1. Basics of Android

What is Android...?

- Android is an open source and Linux-based Operating System for mobile devices such as smart phones and tablet computers.
- Android was developed by the Open Handset Alliance, led by Google, and other companies.

First Release

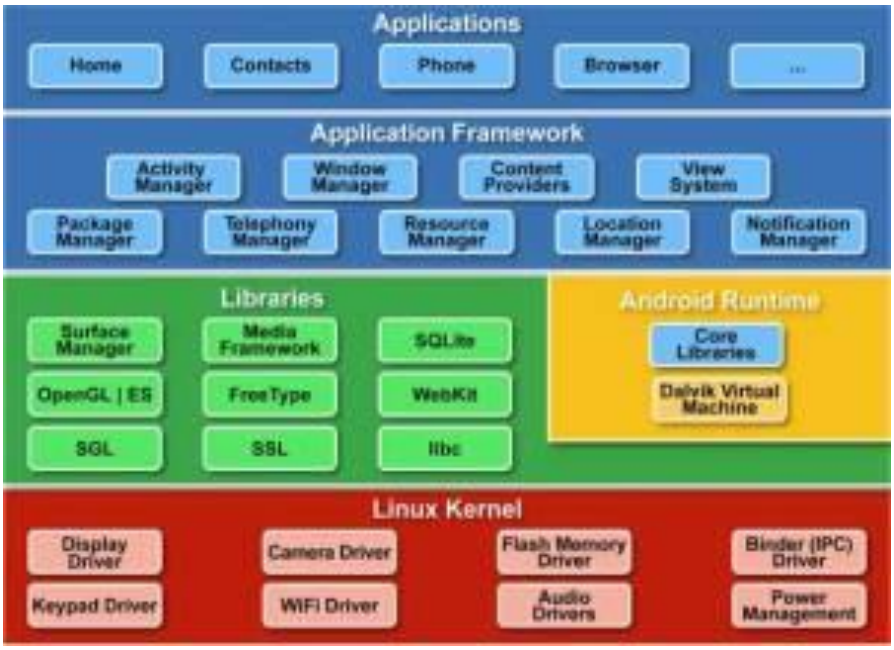
- The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007.
- The first commercial version, Android 1.0, was released in September 2008.
- The first android phone was launched by HTC on October 2008(HTC Dream – T-Mobile G1 in USA and some parts of Europe).

Application

- Set of Programs with dedicated functionality is known as Application
- Two types of Applications based on provided interface
    1. System application - Application which helps to communicate between hardware and user application.
        - Ex: Operating Systems, Driver Software etc...
    2. User application - Applications which provides solution for common problems.
        - Ex: MS Office, Banking Applications, Mail & Message Applications, Video Players, etc...
- There are 2 types of Applications based on where it is installed
    1 Standalone application (unshared)
    2 Web Application (shared)
- Standalone Applications are present in our own computer and they are dedicated per device.
- For ex : Adobe Reader , Web Browser , Media Player etc.
- Web applications are not present in our own computer but they are present in some other computer where our own computer and that computer are “Network Connected”
- For ex : Gmail , Facebook ,Twitter etc.
- There are two types of Standalone Applications
- Desktop Applications
- Mobile Applications
- Desktop Applications: As the name implies ,they are present in our “Desktop/Computer”.
- Mobile Applications: As the name implies ,they are present in our “Smart Phone”.

- Web Applications can be of 3 Tiered(Layered) or 2 tiered
- 3-Tier architecture application: accessed using “Web Browser”
    - Layer1 - Client
    - Layer2 - Server
    - Layer3 - DB
- 2-Tier architecture application: accessed using “Mobile App”
    - Layer1 - Client[DB maintained in Client's device].
    - Layer2 - Server
- Now a days users depend more on Mobile App.

Android architecture



Linux kernel

- It has Linux Version 2.6.x for core system services and thus android handles only “Kernel” portion in Linux
- A kernel is a central component of an operating system. It acts as an interface between the user applications and the hardware.
- The main tasks of the kernel are :
    - Process management
    - Device management
    - Memory management
    - Interrupt handling
    - I/O communication
    - File system etc

Android Runtime

- For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART). ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a byte code format designed especially for Android that's optimized for minimal memory footprint.

Core Libraries- Uses the JAVA Programming Language

Dalvik Virtual Machine

- Android Application operates in its own process with the specific instance of the Dalvik virtual machine (DVM).
- The DalvikVM is Java based licenses free VM.
- It executes files in the Dalvik Executable (.dex) format.

Libraries

1. *Libc*: it is c standard lib.
2. *SSL*: Secure Socket Layer for security
3. *SGL*: 2D picture engine where SGL is "Scalable Graphics Library"
4. *OpenGL|ES*: 3D image engine
5. *Media Framework*: essential part of Android multi-media
6. *SQLite*: Embedded database
7. *Web Kit*: Kernel of web browser
8. *Free Type*: Bitmap and Vector
9. *Surface Manager*: Manage different windows for different applications

Application framework

- A rich and extensible View System you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all apps to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of apps and provides a common navigation back stack
- Content Providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

Application

- Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more.

Android Studio project structure

Android View

manifests/

- AndroidManifest.xml is one of the most important file in the Android project structure.
- It contains information of the package, including components of the application
- It is responsible to protect the application to access any protected parts by providing the permissions
- It also declares the android api that the application is going to use

java/

- The java folder contains the Java source code files of the application organized into packages.

res/

- Res folder is where all the external resources for the application are stored.
- **Drawable:** The folders are to provide alternative image resources to specific screen configurations.
- **Layout:** It contains XML files that define the User Interface of the application
- **Mipmap:** The mipmap folder is used for placing the app icons only..
- **Values :** XML files that define simple values such as strings, arrays, integers, dimensions, colors, styles etc. are placed in this folder

Gradle Scripts

- Gradle scripts are used to automate tasks.
- For the most part, Android Studio performs application builds in the background without any intervention from the developer.
- This build process is handled using the Gradle system,

Project View

.idea/

- In this folder the project specific metadata is stored by Android Studio.

Project Module(app)

- This is the actual project folder where the application code resides. The application folder has following sub directories

- **build** : This has all the complete output of the make process i.e. classes.dex, compiled classes and resources, etc. The important part is that the R.java is found here under build/generated/source/r/…/R.java
- **libs** : This is a commonly seen folder in eclipse and android studio, which optionally can hold the libraries or .jar files
- **src** : The src folder can have both application code and android unit test script. You will find two folders named “androidTest” and “main” correspond to src folder.
- **Gradle**: This is where the gradle build systems jar wrapper is found. This jar is how Android Studio communicates with gradle installed in Windows/MAC.

External Libraries

- This is not actually a folder but a place where Referenced Libraries and information on targeted platform SDK are shown.

What is R.java?

- Android R.java is an auto-generated file by **aapt** (Android Asset Packaging Tool) that contains resource IDs for all the resources of res/ directory.
- On creating any component in the activity_main.xml file, id for the corresponding component is automatically created in this file.
- This id can be used in the activity source file to perform any action on the component.

Build Process

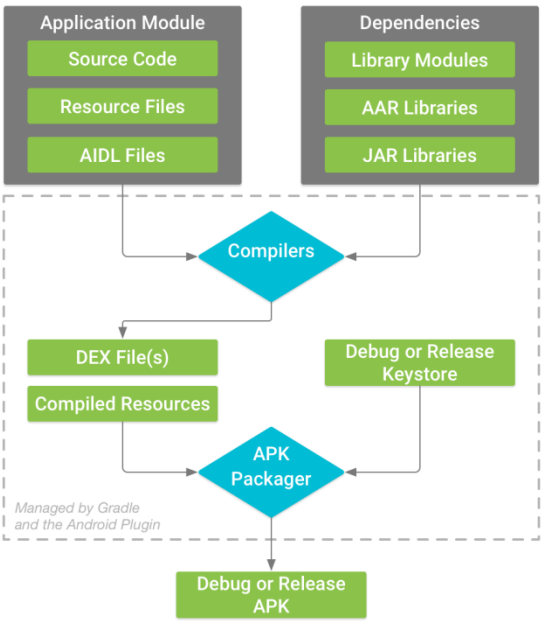
AIDL
Android Interface Defininatoin Language

AAR
Android Archive

JAR
Java Archive

DEX
Dalvik Executable

APK
Android Application Package



The build process for a typical Android app module is follows these general steps:

- The compilers convert your source code into DEX (Dalvik Executable) files, which include the bytecode that runs on Android devices, and everything else into compiled resources.
- The APK Packager combines the DEX files and compiled resources into a single APK. Before your app can be installed and deployed onto an Android device, the APK must be signed.
- The APK Packager signs your APK using either the debug or release keystore:
    - If you are building a debug version of your app, that is, an app you intend only for testing and profiling, the packager signs your app with the debug keystore.
    - Android Studio automatically configures new projects with a debug keystore.
    - If you are building a release version of your app that you intend to release externally, the packager signs your app with the release keystore.
- Before generating your final APK, the packager uses the zipalign tool to optimize your app to use less memory when running on a device.
- At the end of the build process, you have either a debug APK or release APK of your app that you can use to deploy, test, or release to external users.

Application Components

- Application components are the essential **building blocks of an Android application.**
- There are four different types of application components.
    1. Activities
    2. Services
    3. Broadcast receivers
    4. Content providers

Activities

- 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.
- Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others.

Services

- A service is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- A service does not provide a user interface.

- 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

- A broadcast receiver is a component that responds to system-wide broadcast announcements.
- Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.
- Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use.
- Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.

Content providers

- A content provider manages a shared set of application data.
- You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access.
- Through the content provider, other applications can query or even modify the data (if the content provider allows it).
- For example, the Android system provides a content provider that manages the user's contact information.

Activity Life Cycle

- Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class.
- The android Activity is the subclass of ContextThemeWrapper class.

onCreate()

- You must implement this callback, which fires when the system first creates the activity.
- On activity creation, the activity enters the Created state.
- In the onCreate() method, you perform basic application startup logic that should happen only once for the entire life of the activity.

onStart()

- When the activity enters the Started state, the system invokes this callback.

- The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.

onResume()

- When the activity enters the Resumed state, it comes to the foreground, and then 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 an event might be, for instance, receiving a phone call, the user's navigating to another activity, or the device screen's 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).
- Use the onPause() method to pause operations such animations and music playback that should not continue while the Activity is in the Paused state, and that you expect to resume shortly.
- There are several reasons why an activity may enter this state. For example: Some event interrupts app execution, as described in the onResume() section. This is the most common case.

onStop()

- When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the onStop() callback.
- This may occur, for example, when a newly launched activity covers the entire screen.
- The system may also call onStop() when the activity has finished running, and is about to be terminated.
- In the onStop() method, the app should release almost all resources that aren't needed while the user is not using it.

