



# **FULL STACK DEVELOPMENT WORKSHEET - 5**

## **FIND OUTPUT OF THE PROGRAMS WITH EXPLANATION:**

**Q1.//Stringbuffer**

```
public class Main  
{  
  
    public static void main(String args[])  
  
    {  
  
        String s1 = "abc";  
  
        String s2 = s1;  
  
        s1 += "d";  
  
        System.out.println(s1 + " " + s2 + " " + (s1 == s2));  
  
        StringBuffer sb1 = new StringBuffer("abc");  
  
        StringBuffer sb2 = sb1;  
  
        sb1.append("d");  
  
        System.out.println(sb1 + " " + sb2 + " " + (sb1 == sb2));  
  
    }  
  
}
```

**Ans.** Sure, let's analyze the given Java program step by step.

**String part:**

**i)String s1 = "abc";**

creates the string literal "abc" and puts it in s1.

**ii)String s2 = s1;**

Now, s2 refers to the same thing as s1, namely, the string "abc".

**iii)** `s1 += "d";`

Assigning "abcd" to s1 generates a new string object.

**iv)** `System.out.println(s1 + " " + s2 + " " + (s1 == s2));`

In this instance, s1 is "abcd," s2 is still "abc," and the condition (s1 == s2) verifies that s1 and s2 refer to the same object, which is false because s1 and s2 are now distinct objects.

#### Output for String Part:

```
arduino Copy code  
abcd abc false
```

#### StringBuffer Part:

**i)** `StringBuffer sb1 = new StringBuffer("abc");`

includes "abc" to a StringBuffer object that is created and assigned to sb1.

**ii)** `StringBuffer sb2 = sb1`

Now, sb2 and sb1 are referencing the same StringBuffer object.

**iii)** `sb1.append("d");`

adds "abcd" to the StringBuffer object that sb1 is referencing.

**iv)** `System.out.println(sb1 + " " + sb2 + " " + (sb1 == sb2));`

In this case, (sb1 == sb2) is true as both sb1 and sb2 belong to the same StringBuffer object.

#### Output for StringBuffer Part:

```
arduino Copy code  
abcd abcd true
```

#### Final Output:

```
arduino Copy code  
abcd abc false  
abcd abcd true
```

**Explanation:**

S1 and S2 now refer to distinct objects since the += operator in the String section produces a new string object. As such, (s1 == s2) is false.

Both sb1 and sb2 refer to the same StringBuffer object in the StringBuffer section. When accessed with sb2, any modifications made to the object through sb1 are reflected. (sb1 == sb2) is thus true.

**Q2. // Method overloading****public class Main**

```
{  
  
    public static void FlipRobo(String s)  
    {  
        System.out.println("String");  
    }  
  
    public static void FlipRobo(Object o)  
    {  
        System.out.println("Object");  
    }  
  
    public static void main(String args[])  
    {  
        FlipRobo(null);  
    }  
}
```

**Ans.** Two FlipRobo methods in this Java programme show how to use method overloading. You can have many methods in a class with the same name but different arguments given to method overloading. Java will choose the most specific method to call based on the available parameter types when you call the FlipRobo method in the main function with a null argument. Let's examine the programme in detail:

```
java Copy code
public class Main {
    public static void FlipRobo(String s) {
        System.out.println("String");
    }

    public static void FlipRobo(Object o) {
        System.out.println("Object");
    }

    public static void main(String args[]) {
        FlipRobo(null);
    }
}
```

#### i)Method Definitions:

The FlipRobo(String s) function receives a String parameter.

The FlipRobo(Object o) function receives an object-type parameter.

#### ii)Method Invocation:

The FlipRobo(null) statement is called in the main function. Both String and Object types can be matched by the argument null. Java, on the other hand, will always select the most precise method that fits the specified parameters. An Object is not as specific in this situation as a String is.

#### iii)Output Explanation:

It is possible for both methods to accept the null value that you send to FlipRobo. Java, on the other hand, always resolves method overloading by selecting the most specific method.

Since String is a more precise type than Object in this instance, the String version of the procedure is more detailed. As a result, the program's output will be:

```
arduino Copy code
String
```

"String" will be printed as the result of the FlipRobo(String s) method being called by the programme.

