**1.Explain 6 components that are building blocks of the android application**.

Android applications are built using key components that define their structure and behavior. Here are six fundamental building blocks of an Android application:

**1. Activities**

* An **Activity** represents a single screen in an application.
* It handles user interactions and acts as an entry point for users.
* Example: A login screen or a home page.

**2. Services**

* A **Service** runs in the background to perform long-running operations without a user interface.
* It can be used for tasks like playing music, fetching data, or syncing with a server.
* Example: A music player running in the background.

**3. Broadcast Receivers**

* **Broadcast Receivers** respond to system-wide broadcast messages or app-specific broadcasts.
* They enable communication between different components or applications.
* Example: Detecting when the battery is low or when the device connects to Wi-Fi.

**4. Content Providers**

* A **Content Provider** manages access to structured data such as databases, files, or cloud storage.
* It allows different applications to share data securely.
* Example: The Contacts app providing access to saved contacts.

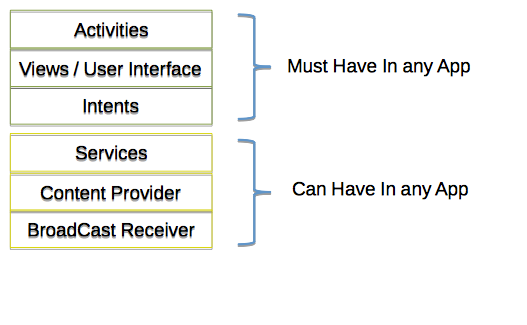
**5. Fragments**

* A **Fragment** is a modular section of an Activity that can be reused across multiple screens.
* It helps in building responsive UIs for different device sizes.
* Example: A news app with a list of articles on one side and details on the other in a tablet layout.

**6. Intents**

* An **Intent** is a messaging object used to request actions from other components (inside or outside the app).
* It facilitates communication between activities, services, and broadcast receivers.
* Example: Opening the camera app from within another app.

Each of these components plays a crucial role in designing and developing an efficient Android application.



**2.Explain the purpose and importance of different types of adapter in list view.(4 marks)**

**🔹 What is an Adapter?**

An **Adapter** in Android is used to **connect data sources** (like Arrays, Databases, or APIs) to **ListView**, allowing data to be displayed in a structured manner. It acts as a **bridge** between the data and UI components.

**🔹 Importance of Adapters in ListView**

✅ **Manages Data Efficiently** – Fetches and displays data in a structured format.  
✅ **Optimizes Memory Usage** – Loads only visible list items and reuses views.  
✅ **Supports Custom Layouts** – Allows custom designs with images, text, and buttons.  
✅ **Works with Multiple Data Sources** – Can fetch data from **Arrays, JSON, APIs, or Databases**.  
✅ **Improves App Performance** – Helps avoid lag in scrolling large datasets.

**🔹 Types of Adapters in ListView**

| **Adapter Type** | **Purpose** | **Example Use Case** |
| --- | --- | --- |
| **ArrayAdapter** | Displays a simple list of text-based items | List of countries, names, or cities |
| **BaseAdapter** | Used for **custom layouts** with multiple views | Contacts list with images & buttons |
| **SimpleAdapter** | Displays **structured data** (key-value pairs) | Employee records from an API |
| **CursorAdapter** | Fetches and displays data from an **SQLite database** | Contacts app fetching saved numbers |

**🔹 Example Usage of Adapters**

1️⃣ **ArrayAdapter Example (List of Countries)**

java

CopyEdit

String[] countries = {"USA", "India", "UK", "Canada"};

ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1, countries);

listView.setAdapter(adapter);

📌 **Use Case:** A simple text list of countries.

