AndroidPub



# Android Interview Questions Cheat Sheet — Part I

Anitaa Murthy May 30, 2018 · 29 min read


Image Credits: https://www.ipsr.edu.in/blog/most-frequently-asked-interview-questions-in-android/

A set of questions that I have accumulated over the years in preparation for my many Android Interviews Prep.

## 1. What is Application?

*The Application class in Android is the base class within an Android app that contains all other components such as activities and services. The Application class, or any*

*subclass of the Application class, is instantiated before any other class when the process for your application/package is created.*

## 2. What is Context?

*A **Context** is a handle to the system; it provides services like resolving resources, obtaining access to databases and preferences, and so on. An Android app has activities. Context is like a handle to the environment your application is currently running in.*

***Application Context:** This context is tied to the lifecycle of an application. The application context can be used where you need a context whose lifecycle is separate from the current context or when you are passing a context beyond the scope of an activity.*

***Activity Context:** This context is available in an activity. This context is tied to the lifecycle of an activity. The activity context should be used when you are passing the context in the scope of an activity or you need the context whose lifecycle is attached to the current context.*

## 3. What is Armv7?

*There are 3 CPU architectures in Android. **ARMv7** is the most common as it is optimised for battery consumption. **ARM64** is an evolved version of that that supports 64-bit processing for more powerful computing. **ARMx86**, is the least used for these three, since it is not battery friendly. It is more powerful than the other two.*

## 4. Why bytecode cannot be run in Android?

*Android uses **DVM** (Dalvik Virtual Machine ) rather using JVM(Java Virtual Machine).*

## 5. What is a BuildType in Gradle? And what can you use it for?

*Build types define properties that Gradle uses when building and packaging your Android app.*

1. *A build type defines how a module is built, for example whether ProGuard is run.*
2. *A product flavour defines what is built, such as which resources are included in the build.*
3. *Gradle creates a build variant for every possible combination of your project's product flavours and build types.*

## 6. Explain the build process in Android:

1. *First step involves compiling the resources folder (/res) using the aapt (android asset packaging tool) tool. These are compiled to a single class file called R.java. This is a class that just contains constants.*
2. *Second step involves the java source code being compiled to .class files by javac, and then the class files are converted to Dalvik bytecode by the "dx" tool, which is included in the sdk 'tools'. The output is classes.dex.*
3. *The final step involves the android apkbuilder which takes all the input and builds the apk (android packaging key) file.*

## 7. What is the Android Application Architecture?

*Android application architecture has the following components:*

1. ***Services** – It will perform background functionalities*
2. ***Intent** – It will perform the inter connection between activities and the data passing mechanism*
3. ***Resource Externalization** – strings and graphics*
4. ***Notification** – light, sound, icon, notification, dialog box and toast*
5. ***Content Providers** – It will share the data between applications*

## 8. Describe activities

*Activities are basically containers or windows to the user interface.*

## 9. Lifecycle of an Activity

- `OnCreate()` *: This is when the view is first created. This is normally where we create views, get data from bundles etc.*
- `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.*
- `OnResume()` *: Called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it.*

- `OnPause()` : *Called as part of the activity lifecycle when an activity is going into the background, but has not (yet) been killed.*
- `OnStop()` : *Called when you are no longer visible to the user.*
- `OnDestroy()` : *Called when the activity is finishing*
- `OnRestart()` : *Called after your activity has been stopped, prior to it being started again*

### 10. What's the difference between onCreate() and onStart()?

- *The onCreate() method is called once during the Activity lifecycle, either when the application starts, or when the Activity has been destroyed and then recreated, for example during a configuration change.*
- *The onStart() method is called whenever the Activity becomes visible to the user, typically after onCreate() or onRestart().*

### 11. Scenario in which only onDestroy is called for an activity without onPause() and onStop()?

*If finish() is called in the OnCreate method of an activity, the system will invoke onDestroy() method directly.*

### 12. Why would you do the setContentView() in onCreate() of Activity class?

*As onCreate() of an Activity is called only once, this is the point where most initialisation should go. It is inefficient to set the content in onResume() or onStart() (which are called multiple times) as the setContentView() is a heavy operation.*

### 13. onSavedInstanceState() and onRestoreInstanceState() in activity?

`OnRestoreInstanceState()` *- When activity is recreated after it was previously destroyed, we can recover the saved state from the Bundle that the system passes to the activity. Both the* `onCreate()` *and* `onRestoreInstanceState()` *callback methods receive the same Bundle that contains the instance state information. But because the* `onCreate()` *method is called whether the system is creating a new instance of your activity or recreating a previous one, you must check whether the state Bundle is null before you attempt to read it. If it is null, then the system is creating a new instance of the activity, instead of restoring a previous one that was destroyed.*

