**Most frequently asked Java Questions-**

1. Why is Java said to be platform independent?

The execution of the code does not depend upon the OS

2. Difference between ‘throw’ and ‘throws’ in Java Exception Handling?

throw keyword is used to throw Exception from any method or static block whereas throws is used to indicate that which Exception can possibly be thrown by this method.

3. Is there ever a scenario where we can skip the finally block in a try catch?

By Calling System.exit(0) in try or catch block, we can skip the finally block. System.exit(int) method can throw a SecurityException. If System.exit(0) exits the JVM without throwing that exception, then finally block will not execute. But, if System.exit(0) does throw security exception then finally block will be executed.

4. What are anonymous classes?

An anonymous class is just what its name implies — it has no name. It combines the class declaration and the creation of an instance of the class in one step. Since anonymous classes have no name, objects cannot be instantiated from outside the class in which the anonymous class is defined. In fact, an anonymous object can only be instantiated from within the same scope in which it is defined.

Rules:

* An anonymous class must always extend a super class or implement an interface but it cannot have an explicit extends or implements clause.
* An anonymous class must implement all the abstract methods in the super class or the interface.
* An anonymous class always uses the default constructor from the super class to create an instance.
* Example:

MyButton.setOnClickListener(new Button.OnClickListener {  
 @override  
 public void onClick(View view){  
 //some code  
 }  
 });

5. Why is the main method static in java?

The method is static because otherwise there would be ambiguity on which method to be called. If static is removed from the main method, Program compiles successfully. But at runtime throws an error “NoSuchMethodError”.

We can overload the main method in Java. But the program doesn’t execute the overloaded main method when we run your program, we need to call the overloaded main method from the actual main method only. To run a method without calling this main method, we would need to execute a static block.

In order to avoid NoSuchMethodError, we can call System.exit(0) after the static method.

Note: Any method declared static will be executed even before the main class is executed.

Example:

public class Hello {   
 static {   
 System.out.println("Hello, World!");   
 }   
 }

6. What is garbage collector? How does it work?

All objects are allocated on the heap area managed by the JVM. As long as an object is being referenced, the JVM considers it alive. Once an object is no longer referenced and therefore is not reachable by the application code, the garbage collector removes it and reclaims the unused memory.

7. Difference between stack memory & heap memory?

Stack is used for static memory allocation and Heap for dynamic memory allocation, both stored in the computer’s RAM .

Variables allocated on the stack are stored directly to the memory and access to this memory is very fast, and it’s allocation is dealt with when the program is compiled. When a function or a method calls another function which in turns calls another function etc., the execution of all those functions remains suspended until the very last function returns its value. The stack is always reserved in a LIFO order, the most recently reserved block is always the next block to be freed. This makes it really simple to keep track of the stack, freeing a block from the stack is nothing more than adjusting one pointer.

Variables allocated on the heap have their memory allocated at run time and accessing this memory is a bit slower, but the heap size is only limited by the size of virtual memory . Element of the heap have no dependencies with each other and can always be accessed randomly at any time. You can allocate a block at any time and free it at any time. This makes it much more complex to keep track of which parts of the heap are allocated or free at any given time.

You can use the stack if you know exactly how much data you need to allocate before compile time and it is not too big. You can use heap if you don’t know exactly how much data you will need at runtime or if you need to allocate a lot of data.

In a multi-threaded situation each thread will have its own completely independent stack but they will share the heap. Stack is thread specific and Heap is application specific. The stack is important to consider in exception handling and thread executions.

Heap memory is used by all the parts of the application whereas stack memory is used only by one thread of execution.

Whenever an object is created, it’s always stored in the Heap space and stack memory contains the reference to it. Stack memory only contains local primitive variables and reference variables to objects in heap space.

Objects stored in the heap are globally accessible whereas stack memory can’t be accessed by other threads.

Memory management in stack is done in LIFO manner whereas it’s more complex in Heap memory because it’s used globally. Heap memory is divided into Young-Generation, Old-Generation etc, more details at Java Garbage Collection.

Stack memory is short-lived whereas heap memory lives from the start till the end of application execution.

When stack memory is full, Java runtime throws java.lang.StackOverFlowError whereas if heap memory is full, it throws java.lang.OutOfMemoryError: Java Heap Space error.

Stack memory size is very less when compared to Heap memory. Because of simplicity in memory allocation (LIFO), stack memory is very fast when compared to heap memory.

8. Explain OOPs concept

Object Oriented Programming is a programming style that involves concepts such as Classes, objects,Abstraction, Encapsulation, Inheritance, Polymorphism.

9. What is Inheritance?

Inheritance is the process by which objects of one class acquire the properties & objects of another class. The two most common reasons to use inheritance are:  
a) To promote code reuse. b) To use polymorphism.