**Q3. class First**

```
{  
  
    public First() { System.out.println("a"); }  
  
}  
  
class Second extends First  
  
{  
  
    public Second() { System.out.println("b"); }  
  
}  
  
class Third extends Second  
  
{  
  
    public Third() { System.out.println("c"); }  
  
}  
  
public class MainClass  
  
{  
  
    public static void main(String[] args)  
  
    {  
  
        Third c = new Third();  
  
    }  
  
}
```

**Ans.** First, Second, and Third are the three classes in this Java programme. Second is a subclass of First, while Third is a subclass of Second. In the main procedure, the programme generates an object of the Third class. Let's analyse the programme in detail:

```
class First {  
    public First() {  
        System.out.println("a");  
    }  
}  
  
class Second extends First {  
    public Second() {  
        System.out.println("b");  
    }  
}  
  
class Third extends Second {  
    public Third() {  
        System.out.println("c");  
    }  
}  
  
public class MainClass {  
    public static void main(String[] args) {  
        Third c = new Third();  
    }  
}
```

### i)Class Definitions:

The constructor of the first class prints "a".

The constructor of the second class, which extends First, outputs "b".

The constructor of the third class, which extends the Second class, outputs "c".

### ii)Object Creation:

An object of the Third class is generated in the main method: `c = new Third();` third

### iii)Output Explanation:

The constructor chain is called upon the creation of an object of type Third. A subclass constructor in Java calls for the constructor of its superclass implicitly before starting to run its own code.

The constructor invocation order is as follows:

Third extends Second, which extends First, hence the First class constructor is executed. Thus, "a" is output.

Since Second is the superclass of Third, it is called by the constructor of Second class. Thus, "b" is output.

Since the object is generated for this class, the third class constructor is invoked. Thus, "c" is output.

As a result, the program's output will be:

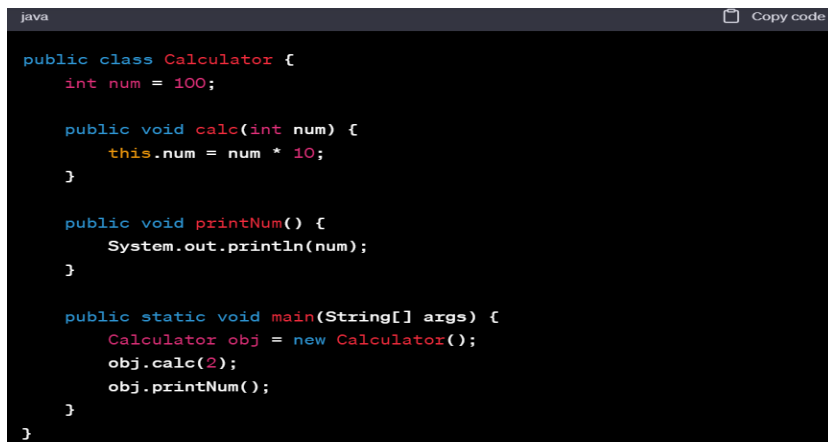


```
css Copy code  
a  
b  
c
```

#### Q4. public class Calculator

```
{  
  
    int num = 100;  
  
    public void calc(int num) { this.num = num * 10; }  
  
    public void printNum() { System.out.println(num); }  
  
    public static void main(String[] args)  
    {  
  
        Calculator obj = new Calculator();  
  
        obj.calc(2);  
  
        obj.printNum();  
  
    }  
}
```

**Ans.** The Calculator class in this Java programme has an instance variable called num, two methods called calc(int num) and printNum(), and a main function that creates an object of the Calculator class and calls its methods. Let's analyse the programme in detail:



```
java Copy code  
  
public class Calculator {  
    int num = 100;  
  
    public void calc(int num) {  
        this.num = num * 10;  
    }  
  
    public void printNum() {  
        System.out.println(num);  
    }  
  
    public static void main(String[] args) {  
        Calculator obj = new Calculator();  
        obj.calc(2);  
        obj.printNum();  
    }  
}
```



### **i)Class Definitions:**

The instance variable num of the calculator class is initialised to 100.

It provides a method called calc(int num) that multiplies an integer by 10 and assigns the result to the instance variable num.

The value of the instance variable num is printed via the printNum() function.

### **ii) Object Creation and Method Calls:**

Calculator obj = new Calculator(); creates an object of the Calculator class.

Upon using the calc(2) function on this object, the parameter 2 is multiplied by 10 and the result (20) is assigned to the instance variable num.

