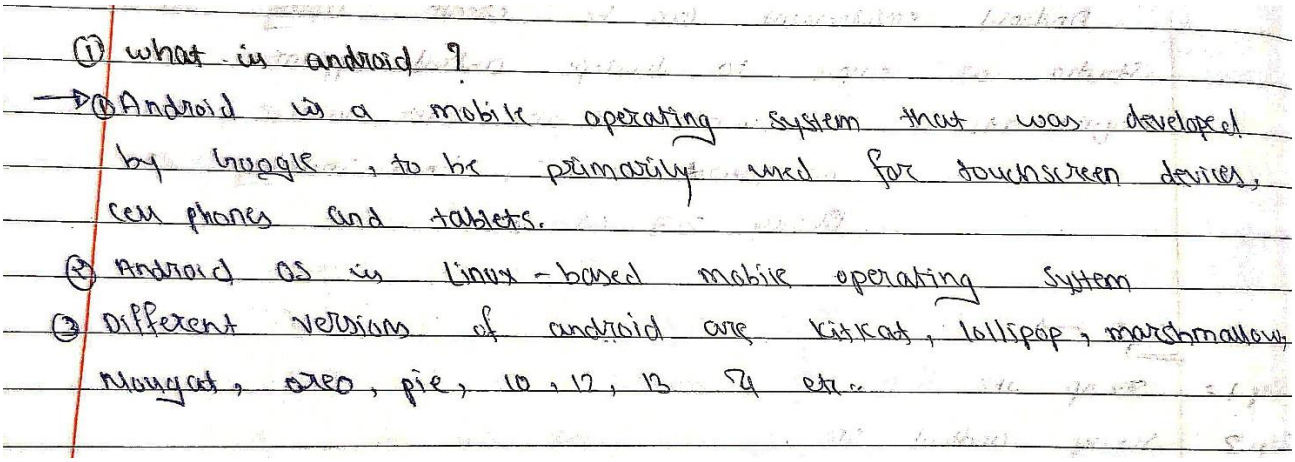
MAD CAT 1 QB ANSWERS

a) What is android? Explain their versions?



Code name	Version numbers	API level	Release date
No codename	1.0	1	September 23, 2008
No codename	1.1	2	February 9, 2009
Cupcake	1.5	3	April 27, 2009
Donut	1.6	4	September 15, 2009
Eclair	2.0 - 2.1	5 - 7	October 26, 2009
Froyo	2.2 - 2.2.3	8	May 20, 2010
Gingerbread	2.3 - 2.3.7	9 - 10	December 6, 2010
Honeycomb	3.0 - 3.2.6	11 - 13	February 22, 2011
Ice Cream Sandwich	4.0 - 4.0.4	14 - 15	October 18, 2011
Jelly Bean	4.1 - 4.3.1	16 - 18	July 9, 2012
KitKat	4.4 - 4.4.4	19 - 20	October 31, 2013
Lollipop	5.0 - 5.1.1	21- 22	November 12, 2014
Marshmallow	6.0 - 6.0.1	23	October 5, 2015
Nougat	7.0	24	August 22, 2016
Nougat	7.1.0 - 7.1.2	25	October 4, 2016
Oreo	8.0	26	August 21, 2017

Oreo	8.1	27	December 5, 2017
Pie	9.0	28	August 6, 2018
Android 10	10.0	29	September 3, 2019
Android 11	11	30	September 8, 2020

Android Version 1.0 to 1.1: No codename

Android officially publish its Android version 1.0 in September 2008. It is the initial version of Android operating system. It supports Web browser to show HTML and XHTML web pages, camera, access web email server (POP3, IMAP4, and SMTP). This version contains Google Calendar, Google Maps, Google Sync, Google Search, Google Talk, Instant messaging, Media player, Notifications appear in the status bar, wallpaper, YouTube video player, Alarm Clock, Calculator, Dialer, Pictures (Gallery), Wi-Fi and Bluetooth support.

Android version 1.5: Cupcake

On April 27, 2009, the Android updated to 1.5 with the codename of the dessert item (Cupcake). It has Linux kernel 2.6.27. It supports third-party virtual keyboard, Video recording and playback in MPEG-4, Copy and paste feature, Animated screen translations, auto-rotation option, ability to upload a video to YouTube, upload photos to Picasa, check phone usage history.

Android version 1.6: Donut

On September 15, 2009, Android 1.6 was released with the name Donut. It contains numerous new features such as voice and text entry search, bookmark history, contacts, web, "speak" a string of text, faster camera access, user can select multiple photos for deletion, support text-to-speech engine, WVGA screen resolutions.

Android version 2.0 to 2.1: Eclair

On October 26, 2009, Android 2.0 was released, whose codename was Eclair. It was based on Linux kernel 2.6.29. It contains the several new features as expanded account sync, Microsoft Exchange email support, Bluetooth 2.1, ability to tap a Contact photo and select to call, SMS, ability to search all saved SMS, MMS messages, delete the oldest message automatically when the defined limit is reached, Minor API, bug fixes.

Android version 2.2 to 2.2.3: Froyo

On May 20, 2010, Android 2.2 (Froyo) was released based on Linux kernel 2.6.32. It contains several features as speed, memory, performance optimization. JIT compilation, Integration of Chrome's V8, JavaScript engine into the Browser application, support Android Cloud to Device Messaging service, Adobe Flash support, security updates, and performance improvement.

Android version 2.3 to 2.3.7: Gingerbread

On December 6, 2010, the Android 2.3 (Gingerbread) was released based on Linux kernel 2.6.35. It includes the following changes: support for extra-large screen size and resolutions, updated user interface design with increased simplicity and speed, enhanced copy/paste functionality, select a word by press-holding, support Near Field Communication (NFC), headphone virtualization, new Download Manager.

It has improved bug fixes for Nexus S, voice or video chat using Google Talk, network performance for Nexus S 4G, Gmail application, battery efficiency, fixed a voice search bug, Google Wallet support for Nexus S 4G.

Android version 3.0 to 3.2.6: Honeycomb

On February 22, 2011, Android 3.0 (Honeycomb) was launched for the first tablet for Android-based on Linux kernel 2.6.36. It contains the features like "holographic" user interface for tablet, added system Bar, simplified multitasking tapping Recent Application in system Bar, redesign the keyboard making fast typing, quick access to camera exposure, hardware acceleration, support for multi-core processor, UI refinements, connectivity for USB accessories, support for joysticks and gamepads, high-performance Wi-Fi lock, improved hardware support, Google Books, fixed data connectivity issues when coming out of Airplane mode.

Android version 4.0 to 4.0.4: Ice Cream Sandwich

On October 19, 2011, Android 4.0.1 (Ice Cream Sandwich) was launched, which was based on Linux kernel 3.0.1. It was the last version of officially support Adobe System Flash player. It introduces the numerous new features: refinements to "Holo" interface with new Roboto font family, separation of widgets in a new tab, integrated screenshot capture, improved error correction on the keyboard, improved copy and paste functionality, build-in photo editor, fixed minor bugs, improvement to graphics, spell-checking, better camera performance.

Android version 4.1 to 4.3.1: Jelly Bean

On June 27, 2012, Google announced Android 4.1(Jelly Bean) in the Google I/O conference. It is based on Linux kernel 3.0.31. It updates to following features: smoother user interface, enhance accessibility, expandable notification, fixed bug on Nexus 7, one-finger gestures to expand/collapse notifications, lock screen improvement, multiple user accounts (tablets only), new clock application, Bluetooth low energy support, volume for incoming call, 4K resolution support, native emoji support, bug fixes for the Nexus 7 LTE.

Android version 4.4 to 4.4.4: KitKat

On September 3, 2013, Google announced Android 4.4 (KitKat). Initially, its code name was "Key Lime Pie". Google started on Google's Nexus 5 on October 31, 2013. The minimum required amount of RAM should available to Android is 340 MB. The other devices with less than 512 MB of RAM must report themselves as "low RAM" devices. It includes several new features as clock no longer display bold hours, wireless printing capability, WebViews are based on Chromium engine, sensor batching, built-in screen recording feature, better application compatibility, camera application loads Google+ Photo instead of Gallery.

Android version 5.0 to 5.1.1: Lollipop

Android 5.0 "Lollipop" was initially named "Android L" on June 25, 2014. It was officially introduced on November 12, 2014. Lollipop provides several features like redesigned user interface, support for 64-bit CPUs, support for print previews, material design, Project Volta for battery life improvement, multiple user accounts, audio input, and output through USB devices, join Wi-Fi networks, support for multiple SIM cards, device protection, high-definition voice calls, native Wi-Fi calling support.