`onSaveInstanceState()` *- is a method used to store data before pausing the activity.*

## 14. Launch modes in Android?

- ***Standard****: It creates a new instance of an activity in the task from which it was started. Multiple instances of the activity can be created and multiple instances can be added to the same or different tasks.*
  *Eg: Suppose there is an activity stack of A -> B -> C.*
  *Now if we launch B again with the launch mode as* ***"standard"****, the new stack will be A -> B -> C -> B.*

- ***SingleTop****: It is the same as the standard, except if there is a previous instance of the activity that exists in the* ***top*** *of the stack, then it will* ***not*** *create a new instance but rather send the intent to the existing instance of the activity.*
  *Eg: Suppose there is an activity stack of A -> B.*
  *Now if we launch C with the launch mode as* ***"singleTop"****, the new stack will be A -> B -> C as usual.*
  *Now if there is an activity stack of A -> B -> C.*
  *If we launch C again with the launch mode as* ***"singleTop"****, the new stack will still be A -> B -> C.*

- ***SingleTask****: A new task will always be created and a new instance will be pushed to the task as the root one. So if the activity is already in the task, the intent will be redirected to onNewIntent() else a new instance will be created. At a time only one instance of activity will exist.*
  *Eg: Suppose there is an activity stack of A -> B -> C -> D.*
  *Now if we launch D with the launch mode as* ***"singleTask"****, the new stack will be A -> B -> C -> D as usual.*
  *Now if there is an activity stack of A -> B -> C -> D.*
  *If we launch activity B again with the launch mode as* ***"singleTask"****, the new activity stack will be A -> B. Activities C and D will be destroyed.*

- ***SingleInstance****: Same as single task but the system does not launch any activities in the same task as this activity. If new activities are launched, they are done so in a separate task.*
  *Eg: Suppose there is an activity stack of A -> B -> C -> D. If we launch activity B again with the launch mode as* ***"singleInstance"****, the new activity stack will be:*

*Task1 — A -> B -> C*

*Task2 — D*

### 15. How does the activity respond when the user rotates the screen?

*When the screen is rotated, the current instance of activity is destroyed a new instance of the Activity is created in the new orientation. The onRestart() method is invoked first when a screen is rotated. The other lifecycle methods get invoked in the similar flow as they were when the activity was first created.*

### 16. How to prevent the data from reloading and resetting when the screen is rotated?

- *The most basic approach would be to use a combination of **ViewModels** and* `onSaveInstanceState()` *. So how we do we that?*

- *Basics of ViewModel: A ViewModel is **LifeCycle-Aware.** In other words, a ViewModel will not be destroyed if its owner is destroyed for a configuration change (e.g. rotation). The new instance of the owner will just re-connected to the existing ViewModel. So if you rotate an Activity three times, you have just created three different Activity instances, but you only have one ViewModel.*

- *So the common practice is to store data in the ViewModel class (since it persists data during configuration changes) and use OnSaveInstanceState to store small amounts of UI data.*

- *For instance, let's say we have a search screen and the user has entered a query in the Edittext. This results in a list of items being displayed in the RecyclerView. Now if the screen is rotated, the ideal way to prevent resetting of data would be to store the list of search items in the ViewModel and the query text user has entered in the OnSaveInstanceState method of the activity.*

### 17. Mention two ways to clear the back stack of Activities when a new Activity is called using intent

*The first approach is to use a **FLAG_ACTIVITY_CLEAR_TOP** flag. The second way is by using **FLAG_ACTIVITY_CLEAR_TASK** and **FLAG_ACTIVITY_NEW_TASK** in conjunction.*

### 18. What's the difference between FLAG_ACTIVITY_CLEAR_TASK and FLAG_ACTIVITY_CLEAR_TOP?

*FLAG_ACTIVITY_CLEAR_TASK is used to clear all the activities from the task including any existing instances of the class invoked. The Activity launched by intent becomes the new root of the otherwise empty task list. This flag has to be used in conjunction with FLAG_ ACTIVITY_NEW_TASK.*