Next, the changed instance variable num's value of 20 is printed by calling the printNum() function.

### **iii)Output Explanation:**

The calc(2) function changes the num variable to 20 when the programme runs. Therefore, the modified value of 20 is printed by the printNum() function.

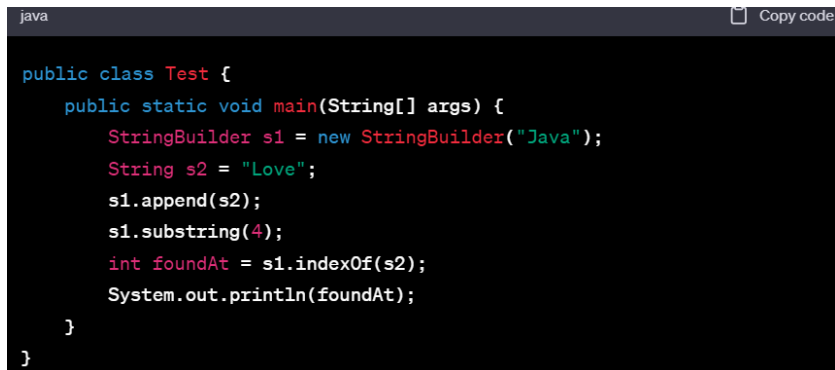
The program's output will be:

A screenshot of a terminal window with a dark background. The output of the program is the number '20'. In the top right corner of the terminal window, there is a small icon of a document and the text 'Copy code'.

**Q5.**public class Test

```
{  
  
    public static void main(String[] args)  
    {  
  
        StringBuilder s1 = new StringBuilder("Java");  
  
        String s2 = "Love";  
  
        s1.append(s2);  
  
        s1.substring(4);  
  
        int foundAt = s1.indexOf(s2);  
  
        System.out.println(foundAt);  
  
    }  
}
```

**Ans.** Let's analyse the provided Java programme in detail:



```
java Copy code  
  
public class Test {  
    public static void main(String[] args) {  
        StringBuilder s1 = new StringBuilder("Java");  
        String s2 = "Love";  
        s1.append(s2);  
        s1.substring(4);  
        int foundAt = s1.indexOf(s2);  
        System.out.println(foundAt);  
    }  
}
```

**i) StringBuilder Initialization:**

"Java" is the first value initialised in StringBuilder s1.

"Love" is the initial value assigned to string s2.

## ii) StringBuilder Operations:

S1 now equals "JavaLove" after `s1.append(s2)` concatenates the value of `s2` ("Love") to the end of S1.

The output of `s1.substring(4)`, which generates a substring beginning at index 4, is not assigned to any variable or utilised, hence it has no effect on the value of `s1`. The first `s1` ("JavaLove") is still the same.

## iii) Finding the Index:

The function `s1.indexOf(s2)` looks for the index of the substring "Love" in `s1`. Since index 4 is where "Love" first appears in `s1`, the `indexOf` function will return 4.

## iv) Output Explanation:

The index at which the given substring appears in the original string—in this example, `s1`—is returned by the `indexOf` method. The substring "Love" in the altered string "JavaLove" begins at index 4.

As a result, the `foundAt` variable is set to 4.

The following will be the program's output since it uses `System.out.println(foundAt)` to print the value of `foundAt`:



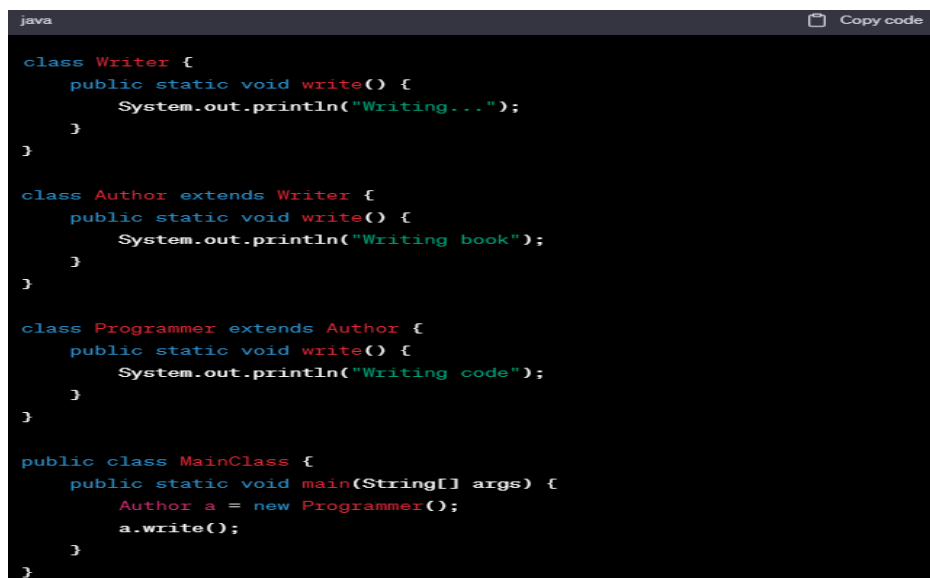
```
4
```

**Q6. class Writer**

```
{  
  
    public static void write()  
    {  
        System.out.println("Writing...");  
    }  
}  
  
class Author extends Writer  
{  
  
    public static void write()  
    {  
        System.out.println("Writing book");  
    }  
  
    public class Programmer extends Author  
    {  
  
        public static void write()  
        {  
            System.out.println("Writing code");  
        }  
  
        public static void main(String[] args)  
        {
```

```
Author a = new Programmer();  
  
a.write();  
  
}  
  
}
```

**Ans.** The provided code has a few issues. Java's declaration of the Programmer class inside the Author class is not syntactically proper. In addition, the curly brackets that close have a mismatch. The updated code may be found below:



```
java Copy code  
  
class Writer {  
    public static void write() {  
        System.out.println("Writing...");  
    }  
}  
  
class Author extends Writer {  
    public static void write() {  
        System.out.println("Writing book");  
    }  
}  
  
class Programmer extends Author {  
    public static void write() {  
        System.out.println("Writing code");  
    }  
}  
  
public class MainClass {  
    public static void main(String[] args) {  
        Author a = new Programmer();  
        a.write();  
    }  
}
```

#### i) Class Definitions:

The static write() function of the writer class outputs "Writing..."

The write() function is overridden by the Author class, which extends Writer, to print "Writing book".

The write() function is overridden by the Programmer class, which extends Author, to output "Writing code".

#### ii) Object Creation and Method Invocation:

It is constructed, called an object of the Author class, and initialised as a Programmer object:  
Author a = new Programmer();

the reference an is used to call the write() function.

### iii) Output Explanation:

What methods can be called during compilation is determined by the reference type (Author).

The override method that is called at runtime is determined by the actual object type (Programmer).

Since an refers to a Programmer object but is of type Author, the Programmer class's override function will be executed.

As a result, the program's output will be:



### Q7.class FlipRobo

```
{  
  
public static void main(String args[])  
{  
  
String s1 = new String("FlipRobo");  
  
String s2 = new String("FlipRobo");  
  
if (s1 == s2) System.out.println("Equal");  
else System.out.println("Not equal");  
  
}  
  
}
```

**Ans.** The new keyword is used to generate two String objects, s1 and s2, in the provided Java programme. The == operator is then used by the programme to compare these two String objects. Let's examine the programme in detail:

```
java Copy code
class FlipRobo {
    public static void main(String args[]) {
        String s1 = new String("FlipRobo");
        String s2 = new String("FlipRobo");
        if (s1 == s2)
            System.out.println("Equal");
        else
            System.out.println("Not equal");
    }
}
```

### i) String Objects Creation:

Two distinct String objects, s1 and s2, were produced using the new keyword. They are distinct objects in memory since they are formed using the new operator, even though their content ("FlipRobo") is the same.

### ii) Comparison Using == Operator:

The software compares the references of s1 and s2 using the == operator. Rather of comparing the contents of s1 and s2, this comparison verifies if they point to the same memory address.

### iii) Output Explanation:

Despite having the same information, s1 and s2's references differ since they were generated as distinct objects. After comparing the references, the == operator determines that s1 and s2 do not refer to the same memory item.

The s1 == s2 condition thus evaluates to false.

Therefore, "Not equal" is sent to the console by the programme.