Android version 6.0 - 6.0.1: Marshmallow

Android 6.0 "Marshmallow" was disclosed under the codename "Android M" on May 28, 2015, for Nexus 5 and Nexus 6 phones, Nexus 9 tablet.

On October 5, 2015, Android lunches "Marshmallow" for all android devices. It contains the various new features as App Standby feature, introduce the Doze mode to save battery life, native fingerprint reader support, run-time permission requests, USB-C support, Unicode 7.0 & 8.0 emoji support.

Android version 7.0 to 7.1.2: Nougat

Android 7.0 "Nougat" was the major release for the Android operating system. Its initial codename was "Android N". It was first released as a developer preview on March 9, 2016, with factory images for the Nexus device.

On August 22, 2016, the final preview built was released with following features: file-based encryption, zoom in the screen, multi-window support, new Data Saver mode, JIT compiler makes 75 percent faster app installation, picture-in-picture support, support manager APIs, circular app icons support, send GIFs directly from the default keyboard, battery usage alerts.

Android version 8.0 to 8.1: Oreo

Android 8.0 "Oreo" was the 8th major release of the Android operating system. It was first released for developer preview on March 21, 2017. The final developer preview was released on July 24, 2017.

On August 21, 2017, its stable version was released with several features: picture-in-picture support, support for Unicode 10.0 emoji (5.0), restructured settings, adoptive icons, notification channels, notification dots, 2 times faster boot time, Google Play Protect, Integrated printing support, Neural network API, shared memory API, Android Oreo Go Edition, autofill framework, automatic light, and dark themes.

Android version 9.0: Pie

Android 9.0 "Pie" was the ninth major version of the Android operating system. It was first announced and preview launched by Google on March 7, 2018. It was officially released on August 6, 2018. It has the following features: the clock has moved to the left of the notification bar, the "screenshot" button has been added, battery percentage always shown on display.

Android version 10:

Android 10 is the tenth extensive version of the Android operating system. Android 10 has developed under the codename "Android Q". It was initially announced by Google on March 13, 2019 and its first beta version was released on same day and its second beta was released on April 3, 2019.

The stable version of Android 10 was released on September 3, 2019. It contains features like new permissions to access location in the background, floating setting panel, support for an AV1 video codec, support for biometric authentication, support the WPA3 Wi-Fi security.

Android 11

Android 11 operating system is the eleventh big release of Android. It is the 18th version of Android mobile OS, which was released on 8 September 2020. The alphabetic naming system of Android, based on deserts, was stopped since Android 10. So therefore, this operating system has branded with "Android 11".

Features included in Android 11

- **Conversations:** Get all your message in one place.
- **Accessibility:** Perceptive apps help us to control and navigate our phone using voice command.
- **Device controls:** Android 11 allows us to control all our connected devices (IOT) from a single point.
- **Content capture:** Android 11 comes with a screen recording feature that captures our phone's current screen activity.
- **Predictive tools:** By predicting our habits and patterns of working, it suggests accordingly.
- **Privacy & security:** Android 11 gives more security and privacy fixes to our smartphone straight from Google Play.
- **Media:** We can play music from other devices connected to our phones.

b) Explain the procedure steps of Installing Android SDK Tools

a) Explain the features of Android & its Application

Features of Android

Android is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below –

Sr.No.	Feature & Description
1	**Beautiful UI** Android OS basic screen provides a beautiful and intuitive user interface.
2	**Connectivity** GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
3	**Storage** SQLite, a lightweight relational database, is used for data storage purposes.
4	**Media support** H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.
5	**Messaging** SMS and MMS
6	**Web browser** Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
7	**Multi-touch** Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
8	**Multi-tasking** User can jump from one task to another and same time various application can run simultaneously.

9	**Resizable widgets** Widgets are resizable, so users can expand them to show more content or shrink them to save space.
10	**Multi-Language** Supports single direction and bi-directional text.
11	**GCM** Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
12	**Wi-Fi Direct** A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
13	**Android Beam** A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

Some of the most common applications are as follows:

1. Navigation

You have apps like Google Maps, which help us locate ourselves and find a route to visit the desired location.

2. Lifestyle

You have apps like Cult. Fit and Step Set Go helps you track your fitness, how many calories you are burning, and how many walking steps you are walking daily.

3. Messaging

There are apps like Whatsapp, Telegram, etc., which help us message any individual or a group of people together.

4. Travel

You have apps like MakeMyTrip, RedBus, etc., making our travel booking, hotel book, or other amenities very easy and at our fingertips.

Web Apps

Web applications are built only the run on browsers. They are mainly the integrations of **HTML**, **CSS,** and **Javascript**. It runs on Chrome, Firefox, and other browsers

1. **E-Commerce Apps:** E-commerce apps are an example of a B2B model. It helps to people to sell and borrow different items and it saves time and money. In e-commerce applications, we can do trading of commercial goods on online marketplaces. To buy specific items and goods, you simply need to make electronic transactions like UPI, Phonepe, etc. through your smartphone or computer. Flipkart, Amazon, OLX, and, Quiker are examples of e-commerce applications.
2. **Educational Apps:** Educational apps are too much used to improve knowledge and peoples get productivity. Apps for education can make people more interactive, more engaged, and perform better. Keeping teaching methods good is integral to getting students engaged in their studies and learning apps are a fantastic way of achieving this. For example, Google Classroom, SoloLearn, edX, Duolingo, etc.
3. **Social Media Apps:** Social media apps give the opportunity to the peoples connect and communicate together. These apps are mainly used for sharing purposes and making fun. Many peoples use social media applications for influence, marketing/ business, entrepreneurship, etc. Instagram, Facebook, WhatsApp, YouTube, LinkedIn, etc. are examples of social media applications.
4. **Productivity Apps:** Productivity apps typically organize and complete complex tasks for you, anything from sending an email to figuring out a tip. The easy-to-use Google Drive app gives users access to all of the files saved to the cloud-based storage service across multiple devices. Productivity applications arise in many different forms and they often take a different approach to improving your workflow. For example, Hive, Todoist, Google Docs, etc.
5. **Entertainment Apps:** Entertainment apps are widely used apps worldwide. It contains OTT platforms and novels and other content. These platforms entertain people and give them much more knowledge about different things. Everyone is watching OTT platforms and those are trending these days, and their development is also in demand all over the world. Hotstar, Netflix, and Amazon prime video are the best examples of this entertainments applications.

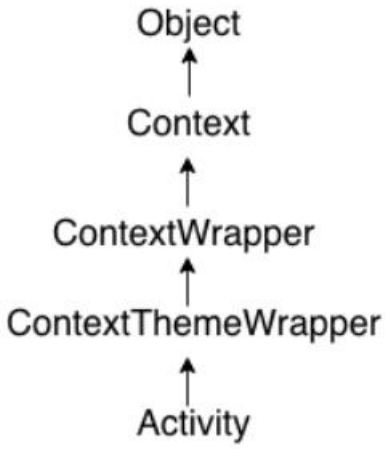
b) What is an Activity ? Explain Activity LifeCycle.

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

An activity is the single screen in android. It is like window or frame of Java.

By the help of activity, you can place all your UI components or widgets in a single screen.