2️⃣ **BaseAdapter Example (Custom Contacts List)**

java

CopyEdit

public class CustomAdapter extends BaseAdapter {

private Context context;

private List<Contact> contacts;

@Override

public View getView(int position, View convertView, ViewGroup parent) {

if (convertView == null) {

convertView = LayoutInflater.from(context).inflate(R.layout.list\_item, parent, false);

}

TextView name = convertView.findViewById(R.id.name);

ImageView profilePic = convertView.findViewById(R.id.profilePic);

Contact contact = contacts.get(position);

name.setText(contact.getName());

profilePic.setImageResource(contact.getImageResId());

return convertView;

}

}

📌 **Use Case:** Used in **contacts apps** like WhatsApp to display names & profile pictures.

**🔹 Conclusion**

Adapters are essential in Android development for **efficiently displaying data** in ListView. Choosing the right adapter ensures **smooth scrolling, better performance, and enhanced user experience**. 🚀

**2b)With the neat diagram explain the difference between Dalvik and java virtual machine**

**🔹 What is JVM?**

Java Virtual Machine (JVM) is used to run **Java applications** on various platforms. It converts Java **bytecode** into machine code using **Just-In-Time (JIT) compilation**.

**🔹 What is DVM?**

Dalvik Virtual Machine (DVM) was specifically designed for **Android OS** to run **Android applications** efficiently. It executes **Dalvik bytecode**, which is optimized for low memory and battery usage.

