Android Interview Questions

Ques 1: What is Android? Reference

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 offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.

Why Android ?

Open Source
Larger Developer and Community Reach
Increased Marketing
Inter App Integration
Reduce Cost of Development
Higher Success Ratio
Rich Development Environment

Features of Android:

Beautiful UI:
Android OS basic screen provides a beautiful and intuitive user interface.
Connectivity
GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
Storage
SQLite, a lightweight relational database, is used for data storage purposes. Other are Realm, File storage etc.
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.
Messaging
SMS and MMS
Web browser
Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
Multi-touch
Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
Multi-tasking
User can jump from one task to another and same time various application can run simultaneously.
Resizable widgets
Widgets are resizable, so users can expand them to show more content or shrink them to save space.
Multi-Language
Supports single direction and bi-directional text.

Ques 2: Who is the founder of Android? Reference

Andy Rubin.

Ques 3: Explain the Android application Architecture. Reference

Android Architecture

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 Frameworks
Android Runtime
Platform Libraries
Linux Kernel

Applications:

Application 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 Frameworks:

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.

Android 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.

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 the last layer of android architecture and also 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.

Ques 4: What are the code names of android? Reference

Aestro
Blender
Cupcake
Donut
Eclair
Froyo
Gingerbread
Honeycomb
Ice Cream Sandwich
Jelly Bean
KitKat
Lollipop
Marshmallow
Nougat
Oreo
Pie

Ques 5: What are the advantages of Android? Reference

Android is an open-source platform allowing UI customization

Licensed under Apache, Android is an open source operating system whose codes developers can change to build customized User Interface. App developers building applications for this platform can get access to the core codes and are at a liberty to change the them to get the customized outcomes. This is not possible when it comes to iOS and app have to strictly adhere to the core code specifications when developing apps for the specific platform.

Continual improvement & removal of old features

Google Android is supported by a huge community of developers and also users who continue to give feedback about the features, their pros and cons. As a result, there is continuous check on the codes and features, making modifications and alterations, bringing in better upgrades all the time. This is one of the reasons why Android is always adding new features while removing older ones that users do not like.

Supports running multiple apps simultaneously

With Android running on a device with good hardware specification, as a user you can have multiple apps running simultaneously. You can continue to listen to music as you check your messages or download files that you've received or even upload them from your device or drive.

Expandable memory & runs on affordable large devices

One of the biggest advantages of using devices running on Android platform is that it supports expandable memory. iOS devices on the other hand do not support external memory expansion by adding memory card to the phone. Users of this platform enjoy the privilege of storing e-books, music, videos and games on their devices.

Affordable Development

You can easily get your own app out in the android market and you don't need to pay big development fees. The application environments are free of charge and you can save a lot of money and there is no limit which means that if you want to make more than one app, you can do that without any problems.

More

Android Is More Customizable Can change almost anything.

In Android, any new publication can be done easily and without any review process

Use a Different Messaging App for SMS

Android Offers an Open Platform

Easy access to the Android App Market

Cost Effective

Upcoming versions have a support to save RAW images

Built in Beta Testing and staged rollout

Ques 6: Does android support other languages than java? Reference

Yes, an android app can be developed in C/C++ also using android NDK (Native Development Kit). It makes the performance faster. It should be used with Android SDK.

Ques 7: What are the core building blocks of android? Reference

The core building blocks or fundamental components of android are:

Activity:

Being similar to a Frame in AWT, an activity as a class represents a single screen. It is a core building block, i.e., the fundamental component of android.

View:

The UI element including button, label, text field, etc, and anything that one can see is a view.

Intent:

Along with invoking the components, the Intent is used for many purposes including:

Start the service
Launch an activity
Display a web page
Display a list of contacts
Broadcast a message
Dial a phone call etc.
Example: To view the webpage:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse('http://www.example.com'));
startActivity(intent);
```

Types of Intent:

1. **Implicit Intent**
2. **Explicit Intent**

Service:

The component of android which is used to run the long running tasks into the background. The service don't have any UI

Types of Services:

1. **Background Services**
2. **Foreground Services**
3. **Bound Services**

Content Provider:

Use to share the data between the applications.

Fragment:

Being a part of an activity, one or more fragments can be displayed on the screen at the same time by the activity. It have it's own UI and lifecycle.

AndroidManifest.xml:

Information about activities, content providers, permissions, etc is in the AndroidManifest.xml which is like the web.xml file in Java EE.

Ques 8: What is activity in Android? Reference

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI

with `setContentView(View)`. While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with `R.attr.windowIsFloating` set), Multi-Window mode or embedded into other windows. There are two methods almost all subclasses of Activity will implement:

`onCreate(Bundle)` is where you initialize your activity. Most importantly, here you will usually call `setContentView(int)` with a layout resource defining your UI, and using `findViewById(int)` to retrieve the widgets in that UI that you need to interact with programmatically.
`onPause()` is where you deal with the user pausing active interaction with the activity. Any changes made by the user should at this point be committed (usually to the `ContentProvider` holding the data). In this state the activity is still visible on screen.

Activity Lifecycle

Activities in the system are managed as activity stacks. When a new activity is started, it is usually placed on the top of the current stack and becomes the running activity -- the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits. There can be one or multiple activity stacks visible on screen.

An activity has essentially four states:

If an activity is in the foreground of the screen (at the highest position of the topmost stack), it is ***active* or *running***. This is usually the activity that the user is currently interacting with.
If an activity has lost focus but is still presented to the user, it is ***visible***. It is possible if a new non-full-sized or transparent activity has focus on top of your activity, another activity has higher position in multi-window mode, or the activity itself is not focusable in current windowing mode. Such activity is completely alive (it maintains all state and member information and remains attached to the window manager).
If an activity is completely obscured by another activity, it is ***stopped* or *hidden***. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.
The system can drop the activity from memory by either asking it to finish, or simply killing its process, making it ***destroyed***. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

Ques 9: What are the life cycle methods of android activity? Reference

There are 7 life-cycle methods of activity. They are as follows:

1. onCreate()

   Called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc. This method also provides you with a Bundle containing the activity's previously frozen state, if there was one.

   Always followed by `onStart()`.

2. onStart()

   Called when the activity is becoming visible to the user. Followed by `onResume()` if the activity comes to the foreground, or `onStop()` if it becomes hidden.

3. onRestart()

   Called after your activity has been stopped, prior to it being started again. Always followed by `onStart()`

4. onResume()

   Called when the activity will start interacting with the user. At this point your activity is at the top of its activity stack, with user input going to it. Always followed by `onPause()`.

5. onPause()

   Called when the activity loses foreground state, is no longer focusable or before transition to stopped/hidden or destroyed state. The activity is still visible to user, so it's recommended to keep it visually active and continue updating the UI. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either `onResume()` if the activity returns back to the front, or `onStop()` if it becomes invisible to the user.

6. onStop()

   Called when the activity is no longer visible to the user. This may happen either because a new activity is being started on top, an existing one is being brought in front of this one, or this one is being destroyed. This is typically used to stop animations and refreshing the UI, etc. Followed by either `onRestart()` if this activity is coming back to interact with the user,

or `onDestroy()` if this activity is going away.

7. **onDestroy()**

   The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called `Activity#finish` on it), or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the `isFinishing()` method.

Ques 10: What is intent? Reference

It is a kind of message or information that is passed to the components. It is used to launch an activity, display a web page, send SMS, send email, etc. There are two types of intents in android:

1. Implicit Intent
2. Explicit Intent

Ques 11: How are view elements identified in the android program? Reference

View elements can be identified using the keyword findViewById. But with view binding and data binding we can access them directly by id.

Ques 12: Define Android toast. Reference

An android toast provides feedback to the users about the operation being performed by them. It displays the message regarding the status of operation initiated by the user.

Ques 13: Give a list of impotent folders in android Reference

The following folders are declared as impotent in android:

AndroidManifest.xml
build.xml
bin/
src/
res/
assets/

Ques 14: Explain the use of 'bundle' in android? Reference

We use bundles to pass the required data to various subfolders.

Ques 15: What is an application resource file? Reference

The files which can be injected for the building up of a process are called as application resource file.

Ques 16: How are layouts placed in Android? Reference

Layouts in Android are placed as XML files.

Ques 17: Where are layouts placed in Android? Reference

Layouts in Android are placed in the layout folder.

Ques 18: What is the implicit intent in android? Reference

The Implicit intent is used to invoke the system components.

Ques 19: What is explicit intent in android? Reference

An explicit intent is used to invoke the activity class.

Ques 20: How to call another activity in android? Reference