*FLAG_ACTIVITY_CLEAR_TOP on the other hand, if set and if an old instance of this Activity exists in the task list then barring that all the other activities are removed and that old activity becomes the root of the task list. Else if there's no instance of that activity then a new instance of it is made the root of the task list. Using FLAG_ACTIVITY_NEW_TASK in conjunction is a good practice, though not necessary.*

## 19. Describe content providers

*A ContentProvider provides data from one application to another, when requested. It manages access to a structured set of data. It provides mechanisms for defining data security. ContentProvider is the standard interface that connects data in one process with code running in another process.*

*When you want to access data in a ContentProvider, you must instead use the ContentResolver object in your application's Context to communicate with the provider as a client. The provider object receives data requests from clients, performs the requested action, and returns the results.*

## 20. Access data using Content Provider:

*Start by making sure your Android application has the necessary read access permissions. Then, get access to the ContentResolver object by calling getContentResolver() on the Context object, and retrieving the data by constructing a query using ContentResolver.query().*

*The ContentResolver.query() method returns a Cursor, so you can retrieve data from each column using Cursor methods.*

## 21. Describe services

*A `Service` is an application component that can perform long-running operations in the background, and it doesn't provide a user interface. It can run in the background, even when the user is not interacting with your application. These are the three different types of services:*

- ***Foreground Service:*** *A foreground service performs some operation that is noticeable to the user. For example, we can use a foreground service to play an audio track. A Notification must be displayed to the user.*

- ***Background Service:*** *A background service performs an operation that isn't directly noticed by the user. In Android API level 26 and above, there are restrictions to using background services and it is recommended to use WorkManager in these cases.*

- ***Bound Service:*** *A service is bound when an application component binds to it by calling* `bindService()`*. A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results. A bound service runs only as long as another application component is bound to it.*

## 22. Difference between Service & Intent Service

- *Service is the base class for Android services that can be extended to create any service. A class that directly extends Service runs on the main thread so it will block the UI (if there is one) and should therefore either be used only for short tasks or should make use of other threads for longer tasks.*

- *IntentService is a subclass of Service that handles asynchronous requests (expressed as "Intents") on demand. Clients send requests through startService(Intent) calls. The service is started as needed, handles each Intent in turn using a worker thread, and stops itself when it runs out of work.*

## 23. Difference between AsyncTasks & Threads?

- ***Thread*** *should be used to separate long running operations from main thread so that performance is improved. But it can't be cancelled elegantly and it can't handle configuration changes of Android. You can't update UI from Thread.*

- ***AsyncTask*** *can be used to handle work items shorter than 5ms in duration. With AsyncTask, you can update UI unlike java Thread. But many long running tasks will choke the performance.*

## 24. Difference between Service, Intent Service, AsyncTask & Threads

- ***Android service*** *is a component that is used to perform operations on the background such as playing music. It doesn't has any UI (user interface). The*

*service runs in the background indefinitely even if application is destroyed.*

- ***AsyncTask*** *allows you to perform asynchronous work on your user interface. It performs the blocking operations in a worker thread and then publishes the results on the UI thread, without requiring you to handle threads and/or handlers yourself.*

- ***IntentService*** *is a base class for Services that handle asynchronous requests (expressed as Intents) on demand. Clients send requests through startService(Intent) calls; the service is started as needed, handles each Intent in turn using a worker thread, and stops itself when it runs out of work.*

- *A* ***thread*** *is a single sequential flow of control within a program. Threads can be thought of as mini-processes running within a main process.*

### 25. What are Handlers?

*Handlers are objects for managing threads. It receives messages and writes code on how to handle the message. They run outside of the activity's lifecycle, so they need to be cleaned up properly or else you will have thread leaks.*

- *Handlers allow communicating between the background thread and the main thread.*

- *A Handler class is preferred when we need to perform a background task repeatedly after every x seconds/minutes.*

### 26. What is a Job Scheduling?

*Job Scheduling api, as the name suggests, allows to schedule jobs while letting the system optimize based on memory, power, and connectivity conditions. 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. Some scenarios:*

- *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*
- You can click on this link to learn more about Job Schedulers.

### 27. What is the relationship between the life cycle of an AsyncTask and an Activity? What problems can this result in? How can these problems be avoided?

