InterviewBit

Java Interview Questions for 5 years Experience

Contents

Java Interview Questions for Experienced

Java Interview Questions for Experienced (.....Continued)

Let's get Started

Introduction:

If you have 5+ years of experience in the field of software development and more specifically in Java, it is very much necessary to know all the concepts of Java in depth. The interviewer can ask you the trickiest of questions that might confuse you and even rattle your foundation of what you know about Java. You need to be well-prepared to know the concepts in depth and prove why you are a developer with 5+ years of experience. In this section, we will see what are the most commonly asked Core Java Interview questions and how you can answer them. We will cover a wide range of important Java topics that are the applications of variables, collections, polymorphism, strings, datatypes, and threads.

Java Interview Questions for Experienced

1. Differentiate between Volatile and Transient Variable in Java.

Volatile Variable	Transient Variable
The volatile keyword against the variable indicates that the content of the variable is stored in the main memory and every read of the variable should be done from the main memory and not the CPU cache and every write should be written to the main memory and not just to the CPU cache.	Transient is used when we do not want the variable to be serialised.
Volatile ensures that the JVM does not re-order the variables and ensures that the synchronization issues are avoided.	Transient provides flexibility and control over the attributes of objects from being serialized.
Volatile variables do not have any default values.	Transient variables are initialized with default value corresponding to the data type at the time of deserialization.
Volatile variables can be used along with the static keyword.	Transient variables cant be used along with static keywords because the static variables are class-level variables and not related to the individual instances. This matters during serialization.

2. Differentiate between the Vector and ArrayList collections in Java.

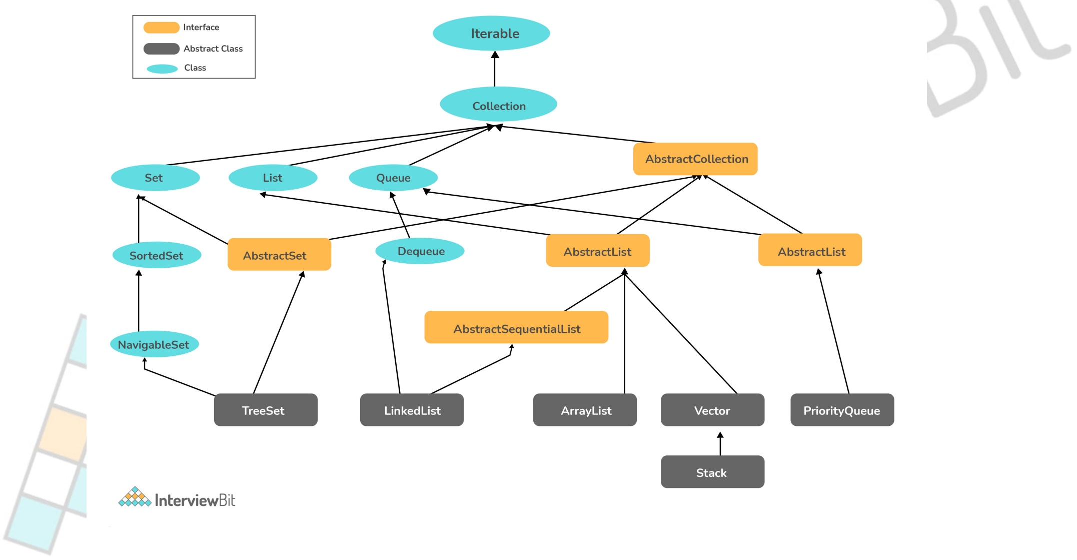
ArrayList and Vector collection classes both implement the List interface and are derived from AbstractList. Both these classes are index-based, meaning objects can be retrieved from the collection based on the index using the `get(index)` method.

Category	Vector	ArrayList
Synchronization	Vector is synchronized and thread-safe by default. This means that the internal state of the Vector is not compromised even if multiple threads are operating on it.	ArrayList is neither thread-safe nor synchronized.
Speed	Since Vector is synchronized, it works relatively slower than ArrayList.	ArrayList is faster than Vector because there is no overhead of maintaining synchronization.

3. How is Collection different from Collections in Java?

Collection is an interface whereas Collections is a java utility class containing only static methods that can operate on collections like ArrayList, Set, Queue, etc. and both belong to Collection Framework and are present in the `java.util` package.