```
Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
startActivity(i);
```

Ques 21: What is service in android? Reference

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.

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.

> To carry out a downloading task in the background, the **startService()** method will be called. Whereas to get information regarding the download progress and to pause or resume the process while the application is still in the background, **the service must be bounded with a component** which can perform these tasks.

A user-defined service can be created through a normal class which is extending the **class Service**. Further, to carry out the operations of service on applications, there are certain callback methods which are needed to be **overridden**. The following are some of the important methods of Android Services:

1. onStartCommand()
   The Android service calls this method when a component(eg: activity) requests to start a service using startService(). Once the service is started,it can be stopped explicitly using stopService() or stopSelf() methods.
2. onBind()

This method is mandatory to implement in android service and is invoked. Whenever an application component calls the bindService() method in order to bind itself with a service. User-interface is also provided to communicate, with the service effectively by returning an IBinder object. If the binding of service is not required then the method must return null.

3. onUnbind()

   The Android system invokes this method when all the clients get disconnected from a particular service interface.

4. onRebind()

   Once all clients are disconnected from the particular interface of service and there is a need to connect the service with new clients, the system calls this method.

5. onCreate()

   Whenever a service is created either using onStartCommand() or onBind(), the android system calls this method. This method is necessary to perform a one-time-set-up.

6. onDestroy()

   When a service is no longer in use, the system invokes this method just before the service destroys as a final clean up call. Services must implement this method in order to clean up resources like registered listeners, threads, receivers, etc.

Ques 22: What is the name of the database used in android? Reference

SQLite: An opensource and lightweight relational database for mobile devices.

Ques 23: What is AAPT? Reference

AAPT is an acronym for android asset packaging tool. It handles the packaging process.

Ques 24: What is a content provider? Reference

A content provider is used to share information between Android applications.

Ques 25: What is fragment? Reference

A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments cannot live on their own--they must be hosted by an activity or another fragment.

Ques 26: What is ADB? Reference

ADB stands for Android Debug Bridge. It is a versatile command-line tool that lets you communicate with a device. The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device.

Ques 27: What is the Google Android SDK? Reference

The Google Android SDK is a toolset which is used by developers to write apps on Android-enabled devices. It contains a graphical interface that emulates an Android-driven handheld environment and allows them to test and debug their codes.

Ques 28: Name the dialog box which is supported by Android? Reference

Alert Dialog
Progress Dialog
Date Picker Dialog
Time picker Dialog

Ques 29: Define Android Architecture Reference

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.

The main components of android architecture are following:-

Applications
Application Framework
Android Runtime
Platform Libraries
Linux Kernel

Ques 30: What do you mean by a drawable folder in Android? Reference

In Android, a drawable folder is compiled a visual resource that can use as a background, banners, icons, splash screen, etc.

Ques 31: What is a singleton class in Android? Reference

A singleton class is a class which can create only an object that can be shared by all other classes.

Ques 32: What is application Widgets in Android? Reference

Application widgets are miniature application views that can be embedded in other applications and receive periodic updates.

Ques 33: What is nine-patch images tool in Android? Reference

We can change bitmap images into nine sections with four corners, four edges, and an axis.

Ques 34: What is ADT in Android? Reference

ADT stands for Android Development Tool. It is used to develop the applications and test the applications.

Ques 35: Describe the Android Framework

The Android Framework is an important aspect of the Android Architecture. Here you can find all the classes and methods that developers would need in order to write applications on the Android environment.

Ques 36: Differentiate Activities from Services Reference

Activities can be closed, or terminated anytime the user wishes. On the other hand, services are designed to run behind the scenes, and can act independently. Most services run continuously, regardless of whether there are certain or no activities being executed.

Ques 37: What is Cold Start? Reference

A cold start refers to an app's starting from scratch: the system's process has not, until this start, created the app's process. Cold starts happen in cases such as your app's being launched for the first time since the device booted, or since the system killed the app. This type of start presents the greatest challenge in terms of minimizing startup time, because the system and app have more work to do than in the other launch states.

At the beginning of a cold start, the system has three tasks. These tasks are:

1. Loading and launching the app.
2. Displaying a blank starting window for the app immediately after launch.
3. Create the app process.

As soon as the system creates the app process, the app process is responsible for the next stages:

1. Creating the app object.
2. Launching the main thread.
3. Creating the main activity.
4. Inflating views.
5. Laying out the screen.
6. Performing the initial draw.

Ques 38: What is Worm Start? Reference

A warm start encompasses some subset of the operations that take place during a cold start; at the same time, it represents more overhead than a hot start. There are many potential states that could be considered warm starts. For instance:

The user backs out of your app, but then re-launches it. The process may have continued to run, but the app must recreate the activity from scratch via a call to `onCreate()`.

The system evicts your app from memory, and then the user re-launches it. The process and the activity need to be restarted, but the task can benefit somewhat from the saved instance state bundle passed into `onCreate()`.

Ques 39: What is Hot Start? Reference

A hot start of your application is much simpler and lower-overhead than a cold start. In a hot start, all the system does is bring your activity to the foreground. If all of your application's activities are still resident in memory, then the app can avoid having to repeat object initialization, layout inflation, and rendering.

Ques 40: What is WebView? Reference

WebView is a view that display web pages inside your application. You can also specify HTML string and can show it inside your application using WebView. WebView makes turns your application to a web application.

Ques 41: Android Services. Reference

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.

Types of **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.

Ques 42: What is Broadcast Receivers? Reference

Broadcast in android is the system-wide events that can occur when the device starts, when a message is received on the device or when incoming calls are received, or when a device goes to airplane mode, etc. Broadcast Receivers are used to respond to these system-wide events. **Broadcast Receivers** allow us to register for the system and application events, and when that event happens, then the register receivers get notified.

There are mainly two types of Broadcast Receivers:

1. Static Broadcast Receivers: These types of Receivers are declared in the manifest file and works even if the app is closed.
2. Dynamic Broadcast Receivers: These types of receivers work only if the app is active or minimized. Since from API Level 26, most of the broadcast can only be caught by the dynamic receiver

Public methods:

```
public final void abortBroadcast ()
public final void clearAbortBroadcast ()
```

public final boolean getAbortBroadcast ()
public final boolean getDebugUnregister ()
public final int getResultCode ()
public final String getResultData ()
public final boolean isOrderedBroadcast ()
public abstract void onReceive (Context context, Intent intent)

Ques 43: Whate is Intent Filter Reference

Specifies the types of intents that an activity, service, or broadcast receiver can respond to. An intent filter declares the capabilities of its parent component — what an activity or service can do and what types of broadcasts a receiver can handle. It opens the component to receiving intents of the advertised type, while filtering out those that are not meaningful for the component.
Most of the contents of the filter are described by its , , and subelements.

Ques 44: What is ANR? Reference

When the UI thread of an Android app is blocked for too long, an **"Application Not Responding"** (ANR) error is triggered. If the app is in the foreground, the system displays a dialog to the user. The ANR dialog gives the user the opportunity to force quit the app.

ANRs are a problem because the app's main thread, which is responsible for updating the UI, can't process user input events or draw, causing frustration to the user.

An ANR will be triggered for your app when one of the following conditions occur:

While your activity is in the foreground, your app has not responded to an input event or BroadcastReceiver (such as key press or screen touch events) within 5 seconds.
While you do not have an activity in the foreground, your BroadcastReceiver hasn't finished executing within a considerable amount of time.

Ques 45: What is Contente Resolver Reference

The **Content Resolver** is the single, global instance in your application that provides access to your (and other applications') content providers. The Content Resolver behaves exactly as its name implies: it accepts requests from clients, and *resolves* these requests by directing them to the content provider with a distinct authority. To do this, the Content Resolver stores a mapping from authorities to **Content Providers**. This design is important, as it allows a simple and secure means of accessing other applications' Content Providers.

The Content Resolver includes the **CRUD** (create, read, update, delete) methods corresponding to the abstract methods (insert, query, update, delete) in the Content Provider class. The Content Resolver does not know the implementation of the Content Providers it is interacting with (nor does it need to know); each method is passed an URI that specifies the Content Provider to interact with.

Ques 46: What is Sticky Intent? Reference

Sticky Intent: These are the Intents which sticks with Android for future broadcast listener.

Sticky Intent is also a type of Intent which allows communication between a function and a service sendStickyBroadcast(), performs a sendBroadcast(Intent) known as sticky, the **Intent you are sending stays around after the broadcast is complete, so that others can quickly retrieve that data through the return value of registerReceiver(BroadcastReceiver, IntentFilter)**. In all other ways, this behaves the same as sendBroadcast(Intent).

One example of a sticky broadcast sent via the operating system is ACTION_BATTERY_CHANGED. When you call registerReceiver() for that action — even with a null BroadcastReceiver — you get the Intent that was last Broadcast for that action. Hence, you can use this to find the state of the battery without necessarily registering for all future state changes in the battery.

Ques 47: Constraint Layout vs Relative Layout Reference

ConstraintLayout has **flat view hierarchy** unlike other layouts, so does **a better performance than relative layout**. Yes, this is the biggest advantage of Constraint Layout, the only single layout can handle your UI. Where in the Relative layout you needed multiple nested layouts (LinearLayout + RelativeLayout).

1. Does the ConstraintLayout have better performance than a nested Layout?

Yes, `ConstraintLayout` has designed with performance optimization in mind, more effective, easy use, and trying to eliminate as many pass scenarios as possible. This is done by eliminating the deeply-nested view hierarchies with flat view hierarchies.

2. Can we replace RelativeLayout with ConstraintLayout completly?

Yes, you can completely replace RelativeLayout with ConstraintLayout. `ConstraintLayout` does all that `RelativeLayout` does, and more.

3. Do we don't need to write many dimens.xml for all screen resolution to have a responsive app?

It depends on what you want to do – using dimensions might still be useful, for example- you might want the same layout but with different margins for different sizes of devices. That time you have to add many dimen.xml resource files.

Ques 48: What is Proguard in Android? Reference

Proguard is a great tool for creating a production-ready application in Android. It assists us in reducing code and making apps faster. Proguard is included by default in Android Studio and can help in a variety of ways, a few of which are listed below.

1. It obfuscates the code, which means that it changes the names to something smaller, such as A for MainViewModel. After obfuscating the app, reverse engineering becomes difficult.
2. It shrinks the resources, ignoring resources that are not called by our Class files, are not used in our Android app, such as images from drawable, and so on. This will significantly reduce the app's size. To keep your app light and fast, you should always shrink it.

Ques 49: What is Multidex in Android? Reference

In Android, the compilers convert your source code into DEX files. This DEX file contains the compiled code used to run the app. But there is a limitation with the DEX file. The DEX file limits the total number of methods that can be referenced within a single DEX file to 64K i.e. 65,536 methods. So, you can't use more than 64K methods in a particular DEX file. These 64K methods include Android framework methods, library methods, and methods in our code also. This limit of 64K is referred to as the "**64K reference limit**".

So, if our app exceeds 65,536 methods, we will encounter a build error that indicates our app has reached the limit of the Android build architecture. The error is as follows:

```
Too many field references: 131000; max is 65536. You may try using --multi-dex option.
```

Older versions of the build system report a different error, which also indicates the same problem:

```
Conversion to Dalvik format failed: Unable to execute dex: method ID not in [0, 0xffff]: 65536
```

Both the above error are known as **64K reference limit** i.e we are trying to use more than 64K methods in our code. So, here comes the role of Multidex support in our Android Project.

Ques 50: Android RecyclerView? Reference

RecyclerView makes it easy to efficiently display large sets of data. You supply the data and define how each item looks, and the RecyclerView library dynamically creates the elements when they're needed.

As the name implies, RecyclerView *recycles* those individual elements. When an item scrolls off the screen, RecyclerView doesn't destroy its view. Instead, RecyclerView reuses the view for new items that have scrolled onscreen. This reuse vastly improves performance, improving your app's responsiveness and reducing power consumption.

Note: Besides being the name of the class, RecyclerView is also the name of the library. In this page, `RecyclerView` in `code font` always means the class in the RecyclerView library.

Ques 51: Android Listview? Reference

A **ListView** is a type of AdapterView that displays a vertical list of scroll-able views and each view is placed one below the other. Using adapter, items are inserted into the list from an array or database. For displaying the items in the list method **`setAdaptor()`** is used. **`setAdaptor()`** method conjoins an adapter with the list.

Android ListView is a ViewGroup that is used to display the list of items in multiple rows and contains an adapter that automatically inserts the items into the list.

The main purpose of the adapter is to fetch data from an array or database and insert each item that placed into the list for the desired result. So, it is the main source to pull data from strings.xml file which contains all the required strings in Java or XML files.

Ques 52: What is Context on Android? Reference

it's the context of current state of the application/object. It lets newly-created objects understand what has been going on. Typically, you call it to get information regarding another part of your program (activity and package/application).

Ques 53: What are the permission protection levels in Android?

Normal — A lower-risk permission that gives requesting applications access to isolated application-level features, with minimal risk to other applications, the system, or the user. The system automatically grants this type of permission to a requesting application at installation, without asking for the user's explicit approval.
Dangerous — A higher-risk permission. Any dangerous permissions requested by an application may be displayed to the user and require confirmation before proceeding, or some other approach may be taken to avoid the user automatically allowing the use of such facilities.
Signature — A permission that the system grants only if the requesting application is signed with the same certificate as the application that declared the permission. If the certificates match, the system automatically grants the permission without notifying the user or asking for the user's explicit approval.
SignatureOrSystem — A permission that the system grants only to applications that are in the Android system image or that are signed with the same certificate as the application that declared the permission.

Ques 54: What is a JobScheduler? Reference

The **JobScheduler** API performs an operation for your application when a set of predefined conditions are met (such as when a device is plugged into a power source or connected to a Wi-Fi network). This allows your app to perform the given task while being considerate of the device's battery at the cost of timing control.

Unlike the **AlarmManager** class, the timing isn't exact. Compared to a custom **SyncAdapter** or the **AlarmManager**, the JobScheduler supports batch scheduling of jobs. The Android system can combine jobs so that battery consumption is reduced. JobManager makes handling uploads easier as it handles automatically the unreliability of the network. It also survives application restarts. Here are example when you would use this job scheduler:

Tasks that should be done once the device is connect to a power supply
Tasks that require network access or a Wi-Fi connection.
Task that are not critical or user facing
Tasks that should be running on a regular basis as batch where the timing is not critical

---

Ques 55: What is the ViewHolder pattern? Why should we use it? Reference

Every time when the adapter calls `getView()` method, the `findViewById()` method is also called. This is a very intensive work for the mobile CPU and so affects the performance of the application and the battery consumption increases. **ViewHolder** is a design pattern which can be applied as a way around repeated use of `findViewById()`.

A ViewHolder holds the reference to the id of the view resource and calls to the resource will not be required after you "find" them: Thus performance of the application increases.

Ques 56: What is Design Pattern? Reference

A **Design Pattern** is a general, reusable solution to a commonly occurring problem within a given context.

So, it is basically a pattern that can be followed to solve a particular feature. These are the best practices that can be used by any programmer to build an application.

In Android also, we use some Design Pattern that is used to make our code easier to understand and more reusable.

There are a number of Design Patterns that can be used in Software Development and all these can be classified into the following three categories:

Creational Pattern
Structural Pattern
Behavioural Pattern

Ques 57: What is MVVM? Reference

Model — View — ViewModel (MVVM) is the industry-recognized software architecture pattern that overcomes all drawbacks of MVP and MVC design patterns. MVVM suggests separating the data presentation logic(Views or UI) from the core business logic part of the application.

The separate code layers of MVVM are:

Model: This layer is responsible for the abstraction of the data sources. Model and ViewModel work together to get and save the data.
View: The purpose of this layer is to inform the ViewModel about the user's action. This layer observes the ViewModel and does not contain any kind of application logic.
ViewModel: It exposes those data streams which are relevant to the View. Moreover, it serves as a link between the Model and the View.

MVVM pattern has some similarities with the MVP(Model — View — Presenter) design pattern as the Presenter role is played by ViewModel. However, the drawbacks of the MVP pattern has been solved by MVVM in the following ways:

1. ViewModel does not hold any kind of reference to the View.
2. Many to-1 relationships exist between View and ViewModel.
3. No triggering methods to update the View.

Ways to Implement MVVM in the Project

There are 2 ways to implement MVVM design pattern in Android projects:

1. Using the DataBinding library released by Google
2. Using any tool like RxJava for DataBinding.

Ques 58: What happens when we navigate from Activity A to Activity B? Reference

1. Navigate from Activity A to Activity B

When activity A is created, `onCreate()` is called. Then `onStart()` is called. Once the user is able to interact with the app, `onResume()` is called. Now the user opens Activity B.

`onPause()` of activity A will get called. Then activity B's `onCreate()` followed by `onStart()` and `onResume()`. Finally activity A's `onStop()` is triggered.

Here is the callbacks order

Activity A -> `onCreate()` , `onStart()`, `onResume()`, <Activity B started>, `onPause()`

Activity B -> `onCreate()`, `onStart()`, `onResume()`

Activity A -> `onStop()`, `onSaveInstanceState()`

---

2. Navigate from Activity B back to Activity A

When the back button is pressed, Activity B's `onPause()` is triggered. Then activity A's `onRestart()` is called followed by `onStart()` and `onResume()`. After that activity B's `onStop()` gets called followed by `onDestroy()`.

Here is the callbacks order

Activity B -> <Back is pressed>, `onPause()`

Activity A -> `onRestart()`, `onStart()`, `onResume()`

Activity B -> `onStop()`, `onDestroy()`

Ques 59: Differenc between Serializable and Parcelable Reference

To transferring objects between Android components, use to convert it into **Serializable or Parcelable**, but why?

When an application is in the background, its process may be killed by the OS in case of low memory. And by returning back the user to the app, the OS creates a new process for the application. Therefore, with passing an instance of the object to the Bundle, when the process is changed, the reference of the object won't be in the new process and it wouldn't be possible to use that object. Also, in the case of PendingIntents, if the owing application is killed, the PendingIntent itself will remain usable from other processes that have been given it.
So, the OS instead of saving the reference of an object must save its values and it would be possible just by converting that object to the Parcelable or Serializable.