10. Does Java support multiple inheritance?

Java supports multiple inheritance by interface only since it can implement multiple interfaces but can extend only one class.

11. What is Encapsulation?

* Encapsulation involves binding code and data together as a single unit.
* Encapsulation is a technique used for hiding the properties and behaviours of an object and allowing outside access only as appropriate. It prevents other objects from directly altering or accessing the properties or methods of the encapsulated object.
* For instance, a class can be an encapsulated class if all the variables in it are defined as Private and by providing getter and setter methods.

12. What is Abstract class?

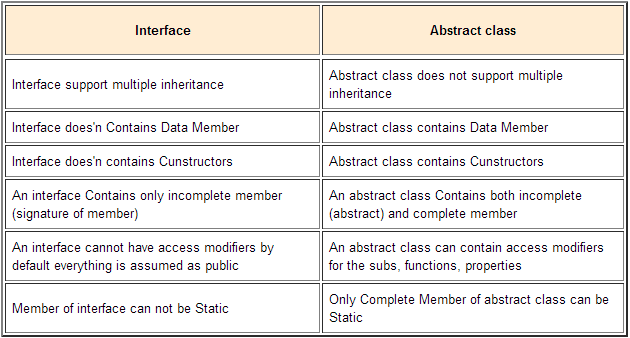
* Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.
* If even a single method is abstract, the whole class must be declared abstract.
* Abstract classes may not be instantiated and require subclasses to provide implementations for the abstract methods.
* You can’t mark a class as both abstract and final.
* Non-abstract methods can access a method that you declare as abstract.

13. What are Interfaces?

* Interfaces are only declared methods that an implementing class would need.
* Interfaces cannot be marked as final. Interface variables must be static or final.
* Interfaces cannot be instantiated directly.
* Marker Interfaces: Marker interfaces are those which do not declare any required Methods. The java.io. Serializable interface is a typical marker interfaces. These do not contain any methods, but classes must implement this interface in order to be serialized and de-serialized.

14. Difference between Abstract and Interfaces?

Abstract classes can have non abstract methods. It can have instance variables. We have provide default implementation to abstract class method. A class can extend only one abstract class.A class can implement multiple interfaces.



15. What is Polymorphism?

Polymorphism is when an object takes on multiple forms. For instance, String is a subclass of Object class.

Polymorphism manifests itself in Java in the form of multiple methods having the same name. In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods). In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

Polymorphism is a characteristic of being able to assign a different meaning or usage to something in different contexts — specifically, to allow an entity such as a variable, a function, or an object to have more than one form.

2 forms of polymorphism:

* Compile time polymorphism: The flow of control is decided during the compile time itself. By overloading.
* Run time polymorphism: is done using inheritance and interface. The flow of control is decided during the runtime. Overriding: Overriding will have the same method name with the same parameters. One will be the parent class method and the other will be the child class method. Overloading occurs when the same method name is declared but with different parameters.

16. What is Method overloading?

Method Overloading means to have two or more methods with same name in the same class with different arguments.

Note:

* Overloaded methods MUST change the argument list
* Overloaded methods CAN change the return type
* Overloaded methods CAN change the access modifier
* Overloaded methods CAN declare new or broader checked exceptions
* A method can be overloaded in the same class or in a subclass

17. What is Method overriding?

Method overriding occurs when sub class declares a method that has the same type arguments as a method declared by one of its superclass

* You can’t override a method marked public and make it protected
* You cannot override a method marked final
* You cannot override a method marked static

Note: Static methods cannot be overridden. Overloaded methods can still be overridden.

18. Why would you not call abstract method in constructor?

The problem is that the class is not yet fully initialised, and when the method is called in a subclass, it may cause trouble.

19. Composition over inheritance?

Composition is typically “has a” or “uses a” relationship. In the below example, the Employee class has a Person. It does not inherit from Person but instead gets the Person object passed to it, which is why it is a “has a” Person.

class Person {  
 String Title;  
 String Name;  
 Int Age; public Person(String title, String name, String age) {  
 this.Title = title;  
 this.Name = name;  
 this.Age = age;  
 }}class Employee {  
 Int Salary;  
 private Person person; public Employee(Person p, Int salary) {  
 this.person = p;  
 this.Salary = salary;  
 }  
}

20. Difference between Encapsulation & Abstraction?

* Abstraction focuses on the outside view of an object (i.e. the interface)
* Encapsulation (information hiding) prevents clients from seeing it’s inside view.
* Abstraction solves the problem in the design side while Encapsulation is the Implementation.

21. Constructors vs Methods?

Constructors must have the name as the class name and does not have a return type. It can be used to instantiate any objects in the class whereas methods have no such rule and is another member of the class. Constructors cannot be inherited but a derived class can call the super constructor of parent class.

* this(): Constructors use this to refer to another constructor in the same class with a different parameter list.
* super(): Constructors use super to invoke the superclass's constructor.

