# Android Interview Topic Review

**Copy this document to create a personal version you can add your answers to.**

Please review the questions below in each section, research the answers on our [CodePath guides](http://guides.codepath.com) and any other sources (StackOverflow, [official docs](https://developer.android.com/develop/index.html)) and then write up a complete answer (between a few sentences and a few paragraphs) for each of the questions below.

## 1. Core Java

Research and answer the following core Java questions:

1. What is the difference between abstract classes and interfaces?

* Abstract class is used to define template with some default functionality being implemented.
  + A class for which no objects can be created is an abstract class since few functions will not have implementations.
  + The way to extend Abstract class is by using “Extends” keyword.A concrete child class of an Abstract Class must define all the abstract methods. An Abstract child class can have abstract methods.
  + Abstract classes can have non-final, non-static and any access modifier for variables
* Interface is a keyword and it is used to define the template or blue print of an object.Interface are those where no functions are implemented.
  + We can create object for an interface. The Interface reference can hold the objects of the class which are implementing this interface.
  + The way to extend Interface using by using “Implements” keyword.There are only the signatures of the methods, which implies that the methods do not have a body. The interface can't do anything. It's just a pattern.
  + An *interface* extending another interface need not provide default implementation for methods inherited from the parent interface.
  + The variables declared in Java interface are by default final,public and static.

1. Describe the value of an interface and why are they important?

Interface is a blueprint of a class. It has abstract methods, i.e, methods do not have implementation and has only signature. Interfaces are used to achieve abstraction and multiple Inheritance, Loose coupling in java.

1. What are anonymous classes?

Anonymous class is an inner class without a name and for which only a single object is created.

An anonymous inner class can be useful when making an instance of an object with certain “extras” such as overloading methods of a class or interface, without having to actually subclass a class. Its useful while implementation of listener interfaces.

* Anonymous inner class are mainly created in two ways:
  + Class (may be abstract or concrete)
  + Interface
* A normal class can implement any number of interfaces but anonymous inner class can implement only one interface at a time.
* A regular class can extend a class and implement any number of interface simultaneously. But anonymous Inner class can extend a class or can implement an interface but not both at a time.
* For regular/normal class, we can write any number of constructors but we cant write any constructor for anonymous Inner class because anonymous class does not have any name and while defining constructor class name and constructor name must be same.

1. What is the difference between using == and .equals on a string?

* == - Actually compares the objects based on memory reference instead of the actual values of the string. so "==" operator will return true only if two object reference it is comparing represent exactly same object otherwise "==" will return false.
* .equals - compares the values of the string objects.

1. What is composition and what is inheritance in object-oriented programming? Why would you use one over another?
   * Functional ways of establishing relationships with classes is the goal of object oriented programming.
   * This can be done in 2 ways
     + Inheritance
     + Composition
   * Inheritance is an *"is-a"* relationship. Composition is a *"has-a"*.
   * We do composition by having an instance of another class C as a field of our class, instead of extending C. We can use the functionality of the class C using C’s object declared in our class.
   * Inheritance means reusing the complete functionality of a class. Child class should use all the methods of the super class and child class will be tightly coupled with the super class and code will be duplicated in both the classes in case of inheritance
2. What is polymorphism and give an example where this is useful?
   * The word ‘polymorphism’ literally means ‘a state of having many shapes’ or ‘the capacity to take on different forms’. When applied to object oriented programming languages like Java, it describes a language’s ability to process objects of various types and classes through a single, uniform interface.
   * Polymorphism in Java has two types: Compile time polymorphism (static binding) and Runtime polymorphism (dynamic binding). Method overloading is an example of static polymorphism, while method overriding is an example of dynamic polymorphism.
   * Example Static polymorphism
     + class DemoOverload{  
         
        public int add(int x, int y){ //method 1  
         
        return x+y;  
         
        }  
         
        public int add(int x, int y, int z){ //method 2  
         
        return x+y+z;  
         
        }  
         
        public int add(double x, int y){ //method 3  
         
        return (int)x+y;  
         
        }  
         
        public int add(int x, double y){ //method 4  
         
        return x+(int)y;  
         
        }  
         
       }  
         
       class Test{  
         
        public static void main(String[] args){  
         
        DemoOverload demo=new DemoOverload();  
         