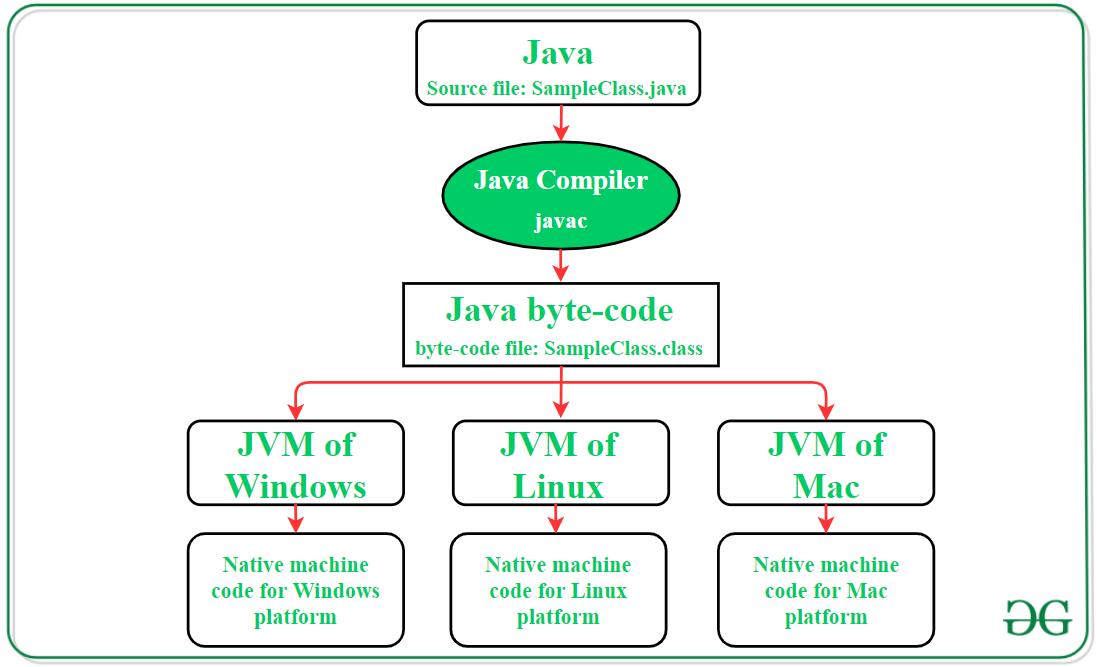
**Key Differences Between DVM and JVM**

| **Feature** | **JVM (Java Virtual Machine)** | **DVM (Dalvik Virtual Machine)** |
| --- | --- | --- |
| **Designed For** | Desktop & Server applications | Android mobile applications |
| **Bytecode Format** | **Java Bytecode** (.class files) | **Dalvik Bytecode** (.dex files) |
| **Compilation Type** | Just-In-Time (JIT) Compilation | Just-In-Time (JIT) & Ahead-Of-Time (AOT) in ART |
| **Memory Usage** | Higher due to separate memory for each instance | Optimized with shared memory |
| **Execution** | Runs each app in its own JVM instance | Uses a single **DVM instance** to run multiple apps efficiently |
| **Performance** | Less optimized for mobile devices | Optimized for **low memory & power consumption** |
| **Replaced By** | Still in use for desktop apps | **Replaced by ART (Android Runtime) in modern Android versions** |

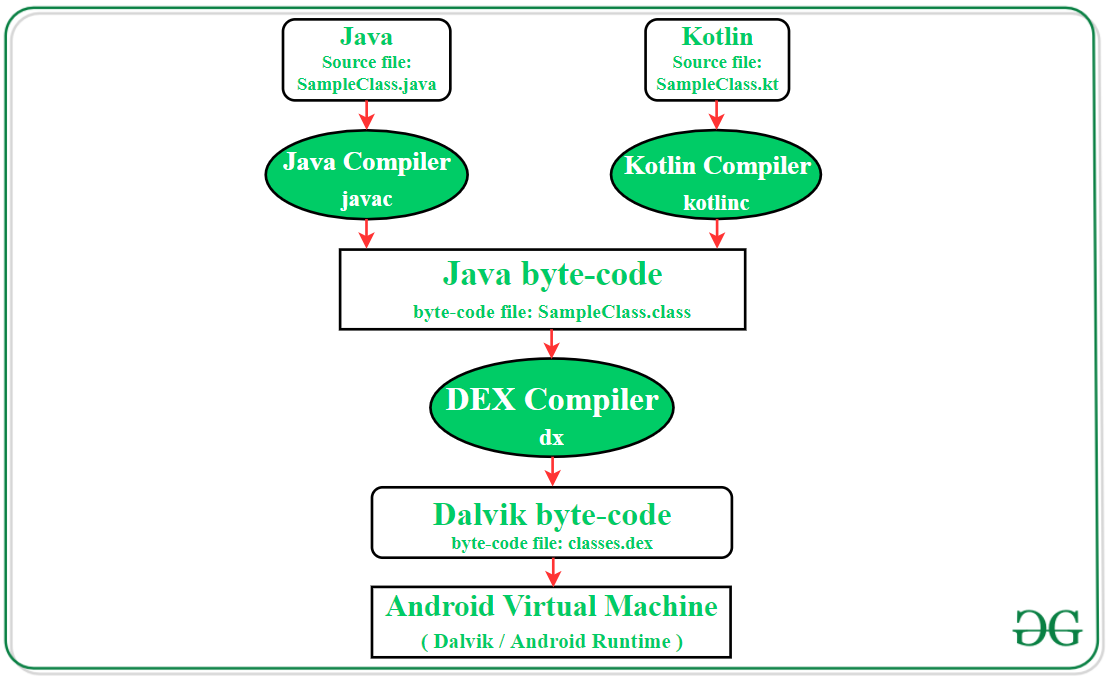
**🔹 Conclusion**

* **JVM** is used for **desktop applications** and provides a standard environment for running Java programs.
* **DVM** was optimized for **Android apps** to ensure **better performance and battery efficiency**.
* **Newer Android versions (Lollipop & above) use ART (Android Runtime) instead of DVM** for better performance.

.



DVM



**3.a)Explain in detail the types of android applications.**

**Types of Android Applications**

Android applications can be categorized based on their functionality, target audience, and purpose. Below are the main types of Android applications:

**1️⃣ Native Applications**

**📌 Definition:**  
Native applications are developed specifically for **Android OS** using **Java, Kotlin, or C++** and run directly on the Android device.

