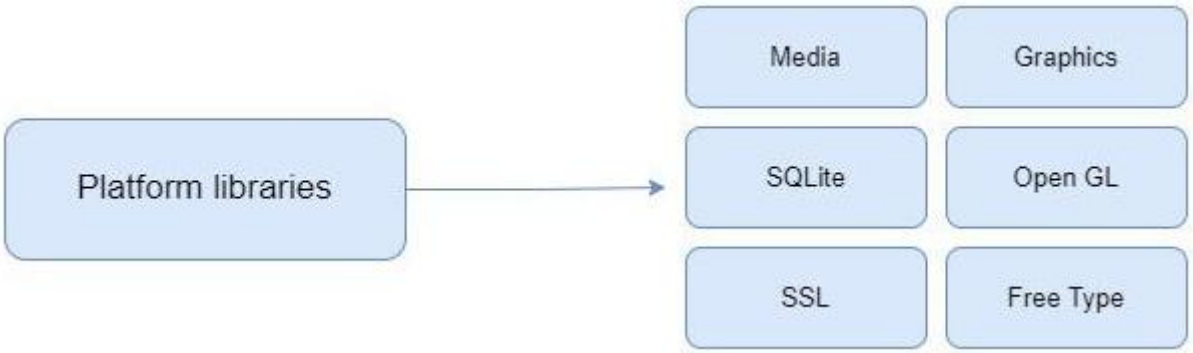
b) **Surface manager** responsible for managing access to the display subsystem.

c) **SGL** and **OpenGL** both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.

d) **SQLite** provides database support and **FreeType** provides font support.

e) **Web-Kit** is open source web browser engine provides all the functionality to display web content and to simplify page loading.

f) **SSL (Secure Sockets Layer)** is security technology to establish an encrypted link between a web server and a web browser.



5. Linux Kernel

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

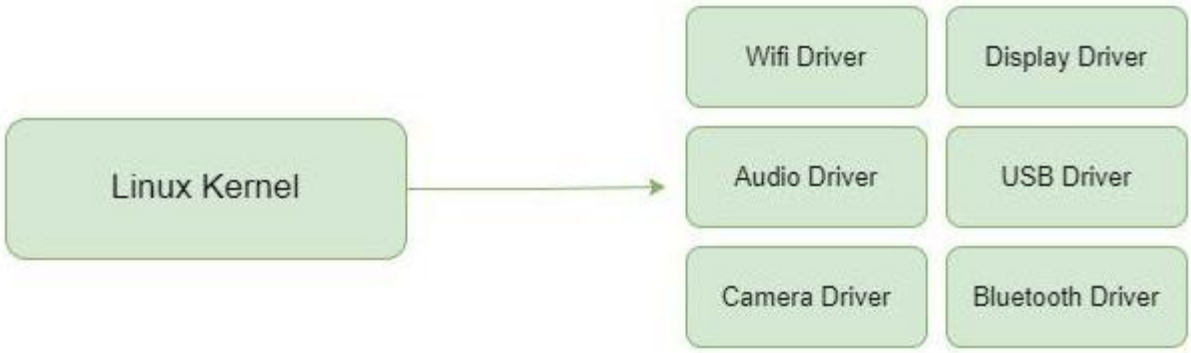
a) **Security:** The Linux kernel handles the security between the application and the system.

b) **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.

c) **Process Management:** It manages the process well, allocates resources to processes whenever they need them.

d) **Network Stack:** It effectively handles the network communication.

e) **Driver Model:** It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.



5.1.1. ZYGOTE

What is Zygote?

- **Zygote** is a **core Android process** that acts as the **template for launching new applications**.
- It is a **specialized virtual machine process** that runs at system startup and preloads **common libraries and resources** used by most Android apps.
- Think of Zygote as a **"parent process"** that spawns new app processes very efficiently, rather than starting each app from scratch.
- This speed up app launch times and reduces memory consumption.

How Zygote Works?

1. **System Startup:**
   - When the Android device boots, the **init process** starts the **Zygote process**.
   - Zygote initializes a **Dalvik/ART virtual machine** and preloads **core libraries, resources, and classes** needed by apps (e.g., android.app, java.lang, android.widget).
2. **Preloading Libraries:**
   - Preloading common libraries avoids repeatedly loading them for each new app.
   - Examples include **UI components, standard Java/Kotlin classes, and graphics libraries**.

3. **Forking New App Processes:**
    - When a user launches an app, the **Activity Manager** requests Zygote to **fork** a new process.
    - Forking creates a **copy of the Zygote process** (child process) with:
        - Its own memory space
        - Same preloaded libraries
    - The child process then **executes the app code**.
4. **Efficiency Advantages:**
    - **Faster app startup:** Most of the heavy lifting (loading classes, initializing runtime) is already done in Zygote.
    - **Reduced memory usage:** Shared memory pages from Zygote are used for multiple apps until modified (copy-on-write).

