Object Oriented Programming Questions & Answer

1. Write a java code to check whether the given number is Palindrome or not.

```java
import java.util.Scanner;


public class PalindromeChecker {
    public static void main(String[] args) {
        System.out.print("Enter your Number:");
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt(); // Change this number to check different inputs

        if (isPalindrome(number)) {
            System.out.println(number + " is a palindrome.");
        } else {
            System.out.println(number + " is not a palindrome.");
        }
    }

    public static boolean isPalindrome(int number) {
        int reversedNumber = 0;
        int originalNumber = number;

        while (number != 0) {
            int digit = number % 10;
            reversedNumber = reversedNumber * 10 + digit;
            number /= 10;
        }

        return originalNumber == reversedNumber;
    }
}
```

2. Sort the following array using java: 23 14 10 21 60 6.

```java
public class BubbleSort {
    public static void main(String[] args) {
        int[] array = { 23, 14, 10, 21, 60, 6 };

        System.out.println("Original Array: ");
        printArray(array);

        bubbleSort(array);

        System.out.println("\nSorted Array: ");
        printArray(array);
    }
    public static void bubbleSort(int[] array) {
        int n = array.length;
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - i - 1; j++) {
                if (array[j] > array[j + 1]) {
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                    swapped = true;
                }
            }

            if (!swapped) {
                break; // Array is already sorted, no need to continue
            }
        }
    }
    public static void printArray(int[] array) {
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i] + " ");
        }
    }
}
```

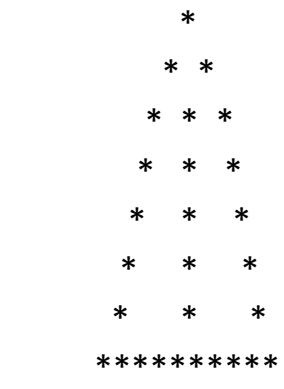
3. Write down a java code to find out the factorial value of a number using recursion.

```java
import java.util.Scanner;

// Factorial Using Recursion
public class Code36_FactorialRecursion {
    public static int factorial(int n) {
        if (n == 0)
            return 1;
        else
            return (n * factorial(n - 1));
    }

    public static void main(String args[]) {
        int i, fact = 1;
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your number for factorial!: ");
        int number = sc.nextInt();// It is the number to calculate
factorial
        fact = factorial(number);
        System.out.println("Factorial " + number + "! is: " + fact);
    }
}
```

4. Star Pattern



```
public class StarPyramid {
    public static void main(String[] args) {
        int n = 20;
        for (int i = 1; i <= n; i++) {
            for (int j = i; j < n; j++) {
                System.out.print(" ");
            }
            if (i == 1) {
                System.out.println("*");
            }
            if (i == 2) {
                System.out.println("***");
            }
            if (i > 2 && i < n) {
                System.out.print("*");
                for (int k = 0; k < i - 2; k++) {
                    System.out.print(" ");
                }
                System.out.print("*");
                for (int k = 0; k < i - 2; k++) {
                    System.out.print(" ");
                }
                System.out.println("*");

            }
            if (i == n) {
                for (int k = 0; k < (i * 2) - 1; k++) {
                    System.out.print("*");
                }
            }
        }
    }
}
```

5. **Differentiate between static polymorphism and Dynamic polymorphism in java.**

Ans:

In Java, polymorphism refers to the ability of an object to take on many forms. There are two types of polymorphism: static polymorphism (also known as compile-time polymorphism) and dynamic polymorphism (also known as runtime polymorphism).

- Static Polymorphism (Compile-time Polymorphism):

Static polymorphism occurs at compile-time and is achieved through method overloading and operator overloading. It allows different methods or operators to have the same name but with different parameters or behaviors. The decision about which method or operator to execute is made by the compiler based on the number, types, and order of arguments.

Example of static polymorphism using method overloading:

```
public class Calculator {
   public int add(int a, int b) {
      return a + b;
   }
   public int add(int a, int b, int c) {
      return a + b + c;
   }
}
```

In this example, the Calculator class has two add methods with different numbers of parameters. Depending on how the method is called, the appropriate add method is selected at compile-time.

- Dynamic Polymorphism (Runtime Polymorphism):

Dynamic polymorphism occurs at runtime and is achieved through method overriding. It allows a subclass to provide its own implementation of a method that is already defined in its superclass. The decision about which method to execute is made by the JVM based on the actual object type at runtime.