Java.util.Collection Hierarchy



Since Collection is an interface dealing with the data type of iterable objects, the interface extends the Iterable interface as shown below:

```
public interface Collections<E>
extends Iterable<E>
```

Since Collections is a java class, it extends the Object class as shown below:

```
public class Collections
extends Object
```

4. What do you understand by the ... in the below method parameters?

```
public void someMethod(String... info){
    // method body
}
```

The 3 dots feature was started in Java 5 and the feature is known as varargs (variable arguments). This simply means that the method can receive one or more/multiple String arguments as shown below:

- `someMethod("Java", "Interview");`
- `someMethod("Java", "Interview", "Questions");`
- `someMethod(new String[]{"Java", "Interview", "Questions"});`

These received arguments can be used as an array and can be accessed by iterating through loops as shown below:

```
public void someMethod(String... info){
    for(String someInfo : info){
        // any operation
    }
    // The info can be accessed using index based loops too
    for( int i = 0; i < info.length; i++){
        String s = info[i];
        //some operation
    }
}
```

5. Can you write a code for representing thread-safe singleton patterns in Java?

A thread-safe singleton class is created which helps in object initialization in the presence of multiple threads. It can be done using multiple ways:

- **Using Enums:** Enums are the simplest means of creating a thread-safe singleton class in Java because the synchronization support is inherently done by Java itself. Enums are by default final and this also helps in preventing multiple initializations at the time of serialization.

```java
public enum ThreadSafeSingleton{
   SINGLETON_INSTANCE;
   public void display(){
       System.out.println("Thread-safe singleton Display");
   }
}
// The Singleton class methods can be invoked as below
ThreadSafeSingleton.SINGLETON_INSTANCE.show();
```

- **Using Static Field Initialization:** Thread-safe singleton can also be created by creating the instance at the time of class loading. This is achieved by making use of static fields as the Classloader guarantees that the instances are initialized during class loading and the instance is not visible until that has been fully created.

```java
public class ThreadSafeSingleton{
   private static final ThreadSafeSingleton INSTANCE = new ThreadSafeSingleton();
   private ThreadSafeSingleton(){ }
   public static ThreadSafeSingleton getInstance(){
       return INSTANCE;
   }
   public void display(){
       System.out.println("Thread-safe Singleon");
   }
}
ThreadSafeSingleton.getInstance().display();
```

But the disadvantage of this way is that the initialization cannot be done lazily and the getInstance() method is called even before any client can call.

- **Using synchronized keyword:** We can make use of the synchronized keyword upon the getInstance method as shown below:
    - In this method, we can achieve lazy initialization, and also since we use synchronized keywords, the object initialization is also thread-safe.
    - The only problem is that since the whole method is synchronized, the performance is impacted in the presence of multiple threads.

```
public class ThreadSafeSingleton
{
 // Creating private instance to make it accessible only by getInstance() method
 private static ThreadSafeSingleton instance;
 private ThreadSafeSingleton()
 {
   // Making constructor private so that objects cant be initialized outside the class
 }
 //synchronized getInstance method
 synchronized public static ThreadSafeSingleton getInstance(){
   if (this.instance == null)
   {
     // if instance is null, initialize
     this.instance = new ThreadSafeSingleton();
   }
   return this.instance;
 }
}
```

- **Double-check locking:** Here, we will be using a synchronized block of code within the getInstance method instead of making the whole method synchronized. This ensures that only a handful of threads have to wait only for the first time thereby not impacting the performance.

```
public class ThreadSafeSingleton {

  // Creating private instance to make it accessible only by getInstance() method
  private static ThreadSafeSingleton instance;
  private ThreadSafeSingleton(){
    // Making constructor private so that objects cant be initialized outside the class
  }

  public static ThreadSafeSingleton getInstance(){
    if (instance == null){
      //synchronized block of code
      synchronized (ThreadSafeSingleton.class){
        if(instance==null)
        {
          // initialize only if instance is null
          instance = new ThreadSafeSingleton();
        }

      }
    }
    return instance;
  }
}
```

6. What is the importance of the hashCode() and equals() contract?

Consider the scenario of the HashMap object. We know that the Key of the HashMap uses the hashCode() and equals() methods for finding index or finding the value of a given key by uniquely identify the key-value pair in the map. If these methods are not implemented properly, then there are chances where two keys would produce the same output for these methods resulting in inaccuracy and a wrong key's value might get updated at the time of updation. Hence, it is very much important to implement the hashCode and the equals method correctly. This can be done properly if we follow the hashCode-equals contract. The contract states that

> If two objects are equal, then the hashCode method should produce the same result for both objects.
> To ensure this, we have to override the hashCode() method whenever we override the equals() method.