*An AsyncTask is not tied to the life cycle of the Activity that contains it. So, for example, if you start an AsyncTask inside an Activity and the user rotates the device, the Activity will be destroyed (and a new Activity instance will be created) but the AsyncTask will not die but instead goes on living until it completes.*

*Then, when the AsyncTask does complete, rather than updating the UI of the new Activity, it updates the former instance of the Activity (i.e., the one in which it was created but that is not displayed anymore!). This can lead to an Exception (of the type java.lang.IllegalArgumentException: View not attached to window manager if you use, for instance, findViewById to retrieve a view inside the Activity).*

*There's also the potential for this to result in a memory leak since the AsyncTask maintains a reference to the Activity, which prevents the Activity from being garbage collected as long as the AsyncTask remains alive.*

*For these reasons, using AsyncTasks for long-running background tasks is generally a bad idea . Rather, for long-running background tasks, a different mechanism (such as a service) should be employed.*

***Note:** AsyncTasks by default run on a single thread using a serial executor, meaning it has only 1 thread and each task runs one after the other.*

### 28. What is the onTrimMemory() method?

`onTrimMemory()` *: Called when the operating system has determined that it is a good time for a process to trim unneeded memory from its process. This will happen for example when it goes in the background and there is not enough memory to keep as many background processes running as desired.*

*Android can reclaim memory for from your app in several ways or kill your app entirely if necessary to free up memory for critical tasks. To help balance the system memory and avoid the system's need to kill your app process, you can implement the* `ComponentCallbacks2` *interface in your Activity classes. The provided onTrimMemory() callback method allows your app to listen for memory related events when your app is in either the foreground or the background, and then release objects in response to app lifecycle or system events that indicate the system needs to reclaim memory.*
**Reference**

## 29. Android Bound Service

*A bound service is a service that allows other android components (like activity) to bind to it and send and receive data. A bound service is a service that can be used not only by components running in the same process as local service, but activities and services, running in different processes, can bind to it and send and receive data.*

- *When implementing a bound service we have to extend Service class but we have to override onBind method too. This method returns an object that implements IBinder, that can be used to interact with the service.*

***Implementing Android bound service with Android Messenger***

- *Service based on Messenger can communicate with other components in different processes, known as Inter Process Communication (IPC), without using AIDL.*
- ***A service handler**: this component handles incoming requests from clients that interact with the service itself.*
- ***A Messenger**: this class is used to create an object implementing IBinder interface so that a client can interact with the service.*
- *Example Implementation: **Link***

## 30. AIDL vs Messenger Queue

- *As Ariq Ahmad mentioned in the response, Messenger Queue builds us a queue and the data/messages are passed between 2 or more processes sequential. But in case of AIDL the messages are passed in parallel.*

- *AIDL is for the purpose when you've to go application level communication for data and control sharing, a scenario depicting it can be : An app requires list of all contacts from Contacts app (content part lies here) plus it also wants to show the call's duration and you can also disconnect it from that app (control part lies here).*

- *In Messenger queues you're more IN the application and working on threads and processes to manage the queue having messages so no Outside services interference here.*

- *Messenger is needed if you want to bind a remote service (e.g. running in another process).*

### 31. What is a ThreadPool? And is it more effective than using several separate Threads?

*Creating and destroying threads has a high CPU usage, so when we need to perform lots of small, simple tasks concurrently, the overhead of creating our own threads can take up a significant portion of the CPU cycles and severely affect the final response time. ThreadPool consists of a task queue and a group of worker threads, which allows it to run multiple parallel instances of a task.*

### 32. Difference between Serializable and Parcelable?

*Serialization is the process of converting an object into a stream of bytes in order to store an object into memory, so that it can be recreated at a later time, while still keeping the object's original state and data.*

***How to disallow serialization?*** *We can declare the variable as transient.*

*Serializable is a standard Java interface. Parcelable is an Android specific interface where you implement the serialization yourself. It was created to be far more efficient than Serializable (The problem with this approach is that reflection is used and it is a slow process. This mechanism also tends to create a lot of temporary objects and cause quite a bit of garbage collection.).*

### 33. Difference between Activity & Service

*Activities are basically containers or windows to the user interface. Services is a component that is used to perform operations on the background. It does not have an UI.*

### 34. How would you update the UI of an activity from a background service?

*We need to register a LocalBroadcastReceiver in the activity. And send a broadcast with the data using intents from the background service. As long as the activity is in the foreground, the UI will be updated from the background. Ensure to unregister the broadcast receiver in the onStop() method of the activity to avoid memory leaks. We can also register a Handler and pass data using Handlers. You can find more details on how to implement here.*

### 35. What is an intent?

*Intents are messages that can be used to pass information to the various components of android. For instance, launch an activity, open a webview etc. Two types of intents-*