Methods: Instance methods on the other hand require an instance of the class to exist before they can be called, so an instance of a class needs to be created by using the new keyword. Class methods are methods which are declared as static. The method can be called without creating an instance of the class.

22. What is the difference between instantiation and initialisation of an object?

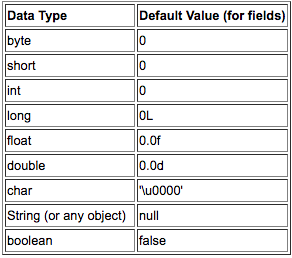
Initialisation is the process of the memory allocation, when a new variable is created. Variables should be explicitly given a value, otherwise they may contain a random value that remained from the previous variable that was using the same memory space. To avoid this problem, Java language assigns default values to data types.

Instantiation is the process of explicitly assigning definitive value to a declared variable.

23. Do objects get passed by reference or value in Java? Elaborate on that.

* Java is always pass-by-value. When we pass the value of an object, we are passing the reference to it.
* Java creates a copy of the variable being passed in the method and then does the manipulations. Hence the change is not reflected in the main method.
* But when you pass an object reference into a method, a copy of this reference is made, so it still points to the same object. This means, that any changes that you make to the insides of this object are retained, when the method exits.
* Java copies and passes the reference by value, not the object. Thus, method manipulation will alter the objects, since the references point to the original objects. But since the references are copies, swaps will fail.

24. Primitives in Java?



25. Difference between == and .equals() method in Java?

We can use == operators for reference comparison (address comparison) and .equals() method for content comparison.  
\* In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.

26. Why strings are Immutable?

Once a value is assigned to a string it cannot be changed. And if changed, it creates a new object of the String. This is not the case with StringBuffer.

27. What is String.intern()? When and why should it be used?

* String.intern() method can be used to to deal with String duplication problem in Java. By carefully using the intern() method you can save a lot of memories consumed by duplicate String instances. A string is duplicate if it contains the same content as another string but occupied different memory location.
* By calling the intern() method on a string object, for instance “abc”, you can instruct JVM to put this String in the pool and whenever someone else creates “abc”, this object will be returned instead of creating a new object. This way, you can save a lot of memory in Java, depending upon how many Strings are duplicated in your program.
* When the intern method is invoked, if the String pool already contains that String object such that equals() return true, it will return the String object from the pool, otherwise it will add that object to the pool of unique String.

28. String pool in Java:

String Pool in java is a pool of Strings stored in Java Heap Memory. When we use double quotes to create a String, it first looks for String with same value in the String pool, if found it just returns the reference else it creates a new String in the pool and then returns the reference. However using new operator, we force String class to create a new String object in heap space. We can use intern() method to put it into the pool or refer to other String object from string pool having same value.

For example, how many strings are getting created in below statement;

String str = new String("Cat");

In above statement, either 1 or 2 string will be created. If there is already a string literal “Cat” in the pool, then only one string “str” will be created in the pool. If there is no string literal “Cat” in the pool, then it will be first created in the pool and then in the heap space, so total 2 string objects will be created.

29. Final modifier?

Final modifiers — once declared cannot be modified. A blank final variable in Java is a final variable that is not initialised during declaration.

* final Classes- A final class cannot have subclasses.
* final Variables- A final variable cannot be changed once it is initialised.
* final Methods- A final method cannot be overridden by subclasses.

30. Finalize keyword?

Finalize is a method used to perform clean up processing just before object is garbage collected.

31. Finally keyword?

finally is a code block and is used to place important code, it will be executed whether exception is handled or not.

32. Static variables?

Variables that have only one copy per class are known as static variables. They are not attached to a particular instance of a class but rather belong to a class as a whole. A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

33. What is reflection?

Java Reflection makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time. It is also possible to instantiate new objects, invoke methods and get/set field values using reflection.

34. Multi threading?

Multiple tasks are running concurrently in a program.

35. Fail-fast & Fail-Safe?

Fail-fast Iterators throws ConcurrentModificationException when one Thread is iterating over collection object and other thread structurally modify Collection either by adding, removing or modifying objects on underlying collection. They are called fail-fast because they try to immediately throw Exception when they encounter failure.

On the other hand [fail-safe](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) Iterators works on copy of collection instead of original collection.

36. What does the keyword synchronized mean?

When you have two threads that are reading and writing to the same ‘resource’, say a variable named ‘test’, you need to ensure that these threads access the variable in an atomic way. Without the synchronized keyword, your thread 1 may not see the change thread 2 made to test.

synchronized variables blocks the next thread’s call to method as long as the previous thread’s execution is not finished. Threads can access this method one at a time.