**🔹 Features:**  
✅ Developed using **Android SDK**.  
✅ Provides **high performance** and **better user experience**.  
✅ Uses device features like GPS, Camera, Sensors, etc.

**🔸 Examples:**

* WhatsApp (Messaging)
* Google Maps (Navigation)
* Instagram (Social Media)

**2️⃣ Web Applications**

**📌 Definition:**  
Web applications run inside a **web browser** using **HTML, CSS, JavaScript** and are not installed on the device. They require an **internet connection** to function.

**🔹 Features:**  
✅ No need for installation; accessed via a browser.  
✅ Uses **WebView** to display web content.  
✅ Limited access to device features.

**🔸 Examples:**

* Google Docs (Online Document Editing)
* Facebook Lite (Web-based Social Media)
* Amazon (E-commerce Website)

**3️⃣ Hybrid Applications**

**📌 Definition:**  
Hybrid apps combine features of **both native and web applications**. They are built using web technologies but wrapped in a **native container** to run like a native app.

**🔹 Features:**  
✅ Developed using **React Native, Flutter, Ionic, or PhoneGap**.  
✅ Runs on multiple platforms (**Android & iOS**).  
✅ Provides access to device features via plugins.

**🔸 Examples:**

* Instagram (Partially hybrid)
* Twitter (Uses web and native components)
* Uber (Built using React Native)

**4️⃣ Gaming Applications**

**📌 Definition:**  
Gaming apps are specifically designed for **entertainment and gaming** purposes. They require **high graphics, animations, and processing power**.

**🔹 Features:**  
✅ Uses **Unity, Unreal Engine, or Android Game SDK**.  
✅ High-performance and interactive.  
✅ May include **in-app purchases and multiplayer options**.

**🔸 Examples:**

* PUBG Mobile (Multiplayer Shooting Game)
* Candy Crush (Puzzle Game)
* Asphalt 9 (Racing Game)

**5️⃣ Business Applications**

**📌 Definition:**  
Business apps help in **productivity, management, and business operations**. They are used in **offices, banks, and industries**.

**🔹 Features:**  
✅ Cloud-based access to data.  
✅ Secure transactions and communication.  
✅ Designed for **business and enterprise use**.

**🔸 Examples:**

* Microsoft Teams (Collaboration & Communication)
* Zoom (Video Conferencing)
* Google Drive (Cloud Storage)

**6️⃣ Educational Applications**

**📌 Definition:**  
These apps are designed for **learning, training, and skill development**.

**🔹 Features:**  
✅ Includes **videos, quizzes, and interactive lessons**.  
✅ Supports **online classes and self-learning**.  
✅ Many provide **certifications and courses**.

**🔸 Examples:**

* Udemy (Online Learning)
* Duolingo (Language Learning)
* Byju’s (Educational Platform)

**7️⃣ Social Media Applications**

**📌 Definition:**  
These apps help people **connect, communicate, and share content** with others.

**🔹 Features:**  
✅ Real-time chatting and messaging.  
✅ Supports **images, videos, and live streaming**.  
✅ Engages users with **notifications and updates**.

**🔸 Examples:**

* Facebook (Social Networking)
* WhatsApp (Instant Messaging)
* TikTok (Short Video Sharing)

**8️⃣ E-Commerce Applications**

**📌 Definition:**  
E-Commerce apps allow users to **buy and sell products online**.

**🔹 Features:**  
✅ Online shopping, cart, and payment gateway.  
✅ Product reviews, recommendations, and discounts.  
✅ Secure transactions and delivery tracking.

**🔸 Examples:**

* Amazon (Online Shopping)
* Flipkart (E-Commerce Store)
* Myntra (Fashion Shopping)

**9️⃣ Health & Fitness Applications**

**📌 Definition:**  
Health apps help users **track fitness, exercise, and medical data**.

**🔹 Features:**  
✅ Tracks **heart rate, calories, and workouts**.  
✅ Syncs with **smartwatches and fitness devices**.  
✅ Provides **diet plans and health tips**.