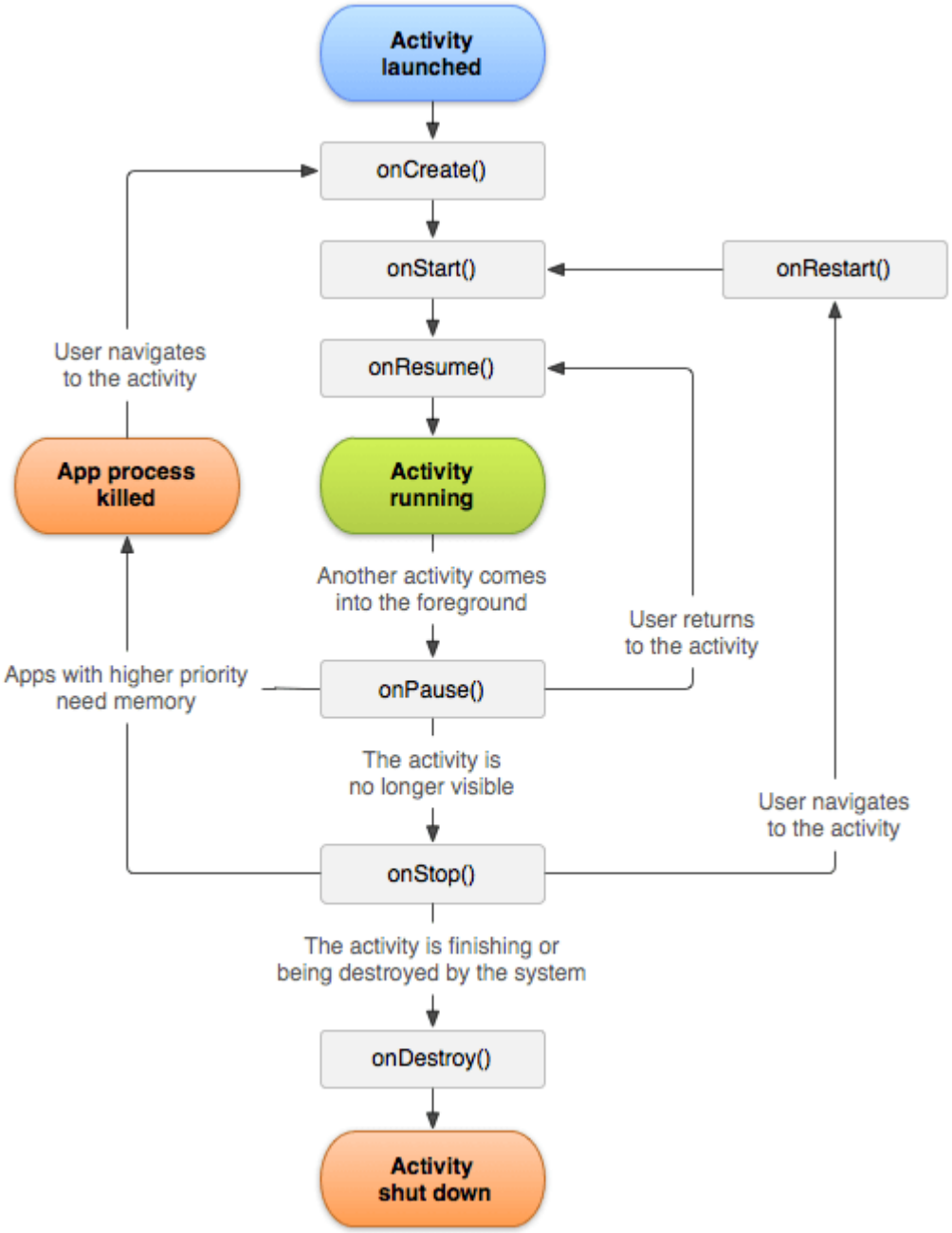
The 7 lifecycle method of Activity describes how activity will behave at different states.



Android Activity Lifecycle methods

Let's see the 7 lifecycle methods of android activity.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

a) Define Intent. Explain type of Intent.

(6) Explain intent and their types.

→ (1) In android Intent is a messaging object which is used to request an action from another app component.

(2) It will help us to maintain the communication betn app component from the same application as well as the component of other application.

(3) There are two types of intent :

a) Explict Intent

b) Implict Intent.

a) Explict Intent :→

It going to connect the internal world of an application such as start activity or send data between two activities. It specifies the component.

b) Implicit Intent :→

It doesn't specify the component. It provides info. on available components provided by the system that is to be invoked.

b) Write a Short Note On :

1. User Interface
2. Widgets
3. Activities
4. Services

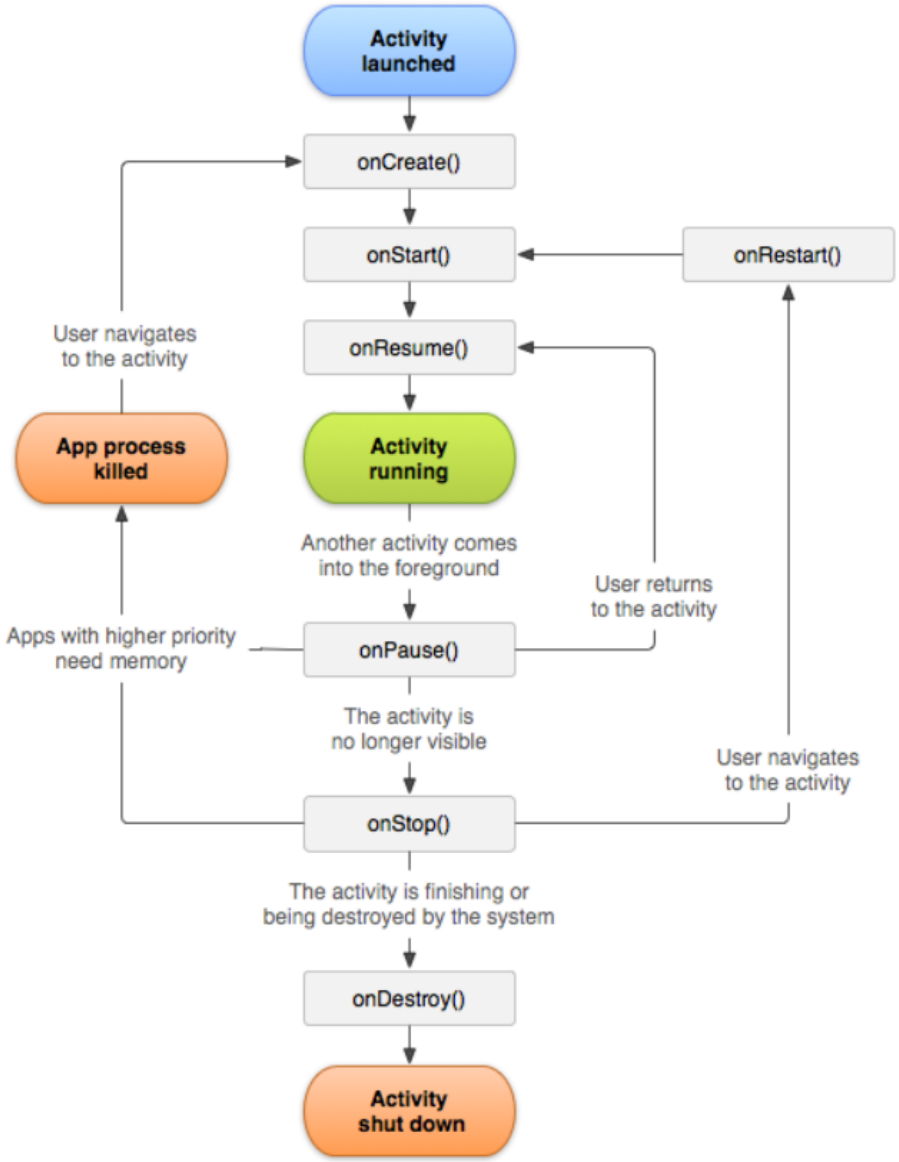
Activities:

An activity is one screen of an app. In that way the activity is very similar to a window in the Windows operating system.
The most specific block of the user interface is the activity. An Android app contains activities, meaning one or more screens.

Examples: Login screen, sign up screen, and home screen.

An activity class is per definition a class in Android. Every application which has UI must inherit it to create a window.

An activity goes through a number of states. Activity lifecycle in Android manages the different states of activity, such as when the activity starts or stops, etc. All of these states are managed by callback methods.



There are seven callback methods:-

onCreate():- This callback method must be created by you, this is fired when your activity is created by the system. For example, when you open the app it will call onCreate(), onStart(), and onResume() methods. These three methods are initiated when you open an app.

onStart():- There should always be an onStart() callback after onCreate() finishes. In the started state users will be able to see the activity but they are not ready for user interaction.

onResume():- In this, activities will be in the foreground and ready for user interaction. You can understand the use of this callback from this example: "when the user presses the phone's answer button it will give onPause() after you end up calling it will again give onResume()".

onPause():- This callback occurs when there is a case of another activity starting in front of the current activity. It is the opposite of onResume(). An example for this callback can be "when you open the app, that is another app from the notification bar, or open settings then it will call onPause() and onStop()".

onStop():- It is the opposite of onStart(). In this case, activity will no longer be visible to the users.

onRestart():- This occurs when the activity is stopped and before the activity starts again. It is a less common callback.

onDestroy():- It is opposite of onCreate(). This starts when the system needs to free memory or when finish() is called. It is used for cleanup which is a very common activity.