- ***Implicit:** Implicit intent is when you call system default intent like send email, send SMS, dial number.*
- ***Explicit:** Explicit intent is when you call an application activity from another activity of the same application.*

### 36. What is a Sticky Intent?

*Sticky Intents allows communication between a function and a service.* `sendStickyBroadcast()` *performs a sendBroadcast(Intent) known as sticky, i.e. 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)`*. For example, if you take an intent for ACTION_BATTERY_CHANGED to get battery change events: 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.*

### 37. What is a Pending Intent?

*If you want someone to perform any Intent operation at future point of time on behalf of you, then we will use Pending Intent.*

### 38. What is an Action?

*Description of the intent. For instance, ACTION_CALL — used to perform calls*

### 39. What are intent Filters?

*Specifies the type of intent that the activity/service can respond to.*

### 40. Describe fragments:

*Fragment is a UI entity attached to Activity. Fragments can be reused by attaching in different activities. Activity can have multiple fragments attached to it. Fragment must be attached to an activity and its lifecycle will depend on its host activity.*

### 41. Describe fragment lifecycle

- `onAttach()` *: The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work.*
- `onCreate()` *: The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.*
- `onCreateView()` *: The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.*
- `onActivityCreated()` *: The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object*
- `onStart()` *: The onStart() method is called once the fragment gets visible.*
- `onResume()` *: Fragment becomes active.*
- `onPause()` *: The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.*
- `onStop()` *: Fragment going to be stopped by calling onStop()*
- `onDestroyView()` *: Fragment view will destroy after call this method*

- `onDestroy()` *:called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.*

### 42. What is the difference between fragments & activities. Explain the relationship between the two.

*An Activity is an application component that provides a screen, with which users can interact in order to do something whereas a Fragment represents a behavior or a portion of user interface in an Activity (with its own lifecycle and input events, and which can be added or removed at will).*

### 43. When should you use a fragment rather than an activity?

- *When there are ui components that are going to be used across multiple activities.*
- *When there are multiple views that can be displayed side by side (viewPager tabs)*
- *When you have data that needs to be persisted across Activity restarts (such as retained fragments)*

### 44. Difference between adding/replacing fragment in backstack?

- ***replace*** *removes the existing fragment and adds a new fragment. This means when you press back button the fragment that got replaced will be created with its onCreateView being invoked.*
- ***add*** *retains the existing fragments and adds a new fragment that means existing fragment will be active and they wont be in 'paused' state hence when a back button is pressed onCreateView is not called for the existing fragment(the fragment which was there before new fragment was added).*
- *In terms of fragment's life cycle events onPause, onResume, onCreateView and other life cycle events will be invoked in case of replace but they wont be invoked in case of add.*

### 45. Why is it recommended to use only the default constructor to create a Fragment?

*The reason why you should be passing parameters through bundle is because when the system restores a fragment (e.g on config change), it will automatically restore your bundle. This way you are guaranteed to restore the state of the fragment correctly to the same state the fragment was initialised with.*

### 46. You're replacing one Fragment with another — how do you ensure that the user can return to the previous Fragment, by pressing the Back button?

*We need to save each Fragment transaction to the backstack, by calling* `addToBackStack()` *before you* `commit()` *that transaction*

### 47. Callbacks invoked during addition of a fragment to back stack and while popping back from back stack:

`addOnBackStackChangedListener` *is called when fragment is added or removed from the backstack. Checkout this link for reference.*

### 48. What are retained fragments?

*By default, Fragments are destroyed and recreated along with their parent Activity's when a configuration change occurs. Calling* `setRetainInstance(true)` *allows us to bypass this destroy-and-recreate cycle, signaling the system to retain the current instance of the fragment when the activity is recreated.*

### 49. Difference between FragmentPagerAdapter vs FragmentStatePagerAdapter?

- ***FragmentPagerAdapter:*** *the fragment of each page the user visits will be stored in memory, although the view will be destroyed. So when the page is visible again, the view will be recreated but the fragment instance is not recreated. This can result in a significant amount of memory being used. FragmentPagerAdapter should be used when we need to store the whole fragment in memory. FragmentPagerAdapter calls* `detach(Fragment)` *on the transaction instead of* `remove(Fragment)`*.*

- ***FragmentStatePagerAdapter****: the fragment instance is destroyed when it is not visible to the User, except the saved state of the fragment. This results in using only a small amount of Memory and can be useful for handling larger data sets. Should be used when we have to use dynamic fragments, like fragments with widgets, as their data could be stored in the savedInstanceState.Also it won't affect the performance even if there are large number of fragments.*

