Mobile Development Introduction to Android

1- Introduction:

- ANDROID: AN OPEN PLATFORM FOR MOBILE DEVELOPMENT
- Android is an open source and Linux-based Operating System for mobile devices such as smartphones and tablet computers. Android was developed by the *Open Handset Alliance Company*, led by Google, and other companies.
- Android is designed primarily for touch screens mobile devices such as smartphones and tablet computers. The operating system has developed a lot in the last 15 years starting from black and white phones to recent smartphones or mini computers. One of the most widely used mobile OS these days is android
- Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

Google's Andy Rubin describes Android as follows:

"The first truly open and comprehensive platform for mobile devices. It includes an operating system, user-interface and applications — all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation."

- The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.
- On June 27, 2012, at the Google I/O conference, Google announced the next Android version, 4.1 Jelly Bean. Jelly Bean is an incremental update, with the primary aim of improving the user interface, both in terms of functionality and performance.

An Android is an ecosystem made up of a combination of three components:

- A free, open-source operating system for embedded devices
- An open-source development platform for creating applications
- Devices, particularly mobile phones, that run the Android operating system and the applications created for it.

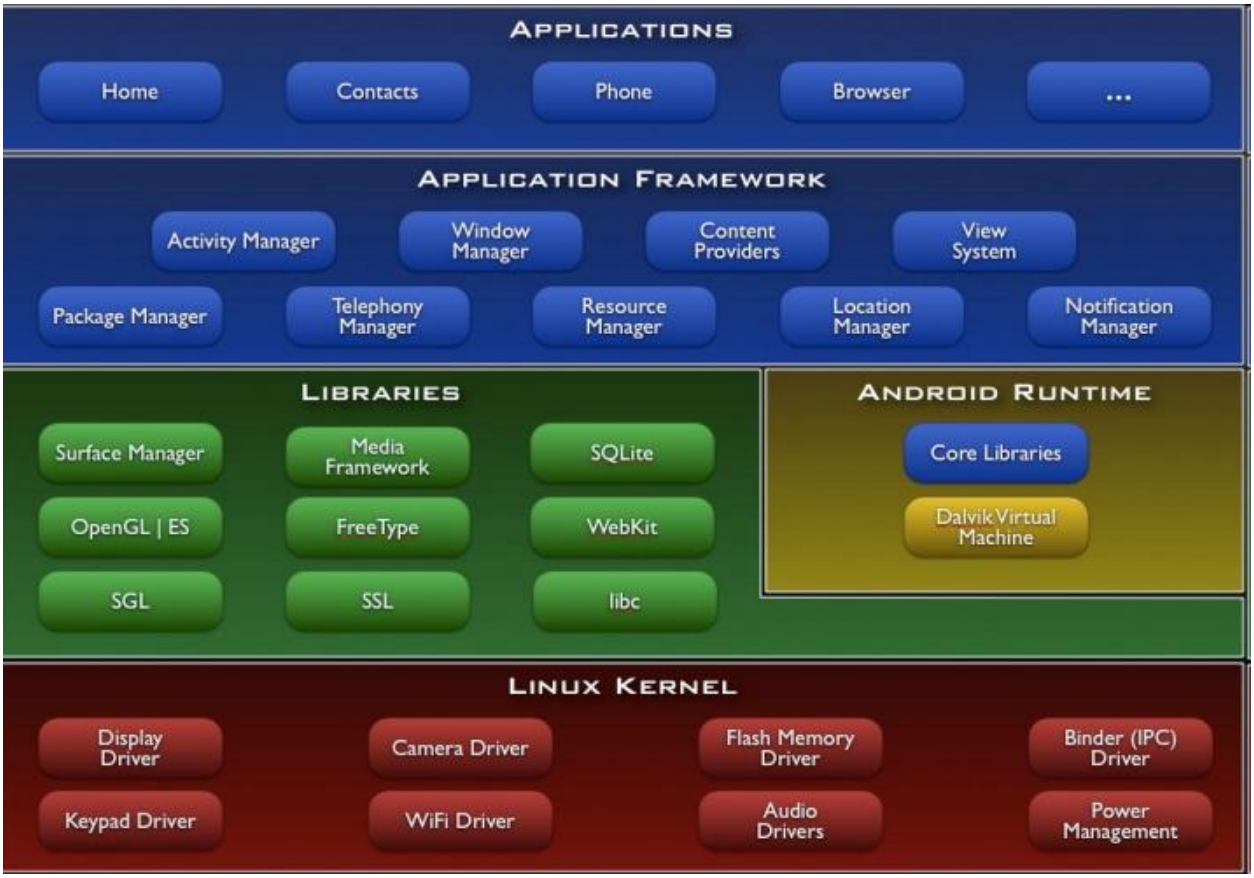
2- Components of Android

Android is made up of several necessary and dependent parts, including the following:

- A Compatibility Definition Document (CDD) and Compatibility Test Suite (CTS) that describe the capabilities required for a device to support the software stack.
- A Linux operating system kernel that provides a low-level interface with the hardware, memory management, and process control, all optimized for mobile and embedded devices.
- Open-source libraries for application development, including SQLite, WebKit, OpenGL, and a media manager.
- A run time used to execute and host Android applications, including the Dalvik Virtual Machine (VM) and the core libraries that provide Android-specific functionality. The run time is designed to be small and efficient for use on mobile devices.
- An application framework that agnostically exposes system services to the application layer, including the window manager and location manager, databases, telephony, and sensors.
- A user interface framework used to host and launch applications.
- A set of core pre-installed applications.
- A software development kit (SDK) used to create applications, including the related tools, plug-ins, and documentation.
- Application program interface APIs provides access to all the underlying services, features, and hardware

3- The Android Software Architecture

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



Android S/W Stack – Application

All applications are written using the Java language and located at the top layer. The developer can write an application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc. Android provides a set of core applications:

- Email Client
- SMS Program
- Calendar
- Maps
- Browser
- Contacts
- Etc

Android S/W Stack – App Framework

- The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.
- Most of the application framework accesses these core libraries through the Dalvik VM, the gateway to the Android Platform.
- The application framework provides the classes used to create Android applications. It also provides a generic abstraction for hardware access and manages the user interface and application resources. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more.

Android S/W Stack – Libraries

In Android stack, the libraries including a set of C/C++ libraries used by **components** of the Android system and exposed to developers through the Android **application framework**. These library include:

- A media library for playback of audio and video media
- A surface manager to provide display management
- Graphics libraries that include SGL and OpenGL for 2D and 3D graphics
- SQLite for native database support
- SSL and WebKit for integrated web browser and Internet security.
- SQLite database which is a useful repository for storage and sharing of application data Browser engine.
- Surface manager – Handling UI Windows
- Interface through Java

Android S/W Stack – Android runtime

- This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android.
- The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language.
- The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.
- The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Android S/W Stack – Linux Kernel

At the bottom of the layers is Linux (Core services) - are handled by a Linux 2.6 kernel. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc. The kernel also provides an abstraction layer between the hardware and the rest of the S/W stack.

- Relying on Linux Kernel 2.6 for core system services
- Memory and Process Management
- Network Stack
- Driver Model
- Security

4- The Android SDK

The Android SDK (software development kit) is a set of development tools used to develop applications for the Android platform that has become Apple's biggest rival in the smartphone space. The Android SDK includes the following:

Required libraries.

Debugger.

An emulator.

Relevant documentation for the Android application program interfaces (APIs).

Sample source code.

Tutorials for the Android OS

- **The Android APIs —** The core of the SDK is the Android API libraries that provide developer access to the Android stack. These are the same libraries that Google uses to create native Android applications.
- **Development tools —** The SDK includes several development tools that let you compile and debug your applications so that you can turn Android source code into executable applications.
- **The Android Virtual Device Manager and emulator —** The Android emulator is a fully interactive mobile device emulator featuring several alternative skins. The emulator runs within an Android Virtual Device (AVD) that simulates a device hardware configuration. Using the emulator you can see how your applications will look and behave on a real Android device.
- **Full documentation —** The SDK includes extensive code-level reference information detailing exactly what's included in each package and class and how to use them. In addition to the code documentation, Android's reference documentation and developer guide explains how to get started, gives detailed explanations of the fundamentals behind Android development, highlights best practices, and provides deep-dives into framework topics.

- **Sample code —** The Android SDK includes a selection of sample applications that demonstrate some of the possibilities available with Android, as well as simple programs that highlight how to use individual API features.

- **Online support —** Android has rapidly generated a vibrant developer community. The Google Groups (http://developer.android.com/resources/community-groups.html#ApplicationDeveloperLists) are active forums of Android developers with regular input from the Android engineering and developer relations teams at Google. Stack Overflow (www.stackoverflow.com/questions/tagged/android) is also a hugely popular destination for Android questions and a great place to find answers to beginner questions.

5- Android Development Framework

- Android applications normally are written using Java or Kotlin as the programming language but executed by means of a custom VM called "*Dalvik"*, rather than a traditional Java VM.