USER INTERFACE:

User interface is the first impression of a software system from the user's point of view. Therefore any software system must satisfy the requirement of user. UI mainly performs two functions −

- Accepting the user's input
- Displaying the output

User interface plays a crucial role in any software system. It is possibly the only visible aspect of a software system as −

- Users will initially see the architecture of software system's external user interface without considering its internal architecture.
- A good user interface must attract the user to use the software system without mistakes. It should help the user to understand the software system easily without misleading information. A bad UI may cause market failure against the competition of software system.
- UI has its syntax and semantics. The syntax comprises component types such as textual, icon, button etc. and usability summarizes the semantics of UI. The quality of UI is characterized by its look and feel (syntax) and its usability (semantics).
- There are basically two major kinds of user interface − a) Textual b) Graphical.
- Software in different domains may require different style of its user interface for e.g. calculator need only a small area for displaying numeric numbers, but a big area for commands, A web page needs forms, links, tabs, etc.

Graphical User Interface

A graphical user interface is the most common type of user interface available today. It is a very user friendly because it makes use of pictures, graphics, and icons - hence why it is called 'graphical'.

It is also known as a **WIMP interface** because it makes use of −

- **Windows** − A rectangular area on the screen where the commonly used applications run.
- **Icons** − A picture or symbol which is used to represent a software application or hardware device.
- **Menus** − A list of options from which the user can choose what they require.
- **Pointers** − A symbol such as an arrow which moves around the screen as user moves the mouse. It helps user to select objects.

Design of user interface starts with task analysis which understands the user's primary tasks and problem domain. It should be designed in terms of User's terminology and outset of user's job rather than programmer's.

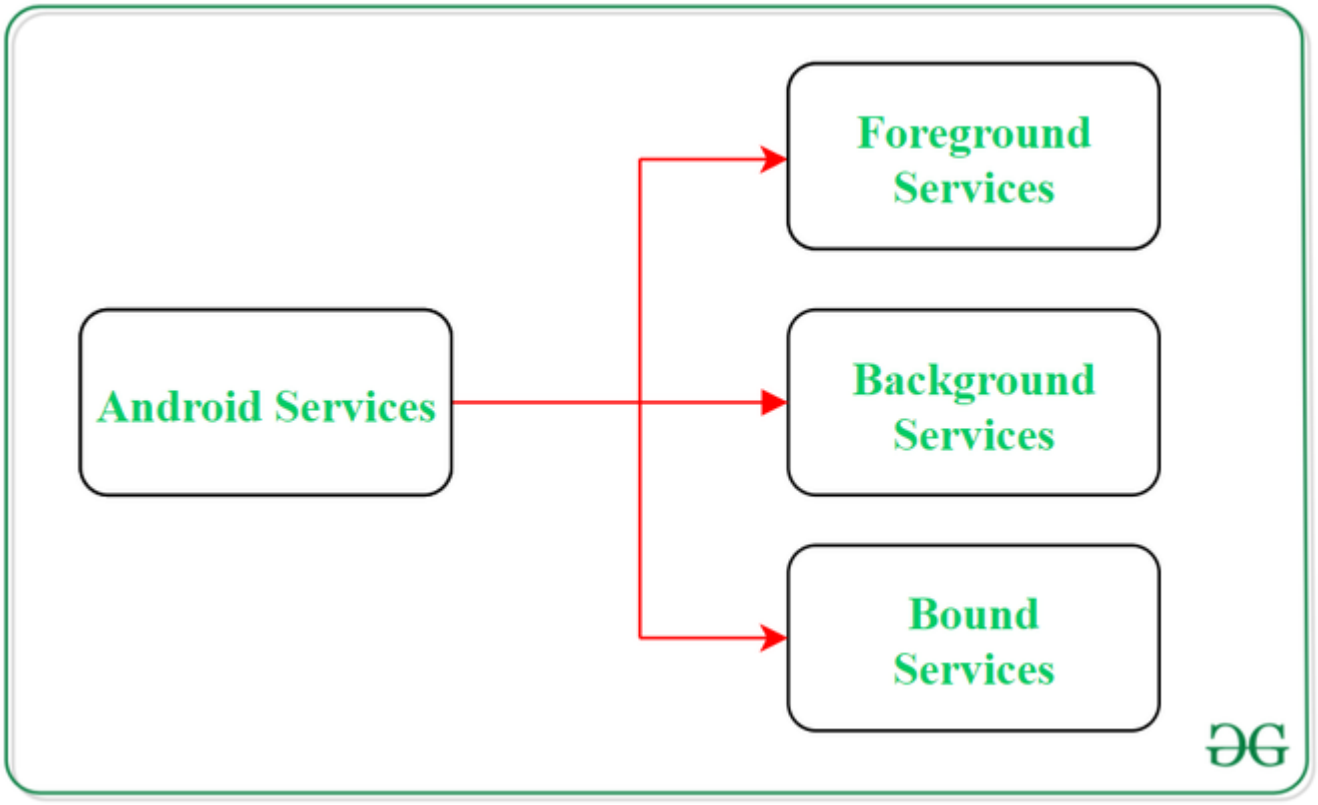
- To perform user interface analysis, the practitioner needs to study and understand four elements −
  - The **users** who will interact with the system through the interface
  - The **tasks** that end users must perform to do their work
  - The **content** that is presented as part of the interface
  - The **work environment** in which these tasks will be conducted
- Proper or good UI design works from the user's capabilities and limitations not the machines. While designing the UI, knowledge of the nature of the user's work and environment is also critical.
- The task to be performed can then be divided which are assigned to the user or machine, based on knowledge of the capabilities and limitations of each. The design of a user interface is often divided into four different levels −
  - **The conceptual level** − It describes the basic entities considering the user's view of the system and the actions possible upon them.
  - **The semantic level** − It describes the functions performed by the system i.e. description of the functional requirements of the system, but does not address how the user will invoke the functions.
  - **The syntactic level** − It describes the sequences of inputs and outputs required to invoke the functions described.
  - **The lexical level** − It determines how the inputs and outputs are actually formed from primitive hardware operations.
- User interface design is an iterative process, where all the iteration explains and refines the information developed in the preceding steps. General steps for user interface design
  - Defines user interface objects and actions (operations).
  - Defines events (user actions) that will cause the state of the user interface to change.
  - Indicates how the user interprets the state of the system from information provided through the interface.
  - Describe each interface state as it will actually look to the end user.

SERVICES:

Services in Android are a special component that facilitates an application to run in the background in order to perform long-running operation tasks. The prime aim of a service is to ensure that the application remains active in the background so that the user can operate multiple applications at the same time. A user-interface is not desirable for android services as it is designed to operate long-running processes without any user intervention. A service can run continuously in the background even if the application is closed or the user switches to another application. Further, application components can bind itself to service to carry out **inter-**

process communication(IPC). There is a major difference between android services and threads, one must not be confused between the two. Thread is a feature provided by the Operating system to allow the user to perform operations in the background. While service is an android component that performs a long-running operation about which the user might not be aware of as it does not have UI.

Types of Android Services



1. Foreground Services:

Services that notify the user about its ongoing operations are termed as Foreground Services. Users can interact with the service by the notifications provided about the ongoing task. Such as in downloading a file, the user can keep track of the progress in downloading and can also pause and resume the process.

2. Background Services:

Background services do not require any user intervention. These services do not notify the user about ongoing background tasks and users also cannot access them. The process like schedule syncing of data or storing of data fall under this service.

3. Bound Services:

This type of android service allows the components of the application like activity to bound themselves with it. Bound services perform their task as long as any application component is bound to it. More than one component is allowed to bind themselves with a service at a time. In order to bind an application component with a service **bindService()** method is used.

The Life Cycle of Android Services

In android, services have 2 possible paths to complete its life cycle namely **Started and Bounded**.