        System.out.println(demo.add(2,3)); //method 1 called  
         
        System.out.println(demo.add(2,3,4)); //method 2 called  
         
        System.out.println(demo.add(2,3.4)); //method 4 called  
         
        System.out.println(demo.add(2.5,3)); //method 3 called  
         
        }  
         
       }
   * Dynamic Polymorphism

class Vehicle{  
  
 public void move(){  
  
 System.out.println(“Vehicles can move!!”);  
  
 }  
  
}  
  
class MotorBike extends Vehicle{  
  
 public void move(){  
  
 System.out.println(“MotorBike can move and accelerate too!!”);  
  
 }  
  
}  
  
class Test{  
  
 public static void main(String[] args){  
  
 Vehicle vh=new MotorBike();  
  
 vh.move(); // prints MotorBike can move and accelerate too!!  
  
 vh=new Vehicle();  
  
 vh.move(); // prints Vehicles can move!!  
  
 }  
  
}

1. When would you make an object value final? When you want to make an variable static?
   * An object is made final when we don’t the value of this object to be modified.
   * Static variable is also called as a class variable. When a member is made static, only a single instance of this member will be created for all the objects of the class and it will shared among all the objects of the class.
2. What is the difference between between method overloading and overriding?
   * Method Overloading is Static polymorphism. Here the method name is same whereas the number of arguments, order, type of the arguments can be different.
   * Method Overriding is dynamic polymorphism or late binding. Here the method name, signature should be same completely and functionality will be overridden from the parent class.
3. What is serialization in Java (not Android)? How do you implement it? How about in Android?
   * Serialization is a mechanism of converting the state of an object into a byte stream so that it can be recreated at a later time, while still keeping the object's original state and data
   * To make a Java object serializable we implement the **java.io.Serializable** interface.
   * The ObjectOutputStream class contains **writeObject()** method for serializing an Object.
   * Only non-static data members are saved via Serialization process. Static data members and transient data members are not saved via Serialization process.
   * Incase of Android also Serialization works. But there is specific implementation in Android called Parcelable. In case of serialization there is too many temporary object that will be created since Serializable uses . Parcelable is 10X faster.
4. Do objects get passed by reference or value in Java? Elaborate on that.
   * Java uses pass by value
     + When passing primitive data, it copies the value of primitive data type.
     + When passing object, it copies the address of object and passes to callee method variable.
5. What are generics and why are they important in Java?
   * Java Generic methods and generic classes enable programmers to specify, with a single method declaration, a set of related methods, or with a single class declaration, a set of related types, respectively.
   * Generics also provide compile-time type safety that allows programmers to catch invalid types at compile time.

Following are the rules to define Generic Methods −

* All generic method declarations have a type parameter section delimited by angle brackets (< and >) that precedes the method's return type ( < E > in the next example).
* Each type parameter section contains one or more type parameters separated by commas. A type parameter, also known as a type variable, is an identifier that specifies a generic type name.
* The type parameters can be used to declare the return type and act as placeholders for the types of the arguments passed to the generic method, which are known as actual type arguments.
* A generic method's body is declared like that of any other method. Note that type parameters can represent only reference types, not primitive types (like int, double and char)
* Generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar *formal parameters* used in method declarations, type parameters provide a way for you to re-use the same code with different inputs. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

Generics code has following benefits

* Stronger type checks at compile time.. A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.
* Elimination of casts.by eliminating casting.
* Enabling programmers to implement generic algorithms.Programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

1. What is one design pattern that you use and find helpful in Java/Android?
   * SingleTon
     + Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.
     + Singleton pattern is used for logging, drivers objects, caching and thread pool.
     + Single pattern can be implemented by using below rules
       1. Private constructor to restrict instantiation of the class from other classes.
       2. Private static variable of the same class that is the only instance of the class.
       3. Public static method that returns the instance of the class, this is the global access point for outer world to get the instance of the singleton class.

In case of android singleton is useful when want the same database object which will be used for read and write operations.

## 2. Core Android

Research and answer the following core Android questions:

* Explain Activity lifecycle including most notable events and the order they run.
* Activities are a fundamental building block of Android applications and they can exist in a number of different states. When an activity changes state, the appropriate lifecycle event method is called, notifying the activity of the impending state change and allowing it to execute code to adapt to that change.
* *Active or Running* – Activities are considered active or running if they are in the foreground, also known as the top of the activity stack.
* *Paused* – When the device goes to sleep, or an activity is still visible but partially hidden by a new, non-full-sized or transparent activity, the activity is considered paused. Paused activities are still alive, that is, they maintain all state and member information, and remain attached to the window manager.
* *Stopped/Backgrounded* – Activities that are completely obscured by another activity are considered stopped or in the background.
* *Restarted* – It is possible for an activity that is anywhere from paused to stopped in the lifecycle to be removed from memory by Android. If the user navigates back to the activity it must be restarted, restored to its previously saved state, and then displayed to the user.
* the Activity class provides a core set of six callbacks: onCreate(), onStart(),onResume(), onPause(), onStop(), and onDestroy().

### **onCreate**() :

### On activity creation, the activity enters the *Created* state. In the onCreate() method, you perform basic application startup logic that should happen only once for the entire life of the activity.After the onCreate() method finishes execution, the activity enters the *Started* state, and the system calls the onStart() and onResume() methods in quick succession.

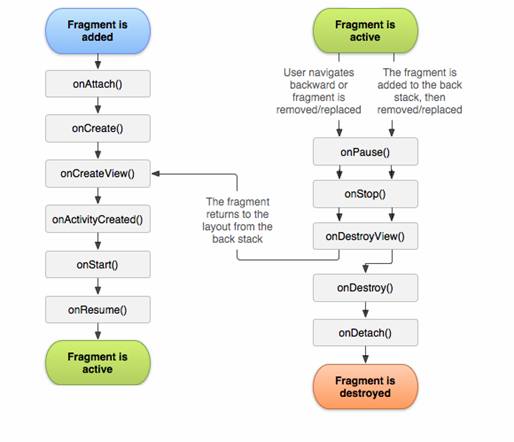
* **OnStart**():
  + The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive. The onStart() method completes very quickly and, as with the Created state, the activity does not stay resident in the Started state. Once this callback finishes, the activity enters the *Resumed* state, and the system invokes the onResume() method.
* **OnResume**() :
  + When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the onResume() callback. This is the state in which the app interacts with the user. The app stays in this state until something happens to take focus away from the app.
* **OnPause**() :
  + This method is invoked when user is leaving the activity. This method has to be used to stop animations, music playback , to release system resources such as broadcast receivers, sensors etc.

### **onStop**()

* + When the activity is no longer visible to the user, it has entered the *Stopped* state, and the system invokes the onStop() callback. This may occur, for example, when a newly launched activity covers the entire screen. The system may also call onStop() when the activity has finished running, and is about to be terminated. We need to use onStop() to perform relatively CPU-intensive shutdown operations

### **onDestroy**()

* + Called before the activity is destroyed. This is the final call that the activity receives. The system either invokes this callback because the activity is finishing due to someone's calling finish(), or because the system is temporarily destroying the process containing the activity to save space.
* Explain Fragment lifecycle including most notable events and the order they run.
  + A Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
  + A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.



* + onAttach(Activity) called once the fragment is associated with its activity.
  + onCreate(Bundle) called to do initial creation of the fragment.
  + onCreateView(LayoutInflater, ViewGroup, Bundle) creates and returns the view hierarchy associated with the fragment.
  + onActivityCreated(Bundle) tells the fragment that its activity has completed its own Activity.onCreate().
  + onStart() makes the fragment visible to the user (based on its containing activity being started).
  + onResume() makes the fragment begin interacting with the user (based on its containing activity being resumed).