### 50. What is Toast in Android?

*Android 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.*

## 51. What are Loaders in Android?

*Loader API was introduced in API level 11 and is used to load data from a data source to display in an activity or fragment. Loaders persist and cache results across configuration changes to prevent duplicate queries.*

*Checkout the Sample Implementation.*

## 52. What is the difference between Dialog & DialogFragment?

*A fragment that displays a dialog window, floating on top of its activity's window. This fragment contains a Dialog object, which it displays as appropriate based on the fragment's state. Dialogs are entirely dependent on Activities. If the screen is rotated, the dialog is dismissed. Dialog fragments take care of orientation, configuration changes as well.*

## 53. Difference between margin & padding?

*Padding will be space added inside the container, for instance, if it is a button, padding will be added inside the button. Margin will be space added outside the container.*

## 54. What is View Group? How are they different from Views?

***View:*** *View objects are the basic building blocks of User Interface(UI) elements in Android. View is a simple rectangle box which responds to the user's actions. Examples are EditText, Button, CheckBox etc. View refers to the* `android.view.View` *class, which is the base class of all UI classes.*

***ViewGroup:*** *ViewGroup is the invisible container. It holds View and ViewGroup. For example, LinearLayout is the ViewGroup that contains Button(View), and other Layouts also. ViewGroup is the base class for Layouts.*

## 55. What is the difference between a regular .png and a nine-patch image?

*It is one of a resizable bitmap resource which is being used as backgrounds or other images on the device. The NinePatch class allows drawing a bitmap in nine sections. The four corners are unscaled; the middle of the image is scaled in both axes, the four edges are scaled into one axis.*

## 56. Difference between RelativeLayout and LinearLayout?

***Linear Layout*** *— Arranges elements either vertically or horizontally. i.e. in a row or column.*

***Relative Layout*** *— Arranges elements relative to parent or other elements.*

### 57. What is ConstraintLayout?

*It allows you to create large and complex layouts with a flat view hierarchy (no nested view groups). It's similar to RelativeLayout in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.*

Checkout the **Sample Implementation** and you can read more about how to implement a simple app with ConstraintLayout here, by yours truly :)

### 58. When might you use a FrameLayout?

*Frame Layouts are designed to contain a single item, making them an efficient choice when you need to display a single View.*

*If you add multiple Views to a FrameLayout then it'll stack them one above the other, so FrameLayouts are also useful if you need overlapping Views, for example if you're implementing an overlay or a HUD element.*

### 59. What is Adapters?

*An adapter responsible for converting each data entry into a View that can then be added to the AdapterView (ListView/RecyclerView).*

### 60. How to support different screen sizes?

- ***Create a flexible layout*** *— The best way to create a responsive layout for different screen sizes is to use ConstraintLayout as the base layout in your UI. ConstraintLayout allows you to specify the position and size for each view according to spatial relationships with other views in the layout. This way, all the views can move and stretch together as the screen size changes.*
- ***Create stretchable nine-patch bitmaps***
- ***Avoid hard-coded layout sizes*** *— Use wrap_content or match_parent.* ***Create alternative layouts*** *— The app should provide alternative layouts to optimise the UI design for certain screen sizes. For eg: different UI for tablets*
- ***Use the smallest width qualifier*** *— For example, you can create a layout named main_activity that's optimised for handsets and tablets by creating different*

*versions of the file in directories as follows:*
***res/layout/main_activity.xml** — For handsets (smaller than 600dp available width)*
***res/layout-sw600dp/main_activity.xml** — For 7" tablets (600dp wide and bigger).*

- *The smallest width qualifier specifies the smallest of the screen's two sides, regardless of the device's current orientation, so it's a simple way to specify the overall screen size available for your layout.*

### 61. Outline the process of creating custom Views:

- *Create a class that Subclass a view*
- *Create a res/values/attrs.xml file and declare the attributes you want to use with your custom View.*
- *In your View class, add a constructor method, instantiate the Paint object, and retrieve your custom attributes.*
- *Override either onSizeChanged() or onMeasure().*
- *Draw your View by overriding onDraw().*
- *Checkout the **Sample Implementation***