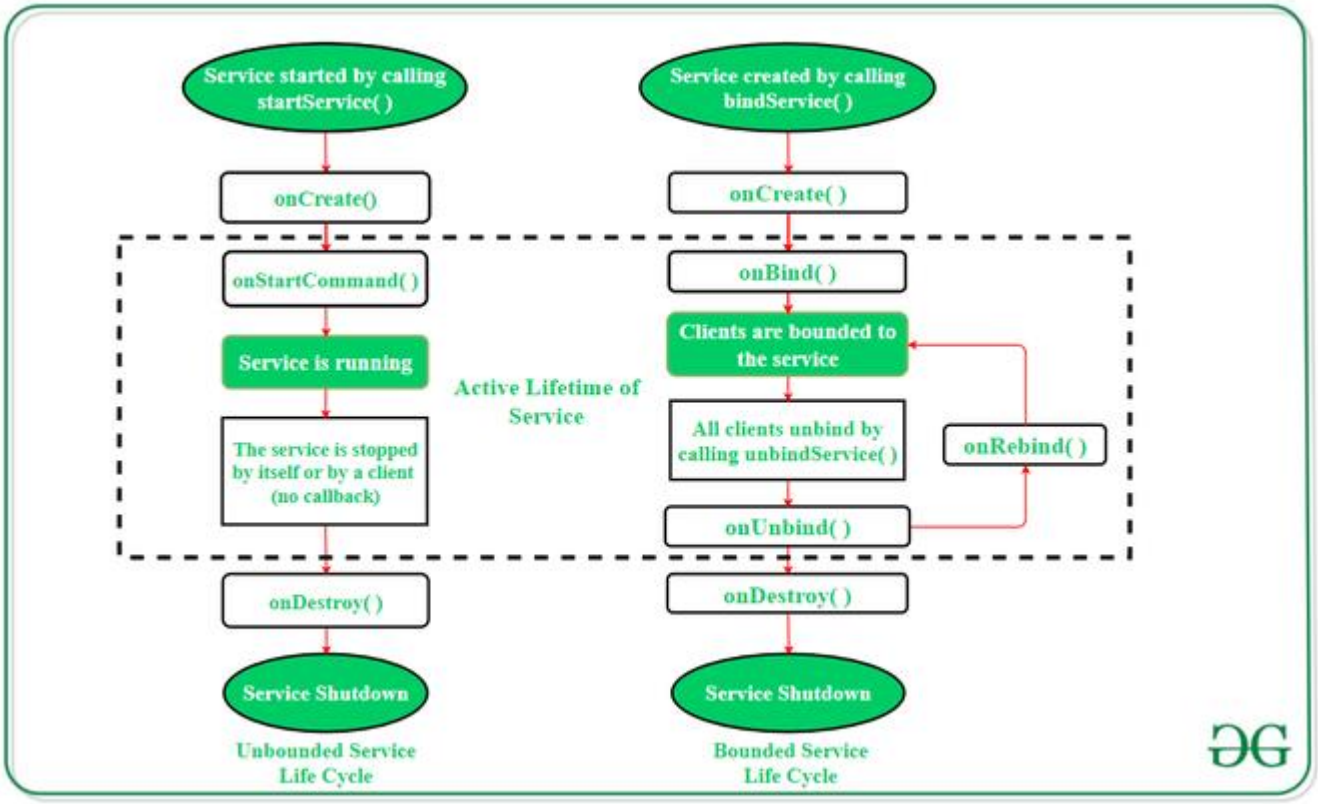
1. Started Service (Unbounded Service):

By following this path, a service will initiate when an application component calls the **startService()** method. Once initiated, the service can run continuously in the background even if the component is destroyed which was responsible for the start of the service. Two option are available to stop the execution of service:

- By calling **stopService()** method,
- The service can stop itself by using **stopSelf()** method.

2. Bounded Service:

It can be treated as a server in a client-server interface. By following this path, android application components can send requests to the service and can fetch results. A service is termed as bounded when an application component binds itself with a service by calling **bindService()** method. To stop the execution of this service, all the components must unbind themselves from the service by using **unbindService()** method.



WIDGETS:

A widget is a small gadget or control of your android application placed on the home screen. Widgets can be very handy as they allow you to put your favourite applications on your home screen in order to quickly access them. You have probably seen some common widgets, such as music widget, weather widget, clock widget e.t.c

Widgets could be of many types such as information widgets, collection widgets, control widgets and hybrid widgets. Android provides us a complete framework to develop our own widgets.

In order to create an application widget, first thing you need is AppWidgetProviderInfo object, which you will define in a separate widget XML file. In order to do that, right click on your project and create a new

folder called **xml**. Now right click on the newly created folder and create a new XML file. The resource type of the XML file should be set to **AppWidgetProvider**. In the xml file, define some properties which are as follows −

```
<appwidget-provider
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:minWidth="146dp"
   android:updatePeriodMillis="0"
   android:minHeight="146dp"
   android:initialLayout="@layout/activity_main">
</appwidget-provider>
```

a) Discuss Android Manifest file

Every project in Android includes a Manifest XML file, which is **AndroidManifest.xml**, located 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 Receivers** that make the application, and using **Intent Filters** and Permissions determines how they coordinate with each other and other applications.

The manifest file also specifies the application metadata, which includes its icon, version number, themes, etc., and additional top-level nodes can specify any required permissions, and unit tests, and define hardware, screen, or platform requirements. The manifest comprises a root manifest tag with a package attribute set to the project's package. It should also include an xmls:android attribute that will supply several system attributes used within the file. We use the versionCode attribute is used to define the current application version in the form of an integer that increments itself with the iteration of the version due to update. Also, the versionName attribute is used to specify a public version that will be displayed to the users.

A manifest file includes the nodes that define the application components, security settings, test classes, and requirements that make up the application.

The **AndroidManifest.xml file** *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

- 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.
- It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

A simple AndroidManifest.xml file looks like this:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.javatpoint.hello"
   android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
      android:minSdkVersion="8"
     android:targetSdkVersion="15" />

  <application
      android:icon="@drawable/ic_launcher"
     android:label="@string/app_name"
      android:theme="@style/AppTheme" >
     <activity
         android:name=".MainActivity"
        android:label="@string/title_activity_main" >
          <intent-filter>
           <action android:name="android.intent.action.MAIN" />

           <category android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
     </activity>
```

```
23.     </application>
24.
25. </manifest>
```

Elements of the AndroidManifest.xml file

The elements used in the above xml file are described below.

<manifest>

manifest is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

<application>

application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

The commonly used attributes are of this element are **icon**, **label**, **theme** etc.

android:icon represents the icon for all the android application components.

android:label works as the default label for all the application components.

android:theme represents a common theme for all the android activities.

<activity>

activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.

android:label represents a label i.e. displayed on the screen.

android:name represents a name for the activity class. It is required attribute.

<intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>

It adds a category name to an intent-filter.

b) Explain Anatomy of Android Application

There are four building blocks to an Android application:

- Activity
- Intent Receiver
- Service
- Content Provider

Not every application needs to have all four, but your application will be written with some combination of these.

Once you have decided what components you need for your application, you should list them in a file called AndroidManifest.xml. This is an XML file where you declare the components of your application and what their capabilities and requirements are. We will discuss soon, what the AndroidManifest.xml is responsible for.

Activity :

Activities are the most common of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the Activity base class. Your class will display a user interface composed of Views and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity.

When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.

Intent and Intent Filters :

Android uses a special class called Intent to move from screen to screen. Intent describe what an application wants done. The two most important parts of the intent data structure are the action and the data to act upon. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI). For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI.