7. How is the classpath variable different from the path variables?

The classpath variables are specific to the Java executables and are used for locating the class files. Whereas, the path variable is a variable present in the operating system and is used for locating the system executables.

8. What is the result of the below code and Why?

```
public class TestClass {
public static void main(String[] args) {
     someMethod(null);
}
public static void someMethod(Object o) {
 System.out.println("Object method Invoked");
}
public static void someMethod(String s) {
 System.out.println("String method Invoked");
}
}
```

The output of this code is "String method Invoked". We know that null is a value that can be assigned to any kind of object reference type in Java. It is not an object in Java. Secondly, the Java compiler chooses the method with the most specific parameters in method overloading. this means that since the String class is more specific, the method with String input parameter is called.

9. How would you help a colleague with lesser Java experience who has trouble in serializing a class?

I would first check if the colleague has implemented the java.io.serializable interface. If this is done, I will check if they are trying to serialize only non-static members since the static members cannot be serialized. I would also check if the fields that are not serialized are defined as transient accidentally.

10. What is the best possible way to call the wait() method - by using the if construct or the loop construct?

wait() should be called in loop construct always because whenever a thread gets resources to execute again, it is always better to check the condition before execution. The standard way of wait and notify usage is as shown below:

```
synchronized (resource) {
   while (wait condition)
     resource.wait(); // release the resource lock and reacquire it post wait
   // operations to perform
}
```

11. Can we use HashMap in a multi-threaded environment?

You can use the HashMap but the probability of working fine depends on the way we use it. For instance, consider the HashMap of configuration properties, if the HashMap initialization was done by using just one thread, and the remaining threads do the task of reading from the map, then HashMap would work perfectly well.

The problem arises when there is at least one thread that updates the Map by means of adding, updating, or deleting the map content. The put() method of the map resizes the map that can cause a deadlock or infinite loop while the threads operate. Hence, during such scenarios, we can use the HashTable or ConcurrentHashMap.

12. What is the result of the below code?

```
public class InterviewbitProblem{
public static void main(String[] arr){
   System.out.println(0.1*3 == 0.3);
   System.out.println(0.1*2 == 0.2);
}
}
```

The statements result in:

`System.out.println(0.1*3 == 0.3);` -> Prints false

`System.out.println(0.1*2 == 0.2);` -> Prints true

This expectation mismatch is due to the error that occurs while rounding float-point numbers and the fact that in Java, only the floating-point numbers that are powers of 2 are represented accurately by the binary representation. The rest of the numbers should be rounded to accommodate the limited bits as required.

13. What is the result of the below Java code?

```
public class InterviewbitProblem{
   public static void main(String[] args) {
       System.out.println(Math.min(Double.MIN_VALUE, 0.0d));
   }
}
```

We should know that in Java, for the type Double, the MIN_VALUE and MAX_VALUE are positive numbers. The Double.MIN_VALUE has the value of {"detectHand":false} which is a positive number but the least of all values belonging to the Double class.

- Hence the program results in printing 0.0 as obviously the Double.MIN_VALUE > 0

14. What will happen if you run 1.0/0.0?

The double class provides certain rules like Double.INFINITY, NaN, -0.0, etc which aids in arithmetic calculations. The given problem will return Double.INFINITY without throwing any Arithmetic Exception.

15. Is it possible to override a method to throw RuntimeException from throwing NullPointerException in the parent class?

Yes, this is possible. But it is not possible if the parent class has a checked Exception. This is due to the below rule of method overriding in cases of checked exceptions:

> The method that wants to override a parent class method can not throw a higher Exception than the overridden method.