37. What does the keyword volatile mean?

* Suppose two threads are working on a method. If two threads run on different processors each thread may have its own local copy of variable. If one thread modifies its value the change might not reflect in the original one in the main memory instantly.
* Now the other thread is not aware of the modified value which leads to data inconsistency.Essentially, volatile is used to indicate that a variable’s value will be modified by different threads. ‘volatile’ tells the compiler that the value of a variable must never be cached as its value may change outside of the scope of the program itself.
* The value of this variable will never be cached thread-locally: all reads and writes will go straight to ‘main memory’
* An access to a volatile variable never has the potential to block: we’re only ever doing a simple read or write, so unlike a synchronized block we will never hold on to any lock.

38. What is Autoboxing and Unboxing?

Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes. For example, converting an int to an Integer, a double to a Double, and so on. If the conversion goes the other way, this is called unboxing.

39. Optionals in Java?

Optional is a container object which is used to contain not-null objects. Optional object is used to represent null with absent value. This class has various utility methods to facilitate code to handle values as ‘available’ or ‘not available’ instead of checking null values.

40. What is externalization?

* In serialization, the JVM is responsible for the process of writing and reading objects. This is useful in most cases, as the programmers do not have to care about the underlying details of the serialization process.
* However, the default serialization does not protect sensitive information such as passwords and credentials.
* Thus externalization comes to give the programmers full control in reading and writing objects during serialization.
* Implement the java.io.Externalizable interface — then you implement your own code to write object’s states in the writeExternal() method and read object’s states in the readExternal() method.

41. What are Data Structures?

Intentional arrangement of a collection of data. There are 5 fundamental behaviours of a data structure: access, insert, delete, find & sort.

42. Explain Big O Notation?

The notation Ο(n) is the formal way to express the upper bound of an algorithm’s running time. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

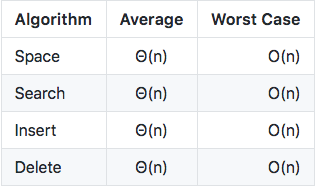
Note: O(1) means that it takes a constant time, like three minutes no matter the amount of data in the set. O(n)means it takes an amount of time linear with the size of the set.

43. Explain Big Omega Notation

The Big Omega Notation is used to describe the best case running time for a given algorithm.

44. Arrays in Java?

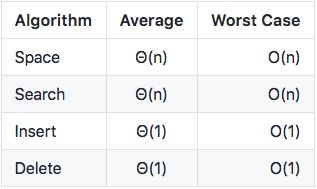
* Arrays is an ordered collection. It will have a fixed length which needs to be defined at the initialisation time whereas lists have a variable length. Arrays are easier to store elements of the same data type. Used internally in stack and queue. It is a convenient way of representing a 2D array.
* Arrays cannot hold generic data types whereas lists can.
* Arrays can store all data types whereas lists cannot store primitive data types, only objects.



Complexity of Arrays

45. Linked Lists in Java?

* A LinkedList contains both a head and a tail. The “Head” is the first item in the LinkedList, while the “Tail” is the last item. It is not a circular data structure, therefore the tail does not have its’ pointer pointing at the Head — the pointer is just null.
* No indices but each node has a pointer pointing to the next element.
* They are dynamic in nature which means they allocate memory only when needed.
* Insertion, deletion, updation is easy
* A linked list is a group of nodes which represent a sequence. Each node consists of a data and a link or reference to the next node in the sequence.
* Singly Linked List: has a node and a pointer to the next node in the sequence.
* Doubly Linked List: has a node and 2 pointers — one to the next node and one to the previous node in the sequence. This is very convenient if you need to be able to traverse stored elements in both directions.



Complexity of LinkedList

46. Binary Trees

* A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.
* The left subtree of a node contains only values less than the parent node’s value.
* The right subtree of a node contains only values greater than or equal to the node’s value.
* Only if the above 2 criteria are matched, then the tree is said to be balanced.
* Advantages of Binary tree over Linked List: In a linked list, the items are linked together through a single next pointer. In a binary tree, as long as the tree is balanced, the search path to each item is a lot shorter than that in a linked list.
* Their disadvantage is that in the worst case they can degenerate into a linked list in terms of efficiency.

47. Stacks:

Stacks are an abstract collection that follow LIFO mechanism. Main functionalities include:

* Push: a new entity added to the top of the stack.
* Pop: an entity is removed from the top of the stack.

The process of accessing data stored in a serial access memory is similar to manipulating data on a stack.

* A stack may be defined to have a bounded capacity i.e. if the stack is full and a new entity cannot be added, then it is considered to be in an overflow state.
* If the stack is empty and an entity cannot be popped, it is considered to be in an underflow state.
* Efficiency of stacks: The time is not dependent of the no of items in the stack so it is very efficient. O(1).