```
new Intent(android.content.Intent.VIEW_ACTION, ContentURI.create("http://anddev
.org"));
```

There is a related class called an IntentFilter. While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or intent receiver, see below) is capable of handling. Activities publish their IntentFilters in the AndroidManifest.xml file.

Intent Receiver :

You can use an IntentReceiver when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight. Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened. Intent receivers are also registered in AndroidManifest.xml, but you can also register them from code using Context.registerReceiver().

Service :

A Service is code that is long-lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service

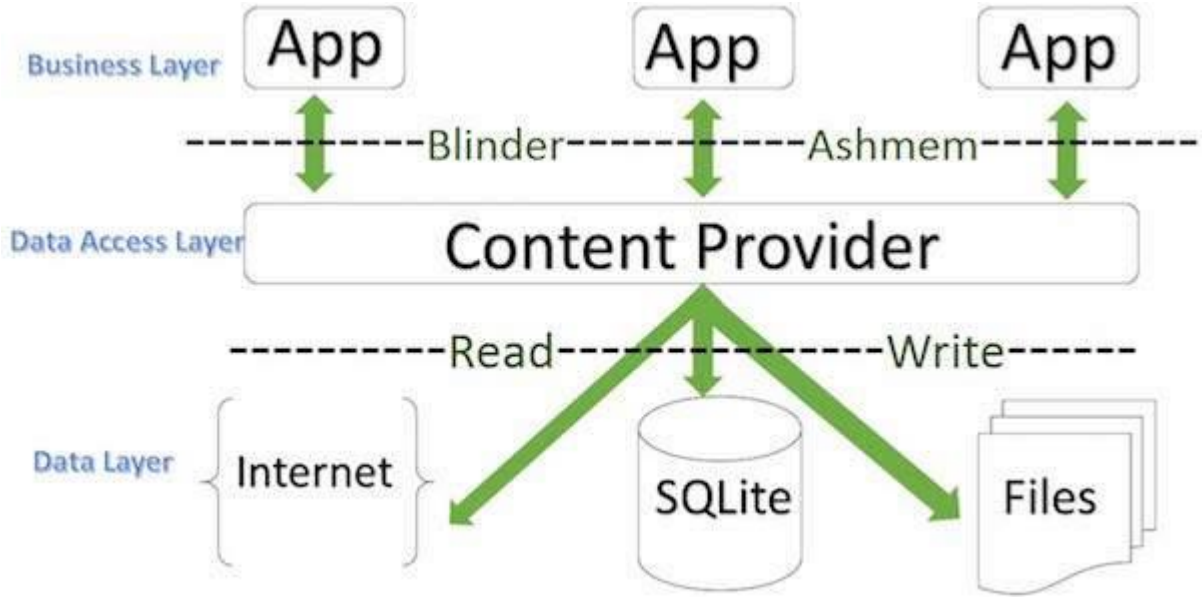
using Context.startService() to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. (You can learn more about the priority given to services in the system by reading Life Cycle of an Android Application.) Note that you can connect to a service (and start it if it's not already running) with the Context.bindService() method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

Content Provider :
Applications can store their data in files, a SQLite database, preferences or any other mechanism that makes sense. A content provider, however, is useful if you want your application's data to be shared with other applications. A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

a) Explain Receiving & Broadcasting Intent

b) Explain Content Provider

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.



ContentProvider

sometimes it is required to share data across applications. This is where content providers become very useful.

Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an **SQlite** database.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class My Application extends  ContentProvider {
}
```

Content URIs

To query a content provider, you specify the query string in the form of a URI which has following format –

```
<prefix>://<authority>/<data_type>/<id>
```

Here is the detail of various parts of the URI –

Sr.No	Part & Description
1	**prefix**

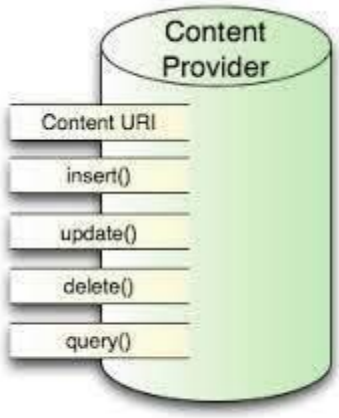
	This is always set to content://
2	**authority** This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider*
3	**data_type** This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this*content://contacts/people*
4	**id** This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*.

Create Content Provider

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the *ContentProviderbaseclass.*
- Second, you need to define your content provider URI address which will be used to access the content.
- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.
- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using <provider> tag.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working –



ContentProvider

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

a) Write a short note on
   1) Intent Filter
   2) User Permission

a) Write a short note on
1) XML File
2) Java File

XML stands for Extensible Markup Language. XML is a markup language much like HTML used to describe data. It is derived from Standard Generalized Markup Language(SGML). Basically, the XML tags are not predefined in XML. We need to implement and define the tags in XML. XML tags define the data and used to store and organize data. It's easily scalable and simple to develop. In Android, the XML is used to implement UI-related data, and it's a lightweight markup language that doesn't make layout heavy. XML only contains tags, while implementing they need to be just invoked.

The simple syntax of XML is

```
<tag_name>Hello World!</tag_name>
```

Basically in Android XML is used to implement the UI-related data. So understanding the core part of the UI interface with respect to XML is important. The User Interface for an Android App is built as the hierarchy of main **layouts, widgets**. The layouts are **ViewGroup** objects or containers that control how the child view should be positioned on the screen. **Widgets** here are view objects, such as Buttons and text boxes.

JAVA FILES:

Java File class represents the files and directory pathnames in an abstract manner. This class is used for creation of files and directories, file searching, file deletion, etc.

The File object represents the actual file/directory on the disk. Following is the list of constructors to create a File object.

Sr.No.	Method & Description
1	**File(File parent, String child)** This constructor creates a new File instance from a parent abstract pathname and a child pathname string.
2	**File(String pathname)** This constructor creates a new File instance by converting the given pathname string into an abstract pathname.
3	**File(String parent, String child)** This constructor creates a new File instance from a parent pathname string and a child pathname string.
4	**File(URI uri)** This constructor creates a new File instance by converting the given file: URI into an abstract pathname.

Once you have *File* object in hand, then there is a list of helper methods which can be used to manipulate the files.

Sr.No.	Method & Description
1	**public String getName()** Returns the name of the file or directory denoted by this abstract pathname.
2	**public String getParent()** Returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
3	**public File getParentFile()** Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.
4	**public String getPath()** Converts this abstract pathname into a pathname string.
5	**public boolean isAbsolute()** Tests whether this abstract pathname is absolute. Returns true if this abstract pathname is absolute, false otherwise.
6	**public String getAbsolutePath()** Returns the absolute pathname string of this abstract pathname.
7	**public boolean canRead()** Tests whether the application can read the file denoted by this abstract pathname. Returns true if and only if the file specified by this abstract pathname exists and can be read by the application; false otherwise.
8	**public boolean canWrite()** Tests whether the application can modify to the file denoted by this abstract pathname. Returns true if and only if the file system actually contains a file denoted by this abstract pathname and the application is allowed to write to the file; false otherwise.
9	**public boolean exists()** Tests whether the file or directory denoted by this abstract pathname exists. Returns true if and only if the file or directory denoted by this abstract pathname exists; false otherwise.
10	**public boolean isDirectory()** Tests whether the file denoted by this abstract pathname is a directory. Returns true if and only if the file denoted by this abstract pathname exists and is a directory; false otherwise.
11	**public boolean isFile()** Tests whether the file denoted by this abstract pathname is a normal file. A file is normal if it is not a directory and, in addition, satisfies other system-dependent criteria. Any non-directory file created by a Java application is guaranteed to be a normal file. Returns true if and only if the file denoted by this abstract pathname exists and is a normal file; false otherwise.
12	**public long lastModified()**

Returns the time that the file denoted by this abstract pathname was last modified. Returns a long value representing the time the file was last modified, measured in milliseconds since the epoch (00:00:00 GMT, January 1, 1970), or 0L if the file does not exist or if an I/O error occurs.

public long length()

13 Returns the length of the file denoted by this abstract pathname. The return value is unspecified if this pathname denotes a directory.

public boolean createNewFile() throws IOException

14 Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. Returns true if the named file does not exist and was successfully created; false if the named file already exists.

public boolean delete()

15 Deletes the file or directory denoted by this abstract pathname. If this pathname denotes a directory, then the directory must be empty in order to be deleted. Returns true if and only if the file or directory is successfully deleted; false otherwise.

public void deleteOnExit()

16 Requests that the file or directory denoted by this abstract pathname be deleted when the virtual machine terminates.

public String[] list()

17 Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.

public String[] list(FilenameFilter filter)

18 Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.

public File[] listFiles()

20 Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.

public File[] listFiles(FileFilter filter)

21 Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.

public boolean mkdir()

22 Creates the directory named by this abstract pathname. Returns true if and only if the directory was created; false otherwise.

public boolean mkdirs()

23 Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories. Returns true if and only if the directory was created, along with all necessary parent directories; false otherwise.

public boolean renameTo(File dest)

24 Renames the file denoted by this abstract pathname. Returns true if and only if the renaming succeeded; false otherwise.

public boolean setLastModified(long time)

25

Sets the last-modified time of the file or directory named by this abstract pathname. Returns true if and only if the operation succeeded; false otherwise.

public boolean setReadOnly()

26 Marks the file or directory named by this abstract pathname so that only read operations are allowed. Returns true if and only if the operation succeeded; false otherwise.

public static File createTempFile(String prefix, String suffix, File directory) throws IOException

27 Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name. Returns an abstract pathname denoting a newly-created empty file.

public static File createTempFile(String prefix, String suffix) throws IOException

28 Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name. Invoking this method is equivalent to invoking createTempFile(prefix, suffix, null). Returns abstract pathname denoting a newly-created empty file.

public int compareTo(File pathname)

29 Compares two abstract pathnames lexicographically. Returns zero if the argument is equal to this abstract pathname, a value less than zero if this abstract pathname is lexicographically less than the argument, or a value greater than zero if this abstract pathname is lexicographically greater than the argument.

public int compareTo(Object o)

30 Compares this abstract pathname to another object. Returns zero if the argument is equal to this abstract pathname, a value less than zero if this abstract pathname is lexicographically less than the argument, or a value greater than zero if this abstract pathname is lexicographically greater than the argument.

public boolean equals(Object obj)

31 Tests this abstract pathname for equality with the given object. Returns true if and only if the argument is not null and is an abstract pathname that denotes the same file or directory as this abstract pathname.

public String toString()

32 Returns the pathname string of this abstract pathname. This is just the string returned by the getPath() method.

a)Explain various Android Terminologies

BASIC TERMINOLOGIES OF ANDROID

Android is an 'Open-source' operating system used to develop an application for mobile devices. It is a Linux based operating system. Initially, it was tenured by 'Open alliance handset' and in 2007 it was occupied by google. The source code published by Google is under the Apache License version 2.0. Android smartphones are used by millions of people worldwide. Under researches, researchers say that around 3 million people are using an android smartphone. The android application completely transforms the way to communicate and interact with each other even in the miles of distance.

