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 JVM installed in it.

No software is really "independent". Eventually, your program has to call the underlying OS in order to make some basic operations, like allocating memory, create new threads etc.
The way to achieve an executable which is "cross platform" is to create specific executable for each OS. Common practice is to write different code for each OS, and then "hide" it in a cross platform interface and compile the relevant code to the relevant OS. For example, std::thread is "cross-platform" to the user who uses this class, but behind the scenes it will call different functions based on the OS which was specified on compile time (such as CreateThread on Windows, but pthread_create on *nix OS's).
So basically, the JVM is a C/C++ executable that was written with different set of functions for each OS, and was compiled separately for each OS. A JVM executable which works on Linux, will not work on Windows, and vice-versa.
That JVM compiles .class files to machine code based on the OS and CPU it currently operating on, so that's why Java programs can "run anywhere".
But basically, it's a *lie*. It's like saying a human being can live on Mars.... *if he lives inside a sealed spaceship with proper temperature, water, food, air and sunlight supply*
So Java program can run anywhere.... *if the JVM already is installed and running on the computer*.

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.

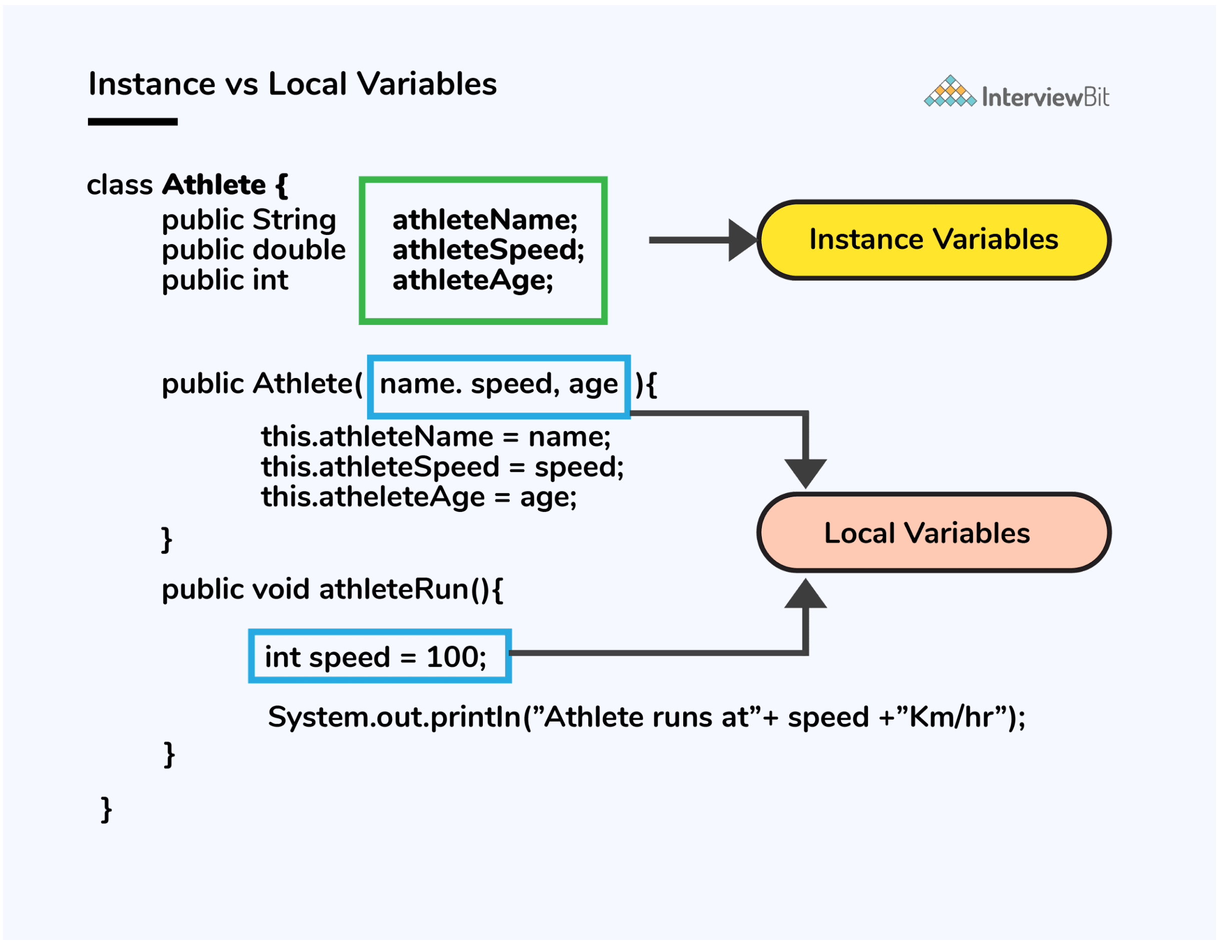
Example:

```
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:

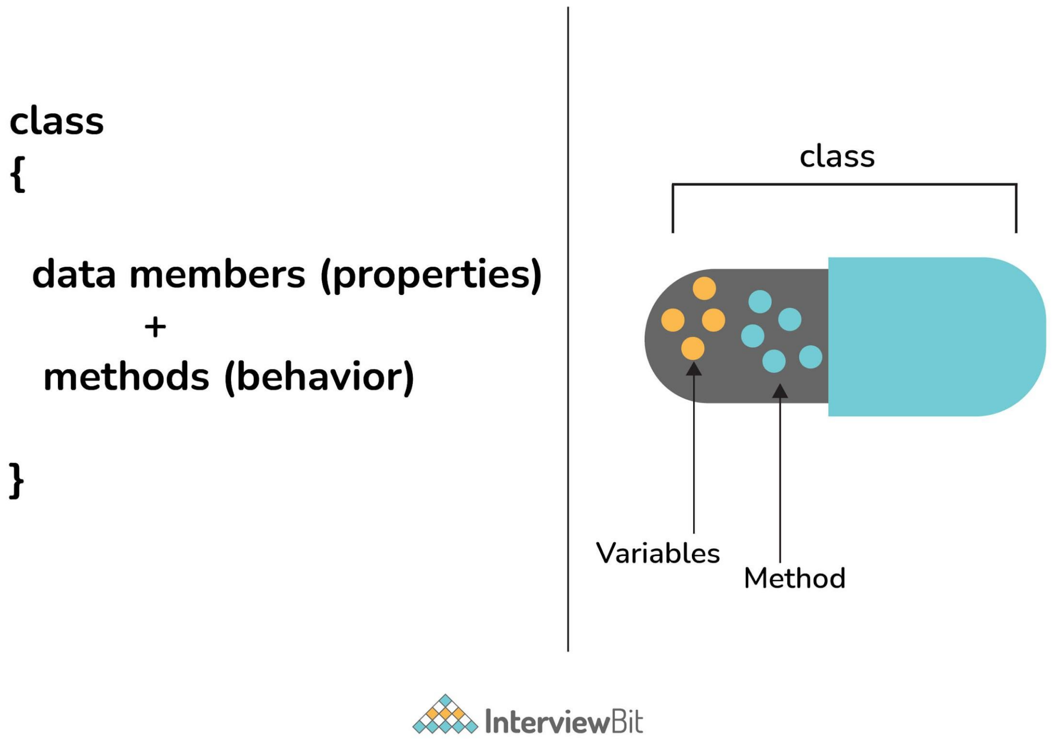
```
public void athlete() {
String athleteName;
double athleteSpeed;
int athleteAge;
}
```



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

        public Athlete( name. speed, age ){
                this.athleteName = name;
                this.athleteSpeed = speed;
                this.atheleteAge = age;
        }
        public void athleteRun(){

                int speed = 100;

                 System.out.println("Athlete runs at"+ speed +"Km/hr");
        }
}
```

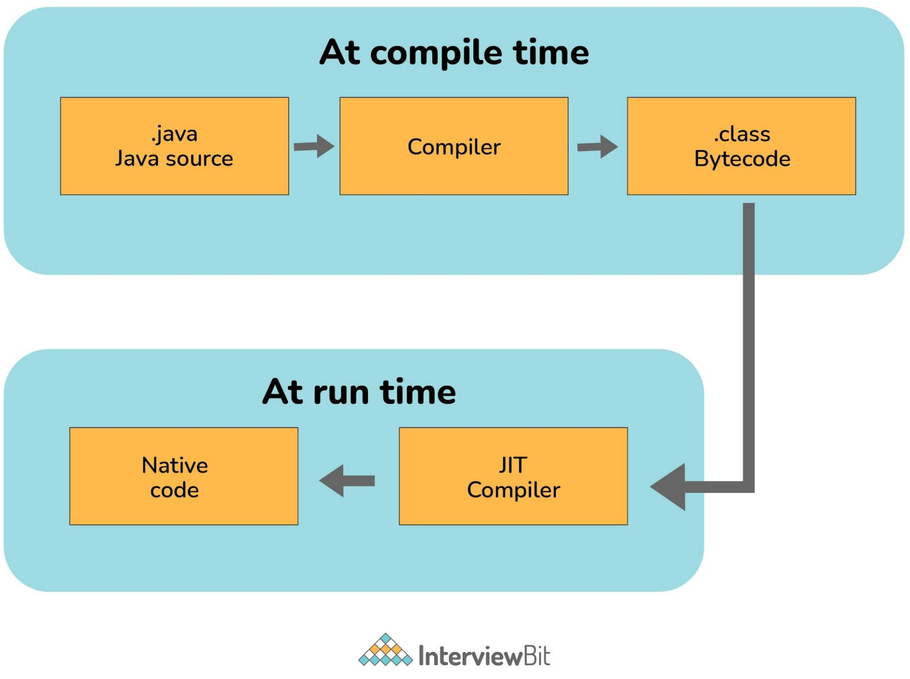
5. What do you mean by data encapsulation?

- 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.



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
}
```

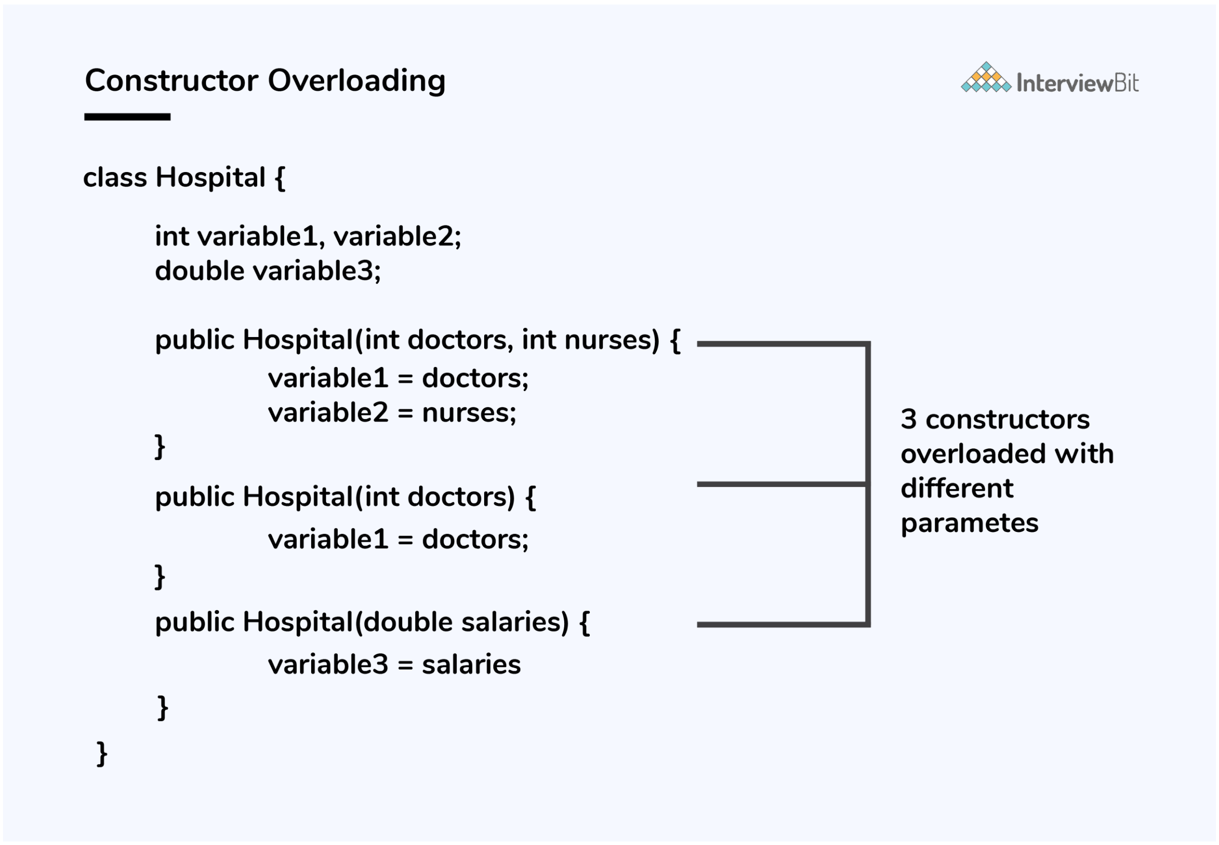
- Using do-while loop:

```
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
}
}
```



```
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.

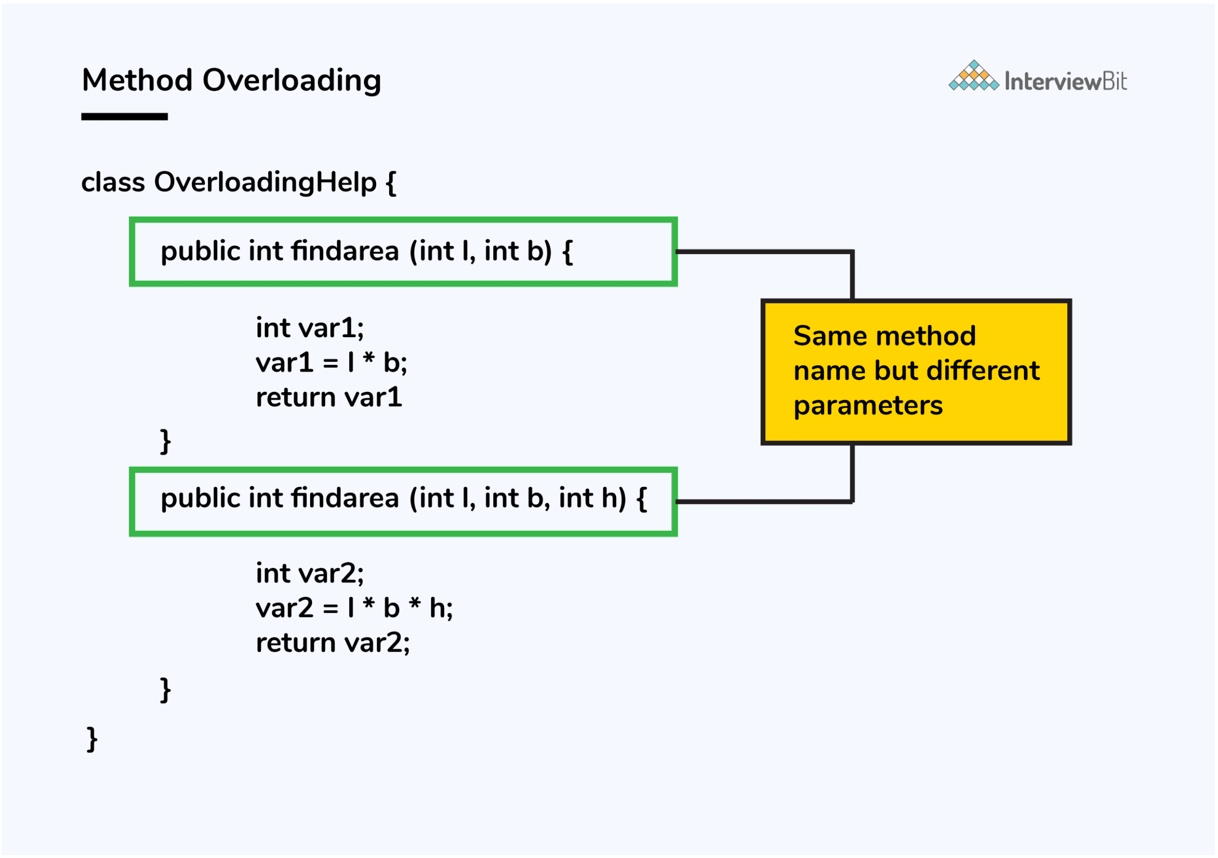
10. Comment on method overloading and overriding by citing relevant examples.

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;
    }
}
```



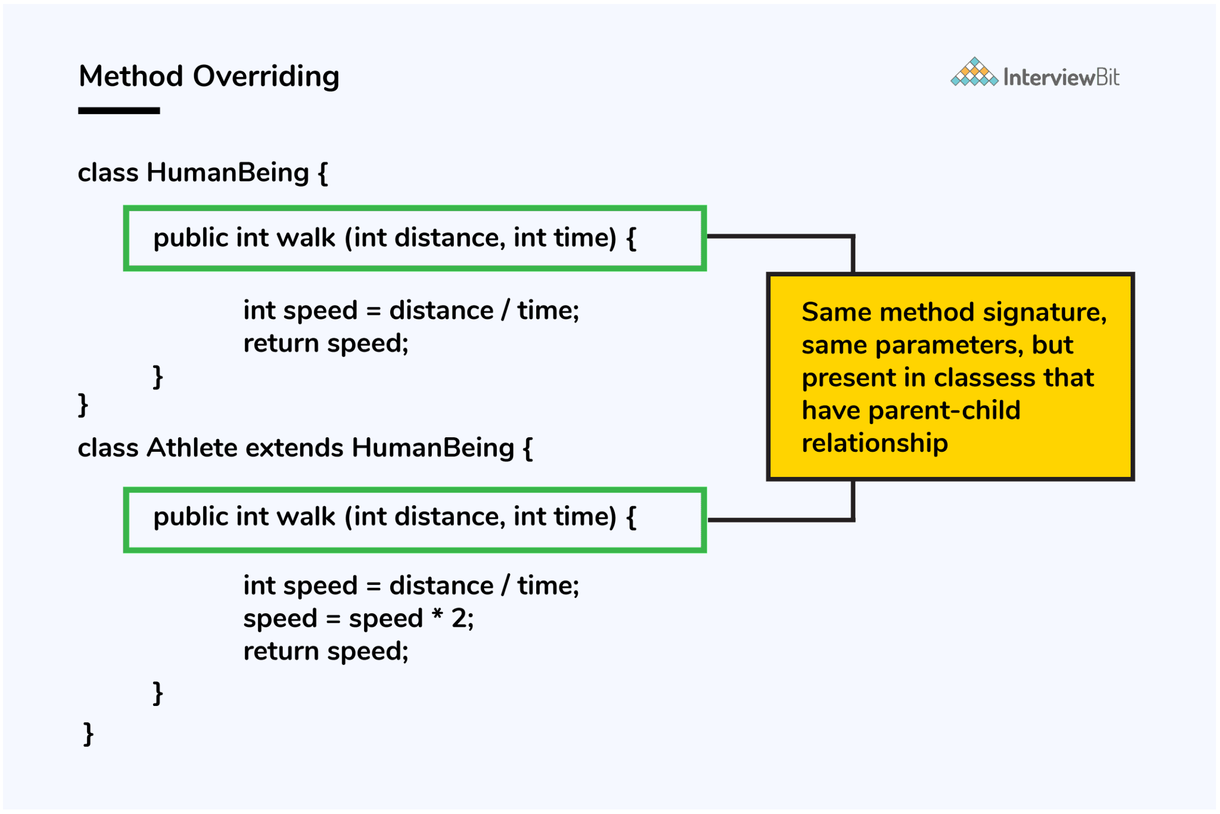
```
class OverloadingHelp {
    public int findarea (int l, int b) {
            int var1;
            var1 = l * b;
            return var1
    }
    public int findarea (int l, int b, int h) {
            int var2;
            var2 = l * b * h;
            return var2;
    }
}
```

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;
       }
}
```


Method Overriding
InterviewBit
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;
}
}
Same method signature, same parameters, but present in classess that have parent-child relationship

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 {
//open sql connection
int variable = 5/0;

}
catch (Exception exception) {
System.out.println("Exception occurred");
}
finally {
//close the connection
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("MasaiSchool");
example = null;
System.gc(); // Garbage collector called
}
public void finalize() {
```

```
// Finalize called
Math.max()
}
```

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{
       protected 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 method of Child class.
        }
    }
```

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.
- Static = class = no runtime polymorphism

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.

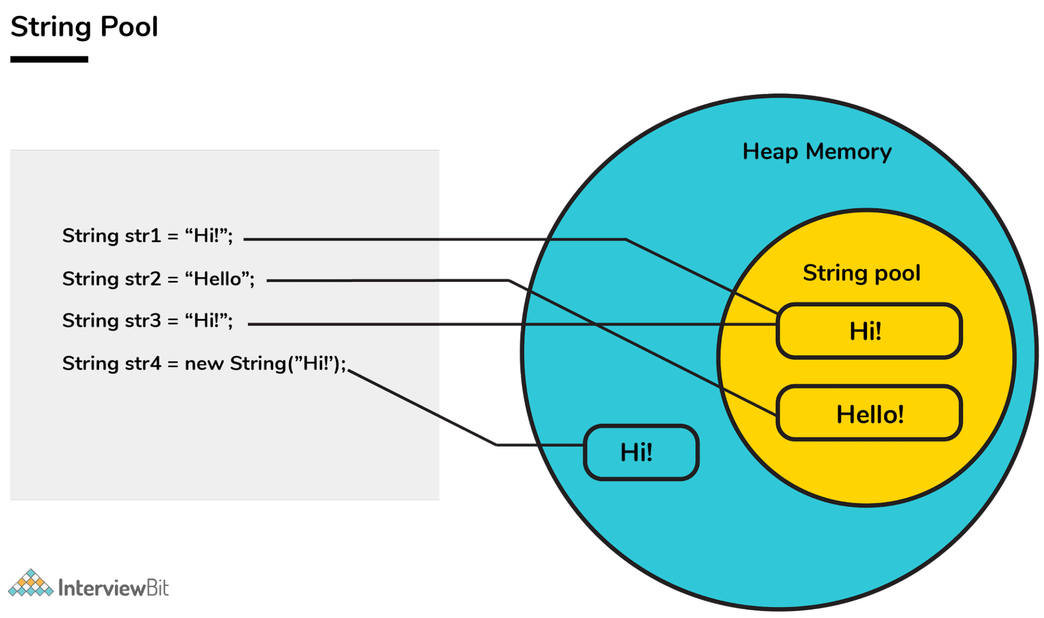
Stack memory is collected *automatically* when the execution path reaches the end of the scope

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 and a StringBuilder?

- Storage area: In string, the String pool serves as the storage area. For StringBuilder, heap memory is the storage area.
- Mutability: A String is immutable, whereas both the StringBuilder is mutable.

String str=”1”;

for(int i=2;i<101;i++){

String s99=str+”99”;

}

i=“1234….100”

```
// String
String first = "MasaiSchool";
String second = new String("MasaiSchool");
// StringBuilder
StringBuilder fourth = new StringBuilder("MasaiSchool");
fourth.append(“);
```

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. In Java 8, default methods in the interfaces.

  default perimeter(....)

- Variable types: Only Static and final variables can 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.

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 implementation requirements, computer program meeting the JVM requirements and instance object for executing the Java byte code and provide the runtime environment for execution.

Main Purpose	JDK is mainly used for code development and execution.	JRE is mainly used for environment creation to execute the code.	JVM provides specifications for all the implementations to JRE.
Tools provided	JDK provides tools like compiler, debuggers, etc for code development	JRE provides libraries and classes required by JVM to run the program.	JVM does not include any tools, but instead, it provides the specification for implementation.

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 MasaiSchoolThreadExample extends Thread{
   public void run(){
       System.out.println("Thread runs...");
   }
```

```
    public static void main(String args[]){
        MasaiSchoolThreadExample ib = new
MasaiSchoolThreadExample();
        ib.start();
    }
}
```

- Implementing the Runnable interface

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

Final variable instantiations are possible inside a constructor and the scope of this applies to the whole class and its objects.	A final variable if initialised inside a method ensures that the variable cant be changed only within the scope of that method.

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 MasaiSchoolTest{
    int num;
    MasaiSchoolTest(int x){
          num = x;
    }
    MasaiSchoolTest(){
          num = 0;
    }
}
class Driver {
    public static void main(String[] args)
    {
          //create a reference
          MasaiSchoolTest ibTestObj = new MasaiSchoolTest(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(MasaiSchoolTest ibObj)
    {
          // Point the object to new reference
          ibObj = new MasaiSchoolTest();
          // 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 MasaiSchoolTest{
    int num;
    MasaiSchoolTest(int x){
          num = x;
    }
    MasaiSchoolTest(){
          num = 0;
    }
}
class Driver{
    public static void main(String[] args)
    {
          //create a reference
          MasaiSchoolTest ibTestObj = new MasaiSchoolTest(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(MasaiSchoolTest 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.