### 62. Briefly describe some ways that you can optimize View usage

- ***Checking for excessive overdraw:** install your app on an Android device, and then enable the "Debug GPU Overview" option.*
- ***Flattening your view hierarchy:** inspect your view hierarchy using Android Studio's 'Hierarchy Viewer' tool.*
- ***Measuring how long it takes each View to complete the measure**, layout, and draw phases. You can also use Hierarchy Viewer to identify any parts of the rendering pipeline that you need to optimise.*

### 63. Bitmap pooling in android?

*Bitmap pooling is a simple technique, that aims to reuse bitmaps instead of creating new ones every time. When you need a bitmap, you check a bitmap stack to see if there are any bitmaps available. If there are not bitmaps available you create a new*

*bitmap otherwise you pop a bitmap from the stack and reuse it. Then when you are done with the bitmap, you can put it on a stack.*

***Find more info here***

### 64. How to load bitmap to memory?

- Checkout this article on it. I couldn't have explained it better myself.

### 65. 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.*

### 66. What is an Application Not Responding (ANR) error, and how can you prevent them from occurring in an app?

*An ANR dialog appears when your UI has been unresponsive for more than 5 seconds, usually because you've blocked the main thread. To avoid encountering ANR errors, you should move as much work off the main thread as possible.*

### 67. What is a singleton class in Android?

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

I took references from this article to implement the below code.

## 68. What's the difference between commit() and apply() in SharedPreferences?

`commit()` *writes the data synchronously and returns a boolean value of success or failure depending on the result immediately.*

`apply()` *is asynchronous and it won't return any boolean response. Also if there is an apply() outstanding and we perform another commit(). The commit() will be blocked until the apply() is not completed.*

## 69. How does RecyclerView work?

- *RecyclerView is designed to display long lists (or grids) of items. Say we want to display 100 row of items. A simple approach would be to just create 100 views, one for each row and lay all of them out. But that would be wasteful because at any point of time, only 10 or so items could fit on screen and the remaining items would be off screen. So RecyclerView instead creates only the 10 or so views that are on screen. This way you get 10x better speed and memory usage.*

- ***But what happens when you start scrolling and need to start showing next views?*** *Again a simple approach would be to create a new view for each new row that you need to show. But this way by the time you reach the end of the list you will have created 100 views and your memory usage would be the same as in the first approach. And creating views takes time, so your scrolling most probably wouldn't be smooth.* ***This is why RecyclerView takes advantage of the fact that as you scroll, new rows come on screen also old rows disappear off screen. Instead of creating new view for each new row, an old view is recycled and reused by binding new data to it.***

- *This happens inside the* `onBindViewHolder()` *method. Initially you will get new unused view holders and you have to fill them with data you want to display. But as you scroll you will start getting view holders that were used for rows that went off screen and you have to replace old data that they held with new data.*

## 70. How does RecyclerView differ from ListView?

- ***ViewHolder Pattern:*** *Recyclerview implements the ViewHolders pattern whereas it is not mandatory in a ListView. A RecyclerView recycles and reuses cells when*

*scrolling.*

- ***What is a ViewHolder Pattern?*** *— A ViewHolder object stores each of the component views inside the tag field of the Layout, so you can immediately access them without the need to look them up repeatedly. In ListView, the code might call* `findViewById()` *frequently during the scrolling of ListView, which can slow down performance. Even when the Adapter returns an inflated view for recycling, you still need to look up the elements and update them. A way around repeated use of* `findViewById()` *is to use the "view holder" design pattern.*

- ***LayoutManager:*** *In a ListView, the only type of view available is the vertical ListView. A RecyclerView decouples list from its container so we can put list items easily at run time in the different containers (linearLayout, gridLayout) by setting LayoutManager.*

- ***Item Animator:*** *ListViews are lacking in support of good animations, but the RecyclerView brings a whole new dimension to it.*