  (Running **virtual machines** has been a standard way to test apps and run virtual operating systems for a long time. Computers are able to run full **virtual machines** with simulated hardware specs and more).

  Dalvik is a discontinued process virtual machine (VM) in Android operating system that executes applications written for Android.

  The Dalvik VM executes Dalvik executable files, a format optimized to ensure minimal memory footprint. You create ".dex" executables by transforming Java language compiled classes using the tools supplied within the SDK

- Programs for Android are commonly written in Java and compiled to bytecode for the Java virtual machine, which is then translated to Dalvik bytecode and stored in .dex (*Dalvik EXecutable*) and .odex (*Optimized Dalvik EXecutable*) files; related terms *odex* and *de-odex* are associated with respective bytecode conversions. The compact Dalvik Executable format is designed for systems that are constrained in terms of memory and processor speed

- Dalvik and the Android run time sit on top of a Linux kernel that handles low-level hardware interaction, including drivers and memory management, while a set of APIs provides access to all the underlying services, features, and hardware.
- Each Android application runs in a separate process within its own Dalvik instance, relinquishing all responsibility for memory and process management to the Android run time, which stops and kills processes as necessary to manage resources.

6- Android Applications

Once developed, Android applications can be packaged easily and sold out either through a store such as **Google Play**, **SlideME**, **Opera Mobile Store**, **Mobango**, **F-droid** and the **Amazon Appstore**.

- Application components are the essential building blocks of an Android application. These components are coupled by the application manifest file **AndroidManifest.xml** that describes each component of the application and how they interact
- Written in Java.
- Good separation (and corresponding security) from other applications:
  - Each application runs in its own process
  - Each process has its own separate VM (Virtual Machine)

Each application is assigned a unique Linux user ID – by default files of that application are only visible

7- Application Fundamentals

- Android apps can be written using Kotlin, Java, and C++ languages. The Android SDK tools compile your code along with any data and resource files into an APK, an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and is the file that Android-powered devices use to install the app.

- This file is the vehicle for distributing the application and installing it on mobile devices; it's the file users download to their devices. All the code in a single .apk file is considered to be one *application*.
- By default, every application runs in its own Linux process. Android starts the process when any of the application's code needs to be executed, and shuts down the process when it's no longer needed and system resources are required by other applications.
- Each process has its own virtual machine (VM), so application code runs in isolation from the code of all other applications.
- By default, each application is assigned a unique Linux user ID. Permissions are set so that the application's files are visible only to that user and only to the application itself — although there are ways to export them to other applications as well.
- It's possible to arrange for two applications to share the same user ID, in which case they will be able to see each other's files. To conserve system resources, applications with the same ID can also arrange to run in the same Linux process, sharing the same VM.

8- Application Components

There are following four main components that can be used within an Android application:

Components	Description
Activities	They they dictate the UI and handle the user interaction to the smartphone screen
Services	They handle background processing associated with an application.
Broadcast Receivers	They handle communication between Android OS and applications.
Content Providers	They handle data and database management issues.

Activity: An activity represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity,

then one of them should be marked as the activity that is presented when the application is launched.

Service: A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

Broadcast Receivers: simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. The data may be stored in the file system, the database or somewhere else entirely

Additional Components

There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them. These components are:

Components	Description
Fragments	Represents a behavior or a portion of user interface in an Activity.
Views	UI elements that are drawn onscreen including buttons, lists forms etc.
Layouts	View hierarchies that control screen format and appearance of the views.
Intents	Messages wiring components together.
Resources	External elements, such as strings, constants and drawables pictures.
Manifest	Configuration file for the application.

Fragment in Android is a component which can be used over an activity to define an independent modular UI component attached to the activity. It functions independently, but as it is linked to the Activity, when an activity is destroyed, the fragment also gets destroyed.

Generally, fragments are used to create multi-pane UI in Android apps.

Intents: An Intent is a messaging object that you can use to request an action from an app component. An Intent is basically an intention to do an action. It's a way to communicate between Android components to request an action from a component, by different components.

It's like a message that Android listens for and then react accordingly by identifying and invoking the app's appropriate component (like an Activity, Service, Content Provider, etc.). It can be within that same app or some other app as well.

In Android, there are two types of Intents:

1. Explicit Intents
2. Implicit Intents

Explicit Intents

When you explicitly define which Android component should be opened on some user action, then you use explicit intents. You generally use an explicit intent to start a new component in your own app, because you know which exact activity or service you want to start. For example, you can start a new activity in response to a user action or start a service to download a file in the background.

Implicit Intent

When you just have to tell what action you want to perform without worrying which component will perform it, then you can use implicit intent.