

**Each question carry equal marks.
Please answer briefly.**

NAME: M OMAR MASOOD

ID: 14305

SUBJECT: Object Oriented Programming

SUBMITTED TO: Sir Ayub

Q1. How many variables are being supported by java justify your answer with the help java coded example for each variable?

ANSWER: There are three kinds of variables in Java.

- Local variables
- Instance variables
- Class/Static variables

Local Variables

- Local variables are declared in methods, constructors, or blocks.
- Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.
- Access modifiers cannot be used for local variables.
- Local variables are visible only within the declared method, constructor, or block.
- Local variables are implemented at stack level internally.
- There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

EXAMPLE:

- Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to only this method.

CODE:

```
public class Test {
    public void pupAge() {
        int age = 0;
        age = age + 7;
        System.out.println("Puppy age is : " + age);
    }

    public static void main(String args[]) {
```

```
        Test test = new Test();
        test.pupAge();
    }
}
```

This will produce the following result –

Output

Puppy age is: 7

CODE WITH OUTPUT:



The screenshot shows an IDE window with two panes. The left pane, titled 'Source File', contains the following Java code:

```
1 public class Test {
2     public void pupAge() {
3         int age = 0;
4         age = age + 7;
5         System.out.println("Puppy age is : " + age);
6     }
7
8     public static void main(String args[]) {
9         Test test = new Test();
10        test.pupAge();
11    }
12 }
```

The right pane, titled 'Result', shows the command prompt output:

```
$javac Test.java
$java -Xmx128M -Xms16M Test
Puppy age is : 7
```

Instance Variables

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.
- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Example

CODE:

```
import java.io.*;
public class Employee {

    // this instance variable is visible for any child class.
    public String name;

    // salary variable is visible in Employee class only.
    private double salary;

    // The name variable is assigned in the constructor.
    public Employee (String empName) {
        name = empName;
    }

    // The salary variable is assigned a value.
    public void setSalary(double empSal) {
        salary = empSal;
    }

    // This method prints the employee details.
    public void printEmp() {
        System.out.println("name : " + name );
        System.out.println("salary :" + salary);
    }

    public static void main(String args[]) {
        Employee empOne = new Employee("Ransika");
        empOne.setSalary(1000);
        empOne.printEmp();
    }
}
```

This will produce the following result –

Output

```
name : Ransika
salary :1000.0
```

CODE WITH OUTPUT:

```
Execute | Share | Source File | STDIN | Result
1 import java.io.*;
2 public class Employee {
3
4 // this instance variable is visible for any child class.
5 public String name;
6
7 // salary variable is visible in Employee class only.
8 private double salary;
9
10 // The name variable is assigned in the constructor.
11 public Employee (String empName) {
12     name = empName;
13 }
14
15 // The salary variable is assigned a value.
16 public void setSalary(double empSal) {
17     salary = empSal;
18 }
19
20 // This method prints the employee details.
21 public void printEmp() {
22     System.out.println("name : " + name );
23     System.out.println("salary : " + salary);
24 }
25
26 public static void main(String args[]) {
27     Employee empOne = new Employee("Ransika");
28     empOne.setSalary(1000);
29     empOne.printEmp();
30 }
31 }
```

\$javac Employee.java
\$java -Xmx128M -Xms16M Employee
name : Ransika
salary :1000.0

Activate
Go to Settir

Class/Static Variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.
- Static variables are stored in the static memory. It is rare to use static variables other than declared final and used as either public or private constants.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case. If the static variables are not public and final, the naming syntax is the same as instance and local variables.