As a fragment is no longer being used, it goes through a reverse series of callbacks:

* + onPause() fragment is no longer interacting with the user either because its activity is being paused or a fragment operation is modifying it in the activity.
  + onStop() fragment is no longer visible to the user either because its activity is being stopped or a fragment operation is modifying it in the activity.
  + onDestroyView() allows the fragment to clean up resources associated with its View.
  + onDestroy() called to do final cleanup of the fragment's state.
  + onDetach() called immediately prior to the fragment no longer being associated with its activity.
* How would you preserve Activity state during a screen rotation?
  + When activity is destroyed because the user presses Back or the activity finishes itself, the system's concept of that Activity instance is gone forever because the behavior indicates the activity is no longer needed. However, if the system destroys the activity due to system constraints (rather than normal app behavior), then although the actual Activity instance is gone, the system remembers that it existed such that if the user navigates back to it, the system creates a new instance of the activity using a set of saved data that describes the state of the activity when it was destroyed. The saved data that the system uses to restore the previous state is called the *instance state* and is a collection of key-value pairs stored in a Bundle object.
  + By default, the system uses the Bundle instance state to save information about each View object in your activity layout (such as the text value entered into an EditText widget). So, if your activity instance is destroyed and recreated, the state of the layout is restored to its previous state with no code required by us.
  + we may use the onSaveInstanceState() method. As the activity begins to stop, the system calls the onSaveInstanceState() method so the activity can save state information with a collection of key-value pairs. The default implementation of this method saves transient information about the state of the activity's view hierarchy, such as the text in an EditText widget or the scroll position of a ListView widget. Your app should implement the onSaveInstanceState() callback *after* the onPause()method, and *before* onStop().
  + When the activity is recreated after it was previously destroyed, we can recover the saved state from the Bundle that the system passes to your activity. Both the onCreate() and onRestoreInstanceState() callback methods receive the same Bundle that contains the instance state information.
* What are Intents and what are the major types? When would you use one over another?
  + An Intent is a messaging object you can use to request an action from another app component. Intents are used for below cases
    - Starting an activity
    - Starting a service
    - Delivering a broadcast

Type of Intents :