Serializable Objects

Serializing objects means converting an object's state to a byte stream and deserializing means reverting back a byte stream into a copy of the object.

What are Serializable-Objects used for?

Storing objects' data into a file on disk.
In games, the state of the game can be stored into a file on disk by using Serializable.
Sending data over the network.
Transferring objects between different components in Android.

How to create a Serializable object?

Creating a Serializable object is as simple as implementing the **java.io.Serializable** interface in your class (There are no methods to override).

Parcelable

A parcel is highly optimized for local IPC (Inter-Process Communication — You can get familiar with IPC here) also is not attempt to save data in any kind of persistent storage.

How to create a Parcelable object?

Creating a parcelable object is not as simple as creating a Serializable object. First of all the class must implement the Parcelabel interface and fill all its required methods, then, our class must have a non-null static field called CREATOR with the type of Parcelable.Creator.

In comparison with Serializable, making a Parcelable object requires a lot of boilerplate code, especially for java developers. But in Serializable, reflection is used and through the process, many temporary objects will be created. Thus, much memory will be used. On the other side, Parcelable is faster and more optimized than Serialization because it's the developer's responsibility to build the parcel object, so

there is no need to use reflection.
But keep in mind that if you want to save an object in a file or save the state of your game into a file on the storage you still must use Serializable instead of Parcelable.

> Parcel is **not** a general-purpose serialization mechanism. This class (and the corresponding `Parcelable` API for placing arbitrary objects into a Parcel) is designed as a high-performance IPC transport. As such, it is not appropriate to place any Parcel data in to persistent storage: changes in the underlying implementation of any of the data in the Parcel can render older data unreadable.

Summary

In this article, we've learned why and how to use Parcelable or Serializable objects in case of sending data between Android components. Below is a short summary of what we learned:

1. Sending object references between Android components is not possible because by changing the process, the object references won't be in the new process, so we must make our objects Parcelabel or Serializable till the OS be able to save the values of the objects
2. We have two choices to transfer objects between Android components, that are making objects Parcelaber or Serializable
3. Unlike Serializable, **in Parcelable reflection won't be used** so it is **faster** and has **better performance**
4. Android Parcelabe is not made to save objects into the files, so if you want to save an object in the file you must use Serializable

---

Ques 60: Explain the Android Interface Definition Language

Android Interface Definition Language (AIDL) enables communication between a client and service on Android. The data is broken into small portions that are easily recognized by the Android platform.

Ques 61: What is an "Emulator"?

Android's emulator tool helps developers test the user interface on a computer. This way, it becomes easier for developers to write and test different codes for the application. The process of debugging also becomes possible through emulators. Thus, emulators provide a safe platform for testing codes in their early phases and the later stage when the bugs need to be dealt with.

Ques 62: What is an Activitycreator?

An ActivityCreator is an initial step toward creating the Android project. It's a shell script that creates a new file system necessary for writing codes for the Android system.

Ques 63: What is Android Runtime?

Android Runtime (ART) is an application that acts as a runtime environment for Android OS. It replaces Dalvik, another discontinued virtual machine. ART translates the bytecode of applications into native instructions, which are carried out by the device's runtime environment.

Ques 64: Explain the term ANR

The Android OS displays the term ANR for short as a notification that pops up in the lower section of your device's screen every time an app stops responding to the user's action for a considerable amount of time.

Ques 65: What are Android Frameworks?

Android frameworks are a set of APIs that allow developers to write apps more quickly by providing tools such as intents, text fields and more. The frameworks essentially act as a software tool kit that allows for quick construction of the skeleton of an application.

Ques 66: What is an API in Android?

An Application Programming Interface (API) is simply a set of specifications or rules that determine how software components can interact with each other. Companies that release an API do so to let developers build products based on their services.

Ques 67: Can we prevent ANR in the Android system?

Yes, it is possible to prevent ANR in the Android system. ANR (Application Not Responding) occurs when the Android system stops an app that has been responsive for a long time. The prevention can be made by creating a child thread wherein most of the actual workings of the codes can be applied.

Ques 68: What are Default Resources? How are they useful?

Dalvik serves as a virtual machine responsible for running every Android application. Because of Dalvik, a device can run multiple instances of virtual machines smoothly.

Ques 69: How to Reduce APK Size in Android? Reference

Some ways to reduce the apk size:

Remove unused sources
Use of Vector Drawables
Reuse your code
Compress PNG and JPEG files
Use of Lint
Use images in WebP file format
Use of proguard

It makes the application difficult to reverse engineer.

It helps to reduce the size of the application by removing the unused classes and methods.
Use of ShrinkResources
Use of R8 to reduce the size of APK
Limit the usage of external libraries
Use the Android Size Analyzer tool
Generate App Bundles instead of APK
Use of Multiple APK files
Reduce library size
Use of Resconfigs
Remove libraries that are only required for debugging

Ques 70: What is PendingIntents? Reference

A **`PendingIntent`** object wraps the functionality of an `Intent` object while allowing your app to specify something that another app should do, on your app's behalf, in response to a future action. For example, the wrapped intent might be invoked when an alarm goes off, or when the user taps on a notification.

A key aspect of pending intents is that another app invokes the intent on your app's behalf. That is, the other app uses your app's identity when invoking the intent.

In order for the `PendingIntent` to have the same behavior as if it were a normal `Intent`, the system triggers the `PendingIntent` with the same identity as it was created with. In most situations, such as the alarm and notifications, this is the identity of the app itself.

Immutable PendingIntent:

> In this pendingIntent we have an exact action we know we want to perform, we construct a `PendingIntent` that cannot be modified by the app we pass it to by utilizing a flag called `FLAG_IMMUTABLE`

Mutable PendingIntent:

> In this pendingIntent we can modifiy the pendingIntent by the reveiver app or the app we pass it, we use this by flag called `FLAG_MUTABLE`

`FLAG_IMMUTABLE`: Indicates the Intent inside the `PendingIntent` cannot be modified by other apps that pass an `Intent` to `PendingIntent.send()`. An app can always use `FLAG_UPDATE_CURRENT` to modify its own PendingIntents

Prior to Android 12, a `PendingIntent` created without this flag was mutable by default.

`FLAG_MUTABLE`: Indicates the Intent inside the `PendingIntent` should allow its contents to be updated by an app by merging values from the intent parameter of `PendingIntent.send()`.

Ques 71: Whats new in Android 12 for developers? Reference

The main emphasis in the Android 12 release was made on enhancing OS security and data privacy, as well as redesigning the system.

1. Design
   New Material You design
   Standard Splash Screen for all apps
2. Updating UI effects
   Ripple effect
   Overscroll
   RenderEffect API
3. Widgets
   Rounded edges on the background of all widgets on Android 12+
   Widgets can use dynamic colors from the theme, which are new to Material You
   Initial widget setup may be optional and you may skip this step
   CheckBox, Switch and RadioButton support has been added. Various Todo lists and applications with settings via toggle switches
4. Rounded Corner API
5. Rich Content Insertion
6. Notifications
   I n MessagingStyle and BigPictureStyle you can add animated pictures
   You can require mandatory user authorization before sending the Intent
   New call notification style CallStyle, which allows you to create different styles for incoming, outgoing and ongoing calls. You can also set the importance for calls