Example

CODE:

```
import java.io.*;
public class Employee {

    // salary variable is a private static variable
    private static double salary;

    // DEPARTMENT is a constant
    public static final String DEPARTMENT = "Development ";

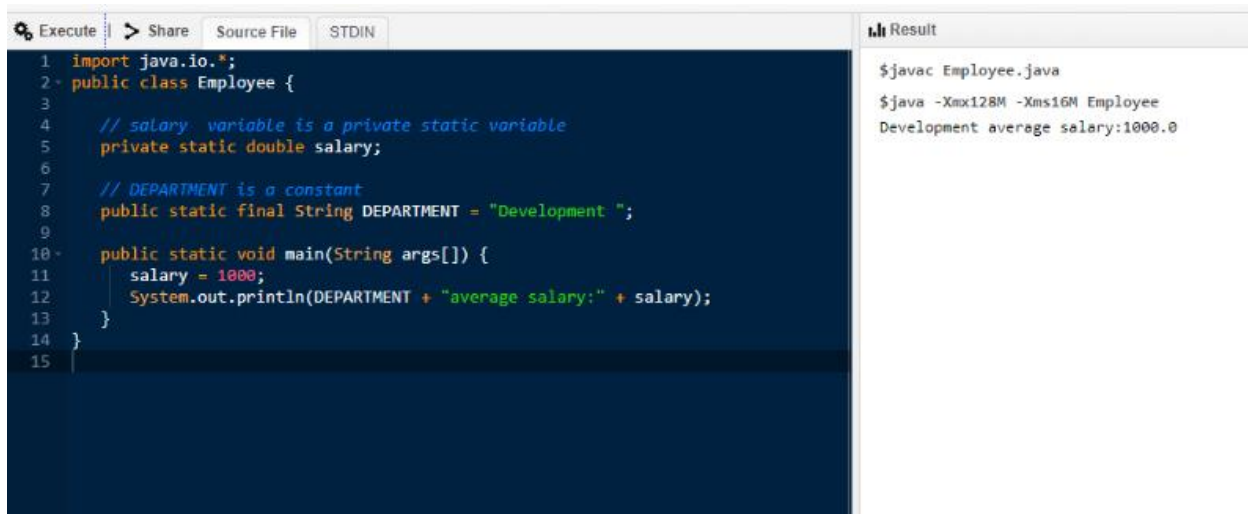
    public static void main(String args[]) {
        salary = 1000;
        System.out.println(DEPARTMENT + "average salary:" + salary);
    }
}
```

This will produce the following result –

Output

Development average salary:1000

CODE WITH OUTPUT:

A screenshot of an IDE window. The left pane shows Java code for an Employee class. The right pane shows the terminal output of the code execution. The code defines a private static double variable 'salary' with a value of 1000 and a public static final String variable 'DEPARTMENT' with the value 'Development '. The main method prints the department name followed by 'average salary:' and the salary value. The terminal output shows the command '\$javac Employee.java' followed by '\$java -Xmx128M -Xms16M Employee' and the output 'Development average salary:1000.0'.

```
Execute | Share | Source File | STDIN | Result
1 import java.io.*;
2 public class Employee {
3
4     // salary variable is a private static variable
5     private static double salary;
6
7     // DEPARTMENT is a constant
8     public static final String DEPARTMENT = "Development ";
9
10    public static void main(String args[]) {
11        salary = 1000;
12        System.out.println(DEPARTMENT + "average salary:" + salary);
13    }
14 }
15

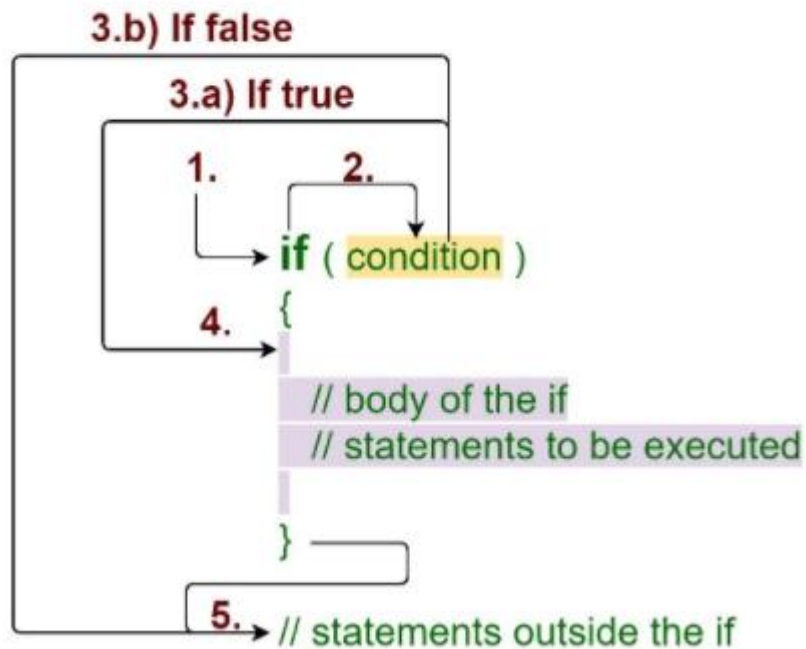
$javac Employee.java
$java -Xmx128M -Xms16M Employee
Development average salary:1000.0
```

Q2. Why “If” is used in java justify your answer with the help java coded example and explain in detail?