### iv) Output:

```
mathematica Copy code
Not equal
```

```
Q8.class FlipRobo

{

public static void main(String args[])

{

try

{

System.out.println("First statement of try block");

int num=45/3;

System.out.println(num);

}

catch(Exception e)

{

System.out.println("FlipRobo caught Exception");

}

finally

{

System.out.println("finally block");

}

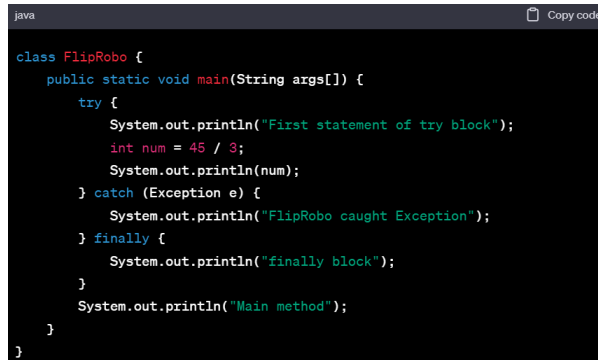
System.out.println("Main method");

}

}
```



**Ans.** The Java programme that is given has a try-catch-finally block. Let's analyse the programme in detail:



#### **i) Try Block:**

The code that is contained within the try block aims to run `int num = 45 / 3;`, which determines that num is 15.

"First statement of try block" is printed first, followed by the value of num (15).

#### **ii) Catch Block (Not Executed):**

The catch block is not performed since there isn't an exception thrown in the try block. Since there are no exceptions seen by the programme, the catch block is skipped.

#### **iii) Finally Block:**

The finally block always gets to work, even if there's no exception captured. It prints "finally block" in this case.

#### **iv) After Finally Block:**

Following the finally block, the programme proceeds to run the line that outputs "Main method" outside of the try-catch-finally block.

#### **v) Output Explanation:**

The catch block is not run since there are no exceptions seen by the programme.

"finally block" is printed as the finally block always runs.

"Main method" is then printed by the programme.

vi) Output:

```
csharp Copy code
First statement of try block
15
finally block
Main method
```

**Q9.class FlipRobo**

```
{
// constructor
FlipRobo()
{
System.out.println("constructor called");
}
static FlipRobo a = new FlipRobo(); //line 8
public static void main(String args[])
{
FlipRobo b; //line 12
b = new FlipRobo();
}
}
```

**Ans.** There is a class called FlipRobo with a constructor in this Java programme. There's also a static variable of type FlipRobo that is initialised at line 8 with a new instance of FlipRobo. Next, a new instance of FlipRobo is assigned to a local variable called b, which is of type FlipRobo, which is defined at line 12 of the main method. Let's analyse the programme in detail:

```
java Copy code
class FlipRobo {
    // constructor
    FlipRobo() {
        System.out.println("constructor called");
    }

    static FlipRobo a = new FlipRobo(); // line 8

    public static void main(String args[]) {
        FlipRobo b; // line 12
        b = new FlipRobo();
    }
}
```

### i) Static Variable Initialization (line 8):

A new instance of FlipRobo is used to initialise a static variable of type a. "constructor called" is displayed in this phase because the constructor is triggered by this initialization.

### ii) Main Method Execution (lines 12-14):

At line 12, a local variable of type FlipRobo, b, is defined but not initialised.

A new instance of FlipRobo is created and assigned to the variable b inside the main method. "constructor called" is printed once again as a result of the constructor being triggered by this.

### iii) Output Explanation:

Initialization of the static variable a occurs upon loading the FlipRobo class. FlipRobo creates an instance during initialization, calling the constructor and displaying "constructor called" once.

A new instance of FlipRobo is created within the main function, calling the constructor and displaying "constructor called" once more. A local variable named b is also defined.