* Explicit Intent : specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name.
* Implicit Intent : do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. Android OS takes the action on the app component whichever is appropriate.If the intent matches an intent filter, the system starts that component and delivers it the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.
* What is the difference between ListView and RecyclerView?
  + Recycler View is the successor of ListView and gridView.Major differences are as mentioned below;
  + **Required ViewHolder in Adapters** - ListView adapters do not require the use of the ViewHolder pattern to improve performance. In contrast, implementing an adapter for RecyclerView requires the use of the ViewHolder pattern for which it uses RecyclerView.Viewholder.
  + **Customizable Item Layouts** - ListView can only layout items in a vertical linear arrangement and this cannot be customized. In contrast, the RecyclerView has a RecyclerView.LayoutManager that allows any item layouts including horizontal lists or staggered grids.
  + **Easy Item Animations** - ListView contains no special provisions through which one can animate the addition or deletion of items. In contrast, the RecyclerView has the RecyclerView.ItemAnimator class for handling item animations.
  + **Manual Data Source** - ListView had adapters for different sources such as ArrayAdapter and CursorAdapter for arrays and database results respectively. In contrast, the RecyclerView.Adapter requires a custom implementation to supply the data to the adapter.
  + **Manual Item Decoration** - ListView has the android:divider property for easy dividers between items in the list. In contrast, RecyclerView requires the use of a RecyclerView.ItemDecoration object to setup much more manual divider decorations.
  + **Manual Click Detection** - ListView has a AdapterView.OnItemClickListener interface for binding to the click events for individual items in the list. In contrast, RecyclerView only has support for RecyclerView.OnItemTouchListener which manages individual touch events but has no built-in click handling.
* What is the ViewHolder pattern? Why should we use it?
  + In case of ListView , for every row being created, we have to make continuous calls to FindViewById for getting the references to the view objects and associate them to the row View. Even when the Adapter returns an inflated view for recycling, we still need to look up the elements and update them
  + We can improve the performance by applying ViewHolder pattern which speeds up the population of the ListView considerably by caching view lookups for smoother, faster item loading.
  + ViewHolder class puts in the cache the call to findViewById() and hence the performance is improved.A ViewHolder object stores each of the component views inside the tag field of the Layout, so we can immediately access them without the need to look them up repeatedly.
* Explain all the major Android components (e.g activities, services, broadcast receivers, content providers) and how they each are used in apps.
  + Activities: An activity represents a single screen with a user interface,in-short Activity performs actions on the screen. Activities are one of the fundamental building blocks of apps on the Android platform. They serve as the entry point for a user's interaction with an app, and are also central to how a user navigates within an app (as with the Back button) or between apps (as with the Recents button).
  + Services : A Service is an application component that can perform long-running operations in the background, and it does not provide a user interface. Example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.
  + Content Providers:
    - Content providers can help an application manage access to data stored by itself, stored by other apps, and provide a way to share data with other apps.
    - They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process.
  + Broadcast receivers:
    - Android apps can send or receive broadcast messages from the Android system and other Android apps, similar to the publish-subscribe design pattern.
    - These broadcasts are sent when an event of interest occurs. For example, the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging.
    - Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.
* Explain the difference between Service, IntentService, and AsyncTask? When would you use one over another?
  + Service
    - A Service is an application component that can perform long-running operations in the background, and it does not provide a user interface.
    - In Order to update the UI we need to make use of Handlers. Services cannot update the UI directly.
    - Services by default run on main thread of the android application. We have to start a new thread and then make the service run on it if required.
    - When an application who created services terminates service also terminates.
    - To Use it extend the service class and implement the onStartCommand callback function.
  + Intent Service
    - Intent service is a kind of service that handles async requests on demans. Intent Service creates a separate worker thread. It executes the job on this worker thread and once the job is over, intent service stops itself.
    - This "work queue processor" pattern is commonly used to offload tasks from an application's main thread. The IntentService class exists to simplify this pattern and take care of the mechanics.
    - To use it, extend IntentService and implement onHandleIntent(Intent).
  + Async Tasks
    - AsyncTask enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.
    - AsyncTasks should ideally be used for short operations (a few seconds at the most.)
    - An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. Extend the class AsyncTask to use AsyncTasks.
* Explain what Looper, Handler and HandlerThread are and how they are used.
  + Looper
    - Android modifies the user interface and handles input events from one single thread, called the *main thread*. Android collects all events in this thread in a queue and processes this queue with an instance of the Looperclass.
    - Threads by default do not have a message loop associated with them; to create one, call prepare() in the thread that is to run the loop, and then loop() to have it process messages until the loop is stopped.
  + Handler
    - Handler objects always has the reference to the thread which created it.
    - It provides a channel to send data to this thread, for example the main thread. The data which can be posted via the Handler class can be an instance of the Message or the Runnable class.
    - A Handler is particular useful if you have want to post multiple times data to the main thread.
  + HandlerThread
    - A class which starts a new thread that has a looper in it. The looper can then be used to create handler classes
* Explain the difference between services and threads in Android? How and why are they used?
  + Services
    - A Service is an application component that can perform long-running operations in the background, and it does not provide a user interface.
    - Services by default run on main thread of the android application. We have to start a new thread and then make the service run on it if required.
    - When an application who created services terminates service also terminates.
    - To Use it extend the service class and to start service call startService()
    - Service is a component of android, so it has priority levels to be considered while destroying an application due to low memory.
    - Blocks main thread if it is long running
  + Thread
    - It is OS level feature that allows us to do some operation in background.
    - Extends the class Thread or Implement the interface runnable for implementation of threads.
    - Thread.start method
    - Thread starts on its own new thread.
    - Manual thread management and code becomes difficult to read
* Why should you avoid running non-ui code on the main thread?
  + If we run the background tasks on main thread, it blocks the UI and the user interaction will not happen since the UI thread is blocked. Hence the application cannot process the UI events, input events etc.
  + This is a bad user experience. If the background tasks takes more than 5secs to finish, then the android application show Application Not responding (ANR) dialog.
* What are the options for persisting data in an Android app? When would you use the various different options available?
  + **Shared Preferences** - Easily save basic data as key-value pairs in a private persisted dictionary.
  + **Local Files** - Save arbitrary files to internal or external device storage.

Android can read/write files to internal as well as external storage. Applications have access to an application-specific directory where preferences and sqlite databases are also stored.