**🔸 Examples:**

* Google Fit (Fitness Tracking)
* MyFitnessPal (Calorie Counter)
* Practo (Online Doctor Consultation)

**3.B)write a note on android SDK features**.

**Android SDK Features**

The **Android Software Development Kit (SDK)** is a collection of tools, libraries, and resources that developers use to create Android applications. It includes everything needed to build, test, and debug apps for Android devices.

**🔹 Key Features of Android SDK**

**1️⃣ Android Emulator**

✅ Allows developers to test applications on a **virtual Android device**.  
✅ Supports different **screen sizes, versions, and hardware configurations**.  
✅ Speeds up development by enabling testing without a physical device.

**2️⃣ Android Studio**

✅ The official **Integrated Development Environment (IDE)** for Android development.  
✅ Provides **code editor, debugging tools, and UI design tools**.  
✅ Supports **Kotlin, Java, and C++** for app development.

**3️⃣ SDK Tools & Build System**

✅ Includes tools like **ADB (Android Debug Bridge)** and **Gradle Build System**.  
✅ Helps in **compiling, packaging, and signing** Android applications.  
✅ Automates **dependency management** and app deployment.

**4️⃣ UI Design & Layout Editor**

✅ Provides a **visual editor** to design app layouts using **XML**.  
✅ Supports **drag-and-drop UI components** like buttons, text fields, and images.  
✅ Allows previewing UI on different **screen sizes and resolutions**.

**5️⃣ API Libraries**

✅ Includes libraries for **graphics, media, location services, and sensors**.  
✅ Supports features like **Google Maps, Firebase, Camera, and Bluetooth**.  
✅ Enables integration with **cloud services and databases**.

**6️⃣ Testing & Debugging Tools**

✅ Provides tools like **Logcat, Profiler, and Debugger** to track app performance.  
✅ Supports **JUnit and Espresso** for unit and UI testing.  
✅ Helps identify and fix **bugs, crashes, and memory leaks**.

**7️⃣ Support for Multiple Android Versions**

✅ Allows developers to create apps for **different Android versions (API Levels)**.  
✅ Ensures backward compatibility with **older Android devices**.

**8️⃣ Google Play Services Integration**

✅ Provides APIs for **authentication, cloud messaging, location tracking, and in-app payments**.  
✅ Enables apps to use **Google Sign-In, Google Maps, and Firebase**.

**🔹 Conclusion**

The Android SDK is an essential toolkit for Android development, offering **powerful tools, APIs, and libraries** to build, test, and optimize apps efficiently. 🚀

**4.Explain with a neat diagram fragment lifecycle.**

**📌 Fragment Lifecycle in Android**

A **Fragment** is a reusable portion of an Android activity's UI that allows modular design and flexible UI management. Like activities, fragments have their own **lifecycle** managed by the Android system.