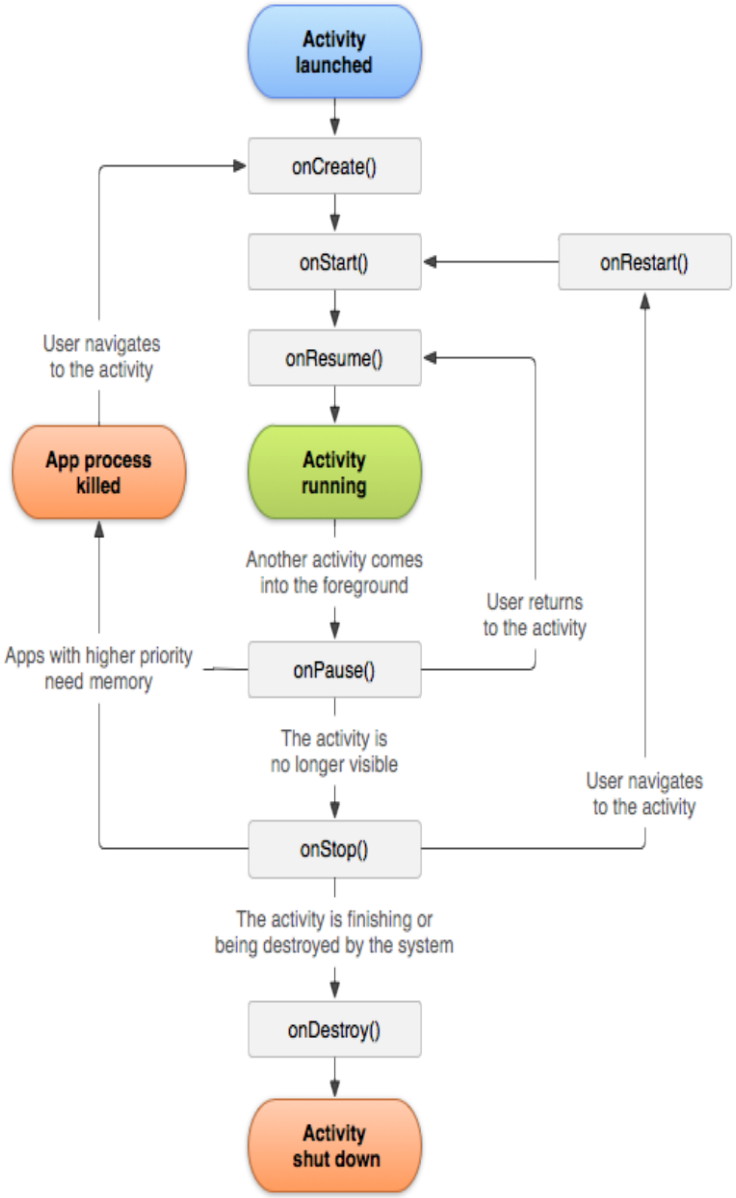
onRestart ()

- This callback method called after onStop() when the current activity is being re-displayed to the user (the user has navigated back to it).
- It will be followed by onStart() and then onResume().

onDestroy()

- Called before the activity is destroyed. This is the final call that the activity receives.

- The system either invokes this callback because the activity is finishing due to someone's calling finish(), or because the system is
- temporarily destroying the process containing the activity to save space.
- The system may also call this method when an orientation change occurs, and then immediately call onCreate() to recreate the process (and the components that it contains)
- in the new orientation.
- The onDestroy() callback releases all resources that have not yet been released by earlier callbacks such as onStop().



Layout, View, ViewGroup

- A layout defines the visual structure for a user interface, such as the UI for an activity or app widget.
- A View is an object that draws something on the screen that the user can interact with.
- A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.
- You define your layout in an XML file which offers a human-readable.
- The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior.
- In general, the XML vocabulary for declaring UI elements closely follows the structure and naming of the classes and methods, where element names correspond to class names and attribute names correspond to methods.

Android units of measurements px, in, mm, dp, dip and sp

- Web designers have always designed web pages in pixels.
- The problem with pixels is that with increased screen resolutions the,
- with more dots for inch, the display content such as text and images become smaller and smaller.
- And with so many devices today with varied resolutions it becomes harder for designers to design the best interface suited
- for all of them. Here comes the dip and sp to rescue for android programmers.

Android supports the following measurements:

- px (Pixels) - Actual pixels or dots on the screen.
- in (Inches) - Physical size of the screen in inches.
- mm (Millimeters) - Physical size of the screen in millimeters.
- pt (Points) - 1/72 of an inch.
- dp (Density-independent Pixels) - An abstract unit that is based on the physical density of the screen.
    - These units are relative to a 160 dpi screen, so one dp is one pixel on a 160 dpi screen.
    - The ratio of dp-to-pixel will change with the screen density, but not necessarily in direct proportion. "dip" and "dp" are same.
- sp (Scale-independent Pixels) - Similar to dp unit, but also scaled by the user's font size preference.

Note: Always use sp for font sizes and dp for everything else.