ANSWER:

The Java if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

If statement



Syntax:

```
If condition
{
    // statement to execute if
    // condition is true
}
```

Working of if statement

1. Control falls into the if block.
2. The flow jumps to Condition.
3. Condition is tested.
 - a. If Condition yields true, goto Step 4.
 - b. If Condition yields false, goto Step 5.
4. The if-block or the body inside the if is executed.
5. Flow steps out of the if block.

Operation:

The condition after evaluation of if-statement will be either true or false. The if statement in Java accepts boolean values and if the value is true then it will execute the block of statements under it.

Note: If we do not provide the curly braces '{' and '}' after if(condition) then by default if statement will consider the immediate one statement to be inside its block. For example,

```
if(condition)
    statement1;
    statement2;

// Here if the condition is true, if block
// will consider only statement1 to be inside
// its block.
```

CODE:

// Java program to illustrate If statement

```
class IfDemo {
    public static void main(String args[])
    {
        int i = 10;

        if (i < 15)
            System.out.println("10 is less than 15");

        // This statement will be executed
        // as if considers one statement by default
        System.out.println("Outside if-block");
    }
}
```

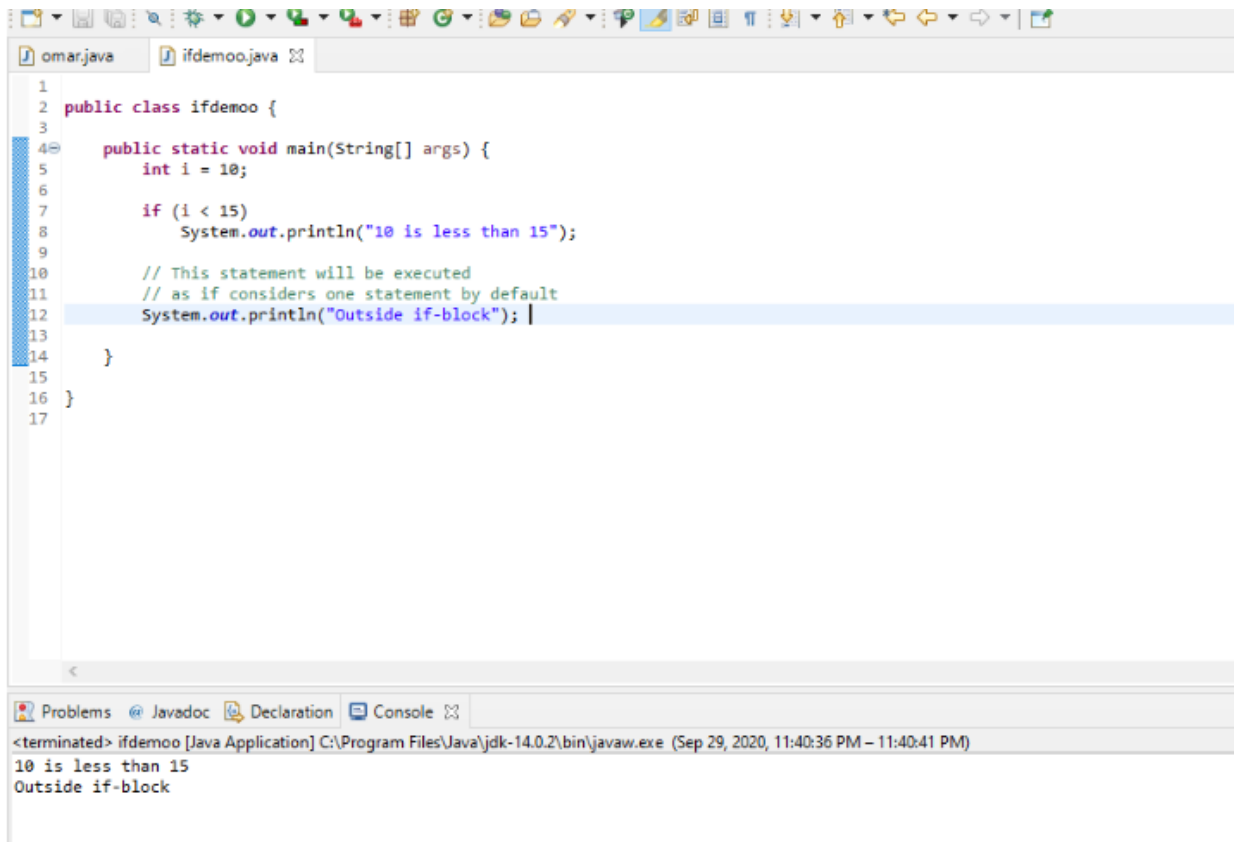
Output:

```
10 is less than 15
Outside if - block
```

