Unit-I

1. Mobile Applications: -

A mobile application (also called a mobile app) is a type of application designed to run on a mobile device, which can be a smartphone or tablet computer. Even if apps are usually small software units with limited function, they still manage to provide users with quality services and experiences. Mobile apps are designed with consideration for the demands and constraints of the devices and also to take advantage of any specialized capabilities they have.

Apps are divided into two broad categories: native apps and web apps. Native apps are built for a specific mobile operating system, usually iOS or Android. Native apps enjoy better performance and a more finely-tuned user interface (UI), and usually need to pass a much stricter development and quality assurance process before they are released.

There are several types of apps currently available.

- **Gaming apps:** The equivalent of computer video games, they are among the most popular types of apps. They account for one-third of all app downloads and three-fourths of all consumer spending
- **Productivity apps:** These focus on improving business efficiency by easing various tasks such as sending emails, tracking work progress, booking hotels, and much more.
- **Lifestyle and entertainment apps:** Increasingly popular, these encompass many aspects of personal lifestyle and socialization such as dating, communicating on social media, as well as sharing (and watching) videos. Some of the most widely known apps such as Netflix, Facebook fall into this category.

Characteristics of Mobile Applications: -

1. **High and Consistent Performance:** High and consistent performance should be the first priority in developing the app. It should be well-tested and consistent in its performance even under extreme conditions. The ideal mobile app in terms of performance is the one that consumes minimum CPU and battery power, and does not require huge storage space.
2. **Unique, Appealing and Easy to Use:** Mobile app should be memorable and stand out from the rest. The mobile app needs to be amazing, pretty and appealing to the target audience. It must be uniquely packaged and fully branded.
3. **Platform Appropriate:** When designing and developing a mobile app, make sure that it is appropriate for a handheld mobile device. Remember that mobile technology has many different platforms, so ensure that you have an appropriate design for all brands and devices.
4. **Responsive Customer Support and Regular Update:** Expect that not all users of the app will be tech savvy. There will always be somebody who will need help on downloading and using the app. It is important that you respond to queries and requests promptly.
5. **Affordable:** The app may be appealing and beautiful, but if it is not affordable, it is doomed to fail. So, offer both a free version and full-feature paid version. However, it is still important to have the full-feature paid version affordable.
6. **Individualization or Personalisation:** Creating individualized content based on personalized context or usage is another feature. Everyone wish that my application should fit my needs and behave like what I want to do. This specific feature not only

covers personalized content but wants to control over shared and store data for more actions.

7. **Security:** There are several aspects of security like transferred data over network via carrier network. Some applications sync data with online, web apps, so the storage of the information on server must be secure. Another critical security breach can be the mobile device itself as I do not want anybody to play with my mobile phone.

Benefits of Mobile Applications: -

- It utilizes the smartphone's existing features like shopping, taking a photo or using GPS location.
- If you have an app, you can improve user experience. Because smartphone market is growing exponentially. It would be an incentive to let users accomplish a specific action via your app.
- It gives more value to customers, because more users prefer mobile devices to desktops
- The best part is generating revenue. There are many ways to let your app gain you like paying a fee to download.
- Mobile apps make your business easily accessible to clients via app store search. Of course, it needs time and effort to be listed on search.
- Mobile apps increase the brand awareness by letting customers easily post, share, or tweet about your services.

2. Basics of Android: -

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, led by Google, and other companies. **Android** is a complete set of software for mobile devices such as tablet computers, notebooks, smartphones, electronic book readers, set-top boxes etc. It contains a **Linux-based Operating System**, **middleware** and **key mobile applications**. It can be thought of as a mobile operating system. But it is not limited to mobile only. It is currently used in various devices such as mobiles, tablets, televisions etc.

Open Handset Alliance - It's a consortium of 84 companies such as google, Samsung, AKM, synaptics, KDDI, Garmin, Teleca, eBay, Intel etc. It was established on 5th November, 2007, led by Google. It is committed to advance open standards, provide services and deploy handsets using the Android Platform.

FEATURES OF ANDROID

- Background Wi-Fi location still runs even when
- Wi-Fi is turned off
- Developer logging and analyzing enhancements
- It is optimized for mobile devices.
- It enables reuse and replacement of components.
- Java support ,media support, multi touch, video
- calling,multi tasking ,voice based features, screen capture, camera, Bluetooth, GPS, compass and accelerometer,3G.

Advantages:

•The ability for anyone to customize the Google Android platform
•It gives you better notification.
•It lets you choose your hardware.

•It has better app market(1,80,000 application)
•A more mature platform
• With the support of many applications, the user can change the screen display.
•With Google chrome you can open many window at once.
• Supports all Google services: Android operating system supports all of Google services ranging from Gmail to Google reader. all Google services can you have with one operating system, namely Android.

DIS-ADVANTAGES:

- Android Market is less control of the manager, sometimes there are malware.
- Wasteful Batteries, This is because the OS is a lot of "process" in the background causing the battery quickly drains.
- Sometimes slow device company issued an official version of Android your own .
- Extremely inconsistence in design among apps.
- Very unstable and often hang or crash.

3. Importance of Mobile Application: -

The importance of mobile phones in our everyday life and activities is undeniably unending. This is so because there is ongoing tremendous transformation in that mobile phones are no longer the ordinary communication device it used to be. It has become the colossal point of attention for individuals and businesses alike, courtesy of the various incredible features and opportunities that mobile phones offer. The cumulative progress of mobile technology, the availability and access to high speed internet and the remarkable communicative interface in these devices results into a whole level of new and innovative experience mobile computing. This is made possible through the development of mobile applications (mobile apps). Today, the availability of mobile apps is on the increase such that it is produce a noticeable change in the way humans feel and experience computing. Few years ago, in other to access the internet, check and read mails, one had to use the computer but today this has changed because computing is now carried everywhere in mobile phones. Imagine buying a train ticket on the go, this is something our ancestors never imagined or did. Imagine not going to the bank but still transfer money to family and friends. All thanks to app developers and top app development companies. No matter which, they have come to the rescue enabling easy life.

Mobile apps are applications designed to perform a specific task. Time consumption has been reduced with the popularity of mobile apps and now goods and services are at fingertip. The services that mobile apps and smartphone provide are plenty.

1.Onilne Shopping: is one of the most important advantages of mobile apps, with which you can get your favourite clothes, gadgets, accessories and raw materials online without visiting shops. Customers can view, buy and search their favourite commodity of size, shape and colour. Online shopping is a kind of business where you act both as seller and buyer.

2. Food Delivery and Taxi Apps: If you have an app installed, you can get cooked food from your favourite hotel which would be delivered at your doorstep at reasonable price. Many apps today provide their customers taxi services at reasonable rates too.

3. Banking Apps: Banking services that mobile apps provide their customers are numerous. Customers can view debits and credits of their personal account, transfer money without the barrier of time and distance, view current account balance etc. Today central government and state government are trying their maximum to build a cashless economy all over the country, along with the banks they have come up with several mobile apps to attain their goal of

building a cashless economy. Customers can pay all their bills online, especially the electricity bills, phone bills, water bills etc.., without standing in long queues.

4.E-Tickets: Without standing in long queues you can check the availability of seats and book tickets for your movie, bus, train and aeroplane. Today we have separate apps defined for each service and you don't need to carry any hard copy tickets anymore. Your tickets will be in the form of an email or a text message.

5. Reading and Educational Apps: Due to busy schedule you may not get time to read daily Newspapers. But if you have a News reading app installed in your phones it would keep you updated with latest News. Many people love reading books at their leisure times but buying them may not be affordable and carrying them long distance is difficult. But now you can buy books online using mobile apps and can carry them long distance. Educational apps provide its customers with online classes on specific topics and conduct tests depending on the topic.

4. Scope of Mobile Apps:

Mobile applications can be developed across a range of platforms--from iPhone, Android, iPad to Windows mobile amongst others. The scope of mobile application development has grown so explosively not only because of the number of development platforms, but because of the nearly limitless range of ways it enhances our lives. Since smartphone sales alone have long eclipsed personal computer sales, the paradigm has shifted dramatically. By the end of 2014 it has been predicted that there will be nearly twice as many smartphones sold as compared to computers. This isn't even including tablets, wearable technology and other mobile devices such as the Kindle. The rapid nature of this paradigm shift and explosive growth in scope of mobile application development has left many companies grasping at straws when it comes to breaking into this market. Even seasoned mobile application developers themselves must work daily to stay ahead of the curve and ensure that their skill set meets the current market demand.

How have customer habits changed the scope of mobile application development?