## 71. How would you implement swipe animation in Android

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:shareInterpolator=”false”>
<translate android:fromXDelta=”-100%”
           android:toXDelta=”0%”
           android:fromYDelta=”0%”
           android:toYDelta=”0%”
           android:duration=”700"/>
</set>
```

## 72. Arraymap/SparseArray vs HashMap in Android?

Article 1 on the subject

Article 2 on the subject

## 73. How to plug memory Leak in Android?

*Checkout this awesome article by yours truly :)*

## 74. How to reduce apk size in Android?

*Checkout this awesome article.*
***The summary the article states:***

- *Enable proguard in your project by adding following lines to your release build type.*
- *Enable shrinkResources .*
- *Strip down all the unused locale resources by adding required resources name in "resConfigs".*
- *Convert all the images to the webp or vector drawables.*

## 75. How to reduce build time of an android application?

*Checkout this awesome article.*
***What I got from the article was:*** *A few commands we can add to the gradle.properties file:*

- `org.gradle.configureondemand=true` *- This command will tell gradle to only build the projects that it really needs to build.*
- *Use Daemon —* `org.gradle.daemon=true` *- Daemon keeps the instance of the gradle up and running in the background even after your build finishes. This will remove the time required to initialize the gradle and decrease your build timing significantly.*
- `org.gradle.parallel=true` *- Allow gradle to build your project in parallel. If you have multiple modules in you project, then by enabling this, gradle can run build operations for independent modules parallelly.*
- *Increase Heap Size —* `org.gradle.jvmargs=-Xmx3072m -XX:MaxPermSize=512m -XX:+HeapDumpOnOutOfMemoryError -Dfile.encoding=UTF-8` *- Since android studio 2.0, gradle uses dex in the process to decrease the build timings for the project. Generally, while building the applications, multiple dx processes runs on different VM instances. But starting from the Android Studio 2.0, all these dx processes runs in the single VM and that VM is also shared with the gradle. This decreases the build time significantly as all the dex process runs on the same VM instances. But this requires larger memory to accommodate all the dex processes and*

*gradle. That means you need to increase the heap size required by the gradle daemon. By default, the heap size for the daemon is about 1GB.*

- *Ensure that dynamic dependency is not used. i.e. do not use* `implementation 'com.android.support:appcompat-v7:27.0.+'`.
  *This command means gradle will go online and check for the latest version every time it builds the app.*
  *Instead use fixed versions i.e.* `'com.android.support:appcompat-v7:27.0.2'`

*I followed the steps in there and reduced by build time from **167** seconds to **65** seconds ~ **38%.***

## 76. Android Architecture Components?

*A collection of libraries that help you design robust, testable, and maintainable apps.*

*Checkout the Official documentation.*

- ***Room***
  *Official documentation*
  *Article on how to implement Room Db*
  *Sample implementation*

- ***Live Data***
  *Official documentation*
  *Sample implementation*

- ***ViewModel***
  *Official documentation*
  *Sample implementation*

- ***Data Binding***
  *Official documentation*
  *Sample implementation*

- ***Lifecycles***
  *Official documentation*

## 77. Difference between MVC & MVP & MVVM?

*MVC is the **Model-View-Controller** architecture where model refers to the data model classes. The view refers to the xml files and the controller handles the business logic. The issue with this architecture is unit testing. The model can be easily tested since it is not tied to anything. The controller is tightly coupled with the android apis making it difficult to unit test. Modularity & flexibility is a problem since the view and the controller are tightly coupled. If we change the view, the controller logic should also be changed. Maintenance is also an issues.*

***MVP architecture: Model-View-Presenter architecture**. The View includes the xml and the activity/fragment classes. So the activity would ideally implement a view interface making it easier for unit testing (since this will work without a view).*
***Sample Implementation***

***MVVM: Model-View-ViewModel Architecture** . The Model comprises data, tools for data processing, business logic. The View Model is responsible for wrapping the model data and preparing the data for the view. IT also provides a hook to pass events from the view to the model.*
***Sample Implementation***

## 78. S.O.L.I.D principles in software development?

- *The Single Responsibility Principle (SRP)*
- *The Open-Closed Principle (OCP)*
- *The Liskov Substitution Principle (LSP)*
- *The Interface Segregation Principle (ISP)*
- *The Dependency Inversion Principle (DIP)*

*You can checkout the whole article on what this means from here.*

## 79. RxJava in Android?

- *RxJava Introduction and Operators for creating Observables*
- *RxJava Operators: Operators for Transforming Observables*
- *RxJava Operators: Operators for Filtering Observables*

- *RxJava Operators: Operators for Combining Observables*
- *RxJava Operators: Utility Operators*
- *RxJava Operators: Conditional and Boolean Operators*
- *RxJava Operators: Mathematical and Aggregate Operators*
- *RxJava: Different types of Observables*

## Thanks for reading!

Hope you find this useful. This is just a list of questions I personally found useful in interviews. This list is by no means exhaustive. Let me know your thoughts in the responses and don't forget to clap if you found the article helpful.

You can checkout the entire set of interview questions along with sample coding programs in Github.

As promised, here is an article on the possible list of Java questions you might be asked in an Android interview.

Happy Coding!

Android Interview Questions Java Interview Tips