Zygote and Android Runtime (ART / DVM):

- Zygote runs inside the **Android Runtime** environment.
- It contains:
    - **Core libraries**
    - **ART/DVM runtime**
- By forking a new process from Zygote, the system avoids creating a fresh virtual machine for each app.

Key Components of Zygote:

1. **Preloaded Classes & Resources** – Libraries that are common to almost all apps.
2. **Main Thread (Looper)** – Keeps Zygote alive and ready to fork new processes.
3. **Socket Listener** – Zygote listens for requests from the **Activity Manager** to fork new processes.
4. **Copy-On-Write Memory** – Allows multiple processes to **share memory** until one process modifies it.

Zygote Lifecycle:

1. **Startup:** System boots → init process → start Zygote → preload resources.
2. **Idle state:** Zygote waits for fork requests.
3. **Forking:** Activity Manager asks Zygote to create a new process for an app → Zygote forks → child process becomes the app process.
4. **Running app:** Child process runs the application code independently.

Advantages of Zygote:

- **Faster application launch** – Preloading avoids initializing everything from scratch.
- **Memory efficiency** – Shared memory between apps until modified.
- **Consistency** – All apps start with the same environment and libraries.
- **Lightweight** – Reduces overhead of creating new processes and virtual machines.

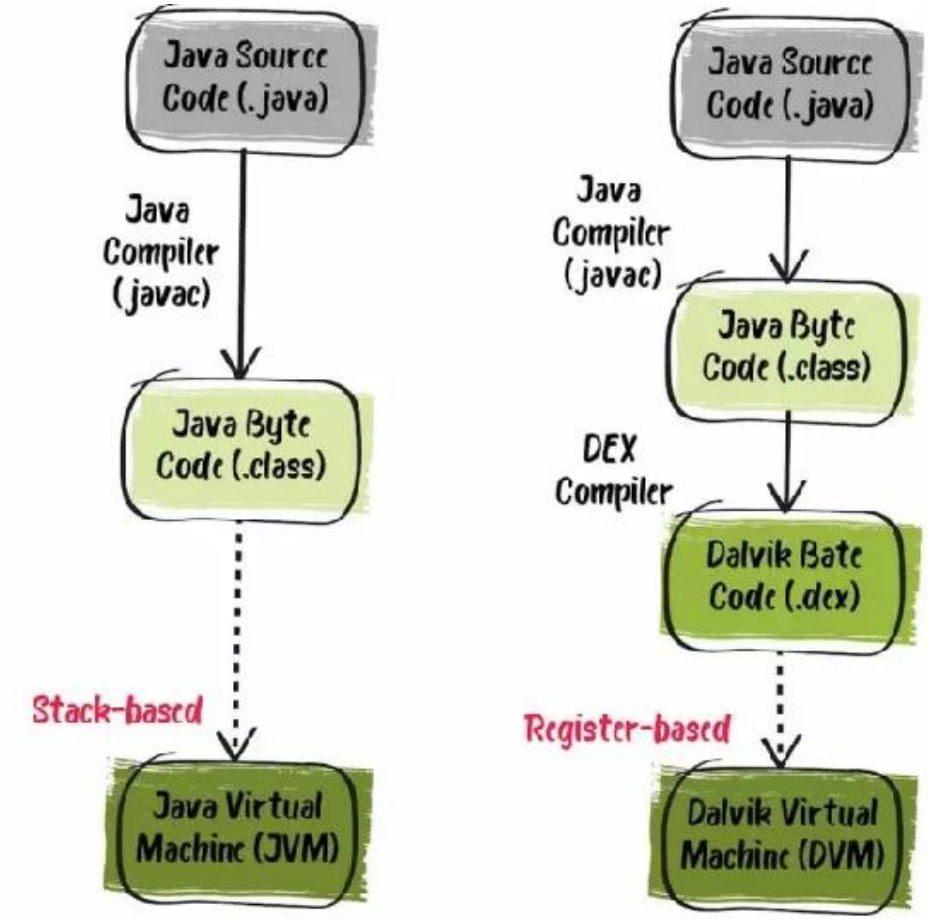
DALVIK VIRTUAL MACHINE (DVM)

What is Dalvik?

- Dalvik is the **original virtual machine** used in Android to run applications.
- It's similar to the **Java Virtual Machine (JVM)** but specially designed for **mobile devices**.

Key points:

- Runs **.dex (Dalvik Executable) files**, which are optimized bytecode for Android.
- Designed to **use minimal memory** and handle **multiple processes simultaneously**.
- Uses a **register-based architecture**, unlike JVM which is stack-based.