Architecture of Android

The android architecture is the combination of four layers. However, one layer is partitioned into two sections so, it consists of five sections in all.

- ***LINUX KERNEL:***

  It is the lowest layer of android architecture. It serves as the abstraction between the hardware level of the device and contains all the required hardware drivers.

- ***LIBRARIES:***

  It holds libraries for storing the data along the libraries uses for internet security purposes.

- ***ANDROID RUNTIME:***

  This section has the libraries which enable the developer to write the code of android application using any standard programming language.

- ***APPLICATION FRAMEWORK:***

  It is the third layer. It serves the services to the application layer in the form of java classes.

- ***Application:***

  It is the foremost layer that is used to install the applications in the devices.

Essential Tools for android application development

The most important tools required for application development are:

- JDK: Java Development kit that is the combination of two premier tools JAVA VIRTUAL MACHINE and JAVA RUNTIME ENVIRONMENT. It helps in compiling and running the java program.
- ANDROID STUDIO: The primary goal to introduce the android studio is to develop the android applications in the most stable tool. At first android applications were developed in eclipse but, app developers face multiple issues in eclipse. Then in 2007, google brings the android studio that is used particularly introduced for android.

Terminologies Correlated to Android

- XML file

  The preeminent file is used for the structure of an android project. It has complete information about all the components and packages. It initializes the API that is further used by an application.

- View

  It is the component of the User Interface that occupies the rectangular area on the screen.

- Layout

  It properly aligned the views on the screen.

- Activity

  Activity is a User interface screen through which the user interacts. Users have a right to place the UI elements in any way according to the Users choice.

- Emulator

  The emulator is the virtual device smartphone provided with an android studio. You can run your created application on the emulator and test its UI and function according to the needs.

- Intent

  It acts as a communicating object. You can establish a communication between two or more than two components as services, broadcast receivers. It is used to start and end the activity and services components.

- Services

It is used to run the process even in the background. There is no defined UI for service. Any component can start the service and end the services. You can easily switch between the applications even if the services are running the background.

- Content Provider

  It implemented in two ways:

1. You can use implement the existing content provider in your application.
2. However, you can also create a new content provider that will provide or share the data with other applications.

b) Explain Architecture of Android

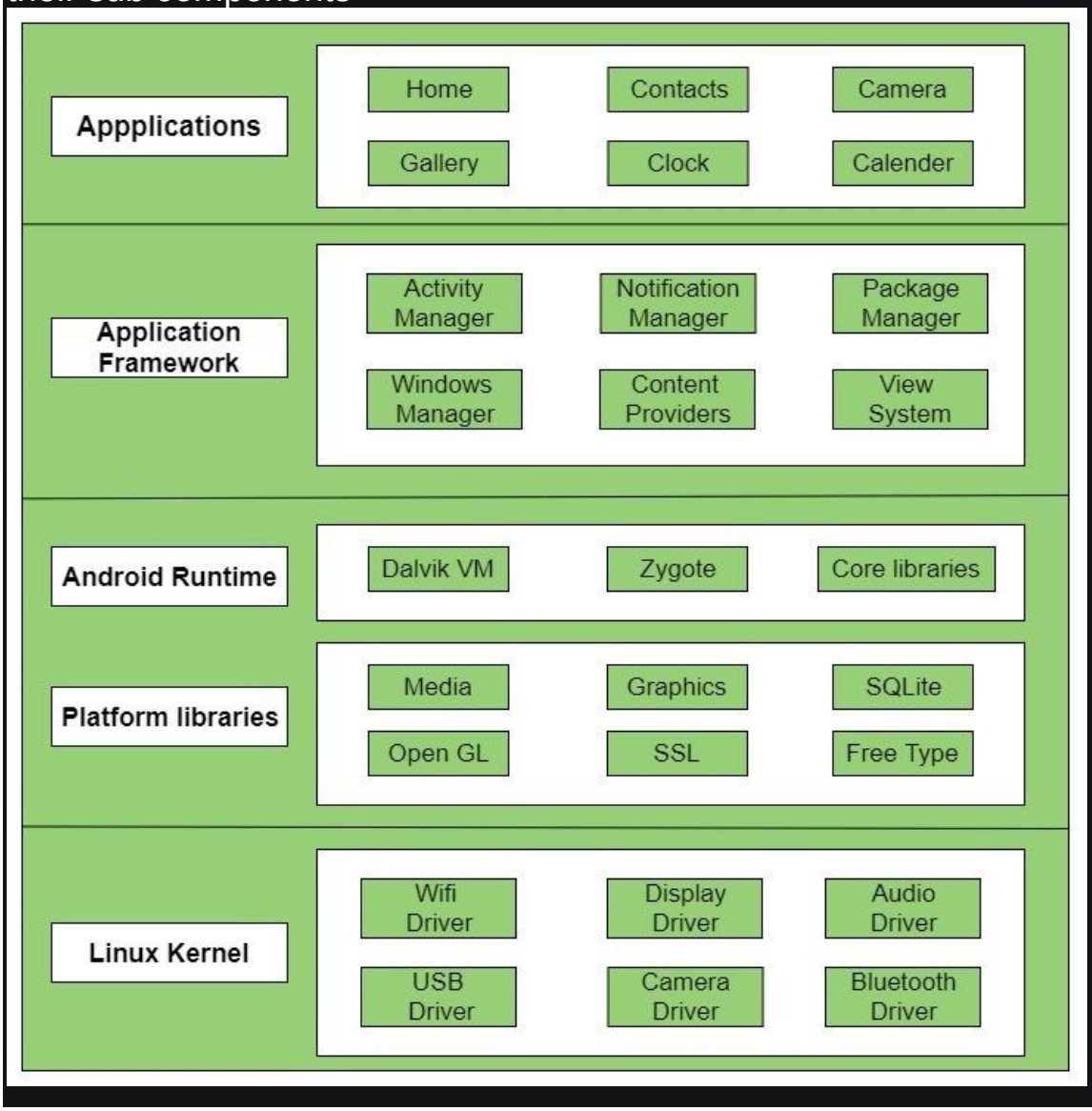
Android architecture contains different number of components to support any android device needs. Android software contains an open-source Linux Kernel having collection of number of C/C++ libraries which are exposed through an application framework services.

Among all the components Linux Kernel provides main functionality of operating system functions to smartphones and Dalvik Virtual Machine (DVM) provide platform for running an android application.

The main components of android architecture are following:-

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

Pictorial representation of android architecture with several main components and their sub components –



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.

On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using linux kernal.

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. Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation.

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.

On the top of Native libraries and android runtime, there is android framework. Android framework includes **Android API's** such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

Application runtime –
Android Runtime environment is one of the most important part of Android. It contains components like core libraries and the Dalvik virtual machine(DVM). Mainly, it provides the base for the application framework and powers our application with the help of the core libraries.

Like Java Virtual Machine (JVM), **Dalvik Virtual Machine (DVM)** is a register-based virtual machine and specially designed and optimized for android to ensure that a device can run multiple instances efficiently. It depends on the layer Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

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 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.

It is the heart of android architecture that exists at the root of android architecture. **Linux kernel** is responsible for device drivers, power management, memory management, device management and resource access.

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.

a) Explain

1) src. 2) res 3) assets 4) value

b)What are the challenges of Android Development

1. API Compatibility

Many developers use third-party APIs to improve the functionality and compatibility of mobile devices. However, not every third-party API available to developers for Android app development is of the highest quality. Some APIs were developed to work with a particular Android version and won't work on devices running another edition of the OS. Developers typically must develop methods to make the same API work across the various Android versions, which they usually find extremely challenging.