**🔹 Fragment Lifecycle Diagram**

pgsql

CopyEdit

+------------------------+

| onAttach() | 🔹 Fragment is attached to the Activity

+------------------------+

↓

+------------------------+

| onCreate() | 🔹 Initializes fragment, loads data

+------------------------+

↓

+------------------------+

| onCreateView() | 🔹 Creates and returns UI layout

+------------------------+

↓

+------------------------+

| onActivityCreated() | 🔹 Activity is fully created

+------------------------+

↓

+------------------------+

| onStart() | 🔹 Fragment becomes visible

+------------------------+

↓

+------------------------+

| onResume() | 🔹 Fragment is active and interactive

+------------------------+

↓

+------------------------+

| onPause() | 🔹 Fragment is partially hidden

+------------------------+

↓

+------------------------+

| onStop() | 🔹 Fragment is fully hidden

+------------------------+

↓

+------------------------+

| onDestroyView() | 🔹 Removes UI elements from memory

+------------------------+

↓

+------------------------+

| onDestroy() | 🔹 Cleans up fragment resources

+------------------------+

↓

+------------------------+

| onDetach() | 🔹 Fragment is detached from activity

+------------------------+

**🔹 Explanation of Each Fragment Lifecycle Method**

| **Method** | **Description** |
| --- | --- |
| **onAttach()** | Called when the fragment is attached to the activity. |
| **onCreate()** | Initializes the fragment (data loading, saved states). |
| **onCreateView()** | Creates and returns the fragment’s UI layout. |
| **onActivityCreated()** | Called after the activity's onCreate() method is completed. |
| **onStart()** | Fragment becomes visible to the user. |
| **onResume()** | Fragment is fully visible and interactive. |
| **onPause()** | Called when the fragment is partially visible (another activity appears). |
| **onStop()** | Called when the fragment is completely hidden. |
| **onDestroyView()** | Removes the fragment’s UI elements from memory. |
| **onDestroy()** | Cleans up fragment resources and states. |
| **onDetach()** | Called when the fragment is detached from the activity. |

**🔹 Key Points to Remember**

✅ Fragments are **lightweight** and allow dynamic UI updates.  
✅ They can be **reused** in multiple activities.  
✅ onCreateView() is the most important method for setting up UI.  
✅ onDestroyView() releases UI-related resources to avoid **memory leaks**.

**5.Explain the different types of layout in android with example**.

**📌 Types of Layouts in Android**

In Android, **layouts** define the **UI structure** of an application. They organize **views (UI elements)** such as buttons, text fields, and images on the screen.

**1️⃣ Linear Layout**

✅ **Arranges elements in a single direction** (**horizontal or vertical**).  
✅ Uses **android:orientation="vertical"** or **"horizontal"**.  
✅ Simple but not efficient for complex UIs.

**🔹 Example (Vertical Linear Layout):**

xml

CopyEdit

<LinearLayout

android:orientation="vertical"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:text="Hello, Android!"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<Button

android:text="Click Me"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

</LinearLayout>

📌 **Use Case:** Login form, simple lists, basic UI structures.

**2️⃣ Relative Layout**

✅ Arranges elements **relative to each other** or **the parent**.  
✅ Uses attributes like **android:layout\_below, android:layout\_toRightOf**.  
✅ More flexible than **LinearLayout**.

**🔹 Example:**

xml

CopyEdit

<RelativeLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:id="@+id/textView"

android:text="Enter Name:"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<EditText

android:layout\_below="@id/textView"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"/>

</RelativeLayout>

📌 **Use Case:** Forms, chat screens, complex layouts.

**3️⃣ Constraint Layout**

✅ **Advanced and flexible** layout, introduced to replace **RelativeLayout**.  
✅ Uses **constraints** to position UI elements **without nested layouts**.  
✅ Improves **performance** and reduces layout complexity.

**🔹 Example:**

xml

CopyEdit

<androidx.constraintlayout.widget.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<Button

android:id="@+id/button"

android:text="Press Me"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>

📌 **Use Case:** Modern Android UIs, dynamic and responsive layouts.

**4️⃣ Table Layout**

✅ Organizes views **in rows and columns**, similar to a **table**.  
✅ Uses **TableRow** to define rows.  
✅ Suitable for **grid-like** structures.

**🔹 Example:**

xml

CopyEdit

<TableLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TableRow>

<TextView android:text="Name" />

<EditText android:hint="Enter Name"/>

</TableRow>

<TableRow>

<TextView android:text="Email" />

<EditText android:hint="Enter Email"/>

</TableRow>

</TableLayout>

📌 **Use Case:** Forms, timetable apps, structured data.

**5️⃣ Frame Layout**

✅ Designed to **hold a single child view** (but can overlap multiple views).  
✅ Often used for **fragment containers**.