How Dalvik Works:

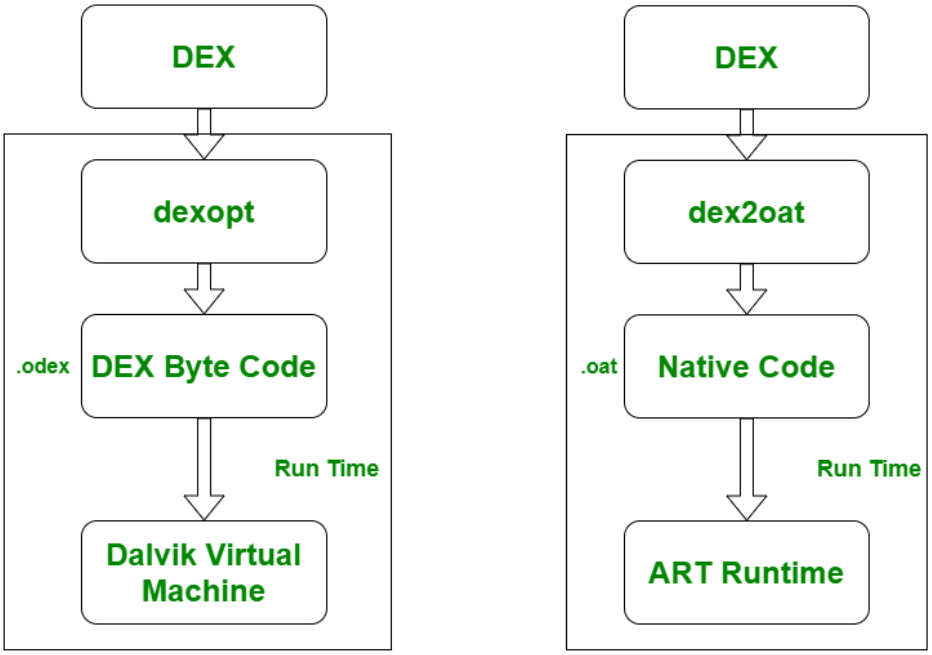
1. Android apps are usually written in **Java/Kotlin**.
2. Code is compiled to **Java bytecode** first.
3. The Java bytecode is then converted into **.dex (Dalvik Executable) format**.
4. DVM executes the .dex bytecode inside each app process, **isolated from other apps**.

Advantages of Dalvik:

- **Optimized for mobile devices** with limited RAM and CPU.
- Supports **multiple instances** efficiently.
- Enables **sandboxed execution**, improving security.

Limitations:

- Uses **Just-In-Time (JIT) compilation**, which compiles code **while the app is running** → can be slower at runtime.
- Less efficient for modern devices with higher processing power.



ANDROID RUNTIME (ART)

What is ART?

- ART (Android Runtime) is the **replacement for Dalvik** starting from **Android 5.0 Lollipop**.
- It improves **performance, battery life, and responsiveness** by using **Ahead-of-Time (AOT) compilation**.

Key points:

- Compiles app **bytecode into native machine code before execution** (at install time).
- Eliminates the **runtime overhead of JIT** compilation in Dalvik.
- Maintains the **same API and compatibility** with Dalvik apps.

How ART Works?

1. App code is written in **Java/Kotlin**.
2. Java bytecode is converted to **.dex files**.
3. ART **pre-compiles .dex files into native code** when the app is installed.
4. When the app runs, it executes **native machine code directly**, making it faster.

Advantages of ART over Dalvik:

Feature	Dalvik	ART
Compilation	JIT (at runtime)	AOT (at install time)
App launch speed	Slower	Faster
Memory usage	Higher	Lower (due to optimized execution)
Battery efficiency	Moderate	Better (less CPU usage)
Profiling	Limited	Supports JIT for runtime profiling + AOT

Key Differences Between Dalvik and ART

1. **Compilation Method:**
   - Dalvik: **JIT** → compiles code while app is running.
   - ART: **AOT** → compiles code at install time, executes native code.
2. **Performance:**
   - Dalvik: Slower startup, higher runtime overhead.
   - ART: Faster app launch, smoother execution.
3. **Memory & Battery:**
   - ART reduces CPU usage and memory overhead, improving **battery life**.
4. **Debugging & Profiling:**
   - ART provides better support for **profiling, garbage collection, and debugging**.

Dalvik/ART in Android Architecture:

- Both Dalvik and ART sit in the **Android Runtime layer**.
- They rely on the **Linux Kernel** for **memory management, threading, and low-level services**.
- They execute apps in **isolated processes**, providing **security and stability**.
- ART retains the same **core libraries and APIs** as Dalvik, ensuring **backward compatibility**.

