JAVA
INTERVIEW QUESTIONS

Learn Coding from Scratch

CODING SCRATCH

8800373756

8800373756

NotesGallery
JobsPur

www.codingscratch.net

What is Java?

Java is a high-level programming language that was developed by James Gosling in the year 1982. It is based on the principles of object-oriented programming and can be used to develop large-scale applications.

Java Interview Questions for Freshers

1. Why is Java a platform independent language?

Java language was developed so that it does not depend on any hardware or software because 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.

Create A Free Personalised Study Plan
Get into your dream companies with expert guidance

Real-Life Problems

Prep for Target Roles

Custom Plan Duration

Create My Plan

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. Difference between Heap and Stack Memory in java. And how java utilizes this.

Stack memory is the portion of memory that was assigned to every individual program. And it was fixed. On the other hand, Heap memory is the portion that was not allocated to the java program but it will be available for use by the java program when it is required, mostly during the runtime of the program.
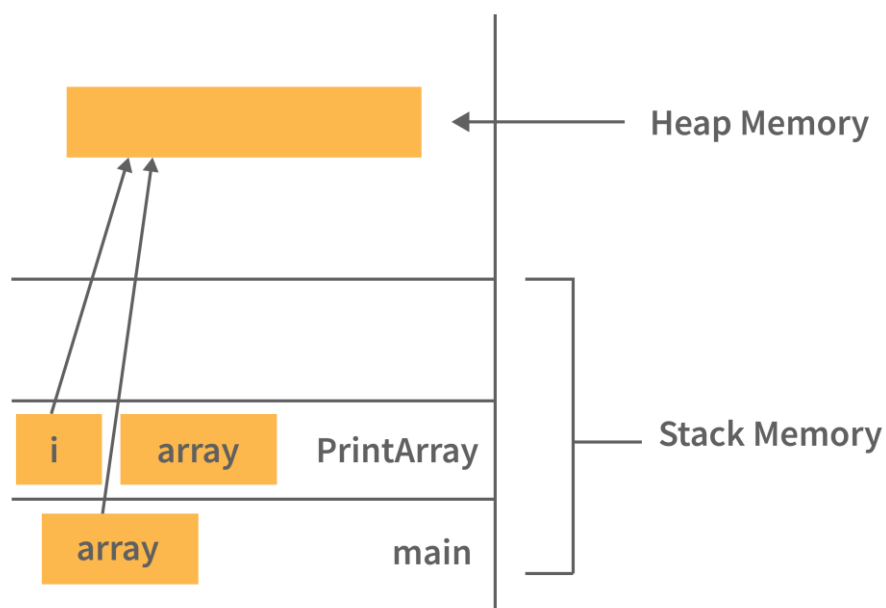
Java Utilizes this memory as -

- When we write a java program then all the variables, methods, etc are stored in the stack memory.
- And when we create any object in the java program then that object was created in the heap memory. And it was referenced from the stack memory.

Example- **Consider the below java program**:

```java
class Main {
  public void printArray(int[] array){
    for(int i : array)
      System.out.println(i);
  }
  public static void main(String args[]) {
    int[] array = new int[10];
    printArray(array);
  }
}
```

For this java program. The stack and heap memory occupied by java is -

Main and PrintArray is the method that will be available in the stack area and as well as the variables declared that will also be in the stack area.

And the Object (Integer Array of size 10) we have created, will be available in the Heap area because that space will be allocated to the program during runtime.

You can download a PDF version of Java Interview Questions.

  Download PDF

---

4. Can java be said to be the complete object-oriented programming language?

It is not wrong if we claim that Java is the complete object-oriented programming language because everything in Java is under the classes and we can access them by creating the objects.

But we can even say that Java is not a completely object-oriented programming language because it has the support of primitive data types like int, float, char, boolean, double, etc.

Now for the question: **Is Java a completely object-oriented programming language?** We can say that - Java is not a pure object-oriented programming language, because it has direct access to primitive data types. And these primitive data types don't directly belong to the Integer classes.

5. How is Java different from C++?

- C++ is only a compiled language, whereas Java is compiled as well as an interpreted language.
- Java programs are machine-independent whereas a c++ program can run only in the machine in which it is compiled.
- C++ allows users to use pointers in the program. Whereas java doesn't allow it. Java internally uses pointers.
- C++ supports the concept of Multiple inheritances whereas Java doesn't support this. And it is due to avoiding the complexity of name ambiguity that causes the diamond problem.

6. 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.

7. 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.
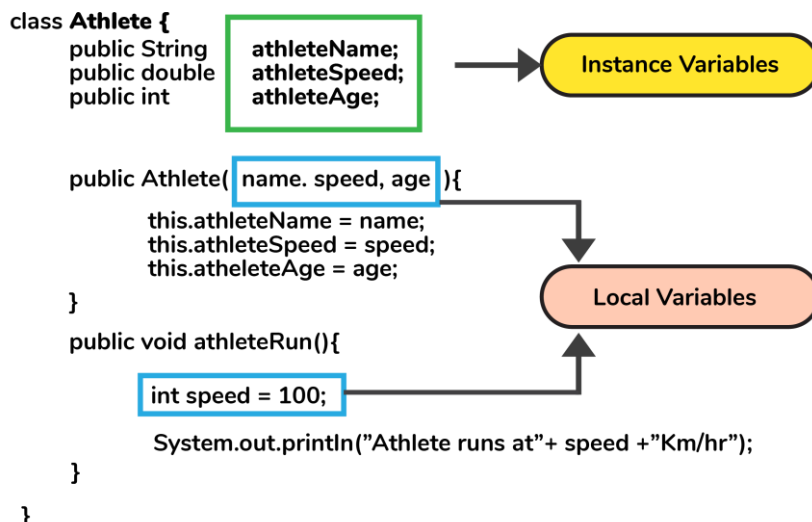
Example:

```
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;
}
```
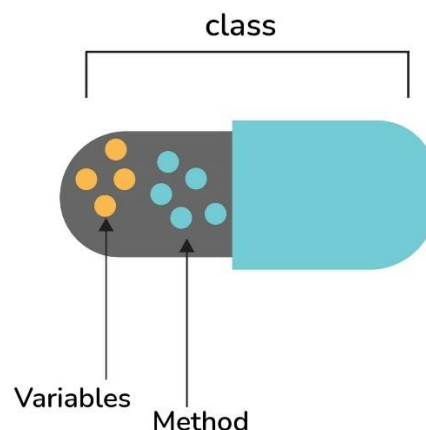
8. What are the default values assigned to variables and instances in java?

- There are no default values assigned to the variables in java. We need to initialize the value before using it. Otherwise, it will throw a compilation error of (**Variable might not be initialized**).
- But for instance, if we create the object, then the default value will be initialized by the default constructor depending on the data type.
- If it is a reference, then it will be assigned to null.
- If it is numeric, then it will assign to 0.
- If it is a boolean, then it will be assigned to false. Etc.
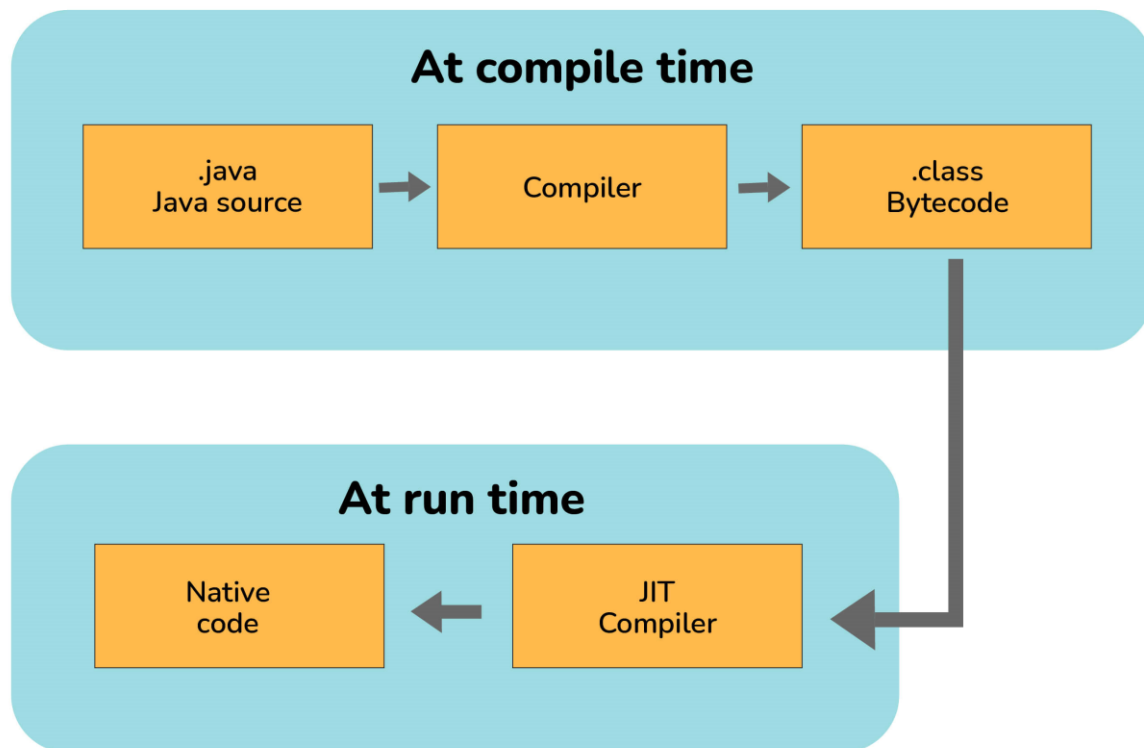
9. What do you mean by data encapsulation?

- Data Encapsulation is an Object-Oriented Programming concept of hiding the data attributes and their behaviours 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)


}
```

10. 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.

At compile time

```
.java            →    Compiler    →    .class
Java source                            Bytecode
```

At run time

```
Native    ←    JIT
code           Compiler
```

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

We are already aware of the **(==) equals** operator. That we have used this to compare the equality of the values. But when we talk about the terms of object-oriented programming, we deal with the values in the

form of objects. And this object may contain multiple types of data. So using the **(==) operator** does not work in this case. So we need to go with the .**equals() method.**
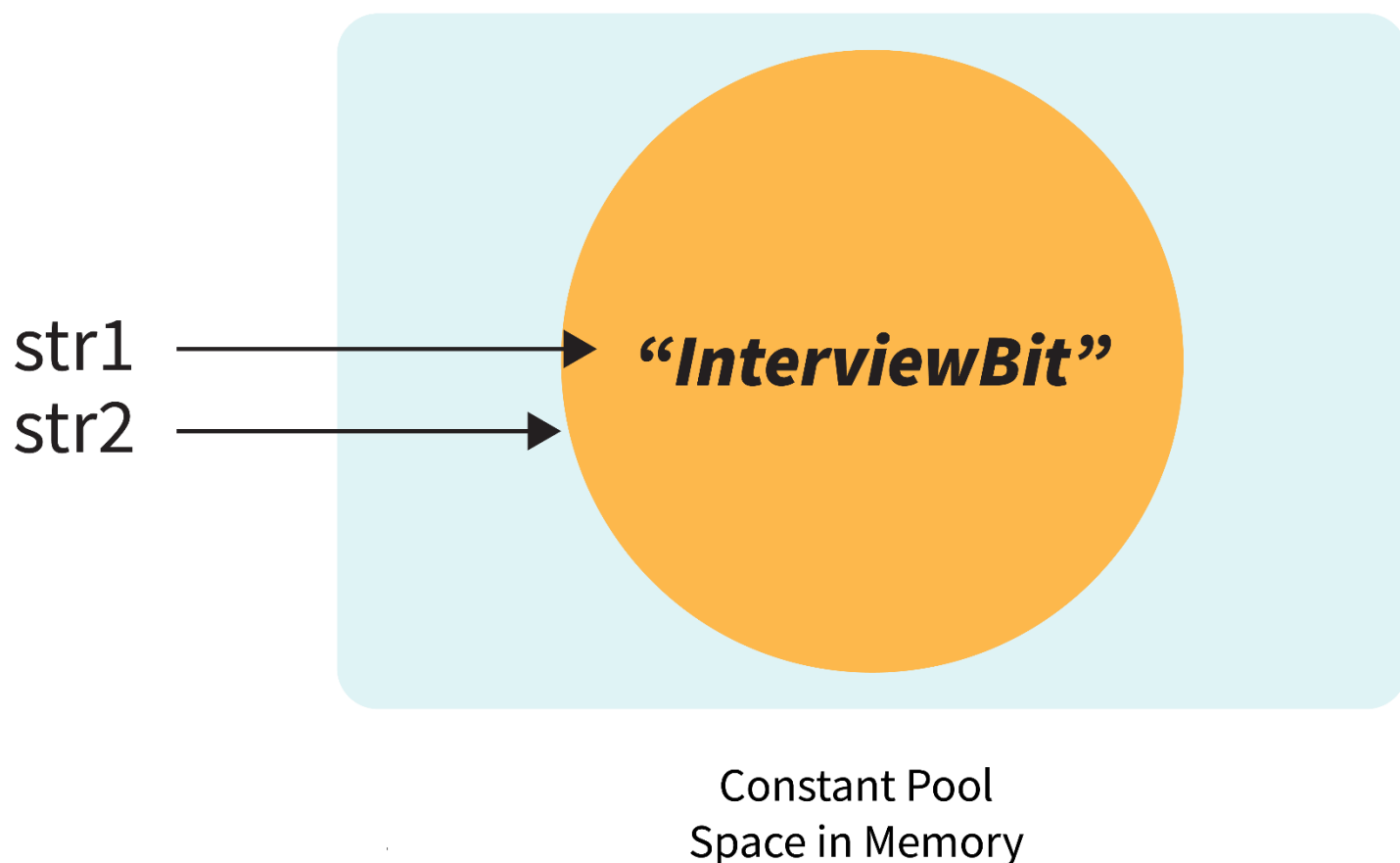
Both [**(==)** and .**equals()**] primary functionalities are to compare the values, but the secondary functionality is different.

So in order to understand this better, let's consider this with the example -

```
String str1 = "InterviewBit";
String str2 = "InterviewBit";

System.out.println(str1 == str2);
```
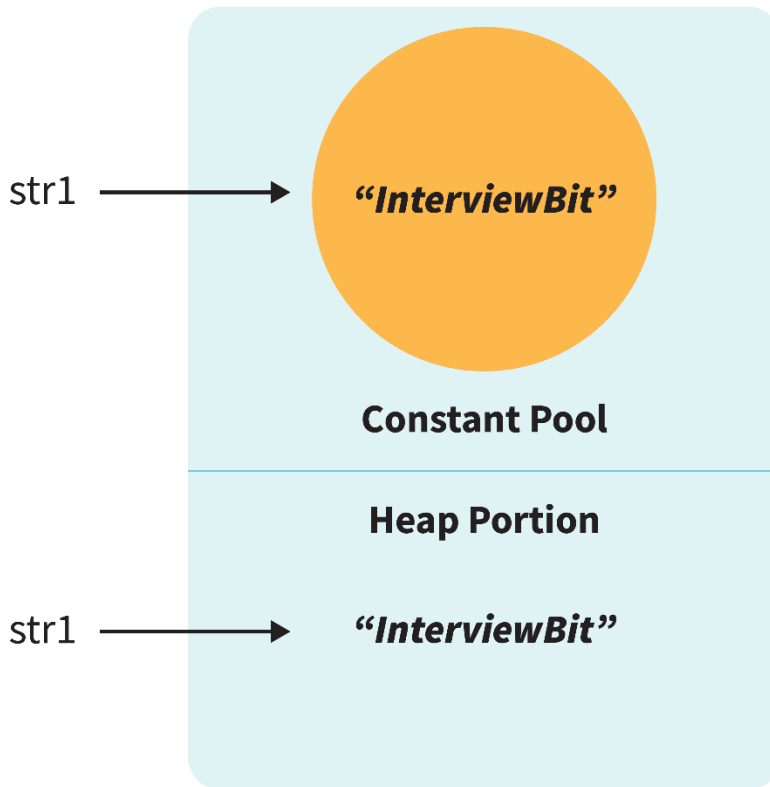
This code will print true. We know that both strings are equals so it will print true. But here **(==) Operators** don't compare each character in this case. It compares the memory location. And because the string uses the constant pool for storing the values in the memory, both **str1** and **str2** are stored at the same memory location. See the detailed Explanation in Question no 73: **Link**.

str1
str2 → **"InterviewBit"**

Constant Pool
Space in Memory

Now, if we modify the program a little bit with -

```
String str1 = new String("InterviewBit");
String str2 = "InterviewBit";
```

```
System.out.println(str1 == str2);
```


Then in this case, it will print false. Because here no longer the constant pool concepts are used. Here, new memory is allocated. So here the memory address is different, therefore (==) Operator returns false. But the twist is that the values are the same in both strings. So how to compare the values? Here the .equals() method is used.

.equals() method compares the values and returns the result accordingly. If we modify the above code with -

```
System.out.println(str1.equals(str2));
```

Then it returns true.

equals()	==
This is a method defined in the Object class.	It is a binary operator in Java.
The .equals() Method is present in the Object class, so we can override our custom .equals() method in the custom class, for objects comparison.	It cannot be modified. They always compare the HashCode.
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.

12. 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);
```

Get Access to 250+ Guides with Scaler Mobile App!
Experience free learning content on the Scaler Mobile App

4.5
100K+
Play Store

13. 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 {
```

```java
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
}
}
```

Constructor Overloading

InterviewBit

```java
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
        }
}
```

3 constructors overloaded with different parametes

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

14. Define Copy constructor in java.

Copy Constructor is the constructor used when we want to initialize the value to the new object from the old object of the same class.

```java
class InterviewBit{
    String department;
```

```
    String service;
    InterviewBit(InterviewBit ib){
        this.departments = ib.departments;
        this.services = ib.services;
    }
}
```

Here we are initializing the new object value from the old object value in the constructor. Although, this can also be achieved with the help of object cloning.

15. Can the main method be Overloaded?

Yes, It is possible to overload the main method. We can create as many overloaded main methods we want. However, JVM has a predefined calling method that JVM will only call the main method with the definition of -

public static void main(string[] args)

Consider the below code snippets:

```
class Main {
    public static void main(String args[]) {
        System.out.println(" Main Method");
    }
    public static void main(int[] args){
        System.out.println("Overloaded Integer array Main Method");
    }
    public static void main(char[] args){
        System.out.println("Overloaded Character array Main Method");
    }
    public static void main(double[] args){
        System.out.println("Overloaded Double array Main Method");
    }
    public static void main(float args){
        System.out.println("Overloaded float Main Method");
    }
}
```

16. 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;
    }
}
```

Method Overloading

InterviewBit

```
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;

        }

    }
```

Same method name but different parameters

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) {
```

```java
        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;
>   }
> **}**

Same method signature, same parameters, but present in classess that have parent-child relationship

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.

17. 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:

```java
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.

18. 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.

19. 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
}
```

20. Is it possible that the 'finally' block will not be executed? If yes then list the case.

Yes. It is possible that the 'finally' block will not be executed. The cases are-

- Suppose we use **System.exit()** in the above statement.
- If there are fatal errors like Stack overflow, Memory access error, etc.

21. Identify the output of the java program and state the reason.

```
1. public class InterviewBit
2. {
3.        public static void main(String[] args) {
4.                final int i;
5.                i = 20;
6.                int j = i+20;
7.                i = j+30;
8.           System.out.println(i + " " + j);
9.        }
10. }
```

The above code will generate a compile-time error at Line 7 saying - **[error: variable i might already have been initialized]**. It is because variable 'i' is the final variable. And final variables are allowed to be initialized only once, and that was already done on line no 5.

22. 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:

- o Accessing data members of parent class when the member names of the class and its child subclasses are same.
- o To call the default and parameterized constructor of the parent class inside the child class.
- o 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.

```java
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!");
    }
}
class Child extends Parent{
    private int num = 2;

    Child(){
        //super constructor call should always be in the first line
        // super();         // Either call default super() to call default parent constructor OR
        super("Call Parent");   // call parameterised super to call parameterised parent constructor.
        System.out.println("Child class default 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("Child class foo!");
        super.foo();   //Calls foo method of Parent class inside the Overriden foo method of Child class.
    }
}

public class DemoClass {
  public static void main(String args[]) {
   Child demoObject=new Child();
   demoObject.foo();
   /*
   This would print -
   Parent class parameterised constructor.
   Child class default Constructor
   Child class foo!
   Parent class foo!
   */
  }
}
```

23. 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.

24. Why is the main method static in Java?

The main method is always static because static members are those methods that belong to the classes, not to an individual object. So if the main method will not be static then for every object, It is available. And that is not acceptable by JVM. JVM calls the main method based on the class name itself. Not by creating the object.

Because there must be only 1 main method in the java program as the execution starts from the main method. So for this reason the main method is static.

25. 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.

26. Difference between static methods, static variables, and static classes in java.

- **Static Methods and Static variables** are those methods and variables that belong to the class of the java program, not to the object of the class. This gets memory where the class is loaded. And these can directly be called with the help of class names.
    - For example - We have used mathematical functions in the java program like - max(), min(), sqrt(), pow(), etc. And if we notice that, then we will find that we call it directly with the class name. Like - Math.max(), Math.min(), etc. So that is a static method. And Similarly static variables we have used like (length) for the array to get the length. So that is the static method.
- **Static classes** - A class in the java program cannot be static except if it is the inner class. If it is an inner static class, then it exactly works like other static members of the class.

27. 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.

28. What is a ClassLoader?

- Java Classloader is the program that belongs to JRE (Java Runtime Environment). The task of ClassLoader is to load the required classes and interfaces to the JVM when required.

- **Example**- To get input from the console, we require the scanner class. And the Scanner class is loaded by the ClassLoader.

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

Heap.

30. What are shallow copy and deep copy in java?

To copy the object's data, we have several methods like deep copy and shallow copy.

Example -

```
class Rectangle{
int length = 5;
    int breadth = 3;
}
```

Object for this Rectangle class - **Rectangle obj1 = new Rectangle();**

- **Shallow copy** - The shallow copy only creates a new reference and points to the same object. Example - For Shallow copy, we can do this by -
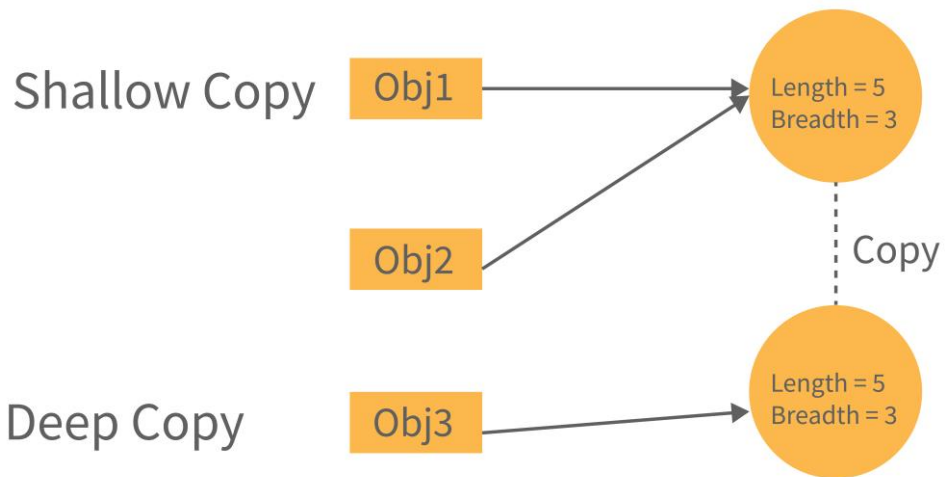
```
Rectangle obj2 = obj1;
```

Now by doing this what will happen is the new reference is created with the name obj2 and that will point to the same memory location.

- **Deep Copy** - In a deep copy, we create a new object and copy the old object value to the new object. Example -

```
Rectangle obj3 = new Rectangle();
Obj3.length = obj1.length;
Obj3.breadth = obj1.breadth;
```

Both these objects will point to the memory location as stated below -

Shallow Copy — Obj1, Obj2 → Length = 5, Breadth = 3

Copy

Deep Copy — Obj3 → Length = 5, Breadth = 3

Now, if we change the values in shallow copy then they affect the other reference as well. Let's see with the help of an example -

```java
class Rectangle
{
int length = 5;
   int breadth = 3;
}
public class Main
{
public static void main(String[] args) {
 Rectangle obj1 = new Rectangle();
 //Shallow Copy
      Rectangle obj2 = obj1;

      System.out.println(" Before Changing the value of object 1, the object2 will be - ");
      System.out.println(" Object2 Length = "+obj2.length+", Object2 Breadth = "+obj2.breadth);

      //Changing the values for object1.
      obj1.length = 10;
      obj1.breadth = 20;

      System.out.println("\n After Changing the value of object 1, the object2 will be - ");
      System.out.println(" Object2 Length = "+obj2.length+", Object2 Breadth = "+obj2.breadth);

}
}
```

Output -

```
Before Changing the value of object 1, the object2 will be -
Object2 Length = 5, Object2 Breadth = 3

After Changing the value of object 1, the object2 will be -
Object2 Length = 10, Object2 Breadth = 20
```

We can see that in the above code, if we change the values of object1, then the object2 values also get changed. It is because of the reference.

Now, if we change the code to deep copy, then there will be no effect on object2 if it is of type deep copy. Consider some snippets to be added in the above code.

```java
class Rectangle
{
   int length = 5;
   int breadth = 3;
}
public class Main
{
public static void main(String[] args) {
 Rectangle obj1 = new Rectangle();
 //Shallow Copy
        Rectangle obj2 = new Rectangle();
        obj2.length = obj1.length;
        obj2.breadth = obj1.breadth;

        System.out.println(" Before Changing the value of object 1, the object2 will be - ");
        System.out.println(" Object2 Length = "+obj2.length+", Object2 Breadth = "+obj2.breadth);

        //Changing the values for object1.
        obj1.length = 10;
        obj1.breadth = 20;

        System.out.println("\n After Changing the value of object 1, the object2 will be - ");
        System.out.println(" Object2 Length = "+obj2.length+", Object2 Breadth = "+obj2.breadth);

}
}
```

The above snippet will not affect the object2 values. It has its separate values. The output will be

```
Before Changing the value of object 1, the object2 will be -
Object2 Length = 5, Object2 Breadth = 3

After Changing the value of object 1, the object2 will be -
Object2 Length = 5, Object2 Breadth = 3
```

Now we see that we need to write the number of codes for this deep copy. So to reduce this, In java, there is a method called **clone().**

The **clone()** will do this deep copy internally and return a new object. And to do this we need to write only 1 line of code. That is - **Rectangle obj2 = obj1.clone();**
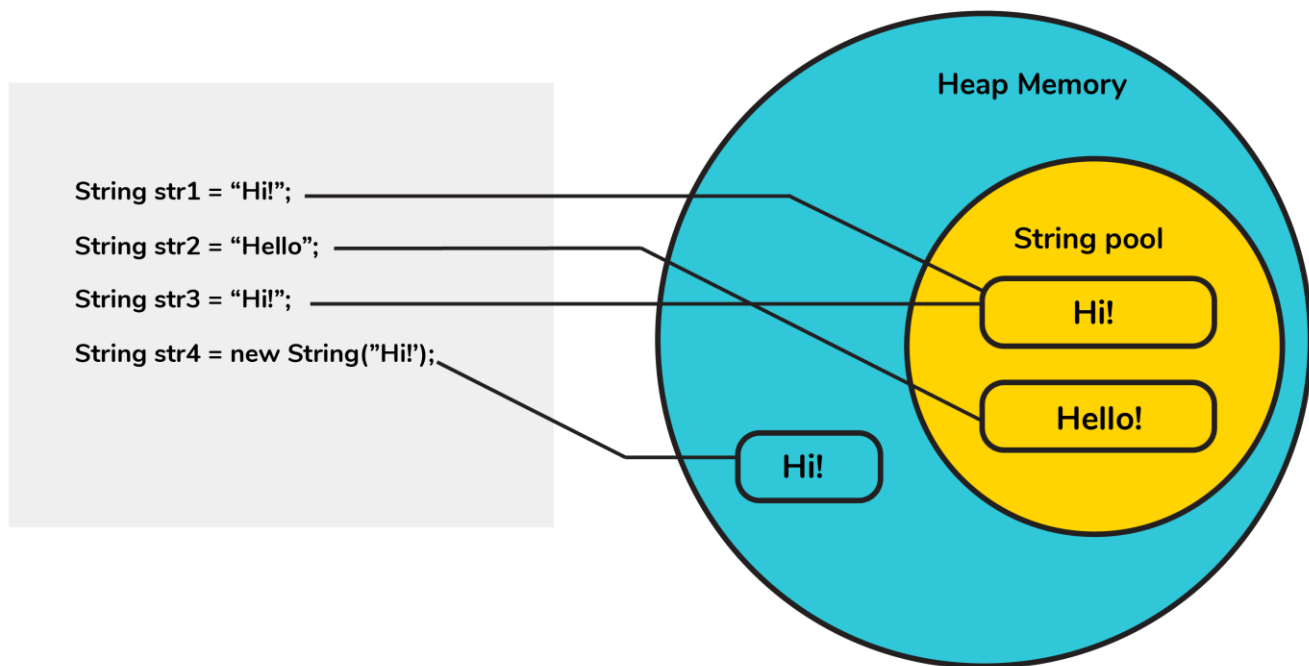
Java Intermediate Interview Questions

1. 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.

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.

2. What is a singleton class in Java? And How to implement a singleton class?

Singleton classes are those classes, whose objects are created only once. And with only that object the class members can be accessed.

Understand this with the help of an example-:

Consider the water jug in the office and if every employee wants that water then they will not create a new water jug for drinking water. They will use the existing one with their own reference as a glass. So programmatically it should be implemented as -

```java
class WaterJug{
  private int waterQuantity = 500;
  private WaterJug(){}
  private WaterJug object = null;

  // Method to provide the service of Giving Water.
  public int getWater(int quantity){
    waterQuantity -= quantity;
    return quantity;
  }
  // Method to return the object to the user.
  public static Waterjug getInstance(){
    // Will Create a new object if the object is not already created and return the object.
    if(object == null){
      object = new WaterJug();
    }
    return object;
  }
}
```

In the above class, the Constructor is private so we cannot create the object of the class. But we can get the object by calling the method **getInstance()**. And the getInstance is static so it can be called without creating the object. And it returns the object. Now with that object, we can call **getWater()** to get the water.

```java
Waterjug glass1 = WaterJug.getInstance();
glass1.getWater(1);
```

We can get the single object using this getInstance(). And it is static, so it is a thread-safe singleton class. Although there are many ways to create a thread-safe singleton class. So thread-safe classes can also be:

- When singletons are written with double-checked locking, they can be thread-safe.
- We can use static singletons that are initialized during class loading. Like we did in the above example.
- But the most straightforward way to create a thread-safe singleton is to use Java enums.

3. Which of the below generates a compile-time error? State the reason.

1. **int[] n1 = new int[0];**
2. **boolean[] n2 = new boolean[-200];**
3. **double[] n3 = new double[2241423798];**
4. **char[] ch = new char[20];**

We get a compile-time error in line 3. The error we will get in Line 3 is - **integer number too large**. It is because the array requires size as an integer. And Integer takes 4 Bytes in the memory. And the number (**2241423798**) is beyond the capacity of the integer. The maximum array size we can declare is - (**2147483647**).

Because the array requires the size in integer, none of the lines (1, 2, and 4) will give a compile-time error. The program will compile fine. But we get the runtime exception in line 2. The exception is
- **NegativeArraySizeException**.

Here what will happen is - At the time when JVM will allocate the required memory during runtime then it will find that the size is negative. And the array size can't be negative. So the JVM will throw the exception.

4. 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");
```

5. 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();
}
```

6. Is this program giving a compile-time error? If Yes then state the reason and number of errors it will give. If not then state the reason.

```
abstract final class InterviewBit{
2.    public abstract void printMessage();
3. }
4. class ScalarAcademy extends InterviewBit{
5.    public void printMessage(){
6.       System.out.println("Welcome to Scalar Academy By InterviewBit");
7.    }
8. }
9. class ScalarTopics extends ScalarAcademy{
10.   public void printMessage(){
11.      System.out.println("Welcome to Scalar Topics By Scalar Academy");
12.   }
13. }
public class Main{
        public static void main(String[] args) {
            InterviewBit ib = new ScalarTopics();
            ib.printMessage();
        }
}
```

The above program will give a compile-time error. The compiler will throw 2 errors in this.

- **[Illegal Combination of modifiers: abstract and final]** at line 1.
- **[Cannot inherit from final 'InterviewBit']** at line 4.

It is because abstract classes are incomplete classes that need to be inherited for making their concrete classes. And on the other hand, the final keywords in class are used for avoiding inheritance. So these combinations are not allowed in java.

7. What is a Comparator in java?

Consider the example where we have an ArrayList of employees like(EId, Ename, Salary), etc. Now if we want to sort this list of employees based on the names of employees. Then that is not possible to sort using the Collections.sort() method. We need to provide something to the sort() function depending on what values we have to perform sorting. Then in that case a comparator is used.

Comparator is the interface in java that contains the compare method. And by overloading the compare method, we can define that on what basis we need to compare the values.

8. 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.

9. 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.

10. 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.

11. What do we get in the JDK file?

- **JDK**- For making java programs, we need some tools that are provided by JDK (Java Development Kit). JDK is the package that contains various tools, Compiler, Java Runtime Environment, etc.
- **JRE** - To execute the java program we need an environment. (Java Runtime Environment) JRE contains a library of Java classes + JVM. **What are JAVA Classes?** It contains some predefined methods that help Java programs to use that feature, build and execute. *For example* - there is a system class in java that contains the print-stream method, and with the help of this, we can print something on the console.
- **JVM** - (Java Virtual Machine) JVM is a part of JRE that executes the Java program at the end. Actually, it is part of JRE, but it is software that converts bytecode into machine-executable code to execute on hardware.

JDK

JRE

JVM

Development Tools+

Class Library +

12. 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.
Summary	JDK = (JRE) + Development tools	JRE = (JVM) + Libraries to execute the application	JVM = Runtime environment to execute Java byte code.

13. 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.

14. 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.

15. What are the different ways of threads usage?

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

```java
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**

```java
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.

16. What are the different types of Thread Priorities in Java? And what is the default priority of a thread assigned by JVM?

There are a total of 3 different types of priority available in Java.

MIN_PRIORITY: It has an integer value assigned with 1.
MAX_PRIORITY: It has an integer value assigned with 10.
NORM_PRIORITY: It has an integer value assigned with 5.

In Java, Thread with MAX_PRIORITY gets the first chance to execute. But the default priority for any thread is NORM_PRIORITY assigned by JVM.

17. What is the difference between the program and the process?

- A program can be defined as a line of code written in order to accomplish a particular task. Whereas the process can be defined as the programs which are under execution.
- A program doesn't execute directly by the CPU. First, the resources are allocated to the program and when it is ready for execution then it is a process.

18. What is the difference between the 'throw' and 'throws' keyword in java?

- The '**throw**' keyword is used to manually throw the exception to the calling method.
- And the '**throws**' keyword is used in the function definition to inform the calling method that this method throws the exception. So if you are calling, then you have to handle the exception.

Example -

```java
class Main {
  public static int testExceptionDivide(int a, int b) throws ArithmeticException{
    if(a == 0 || b == 0)
      throw new ArithmeticException();
    return a/b;
  }
  public static void main(String args[]) {
    try{
      testExceptionDivide(10, 0);
```

```
        }
    catch(ArithmeticException e){
        //Handle the exception
        }
    }
  }
}
```

Here in the above snippet, the method testExceptionDivide throws an exception. So if the main method is calling it then it must have handled the exception. Otherwise, the main method can also throw the exception to JVM.

And the method testExceptionDivide 'throws' the exception based on the condition.

19. 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.

20. Identify the output of the below java program and Justify your answer.

```java
class Main {
    public static void main(String args[]) {
        Scaler s = new Scaler(5);
    }
}
class InterviewBit{
    InterviewBit(){
        System.out.println(" Welcome to InterviewBit ");
    }
}
class Scaler extends InterviewBit{
```

```
    Scaler(){
        System.out.println(" Welcome to Scaler Academy ");
    }
    Scaler(int x){
        this();
        super();
        System.out.println(" Welcome to Scaler Academy 2");
    }
}
```

The above code will throw the compilation error. It is because the **super()** is used to call the parent class constructor. But there is the condition that super() must be the first statement in the block. Now in this case, if we replace **this()** with **super()** then also it will throw the compilation error. Because this() also has to be the first statement in the block. So in conclusion, we can say that we cannot use **this()** and **super()** keywords in the same block.

21. 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:

```java
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
```
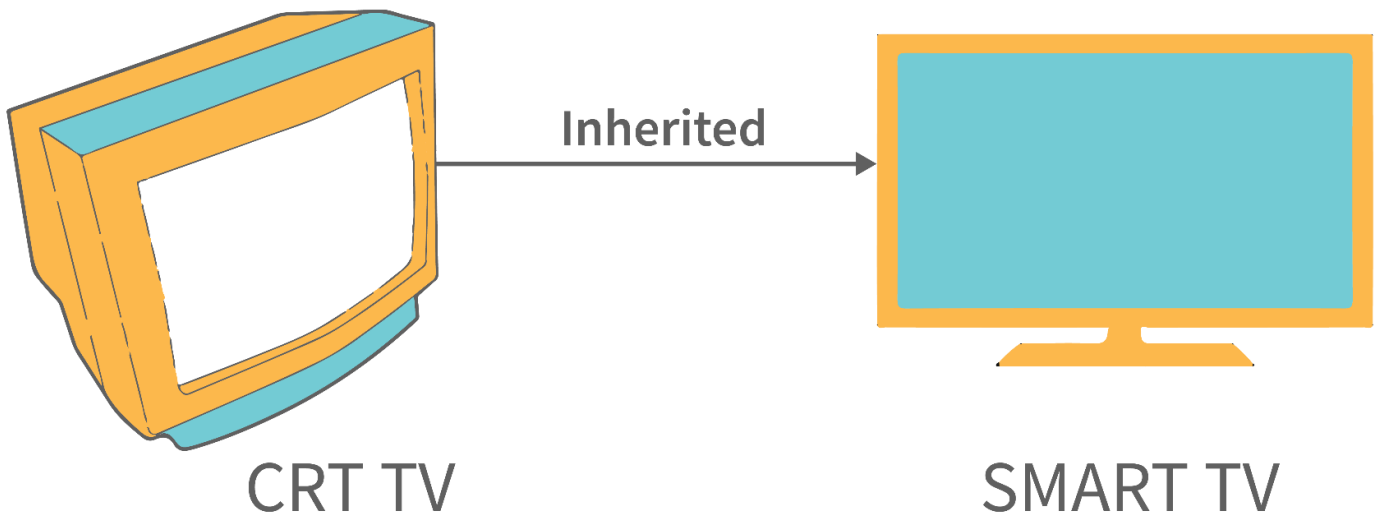
22. What is the 'IS-A ' relationship in OOPs java?

'IS-A' relationship is another name for inheritance. When we inherit the base class from the derived class, then it forms a relationship between the classes. So that relationship is termed an 'IS-A' Relationship.

Example - Consider a Television (Typical CRT TV). Now another Smart TV that is inherited from television class. So we can say that the Smart iv is also a TV. Because CRT TV things can also be done in the Smart TV.

Inherited

CRT TV

SMART TV

So here 'IS-A' Relationship formed. **[SmartTV 'IS-A' TV]**.

23. 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.

24. 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.

25. 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.

26. Consider the below program, identify the output, and also state the reason for that.

```java
public class Main{
public static void main(String[] args) {
 System.out.println(" Hello. Main Method. ");
}
public static void main(int[] args) {
 System.out.println(" Hello. Main Method2. ");
}
}
```

The output of the above program will be **Hello. Main Method**. This is because JVM will always call the main method based on the definition it already has. Doesn't matter how many main methods we overload it will only execute one main method based on its declaration in JVM.

27. Can we make the main() thread a daemon thread?

In java multithreading, the main() threads are always non-daemon threads. And there is no way we can change the nature of the non-daemon thread to the daemon thread.

28. 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.

29. What do you understand by Object Cloning and how do you achieve it in Java?
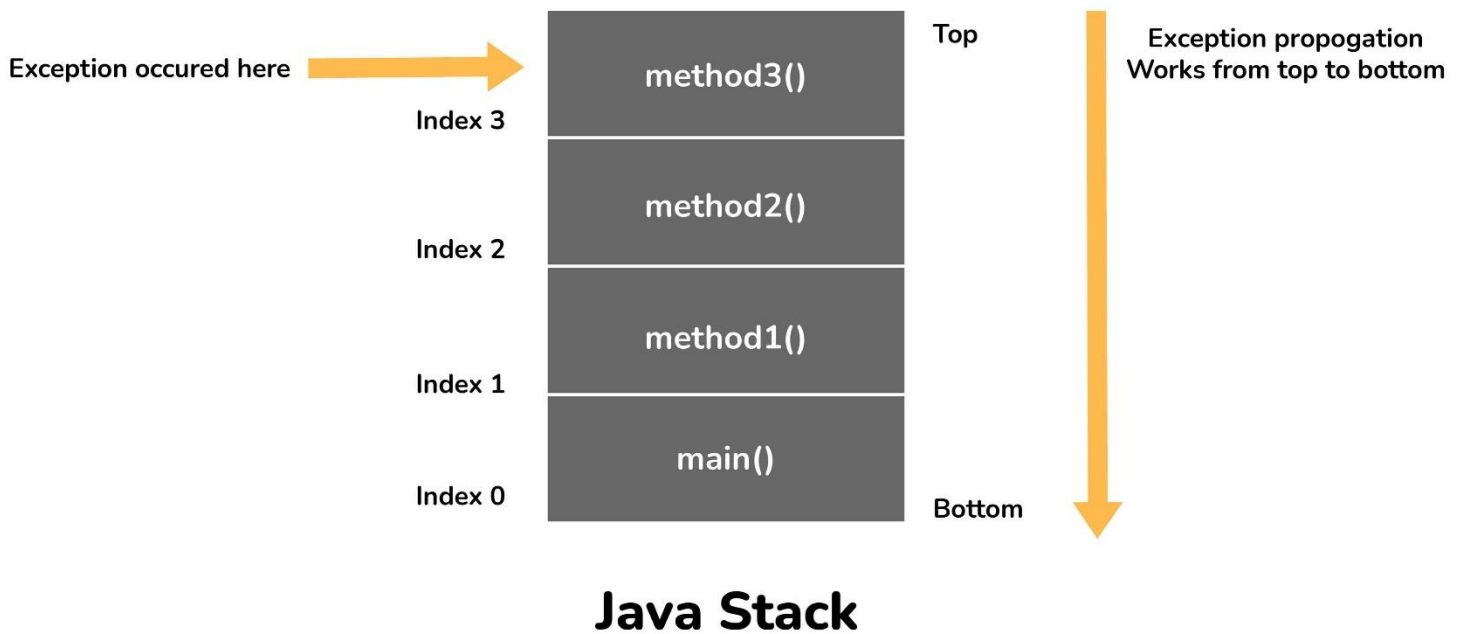
- It is the process of creating an exact copy of any object. In order to support this, a java class has to implement the Cloneable interface of java.lang package and override the clone() method provided by the Object class the syntax of which is:

```java
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.

30. 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

31. How do exceptions affect the program if it doesn't handle them?

Exceptions are runtime errors. Suppose we are making an android application with java. And it all works fine but there is an exceptional case when the application tries to get the file from storage and the file doesn't exist (This is the case of exception in java). And if this case is not handled properly then the application will crash. This will be a bad experience for users. This is the type of error that cannot be controlled by the programmer. But programmers can take some steps to avoid this so that the application won't crash. The proper action can be taken at this step.

32. 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.

33. 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
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.

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

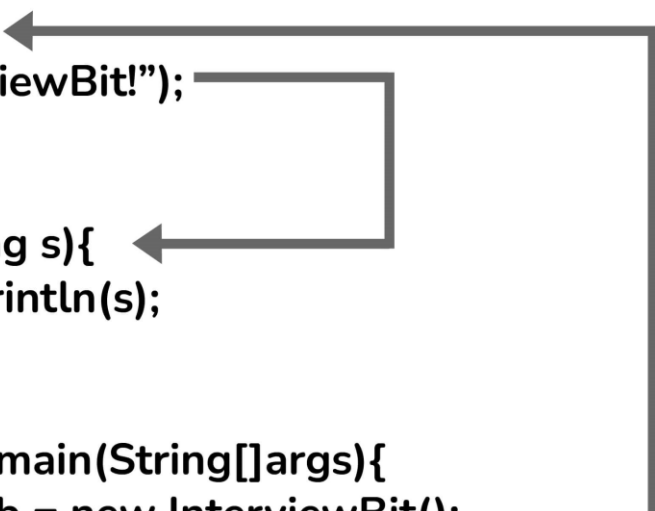Yes, the concept can be termed as constructor chaining and can be achieved using this().

```
public class InterviewBit{                    Constructor Chaining

    InterviewBit(){
        this("Hi InterviewBit!");
    }

    InterviewBit(String s){
        System.out.println(s);
    }

    public static void main(String[]args){
        InterviewBit ib = new InterviewBit();
    }
}
```
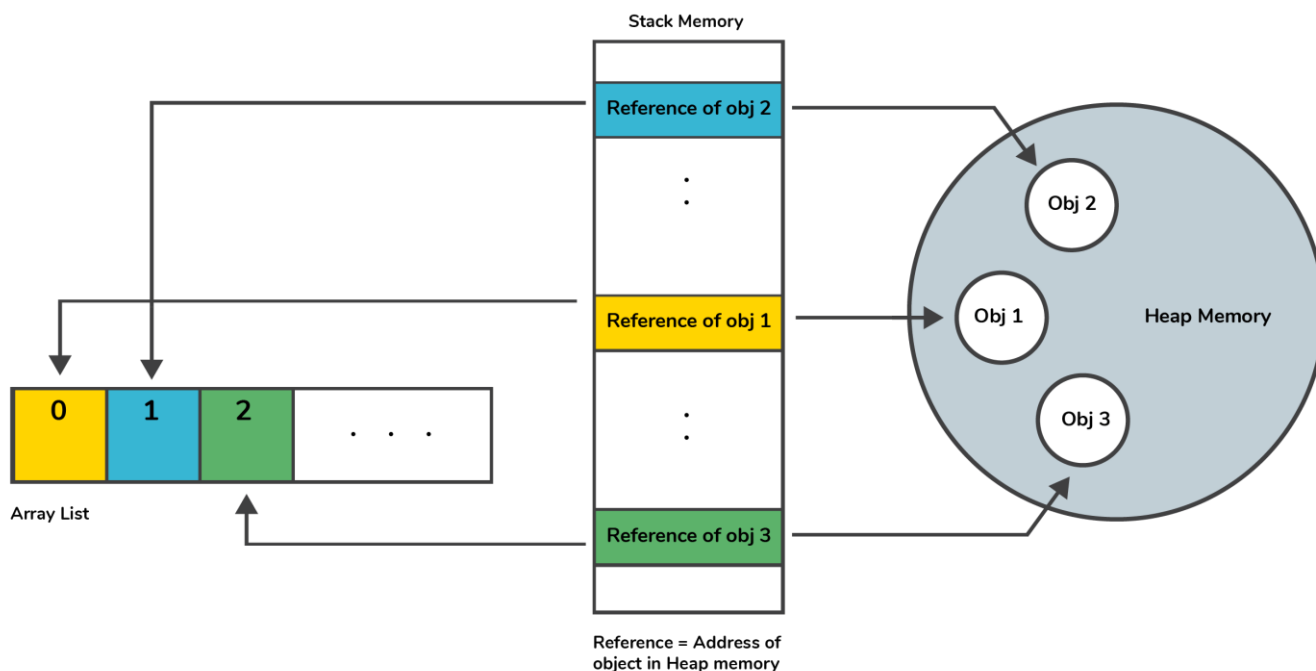
35. Contiguous memory locations are usually used for storing actual values in an array but not in ArrayList. Explain.

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.

Sorting of objects in Array List

Stack Memory

Reference of obj 2

Reference of obj 1

Reference of obj 3

Heap Memory

Obj 2

Obj 1

Obj 3

Reference = Address of object in Heap memory

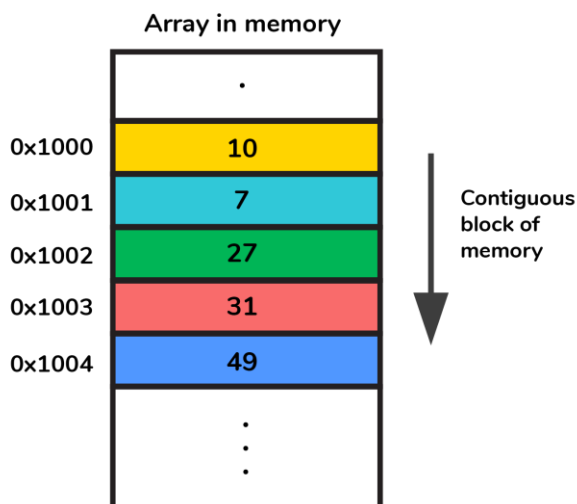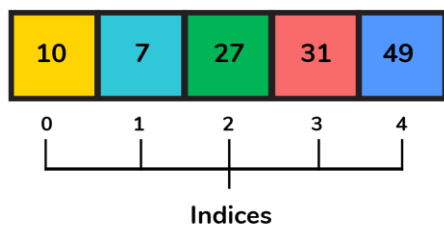| 0 | 1 | 2 | . . . |

Array List

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.

Sorting values in Array

How Array look like?

Array

| 10 | 7 | 27 | 31 | 49 |

0   1   2   3   4

Indices

Array in memory

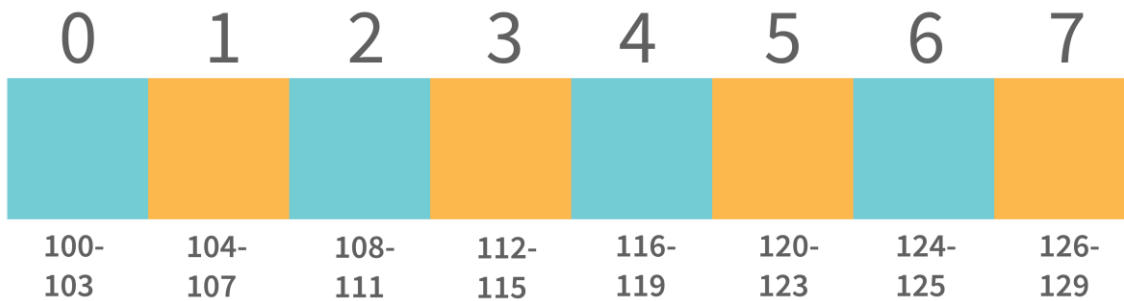| 0x1000 | 10 |
0x1001	7
0x1002	27
0x1003	31
0x1004	49

Contiguous block of memory

36. Why does the java array index start with 0?

It is because the 0 index array avoids the extra arithmetic operation to calculate the memory address.
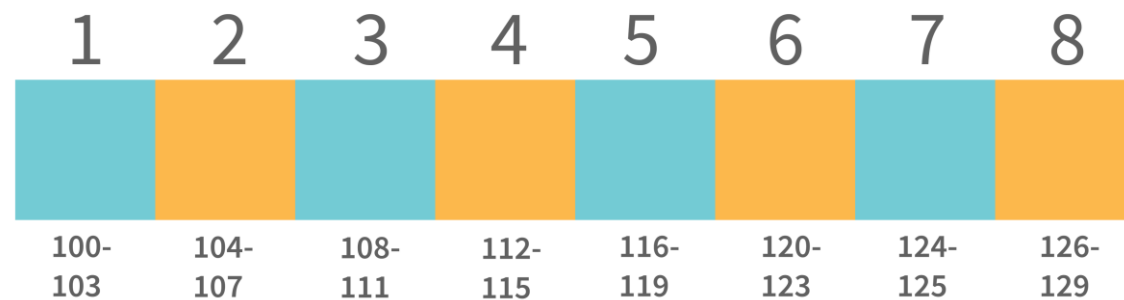
Example - Consider the array and assume each element takes 4-byte memory space. Then the address will be like this -

0	1	2	3	4	5	6	7
100-103	104-107	108-111	112-115	116-119	120-123	124-125	126-129

Now if we want to access index 4. Then internally java calculates the address using the formula -

[Base Address + (index * no_of_bytes)]. So according to this. The starting address of the index 4 will be - [100 + (4*4)] = 116. And exactly that's what the address is calculated.
Now consider the same with 1 index Array -

1	2	3	4	5	6	7	8
100-103	104-107	108-111	112-115	116-119	120-123	124-125	126-129

Now if we apply the same formula here. Then we get - 116 as the starting address of the 4th index. Which is wrong. Then we need to apply formula - [Base Address + ((index-1) * no_of_bytes)].
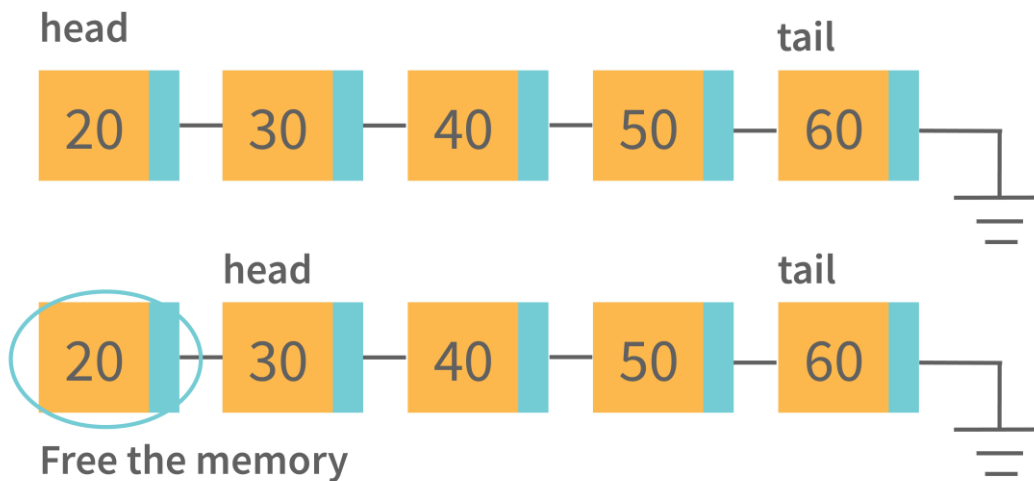
And for calculating this, an extra arithmetic operation has to be performed. And consider the case where millions of addresses need to be calculated, this causes complexity. So to avoid this,) the index array is supported by java.

37. Why is the remove method faster in the linked list than in an array?

In the linked list, we only need to adjust the references when we want to delete the element from either end or the front of the linked list. But in the array, indexes are used. So to manage proper indexing, we
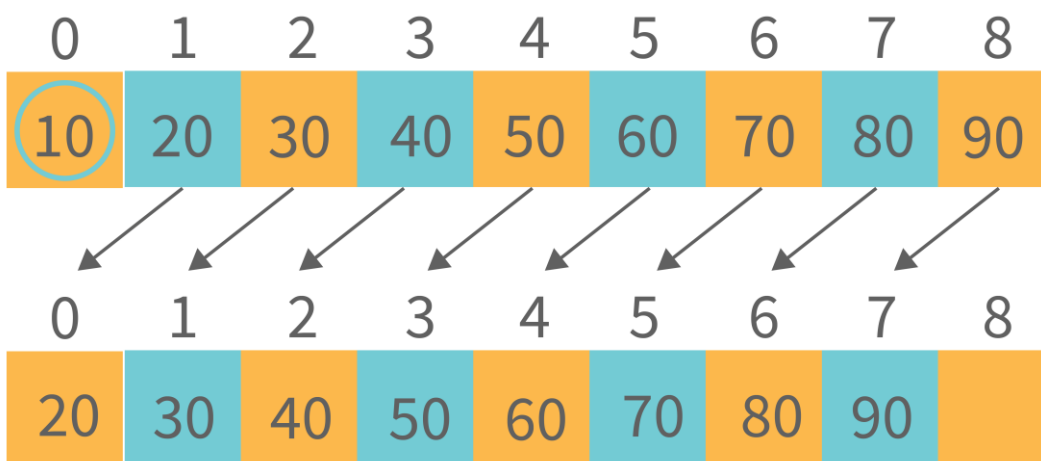
need to adjust the values from the array So this adjustment of value is costlier than the adjustment of references.

Example - To Delete from the front of the linked list, internally the references adjustments happened like this.

The only thing that will change is that the head pointer will point to the head's next node. And delete the previous node. That is the constant time operation.

Whereas in the ArrayList, internally it should work like this-

For deletion of the first element, all the next element has to move to one place ahead. So this copying value takes time. So that is the reason why removing in ArrayList is slower than LinkedList.

38. How many overloaded add() and addAll() methods are available in the List interface? Describe the need and uses.

There are a total of 4 overloaded methods for add() and addAll() methods available in List Interface. The below table states the description of all.

Return Type	Method Description
boolean	**add(Element e)**: This method is used for adding the element at the end of the List. The Datatype of the element is of any type it has been initially assigned with. It returns the boolean indicating successfully inserted or not.
void	**add(int index, Element e)**: This method is the overloaded version of add() method. In this, along with the element, the index is also passed to the method for the specific index the value needs to be inserted.
boolean	**addAll(Collection <extends ? Element > c)**: This method helps to add all elements at the end of collections from the list received in the parameter. It contains an iterator that helps to iterate the list and add the elements to the collection.
boolean	**addAll(int index, Collection <extends ? Element > c)**: This is the overloaded method for addAll() method. In this along with the list, we can pass the specified index from which the list elements need to be added.

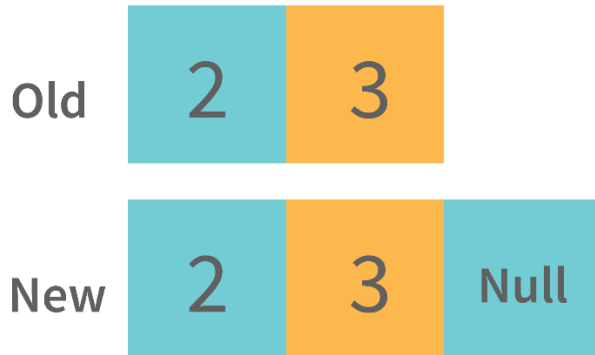## 39. How does the size of ArrayList grow dynamically? And also state how it is implemented internally.

ArrayList is implemented in such a way that it can grow dynamically. We don't need to specify the size of ArrayList. For adding the values in it, the methodology it uses is -

1. Consider initially that there are 2 elements in the ArrayList. **[2, 3]**.

2. If we need to add the element into this. Then internally what will happen is-

- ArrayList will allocate the new ArrayList of Size (current size + half of the current size). And add the old elements into the new. Old - [2, 3],   New - [2, 3, null].

Old **2** **3**

New **2** **3** **Null**

- Then the new value will be inserted into it. [2, 3, 4, null]. And for the next time, the extra space will be available for the value to be inserted.
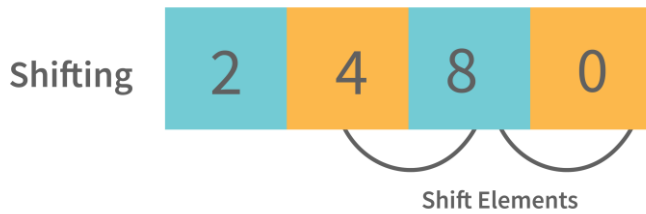
New 2 3 4 Null

3. This process continues and the time taken to perform all of these is considered as the amortized constant time.

This is how the ArrayList grows dynamically. And when we delete any entry from the ArrayList then the following steps are performed -

1. It searches for the element index in the array. Searching takes some time. Typically it's O(n) because it needs to search for the element in the entire array.

Search 3 2 (3) 4 8
Found

2. After searching the element, it needs to shift the element from the right side to fill the index.

So this is how the elements are deleted from the ArrayList internally. Similarly, the search operations are also implemented internally as defined in removing elements from the list (searching for elements to delete).

Java Interview Questions for Experienced

1. 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;
 }
}
```

2. What is the difference between '>>' and '>>>' operators in java?

These 2 are the bitwise right shift operators. Although both operators look similar. But there is a minimal difference between these two right shift operators.

- **'>>' Bitwise Right Shift Operator**- This operator shifts each bit to its right position. And this maintains the signed bit.
- **'>>>' Bitwise Right Shift Operator with trailing zero**- This operator also shifts each bit to its right. But this doesn't maintain the signed bit. This operator makes the Most significant bit to 0.

Example- Num1 = 8, Num2 = -8.

So the binary form of these numbers are -

Num1 = 00000000 00000000 00000000 00001000
Num2 = 11111111 11111111 11111111 11111000

'>>' Operator : 8 >> 1 (Shift by one bit) :

Num1 = 00000000 00000000 00000000 00000100
Num2 = 11111111 11111111 11111111 11111100

'>>>' Operator : 8 >>> 1 (Shift by one bit) =

Num1 = 00000000 00000000 00000000 00000100
Num2 = 01111111 11111111 11111111 11111100

3. What are Composition and Aggregation? State the difference.

Composition, and Aggregation help to build (Has - A - Relationship) between classes and objects. But both are not the same in the end. **Let's understand with the help of an example.**

- Consider the University as a class that has some departments in it. So the university will be the container object. And departments in it will contain objects. Now in this case, if the container object destroys then the contained objects will also get destroyed automatically. So here we can say that there is a strong association between the objects. So this Strong Association is called **Composition**.
- Now consider one more example. Suppose we have a class department and there are several professors' objects there in the department. Now if the department class is destroyed then the professor's object will become free to bind with other objects. Because container objects (Department) only hold the references of contained objects (Professor's). So here is the weak association between the objects. And this weak association is called **Aggregation**.
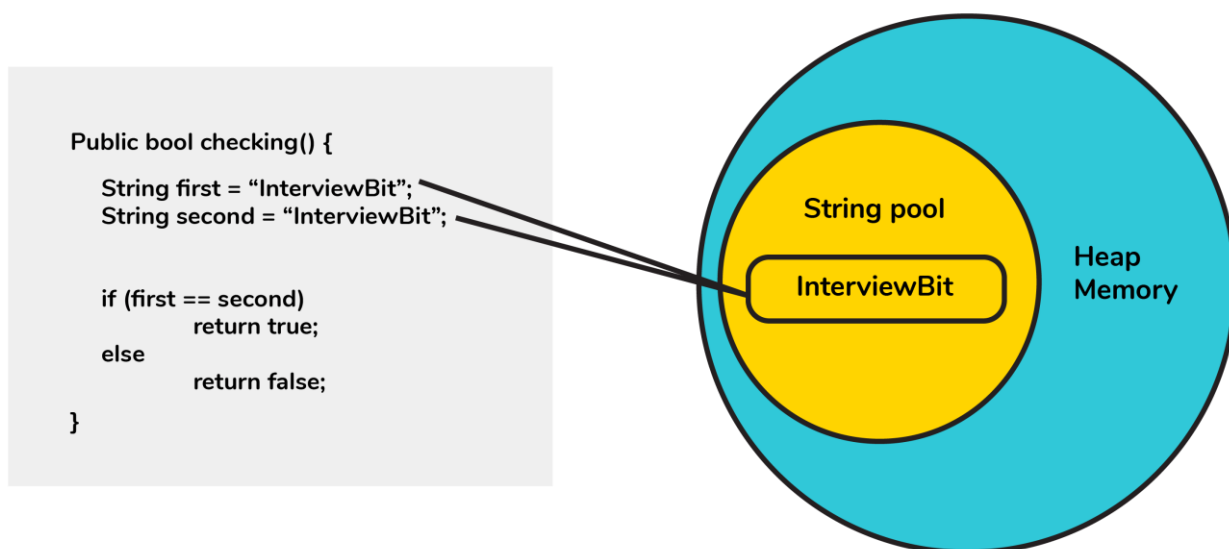
4. How is the creation of a String using new() different from that of a literal?

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.

String Pool by means of assignment operator

```
Public bool checking() {

    String first = "InterviewBit";
    String second = "InterviewBit";


    if (first == second)
            return true;
    else
            return false;

}
```

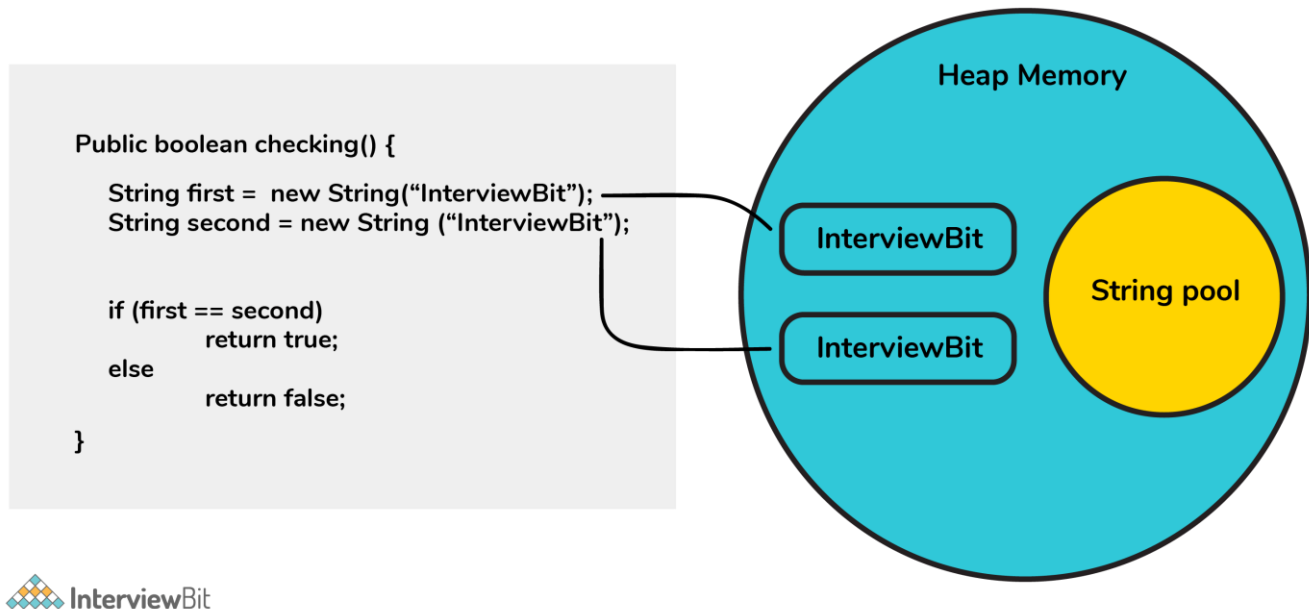String pool

Heap Memory

InterviewBit

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

InterviewBit

5. How is the 'new' operator different from the 'newInstance()' operator in java?

Both 'new' and 'newInstance()' operators are used to creating objects. The difference is- that when we already know the class name for which we have to create the object then we use a new operator. But suppose we don't know the class name for which we need to create the object, Or we get the class name from the command line argument, or the database, or the file. Then in that case we use the 'newInstance()' operator.

The '**newInstance()**' keyword throws an exception that we need to handle. It is because there are chances that the class definition doesn't exist, and we get the class name from runtime. So it will throw an exception.

6. 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"));
}
```

7. 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;
    }
}
```

Without Synchronization

```
package anonymous;
public class Counting {

    private int increase_counter;

    public int increase() {

        increase_counter = increase_counter + 1;

        return increase_counter;

    }
}
```
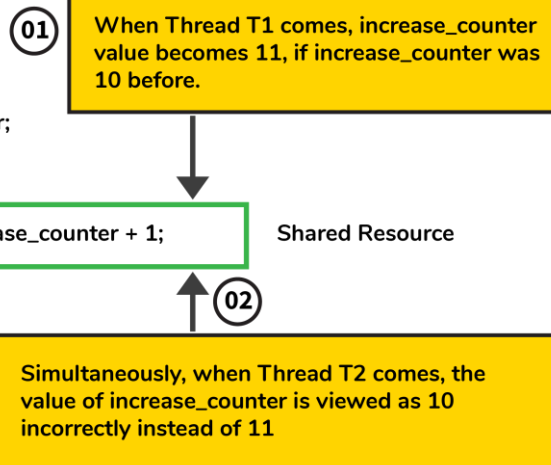
01 When Thread T1 comes, increase_counter value becomes 11, if increase_counter was 10 before.

Shared Resource

02 Simultaneously, when Thread T2 comes, the value of increase_counter is viewed as 10 incorrectly instead of 11

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;
    }
}
```

With Synchronization

```
public class Counting {

    private int increase_counter;

    public synchronized int increase() {
        increase_counter = increase_counter + 1;
        return increase_counter;
    }

}
```
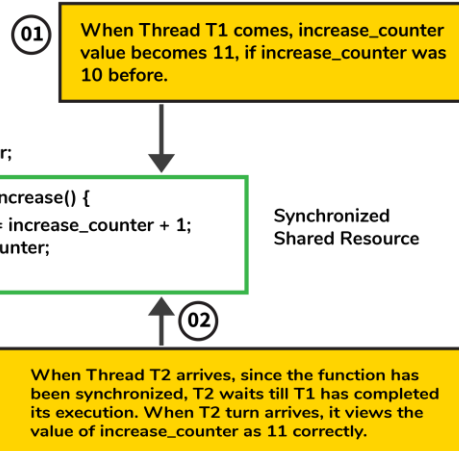
01 When Thread T1 comes, increase_counter value becomes 11, if increase_counter was 10 before.

Synchronized Shared Resource

02 When Thread T2 arrives, since the function has been synchronized, T2 waits till T1 has completed its execution. When T2 turn arrives, it views the value of increase_counter as 11 correctly.

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.

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

```java
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:

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

9. What will be the output of the below java program and define the steps of Execution of the java program with the help of the below code?

```java
class InterviewBit{
  int i;
  static int j;
  {
    System.out.println(" Instance Block 1. Value of i = "+i);
  }
  static{
    System.out.println(" Static Block 1. Value of j = "+j);
    method_2();
  }
  {
    i = 5;
  }
  static{
    j = 10;
  }
  InterviewBit(){
    System.out.println(" Welcome to InterviewBit ");
  }
  public static void main(String[] args){
    InterviewBit ib = new InterviewBit();
  }
  public void method_1(){
    System.out.println(" Instance method. ");
  }
  static{
```

```
        System.out.println(" Static Block 2. Value of j = "+j);
    }
    {
        System.out.println(" Instance Block 2. Value of i = "+i);
        method_1();
    }
    public static void method_2(){
        System.out.println(" Static method. ");
    }
}
```

The Output we get by executing this program will be

Static Block 1. Value of j = 0
Static method.
Static Block 2. Value of j = 10
Instance Block 1. Value of i = 0
Instance Block 2. Value of i = 5
Instance method.
Welcome to InterviewBit

This is a java tricky interview question frequently asked in java interviews for the experienced. The output will be like this because, when the java program is compiled and gets executed, then there are various steps followed for execution. And the steps are -

- Identification of Static Members from top to bottom.
- Execution of Static variable assignment and a Static block from top to bottom.
- Execution of the main method.
- Identification of Instance Members from top to bottom.
- Execution of Instance variable assignment and Instance block from top to bottom.
- Execution of Constructor.

In above steps from 4 to 6, will be executed for every object creation. If we create multiple objects then for every object these steps will be performed.

Now from the above code, the execution will happen like this -

1. In the step of identification of static members. It is found that -

- static int j.
- static block.
- main method.
- static method_2.

During identification, the JVM will assign the default value in the static int j variable. Then it is currently in the state of reading and indirectly writing. Because the original value is not assigned.

2. In the next step, it will execute the static block and assign the value in static variables.

- First static block it will print and because execution from top to bottom and original value in j is not assigned. So it will print the default value of 0.
- After executing static block 1. It will execute the static method_1 because it is called from the static block 1.
- Then it will assign the original value of 5 in the j variable. And executes the remaining static block.

3. Now it will execute the main method. In which it will create an object for the class InterviewBit. And then the execution of instances will happen.

4. Identify the instance variables and blocks from top to bottom.

- int i.
- Instance block 1.
- Instance method_1.

Like a static variable, the instance variable also has been initialized with the default value 0 and will be in the state of reading and writing indirectly.

5. It will execute the instance methods and assign the original value to the instance variable.

- Prints the Instance block 1. And the current value of i is not assigned till now, so it will print 0.
- Assign the original value to i. Then print instance block 2. And after that instance method will be called and printed because it is being called in the instance block.

6. And at the last step, the constructor will be invoked and the lines will be executed in the constructor.

This is how the java program gets executed.

10. Define System.out.println().

System.out.println() is used to print the message on the console. **System** - It is a class present in **java.lang package**. Out is the static variable of type PrintStream class present in the **System class. println()** is the method present in the PrintStream class.

So if we justify the statement, then we can say that if we want to print anything on the console then we need to call the **println()** method that was present in PrintStream class. And we can call this using the output object that is present in the System class.
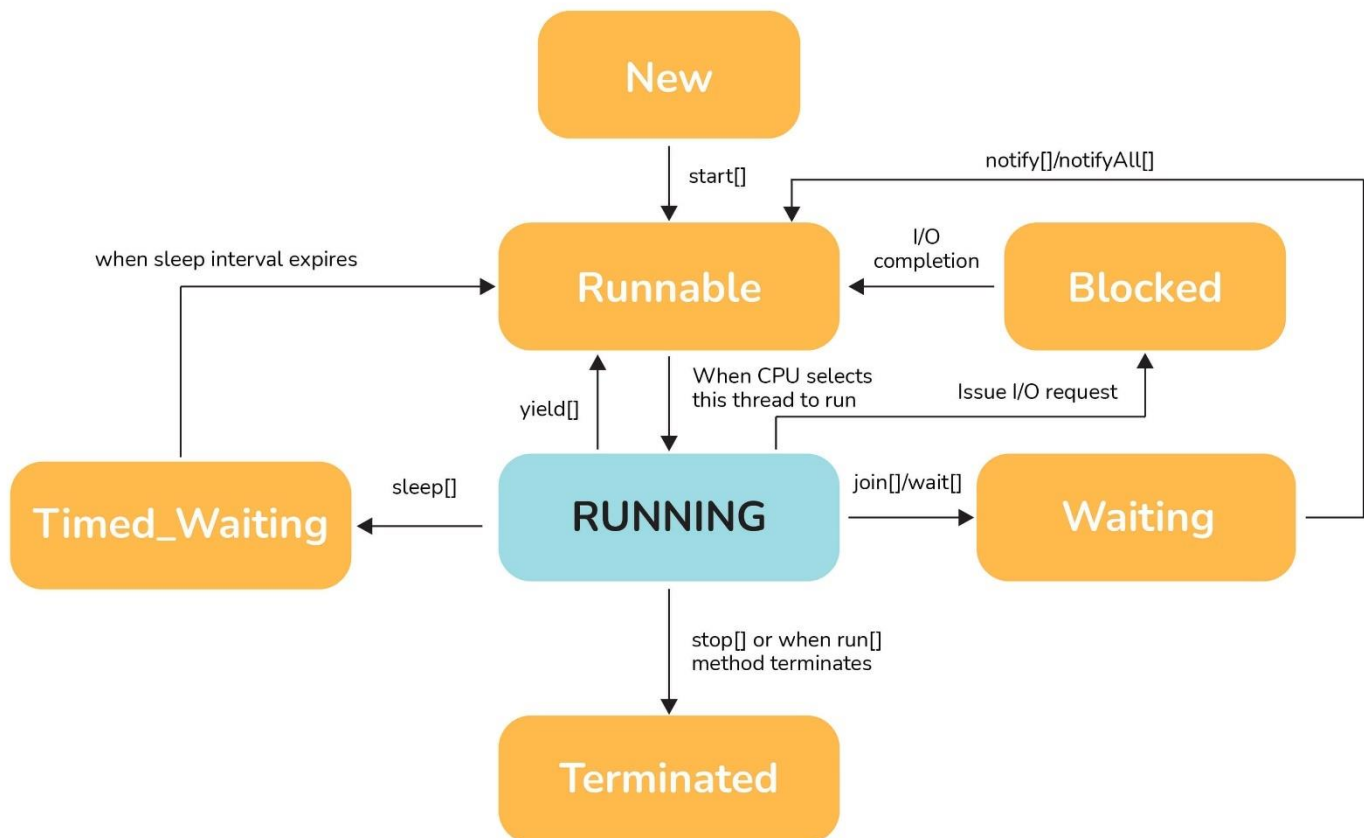
11. Can you explain the Java thread lifecycle?

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.

12. 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.

13. 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.

14. 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.

15. 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.

16. 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.

17. Explain the term "Double Brace Initialisation" in Java?

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

```java
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.

18. 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.

19. 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.

20. 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.

```java
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.

```java
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 referred by any reference variable. Therefore, it  is eligible for GC */
  }
}
```

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.

```java
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
    */
  }
}
```

21. In the below Java Program, how many objects are eligible for garbage collection?

```
class Main{
  public static void main(String[] args){
    int[][] num = new int[3][];
    num[0] = new int[5];
    num[1] = new int[2];
    num[2] = new int[3];

    num[2] = new int[5];
    num[0] = new int[4];
    num[1] = new int[3];

    num = new int[2][];
  }
}
```

In the above program, a total of 7 objects will be eligible for garbage collection. Let's visually understand what's happening in the code.

In the above figure on line 3, we can see that on each array index we are declaring a new array so the reference will be of that new array on all the 3 indexes. So the old array will be pointed to by none. So these three are eligible for garbage collection. And on line 4, we are creating a new array object on the older reference. So that will point to a new array and older multidimensional objects will become eligible for garbage collection.

22. What is the best way to inject dependency? Also, state the reason.

There is no boundation for using a particular dependency injection. But the recommended approach is -

Setters are mostly recommended for optional dependencies injection, and constructor arguments are recommended for mandatory ones. This is because constructor injection enables the injection of values into immutable fields and enables reading them more easily.

23. How we can set the spring bean scope. And what supported scopes does it have?

A scope can be set by an annotation such as the @Scope annotation or the "scope" attribute in an XML configuration file. Spring Bean supports the following five scopes:

- Singleton
- Prototype
- Request
- Session
- Global-session

24. What are the different categories of Java Design patterns?

Java Design patterns are categorized into the following different types. And those are also further categorized as

Structural patterns:

- Adapter
- Bridge
- Filter
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behavioral patterns:

- Interpreter
- Template method/ pattern
- Chain of responsibility
- Command pattern
- Iterator pattern
- Strategy pattern
- Visitor pattern

J2EE patterns:

- MVC Pattern

- Data Access Object pattern
- Front controller pattern
- Intercepting filter pattern
- Transfer object pattern

Creational patterns:

- Factory method/Template
- Abstract Factory
- Builder
- Prototype
- Singleton

25. What is a Memory Leak? Discuss some common causes of it.

The Java Garbage Collector (GC) typically removes unused objects when they are no longer required, but when they are still referenced, the unused objects cannot be removed. So this causes the memory leak problem. **Example** - Consider a linked list like the structure below -

In the above image, there are unused objects that are not referenced. But then also Garbage collection will not free it. Because it is referencing some existing referenced object. So this can be the situation of memory leak.

Some common causes of Memory leaks are -

- When there are Unbounded caches.
- Excessive page swapping is done by the operating system.
- Improper written custom data structures.
- Inserting into a collection object without first deleting it.
  etc.

26. Assume a thread has a lock on it, calling the sleep() method on that thread will release the lock?

A thread that has a lock won't be released even after it calls sleep(). Despite the thread sleeping for a specified period of time, the lock will not be released.

Java Programming Interview Questions

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

```
/*
 * Java program to check if a given inputted string is palindrome or not using recursion.
 */
import java.util.*;
```

```java
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.length() - 1));
  }
}
```

2. Write a Java Program to print Fibonacci Series using Recursion.

```java
class InterviewBit {
  public static void printFibonacci(int val_1, int val_2, int num){
    //Base Case
    if(num == 0)
      return;

    //Printing the next Fibonacci number
    System.out.print( val_1 + val_2 + " ");

    //Recursively calling for printing Fibonacci for remaining length
    printFibonacci(val_2, val_1+val_2, --num);
  }
  public static void main(String args[]) {
    System.out.println(" *** Fibonacci Series *** ");

    //Printing the first two values
    System.out.print("0 1 ");

    //Calling Method to print the fibonacci for length 10
    printFibonacci(0, 1, 10);
  }
}
```

In the above code, we are printing the base 2 Fibonacci values 0 and 1. And then based on the length of Fibonacci to be printed, we are using the helper function to print that.

3. 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.

```java
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);
 }
 }
}
```

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

```java
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);
  }
}
```

5. 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.

```java
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;
    }
}
```

6. 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");
        }
    }
}
```

7. Write a Java program to create and throw custom exceptions.

```
class InterviewBit {
    public static void main(String args[]) throws CustomException {

        // Throwing the custom exception be passing the message
        throw new CustomException(" This is my custom Exception ");
    }
```

```
}
//Creating Custom Exception Class
class CustomException extends Exception{
    //Defining Constructor to throw exception message
    public CustomException(String message){
        super(message);
    }
}
```

We have created the exception class named with CustomException and called the base exception constructor with the error message that we want to print. And to avoid handling exceptions in the main method, we have used the throws keyword in the method declaration.

8. Write a Java program to reverse a string.

```
class InterviewBit{
    public static void main(String[] args){
        //Input String
        String str = "Welcome to InterviewBit";

        //Pointers.
        int i = 0, j = str.length()-1;

        //Result character array to store the reversed string.
        char[] revString = new char[j+1];

        //Looping and reversing the string.
        while(i < j){
            revString[j] = str.charAt(i);
            revString[i] = str.charAt(j);
            i++;
            j--;
        }
        //Printing the reversed String.
        System.out.println("Reversed String = " + String.valueOf(revString));
    }
}
```

In the above code, we are storing the last character from the string to the first and the first value to the last in the output character array. And doing the same thing in the loop for the remaining 2nd to n-1 characters. This is how the string will be reversed.

9. Write a Java program to rotate arrays 90 degree clockwise by taking matrices from user input.

```
mport java.util.Scanner;
public class InterviewBit
{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int no;
        System.out.print("Enter size of Array : ");
        no = sc.nextInt();
        int[][] a = new int[no][no];
```

```java
        System.out.print("Enter  "+ no*no+" Element Array : ");

        for(int i = 0; i<no; i++){
            for(int j = 0; j<no; j++){
                a[i][j] = sc.nextInt();
            }
        }
        System.out.print("\nArray Before Rotation\n\n");
        for(int i = 0; i<no; i++){
            for(int j = 0; j<no; j++){
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }

        System.out.println("\n");
        //Rotation

        //Transpose
        for(int i = 0; i < no; i++){
            for(int j = i; j < no; j++){
                int temp = a[i][j];
                a[i][j] = a[j][i];
                a[j][i] = temp;
            }
        }

        //Reverse Each row
        for(int i = 0; i < no; i++){
            int l, j;
            for(j = 0, l = no -1; j < l; j++){
                int temp = a[i][j];
                a[i][j] = a[i][l];
                a[i][l] = temp;
                l--;
            }
        }

        System.out.println("Array After Rotation - \n");

        for(int i = 0; i<no; i++){
            for(int j = 0; j<no; j++){
                System.out.print(a[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

In the above code, for rotating the matrix to 90 degrees we are first transposing the matrix so the row becomes the column. And after that, we are reversing each row in the matrix. So this is how the matrix got rotated.

10. Write a java program to check if any number given as input is the sum of 2 prime numbers.

Example :

Input - 18

Output -

18 = 13 + 5
18 = 11 + 7

```java
public class InterviewBit
{
    // Method to Check Prime Number
    private static int check_prime(int num){
        int flag = 0;
        for(int i = 2; i<=num/2; i++){
            if(num%i == 0){
                flag = 1;
                return 1;
            }
        }
        if(flag == 0)
            return 0;
        return 1;
    }
    // Method to get print the prime sum
    private static void find(int num){
        for(int i = 2; i <= num/2; i++){
            if(check_prime(i) == 0){
                if(check_prime(num-i) == 0)
                    System.out.println(num + " = "+ (num-i) + " "+ i);
            }
        }
    }

            public static void main(String[] args) {
                    find(18);
            }
}
```

In the above code, for any number **n**, we find all the 2 pairs of numbers that are added together resulting in **n**. And each checking number if it is prime. If it is prime then we are printing that.

11. Write a Java program for solving the Tower of Hanoi Problem.

```java
public class InterviewBit
{
    //Recursive Method for Solving the Tower of hanoi.
    private static void TOH(char source, char auxiliary, char destination, int numOfDisk){
            if (numOfDisk > 0){
                    TOH(source, destination, auxiliary, numOfDisk-1);
                    System.out.println("Move 1 disk from "+source+" to "+destination+" using "+auxiliary+".");
                    TOH(auxiliary, source, destination, numOfDisk-1);
            }
    }
            public static void main(String[] args) {
                    TOH('A','B','C', 3);
```

```
        }
}
```

In the above code we are first moving the **n-1** disk from Tower **A** to Tower **B**, then moving that **nth** disk from Tower **A** to Tower **C**, and finally, the remaining **n-1** disk from Tower **B** to Tower **C**. And we are doing this recursively for the **n-1** disk.

12. Implement Binary Search in Java using recursion.

```java
public class Main
{
    //Recursive method for binary search
    private static boolean binarySearch(int[] arr, int low, int high, int key){

        //Calculating Mid.
        int mid = (low + high)/2;

        //Base Case.
        if(low > high)
            return false;

        //Checking if the key is found in the middle.
        if(arr[mid] == key)
            return true;

        //Searching on the left half if a key exists there.
        if(key < arr[mid])
            return binarySearch(arr, low, mid-1, key);

        //Searching on the other half otherwise.
        return binarySearch(arr, mid+1, high, key);
    }
public static void main(String[] args) {

    int[] arr = {2, 5, 9, 13, 17, 21, 30};
    if(binarySearch(arr, 0, (arr.length-1), 30))
        System.out.println(" Element Found. ");
    else
        System.out.println(" Element not Found.");
}
}
```

In the above code, we are finding the middle element each time and checking if the element is in the middle or not. If it is not, then we check on which side from the middle it exists. And Recursively searching on the particular subarray. So this way we are reducing the search space by 2 every time. So the search time is very low.
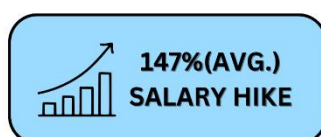
Conclusion

1. 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 has ever-growing usage in the software community.

Learn Coding From Scratch

CODING SCRATCH

100 % SUCCESS RATE	CAREER SERVICES	100 % PLACEMENT ASSISTANCE
LEARN FROM SCRATCH	147%(AVG.) SALARY HIKE	23 LPA (AVG.)CTC

NOTES GALLERY

☝JOIN For Freshers Notes 📌📚📖- ❗
https://t.me/codingbugs ☝CODING RESOURCE
HANDWRITTEN NOTES & INTERVIEW SERIES📖 Admin:...

Telegram

Jobs - Internship Official WhatsApp Channel
JOIN FOR JOBS UPDATE
whatsapp.com

EXPLORE MORE

+91 8800373756

www.codingscratch.net