* + **SQLite Database** - Persist data in tables within an application specific database.Used for complex local data manipulation or for raw speed.
  + **ORM(Object Relational Mappers)** - Describes and persist model objects using a higher level query/update syntax. Used to store simple relational data locally to reduce SQL boilerplate.
* What is the difference between a fragment and an activity? Explain the relationship between the two.
  + Activity
    - Activity is an android component. Activity can have multiple fragments, but fragment cannot contain activities.
    - When an activity is placed to the backstack of activities the user can navigate back to the previous activity by just pressing the back button.
    - Activity can exist independently.
    - Without using fragment in Activity we can't create multi-pane UI.but using multiple fragments in single activity we can create multi-pane UI.
    - Activity has own life cycle but fragment has there own life cycle.
    - We need to mention activity in Android Manifest.
  + Fragments
    - Fragments are part of activity lifecycle which contribute its own UI to that activity.
    - Fragment cannot exist independent of an activity.
    - No need to mention fragments in android manifest.
* How would you communicate between two Fragments? Describe the best practices for communication.
  + Once we have the fragments, we need to associate them with an activity and connect them using activity to make a composite UI.
  + All Fragment-to-Fragment communication is done through the associated Activity. Two Fragments should never communicate directly.
  + Define an Interface :
    - To allow a Fragment to communicate up to its Activity, we can define an interface in the Fragment and implement in the activity.
    - The Fragment captures the interface implementation during its onAttach() lifecycle method and can then call the Interface methods in order to communicate with the Activity.
  + Deliver a message to Fragment
    - The host activity can deliver a message to the fragment, by capturing the Fragment instance with findFragmentById(), then directly call the fragment's public methods.
* What are "launch modes"? What are some common ways you might use launch modes.
  + A task is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack—the *back stack*)—in the order in which each activity is opened
  + Launch modes allow you to define how a new instance of an activity is associated with the current task. The launchMode attribute specifies an instruction on how the activity should be launched into a task and needs to be mentioned in the android Manifest file.
    - Standard

This is the default launch mode. The system creates a new instance of the activity in the task from which it was started and routes the intent to it. The activity can be instantiated multiple times, each instance can belong to different tasks, and one task can have multiple instances.

* + - Single Top

If an instance of the activity already exists at the top of the current task, the system routes the intent to that instance through a call to its onNewIntent() method, rather than creating a new instance of the activity.

* + - Single Task

The system creates a new task and instantiates the activity at the root of the new task.However, if an instance of the activity already exists in a separate task, the system routes the intent to the existing instance through a call to its onNewIntent() method, rather than creating a new instance. Only one instance of the activity can exist at a time.

* + - Single Instance

Same as "singleTask", except that the system doesn't launch any other activities into the task holding the instance. The activity is always the single and only member of its task; any activities started by this one open in a separate task.

* What is a BroadcastReceiver? What is a LocalBroadcastManager? What situations might you commonly use these?
  + Android apps can send or receive broadcast messages from the Android system and other Android apps, similar to the publish-subscribe design pattern. These broadcasts are sent when an event of interest occurs. Broadcast receivers are used to receive and handle broadcasts intents.
  + Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.
  + We can either dynamically register an instance of this class with Context.registerReceiver() or statically declare an implementation with the<receiver> tag in your AndroidManifest.xml.
  + Android sends broadcast in 3 ways
    - The sendOrderedBroadcast(Intent, String) method sends broadcasts to one receiver at a time.
    - The sendBroadcast(Intent) method sends broadcasts to all receivers in an undefined order. This is called a Normal Broadcast.
    - The LocalBroadcastManager.sendBroadcast method sends broadcasts to receivers that are in the same app as the sender. If we don't need to send broadcasts across apps, we use local broadcasts.
  + For Security concerns, in many cases, we dont need to send broadcasts to components outside our app.then we can send and receive local broadcasts with the LocalBroadcastManager.
  + We should not start activities from broadcast receivers because the user experience is jarring; especially if there is more than one receiver. Instead, consider displaying a notification.
* What is a ContentProvider and what is it typically used for? Would you commonly use them in an app, and if not, why not?
  + Content providers are Android’s central mechanism that enables you to access data of other applications – mostly information stored in databases or flat files. Without content providers accessing data of other apps would be a mess.
  + Yes, content providers will be commonly used to save the local application data and fast access to data.
* How do you handle Bitmaps in Android and what are common issues associated with them? How do you address these issues?
  + Bitmaps are usually to display some images in android using ImageView.
  + Below are the issues of bitmaps in android since the loading of bitmaps can be tricky.
    - Bitmaps can very easily exhaust an app's memory budget. Such a large memory demand can immediately use up all the memory available to the app.
    - Loading bitmaps on the UI thread can degrade the app's performance, causing slow responsiveness or even ANR messages. It is therefore important to manage threading appropriately when working with bitmaps.
    - If the app is loading multiple bitmaps into memory, we need to skillfully manage memory and disk caching. Otherwise, the responsiveness and fluidity of the app's UI may suffer.
  + We can overcome these issues by using image loading libraries such as Glide, Picasso, Fresco to fetch, decode, and display bitmaps in the app. These image loading libraries also take care of caching and memory management for bitmaps.
* What is the function of an intent filter? When would you use them in an app?
  + In order to advertise which intents our app can handle or receive, we need to declare the filters for intents. These are intent filter and needs to be declared in the Android Manifest xml file.
  + Each intent filter specifies the type of intents it accepts based on the intent's action, data, and category.
  + The system delivers an implicit intent to your app component only if the intent can pass through one of your intent filters. An app component should declare separate filters for each unique job it can do.
  + Each intent filter is defined by an <intent-filter> element in the app's manifest file, nested in the corresponding app component (such as an <activity> element). Inside the <intent-filter>, you can specify the type of intents to accept using one or more of these three elements:
    - <action> : Declares the intent action accepted, in the name attribute. The value must be the literal string value of an action, not the class constant.
    - <data> : Declares the type of data accepted, using one or more attributes that specify various aspects of the data URI and MIME type.
    - <category> : Declares the intent category accepted, in the name attribute. The value must be the literal string value of an action, not the class constant.
  + An implicit intent is tested against a filter by comparing the intent to each of the three elements. To be delivered to the component, the intent must pass all three tests.