### iv) Output:

```
kotlin Copy code
constructor called
constructor called
```

```
Q10.class FlipRobo

{

static int num;

static String mystr;

// constructor

FlipRobo()

{

num = 100;

mystr = "Constructor";

}

// First Static block

static

{

System.out.println("Static Block 1");

num = 68;

mystr = "Block1";

}

// Second static block

static

{

System.out.println("Static Block 2");
```

```
num = 98;

mystr = "Block2";

}

public static void main(String args[])

{

FlipRobo a = new FlipRobo();

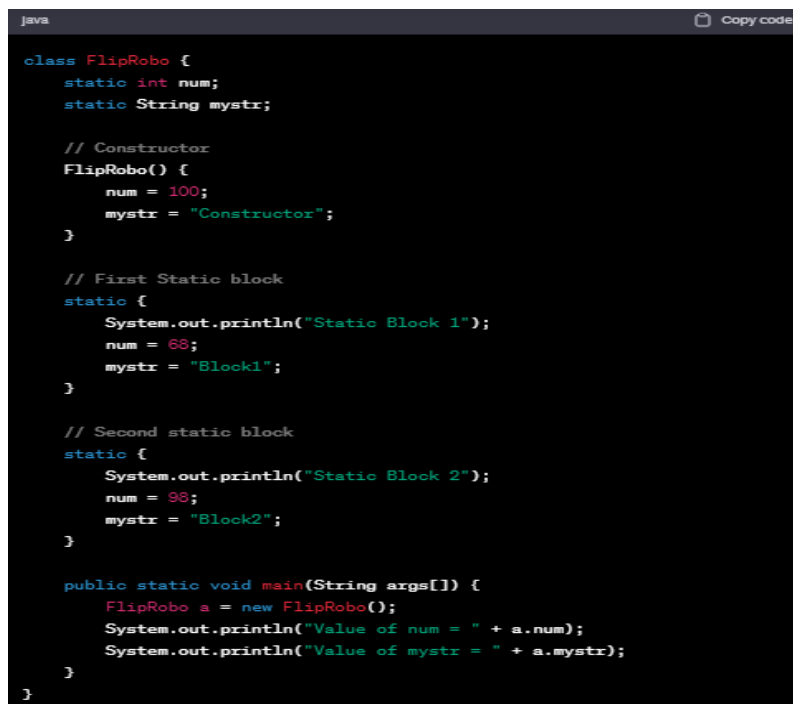
System.out.println("Value of num = " + a.num);

System.out.println("Value of mystr = " + a.mystr);

}

}
```

**Ans.** This Java programme has two static blocks, a constructor, the class FlipRobo, and the static variables num and mystr. Let's analyse the programme in detail:



```
java Copy code

class FlipRobo {
    static int num;
    static String mystr;

    // Constructor
    FlipRobo() {
        num = 100;
        mystr = "Constructor";
    }

    // First Static block
    static {
        System.out.println("Static Block 1");
        num = 68;
        mystr = "Block1";
    }

    // Second static block
    static {
        System.out.println("Static Block 2");
        num = 98;
        mystr = "Block2";
    }

    public static void main(String args[]) {
        FlipRobo a = new FlipRobo();
        System.out.println("Value of num = " + a.num);
        System.out.println("Value of mystr = " + a.mystr);
    }
}
```

**i) First Static Block (line 11-15):**

First, a static block initialises mystr to "Block1" and num to 68.

"Static Block1" is printed to the console.

**ii) Second Static Block (line 17-21):**

Mystr is set to "Block2" and num is initialised to 98 in the second static block.

"Static Block 2" is printed to the console.

**iii) Constructor (line 8-10):**

The constructor initialises mystr to "Constructor" and num to 100.

**iv) Main Method (line 23-28):**

An object an of type FlipRobo is created within the main method. The constructor is triggered by this.

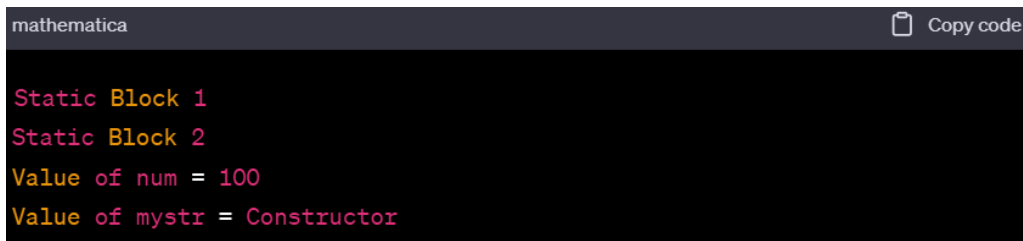
Next, it uses System.out.println to output the values of num and mystr.

**v) Output Explanation:**

"Static Block 1" and "Static Block 2" are printed because the static blocks are run when the class loads.

When an object called an is formed, the constructor is invoked. It sets mystr to "Constructor" and num to 100.

The values of num and mystr are printed when System.out.println commands are run.

**vi) Output:**

```
mathematica Copy code
Static Block 1
Static Block 2
Value of num = 100
Value of mystr = Constructor
```