Explanation:

1. Program starts.
2. i is initialized to 10.

3. if-condition is checked. $10 < 15$, yields true.
 - 3.a) "10 is less than 15" gets printed.
4. "Outside if-block" is printed



```
1 public class ifdemoo {
2
3
4     public static void main(String[] args) {
5         int i = 10;
6
7         if (i < 15)
8             System.out.println("10 is less than 15");
9
10        // This statement will be executed
11        // as if considers one statement by default
12        System.out.println("Outside if-block");
13
14    }
15
16 }
17
```

Problems Javadoc Declaration Console

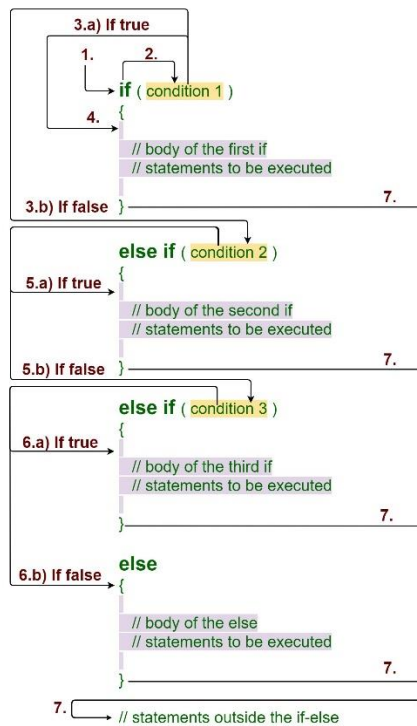
```
<terminated> ifdemoo [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (Sep 29, 2020, 11:40:36 PM - 11:40:41 PM)
10 is less than 15
Outside if-block
```

Q3. Why "if else if" is used in java justify your answer with the help java coded example and explain in detail?

ANSWER:

Java if-else-if ladder is used to decide among multiple options. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

If - else if ladder



Syntax:

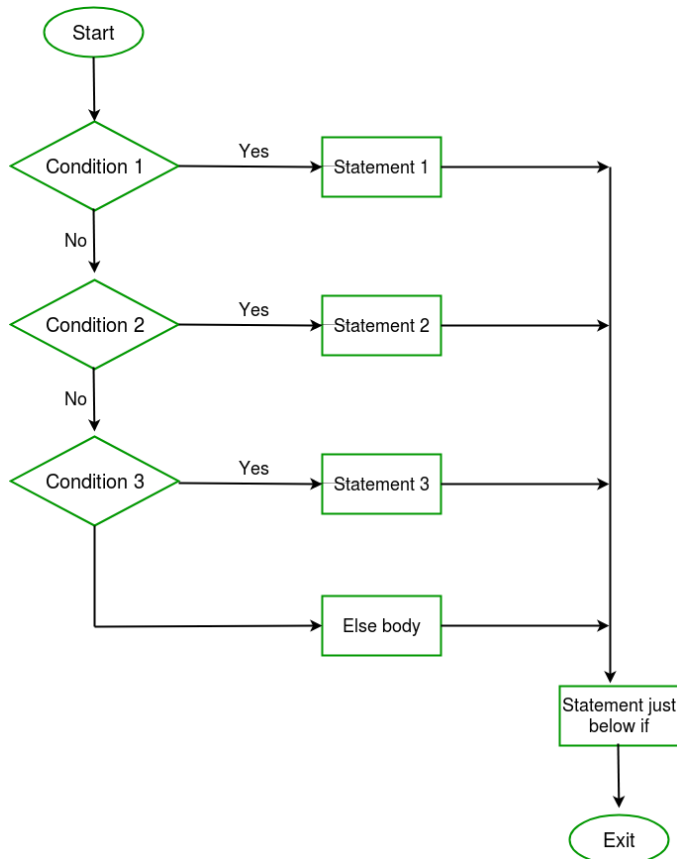
```
if (condition)
    statement 1;
else if (condition)
    statement 2;
.
.
else
    statement;
```

Working of the if-else-if ladder:

1. Control falls into the if block.
2. The flow jumps to Condition 1.
3. Condition is tested.
 - a. If Condition yields true, goto Step 4.
 - b. If Condition yields false, goto Step 5.
4. The present block is executed. Goto Step 7.
5. The flow jumps to Condition 2.
 - a. If Condition yields true, goto step 4.
 - b. If Condition yields false, goto Step 6.