2. Software Fragmentation

Different devices run various versions, and this presents a huge task for designers to stay on top of modifications.

However, the majority of Android App Developers use the most recent version. We always state that the use of advanced technology can help users. However, the acceptance rate of the most current version of Android is low. For example, Android 6.0 Marshmallow currently has a greater percentage of market shares over Android 7.0 Nougat as well as Android 8.0 Oreo. Therefore, developers need to consider multiple versions of Android when creating mobile applications.

Android App Developers, the most trusted Android apps developer in India, creates full-fledged Android apps that are extremely compatible with security, safety, and data protection.

3. Hardware Specifications

Unlike other operating systems on mobiles, Android is open source, and Google lets device makers modify, expand, and extend the Android operating system with no limitations. Therefore, the hardware features offered by different Android devices vary. For instance, Android devices made by specific firms have fingerprint sensors, while other devices do not have support for fingerprint sensors. So, developers can't create a mobile application that can provide a more user-friendly experience by using the device's fingerprint sensor. The app must implement various authentication methods to match the different physical features of Android devices.

4. Testing Fragmentation

Every device comes with its unique testing environment, and testers must follow various testing methods each time. Although there are tools to automate testing, strategies depend on the app's design developed for app development.

5. UI Design Rules

The user interface of an app is a crucial factor in determining the retention rate of users. It improves the look of the application and also its compatibility with other functions on one page. Google does not offer any standard user interface for Android application development services. However, every company must design custom UI interfaces that meet their requirements and standards.

The Android developers should look at different ways to design a more user-friendly UI layout across all devices. Additionally, they should be vigilant at this point as it can impact the overall user experience of the Android application.

6. Security Concerns

Android is an open-source operating system; It is more susceptible to security concerns. Even with its security, it is still possible that malware could be introduced into the apps, redirecting users to other URLs without the developer's permission.

7. Patent Issues

Google does not have any guidelines to evaluate the level of quality in the new apps being added to the Play Store. The absence of quality assessment guidelines creates various patent-related problems for developers. To stay clear of patent-related issues, certain developers must modify or redesign their apps to avoid patent issues in the future.

Based on our own experiences, We've tried to tackle the common issues encountered by Android app developers. We are sure being alert to these obstacles will allow developers to build successful apps without hassle.

a) What is Drawable ? Explain type of drawable

A drawable resource is a general concept for a graphic that can be drawn to the screen and which you can retrieve with APIs such as `getDrawable(int)` or apply to another XML resource with attributes such as `android:drawable` and `android:icon`. There are several different types of drawables:

Bitmap File
: A bitmap graphic file (`.png`, `.webp`, `.jpg`, or `.gif`). Creates a `BitmapDrawable`.

Nine-Patch File
: A PNG file with stretchable regions to allow image resizing based on content (`.9.png`). Creates a `NinePatchDrawable`.

Layer List
: A Drawable that manages an array of other Drawables. These are drawn in array order, so the element with the largest index is be drawn on top. Creates a `LayerDrawable`.

State List
: An XML file that references different bitmap graphics for different states (for example, to use a different image when a button is pressed). Creates a `StateListDrawable`.

Level List
: An XML file that defines a drawable that manages a number of alternate Drawables, each assigned a maximum numerical value. Creates a `LevelListDrawable`.

Transition Drawable
: An XML file that defines a drawable that can cross-fade between two drawable resources. Creates a `TransitionDrawable`.

Inset Drawable
: An XML file that defines a drawable that insets another drawable by a specified distance. This is useful when a View needs a background drawable that is smaller than the View's actual bounds.

Clip Drawable
: An XML file that defines a drawable that clips another Drawable based on this Drawable's current level value. Creates a `ClipDrawable`.

Scale Drawable
: An XML file that defines a drawable that changes the size of another Drawable based on its current level value. Creates a `ScaleDrawable`

Shape Drawable
: An XML file that defines a geometric shape, including colors and gradients. Creates a `GradientDrawable`.

Also see the Animation Resource document for how to create an `AnimationDrawable`.

b) Write a short note on
1) Toast Notification
2) SQLite

A toast **provides simple feedback about an operation in a small popup**. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

Andorid Toast can be used to display information for the short period of time. A toast contains message to be displayed quickly and disappears after sometime.

The android.widget.Toast class is the subclass of java.lang.Object class.

You can also create custom toast as well for example toast displaying image. You can visit next page to see the code for custom toast.

Toast class is used to show notification for a particular interval of time. After sometime it disappears. It doesn't block the user interaction.

There are only 2 constants of Toast class which are given below.

Constant	Description
public static final int LENGTH_LONG	displays view for the long duration of time.
public static final int LENGTH_SHORT	displays view for the short duration of time.

SQLite is an **open-source relational database** i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.

It is embedded in android bydefault. So, there is no need to perform any database setup or administration task.

Here, we are going to see the example of sqlite to store and fetch the data. Data is displayed in the logcat. For displaying data on the spinner or listview, move to the next page.

SQLiteOpenHelper class provides the functionality to use the SQLite database.

The android.database.sqlite.SQLiteOpenHelper class is used for database creation and version management. For performing any database operation, you have to provide the implementation of **onCreate()** and **onUpgrade()** methods of SQLiteOpenHelper class.

The main package is android.database.sqlite that contains the classes to manage your own databases

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise


a) Discuss in detail about Android Platform. List steps for building Android Application

b) What are the design standards of a user interface? and its types.

a) Explain SDK & its Benefits

a software development kit (SDK) is made up of lines of code. SDK is a downloadable software package that contains the tools you need to build on a platform. Some SDKs are crucial for developing a platform-specific app. For example, the development of an Android app on Java platform needs a Java Development Kit, for iOS apps the iOS SDK, and for Universal Windows Platform the .NET Framework SDK. Some SDKs are installed in apps to provide analytics and data about the activity. Leading cases include Google, Apple, and Facebook.
SDK's can be used in cases as simple as the implementation of one or more application programming interfaces (APIs) in the form of some libraries to interface to a particular programming language or to include advanced hardware that can communicate with an appropriate embedded system. SDKs are widely associated with mobile native apps, it's possible to make use of them in connection with websites, set-top boxes, and other digital platforms. A mobile SDK is used for building a mobile application for iOS and Android devices. It provides resources for the same and customizes the applications.

A software development kit (SDK) is a set of software tools and programs provided by hardware and software vendors that developers can use to build applications for specific platforms. SDKs help developers easily integrate their apps with a vendor's services.

SDKs include documentation, application programming interfaces (APIs), code samples, libraries and processes, as well as guides that developers can use and integrate into their apps. Developers can use SDKs to build and maintain applications without having to write everything from scratch.

The following are the key benefits of using SDKs:

- **Time saving.** SDKs let developers easily and quickly build the standard components of their apps and add functionality to them. SDKs are usually all-in-one products and don't need to be integrated with other components, which can slow down the development process.
- **Easier integration.** Developers use SDKs for simple functions, such as logging in, location services and mobile payments. However, some SDKs help developers build

more complex app features, such as augmented reality and virtual reality, and add new features. SDKs reduce the complexity of integrations by simplifying standard processes, such as creating authorization signatures and interpreting SMS messages in native languages or platforms.

- **Documentation and code libraries.** SDKs include documentation, tutorials APIs, code samples, libraries and processes. They also provide guides developers can use and integrate into their apps. Developers use SDKs to build and maintain applications without having to write everything from scratch.
- **Enhanced functionality.** SDKs let developers enhance apps with more functionality, such as push notifications and ads. SDKs also help developers create new tools and make the process easier because everything is prebuilt. For example, if a developer wants to share images or text from an app directly on Facebook, they could look for Facebook's Android SDK to find the necessary code that would work for an Android device. This speeds deployment because the developer doesn't have to write the code from scratch.
- **Brand credibility.** Apps supported by an SDK toolkit gain exposure as they can get published on app stores where customers search for and buy apps. These apps are less prone to unexpected crashes and have easier third-party integrations, giving them better ratings in the app store and boosting their brand's credibility.
- **Cost savings.** Because of their shorter development cycles, apps built with SDKs can offer substantial cost savings. In addition, SDK integrations don't require specialized technical skills, which lets organizations perform in-house integrations rather than pay to hire outside professionals.
- **Customization.** SDKs are available for a variety of use cases and provide the ability to develop apps with personalized user experiences.

b) Explain Design User Interface Architecture