PERMISSIONS IN ANDROID OS

Introduction:

- **Permissions in Android** are a **security mechanism** used to control how applications access sensitive data and system resources on a device.
- Since Android applications may need access to hardware features (camera, microphone, storage, etc.) or user data (contacts, location), permissions ensure that apps **cannot use these resources without the user's approval**.
- The permission system protects **user privacy, system stability, and device security**.

Purpose of Permissions:

The Android permission system is designed to:

- ✓ Protect **user data** (contacts, messages, photos).
- ✓ Prevent **unauthorized access** to system features.
- ✓ Ensure **secure interaction between applications**.
- ✓ Allow users to **control which apps access their resources**.

Without permissions, any app could freely access private information or hardware resources.

How Android Permissions Work?

Android uses a **sandbox security model**, meaning:

- ✓ Each application runs in its **own isolated environment**.
- ✓ Apps **cannot access other apps' data or system resources** unless they are granted permission.
- ✓ Permissions must be **declared by the developer** and **approved by the user**.

Permission Flow:

1. Developer declares permissions in the **AndroidManifest.xml** file.
2. When installing or running the app, Android **asks the user to grant permission**.
3. If the user approves, the app can access that feature.
4. If denied, the app cannot use that resource.

Example declaration in **AndroidManifest.xml**:

```
<uses-permission android:name="android.permission.CAMERA"/>
```

This permission allows the application to **use the device camera**.

Types of Android Permissions:

Android classifies permissions into several categories.

1. Normal Permissions:

- Normal permissions allow access to **less sensitive resources** that pose minimal risk to the user.
- These permissions are **automatically granted** by the system during installation.

Examples

- ✓ Access internet
- ✓ Set wallpaper
- ✓ Access network state

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

Apps do not need explicit user approval for these permissions.

2. Dangerous Permissions:

- Dangerous permissions allow access to **sensitive user data or hardware features**.
- These require **explicit user approval at runtime**.

Examples:

Permission	Purpose
Camera	Access device camera
Contacts	Read user contacts
Location	Access GPS location
Microphone	Record audio
Storage	Read/write files

Example:

<uses-permission

android:name="android.permission.ACCESS_FINE_LOCATION"/>

When the app tries to use location, Android shows a **permission request dialog**.

3. Signature Permissions:

- Signature permissions are granted **only if two applications are signed with the same developer certificate**.
- These are mainly used for **communication between apps from the same developer or system components**.

Example use:

- Google apps sharing internal services.

4. Special Permissions:

- Special permissions control **very sensitive system features**.
- These permissions must be manually enabled by the user in **device settings**.

Examples include:

- ✓ Draw over other apps
- ✓ Modify system settings
- ✓ Battery optimization exemptions

Example: Chat heads feature in messaging apps.

Permission Groups:

- Android organizes permissions into **groups**.
- If a user grants one permission from a group, Android may automatically grant other permissions in the same group.

Example Permission Groups

Group	Permissions
Camera	Camera access
Location	GPS, network location
Contacts	Read/write contacts
Storage	Read/write external storage
Microphone	Record audio

Grouping simplifies permission management for users.

Runtime Permission Model (Android 6.0+):

- Before Android 6.0, permissions were granted **during installation**.
- From Android 6.0 onwards, Android introduced **runtime permissions**.

Runtime Permission Process:

1. App requests permission during runtime.
2. System displays a **dialog box**.
3. User can **Allow or Deny**.
4. User can change permissions later in **Settings**.

Example dialog:

Allow Maps to access your location?

[Allow] [Deny]

This model gives **more control to the user**.

Permission Revocation:

Users can revoke permissions anytime.

Steps:

1. Open **Settings**
2. Go to **Apps**
3. Select the application
4. Choose **Permissions**
5. Enable or disable specific permissions

This helps users maintain **privacy control**.

Permission Best Practices:

Developers should follow certain guidelines:

- ✓ Request **only necessary permissions**.
- ✓ Provide **clear explanation** before requesting sensitive permissions.
- ✓ Handle **permission denial gracefully**.
- ✓ Avoid requesting multiple permissions unnecessarily.

Following these practices improves **user trust and app security**.

Example Scenario:

Consider a **navigation app**.

It may require:

Permission	Purpose
Location	Track user location
Internet	Download maps
Storage	Save offline maps

If the user denies **location permission**, the app cannot provide navigation functionality.

Role of Permissions in Android Security:

Permissions are part of Android's **multi-layered security model**, which includes:

- ✓ **Application sandbox**
- ✓ **Linux kernel security**
- ✓ **App signing**
- ✓ **Permission framework**

Together, these mechanisms ensure that apps **operate safely without compromising user privacy or system integrity**.