   The area in the notification allocated for Custom View is now getting smaller
7. Media
   AVIF support
   Compatible Media Transcoding
8. Other
   Vibration feedback generation based on audio stream. It will be very useful for games
   The ability to use a high-resolution camera has been added, meaning you can capture 108 megapixels and other huge sizes.
   When switching audio between applications, the volume will fade instead of an abrupt pause.
   Camera2 API now contains extensions for using special modes similar to CameraX Extentions
9. Security and privacy
   Approximate location
   Expedited Jobs API(Important): Expedited job allows an application to perform short, important tasks while giving the system better control over access to resources. Expedited Job is between the Foreground Service and a regular job from the JobScheduler in terms of its characteristics:
   A short period is a couple of minutes. If your job takes longer, the system may stop the expedited job.
   Expedited job is less susceptible to management effects from battery saver systems such as Battery Saver and Doze Mode.
   The system starts an expedited job if the system has enough resources for this, and if the number of job already started doesn't exceed the limit set in the system, and also if there is enough free RAM. These requirements are more stringent than other job types.
   Restrictions on the use of the Foreground Service (Important)

The main change in Android 12 that will confuse Android developers is that the Foreground Service can no longer be launched from the background, except in a few cases. If you have an unsuccessful attempt to start the Service, the application will crash with a ForegroundServiceStartNotAllowedException.

The following cases are exceptional:

High Priority Push
As a reaction to broadcasts ACTION_BOOT_COMPLETED, ACTION_LOCKED_BOOT_COMPLETED, or ACTION_MY_PACKAGE_REPLACED
Application has special permissions or roles
Application ignores battery optimizations
Exact alarm
and some other specific cases
Applications hibernation

> In Android 11, the automatic reset of permissions for unused applications was implemented successfully. If the user doesn't interact with your application for a very long time (several months), then the system will reset all received permissions and send him to hibernation:
>
> The application cache will be cleared

The application will not be able to run job or alert in background
The application will not receive push notifications, including high-priority FCM
All restrictions will be removed after the first interaction with your application, but you will need to re-create the job, alarm and notifications that you need. Similar behavior is similar when the user forcibly stops applications in the settings
You can ask the user to turn off the “Revoke permissions and clear memory” setting in the settings if your application involves working in the background, for example, sending a location between family members.

Ques 72: Memory Leaks in Android? Reference

*When your code allocates **memory** to an object, but never de-allocates it.*

What is memory Leak in Android?
Whenever an object(A) which is not required anymore is being referenced by some other object(B) which is currently in use, would make garbage collector to consider the object(A) as an useful object .Hence GC will not be able to remove the object(A) thinking somebody is still dependent on this. This eventually leads to Memory leaks.

How Garbage collector works?

Before we proceed further, let us understand briefly how Garbage collector works in Android.

“Garbage collection is the process by which Android programs perform automatic memory management using several GC algorithm e.g Mark and Sweep.

Whenever an android application starts, it creates one memory tree considering starting object as root of the tree. Root creates some other objects which can have its reference directly or indirectly and some other object being instantiated from these objects resulting in a chain of references and thus forms a memory tree.

Garbage collector starts from GC root and traverse to each and every objects reference linked directly or indirectly to the root and mark it as currently in use.All the unmarked object references are considered as unused and are eligible for garbage collection.

Now, just think of the scenario where some unused object which needs to removed from memory is being marked as active by GC only because someone has reference of it. Thus, it will not eligible for garbage collection resulting in memory leaks”.

How Memory leak can ruin your party ?

Whenever memory leaks happens, objects memory is not getting de-allocated even if they are unused, heaps memory keeps on increasing as the app is being used. This leads to the following problems:

1. As memory is required to create new objects to perform some tasks, app will have shortage of available memory .As a result Android system will trigger more frequent GC event than usual to claim the necessary memory. *More the GC has to run, more the application performance deteriorates.*

   GC events are stop-the-world events which means when GC happens, the rendering of UI and processing of events will stop.

   “This impacts depend upon the types of GC events . When minor GCs will not be able to reclaim the memory, and the heap will keep on increasing, system will force a larger GC to kick off, which pauses the entire application main thread for around 50ms to 100ms. Since, Android has a 16ms drawing window and GC takes long than that, Android will start losing frames and lagging seriously .Generally, 100 to 200ms is the threshold beyond which users will perceive slowness in an application.”

2. If previous steps problem does not get solved, heap memory of the app will gradually increase .As a result, app will ask for more memory , however at one point of time system will refuse to allocate further memory which eventually leads to “**OutOfMemoryError**” and **App crashes**.

How to identify a memory leak?

1. **Using Android Profiler:**

   The **Memory Profiler** is a component in the Android Profiler that helps you identify memory

leaks and memory churn that can lead to stutter, freezes, and even app crashes. It shows a realtime graph of your app's memory use and lets you capture a heap dump, force garbage collections, and track memory allocations.

To open the Memory Profiler, follow these steps:

1. *Click **View > Tool Windows > Profiler** (you can also click **Profile** in the toolbar).*
2. *Select the device and app process you want to profile from the Android Profiler toolbar.*
3. *Click anywhere in the **MEMORY** timeline to open the Memory Profiler.*

   Alternatively, you can inspect your app memory from the command line with dumpsys, and also see GC events in logcat.

   When you first open the Memory Profiler, you'll see a detailed timeline of your app's memory use and access tools to force garbage collection, capture a heap dump, and record memory allocations.

2. **Using LeakCanary:**
   LeakCanary is a open source library given by square for memory leak detection in Android.
   It runs along with your application, dumps memory whenever required, identify potential memory leaks and gives a notification with a clean and complete stack trace to find the root cause of the leak.

Common patterns that lead to memory leaks & how to fix them

1. **Unregistered Listeners:**

There are multiple scenarios when we register a listener to Activity or Fragment but forget to unregister it. These listeners hold strong reference to it and prevent it from being garbage collected even when it is unused.

2. Unregistered Local Broadcast receivers:
Sometimes we need to register a local broadcast receiver in an activity.

However, if we don't unregister it, it holds the strong reference to the activity .Even if activity is no longer needed reference prevents it to be garbage collected and hence memory leak occurs.

3. Inner Class Reference(AsyncTask in this case):
Inner class are very commonly used by many Android developers to perform various task .However , non-static Inner class unknowingly holds the implicit reference of the parent class which can lead to memory leaks.

Solution :

Make the inner class as static.
Do not pass any Outer class(Activity/Fragment) reference to inner class. If at all, reference is required, pass it as Weak reference.

4. Anonymous Class Reference:
Similar to non static inner class, anonymous class also holds implicit reference to parent class. Therefore, it is recommended to use static inner class wherever anonymous class is intended to do long running process.

5. Handler Reference :
When a Handler is instantiated on the main thread, it is associated with the Looper's message queue. Messages posted to the message queue will hold a reference to the Handler so that the framework can call Handler#handleMessage(Message) when the Looper eventually processes the message.It is very nicely explained here.

6. Thread & Timer Task :
We can come across the same pattern of memory leak while dealing with Thread and TimerTask as well.Solution is again similar as previously discussed.