**🔹 Example:**

xml

CopyEdit

<FrameLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<ImageView

android:src="@drawable/sample\_image"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

<TextView

android:text="Overlay Text"

android:layout\_gravity="center"/>

</FrameLayout>

📌 **Use Case:** Overlapping views, **fragments**, splash screens.

**6️⃣ Grid Layout**

✅ Displays items **in a grid-like structure** (**rows & columns**).  
✅ More flexible than **TableLayout**.

**🔹 Example:**

xml

CopyEdit

<GridLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:columnCount="2">

<Button android:text="1"/>

<Button android:text="2"/>

<Button android:text="3"/>

<Button android:text="4"/>

</GridLayout>

📌 **Use Case:** **Image galleries, dashboards, calculators**.

**🔹 Conclusion**

Each layout type serves a **different purpose**:

* **LinearLayout** → Simple, single-direction views.
* **RelativeLayout** → Views positioned relative to each other.
* **ConstraintLayout** → Modern, flexible, and optimized.
* **TableLayout** → Grid-like structure for forms.
* **FrameLayout** → Overlaying views, fragments.
* **GridLayout** → Multi-column structures like **dashboards**.

**6.With the help of neat diagram explain android architecture in detail**.

**1️⃣ Linux Kernel (Base Layer)**

✅ **Foundation of Android OS**, providing low-level system functionalities.  
✅ Handles **memory management, security, drivers, and process management**.  
✅ Includes **drivers** for **Wi-Fi, Bluetooth, Camera, Audio, Display, etc.**

**🔹 Key Components:**

* **Power Management** (Battery & CPU usage).
* **Drivers** (Display, Camera, USB, Wi-Fi, etc.).
* **Process Management** (Manages running apps & resources).

📌 **Example:** The Linux Kernel ensures that an Android device’s **Wi-Fi driver** works correctly.

**2️⃣ Hardware Abstraction Layer (HAL)**

✅ Acts as a **bridge between hardware components and the software layer**.  
✅ HAL APIs allow the Android system to communicate with **device hardware**.

**🔹 Key Components:**

* **Camera HAL** (Interacts with the Camera hardware).
* **Audio HAL** (Handles sound output & microphone input).
* **Bluetooth & Wi-Fi HAL** (Manages wireless connections).

📌 **Example:** When a user takes a photo, **HAL ensures that the Camera hardware works with the software properly**.

**3️⃣ Android Runtime (ART) & Core Libraries**

✅ **ART (Android Runtime)** is responsible for **executing apps efficiently**.  
✅ Replaces the older **Dalvik Virtual Machine (DVM)** for better performance.  
✅ Uses **Just-In-Time (JIT) and Ahead-Of-Time (AOT) Compilation** for faster execution.

**🔹 Key Components:**

* **Core Libraries** (Java & Kotlin APIs for app development).
* **Garbage Collector** (Manages memory automatically).

📌 **Example:** ART optimizes an app like **WhatsApp** to run smoothly by managing memory and execution speed.

**4️⃣ Android Framework**

✅ Provides **pre-built services** for app development.  
✅ Defines APIs for **UI design, background services, notifications, sensors, etc.**

**🔹 Key Components:**

* **Activity Manager** (Manages app lifecycle & navigation).
* **Content Providers** (Handles data sharing between apps).
* **View System** (UI elements like buttons, text fields, layouts).
* **Broadcast Receiver** (Listens for system-wide events like battery low).

📌 **Example:** The **Activity Manager** ensures that when a user switches between apps, they don’t restart unnecessarily.

**5️⃣ Applications Layer (Top Layer)**

✅ Contains **pre-installed apps & user-installed apps**.  
✅ Apps interact with the **framework & runtime** to perform actions.

