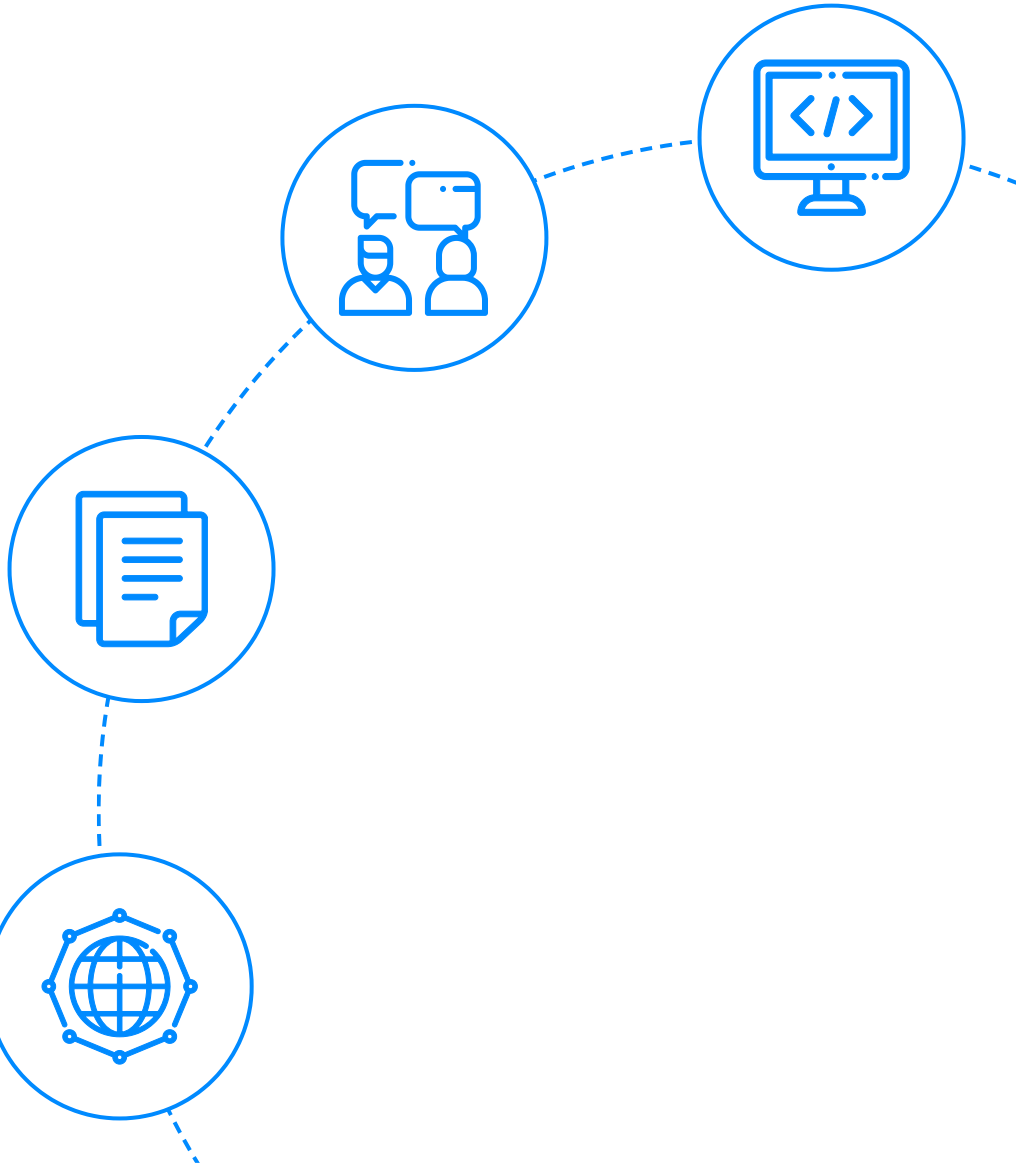
InterviewBit

Java Interview Questions



To view the live version of the page, click here.

Contents

Java Basic Interview Questions

Java Intermediate Interview Questions

Java Intermediate Interview Questions (.....Continued)

19. Apart from the security aspect, what are the reasons behind making strings immutable in Java?
20. How would you differentiate between a String, StringBuffer, and a StringBuilder?
21. Using relevant properties highlight the differences between interfaces and abstract classes.
22. In Java, static as well as private method overriding is possible. Comment on the statement.
23. What makes a HashSet different from a TreeSet?
24. Why is the character array preferred over string for storing confidential information?
25. What are the differences between JVM, JRE and JDK in Java?
26. What are the differences between HashMap and HashTable in Java?
27. What is the importance of reflection in Java?
28. What are the different ways of threads usage?
29. What are the differences between constructor and method of a class in Java?
30. Java works as “pass by value” or “pass by reference” phenomenon?
31. Which among String or String Buffer should be preferred when there are lot of updates required to be done in the data?
32. How to not allow serialization of attributes of a class in Java?
33. What happens if the static modifier is not included in the main method signature in Java?
34. What happens if there are multiple main methods inside one class in Java?
35. What do you understand by Object Cloning and how do you achieve it in Java?
36. How does an exception propagate in the code?
37. Is it mandatory for a catch block to be followed after a try block?
38. Will the finally block get executed when the return statement is written at the end of try block and catch block as shown below?

Java Intermediate Interview Questions (.....Continued)

39. Can you call a constructor of a class inside the another constructor?
40. Contiguous memory locations are usually used for storing actual values in an array but not in ArrayList. Explain.

Java Advanced Interview Questions

41. Although inheritance is a popular OOPs concept, it is less advantageous than composition. Explain.
42. How is the creation of a String using new() different from that of a literal?
43. Is exceeding the memory limit possible in a program despite having a garbage collector?
44. Why is synchronization necessary? Explain with the help of a relevant example.
45. In the given code below, what is the significance of ... ?
46. Can you explain the Java thread lifecycle?
47. What could be the tradeoff between the usage of an unordered array versus the usage of an ordered array?
48. Is it possible to import the same class or package twice in Java and what happens to it during runtime?
49. In case a package has sub packages, will it suffice to import only the main package? e.g. Does importing of com.myMainPackage.* also import com.myMainPackage.mySubPackage.*?
50. Will the finally block be executed if the code System.exit(0) is written at the end of try block?
51. What do you understand by marker interfaces in Java?
52. Explain the term “Double Brace Initialisation” in Java?
53. Why is it said that the length() method of String class doesn't return accurate results?
54. What is the output of the below code and why?
55. What are the possible ways of making object eligible for garbage collection (GC) in Java?

Java Interview Programs

(.....Continued)

56. Check if a given string is palindrome using recursion.
57. Write a Java program to check if the two strings are anagrams.
58. Write a Java Program to find the factorial of a given number.
59. Given an array of non-duplicating numbers from 1 to n where one number is missing, write an efficient java program to find that missing number.
60. Write a Java Program to check if any number is a magic number or not. A number is said to be a magic number if after doing sum of digits in each step and inturn doing sum of digits of that sum, the ultimate result (when there is only one digit left) is 1.

Conclusion

61. Conclusion

Let's get Started

Do you have what it takes to ace a Java Interview? We are here to help you in consolidating your knowledge and concepts in Java. The following article will cover all the popular Java interview questions for freshers as well as experienced candidates in depth.

Go through all the questions to enhance your chances of performing well in the interviews. The questions will revolve around the basic and core fundamentals of Java.

So, let's dive deep into the plethora of useful interview questions on Java.

Java Basic Interview Questions

1. Why is Java a platform independent language?

Java language was developed in such a way that it does not depend on any hardware or software due to the fact that the compiler compiles the code and then converts it to platform-independent byte code which can be run on multiple systems.

- The only condition to run that byte code is for the machine to have a runtime environment (JRE) installed in it.

2. Why is Java not a pure object oriented language?

Java supports primitive data types - byte, boolean, char, short, int, float, long, and double and hence it is not a pure object-oriented language.

3. Pointers are used in C/ C++. Why does Java not make use of pointers?

Pointers are quite complicated and unsafe to use by beginner programmers. Java focuses on code simplicity, and the usage of pointers can make it challenging. Pointer utilization can also cause potential errors. Moreover, security is also compromised if pointers are used because the users can directly access memory with the help of pointers.

Thus, a certain level of abstraction is furnished by not including pointers in Java. Moreover, the usage of pointers can make the procedure of garbage collection quite slow and erroneous. Java makes use of references as these cannot be manipulated, unlike pointers.

4. What do you understand by an instance variable and a local variable?

Instance variables are those variables that are accessible by all the methods in the class. They are declared outside the methods and inside the class. These variables describe the properties of an object and remain bound to it at any cost.

All the objects of the class will have their copy of the variables for utilization. If any modification is done on these variables, then only that instance will be impacted by it, and all other class instances continue to remain unaffected.