48. Queues:

Queues are an abstract collection that follow FIFO mechanism. The entities in the queue are kept in an order. Main functionalities include:

* enqueue: Add an item to the end of the queue. Dequeue: remove an item from the start of the queue.
* Front: retrieves the first item from the queue.
* A queue may be defined to have a bounded capacity i.e. if the queue is full and a new entity cannot be added, then it is considered to be in an overflow state.
* If the queue is empty and an entity cannot be popped, it is considered to be in an underflow state.
* Efficiency of queues: The time is not dependent of the no of items in the queue so it is very efficient. O(1).
* A double ended queue (deque): is an abstract collection which differs from queue in a way that an item can be added/removed from either side of the queue.
* An input-restricted deque: is when deletion takes place at either end but insertion takes place at only one end.
* An output-restricted deque: is when insertion takes place at either end but deletion takes place only at one end. A common occurrence of deque is doubly linked list.
* Priority queue: same as queue but has a priority associated with it. Items are retrieved based on their priority.

49. Blocking Queues:

A blocking queue is a queue that blocks when you try to dequeue from it and the queue is empty, or if you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue. A thread trying to enqueue an item in a full queue is blocked until some other thread makes space in the queue.

50. Difference between stacks & queues?

Stack and Queue both are the non-primitive data structures. The main differences between stack and queue are that stack uses LIFO (last in first out) method to access and add data elements whereas Queue uses FIFO (First in first out) method to access and add data elements.

51. What is a deadlock in Java

A deadlock occurs when a thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process.

52. What is the List interface & Set interface?

List interface supports for ordered collection of objects and it may contain duplicates. The Set interface provides methods for accessing the elements of a finite mathematical set. Sets do not allow duplicate elements

53. Difference between ArrayList & Vectors?

Vectors are thread safe (synchronized) whereas arraylists are not. So, performance of array lists are better than vectors. \*In ArrayList, you have to start searching for a particular element from the beginning of an Arraylist. But in the Vector, you can start searching for a element from a particular position in a vector. This makes the search operation in Vector faster than in ArrayList. Vectors have a default size of 10 whereas arraylists size can be dynamic.

54. Insertion and deletion in ArrayList is slow compared to LinkedList?

* ArrayList internally uses an array to store the elements, when that array gets filled by inserting elements a new array of roughly 1.5 times the size of the original array is created, and all the data of old array is copied to new array.
* During deletion, all elements present in the array after the deleted elements have to be moved one step back to fill the space created by deletion. In linked list data is stored in nodes that have reference to the previous node and the next node so adding element is simple as creating the node and updating the next pointer on the last node and the previous pointer on the new node. Deletion in linked list is fast because it involves only updating the next pointer in the node before the deleted node and updating the previous pointer in the node after the deleted node.

55. Implementations of Map?

* TreeMap: sorted based on ascending order of keys. For inserting, deleting, and locating elements in a Map, the HashMap offers the best alternative. If, however, you need to traverse the keys in a sorted order, then TreeMap is your better alternative.
* HashTable: Does not allow null values. It is not fail-safe, and it is synchronized whereas HashMap allows null values and it is fail-safe and it is not synchronized.
* LinkedHashMap: This is a subclass of Hashmap. The order of insertion is preserved since it has a linkedList.

56. Difference between Enumeration and Iterators?

* Enumeration does not include remove () method whereas iterators do. Enumerators act as read only interface as it provides methods to read and traverse through a collection.
* ListIterator: is just like an iterator except it allows access to the collection in either the forward or backward direction

57. How HashMap works in Java?

* HashMap in Java works on hashing principle. It is a data structure which allows us to store object and retrieve it in constant time O(1) provided we know the key. When we call put method, hashcode() method of the key object is called so that hash function of the map can find a bucket location to store Entry object.
* If two different objects have the same hashcode: in this case, a linked list is formed at that bucket location and a new entry is stored as next node. After finding bucket location, we will call keys.equals() method to identify a correct node in LinkedList and return associated value object for that key in Java HashMap

58. Generics in Java

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

* Type-safety: We can hold only a single type of objects in generics. It doesn’t allow to store other objects.
* Type casting is not required: There is no need to typecast the object.
* Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

**Most Frequently Asked Android Questions-**

## 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](https://developer.android.com/reference/android/arch/lifecycle/ViewModel" \t "_blank) and onSaveInstanceState() . So how we do we that?
* Basics of [ViewModel](https://developer.android.com/reference/android/arch/lifecycle/ViewModel" \t "_blank): 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](https://developer.android.com/reference/android/app/Service.html) 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](https://developer.android.com/guide/topics/ui/notifiers/notifications.html) 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](https://developer.android.com/topic/libraries/architecture/workmanager" \t "_blank) 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

## 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.

## 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.

## 30. AIDL vs Messenger Queue

* 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

## 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.

## 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](https://developer.android.com/reference/android/support/v4/app/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](https://developer.android.com/reference/android/support/v4/app/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.

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.

## 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().

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.

## 64. How to load bitmap to memory?

## 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.

.

## 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?

[SparseArray](https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/util/SparseArray.java) can be used to replace [HashMap](http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/util/HashMap.java) when the key is a primitive type. There are some variants for different key/value types, even though not all of them are publicly available.

Benefits are:

* Allocation-free
* No boxing

Drawbacks:

* Generally slower, not indicated for large collections
* They won't work in a non-Android project

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

Detecting Memory Leaks Using Android Studio’s Monitors.

Detecting Memory Leaks Using Leak Canary.

Detecting Possible Leaks With Infer.

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

* 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? 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.

* Room
* Live Data
* ViewModel
* Data Binding
* Lifecycles
* 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 issue.

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).

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.

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

* The Single Responsibility Principle (SRP) - A class should only have a single responsibility, that is, only changes to one part of the software’s specification should be able to affect the specification of the class.
* The Open-Closed Principle (OCP) - Software entities … should be open for extension, but closed for modification.
* The Liskov Substitution Principle (LSP) - Objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
* The Interface Segregation Principle (ISP) - Many client-specific interfaces are better than one general-purpose interface.
* The Dependency Inversion Principle (DIP) - One should depend upon abstractions, [not] concretions.

**Important Kotlin Questions-**

1: What’s the Target Platform of Kotlin? How is Kotlin-Java interoperability possible?

Java Virtual Machine (JVM) is the Target Platform of Kotlin. Kotlin is 100% interoperable with Java since both, on compilation produce bytecode. Hence Kotlin code can be called from Java and vice-versa.

2: How do you declare variables in Kotlin? How does the declaration differ from the Java counterpart?

There are two major differences between Java and Kotlin variable declaration:

The type of declaration  
In Java the declaration looks like this:

String s = “Java String”;

int x = 10

In Kotlin the declaration looks like:

val s: String = “Hi”

var x = 5

In Kotlin, the declaration begins with a val and a var followed by the optional type. Kotlin can automatically detect the type using type inference.

Default value  
The following is possible in Java:

String s;

The following variable declaration in Kotlin is not valid.

val s: String

3: What’s the difference between val and var declaration? How to convert a String to an Int?

val variables cannot be changed. They’re like final modifiers in Java. A var can be reassigned. The reassigned value must be of the same data type.

fun main(args: Array<String>) {  
val s: String = “Hi”  
var x = 5  
x = “6”.toInt()  
}

We use the toInt() method to convert the String to an Int.

4: What’s Null Safety and Nullable Types in Kotlin? What is the Elvis Operator?

Kotlin puts a lot of weight behind null safety which is an approach to prevent the dreaded Null Pointer Exceptions by using nullable types which are like String? Int?, Float? etc. These act as a wrapper type and can hold null values. A nullable value cannot be added to another nullable or basic type of value.

To retrieve the basic types we need to use safe calls that unwrap the Nullable Types. If on unwrapping, the value is null we can choose to ignore or use a default value instead. The Elvis Operator is used to safely unwrap the value from the Nullable.  
It’s represented as?: over the nullable type. The value on the right-hand side would be used if the nullable type holds a null.

var str: String? = “JournalDev.com”  
var newStr = str?: “Default Value”  
str = null  
newStr = str?: “Default Value”

5:What’s a const? How does it differ from a val?

By default val properties are set at runtime. Adding a const modifier on a val would make a compile-time constant.  
A const cannot be used with a var or on its own.  
A const is not applicable on a local variable.

6: Does Kotlin allow us to use primitive types such as int, float, double?

No kotlin does not support primitive datatypes as like in java. At the language level, we cannot use the above-mentioned types. But the JVM bytecode that’s compiled does certainly have them.

7: What’s the entry point of every Kotlin Program?

The main function is the entry point of every Kotlin program. In Kotlin we can choose not to write the main function inside the class. On compiling the JVM implicitly encapsulates it in a class.  
The strings passed in the form of Array<String> are used to retrieve the command line arguments.

8: How is !! different from ?. in unwrapping the nullable values? Is there any other way to unwrap nullable values safely?

!! is used to force unwrap the nullable type to get the value. If the value returned is a null, it would lead to a runtime crash. Hence a !! operator should be only used when you’re absolutely sure that the value won’t be null at all. Otherwise, you’ll get the dreaded null pointer exception. On the other hand, a ?. is an Elvis Operator that does a safe call.  
We can use the lambda expression let on the nullable value to unwrap safely as shown below.

9: How is a function declared? Why are Kotlin functions known as top-level functions?

fun sumOf(a: Int, b: Int): Int { return a + b }

10: What’s the difference between == and === operators in Kotlin?

== is used to compare the values are equal or not. === is used to check if the references are equal or not.

11: List down the visibility modifiers available in Kotlin. What’s the default visibility modifier?