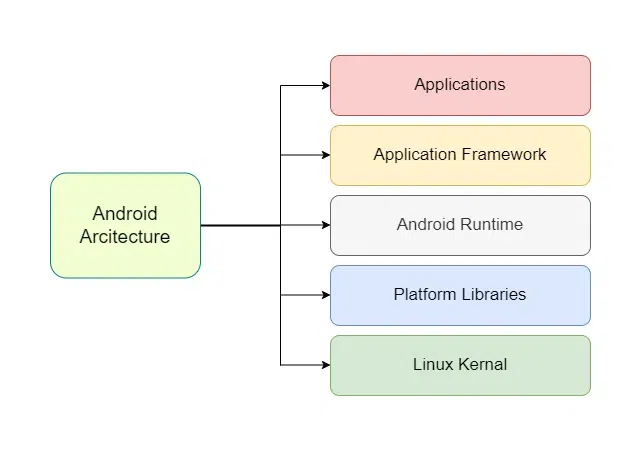
**🔹 Examples:**

* **System Apps**: Phone, Contacts, Settings, Messages.
* **User Apps**: Facebook, Instagram, Google Maps, YouTube.

📌 **Example:** When a user opens **Google Maps**, it accesses the **Location API** from the framework to fetch the current GPS location.

**🔹 Conclusion**

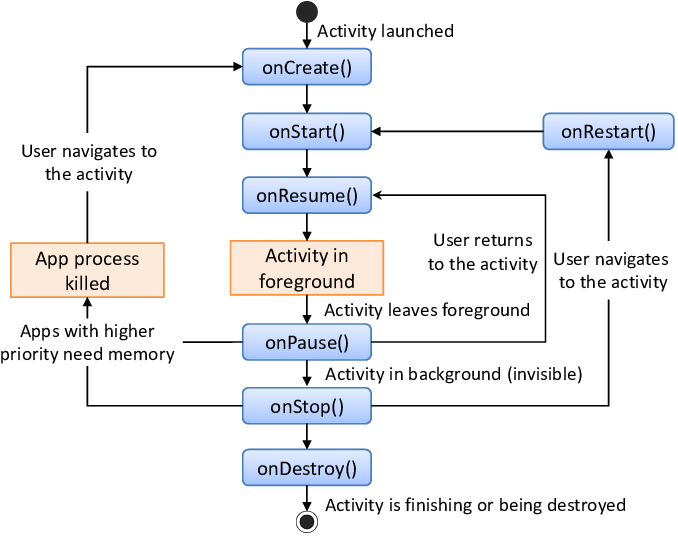
The **Android architecture** ensures efficient performance, security, and smooth interaction between hardware & software. Each layer has a specific role, allowing developers to build powerful applications.



**7.Draw the diagram for activity lifecycle and explain it.**

In [Android](https://www.geeksforgeeks.org/android-app-development-fundamentals-for-beginners/), an **activity** is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities.   
Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it. There are four stages of an activity.

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



| **Method** | **Description** |
| --- | --- |

|  |  |
| --- | --- |
| **onCreate()** | Called when the activity is first created. Used for initializing components. |

|  |  |
| --- | --- |
| **onStart()** | Called when the activity becomes visible to the user. |

|  |  |
| --- | --- |
| **onResume()** | Called when the activity is in the foreground and the user can interact with it. |

|  |  |
| --- | --- |
| **onPause()** | Called when another activity appears, and this activity goes into the background. |

|  |  |
| --- | --- |
| **onStop()** | Called when the activity is no longer visible to the user. |

|  |  |
| --- | --- |
| **onDestroy()** | Called before the activity is destroyed (either manually or by the system). |

**📌 Example Use Case**

📌 **WhatsApp Chat Screen Activity Lifecycle**

* When you **open a chat**, onCreate() → onStart() → onResume() executes.
* If you **press the home button**, onPause() → onStop() runs.
* If you **reopen WhatsApp**, onRestart() → onStart() → onResume() restores the chat.
* If you **exit WhatsApp completely**, onDestroy() is called.