6. The flow jumps to Condition 3.
 - a. If Condition yields true, goto step 4.
 - b. If Condition yields false, execute else block.
Goto Step 7.
7. Exit the if-else-if ladder.

Flowchart if-else-if ladder:



CODE:

```
// Java program to illustrate if-else-if ladder
```

```
import java.io.*;
```

```
class GFG {  
    public static void main(String[] args)  
    {  
        // initializing expression  
        int i = 20;  
  
        // condition 1  
        if (i == 10)  
            System.out.println("i is 10\n");  
    }  
}
```

```
        // condition 2
        else if (i == 15)
            System.out.println("i is 15\n");

        // condition 3
        else if (i == 20)
            System.out.println("i is 20\n");

        else
            System.out.println("i is not present\n");

        System.out.println("Outside if-else-if");
    }
}
```

Output:

i is 20

Outside if-else-if

EXPLANATION:

1. Program starts.
2. i is initialized to 20.
3. condition 1 is checked. $20 == 10$, yields false.
4. condition 2 is checked. $20 == 15$, yields false.
5. condition 3 is checked. $20 == 20$, yields true.
 - 5.a) "i is 20" gets printed.
6. "Outside if-else-if" gets printed.
7. Program ends.

```
omar.java ifdemo.java GFG.java
1 // Java program to illustrate if-else-if ladder
2
3 import java.io.*;
4
5 class GFG {
6     public static void main(String[] args)
7     {
8         // initializing expression
9         int i = 20;
10
11        // condition 1
12        if (i == 10)
13            System.out.println("i is 10\n");
14
15        // condition 2
16        else if (i == 15)
17            System.out.println("i is 15\n");
18
19        // condition 3
20        else if (i == 20)
21            System.out.println("i is 20\n");
22
23        else
24            System.out.println("i is not present\n");
25
26        System.out.println("Outside if-else-if");
27    }
28 }
<
Problems @ Javadoc Declaration Console
<terminated> GFG [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (Sep 29, 2020, 11:57:48 PM – 11:57:53 PM)
i is 20
Outside if-else-if
```

Q4. What are loops, why they are used in java and how many types of loops are being supported by java explain in detail?

ANSWER:

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages Java programming language provides the following types of loop to handle looping requirements. Click the following links to check their detail.

There are three types of loop;

WHILE LOOP:

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

FOR LOOP:

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

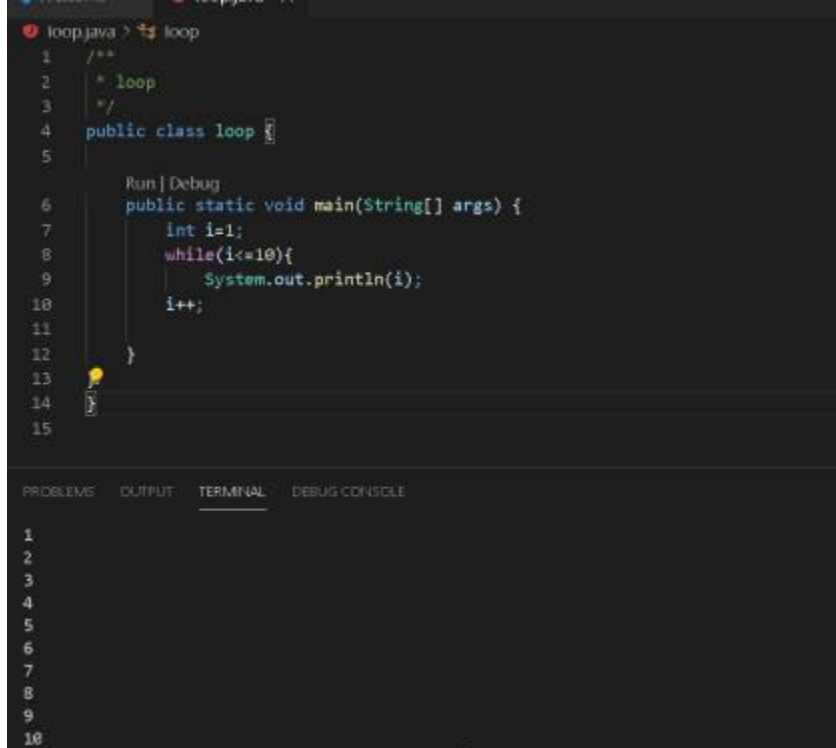
DO WHILE LOOP:

Like a while statement, except that it tests the condition at the end of the loop body.