* public
* internal
* protected
* private

public is the default visibility modifier.

12: Does the following inheritance structure compile?

By default classes are final in Kotlin. To make them non-final, you need to add the open modifier.

open class A { }

class B : A() { }

13: What are the types of constructors in Kotlin? How are they different? How do you define them in your class?

Constructors in Kotlin are of two types:  
**Primary** — These are defined in the class headers. They cannot hold any logic. There’s only one primary constructor per class.  
**Secondary** — They’re defined in the class body. They must delegate to the primary constructor if it exists. They can hold logic. There can be more than one secondary constructors.

**14: What’s init block in Kotlin:**

init is the initializer block in Kotlin. It’s executed once the primary constructor is instantiated. If you invoke a secondary constructor, then it works after the primary one as it is composed in the chain.

**15: How does string interpolation work in Kotlin? Explain with a code snippet?**

String interpolation is used to evaluate string templates.  
We use the symbol $ to add variables inside a string.

val name = “Journaldev.com”

val desc = “$name now has Kotlin Interview Questions too. ${name.length}”

Using {} we can compute an expression too.

**16: What’s the type of arguments inside a constructor?**

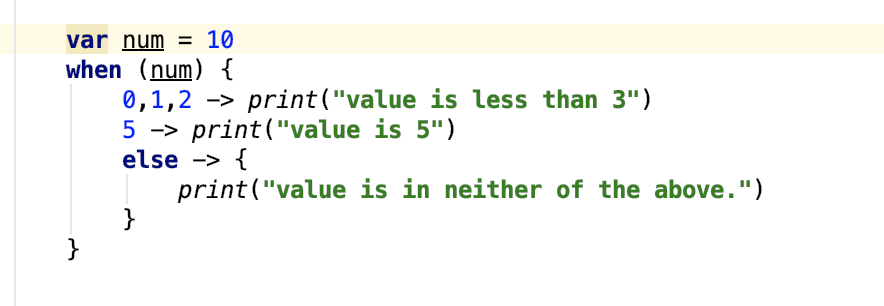
By default, the constructor arguments are val unless explicitly set to var.

**17: Is new a keyword in Kotlin? How would you instantiate a class object in Kotlin?**

**NO**. Unlike Java, in Kotlin, new isn’t a keyword.  
We can instantiate a class in the following way:

**18: What is the equivalent of switch expression in Kotlin? How does it differ from switch?**

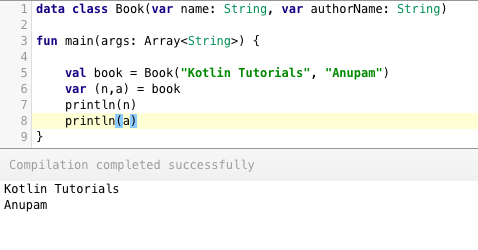
when is the equivalent of switch in Kotlin.  
The default statement in a when is represented using the else statement.



**19: What are data classes in Kotlin? What makes them so useful? How are they defined?**

In Java, to create a class that stores data, you need to set the variables, the getters and the setters, override the toString(), hash() and copy() functions.  
In Kotlin you just need to add the data keyword on the class and all of the above would automatically be created under the hood.

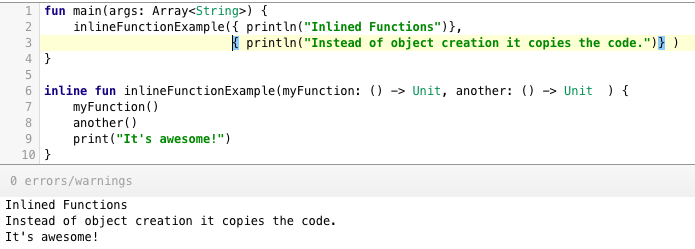
**20: What are destructuring declarations in Kotlin? Explain it with an example.**

Destructuring Declarations is a smart way to assign multiple values to variables from data stored in objects/arrays.  
 [[](https://cdn.journaldev.com/wp-content/uploads/2018/04/kotlin-interview-questions-destructuring-declarations.png)](https://cdn.journaldev.com/wp-content/uploads/2018/04/kotlin-interview-questions-destructuring-declarations.png)

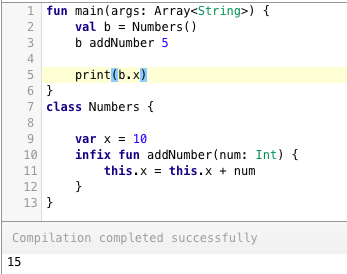
Within parentheses, we’ve set the variable declarations. Under the hood, de structuring declarations create component functions for each of the class variables.