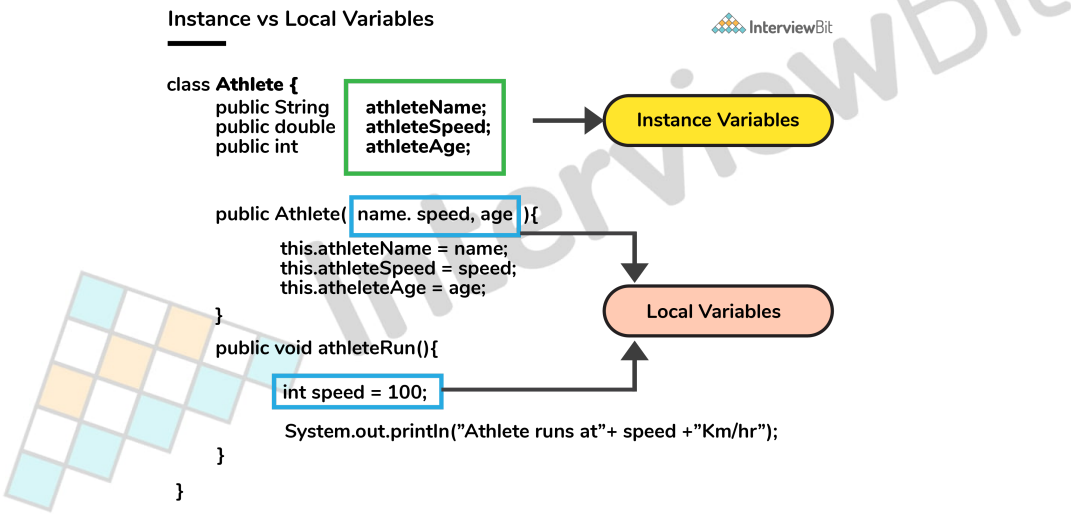
Example:

```
class Athlete {
public String athleteName;
public double athleteSpeed;
public int athleteAge;
}
```

Local variables are those variables present within a block, function, or constructor and can be accessed only inside them. The utilization of the variable is restricted to the block scope. Whenever a local variable is declared inside a method, the other class methods don’t have any knowledge about the local variable.

Example:

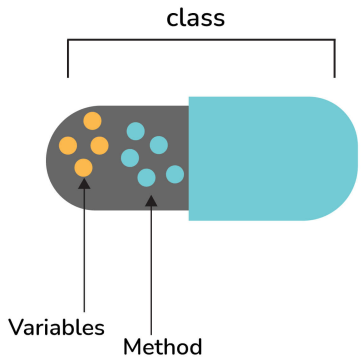
```
public void athlete() {
String athleteName;
double athleteSpeed;
int athleteAge;
}
```



5. What do you mean by data encapsulation?

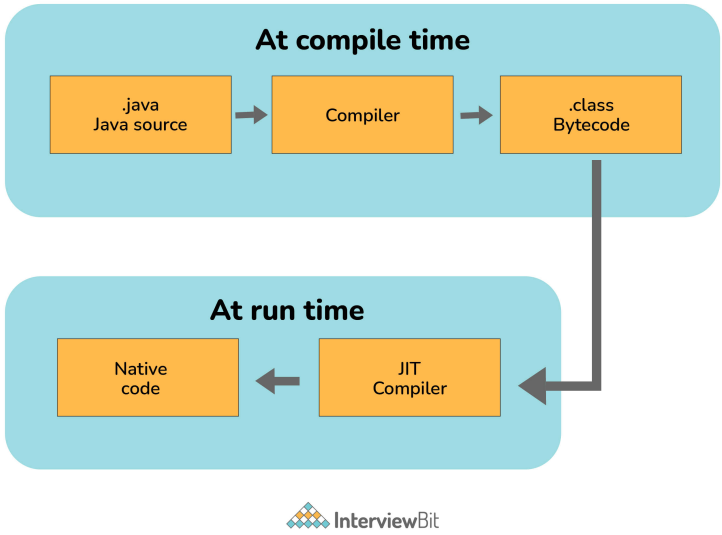
- Data Encapsulation is an Object-Oriented Programming concept of hiding the data attributes and their behaviors in a single unit.
- It helps developers to follow modularity while developing software by ensuring that each object is independent of other objects by having its own methods, attributes, and functionalities.
- It is used for the security of the private properties of an object and hence serves the purpose of data hiding.

```
class
{

  data members (properties)
          +
   methods (behavior)

}
```



6. Tell us something about JIT compiler.

- JIT stands for Just-In-Time and it is used for improving the performance during run time. It does the task of compiling parts of byte code having similar functionality at the same time thereby reducing the amount of compilation time for the code to run.
- The compiler is nothing but a translator of source code to machine-executable code. But what is special about the JIT compiler? Let us see how it works:
    - First, the Java source code (.java) conversion to byte code (.class) occurs with the help of the javac compiler.
    - Then, the .class files are loaded at run time by JVM and with the help of an interpreter, these are converted to machine understandable code.
    - JIT compiler is a part of JVM. When the JIT compiler is enabled, the JVM analyzes the method calls in the .class files and compiles them to get more efficient and native code. It also ensures that the prioritized method calls are optimized.
    - Once the above step is done, the JVM executes the optimized code directly instead of interpreting the code again. This increases the performance and speed of the execution.



7. Can you tell the difference between equals() method and equality operator (==) in Java?

equals()	==
This is a method defined in the Object class.	It is a binary operator in Java.
This method is used for checking the equality of contents between two objects as per the specified business logic.	This operator is used for comparing addresses (or references), i.e checks if both the objects are pointing to the same memory location.

Note:

- In the cases where the equals method is not overridden in a class, then the class uses the default implementation of the equals method that is closest to the parent class.
- Object class is considered as the parent class of all the java classes. The implementation of the equals method in the Object class uses the == operator to compare two objects. This default implementation can be overridden as per the business logic.

8. How is an infinite loop declared in Java?

Infinite loops are those loops that run infinitely without any breaking conditions. Some examples of consciously declaring infinite loop is:

- Using For Loop:

```
for (;;)
{
   // Business logic
   // Any break logic
}
```

- Using while loop:

```
while(true){
   // Business logic
   // Any break logic
}
```

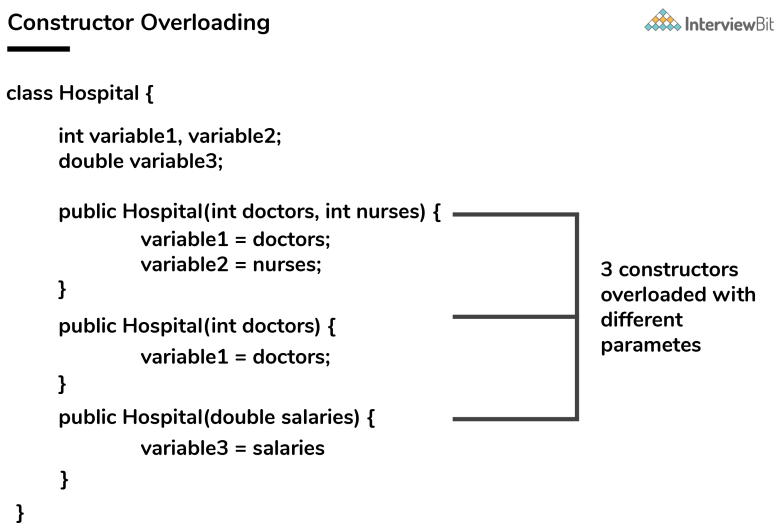
- Using do-while loop:

```
do{
   // Business logic
   // Any break logic
}while(true);
```

9. Briefly explain the concept of constructor overloading

Constructor overloading is the process of creating multiple constructors in the class consisting of the same name with a difference in the constructor parameters. Depending upon the number of parameters and their corresponding types, distinguishing of the different types of constructors is done by the compiler.

```
class Hospital {
int variable1, variable2;
double variable3;
public Hospital(int doctors, int nurses) {
 variable1 = doctors;
 variable2 = nurses;
}
public Hospital(int doctors) {
 variable1 = doctors;
}
public Hospital(double salaries) {
 variable3 = salaries
}
}
```



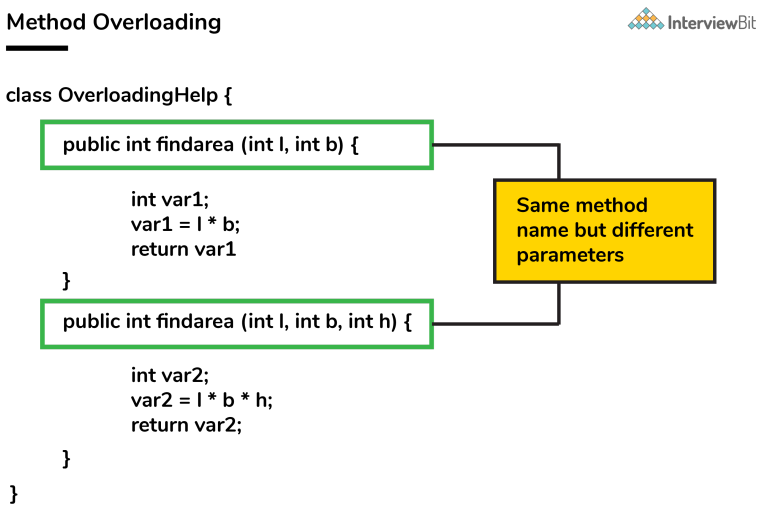
Three constructors are defined here but they differ on the basis of parameter type and their numbers.