## 3. Extended Android

Research and answer the following extended Android questions:

1. What is the difference JVM, DVM and ART?
2. Have you used ConstraintLayout and what do you think of it?
3. What is the architecture you used for your last app? Describe how the files and components were organized and what worked well or could be improved.
4. What is Dependency Injection? Why is it useful? Can you name few libraries?
5. What do you think of RxJava? What are the benefits of using it? Are there any cons.
6. What is Android Data Binding? Do you like using this in your apps, why or why not?
7. What is an ORM? What ORM do you use? What is Room and how do you think this compares with other ORMs?
8. What is a SurfaceView? Why would you use one?
9. What is overdraw? Why is this important and how do you fix this?
10. Have you used Kotlin and what do you think of it?
11. How to avoid memory leaks in Android? How to discover them?
12. Name 3 libraries that you like to use in apps and why?
13. What’s a new Android SDK feature or API that was released recently that looked interesting or are playing with?
14. What is a JobScheduler? How does this differ from other alternatives?
15. How do you play sounds in Android?
16. Describe any experience or knowledge high level of unit testing. What is Espresso used for? What is Mockito used for? What is the difference between unit and instrumented tests?
17. What is obfuscation? What is it used for? What common tools enables this?

## 3. Appendix: Answer Sheet (Work In Progress)

### 3.1 Core Java

**3.1.1 What is the difference between abstract classes and interfaces?**

* An abstract class, is a class that contains both concrete and abstract methods (methods without implementations). An abstract method must be implemented by the abstract class subclasses. Abstract classes cannot be instantiated and need to be extended to be used.
* An interface is like a blueprint/contract of a class (or it may be thought of as a class with methods, but without their implementation). It contains empty methods that represent, what all of its subclasses should have in common. The subclasses provide the implementation for each of these methods. Interfaces are implemented.

**3.1.2 What is serialization in Java (not Android)? How do you implement it? How about in Android?**

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. In Android you may use either the Serializable, or Parcelable interfaces.

It is highly recommended to use Parcelable in Android instead, as Parcelable was created exclusively for Android and it performs about 10x faster than Serializable, because Serializable uses reflection, which is a slow process and tends to create a lot of temporary objects and it may cause garbage collection to occur more often.

### 3.2 Core Android

### 3.3 Extended Android

## 4. References

References to review related to Android interview questions:

* <https://github.com/MindorksOpenSource/android-interview-questions>
* <https://courses.codepath.com/courses/intro_to_android/pages/interview_prep_guide>