Example of dynamic polymorphism using method overriding:

```
public class Animal {
   public void makeSound() {
      System.out.println("Animal makes a sound");
   }
}
```

```
public class Cat extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Cat meows");
    }
}

public class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}
```

In this example, the Animal class has a method makeSound(), which is overridden in the Cat and Dog classes. Depending on the actual type of the object, the appropriate makeSound() method is executed at runtime.

To summarize:

Static polymorphism is resolved at compile-time and is achieved through method overloading and operator overloading.
Dynamic polymorphism is resolved at runtime and is achieved through method overriding, allowing different implementations of a method in a superclass and its subclasses.
Note: Polymorphism is one of the fundamental concepts of object-oriented programming and is supported by many programming languages, including Java.

6. **What is immutable Class...? What should we use to make a class immutable?**

Ans:

An immutable class in Java is a class whose objects cannot be modified after they are created. Once an object of an immutable class is created, its state (i.e., the values of its fields) cannot be changed. Immutable classes provide several benefits, such as thread safety, simplified reasoning about the code, and improved security.

To make a class immutable in Java, you should follow these guidelines:

1. Declare the class as final: By declaring the class as `final`, you prevent it from being subclassed, ensuring that its behavior and state cannot be modified by any subclasses.

2. Declare the fields as final: Declare all the fields in the class as `final`. This ensures that their values cannot be changed once they are initialized in the constructor.

3. Make the fields private: Encapsulate the fields by making them private, ensuring that their values can only be accessed and modified through controlled methods.

4. Don't provide setter methods: Avoid providing any setter methods that allow modifying the fields of the class.

5. Ensure no mutable objects are shared: If your class contains references to mutable objects, make sure to perform a defensive copy of those objects when returning or accepting them as parameters. This prevents external modification of the internal state of the immutable class.

6. Don't override non-final methods: Avoid overriding non-final methods in the immutable class to prevent any behavior modification.

7. Use a copy constructor or factory methods for initialization: Instead of providing a public constructor that directly initializes the fields, consider using a copy constructor or static factory methods. These methods can perform any necessary validations or defensive copying while creating the immutable object.

By following these guidelines, you can design an immutable class in Java. Immutable classes are commonly used for representing values that should not change once they are created, such as String, Integer, and LocalDate, among others.

7. What is Abstraction?

Ans:

Abstraction is an OOPS concept to construct the structure of the real world objects. During this construction only the general states and behaviors are taken and more specific states and behaviors are left aside for the implementers.

8. What is Encapsulation?

Ans:

Encapsulation is an OOPS concept to create and define the permissions and restrictions of an object and its member variables and methods. A very simple example to explain the concept is to make the member variables of a class private and providing public getter and setter methods. Java provides four types of access level modifiers: public, protected, no modifier and private.

9. What is the difference between Abstraction and Encapsulation?

Ans:

"Program to interfaces, not implementations" is the principle for Abstraction and "Encapsulate what varies" is the OO principle for Encapsulation.

Abstraction provides a general structure of a class and leaves the details for the implementers. Encapsulation is to create and define the permissions and restrictions of an object and its member variables and methods.

Abstraction is implemented in Java using interface and abstract class while Encapsulation is implemented using four types of access level modifiers: public, protected, no modifier and private.

10. What is Polymorphism?

Ans:

Polymorphism is the occurrence of something in various forms. Java supports various forms of polymorphism like polymorphic reference variables, polymorphic method, polymorphic return types and polymorphic argument types.

11. What is Inheritance?

Ans:

A subclass can inherit the states and behaviors of its super class is known as inheritance.

12. What is a Class?

Ans: A class is the specification or template of an object.

13. What is an Object?

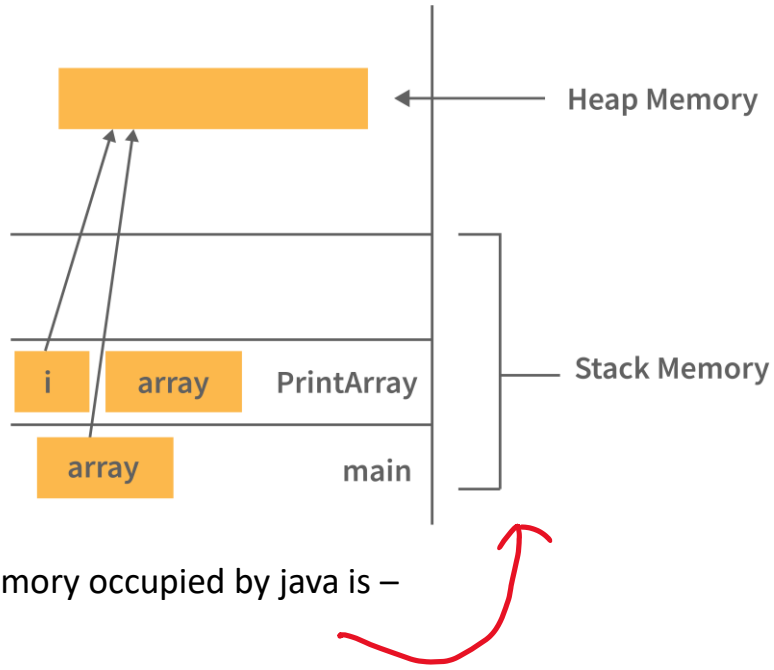
Ans: Object is instance of class.

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

```
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.

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

Ans:

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.

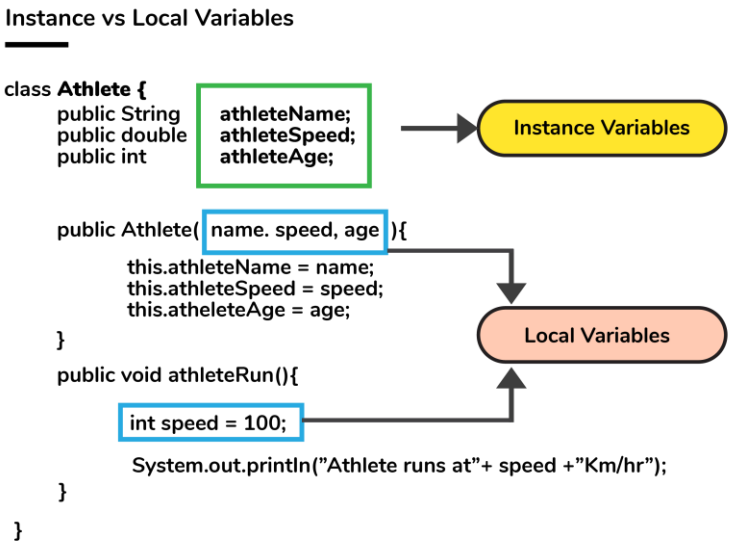
Example:

```
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;
}
```



16.What are the default values assigned to variables and instances in java?

Ans:

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.

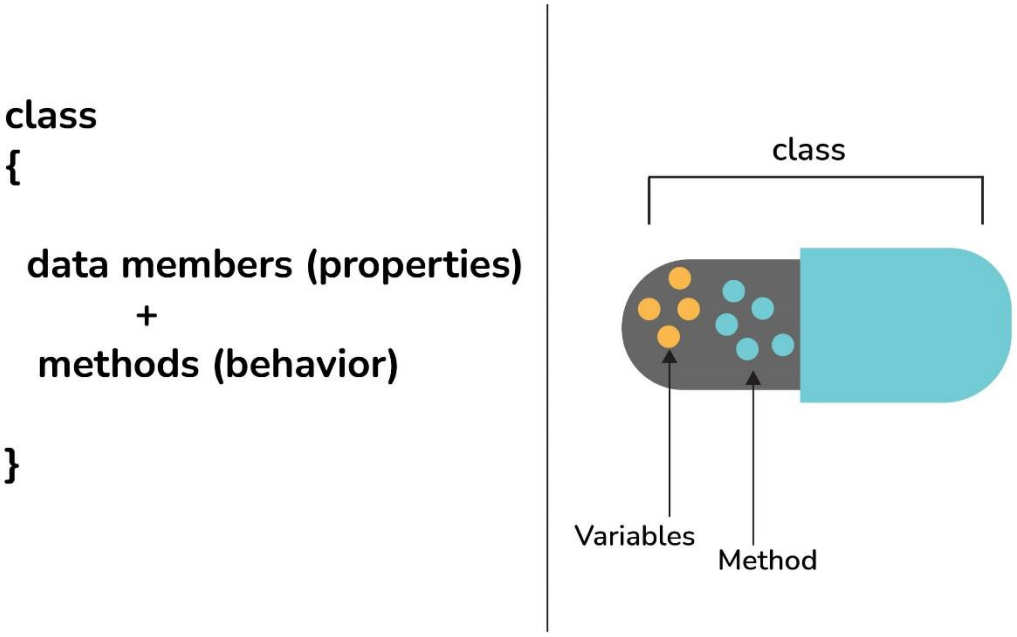
If it is a reference, then it will be assigned to null.

If it is numeric, then it will assign to 0.

If it is a boolean, then it will be assigned to false. Etc.

17.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.

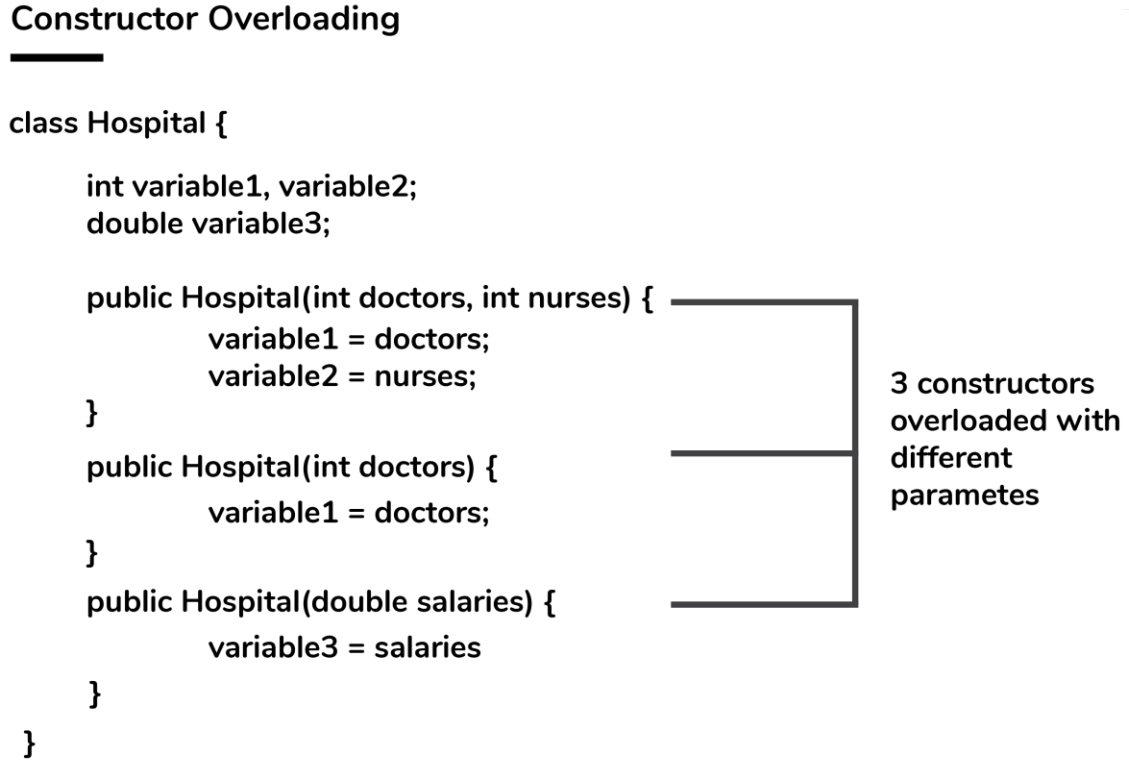


18.Briefly explain the concept of constructor overloading

Ans:

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

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



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

19.Define Copy constructor in java.

Ans:

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.

```
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.

20.Can the main method be Overloaded?

Ans:

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");
    }
}
```

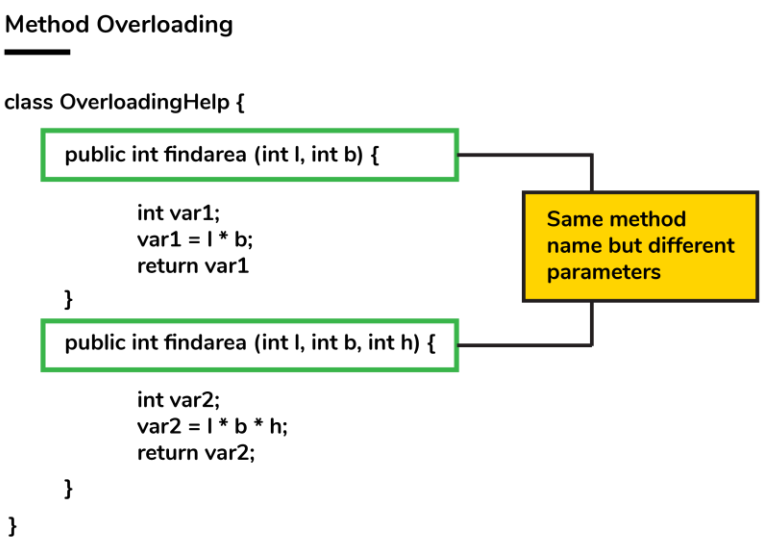
21.Comment on method overloading and overriding by citing relevant examples.

Ans:

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

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

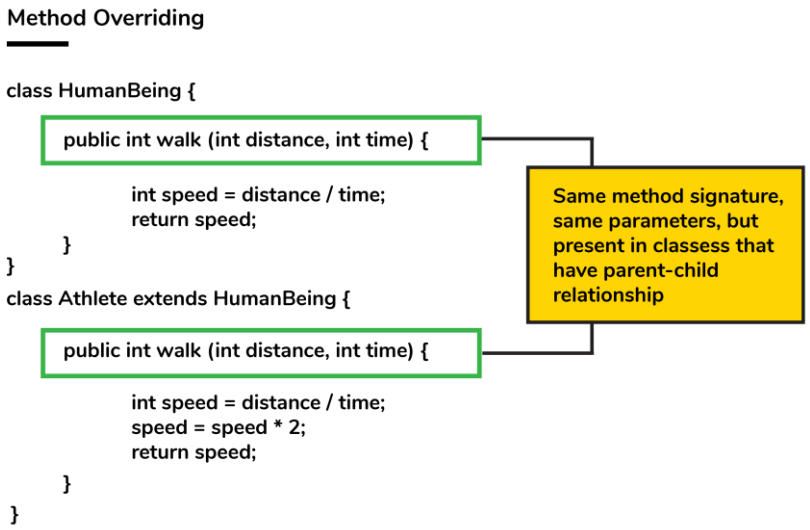
```
class OverloadingHelp {
  public int findarea (int l, int b) {
        int var1;
        var1 = l * b;
        return var1;
  }
  public int findarea (int l, int b, int h) {
        int var2;
        var2 = l * b * h;
        return var2;
  }
}
```



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

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

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



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

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

Ans:

In Java, the final keyword is used to define 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.

23.Why is the main method static in Java?

Ans:

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.

24. Can the static methods be overridden?

Ans:

No! Declaration of static methods having the same signature can be done in the subclass but run time polymorphism cannot 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 can't be overridden.

25.Difference between static methods, static variables, and static classes in java.
Ans:

- 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.

26. What are shallow copy and deep copy in java?
Ans:
To copy the object's data, we have several methods like deep copy and shallow copy.

Example -

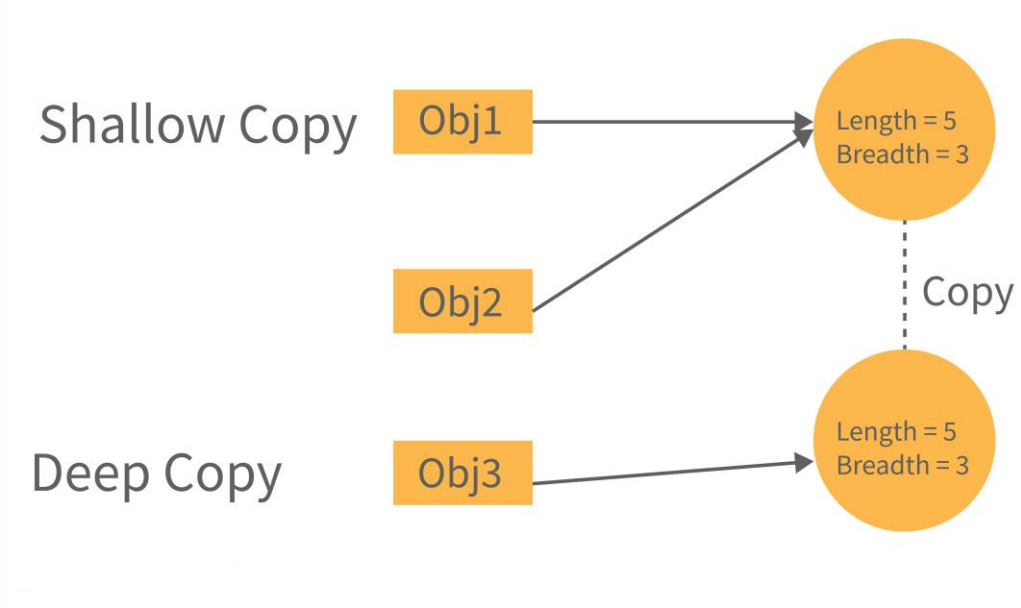
```
class Rectangle{
int length = 5;
    int breadth = 3;
}
```

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 –



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 -

```
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.

```
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();