10. Comment on method overloading and overriding by citing relevant examples.

In Java, **method overloading** is made possible by introducing different methods in the same class consisting of the same name. Still, all the functions differ in the number or type of parameters. It takes place inside a class and enhances program readability.

The only difference in the return type of the method does not promote method overloading. The following example will furnish you with a clear picture of it.

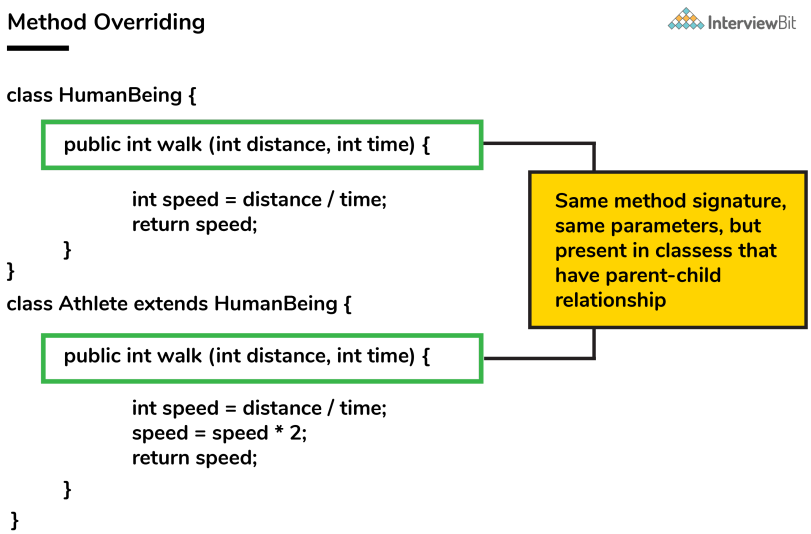
```
class OverloadingHelp {
   public int findarea (int l, int b) {
           int var1;
           var1 = l * b;
           return var1;
   }
   public int findarea (int l, int b, int h) {
           int var2;
           var2 = l * b * h;
           return var2;
   }
}
```



Both the functions have the same name but differ in the number of arguments. The first method calculates the area of the rectangle, whereas the second method calculates the area of a cuboid.

Method overriding is the concept in which two methods having the same method signature are present in two different classes in which an inheritance relationship is present. A particular method implementation (already present in the base class) is possible for the derived class by using method overriding.
Let's give a look at this example:

```
class HumanBeing {
	public int walk (int distance, int time) {
		int speed = distance / time;
		return speed;
	}
}
class Athlete extends HumanBeing {
	public int walk(int distance, int time) {
		int speed = distance / time;
		speed = speed * 2;
		return speed;
	}
}
```



Both class methods have the name walk and the same parameters, distance, and time. If the derived class method is called, then the base class method walk gets overridden by that of the derived class.

11. A single try block and multiple catch blocks can co-exist in a Java Program. Explain.

Yes, multiple catch blocks can exist but specific approaches should come prior to the general approach because only the first catch block satisfying the catch condition is executed. The given code illustrates the same:

```
public class MultipleCatch {
public static void main(String args[]) {
 try {
  int n = 1000, x = 0;
  int arr[] = new int[n];
  for (int i = 0; i <= n; i++) {
   arr[i] = i / x;
  }
 }
 catch (ArrayIndexOutOfBoundsException exception) {
  System.out.println("1st block = ArrayIndexOutOfBoundsException");
 }
 catch (ArithmeticException exception) {
  System.out.println("2nd block = ArithmeticException");
 }
 catch (Exception exception) {
  System.out.println("3rd block = Exception");
 }
}
}
```

Here, the second catch block will be executed because of division by 0 (i / x). In case x was greater than 0 then the first catch block will execute because for loop runs till i = n and array index are till n-1.

12. Explain the use of final keyword in variable, method and class.

In Java, the final keyword is used as defining something as constant /final and represents the non-access modifier.

- **final variable:**
  - When a variable is declared as final in Java, the value can't be modified once it has been assigned.
  - If any value has not been assigned to that variable, then it can be assigned only by the constructor of the class.
- **final method:**
  - A method declared as final cannot be overridden by its children's classes.
  - A constructor cannot be marked as final because whenever a class is inherited, the constructors are not inherited. Hence, marking it final doesn't make sense. Java throws compilation error saying - `modifier final not allowed here`
- **final class:**
  - No classes can be inherited from the class declared as final. But that final class can extend other classes for its usage.

13. Do final, finally and finalize keywords have the same function?

All three keywords have their own utility while programming.

Final: If any restriction is required for classes, variables, or methods, the final keyword comes in handy. Inheritance of a final class and overriding of a final method is restricted by the use of the final keyword. The variable value becomes fixed after incorporating the final keyword. Example:

```
final int a=100;
a = 0;  // error
```

The second statement will throw an error.

Finally: It is the block present in a program where all the codes written inside it get executed irrespective of handling of exceptions. Example:

```
try {
int variable = 5;
}
catch (Exception exception) {
System.out.println("Exception occurred");
}
finally {
System.out.println("Execution of finally block");
}
```

Finalize: Prior to the garbage collection of an object, the finalize method is called so that the clean-up activity is implemented. Example:

```
public static void main(String[] args) {
String example = new String("InterviewBit");
example = null;
System.gc(); // Garbage collector called
}
public void finalize() {
// Finalize called
}
```

14. When can you use super keyword?

- The super keyword is used to access hidden fields and overridden methods or attributes of the parent class.
- Following are the cases when this keyword can be used:
  - Accessing data members of parent class when the member names of the class and its child subclasses are same.
  - To call the default and parameterized constructor of the parent class inside the child class.
  - Accessing the parent class methods when the child classes have overridden them.
- The following example demonstrates all 3 cases when a super keyword is used.

```
public class Parent{
        private int num = 1;

        Parent(){
            System.out.println("Parent class default constructor.");
        }

        Parent(String x){
            System.out.println("Parent class parameterised constructor.");
        }

        public void foo(){
            System.out.println("Parent class foo!");
        }
    }

    public class Child extends Parent{
        private int num = 2;

        Child(){
            System.out.println("Child class default Constructor");

            super();    // to call default parent constructor
            super("Call Parent");    // to call parameterised constructor.
        }

        void printNum(){
            System.out.println(num);
            System.out.println(super.num); //prints the value of num of parent class
        }

        @Override
        public void foo(){
            System.out.println("Parent class foo!");
            super.foo();    //Calls foo method of Parent class inside the Overriden foo
        }
    }
```

15. Can the static methods be overloaded?

Yes! There can be two or more static methods in a class with the same name but differing input parameters.

16. Can the static methods be overridden?

- No! Declaration of static methods having the same signature can be done in the subclass but run time polymorphism can not take place in such cases.
- Overriding or dynamic polymorphism occurs during the runtime, but the static methods are loaded and looked up at the compile time statically. Hence, these methods cant be overridden.

17. What is the main objective of garbage collection?

The main objective of this process is to free up the memory space occupied by the unnecessary and unreachable objects during the Java program execution by deleting those unreachable objects.

- This ensures that the memory resource is used efficiently, but it provides no guarantee that there would be sufficient memory for the program execution.

18. What part of memory - Stack or Heap - is cleaned in garbage collection process?

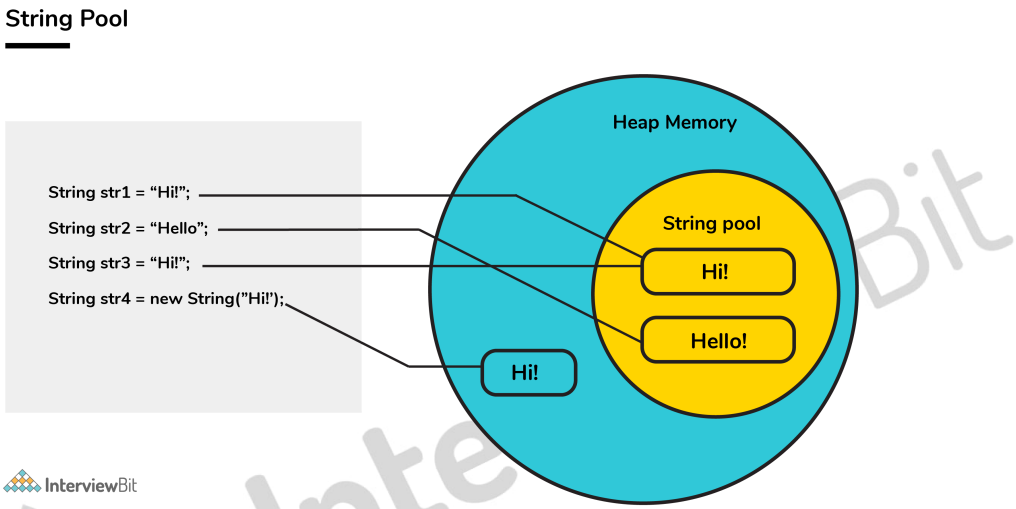
Heap.