This means that if the overridden method is throwing IOException, then the overriding child class method can only throw IOException or its sub-classes. This overriding method can not throw a higher Exception than the original or overridden method.

16. Is there any difference in defining or creating a String by using String literal and by using the new() operator?

Creating string using the new operator ensures that the String is created in the heap alone and not into the string pool. Whereas, creating string using literal ensures that the string is created in the string pool. String pool exists as part of the perm area in the heap. This ensures that the multiple Strings created using literal having same values are pointed to one object and prevents duplicate objects with the same value from being created.

17. What is the output of the below code?

```java
public class InterviewbitProblem{
   public static void main(String[] args) {
       Integer num1 = 1000, num2 = 1000;
       System.out.println(num1 == num2);//1
       Integer num3 = 20, num4 = 20;
       System.out.println(num3 == num4);//2
   }
}
```

Line 1 results in false, whereas line 2 results in true. Let us see why.
In Java, if the references point to different objects and have the same content, they are not equal in terms of using double equals. By this logic, the statement `num3==num4` should have resulted in false. But, the `Integer.java` class in Java has a private inner class called `IntegerCache.java` which helps to cache the Integer objects ranging from `-128 to 127`. All the numbers lying between this range are cached internally by the Integer class. While defining an object as:

```java
Integer num3 = 20;
```

Internally, it is converted to Integer by using:

```
Integer num = Integer.valueOf(20);
```

The valueOf method from the Integer class is defined like:

```
public static Integer valueOf(int i) {
 if (i >= IntegerCache.low && i<=IntegerCache.high)
     return IntegerCache.cache[i + (-IntegerCache.low)];
 return new Integer(i);
}
```

This means that if the value lies in the range -128 to 127, then it returns the Integer instance from the cache, else it creates a new instance. This means that the num3 and num4 point to the same object in the IntegerCache and thereby the comparison results in true.

18. What is the result of the below program?

```
class X {
   static int i = 1111;

   static{
       i = i-- - --i;    //L1
   }

   {
       i = i++ + ++i;    //L2
   }
}

class Y extends X{
   static{
       i = --i - i--;    //L3
   }
   {
       i = ++i + i++;    //L4
   }
}

public class DriverClass{
   public static void main(String[] args){
       Y y = new Y();
       System.out.println(y.i);    //L5
   }
}
```

Let us evaluate the expressions one by one:

L1 ->

```
i = i-- - --i;
i = 1111 - 1109 = 2
```

L2 ->

```
i = i++ + ++i;
i = 0 + 2 = 2
```

L3 ->

```
i = --i - i--;
i = 1 - 1 = 0
```

L4 ->

```
i = ++i + i++;
i = 3 + 3 = 6 = y.i
```

L5 ->
y.i = i from class Y which is 6.
Hence the output is 6.

19. What is the output of the below code and why?

```
public class InterviewbitTest {
   private static int counter = 0;
   void InterviewbitTest() {
       counter = 20;
   }
   InterviewbitTest(int x){
       counter = x;
   }
   public static void main(String[] args) {
       InterviewbitTest interviewbitTest = new InterviewbitTest();
       System.out.println(counter);
   }
}
```

The output of the code is “Compile Error” This is because the code is not able to find any constructor that matches InterviewbitTest().

20. Is it necessary to declare all immutable objects as final?

This is not necessary. The functionality of achieving immutability can also be achieved by defining the members of a class as private and not providing any setter methods to modify/update the values. Any references to the members should not be leaked and the only source of initializing the members should be by using the parameterized constructor.

We should note that the variables declared as final only takes care of not re-assigning the variable to a different value. The individual properties of an object can still be changed.

21. What do you know about Factory Design Pattern in Java?

Factory design pattern is the most commonly used pattern in Java. They belong to Creational Patterns because it provides means of creating an object of different instances. Here, the logic of object creation is not exposed to the client. It is implemented by using a standard interface. It gives provision to select object types that we need for creation. This is why, they are also known as Virtual Constructors. For more information regarding the implementation, you can refer here.

Resources

22. Conclusion

In this section, we have seen that a lot of questions have been asked on polymorphism (Method overloading & overriding), inheritance, collections, strings, and threads. One has to know the concepts and the working of each of these in-depth to ensure you ace the Java Interview.

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions