Order of execution of Initialization blocks and Constructors in Java

Prerequisite : Static blocks, Initializer block, Constructor

In a Java program, operations can be performed on methods, constructors and initialization blocks.

Instance Initialization Blocks : IIB are used to initialize instance variables. IIBs are executed before constructors. They run each time when object of the class is created.

Initializer block : contains the code that is always executed whenever an instance is created. It is used to declare/initialize the common part of various constructors of a class.

Constructors : are used to initialize the object's state. Like methods, a constructor also contains collection of statements(i.e. instructions) that are executed at time of Object creation.

Order of execution of Initialization blocks and constructor in Java

1. Static initialization blocks will run whenever the class is loaded first time in JVM
2. Initialization blocks run in the same order in which they appear in the program.
3. Instance Initialization blocks are executed whenever the class is initialized and before constructors are invoked. They are typically placed above the constructors within braces.

```
// Java code to illustrate order of
// execution of constructors, static
// and initialization blocks
class GFG {

    GFG(int x)
    {
        System.out.println("ONE argument constructor");
    }
```

```
    GFG()
    {
        System.out.println("No  argument constructor");
    }

    static
    {
        System.out.println("1st static init");
    }

    {
        System.out.println("1st instance init");
    }

    {
        System.out.println("2nd instance init");
    }

    static
    {
        System.out.println("2nd static init");
    }

    public static void main(String[] args)
    {
        new GFG();
        new GFG(8);
    }
}
```

Output

```
1st static init
2nd static init
1st instance init
2nd instance init
No  argument constructor
1st instance init
2nd instance init
ONE argument constructor
```

Note : If there are two or more static/initializer blocks then they are executed in the order in which they appear in the source code.

Now, predict the output of the following program-

```
// A tricky Java code to predict the output
// based on order of
// execution of constructors, static
// and initialization blocks
class MyTest {
    static
    {
        initialize();
    }

    private static int sum;

    public static int getSum()
    {
        initialize();
        return sum;
    }

    private static boolean initialized = false;

    private static void initialize()
    {
        if (!initialized) {
            for (int i = 0; i < 100; i++)
                sum += i;
            initialized = true;
        }
    }
}

public class GFG {
    public static void main(String[] args)
    {
        System.out.println(MyTest.getSum());
    }
}
```

Output:

```
9900
```

Explanation:

- Loop in initialize function goes from 0 to 99. With that in mind, you might think that the program prints the sum of the numbers from 0 to 99. Thus sum is 99 × 100 / 2, or 4, 950. The program, however, thinks otherwise. It prints **9900**, fully twice this value.
- To understand its behavior, let's trace its execution.The GFG.main method invokes MyTest.getSum. Before the getSum method can be executed, the VM must initialize the class MyTest. Class initialization executes static initializers i the order they appear in the source.

- The MyTest class has two static initializers: the static block at the top of the class and the initialization of the static field initialized. The block appears first. It invokes the method initialize, which tests the field initialized. Because no value has been assigned to this field, it has the default boolean value of false.
- Similarly, sum has the default int value of 0. Therefore, the initialize method does what you'd expect, adding 4, 950 to sum and setting initialized to true. After the static block executes, the static initializer for the initialized field sets it back to false, completing the class initialization of MyTest. Unfortunately, sum now contains the 4950, but initialized contains false.
- The main method in the GFG class then invokes MyTest.getSum, which in turn invokes initialize method. Because the initialized flag is false, the initializeIf method enters its loop, which adds another 4, 950 to the value of sum, increasing its value to 9, 900. The getSum method returns this value, and the program prints it

This article is contributed by **Shubham Juneja**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Compilation and Execution of a Java Program

Static blocks in Java

Double Brace Initialization in Java

Instance Initialization Block (IIB) in Java

Nested try blocks in Exception Handling in Java

Initialization of local variable in a conditional block in Java

Constructors in Java

Java | Constructors | Question 1

Why Constructors are not inherited in Java?

Java | Constructors | Question 2

Java | Constructors | Question 3

Java | Constructors | Question 4

Java | Constructors | Question 5

Java | Constructors | Question 6