Java Intermediate Interview Questions

19. Apart from the security aspect, what are the reasons behind making strings immutable in Java?

A String is made immutable due to the following reasons:

- **String Pool:** Designers of Java were aware of the fact that String data type is going to be majorly used by the programmers and developers. Thus, they wanted optimization from the beginning. They came up with the notion of using the String pool (a storage area in Java heap) to store the String literals. They intended to decrease the temporary String object with the help of sharing. An immutable class is needed to facilitate sharing. The sharing of the mutable structures between two unknown parties is not possible. Thus, immutable Java String helps in executing the concept of String Pool.



- **Multithreading:** The safety of threads regarding the String objects is an important aspect in Java. No external synchronization is required if the String objects are immutable. Thus, a cleaner code can be written for sharing the String objects across different threads. The complex process of concurrency is facilitated by this method.
- **Collections:** In the case of Hashtables and HashMaps, keys are String objects. If the String objects are not immutable, then it can get modified during the period when it resides in the HashMaps. Consequently, the retrieval of the desired data is not possible. Such changing states pose a lot of risks. Therefore, it is quite safe to make the string immutable.

20. How would you differentiate between a String, StringBuffer, and a StringBuilder?

- **Storage area:** In string, the String pool serves as the storage area. For StringBuilder and StringBuffer, heap memory is the storage area.
- **Mutability:** A String is immutable, whereas both the StringBuilder and StringBuffer are mutable.
- **Efficiency:** It is quite slow to work with a String. However, StringBuilder is the fastest in performing operations. The speed of a StringBuffer is more than a String and less than a StringBuilder. (For example appending a character is fastest in StringBuilder and very slow in String because a new memory is required for the new String with appended character.)
- **Thread-safe:** In the case of a threaded environment, StringBuilder and StringBuffer are used whereas a String is not used. However, StringBuilder is suitable for an environment with a single thread, and a StringBuffer is suitable for multiple threads.
  Syntax:

```
// String
String first = "InterviewBit";
String second = new String("InterviewBit");
// StringBuffer
StringBuffer third = new StringBuffer("InterviewBit");
// StringBuilder
StringBuilder fourth = new StringBuilder("InterviewBit");
```

21. Using relevant properties highlight the differences between interfaces and abstract classes.

- **Availability of methods:** Only abstract methods are available in interfaces, whereas non-abstract methods can be present along with abstract methods in abstract classes.
- **Variable types**: Static and final variables can only be declared in the case of interfaces, whereas abstract classes can also have non-static and non-final variables.
- **Inheritance:** Multiple inheritances are facilitated by interfaces, whereas abstract classes do not promote multiple inheritances.
- **Data member accessibility:** By default, the class data members of interfaces are of the public- type. Conversely, the class members for an abstract class can be protected or private also.
- **Implementation:** With the help of an abstract class, the implementation of an interface is easily possible. However, the converse is not true;

Abstract class example:

```
public abstract class Athlete {
public abstract void walk();
}
```

Interface example:

```
public interface Walkable {
void walk();
}
```

22. In Java, static as well as private method overriding is possible. Comment on the statement.

The statement in the context is completely False. The static methods have no relevance with the objects, and these methods are of the class level. In the case of a child class, a static method with a method signature exactly like that of the parent class can exist without even throwing any compilation error.

The phenomenon mentioned here is popularly known as method hiding, and overriding is certainly not possible. Private method overriding is unimaginable because the visibility of the private method is restricted to the parent class only. As a result, only hiding can be facilitated and not overriding.

23. What makes a HashSet different from a TreeSet?

Although both HashSet and TreeSet are not synchronized and ensure that duplicates are not present, there are certain properties that distinguish a HashSet from a TreeSet.

- **Implementation:** For a HashSet, the hash table is utilized for storing the elements in an unordered manner. However, TreeSet makes use of the red-black tree to store the elements in a sorted manner.
- **Complexity/ Performance:** For adding, retrieving, and deleting elements, the time amortized complexity is O(1) for a HashSet. The time complexity for performing the same operations is a bit higher for TreeSet and is equal to O(log n). Overall, the performance of HashSet is faster in comparison to TreeSet.
- **Methods:** hashCode() and equals() are the methods utilized by HashSet for making comparisons between the objects. Conversely, compareTo() and compare() methods are utilized by TreeSet to facilitate object comparisons.
- **Objects type:** Heterogeneous and null objects can be stored with the help of HashSet. In the case of a TreeSet, runtime exception occurs while inserting heterogeneous objects or null objects.

24. Why is the character array preferred over string for storing confidential information?

In Java, a string is basically immutable i.e. it cannot be modified. After its declaration, it continues to stay in the string pool as long as it is not removed in the form of garbage. In other words, a string resides in the heap section of the memory for an unregulated and unspecified time interval after string value processing is executed.

As a result, vital information can be stolen for pursuing harmful activities by hackers if a memory dump is illegally accessed by them. Such risks can be eliminated by using mutable objects or structures like character arrays for storing any variable. After the work of the character array variable is done, the variable can be configured to blank at the same instant. Consequently, it helps in saving heap memory and also gives no chance to the hackers to extract vital data.

25. What are the differences between JVM, JRE and JDK in Java?

Criteria	JDK	JRE	JVM
Abbreviation	Java Development Kit	Java Runtime Environment	Java Virtual Machine
Definition	JDK is a complete software development kit for developing Java applications. It comprises JRE, JavaDoc, compiler, debuggers, etc.	JRE is a software package providing Java class libraries, JVM and all the required components to run the Java applications.	JVM is a platform-dependent, abstract machine comprising of 3 specifications - document describing the JVM implementatio requirements, computer program meeting the JV requirements and instance object for executing the Java byte code and provide the runtime environment fo execution.
Main Purpose	JDK is mainly used for code development and execution.	JRE is mainly used for environment creation to execute the	JVM provides specifications f all the implementatio to JRE.

26. What are the differences between HashMap and HashTable in Java?

HashMap	HashTable
HashMap is not synchronized thereby making it better for non-threaded applications.	HashTable is synchronized and hence it is suitable for threaded applications.
Allows only one null key but any number of null in the values.	This does not allow null in both keys or values.
Supports order of insertion by making use of its subclass LinkedHashMap.	Order of insertion is not guaranteed in HashTable.

27. What is the importance of reflection in Java?

- The term `reflection` is used for describing the inspection capability of a code on other code either of itself or of its system and modify it during runtime.
- Consider an example where we have an object of unknown type and we have a method 'fooBar()' which we need to call on the object. The static typing system of Java doesn't allow this method invocation unless the type of the object is known beforehand. This can be achieved using reflection which allows the code to scan the object and identify if it has any method called "fooBar()" and only then call the method if needed.

```
Method methodOfFoo = fooObject.getClass().getMethod("fooBar", null);
methodOfFoo.invoke(fooObject, null);
```

- Using reflection has its own cons:
  - Speed — Method invocations due to reflection are about three times slower than the direct method calls.
  - Type safety — When a method is invoked via its reference wrongly using reflection, invocation fails at runtime as it is not detected at compile/load time.
  - Traceability — Whenever a reflective method fails, it is very difficult to find the root cause of this failure due to a huge stack trace. One has to deep dive into the invoke() and proxy() method logs to identify the root cause.
- Hence, it is advisable to follow solutions that don't involve reflection and use this method as a last resort.

28. What are the different ways of threads usage?

- We can define and implement a thread in java using two ways:
  - **Extending the Thread class**

```
class InterviewBitThreadExample extends Thread{
    public void run(){
        System.out.println("Thread runs...");
    }
    public static void main(String args[]){
        InterviewBitThreadExample ib = new InterviewBitThreadExample();
        ib.start();
    }
}
```

- **Implementing the Runnable interface**

```
class InterviewBitThreadExample implements Runnable{
    public void run(){
        System.out.println("Thread runs...");
    }
    public static void main(String args[]){
        Thread ib = new Thread(new InterviewBitThreadExample());
        ib.start();
    }
}
```

- Implementing a thread using the method of Runnable interface is more preferred and advantageous as Java does not have support for multiple inheritances of classes.
- `start()` method is used for creating a separate call stack for the thread execution. Once the call stack is created, JVM calls the `run()` method for executing the thread in that call stack.

29. What are the differences between constructor and method of a class in Java?

Constructor	Method
Constructor is used for initializing the object state.	Method is used for exposing the object's behavior.
Constructor has no return type.	Method should have a return type. Even if it does not return anything, return type is void.
Constructor gets invoked implicitly.	Method has to be invoked on the object explicitly.
If the constructor is not defined, then a default constructor is provided by the java compiler.	If a method is not defined, then the compiler does not provide it.
The constructor name should be equal to the class name.	The name of the method can have any name or have a class name too.
A constructor cannot be marked as final because whenever a class is inherited, the constructors are not inherited. Hence, marking it final doesn't make sense. Java throws compilation error saying - `modifier final not allowed here`	A method can be defined as final but it cannot be overridden in its subclasses.

30. Java works as “pass by value” or “pass by reference” phenomenon?

Java always works as a “pass by value”. There is nothing called a “pass by reference” in Java. However, when the object is passed in any method, the address of the value is passed due to the nature of object handling in Java. When an object is passed, a copy of the reference is created by Java and that is passed to the method. The objects point to the same memory location. 2 cases might happen inside the method:

- **Case 1:** When the object is pointed to another location: In this case, the changes made to that object do not get reflected the original object before it was passed to the method as the reference points to another location.

For example:

```
class InterviewBitTest{
   int num;
   InterviewBitTest(int x){
       num = x;
   }
   InterviewBitTest(){
       num = 0;
   }
}
class Driver {
   public static void main(String[] args)
   {
       //create a reference
       InterviewBitTest ibTestObj = new InterviewBitTest(20);
       //Pass the reference to updateObject Method
       updateObject(ibTestObj);
       //After the updateObject is executed, check for the value of num in the object.
       System.out.println(ibTestObj.num);
   }
   public static void updateObject(InterviewBitTest ibObj)
   {
       // Point the object to new reference
       ibObj = new InterviewBitTest();
       // Update the value
       ibObj.num = 50;
   }
}
Output:
20
```

- **Case 2:** When object references are not modified: In this case, since we have the copy of reference the main object pointing to the same memory location, any changes in the content of the object get reflected in the original object.

For example:

```
class InterviewBitTest{
   int num;
   InterviewBitTest(int x){
       num = x;
   }
   InterviewBitTest(){
       num = 0;
   }
}
class Driver{
   public static void main(String[] args)
   {
       //create a reference
       InterviewBitTest ibTestObj = new InterviewBitTest(20);
       //Pass the reference to updateObject Method
       updateObject(ibTestObj);
       //After the updateObject is executed, check for the value of num in the object.
       System.out.println(ibTestObj.num);
   }
   public static void updateObject(InterviewBitTest ibObj)
   {
       // no changes are made to point the ibObj to new location
       // Update the value of num
       ibObj.num = 50;
   }
}
Output:
50
```

31. Which among String or String Buffer should be preferred when there are lot of updates required to be done in the data?

StringBuffer is mutable and dynamic in nature whereas String is immutable. Every updation / modification of String creates a new String thereby overloading the string pool with unnecessary objects. Hence, in the cases of a lot of updates, it is always preferred to use StringBuffer as it will reduce the overhead of the creation of multiple String objects in the string pool.

32. How to not allow serialization of attributes of a class in Java?

- In order to achieve this, the attribute can be declared along with the usage of `transient` keyword as shown below:

```
public class InterviewBitExample {

   private transient String someInfo;
   private String name;
   private int id;
   // :
   // Getters setters
   // :
}
```

- In the above example, all the fields except `someInfo` can be serialized.

33. What happens if the static modifier is not included in the main method signature in Java?

There wouldn't be any compilation error. But then the program is run, since the JVM cant map the main method signature, the code throws “NoSuchMethodError” error at the runtime.

34. What happens if there are multiple main methods inside one class in Java?

The program can't compile as the compiler says that the method has been already defined inside the class.

35. What do you understand by Object Cloning and how do you achieve it in Java?

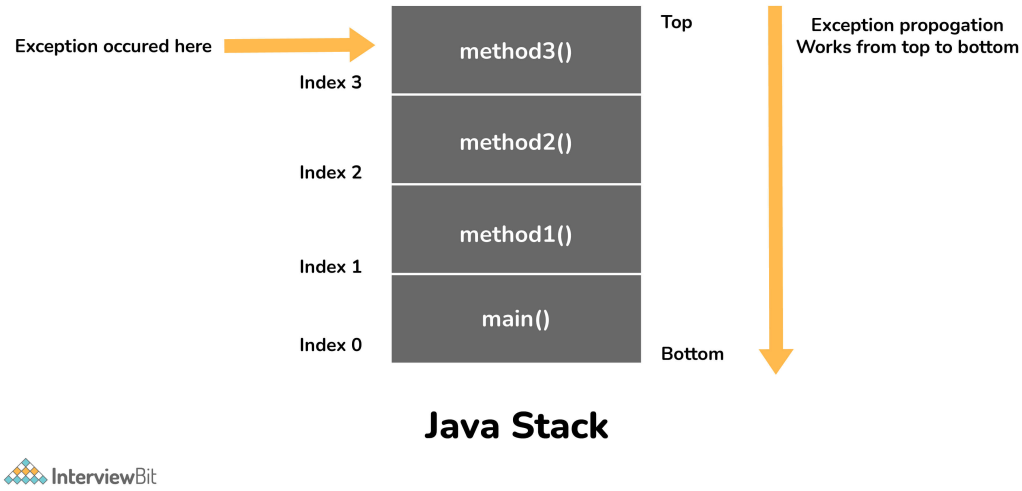
- It is the process of creating an exact copy of any object. In order to support this, a java class has to implement the Cloneable interface of java.lang package and override the clone() method provided by the Object class the syntax of which is:

```
protected Object clone() throws CloneNotSupportedException{
 return (Object)super.clone();
}
```

- In case the Cloneable interface is not implemented and just the method is overridden, it results in CloneNotSupportedException in Java.

36. How does an exception propagate in the code?

When an exception occurs, first it searches to locate the matching catch block. In case, the matching catch block is located, then that block would be executed. Else, the exception propagates through the method call stack and goes into the caller method where the process of matching the catch block is performed. This propagation happens until the matching catch block is found. If the match is not found, then the program gets terminated in the main method.



Java Stack

37. Is it mandatory for a catch block to be followed after a try block?

No, it is not necessary for a catch block to be present after a try block. - A try block should be followed either by a catch block or by a finally block. If the exceptions likelihood is more, then they should be declared using the throws clause of the method.

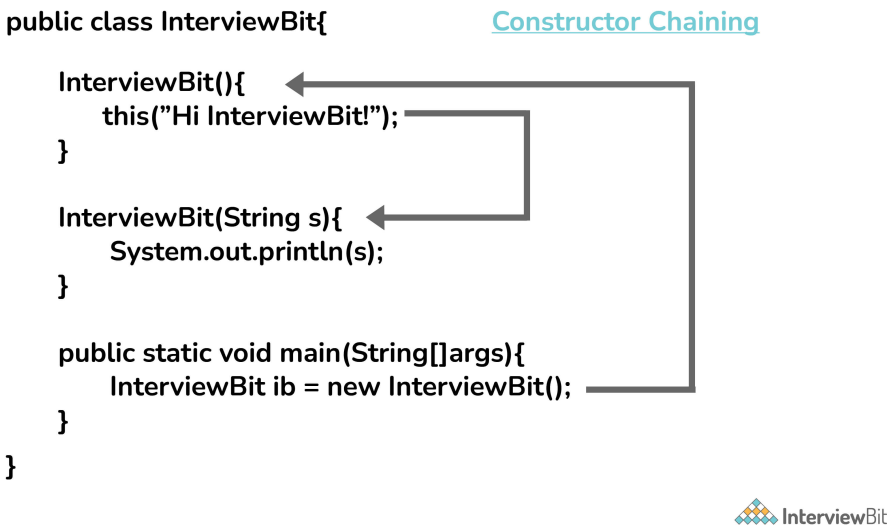
38. Will the finally block get executed when the return statement is written at the end of try block and catch block as shown below?

```
public int someMethod(int i){
   try{
       //some statement
       return 1;
   }catch(Exception e){
       //some statement
       return 999;
   }finally{
       //finally block statements
   }
}
```

finally block will be executed irrespective of the exception or not. The only case where finally block is not executed is when it encounters 'System.exit()' method anywhere in try/catch block.

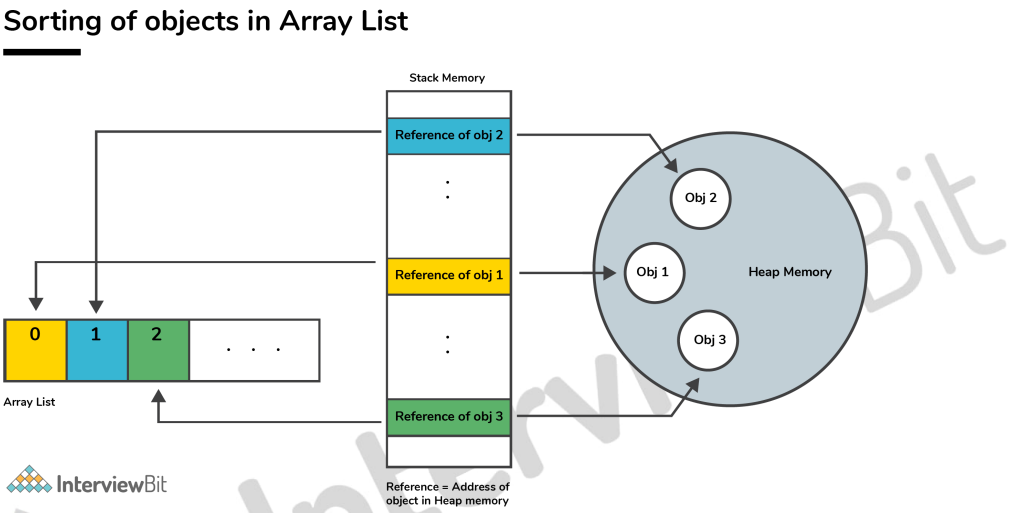
39. Can you call a constructor of a class inside the another constructor?

Yes, the concept can be termed as constructor chaining and can be achieved using `this()` .

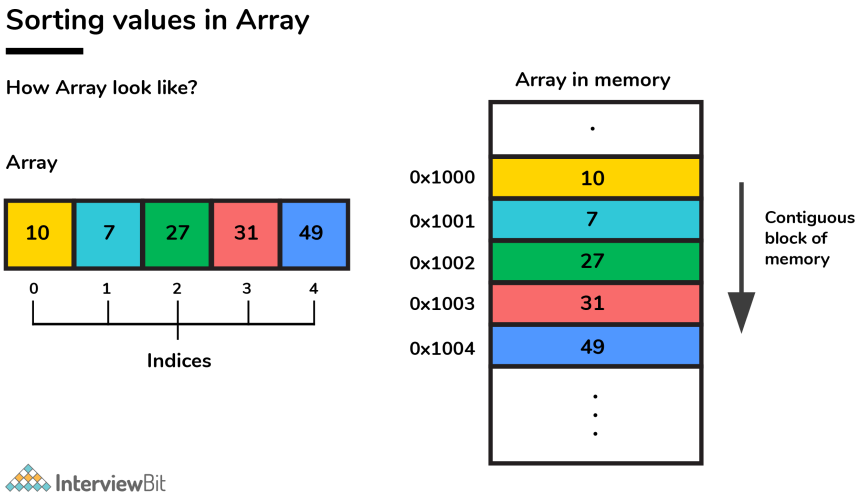


40. Contiguous memory locations are usually used for storing actual values in an array but not in ArrayList. Explain.

In the case of ArrayList, data storing in the form of primitive data types (like int, float, etc.) is not possible. The data members/objects present in the ArrayList have references to the objects which are located at various sites in the memory. Thus, storing of actual objects or non-primitive data types (like Integer, Double, etc.) takes place in various memory locations.



However, the same does not apply to the arrays. Object or primitive type values can be stored in arrays in contiguous memory locations, hence every element does not require any reference to the next element.



Java Advanced Interview Questions

41. Although inheritance is a popular OOPs concept, it is less advantageous than composition. Explain.

Inheritance lags behind composition in the following scenarios:

- Multiple-inheritance is not possible in Java. Classes can only extend from one superclass. In cases where multiple functionalities are required, for example - to read and write information into the file, the pattern of composition is preferred. The writer, as well as reader functionalities, can be made use of by considering them as the private members.
- Composition assists in attaining high flexibility and prevents breaking of encapsulation.
- Unit testing is possible with composition and not inheritance. When a developer wants to test a class composing a different class, then Mock Object can be created for signifying the composed class to facilitate testing. This technique is not possible with the help of inheritance as the derived class cannot be tested without the help of the superclass in inheritance.
- The loosely coupled nature of composition is preferable over the tightly coupled nature of inheritance.

Let's take an example:

```
package comparison;
public class Top {
public int start() {
  return 0;
}
}
class Bottom extends Top {
 public int stop() {
  return 0;
 }
}
```

In the above example, inheritance is followed. Now, some modifications are done to the Top class like this:

```
public class Top {
 public int start() {
  return 0;
 }
 public void stop() {
 }
}
```

If the new implementation of the Top class is followed, a compile-time error is bound to occur in the Bottom class. Incompatible return type is there for the Top.stop() function. Changes have to be made to either the Top or the Bottom class to ensure compatibility. However, the composition technique can be utilized to solve the given problem:

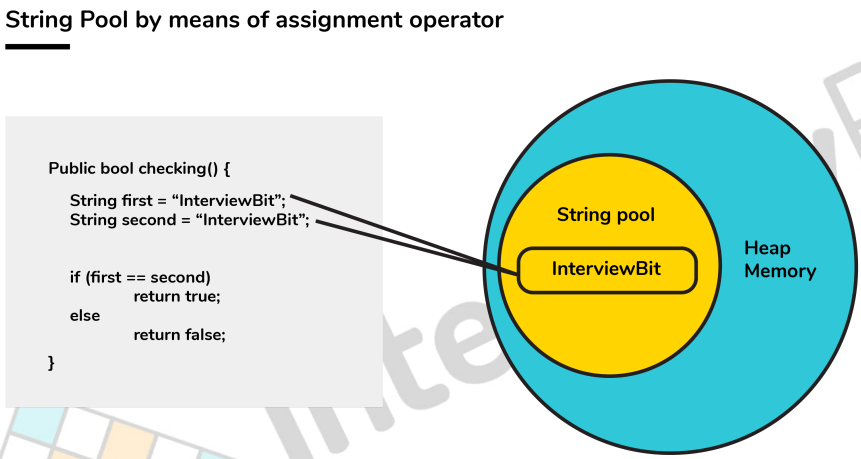
```
class Bottom {
 Top par = new Top();
 public int stop() {
  par.start();
  par.stop();
  return 0;
 }
}
```

42. How is the creation of a String using new() different from that of a literal?

When a String is formed as a literal with the assistance of an assignment operator, it makes its way into the String constant pool so that String Interning can take place. This same object in the heap will be referenced by a different String if the content is the same for both of them.

```
public bool checking() {
String first = "InterviewBit";
String second = "InterviewBit";
if (first == second)
 return true;
else
 return false;
}
```

The checking() function will return true as the same content is referenced by both the variables.

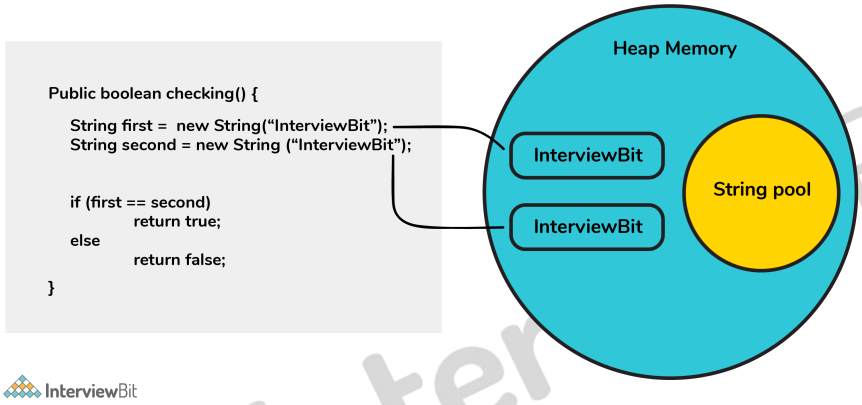


Conversely, when a String formation takes place with the help of a new() operator, interning does not take place. The object gets created in the heap memory even if the same content object is present.

```
public bool checking() {
String first = new String("InterviewBit");
String second = new String("InterviewBit");
if (first == second)
 return true;
else
 return false;
}
```

The checking() function will return false as the same content is not referenced by both the variables.

String Pool by means of new operator



43. Is exceeding the memory limit possible in a program despite having a garbage collector?

Yes, it is possible for the program to go out of memory in spite of the presence of a garbage collector. Garbage collection assists in recognizing and eliminating those objects which are not required in the program anymore, in order to free up the resources used by them.

In a program, if an object is unreachable, then the execution of garbage collection takes place with respect to that object. If the amount of memory required for creating a new object is not sufficient, then memory is released for those objects which are no longer in the scope with the help of a garbage collector. The memory limit is exceeded for the program when the memory released is not enough for creating new objects.

Moreover, exhaustion of the heap memory takes place if objects are created in such a manner that they remain in the scope and consume memory. The developer should make sure to dereference the object after its work is accomplished. Although the garbage collector endeavors its level best to reclaim memory as much as possible, memory limits can still be exceeded.

Let's take a look at the following example:

```
List<String> example = new LinkedList<String>();
while(true){
example.add(new String("Memory Limit Exceeded"));
}
```

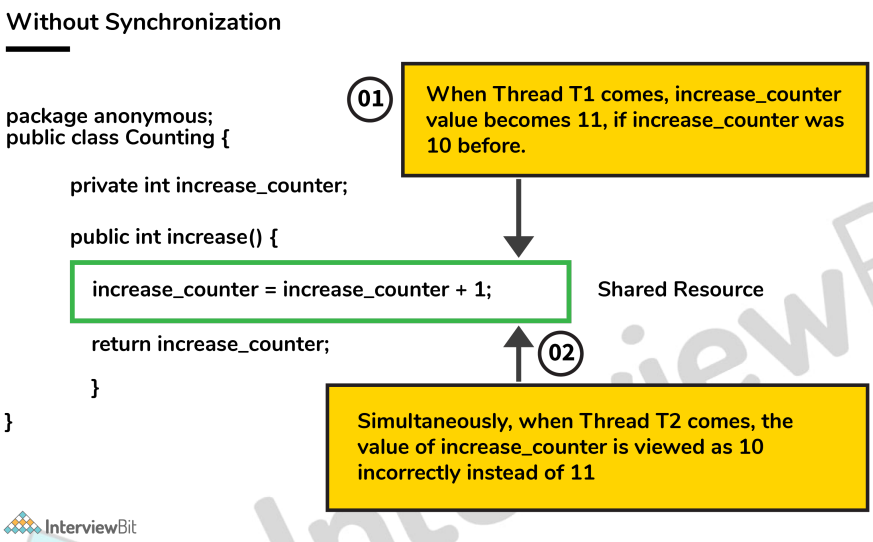
44. Why is synchronization necessary? Explain with the help of a relevant example.

Concurrent execution of different processes is made possible by synchronization. When a particular resource is shared between many threads, situations may arise in which multiple threads require the same shared resource.

Synchronization assists in resolving the issue and the resource is shared by a single thread at a time. Let's take an example to understand it more clearly. For example, you have a URL and you have to find out the number of requests made to it. Two simultaneous requests can make the count erratic.

No synchronization:

```
package anonymous;
public class Counting {
	private int increase_counter;
	public int increase() {
		increase_counter = increase_counter + 1;
		return increase_counter;
	}
}
```

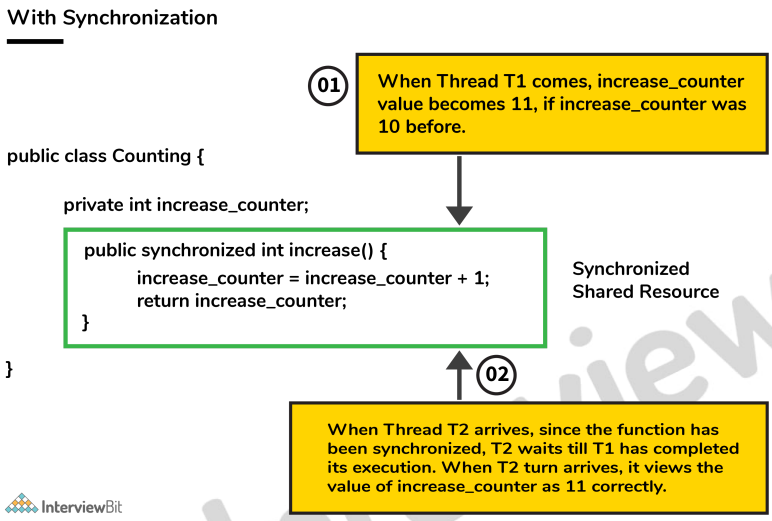


If a thread Thread1 views the count as 10, it will be increased by 1 to 11. Simultaneously, if another thread Thread2 views the count as 10, it will be increased by 1 to 11. Thus, inconsistency in count values takes place because the expected final value is 12 but the actual final value we get will be 11.

Now, the function increase() is made synchronized so that simultaneous accessing cannot take place.

With synchronization:

```
package anonymous;
public class Counting {
	private int increase_counter;
	public synchronized int increase() {
		increase_counter = increase_counter + 1;
		return increase_counter;
	}
}
```



If a thread Thread1 views the count as 10, it will be increased by 1 to 11, then the thread Thread2 will view the count as 11, it will be increased by 1 to 12. Thus, consistency in count values takes place.

45. In the given code below, what is the significance of ... ?

```
public void fooBarMethod(String... variables){
   // method code
}
```

- Ability to provide ... is a feature called varargs (variable arguments) which was introduced as part of Java 5.
- The function having ... in the above example indicates that it can receive multiple arguments of the datatype String.
- For example, the fooBarMethod can be called in multiple ways and we can still have one method to process the data as shown below:

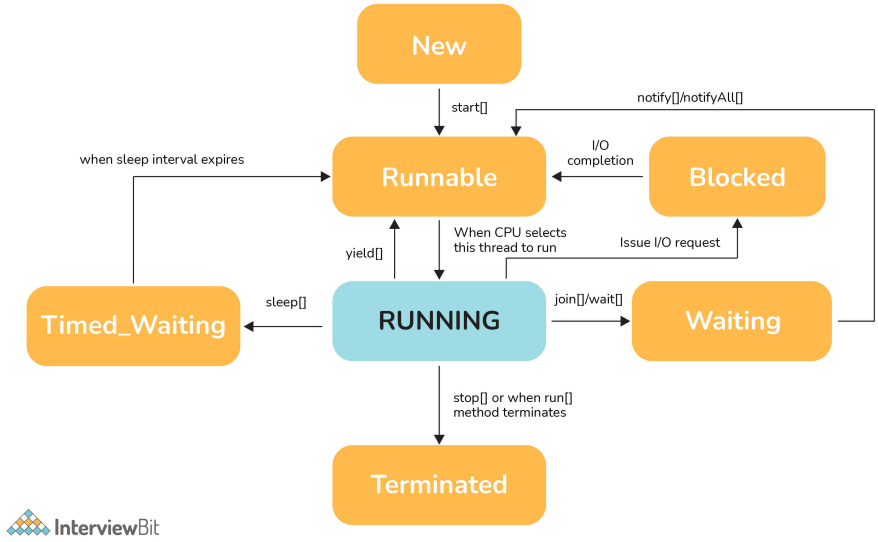
```
fooBarMethod("foo", "bar");
fooBarMethod("foo", "bar", "boo");
fooBarMethod(new String[]{"foo", "var", "boo"});
public void myMethod(String... variables){
   for(String variable : variables){
       // business logic
   }
}
```

46. Can you explain the Java thread lifecycle?

Java thread life cycle is as follows:

- **New** – When the instance of the thread is created and the start() method has not been invoked, the thread is considered to be alive and hence in the NEW state.
- **Runnable** – Once the start() method is invoked, before the run() method is called by JVM, the thread is said to be in RUNNABLE (ready to run) state. This state can also be entered from the Waiting or Sleeping state of the thread.
- **Running** – When the run() method has been invoked and the thread starts its execution, the thread is said to be in a RUNNING state.
- **Non-Runnable (Blocked/Waiting)** – When the thread is not able to run despite the fact of its aliveness, the thread is said to be in a NON-RUNNABLE state. Ideally, after some time of its aliveness, the thread should go to a runnable state.
    - A thread is said to be in a Blocked state if it wants to enter synchronized code but it is unable to as another thread is operating in that synchronized block on the same object. The first thread has to wait until the other thread exits the synchronized block.
    - A thread is said to be in a Waiting state if it is waiting for the signal to execute from another thread, i.e it waits for work until the signal is received.
- **Terminated** – Once the run() method execution is completed, the thread is said to enter the TERMINATED step and is considered to not be alive.

The following flowchart clearly explains the lifecycle of the thread in Java.



47. What could be the tradeoff between the usage of an unordered array versus the usage of an ordered array?

- The main advantage of having an ordered array is the reduced search time complexity of `O(log n)` whereas the time complexity in an unordered array is `O(n)` .
- The main drawback of the ordered array is its increased insertion time which is O(n) due to the fact that its element has to reordered to maintain the order of array during every insertion whereas the time complexity in the unordered array is only O(1).
- Considering the above 2 key points and depending on what kind of scenario a developer requires, the appropriate data structure can be used for implementation.

48. Is it possible to import the same class or package twice in Java and what happens to it during runtime?

It is possible to import a class or package more than once, however, it is redundant because the JVM internally loads the package or class only once.

49. In case a package has sub packages, will it suffice to import only the main package? e.g. Does importing of com.myMainPackage.* also import com.myMainPackage.mySubPackage.*?

This is a big NO. We need to understand that the importing of the sub-packages of a package needs to be done explicitly. Importing the parent package only results in the import of the classes within it and not the contents of its child/sub-packages.

50. Will the finally block be executed if the code System.exit(0) is written at the end of try block?

NO. The control of the program post `System.exit(0)` is immediately gone and the program gets terminated which is why the finally block never gets executed.

51. What do you understand by marker interfaces in Java?

Marker interfaces, also known as tagging interfaces are those interfaces that have no methods and constants defined in them. They are there for helping the compiler and JVM to get run time-related information regarding the objects.

52. Explain the term “Double Brace Initialisation” in Java?

This is a convenient means of initializing any collections in Java. Consider the below example.

```
import java.util.HashSet;
import java.util.Set;

public class IBDoubleBraceDemo{
   public static void main(String[] args){
       Set<String> stringSets = new HashSet<String>()
       {
           {
               add("set1");
               add("set2");
               add("set3");
           }
       };

       doSomething(stringSets);
   }

   private static void doSomething(Set<String> stringSets){
       System.out.println(stringSets);
   }
}
```

In the above example, we see that the stringSets were initialized by using double braces.

- The first brace does the task of creating an anonymous inner class that has the capability of accessing the parent class's behavior. In our example, we are creating the subclass of HashSet so that it can use the add() method of HashSet.
- The second braces do the task of initializing the instances.

Care should be taken while initializing through this method as the method involves the creation of anonymous inner classes which can cause problems during the garbage collection or serialization processes and may also result in memory leaks.

53. Why is it said that the length() method of String class doesn't return accurate results?

- The length method returns the number of Unicode units of the String. Let's understand what Unicode units are and what is the confusion below.
- We know that Java uses UTF-16 for String representation. With this Unicode, we need to understand the below two Unicode related terms:
  - Code Point: This represents an integer denoting a character in the code space.
  - Code Unit: This is a bit sequence used for encoding the code points. In order to do this, one or more units might be required for representing a code point.
- Under the UTF-16 scheme, the code points were divided logically into 17 planes and the first plane was called the Basic Multilingual Plane (BMP). The BMP has classic characters - U+0000 to U+FFFF. The rest of the characters- U+10000 to U+10FFFF were termed as the supplementary characters as they were contained in the remaining planes.
  - The code points from the first plane are encoded using **one** 16-bit code unit
  - The code points from the remaining planes are encoded using **two** code units.

Now if a string contained supplementary characters, the length function would count that as 2 units and the result of the length() function would not be as per what is expected.

In other words, if there is 1 supplementary character of 2 units, the length of that SINGLE character is considered to be TWO - Notice the inaccuracy here? As per the java documentation, it is expected, but as per the real logic, it is inaccurate.

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

```
public class InterviewBit{
   public static void main(String[] args)
   {
       System.out.println('b' + 'i' + 't');
   }
}
```

“bit” would have been the result printed if the letters were used in double-quotes (or the string literals). But the question has the character literals (single quotes) being used which is why concatenation wouldn't occur. The corresponding ASCII values of each character would be added and the result of that sum would be printed.
The ASCII values of ‘b’, ‘i’, ‘t’ are:

- ‘b’ = 98
- ‘i’ = 105
- ‘t’ = 116

```
98 + 105 + 116 = 319
```

Hence 319 would be printed.

55. What are the possible ways of making object eligible for garbage collection (GC) in Java?

First Approach: Set the object references to null once the object creation purpose is served.

```
public class IBGarbageCollect {
  public static void main (String [] args){
       String s1 = "Some String";
           // s1 referencing String object - not yet eligible for GC
       s1 = null; // now s1 is eligible for GC
   }
 }
```

Second Approach: Point the reference variable to another object. Doing this, the object which the reference variable was referencing before becomes eligible for GC.

```
public class IBGarbageCollect {
 public static void main(String [] args){
     String s1 = "To Garbage Collect";
     String s2 = "Another Object";
     System.out.println(s1); // s1 is not yet eligible for GC
     s1 = s2; // Point s1 to other object pointed by s2
     /* Here, the string object having the content  "To Garbage Collect" is not referre
 }
}
```

Third Approach: Island of Isolation Approach: When 2 reference variables pointing to instances of the same class, and these variables refer to only each other and the objects pointed by these 2 variables don't have any other references, then it is said to have formed an “Island of Isolation” and these 2 objects are eligible for GC.

```
public class IBGarbageCollect {
   IBGarbageCollect ib;
   public static void main(String [] str){
       IBGarbageCollect ibgc1 = new IBGarbageCollect();
       IBGarbageCollect ibgc2 = new IBGarbageCollect();
       ibgc1.ib = ibgc2; //ibgc1 points to ibgc2
       ibgc2.ib = ibgc1; //ibgc2 points to ibgc1
       ibgc1 = null;
       ibgc2 = null;
       /*
       * We see that ibgc1 and ibgc2 objects refer
       * to only each other and have no valid
       * references- these 2 objects for island of isolcation - eligible for GC
       */
   }
}
```

Java Interview Programs

56. Check if a given string is palindrome using recursion.

```
/*
* Java program to check if a given inputted string is palindrome or not using recursior
*/
import java.util.*;
public class InterviewBit {
   public static void main(String args[]) {
       Scanner s = new Scanner(System.in);
       String word = s.nextLine();
       System.out.println("Is "+word+" palindrome? - "+isWordPalindrome(word));
   }


   public static boolean isWordPalindrome(String word){
       String reverseWord = getReverseWord(word);
       //if word equals its reverse, then it is a palindrome
       if(word.equals(reverseWord)){
           return true;
       }
       return false;
   }

   public static String getReverseWord(String word){
       if(word == null || word.isEmpty()){
           return word;
       }

       return word.charAt(word.length()- 1) + getReverseWord(word.substring(0, word.ler
   }
}
```

57. Write a Java program to check if the two strings are anagrams.

The main idea is to validate the length of strings and then if found equal, convert the string to char array and then sort the arrays and check if both are equal.

```
import java.util.Arrays;
import java.util.Scanner;
public class InterviewBit {
 public static void main(String[] args) {
   Scanner s = new Scanner(System.in);
   //Input from two strings
   System.out.print("First String: ");
   String string1 = s.nextLine();
   System.out.print("Second String: ");
   String string2 = s.nextLine();
   // check for the length
   if(string1.length() == string2.length()) {
     // convert strings to char array
     char[] characterArray1 = string1.toCharArray();
     char[] characterArray2 = string2.toCharArray();
     // sort the arrays
     Arrays.sort(characterArray1);
     Arrays.sort(characterArray2);
     // check for equality, if found equal then anagram, else not an anagram
     boolean isAnagram = Arrays.equals(characterArray1, characterArray2);
     System.out.println("Anagram: "+ isAnagram);
 }
}
```

58. Write a Java Program to find the factorial of a given number.

```
public class FindFactorial {
   public static void main(String[] args) {
       int num = 10;
       long factorialResult = 1l;
       for(int i = 1; i <= num; ++i)
       {
           factorialResult *= i;
       }
       System.out.println("Factorial: "+factorialResult);
   }
}
```

59. Given an array of non-duplicating numbers from 1 to n where one number is missing, write an efficient java program to find that missing number.

Idea is to find the sum of n natural numbers using the formula and then finding the sum of numbers in the given array. Subtracting these two sums results in the number that is the actual missing number. This results in O(n) time complexity and O(1) space complexity.

```
public class IBMissingNumberProblem {

   public static void main(String[] args) {

       int[] array={4,3,8,7,5,2,6};
       int missingNumber = findMissingNum(array);
       System.out.println("Missing Number is "+ missingNumber);
   }

   public static int findMissingNum(int[] array) {
       int n=array.length+1;
       int sumOfFirstNNums=n*(n+1)/2;
       int actualSumOfArr=0;
       for (int i = 0; i < array.length; i++) {
           actualSumOfArr+=array[i];
       }
       return sumOfFirstNNums-actualSumOfArr;
   }
}
```

60. Write a Java Program to check if any number is a magic number or not. A number is said to be a magic number if after doing sum of digits in each step and inturn doing sum of digits of that sum, the ultimate result (when there is only one digit left) is 1.

Example, consider the number:

- Step 1: 163 => 1+6+3 = 10
- Step 2: 10 => 1+0 = 1 => Hence 163 is a magic number

```
public class IBMagicNumber{

   public static void main(String[] args) {
       int num = 163;
       int sumOfDigits = 0;
       while (num > 0 || sumOfDigits > 9)
       {
           if (num == 0)
           {
               num = sumOfDigits;
               sumOfDigits = 0;
           }
           sumOfDigits += num % 10;
           num /= 10;
       }

       // If sum is 1, original number is magic number
       if(sumOfDigits == 1) {
           System.out.println("Magic number");
       }else {
           System.out.println("Not magic number");
       }
   }
}
```

Conclusion

61. Conclusion

Java is one of the simple high-level languages that provides powerful tools and impressive standards required for application development. It was also one of the first languages to provide amazing threading support for tackling concurrency-based problems. The easy-to-use syntax and the built-in features of Java combined with the stability it provides to applications are the main reasons for this language to have ever-growing usage in the software community.

Join our community and share your java interview experiences.

Recommended tutorials:

Java Tutorial

Puzzles

Coding Interview Questions

Java 8 Interview Questions

How to Become a Java Developer?

Java Frameworks:

Spring

Hibernate

JAVA SE Download

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