Over the years, users have become much more demanding, and market competition has tightened. The app marketplace is crowded. There are roughly two million mobile apps available on both the Apple App Store and Google Play. Social apps, travel planners, business tools, fitness trackers, games, personal finance apps, musical instruments and even the 'kitchen sink' -- apps stores have it all. Mobile app development is a risky and highly competitive business because it's getting harder and harder to create an app that will stand out.

Today, only a smartphone owner knows what they want, and the only way to succeed on the market is to satisfy not just the general, but also the individual and highly personalised needs ofthe modern user.

5. Android Stack: -

```
public class Stack
 extends Vector<E>
java.lang.Object
   java.util.AbstractCollection<E>
      java.util.AbstractList<E>
         java.util.Vector<E>
```

java.util.Stack<E>

The Stack class represents a last-in-first-out (LIFO) stack of objects. It extends class `Vector` with five operations that allow a vector to be treated as a stack. The usual `push` and `pop` operations are provided, as well as a method to `peek` at the top item on the stack, a method to test for whether the stack is `empty`, and a method to `search` the stack for an item and discover how far it is from the top. When a stack is first created, it contains no items. A more complete and consistent set of LIFO stack operations is provided by the Deque interface and its implementations, which should be used in preference to this class. For example:

```
Deque<Integer> stack = new ArrayDeque<Integer>();
```

Public Constructors: Stack() – creates an empty stack.

Public methods	
boolean	empty (): Tests if this stack is empty.
E	peek (): Looks at the object at the top of this stack without removing it from the stack.
E	pop (): Removes the object at the top of this stack and returns that object as the value of this function.
E	push (E item): Pushes an item onto the top of this stack.
Int	search (Object o): Returns the 1-based position where an object is on this stack.

Peek – public E peek

Returns E The object at the top of this stack (last item of the vector object).
Throws EmptyStackException If this stack is empty.

Pop – public E pop

Returns E The object at the top of this stack (last item of the vector object).
Throws EmptyStackException If this stack is empty.

Push – public E push

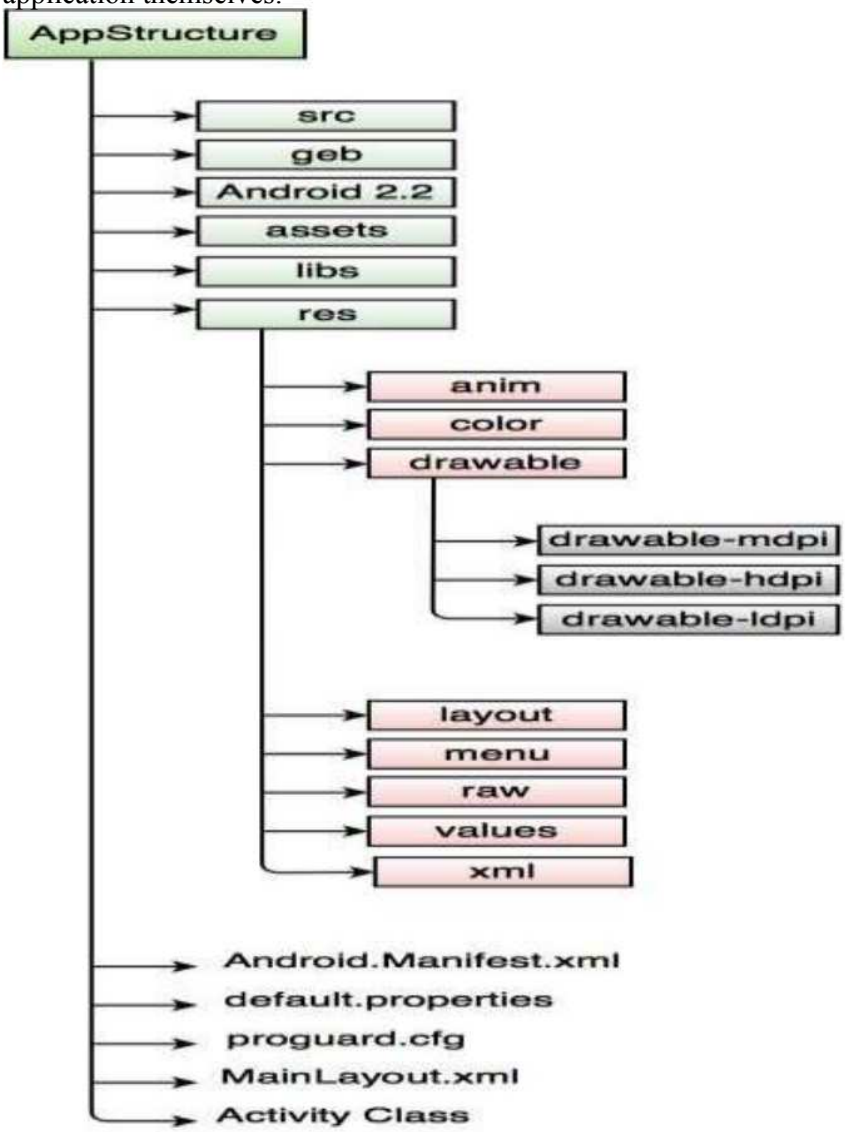
Parameters item E The item to be pushed onto this stack.
Return E The item argument.

Search – public int search (Object o): Returns the 1-based position where an object is on this stack. If the object `o` occurs as an item in this stack, this method returns the distance from the top of the stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance `1`. The `equals` method is used to compare `o` to the items in this stack.

Parameters o Object: the desired object.
Returns int The 1-based position from the top of the stack where the object is located; the return value -1 indicates that the object is not on the stack.

6. Android Application Structure: -

Android uses packages not only to arrange the code in an application but to manage the application themselves.



Every Android project contains several folders, like:

Folder Name	Description
Src	The 'src' stands for **Source Code.** It contains the Java Source files.
Gen	The 'gen' stands for **Generated Java Library.** This library is for Android internal use only.
Android 2.2	The Android Framework Library is stored here.
Assets	It is used to store raw asset files.
Libs	It contains private libraries.
Res	The 'res' stands for **Resource file.** It can store resource files such as pictures, XML files, etc. It contains some additional folders such as

	Drawable, Layout and Values. **anim:** It is used for XML files that are compiled into animation objects. **color:** It is used for XML files that describe colors. **drawable:** It is used to store various graphics files. In Android project structure, **there are three types of drawable folders,** 1. drawable-mdpi 2. drawable-hdpi 3. drawable-ldpi The above drawable folders are required in order to adapt to different screen resolutions. **layout:** It is used for placing the XML layout files, which defines how various Android objects such as textbox, buttons, etc. are organized on the screen. **menu:** It is used for defining the XML files in the application menu. **raw:** The 'raw' stands for **Raw Asset Files.** These files are referenced from the application using a resource identifier in the R class. **For example,** good place for media is MP3 or Ogg files. **values:** It is used for XML files which stores various string values, such as titles, labels, etc. **xml:** It is used for configuring the application components.
AndroidManifest.xml	This file indicates the Android definition file. This file contains the information about the Android application such as minimum Android version, permission to access Android device capabilities such as Internet access permission, phone permission etc.
default.properties	This file contains the project settings, such as build the target. Do not edit this file manually. It should be maintained in a Source Revision Control System.
Proguard.cfg	This file defines how ProGuard optimizes and makes your code unclear.
MainLayout.xml	This file describes the layout of the page. So, all the components such as textboxes, labels, radio buttons, etc. are displaying on the application screen.

Activity class	The application occupies the entire device screen which needs at least one class inherits from the Activity class. OnCreate() method initiates the application and loads the layout page.

7. Android Emulator: -

The **Android emulator** is an **Android Virtual Device (AVD),** which represents a specific Android device.

An Android emulator is a tool that creates virtual Android devices (with software and hardware) on your computer.

- It is a program (a process that runs on your computer's operating system).
- It works by mimicking the guest device's architecture (more on that in a bit).

The Android emulator provides almost all the functionality of a real device. The Android emulator comes with predefined configurations for several Android phones, Wear OS, tablet, Android TV devices.

How an Emulator works:

- **Quick Emulator (QEMU)** - QEMU–shorthand for Quick EMUlator. It's an open-source and incredibly versatile tool. It can run on a large variety of host (workstation) CPUs/OSs and emulate an even larger range of guest CPUs/OSs. QEMU powers most Android emulators (including the one by Android Developer Studio).

  It mimics guest device hardware. Then it translates the Application Binary Interface (ABI) of the guest device to match that of the host device. You equip this with an OS and run it like a program on your computer.

- **Hypervisor** - Before 2017, Android Developer Studio's emulator had to translate Android's ARM architecture to match the Intel/AMD architectures commonly used in PCs. With the release of version 25.3.0, Developer Studio upgraded their emulator to support hardware-assisted virtualization.

  When the guest and the host devices have same instruction architecture QEMU skips the 'binary translation' part and runs the guest device directly on the host CPU. This is called hardware-assisted virtualization.

Capabilities of Emulator:

Data transfer is faster on a virtual device (than a physical device connected via USB). The drag-and-drop file upload lets you place .apk files from your computer to the virtual mobile device. It's particularly great when developers need to quickly test apps under context.

The emulator is also pretty useful when you're working with physical sensors like the accelerometer. If you were testing a specific app feature that relies on the sensors, it'll be easier to configure the settings through the visual, extended controls of the emulator.

Limitations of Emulator:

1. The most popular chipset for Android smartphones out there is ARM v7a. Most PCs/laptops run on Intel (x86). Recall that guest and host CPU architectures need to match for faster emulation. Basically, without a computer equipped with an

ARM processor, **you're stuck with poor emulation of most of the commercially-available Android devices.**

2. The AVD Manager creates separate directories to store each virtual device's user data, SD card data, and cache. A single virtual device can take as much as 3.5GB of your disk space. **Over time, a library of virtual devices will clam up your workstation.**
3. Virtual devices' performance is affected by that of your workstation. **The emulator will crash and burn if you don't have enough free disk space at launch.**
4. Android emulator isn't reliable when it comes to understanding app interactions with the native device environment. For instance, you'd never know:
   - Which background processes your app runs
   - How front-end appears in different brightness levels
   - How the app responds to a complete range of touch-gestures

8. Android SDK: -

The Android SDK (software development kit) is a set of development tools used to develop applications for Android platform. SDK provides a selection of tools required to build Android apps or to ensure the process goes as smoothly as possible. tools required to build Android apps or to ensure the process goes as smoothly as possible. The Android SDK comprises all the tools necessary to code programs from scratch and even test them. These tools provide a smooth flow of the development process from developing and debugging, through to packaging. The Android SDK is compatible with Windows, macOS, and Linux, so you can develop on any of those platforms.
The Android SDK includes the following:

Components of the Android SDK: The Android SDK can be broken down into several components. These include:

A. **Platform-Tools** - The Platform tools are more specifically suited to the version of Android that you want to target. Generally, it is best to install the latest Platform tools, which will be installed by default. After first installation though, you need to keep your Platform-tools constantly updated. The tools should be backwards compatible, meaning that you will still be able to support older versions of Android.

B. **Build-Tools** - As the name suggests, these are also needed to build your Android apps. This includes the zipalign tool for instance, which optimizes the app to use minimal memory when running prior to generating the final APK, and the apksigner which signs the APK for subsequent verification.

C. **SDK-Tools** - You will need these tools regardless of which version of Android you are targeting. These are what will actually create the APK – turning your Java program into an Android app that can be launched on a phone. These include a number of build tools, debugging tools, and image tools. An example is DDMS, which is what lets us use the Android Device Monitor to check the status of an Android device.

D. **The Android Debug Bridge (ADB)** - The ADB is a program that allows you to communicate with any Android device. It relies on Platform-tools in order to understand the Android version that is being used on said device and hence it is included in the Platform-tools package. You can use ADB to access shell tools such as logcat, to query your device ID or even to install apps.

E. **Android Emulator** -The Android emulator is what lets you test and monitor apps on a PC, without necessarily needing to have a device available. To use this, you also get an Android system image designed to run on PC hardware. You'll use the Android Virtual Device manager in order to choose which version of Android you want to emulate, along with the device specifications (screen size, performance etc.).

F. **Google APIs:**
Google provides a number of exclusive Google APIs to make developing your app easier. They also offer a system image for the emulator so you can test your app using the Google APIs.

Benefits of using an SDK:

- It provides a set of tools that allows developers to create a unique interface for the end-users of an app.
- The developers are not required to perform basic tasks related to standard app features, such as data storage, location, user authorization, geofencing, and more.
- It provides developers with robust functionalities such as code reuse, error handling, and consistent performance.
- It ensures that the API provided is implemented correctly.
- It also allows easier upgrade paths and the ability to handle deprecations for specific lower-level APIs.

9. Overview of Android Studio: -

Android Studio is the official integrated development environment (IDE) for Android application development. It is based on the IntelliJ IDEA, a Java integrated development environment for software, and incorporates its code editing and developer tools. To support application development within the Android operating system, Android Studio uses a Gradle-based build system, emulator, code templates, and GitHub integration. Every project in Android Studio has one or more modalities with source code and resource files. These modalities include Android app modules, Library modules, and Google App Engine modules.

Advantages of Android Studio:

A. **Code and Iterate faster than ever** –

- **Apply Changes** ➡Android Studio's Apply Changes feature lets you push code and resource changes to your running app without restarting your app—and, in some cases, without restarting the current activity.
- **Intelligent Code Editor** ➡The code editor helps you write better code, work faster, and be more productive by offering advanced code completion, refactoring, and code analysis. As you type, Android Studio provides suggestions in a dropdown list.
- **Fast and Feature-emulator** ➡ The Android Emulator installs and starts your apps faster than a real device and allows you to prototype and test your app on various Android device configurations: phones, tablets, Android Wear, and Android TV devices.

B. **Code with confidence** –

- **Code Templates and sample app** ➡ Android Studio includes project and code templates that make it easy to add well-established patterns such as a navigation drawer and view pager. You can start with a code template or even right-click an API in the editor and select *Find Sample Code* to search for examples.
- **Testing tools and frameworks** ➡Android Studio provides extensive tools to help you test your Android apps with JUnit 4 and functional UI test frameworks. With Espresso Test Recorder, you can generate UI test code by recording your interactions with the app on a device or emulator.
- **Lintelligence** ➡ Android Studio provides a robust static analysis framework and includes over 365 different lint checks across the entirety of your app. Additionally, it provides several quick fixes that help you address issues in various categories, such as performance, security, and correctness, with a single click.

C. **Eliminate tiresome tasks** –

- **Layout Editor** ➡ When working with XML layout files, Android Studio provides a drag-and-drop visual editor that makes it easier than ever to create a new layout. The Layout Editor was built in unison with the ConstraintLayout API, so you can quickly build a layout that adapts to different screen sizes by dragging views into place and then adding layout constraints with just a few clicks.
- **APK analyser** ➡ You can use the APK Analyzer to easily inspect the contents of your APK. It reveals the size of each component so you can identify ways to reduce the overall APK size.
- **Translations Editor**➡ The translations editor gives you a single view of all of your translated resources, making it easy to change or add translations, and to find missing translations without opening each version of the strings.xml file. It even provides a link to order translation services.

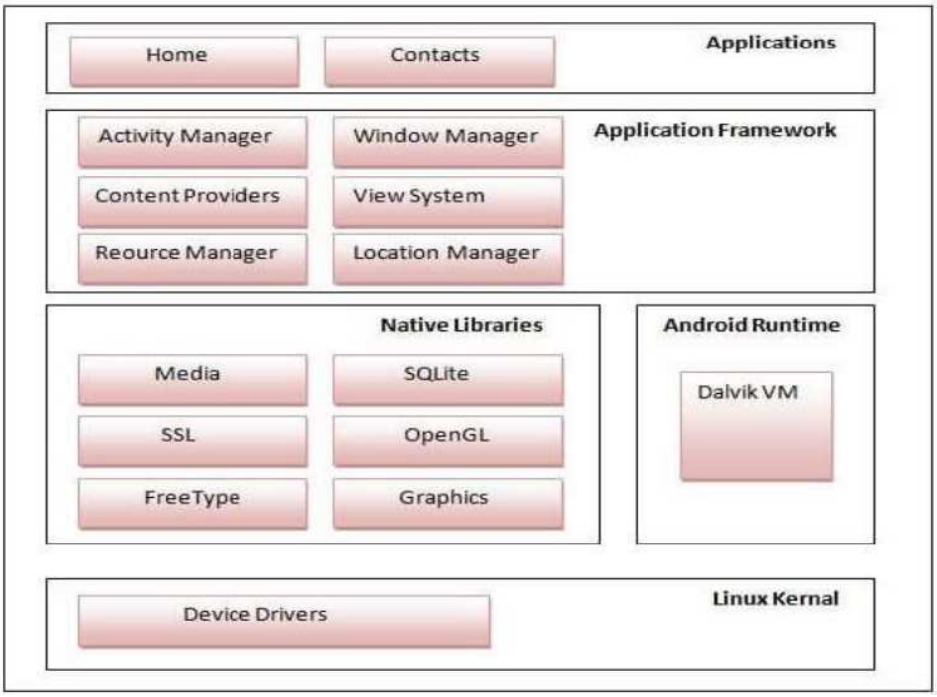
Disadvantages of Android Studio:

- Android Studio needs a very high amount of RAM and a high-end processor to run smoothly, which can't be affordable for everyone.
- Updates in Gradle files can sometimes come up with a hectic improvement in whole code, which can lead us to improve some code and consume precious time.
- Multitasking is very difficult in Android Studio due to its heavy consumption of resources.

10. Android Structure or Android Architecture: -

Android operating system is a stack of software components which is roughly divided into five sections:

1. Applications
2. Application Framework
3. native libraries (middleware),
4. Android Runtime
5. Linux kernel



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.

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. It includes different types of services activity manager, notification manager, view system, package manager etc. which are helpful for the development of our application according to the prerequisite.

Application runtime –

In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.

Platform or Native 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 –

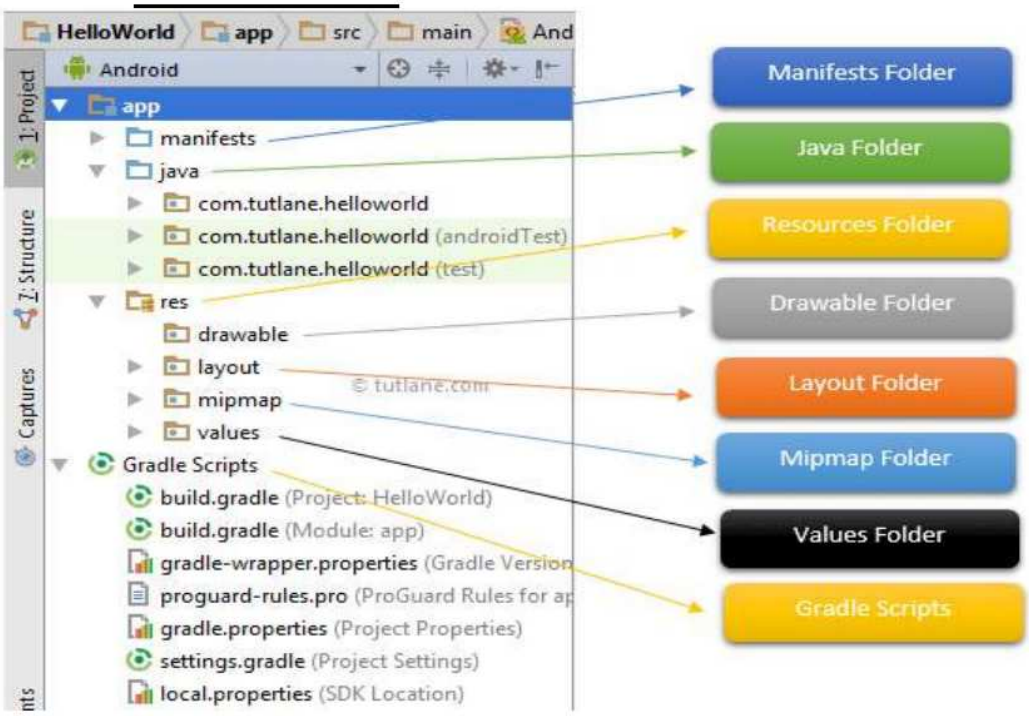
Linux Kernel is heart of the android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc. which are required during the runtime.

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:

- **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.

11. File Structure:-



- **AndroidManifest.xml**: Every project in Android includes a manifest file, which is AndroidManifest.xml, stored in the root directory of its project hierarchy. The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements. This file includes nodes for each of the Activities, Services, Content Providers and Broadcast Receiver that make the application and using Intent Filters and Permissions, determines how they co-ordinate with each other and other applications.

- **java:** The Java folder contains the Java source code files. These files are used as a controller for controlled UI (Layout file). It gets the data from the Layout file and after processing that data output will be shown in the UI layout. It works on the backend of an Android application.
- **drawable**: A Drawable folder contains resource type file (something that can be drawn). Drawable may take a variety of file like Bitmap (PNG, JPEG), Nine Patch, Vector (XML), Shape, Layers, States, Levels, and Scale.
- **layout:** A layout defines the visual structure for a user interface, such as the UI for an Android application. This folder stores Layout files that are written in XML language. You can add additional layout objects or widgets as child elements to gradually build a View hierarchy that defines your layout file.
- **mipmap:** Mipmap folder contains the Image Asset file that can be used in Android Studio application. You can generate the following icon types like Launcher icons, Action bar and tab icons, and Notification icons.
- **colors.xml:** colors.xml file contains color resources of the Android application. Different color values are identified by a unique name that can be used in the Android application program. Below is a sample colors.xml file:

```
<?xml version=”1.0” encoding="utf-8"?>
<resources>
    <color name=”
    colorPrimary”>#3F51B5</color>
    <color name=”
    colorPrimaryDark”>#303F9F</color>
    <color name=”
    colorAccent”>#FF4081</color>
</resources>
```

- **strings.xml:** The strings.xml file contains string resources of the Android application. The different string value is identified by a unique name that can be used in the Android application program. This file also stores string array by using XML language. Below is a sample colors.xml file .

```
<resources>
    <string name=”app_name”
    >Text</string>
</resources>
```

- **styles.xml:** The styles.xml file contains resources of the theme style in the Android application. This file is written in XML language. Below is a sample styles.xml file:

```
<resources>
<!-- Base application theme. -->
    <style name="AppTheme"
    parent="Theme.AppCompat.Light.DarkActionBar">
<!-- Customize your theme here. -->
    <item
    name="colorPrimary">@color/colorPrimary</item>
```

```
                <item
                name="colorPrimaryDark">@color/colorPrimaryDark<
                /item>
                <item
                name="colorAccent">@color/colorAccent</item>
                </style>
        </resources>
```

- **build.gradle (Module: app):** This defines the module-specific build configurations. Here you can add dependencies what you need in your Android application.

12. Android Virtual Device Manager: -

An Android Virtual Device (AVD) is a configuration that defines the characteristics of an Android phone, tablet, Wear OS, Android TV, or Automotive OS device that you want to simulate in the Android Emulator. The AVD Manager is an interface you can launch from Android Studio that helps you create and manage AVDs.

An AVD contains a hardware profile, system image, storage area, skin, and other properties.

Hardware Profile: The hardware profile defines the characteristics of a device as shipped from the factory. The AVD Manager comes preloaded with certain hardware profiles, such as Pixel devices, and you can define or customize the hardware profiles as needed.

System Images: A system image labelled with **Google APIs** includes access to Google Play Services. A system image labelled with the Google Play logo in the **Play Store** column includes the Google Play Store app *and* access to Google Play services, including a **Google Play** tab in the **Extended controls** dialog that provides a convenient button for updating Google Play services on the device.

Storage Area: The AVD has a dedicated storage area on your development machine. It stores the device user data, such as installed apps and settings, as well as an emulated SD card. If needed, you can use the AVD Manager to wipe user data, so the device has the same data as if it were new.

Skin: An emulator skin specifies the appearance of a device. The AVD Manager provides some predefined skins. You can also define your own, or use skins provided by third parties.

Create an AVD: Same as in Android Emulator.

Create a hardware profile: The AVD Manager provides predefined hardware profiles for common devices so you can easily add them to your AVD definitions. If you need to define a different device, you can create a new hardware profile. You can define a new hardware profile from the beginning, or copy a hardware profile as a start. The preloaded hardware profiles aren't editable.

To create a new hardware profile from the beginning:

1. In the **Select Hardware** page, click **New Hardware Profile**.
2. In the **Configure Hardware Profile** page, change the hardware profile properties as needed.
3. Click **Finish**.

To create a hardware profile starting with a copy:

1. In the **Select Hardware** page, select a hardware profile and click **Clone Device**. Or right-click a hardware profile and select **Clone**.
2. In the **Configure Hardware Profile** page, change the hardware profile properties
3. as needed.
4. Click **Finish**.

Edit existing AVDs: From the Your Virtual Devices page, you can perform the following operations on an existing AVD:

1. To edit an AVD, click **Edit this AVD** and make your changes.
2. To delete an AVD, right-click an AVD and select **Delete**. Or click Menu and select **Delete**.
3. To show the associated AVD .ini and .img files on disk, right-click an AVD and select **Show on Disk**. Or click Menu and select **Show on Disk**.
4. To view AVD configuration details that you can include in any bug reports to the Android Studio team, right-click an AVD and select **View Details**. Or click Menu and select **View Details**.

Edit existing hardware profiles: From the **Select Hardware** page, you can perform the following operations on an existing hardware profile:

1. To edit a hardware profile, select it and click **Edit Device**. Or right-click a hardware profile and select **Edit**. Next, make your changes.
2. To delete a hardware profile, right-click it and select **Delete**.

You can't edit or delete the predefined hardware profiles.

Hardware profile properties: You can specify the following properties of hardware profiles in the Configure Hardware Profile page. AVD configuration properties override hardware profile properties, and emulator properties that you set while the emulator is running override them both.

Hardware Profile Property	Description
Device Name	Name of the hardware profile. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (()), and spaces. The name of the file storing the hardware profile is derived from the hardware profile name.
Device Type Screen Size	Select one of the following: Phone/Tablet, Wear OS, Android TV, Chrome OS Device, Android Automotive The physical size of the screen, in inches, measured at the diagonal. If the size is larger than your computer screen, it’s reduced in size at launch.
Screen Resolution	Type a width and height in pixels to specify the total number of pixels on the simulated screen.

Round	Select this option if the device has a round screen, such as some Wear OS devices.
Memory: RAM	Type a RAM size for the device and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Input: Has Hardware Buttons (Back/Home/Menu)	Select this option if your device has hardware navigation buttons. Deselect it if these buttons are implemented in software only. If you select this option, the buttons won't appear on the screen. You can use the emulator side panel to "press" the buttons, in either case.
Input: Has Hardware Keyboard	Select this option if your device has a hardware keyboard. Deselect it if it doesn't. If you select this option, a keyboard won't appear on the screen. You can use your computer keyboard to send keystrokes to the emulator, in either case.
Navigation Style	Select one of the following: None - No hardware controls. Navigation is through the software. D-pad - Directional Pad support.
Supported Device States	Select one or both options: 1. Portrait - Oriented taller than wide. 2. Landscape - Oriented wider than tall.
Cameras Sensors: Accelerometer	To enable the camera, select one or both options: Back-Facing Camera - The lens faces away from the user. Front-Facing Camera - The lens faces toward the user. Select if the device has hardware that helps the device determine its orientation.
Sensors: Gyroscope	Select if the device has hardware that detects rotation or twist. In combination with an accelerometer, it can provide smoother orientation detection and support a six-axis orientation system.
Sensors: GPS	Select if the device has hardware that supports the Global Positioning System (GPS) satellite-based navigation system.
Sensors: Proximity Sensor	Select if the device has hardware that detects if the device is close to your face during a phone call to disable input from the screen.
Default Skin	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the resolution can mean that the screen is cut off, so you can't see the whole screen.

AVD properties: You can specify the following properties for AVD configurations in the Verify Configuration page. The AVD configuration specifies the interaction between the

development computer and the emulator, as well as properties you want to override in the hardware profile.

AVD Property	Description
AVD Name	Name of the AVD. The name can contain uppercase or lowercase letters, numbers from 0 to 9, periods (.), underscores (_), parentheses (()), dashes (-), and spaces. The name of the file storing the AVD configuration is derived from the AVD name.
AVD ID (Advanced)	The AVD filename is derived from the ID, and you can use the ID to refer to the AVD from the command line.
Hardware Profile	Click Change to select a different hardware profile in the **Select Hardware** page.
System Image	Click Change to select a different system image in the **System Image** page. An active internet connection is required to download a new image.
Start-up Orientation	Select one option for the initial emulator orientation: Portrait - Oriented taller than wide. Landscape - Oriented wider than tall.
Camera (Advanced)	To enable a camera, select one or both options: Front - The lens faces away from the user. Back - The lens faces toward the user. The Emulated setting produces a software-generated image, while the Webcam setting uses your development computer webcam to take a picture.
Network: Speed (Advanced)	Select a network protocol to determine the speed of data transfer: GSM - Global System for Mobile Communications HSCSD - High-Speed Circuit-Switched Data GPRS - Generic Packet Radio Service
Network: Latency (Advanced)	Select a network protocol to set how much time (delay) it takes for the protocol to transfer a data packet from one point to another point.
Emulated Performance: Graphics	Select how graphics are rendered in the emulator: Hardware - Use your computer graphics card for faster rendering. Software - Emulate the graphics in software, which is useful if you're having a problem with rendering in your graphics card. Automatic - Let the emulator decide the best option based on your graphics card.

Emulated Performance: Boot option (Advanced)	Cold boot - Start the device each time by powering up from the device-off state. Quick boot - Start the device by loading the device state from a saved snapshot. For details, see Run the emulator with quick boot.
Emulated Performance: Multi-Core CPU (Advanced)	Select the number of processor cores on your computer that you'd like to use for the emulator. Using more processor cores speeds up the emulator.
Memory and Storage: RAM	The amount of RAM on the device. This value is set by the hardware manufacturer, but you can override it, if needed, such as for faster emulator operation. Increasing the size uses more resources on your computer. Type a RAM size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: VM Heap	The VM heap size. This value is set by the hardware manufacturer, but you can override it, if needed. Type a heap size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: Internal Storage	The amount of nonremovable memory space available on the device. This value is set by the hardware manufacturer, but you can override it, if needed. Type a size and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Memory and Storage: SD Card	The amount of removable memory space available to store data on the device. To use a virtual SD card managed by Android Studio, select Studio-managed, type a size, and select the units, one of B (byte), KB (kilobyte), MB (megabyte), GB (gigabyte), or TB (terabyte).
Device Frame: Enable Device Frame	Select to enable a frame around the emulator window that mimics the look of a real device.
Custom Skin Definition (Advanced)	Select a skin that controls what the device looks like when displayed in the emulator. Remember that specifying a screen size that's too big for the skin can mean that the screen is cut off, so you can't see the whole screen.
Keyboard: Enable Keyboard Input (Advanced)	Select this option if you want to use your hardware keyboard to interact with the emulator. It's disabled for Wear OS and Android TV.

13. DDMS:

The Dalvik Debug Monitor Server (DDMS) is a traffic director between the single port that Eclipse (and other Java debuggers) looks for to connect to a Java Virtual Machine (JVM) and the several ports that exist for each Android device or virtual device, and for each instance of the Dalvik virtual machine (VM) on each device. The DDMS also provides a collection of functionality that is accessible through a standalone user interface or through an interface embedded in Eclipse via the ADT plug-in.

The DDMS's user interface provides access to the following:

A list of devices and virtual devices, and the VMs running on those devices

- In the upper-left pane of the DDMS window, you will see listed the Android devices you have connected to your PC, plus any AVDs you have running. Listed under each device or virtual device are the tasks running in Dalvik VMs.

VM information

- Selecting one of the Dalvik VMs running on a device or virtual device causes information about that VM to be displayed in the upper-right pane.

Thread information

- Information for threads within each process is accessed through the "Threads" tab in the upper-right pane of the DDMS window.

Filesystem explorer

- You can explore the filesystem on a device or virtual device using the DDMS filesystem explorer, accessible through the "File explorer" menu item in the Devices menu. It displays the file hierarchy in a window.

Simulating phone calls

- The Emulator Control tab in the upper-right pane of the DDMS window enables you to "fake" a phone call or text message in an emulator.

Screen capture

- The "Screen capture" command in the Device menu fetches an image of the current screen from the selected Android device or virtual device.

Logging

- The bottom pane of the DDMS window displays log output from processes on the selected device or virtual device. You can filter the log output by selecting a filter from among the buttons on the toolbar above the logging pane.

Dumping state for devices, apps, and the mobile radio

- A set of commands in the Device menu enables you to command the device or virtual device to dump state for the whole device, an app, or the mobile radio.

The main services provided by Dalvik Debug Monitor Server are:

- App memory usage statistics (total heap and object allocation statistics)
- App thread statistics
- Device screen capture
- Device file explorer
- Incoming call and SMS spoofing
- Location data spoofing
- Logcat

14. Logcat: -

The **Logcat** window in Android Studio displays system messages, such as when a garbage collection occurs, and messages that you added to your app with the Log class. It displays messages in real time and keeps a history so you can view older messages. To display just the

information of interest, you can create filters, modify how much information is displayed in messages, set priority levels, display messages produced by app code only, and search the log. By default, logcat shows the log output related to the most recently run app only. When an app throws an exception, logcat shows a message followed by the associated stack trace containing links to the line of code.

The Logcat toolbar provides the following buttons:

1. **Clear Logcat** - Click to clear the visible log.
2. **Scroll to the end** - Click to jump to the bottom of the log and see the latest log messages. If you then click a line in the log, the view pauses scrolling at that point.
3. **Up the stack trace and Down the stack trace** - Click to navigate up and down the stack traces in the log, selecting the subsequent filenames that appear in the printed exceptions.
4. **Using soft wraps** - Click to enable line wrapping and prevent horizontal scrolling.
5. **Print** - Click to print the logcat messages.
6. **Restart** - Click to clear the log and restart logcat. Unlike the **Clear logcat** button, this recovers and displays previous log messages.