While loop:

1. while(condition){

2. //code to be executed
3. }



The screenshot shows an IDE window titled 'loop.java' with the following code:

```
1  /**
2   * loop
3   */
4  public class loop {
5
6      public static void main(String[] args) {
7          int i=1;
8          while(i<=10){
9              System.out.println(i);
10             i++;
11         }
12     }
13 }
14
15
```

Below the code editor, the 'TERMINAL' tab is active, displaying the output of the program:

```
1
2
3
4
5
6
7
8
9
10
```

For Loop:

1. for(initialization;condition;incr/decr){
2. //statement or code to be executed
3. }

```
loop.java > loop
1  /**
2   * loop
3   */
4  public class loop {
5
6      Run | Debug
7      public static void main(String[] args) {
8          int n=10;
9          for(int i=0;i<10;i++){
10             System.out.println(n);
11         }
12     }
13 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
10
10
10
10
10
10
10
```

Q5. Write 3's table in decremented form in java which takes input from user write java coded program and explain in detail?

ANSWER:

```
table.java > table > main(String[])
1  import java.util.Scanner;
2  /**
3   * table
4   */
5  public class table {
6
7      Run | Debug
8      public static void main(String[] args) {
9          Scanner s = new Scanner(System.in);
10         System.out.print("Enter number:");
11         int n=s.nextInt();
12         for(int i=10; i >= 1; i--)
13         {
14             System.out.println(n+" * "+i+" = "+n*i);
15         }
16     }
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
a-debug-0.28.0\scripts\launcher.bat' 'C:\Program Files\Java\jdk-12.0.2\bin\java.exe' '-Dfile
ode\User\workspaceStorage\044c50c790f8b01f5c37966914aba029\redhat.java\jdt_ws\question 5_f7
Enter number:10
10 * 10 = 100
10 * 9 = 90
10 * 8 = 80
10 * 7 = 70
10 * 6 = 60
10 * 5 = 50
10 * 4 = 40
10 * 3 = 30
10 * 2 = 20
10 * 1 = 10
```

```
table.java / table / main(String[])
1  import java.util.Scanner;
2  /**
3   * table
4   */
5  public class table {
6
7      Run | Debug
8      public static void main(String[] args) {
9          Scanner s = new Scanner(System.in);
10         System.out.print("Enter number:");
11         int n=s.nextInt();
12         for(int i=10; i >= 1; i--)
13         {
14             System.out.println(n+" * "+i+" = "+n*i);
15         }
16     }
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
a-debug-0.28.0\scripts\launcher.bat' 'C:\Program Files\Java\jdk-12.0.2\bin\java.exe' '-Dfi
ode\User\workspaceStorage\044c50c790f8b01f5c37966914aba029\redhat.java\jdt_ws\question_5_f
Enter number:3
3 * 10 = 30
3 * 9 = 27
3 * 8 = 24
3 * 7 = 21
3 * 6 = 18
3 * 5 = 15
3 * 4 = 12
3 * 3 = 9
3 * 2 = 6
3 * 1 = 3
```



```
table.java > table > main(String[])
1  import java.util.Scanner;
2  /**
3   * table
4   */
5  public class table {
6
7      Run | Debug
      public static void main(String[] args) {
8          Scanner s = new Scanner(System.in);
9          System.out.print("Enter number:");
10         int n=s.nextInt();
11         for(int i=10; i >= 1; i--){
12             {
13                 System.out.println(n+" * "+i+" = "+n*i);
14             }
15         }
16     }
```

PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

```
a-debug-0.28.0\scripts\launcher.bat' 'C:\Program Files\Java\jdk-12.0.2\bin\java
ode\User\workspaceStorage\044c50c790f8b01f5c37966914aba029\redhat.java\jdt_ws
Enter number:6
6 * 10 = 60
6 * 9 = 54
6 * 8 = 48
6 * 7 = 42
6 * 6 = 36
6 * 5 = 30
6 * 4 = 24
6 * 3 = 18
6 * 2 = 12
6 * 1 = 6
```

EXPLANATION:

In this program we take input from user.

The number n will be that user input number.The i will be counter.The i will start from 10.and ends at 1.

Here the counter will go in reverse order.fisrt the number n will be multiplu from I values.(N*I). After that the number n will be same and the(i-1.10-1=9) .and the loop run until the value of i comes to 1.