Solution 1 : Create static inner class to avoid holding of an implicit reference to their enclosing class.

Solution 2: Close the thread in activity#onDestroy() to avoid thread leak. In case of TimerTask cancel it .

7. Other scenarios:
There are other obvious mistakes which can lead to memory leak and must be avoided by Android developers. These scenarios do not require detail discussion, so just pointing them out here:

· Assigning Activity instance to static reference.

· Passing Activity reference or context to a Singleton class.

· Passing Activity reference as listener to some other class and holding strong reference of it.

· Creating static view reference.

· Improper use of Context. Here is an excellent article to find out exactly which Context to use.

· Inefficient use of Bitmaps without recycling it. This may leads to OutOfMemoryError. Please find the proper way of handling it Here.

Ques 73: What are android app security best practices? Reference

1. **Maintain Secure communication with other apps**

   <manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.myapp">
   *<permission android:name=**"my_custom_permission_name"***
   ***android:protectionLevel="signature"** />*

   Use **implicit intents** to show app chooser that provides option to user to launch at least two possible apps on the device for the requested action. This allows users to transfer sensitive information to the app that they trust.
   Non-exported content providers — Unless you intend to send data from your app to other apps, explicitly disallow other apps to access your ContentProvider in manifest using **android:exported="false"(**by default it is "true" for Android version lower than 4.4).
   Apply signature-based permissions while sharing data between two apps that is controlled by you. These permissions do not need user confirmation, but instead it checks that the apps accessing the data are signed using the same signing key. Hence offer more streamlined and secure user experience.

2. **Secure Network communication**

   Communication with web server having **well-known trusted CA certificate** does need any additional steps to take while creating the http request.

   By adding a network security configuration: If your app uses new or custom CAs, you can declare your network's security settings in a configuration file. This process allows you to create the configuration without modifying any app code.
   Create your own trust manager If web server has a certificate signed by a new or custom CA which is not trusted by the mobile device and additionally you cannot use a network security configuration.
   For this scenario, you may need to set up a trust manager and handle all SSL warnings that occurs .
   Certificates Pinning **:**Application can be limited to accept only a set of certificates by either limiting the set of CAs they trust or by certificate pinning.

3. **Provide the right permissions**
   App should request only the minimum number of permissions necessary to function properly.

   It should not add a permission to complete an action that could be completed in another app. Instead, use an intent to defer the request to a different app that already has the necessary permission.

   For example, If an App requires to create a contact to a contact app,delegates the responsibility of creating the contact to a contacts app, which has already been granted the appropriate WRITE_CONTACTS permission.

4. **Security on data storage**

   Cryptography **is the most efficient way to achieve data security.** Hence, Use appropriate **Encryption** mechanism while dealing with data inside app. To achieve greater key security, use the Android Keystore system. Please find a nice article on Encryption here.
   Following describes the best practices for storing data in your device.
   Storing private data within internal storage
   Use external storage cautiously
   Use scoped directory access: If your app needs to access only a specific directory within the device's external storage, use scoped directory access to limit your app's access to a device's external storage accordingly.

   Access app-specific files: If a file doesn't contain private or sensitive information but provides value to the user only in your app, store the file in an app-specific directory on external storage.

   Store only non-sensitive data in cache files

   Use SharedPreferences in private mode

To create or access your your app's `SharedPreferences` objects using `getSharedPreferences()`, use `MODE_PRIVATE` so that your app can access the information within the shared preferences file.

Moreover, `EncryptedSharedPreferences`should be used for more security which wraps the sharedpreferences class and automatically encrypts keys and values.

5. **Keep dependencies and libraries up-to-date to make these points of communication more secure.**

6. **Shrink, obfuscate, and optimize your code with the *R8 compiler***

   If you are building your project using Android Gradle plugin 3.4.0 or higher, the plugin no longer uses ProGuard to perform compile-time code optimization. Instead, the plugin works with the *R8 compiler* to handle the following compile-time tasks:

   Code shrinking (or tree-shaking): detects and safely removes unused classes, fields, methods, and attributes from your app and its library dependencies (making it a valuable tool for working around the 64k reference limit).

   Resource shrinking: removes unused resources from your packaged app, including unused resources in your app's library dependencies.

   Obfuscation: shortens the name of classes and members, which results in reduced DEX file sizes.

   Optimization: inspects and rewrites your code to further reduce the size of your app's DEX files.

Ques 74: Android design pattern in brife? Reference

Defination:

Design patterns is basically a solution or blueprint for a problem that we get over and over again in programming, so they are just typical types of problems we can encounter as programmers, and these design patterns are just a good way to solve those problems, there is a lot of design pattern in android.

It is basically a pattern that can be followed to solve a particular feature. These are the best practices that can be used by any programmer to build an application.

The following are some of the benefits of using Design Pattern in Android:

Understandable code: By using Design Pattern, you can make your code understandable to everyone i.e. for writing code, you can follow certain guidelines and anyone can understand why you have written that code. ***For example***, if you are using singleton pattern, then anyone viewing your code can understand that here you want to make only one object of a class and you want to access the same object throughout the application.
Code reusability: By following the Design Pattern, you can make your code reusable i.e. for performing a certain task at more than one place, you need not write the same code again and again at various places.
Cleaner code: You can make your code cleaner by decoupling the code and it will make the code easier to understand by using Design Pattern.

There are a number of Design Patterns that can be used in Software Development and all these can be classified into the following three categories:

Creational Pattern
Structural Pattern
Behavioural Pattern

Creational Pattern:

The Creational Pattern is used to create some object without showing the logic or the steps that are involved in creating the object. So, every time you want an object, you need not instantiate the object by using the new operator. So, this makes the creation of object easier and can be easily created again and again. Some of the examples of Creational Pattern are **Builder, Singleton, and Dependency Injection.**

Builder pattern:

> In a builder pattern, you are only concerned about what you need from a class and not everything that a class has. For example, when you go to buy some laptop, then you have a list of important features that you want in your laptop like processors, ram, storage, etc. Apart from these features, all the other features like the speaker, battery, etc are some of the optional features for you. So, based on the important features, the shopkeeper will provide you with a laptop that contains all your important features.
>
> Similarly, when we have a model class and that class is having a number of parameter out of which some are important and some are not important then we can use the Builder pattern. By using the Builder pattern, the user need not call all the methods(as in the case of constructors)

present in the class. The user can call only the required methods and even the order of calling of the method is not fixed i.e you can call any method before and after any method. There is no need for new keyword here(for java).

One of the common examples of Builder pattern that we all use in our daily life is that of AlertDialog. In AleartDialog, we call only the required methods like:

```
AlertDialog.Builder(this)
    .setTitle("This is a title")
    .setMessage("This is some message")
    .show()
```

Singleton pattern

There are cases when you need **only one instance of a class**. So, whenever you call the object of the class, then the new object should not be created(only one-time object creation will be there). This design pattern provides a global point of access to a class. ***For example***, the network connection in an application should be done once and not every time because it is a very expensive process. So in these cases, we can use a singleton.

Dependency Injection pattern

Most of the classes have some dependency that is needed for the proper functioning of the class. In the general case, we hard-code all the dependencies needed in the class inside the class itself. But this is a very wrong way of providing dependency to a class because in future if the class requires some more dependency or you want to add more dependency, then you have to update the new dependency in all the classes that will be using that dependency.

For example, a car needs an engine to run. So, instead of providing the engine inside the same car class, we should provide the engine from outside the car class.

So, in the Dependency Injection pattern, we provide the dependency of a class from outside the class and no dependency will be provided in the same class.

Structural Pattern:

In this Design Pattern, we are concerned about the structure of the code. Here we follow some particular structural pattern that will help in understanding the code and the working of code just by looking at the structure of the code. Some of the common Structural Pattern used are Adapter, Bridge, Facade, Proxy, etc.

Adapter pattern

An Adapter is something like a connector that is used to connect two or more incompatible interface. This pattern lets the classes work together.
In Android, RecyclerView and PagerAdapter are some of the good examples of the Adapter pattern.

Facade pattern

In the Facade pattern, a complicated system is wrapped into a simpler system that will help us in getting the values from the complicated system without having knowledge of how the data is being fetched and returned to the view or the presenter.

Behavioural Pattern:

Behavioural Pattern mainly tells how the objects of the classes will communicate with each other. These patterns help us in understanding the code in a better way because by viewing the code we can identify the pattern and then we can understand the code in a better way. Some of the behavioural patterns are Chain of Responsibility, **Interpreter, Mediator, Observer, Command, Model View Controler, Model View ViewModel, etc.**

Observer pattern

The observer pattern is like one-to-many(or one) dependencies between objects. ***For example***, on YouTube, you can subscribe to a particular channel and if some video is uploaded on that channel then all its subscriber will be notified about the change or about the new video.

So, in Android also, when there is a change in some object, then the dependents of that object will be notified about the change and the possible update will be made accordingly in the dependents. This is called an Observer pattern because here we are observing for some change in the object.

For example, In Android, LiveData is an observable data holder class that enables the use of an observer pattern.

Learn how to use LiveData in Android.

Also, RxJava enables the use of an observer pattern in Android.

Learn RxJava.

Model View Controller pattern

Model View Controller or MVC is an Architectural Design pattern this is used to write an organised code for Android applications. Code following the MVC pattern is divided into three parts:

Model: It is the place where you write all your Business logic and data state. So, all the data classes of your application are considered as Model.
View: It is the user interface that a user sees. In Android, the user interface is built with the help of XML. So, all the views that the user sees come under this category.
Controller: A controller is simply a communication medium between the Model and the View. It takes the user input from the view. It processes the request and it sends the data from the Model to the view.

Nowadays, we don't use this pattern because it is declared as "**Massive View Controller**" for Android. Instead of MVC, we use MVP and MVVM.

Model View Presenter

Model View Presenter or MVP is a set of guidelines that are followed to make the code reusable and testable because it decouples the whole code. MVP divides the application into three parts:
Model: It is the place where you write all your Business logic and data state. So, all the data classes of your application are considered as Model.
View: It is the user interface that a user sees. In Android, the user interface is built with the help of XML. So, all the views that the user sees come under this category.
Presenter: The presenter is responsible for giving the data to the view. If view demands some data from the Model, then it is the presenter that will fetch the data from the Model and provide the data to the view to display the required details.

Learn more about MVP

Model View ViewModel pattern

The Model View ViewModel or MVVC is similar to that of MVC but here the controller is replaced by the ViewModel. The three components of the Model View ViewModel are:
Model: It is the place where you write all your Business logic and data state. So, all the data classes of your application are considered as Model.
View: It is the user interface that a user sees. In Android, the user interface is built with the help of XML. So, all the views that the user sees come under this category.
ViewModel: It is a bridge between the Model and the View. Here most of the user interface logic is stored. This is mainly used to store and manage the UI-related and it resolves one of the biggest problems of data loss that was faced due to screen rotation.

Have a look at our MVVM blog to have a better understanding of the same.
Clean Architecture

Clean Architecture defines the way in which the various layers of an app i.e. presentation, use case, domain, data, and framework layer interact with each other. The Clean Architecture produce a system that is testable, UI-independent, and independent of external agencies and libraries

---

Ques 75: Android Lifecycle Interview Questions Reference

This is not a single question, but a list of questions related to Android Lifecycle which some got wrong, You can click on **Learn More** to get the correct question.

1. Launch Fragment by Default.
2. Create Fragment with Constructor
3. Instantiate ViewModel Directly
4. ViewModel as StateRestoration Solution
5. LiveData as State Restoration Solution
6. When is The View Destroyed But Not the Instance
7. Lifecycle Aware Coroutine