**21: What’s the difference between inline and infix functions? Give an example of each.**

Inline functions are used to save us memory overhead by preventing object allocations for the anonymous functions/lambda expressions called. Instead, it provides that functions body to the function that calls it at runtime. This increases the bytecode size slightly but saves us a lot of memory.



infix functions on the other are used to call functions without parentheses or brackets. Doing so, the code looks much more like a natural language.



**21: What’s the difference between lazy and lateinit?**

Both are used to delay the property initializations in Kotlin  
lateinit is a modifier used with var and is used to set the value to the var at a later point.  
lazy is a method or rather say lambda expression. It’s set on a val only. The val would be created at runtime when it’s required.

val x: Int by lazy { 10 }

lateinit var y: String

**22: How to create Singleton classes?**

To use the singleton pattern for our class we must use the keyword object

An object cannot have a constructor set. We can use the init block inside it though.

object MySingletonClass

**23: Does Kotlin have the static keyword? How to create static methods in Kotlin?**

**NO**. Kotlin doesn’t have the static keyword.  
To create static method in our class we use the companion object.  
Following is the kotlin code:

class A { companion object { fun a() : Int = 5 } }

**24: What’s the type of the following Array?**

val arr = arrayOf(1, 2, 3);

The type is Array<Int>. That’s all for Kotlin interview questions and answers.

**25: Explain Higher-Order Functions?**

Higher-Order Functions: A higher-order function is a function that takes functions as parameters or returns a function.

**26: Can you execute Kotlin code without JVM?**

JVM, which stands for Java Virtual Machine is a feature of Kotlin. This feature compiles a Kotlin code into a native code, which can be done without JVM too.

**27: Mention the structural expressions in Kotlin?**

There are three Structural expressions in Kotlin.They are:

* **Return**: It returns from the nearest enclosing function or anonymous function by default.
* **Break**: This expression terminates the closest enclosing loop.
* **Continue**: This expression proceeds you to the next closest enclosing loop.

**28. What are the types of strings available in Kotlin? And, what do you mean by Kotlin String Interpolation?**

Strings are a collection of characters together.Kotlin features two types of strings, and they are:

* Raw string
* Escaped string

In Kotlin String, templates can be evaluated. This evaluation of string templates is called as the string template interpolation.

**29: State the advantages and disadvantages of Kotlin?**

**Advantages:**

Kotlin is simple and easy to learn as its syntax is similar to that of Java.

It is the functional language that is based on **JVM (Java Virtual Machine)**, which removes the boilerplate codes. Upon all this, Kotlin is considered as an expressive language that is easily readable and understandable and the performance is substantially good.  
It can be used by any desktop, web server or mobile-based applications.

**Disadvantages:**

Kotlin does not provide the static modifier, which causes problems for conventional java developer.

In Kotlin, the function declaration can be done in many places in the application, which creates trouble for the developer to understand which function is being called.

**30: Which type of Programming does Kotlin support?**

Kotlin supports only two types of programming, and they are:

* Procedural programming
* Object-oriented programming

## 30: Mention the structural expressions in Kotlin?

The following are the three structural [expressions in Kotlin](https://kotlinlang.org/docs/reference/returns.html).

Return: It returns value from the functions by default.

Break: terminates the loop condition.

Continue: precedes you to the next enclosing loop.

## 31: What are Coroutines?

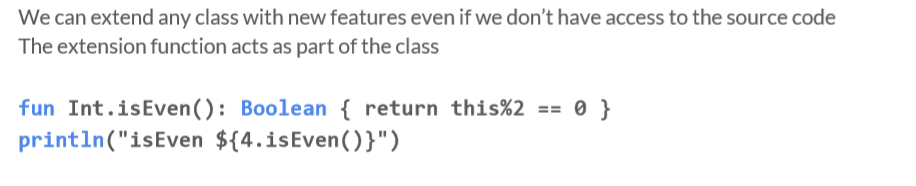
A coroutine is a light-weight thread that doesn’t require any context switching on the processor and will not map on native threads. This is the reason why they are fast in processing the requests.

You can start coroutines in one of two ways:

* launch starts a new coroutine and doesn't return the result to the caller. Any work that is considered "fire and forget" can be started using launch.
* async starts a new coroutine and allows you to return a result with a suspend function called await.

## 32: What are Extension Functions?

We can extend any class with new features even if we don't have access to the source code The extension function acts as part of the class. Basically, an extension function is a member function of a class that is defined outside the class.



## 33: What is lambda expression?

## Lambda Expressions are function literals that help us in writing a function in a short way. They provide us a compact way of writing code. If a function is not declared, but passed immediately as an expression, we call it a lambda, or an anonymous function. It is also called a “function literal.” Let’s define a lambda expression to see how functions act as types. // with type annotation in lambda expression val sum1 = { a: Int, b: Int -> a + b } // without type annotation in lambda expression val sum2:(Int,Int)-> Int = { a , b -> a + b} println(sum1(3,4))