Logcat levels:

Logcat has several levels of log messages, so Android Studio provides multiple ways of filtering the logcat output.

1. **Verbose**: Display all log messages
2. **Debug**: Displays log messages that are useful during development
3. **Info**: Displays expected log messages for regular usage
4. **Warn**: Displays possible issues that are not yet errors
5. **Error**: Displays issues that have caused errors
6. **Assert**: Displays issues that should never happen

Write log messages: The Log class allows you to create log messages that appear in logcat. Generally, you should use the following log methods, listed in order from the highest to lowest priority (or, least to most verbose):

- Log.e.(String, String) (error)
- Log.w(String, String) (warning)
- Log.i(String, String) (information)
- Log.d(String, String) (debug)
- Log.v(String, String) (verbose)

Logcat message format: Every Android log message has a tag and a priority associated with it. The tag of a system log message is a short string indicating the system component from which the message originates (for example, ActivityManager).

Format – *date time PID-TID/package priority/tag: message*
Example - 12-10 13:02:50.071 1901-4229/com.google.android.gms V/AuthZen: Handling delegate intent.

15. **Memory Management: -**

The Android Runtime (ART) and Dalvik virtual machine use paging and memory-mapping (mmapping) to manage memory. This means that any memory an app modifies—whether by allocating new objects or touching mmapped pages—remains resident in RAM and cannot be paged out. The only way to release memory from an app is to release object references that the app holds, making the memory available to the garbage collector. That is with one exception: any files mmapped in without modification, such as code, can be paged out of RAM if the system wants to use that memory elsewhere.

Garbage Collection:

A managed memory environment, like the ART or Dalvik virtual machine, keeps track of each memory allocation. Once it determines that a piece of memory is no longer being used by the program, it frees it back to the heap, without any intervention from the programmer. The mechanism for reclaiming unused memory within a managed memory environment is known as *garbage collection*. Garbage collection has two goals:

- find data objects in a program that cannot be accessed in the future;
- and reclaim the resources used by those objects.

Android's memory heap is a generational one, meaning that there are different buckets of allocations that it tracks, based on the expected life and size of an object being allocated. Each heap generation has its own dedicated upper limit on the amount of memory that objects there can occupy. Any time a generation starts to fill up, the system executes a garbage collection event in an attempt to free up memory. The duration of the garbage collection depends on which generation of objects it's collecting and how many active objects are in each generation.

Share memory:

In order to fit everything, it needs in RAM, Android tries to share RAM pages across processes. It can do so in the following ways:

- Each app process is forked from an existing process called Zygote. The Zygote process starts when the system boots and loads common framework code and resources (such as activity themes). To start a new app process, the system forks the Zygote process then loads and runs the app's code in the new process.
- Most static data is mmapped into a process. This technique allows data to be shared between processes, and also allows it to be paged out when needed. Example static data include: Dalvik code (by placing it in a pre-linked `.odex` file for direct mmapping), app resources (by designing the resource table to be a structure that can be mmapped and by aligning the zip entries of the APK), and traditional project elements like native code in `.so` files.
- In many places, Android shares the same dynamic RAM across processes using explicitly allocated shared memory regions (either with ashmem or gralloc). For example, window surfaces use shared memory between the app and screen compositor, and cursor buffers use shared memory between the content provider and client.

Allocate and reclaim app memory:

The Dalvik heap is constrained to a single virtual memory range for each app process. This defines the logical heap size, which can grow as it needs to but only up to a limit that the system defines for each app. The logical size of the heap is not the same as the amount of physical memory used by the heap. When inspecting your app's heap, Android computes a value called the Proportional Set Size (PSS), which accounts for both dirty and clean pages that are shared with other processes—but only in an amount that's proportional to how many apps

share that RAM. This (PSS) total is what the system considers to be your physical memory footprint. The Dalvik heap does not compact the logical size of the heap, meaning that Android does not defragment the heap to close up space. Android can only shrink the logical heap size when there is unused space at the end of the heap. However, the system can still reduce physical memory used by the heap.

Restrict app memory:

To maintain a functional multi-tasking environment, Android sets a hard limit on the heap size for each app. The exact heap size limit varies between devices based on how much RAM the device has available overall. If your app has reached the heap capacity and tries to allocate more memory, it can receive an OutOfMemoryError.

Switch apps:

When users switch between apps, Android keeps apps that are not foreground—that is, not visible to the user or running a foreground service like music playback— in a cache. For example, when a user first launches an app, a process is created for it; but when the user leaves the app, that process does *not* quit. The system keeps the process cached. If the user later returns to the app, the system reuses the process, thereby making the app switching faster.If your app has a cached process and it retains resources that it currently does not need, then your app—even while the user is not using it— affects the system's overall performance. As the system runs low on resources like memory, it kills processes in the cache.

16. Design Patterns for limited memory:

When composing designs for devices with a limited amount of memory, the most important principle is not to waste memory. This means that the design should be based on the most adequate data structure, which offers the right operations.In addition to this basic principle, a number of other considerations are related to memory management, where the objective is to use memory such that its implementation leads to minimal leaking of abstraction, or, where applicable, the underlying implementation provides improved properties, like for instance better performance if support can be gained from cache.
In the following, some design patterns created for help in designing small memory software:

Linear Data Structures:

In contrast to data structures where a separate memory area is reserved for each item, linear data structures are those where different elements are located next to each other in the memory. The difference in the allocation in the memory also plays a part in the quality properties of data structures.
The principal rule is to *favor linear data structures*. Linear data structures are generally better for memory management than non-linear ones for several reasons, as listed in the following:

- **Less fragmentation**. Linear data structures occupy memory place from one location, whereas non-linear ones can be located in different places. Obviously, the former results in less possibility for fragmentation.
- **Less searching overhead**. Reserving a linear block of memory for several items only takes one search for a suitable memory element in the run-time environment, whereas non-linear structures require one request for memory per allocated element. Combined with a design where one object allocates a number of child objects, this may also lead to a serious performance problem.
- **Design-time management.** Linear blocks are easier to manage at design time, as fewer reservations are made. This usually leads to cleaner designs.
- **Monitoring.** Addressing can be performed in a monitored fashion, because it is possible to check that the used index refers to a legal object.

• **Cache improvement**. When using linear data structures, it is more likely that the next data element is already in cache, as cache works internally with blocks of memory. A related issue is that most caches expect that data structures are used in increasing order of used memory locations. Therefore, it is beneficial to reflect this in designs where applicable.
• **Index uses less memory**. An absolute reference to an object usually consumes 32 bits, whereas by allocating objects to a vector of 256 objects, assuming that this is the upper limit of objects, an index of only 8 bits can be used. Furthermore, it is possible to check that there will be no invalid indexing.

Basic Design Decisions:

In the following, we introduce some basic principles helping in using linear data structures..

- **Allocate all memory at the beginning of a program**. This ensures that the application
  always has all the memory it needs, and memory allocation can only fail at the beginning of the program. Reserving all the resources is particularly attractive when the most important or mandatory features like emergency calls, for instance, are considered, for which resources must always be available.
- **Allocate memory for several items, even if you only need one.** Then, one can build a policy where a number of objects is reserved with one allocation request. These objects can then be used later when needed. This reduces the number of allocation requests, which leads to a less complex structure in the memory.
- **Use standard allocation sizes.** With a standard allocation size, it is easy to reuse a deallocated area in the memory when the next reservation is made. As a result, fragmentation of memory can be prevented, at least to some extent.
- **Reuse objects.** Reusing old objects might require using a pool of free objects. This requires some data structure for managing free and used data structures. This implies that the programmer actively participates in the process of selecting object construction and destruction policy in the design.
- **Release early, allocate late.** By always deallocating as soon as possible the programmer can give more options for memory management, because new objects can be allocated to the area that has just been released as well. In contrast, by allocating memory as late as possible, the developer can ensure that all possible deallocations have been performed before the allocation
- **Use permanent storage or ROM when applicable.** In many situations, it is not even desirable to keep all the data structures in the program memory due to physical restrictions. It is advisable to introduce the custom to save all data to permanent storage as soon as possible.
- **Avoid recursion**. Invoking methods obviously causes stack frames to be generated. While the size of an individual stack frame can be small – for instance, in Kilo Virtual Machine (KVM), which is a mobile Java virtual machine commonly used in early Java enabled mobile phones, the size of a single stack frame is at least 28 bytes (7 × 4 bytes) – functions calling themselves recursively can end up using a lot of stack, if the depth of the recursion is not considered beforehand.

Data Packing:

Data packing is probably the most obvious way to reduce memory consumption. There are several sides to data packing. However, it should be emphasized that selecting the right data structure as the basis of an implementation is usually vastly more important than packing in the majority of practical designs.

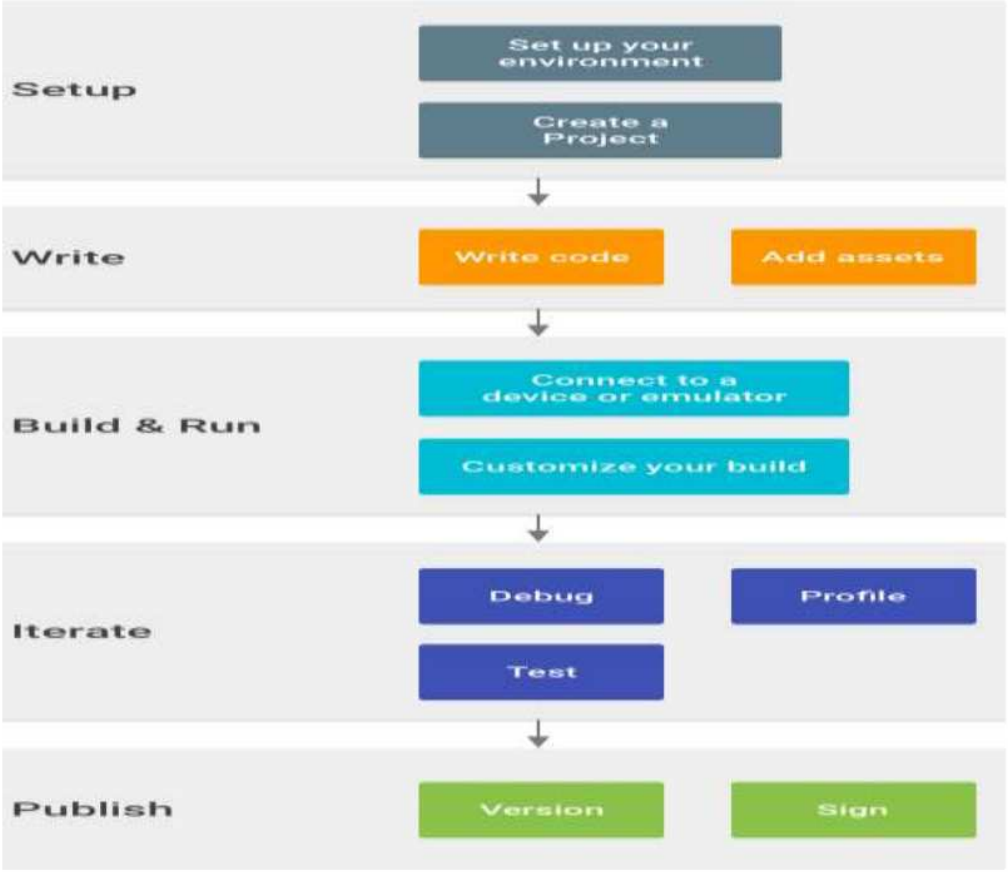
- **Consider word alignment**. Due to word alignment, it is possible that data structures

are not optimally located in the memory.

- **Use compression with care.** In addition to considering the data layout in memory, there are several compression techniques for decreasing the size of a file. Since packing as a technique is simple it can be applied to both code and data structures, at least in principle.

17. Work flow for Application Development: -

The workflow to develop an app for Android is conceptually the same as other app platforms. However, to efficiently build a well-designed app for Android, you need some specialized tools. The following list provides an overview of the process to build an Android app and includes links to some Android Studio tools you should use during each phase of development.



- **Set up your workspace:** This is the phase you probably already finished: Install Android Studio and create a project.
- **Write your app:** Now you can get to work. Android Studio includes a variety of tools and intelligence to help you work faster, write quality code, design a UI, and create resources for different device types.
- **Build and run:** During this phase, you build your project into a debug-gable APK package that you can install and run on the emulator or an Android-powered device. You can also begin customizing your build. For example, you can create build variants that produce different types of APKs from the same project, and shrink your code and resources to make your APK file smaller.

- **Debug, profile, and test:** This is the iterative phase in which you continue writing your app but with a focus on eliminating bugs and optimizing app performance. Of course, creating tests will help you in those endeavours.
- **Publish:** When you're ready to release your app to users, there are just a few more things to consider, such as versioning your app and signing it with a key.

18. Techniques for composing Applications:-

The techniques for composing the application are as follows:

- **KEEP DESIGN RESPONSIVE:**
  Apps must be friendly with a great assortment of devices. Customer choice routinely commands a certain proportion of the market won't adopt the widespread platform, doesn't matter how widespread it is. Android's perceived its prime, and so includes the iPhone.
- **TRY ITERATIVE DESIGN**
  Iterative development means making use of information from assessment techniques as user analysis. Iterate UI to build responsive and engaging apps for a particular demographic.This is also an approach to study valuable programs for future design projects.
- **Pixels**
  Visual design is hugely important in the perceived quality of an app. It might even improve usability. Most developers have some exposure to UI patterns, but developers with visual design skills are *rare*.Delivering high-fidelity mock-ups, drawable resources (i.e. graphic assets) and guidance to developers is the best way to deliver an aesthetically pleasing experience to the end user.
- **SCALE NICELY**
  Android is a platform of many screen densities. There is no set of resolutions to target, rather a density independent measurement scheme for graphics, widgets and layouts. This is covered in depth in a previous Smashing article and the official documentation, so I'll just add a mention of this neat web tool for calculating density pixels.
- **USE 9-PATCH DRAWABLES**
  9-patch drawables allow PNGs to stretch and scale nicely in pre-defined ways. Markings along the top and left edges define the stretchable areas. The padded content area can optionally be defined with markings along the bottom and right edges. **9-patches are essential** for creating and customizing UI widgets.
- **CHECK WITH USERS**
  Designers have various testing approaches to opt from, but obtaining user feedback is recognized as an industry standard.
  This has several design rewards; the main among them is being an easy modification of bugs in an initial stage of development.
  This saves enough expenses and exertions involved with development entirely.
- **BE PICTORIAL**
  An interesting mobile design is highly graphic with a perfect color arrangement to boot.As an authentic graphic art, a wide range of colors, textures, and configuration behaviors will obtain one nowhere in the expedition for flawless UX.

Arresting pictorial components also keep users involved, boosting them to respond for a quality, interactive experience.

- **DEPENDABLY UPDATE ACCORDING TO SAFETY ISSUES**
  Security and agreement needs can be a huge obstruction to efficient development.
  User-friendliness concerns can be evaded through a conference with the back-end team, every so often more conscious of how to stop data gaps. However amenability may be easy to plan for on a yearly basis, security concerns need reliable checking.
- **KEEP MARKET NEEDS IN MIND**
  Marketing strategies for all businesses and this comprise app development! It is significant to understand consumers.
  Build use cases for what a targeted consumer, in fact, wants from an app. The entire categories of factors come into the show here, thus, do in-depth research based on this specific app's emphasis is imperative.
  Users with a tailored trace that displays developers focused on their requirements and interests as a demographic.
  Developers may utilize experiential assessment with usability analysis to build UI that is tempting to users. The only common thing here is interactivity. Interactive apps will never drop to provide visitors with the unique user experience.

19. Dynamic Linking:-

Dynamic linking, often implemented with dynamically linked libraries (DLL), is a common way to partition applications and subsystems into smaller portions, which can be compiled, tested, reused, managed, deployed, and installed separately. For instance, assuming that a design of an application follows the MVC pattern, it is sometimes convenient to implement the model as a separate library that can be loaded when necessary. Then, the library can be reused in some other graphical user interface that needs the same functions, for instance, or by applications that integrate the model to a larger context. Furthermore, the use of dynamically linked libraries can also be a way to interact with a given framework, thus allowing a well-defined boundary between two units of software. In particular, extensions and adaptations to an already existing system can often be more elegantly handled using dynamic libraries than with source code, because recompilations of already compiled parts can be avoided and the correct configuration can predominantly be created with binary files.

The most important conceptual advantages are listed:

• Several applications can use the library in such a fashion that only one copy of the library is needed, thus saving memory.

• Application-specific tailoring can be handled in a convenient fashion, provided that supporting facilities exist.

• Smaller compilations and deliveries are enabled.

• Composition of systems becomes more flexible, because only a subset of all possible software can be included in a device when creating a device for a certain segment.

• It is easier to focus testing to some interface and features that can be accessed using that interface.

• Library structure eases scoping of system components and enables the creation of an explicit unit for management.

• Work allocation can be done in terms of dynamic libraries, if the implementation is carried out using a technique that does not support convenient mechanisms for modularity.

Disadvantages of DLL's:

A common risk associated with dynamic libraries is the fragmentation of the total system into small entities that refer to each other seemingly uncontrollably. When more and more dynamic libraries are added to the system, their versioning can result in cases where some applications work with a certain version of a library, whereas others require an update or a previous or newer version.
Furthermore, managing all the dependencies between libraries is another source of difficulties, as a collection of compatible components is often needed.

Static versus Dynamic DLLs:

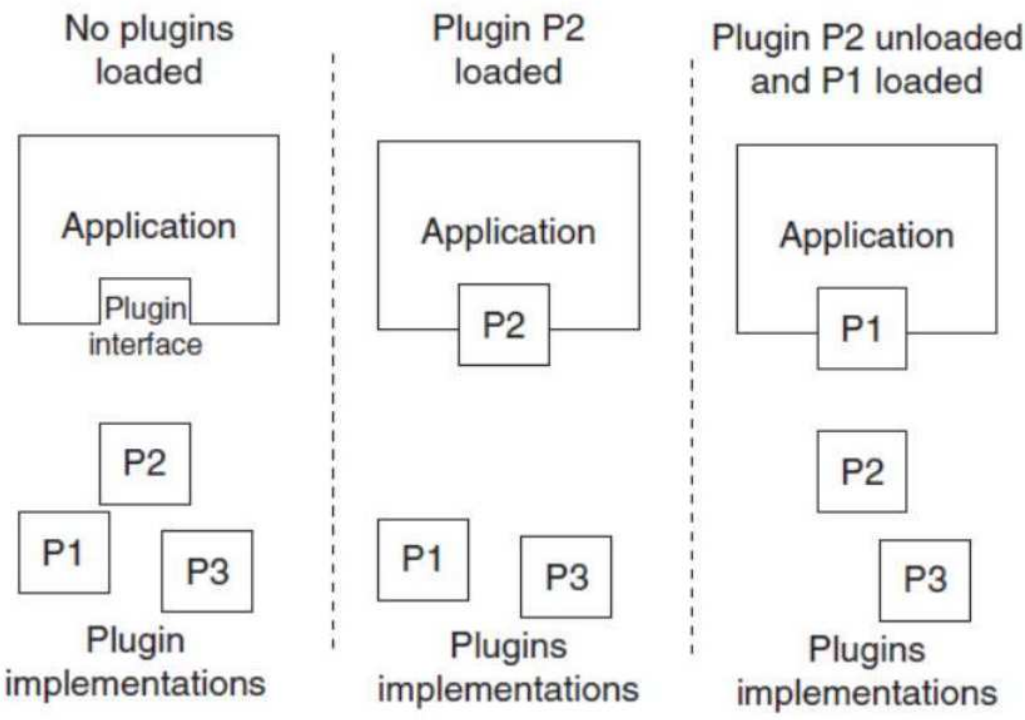
While dynamically linked libraries are all dynamic in their nature, there are two different implementation schemes. One is static linking, which most commonly means that the library is instantiated at the starting time of a program, and the loaded library resides in the memory as long as the program that loaded the library into its memory space is being executed. The benefit is that one can use the same static library in several applications. For instance, when relying on MVC as the architecture of an application, one can imagine a model implemented as a dynamically linked library that can be used by several applications. In contrast to static DLLs, dynamic DLLs, which are often also referred to as plugins, especially if they introduce some special extension, can be loaded and unloaded whenever needed, and the facilities can thus be altered during an execution. The benefit of the approach is that one can introduce new features using such libraries. For instance, in the mobile setting one can consider that sending a message is an operation that is similar to different message types (e.g. SMS, MMS, email), but different implementations are needed for communicating with the network in the correct fashion.

20. Plugins:

Plugins, which dynamically loaded dynamically linked libraries are often referred to as, especially if they play a role of an extension or specialization, are a special type of DLL that enable differentiation of operations for different purposes at runtime. They usually implement a common interface used by an application, but their operations can still differ at the level of implementation. One could implement different plugins for different types of messages that can be sent from a mobile device.

Plugin Principles:

Plugins take advantage of the binary compatibility of the interface provided by a dynamically linked library. The important concepts of a plugin are the interfaces they implement, and the implementations they provide for interfaces. The interface part is used by applications using the plugin for finding the right plugins, and the implementation defines the actual operations. Commonly some special information regarding the interface is provided, based on which the right plugin library can be selected. When a plugin is selected for use, its implementation part is instantiated in the memory similarly to normal dynamically linked libraries.



Implementation-Level Concerns

While plugins in principle are a simple technique, there are several implementation details that must be considered before applying the approach.

To begin with, a framework is commonly used for loading and unloading plugins. This implies a mechanism for extending (or degenerating) an application on the fly when some new services are needed.

Secondly, in order to enable loading, facilities must be offered for searching all the available implementations of a certain interface. This selection process is commonly referred to as resolution. In many implementations, a default implementation is provided if no other libraries are present, especially when a system plugin that can be overridden is used.

Finally, a policy for registering components and removing them is needed in order to enable dynamic introduction of features. This can be based on the use of registry files, or simply on copying certain files to certain locations, for instance.

21. Rules of Thumb for Using Dynamically Loaded Libraries:

In principle, any unit of software can be implemented as a separate library. In practice, however, creating a complex collection of libraries that depend on each other in an ad-hoc fashion is an error that is to be avoided at all cost. In the following, we list some candidate justifications for creating a separate dynamically linked library out of a program component.

• **Reusable or shareable components** should be implemented using dynamically loaded libraries, as otherwise all the applications that use the components must contain them separately. This in turn consumes memory. In addition, sharing can take place in a form where the same model, implemented as a dynamically loaded library, is reused in several different devices that require a specialized user interface for each device.

• **Variation or management point** can be preferable to implement in terms of

dynamic libraries. This makes variation or management more controlled, as it can be directly associated with a software component. Moreover, the library can be easily changed, if updates or modifications are needed. For instance, interfaces can be treated in this fashion even if the underlying infrastructure (e.g. C++) would not offer such an option. Management can also be associated with scoping in general, allowing that a certain module can be evolved separately.

• **Software development processes** such as automated testing may require that all the input is in a form that can be directly processed. Then, performing the tests may require that binary components are delivered.

• **Organizational unit** can be authorized to compose a single library that is responsible for a certain set of functions. Requesting a library then results in a separate deliverable that can be directly integrated into the final system.

22. Resource Management: -

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other resources like static content that your code uses, such as bitmaps, colours, layout definitions, user interface strings, animation instructions, and more.

These resources are always maintained separately in various sub-directories under res/ directory of the project.

S No.	Directory and Resource Type
1	**anim/** XML files that define property animations. They are saved in res/anim/ folder and accessed from the **R.anim** class.
2	**color/** XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.
3	**drawable/** Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class.
4	**layout/** XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.
5	**menu/** XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.
6	**raw/**

	Arbitrary files to save in their raw form. You need to call *Resources.openRawResource()* with the resource ID, which is *R.raw.filename* to open such raw files.
7	**values/** XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory − • arrays.xml for resource arrays, and accessed from the **R.array** class. • integers.xml for resource integers, and accessed from the **R.integer** class. • bools.xml for resource boolean, and accessed from the **R.bool** class. • colors.xml for color values, and accessed from the **R.color** class. • dimens.xml for dimension values, and accessed from the **R.dimen** class. • strings.xml for string values, and accessed from the **R.string** class. • styles.xml for styles, and accessed from the **R.style** class.
8	**xml/** Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various configuration files here which will be used at run time.

Resource Manager: Resource Manager is a tool window for importing, creating, managing, and using resources in your app. You can open the tool window by selecting **View > Tool Windows > Resource Manager** from the menu bar or by selecting **Resource Manager** on the left side bar.

1. Click **Add** to add a new resource to your project. You can add image assets, vector assets, fonts, resource files and values, or you can import resources into your project.
2. The module dropdown lets you view resources specific to a module.
3. Use the search bar to search for a resource across all modules in your project.
4. The Resource Manager groups your resources by type. Use these tabs to switch between each resource type. Click the overflow icon to show additional resource types.
5. The filter button lets you view resources from local dependent modules, external libraries, and the Android framework. You can also use the filter to show theme attributes.
6. The main content area displays previews of your resources. Right-click on a resource to see a context menu where, among other things, you can rename the resource and search your app for where the resource is used.