

Id No 13943

Name Mehran Ali Shah

Subject OOP

Semester Summer-2020

Instructor M.Ayub

Date Sep.30.2020

Question No 1:

Answer:

There are three kinds of variables in Java

1. Local variables
2. Instance variables
3. 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. Local variables are visible only within the declared method, constructor, or block. Local variables are implemented at stack level internally. Access modifiers cannot be used for local variables. 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.

Declaration of Local Variable:

Every local variable declaration statement is contained by a block ({ ... }). We can also declare the local variables in the header of a “for” statement. In this case it is executed in the same manner as if it were part of a local variable declaration statement.

For example: `for(int i=0;i<=5;i++){.....}`

In above example `int i=0` is a local variable declaration. Its scope is only limited to the for loop.

Syntax Of Local Variable:

```
methodName(){  
<DataType> localVarName;  
}
```

Program:

In this program, the variable `age` is a local variable to the function `StudentAge()`.

```
public class StudentDetails {  
    public void StudentAge()  
    {  
        // local variable age  
        int age = 0;  
        age = age + 5;  
        System.out.println("Student age is : " + age);  
    }  
    public static void main(String args[])
```

```

    {
        StudentDetails obj = new StudentDetails();
        obj.StudentAge();
    }
}

```

Output:

Student age is : 21

The screenshot shows an IDE window with a dark theme. On the left, the source code for a Java class named `StudentDetails` is displayed. The code includes a `StudentAge()` method that initializes a local variable `age` to 0, increments it by 21, and prints the result. A `main` method creates an instance of `StudentDetails` and calls `StudentAge()`. On the right, the 'Result' pane shows the command `$javac StudentDetails.java` and `$java -Xmx128M -Xms16M StudentDetails`, followed by the output `Student age is : 21`.

```

1 public class StudentDetails {
2     public void StudentAge()
3     {
4         // local variable age
5         int age = 0;
6         age = age + 21;
7         System.out.println("Student age is : " + age);
8     }
9
10    public static void main(String args[])
11    {
12        StudentDetails obj = new StudentDetails();
13        obj.StudentAge();
14    }
15 }
16

```

```

Result
$javac StudentDetails.java
$java -Xmx128M -Xms16M StudentDetails
Student age is : 21

```

Instance Variables:

Instance variables are non-static variables and are declared in a class outside any method, constructor or block. As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed. Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used. Initialization of Instance Variable is not mandatory. Its default value is 0. Instance Variable can be accessed only by creating objects.

Program:

example we take two instance variables one is name variable of string type and can be accessed by any child class because it is public and the

second is age variable of integer type and can be accessed in the same class record because it is private.

```
public class Record{

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

// this instance age variable is visible in Record class only.
    private int age;
    public Record (String RecName)
    {
        name = RecName;
    }
    public void setAge(int RecSal)
    {
        age = RecSal;
    }
    public void printRec()
    {
        // print the value for "name"
        System.out.println("name : " + name );
    }
}
```

```

        //prints the value for "age"
        System.out.println("age :" + age);
    }
    public static void main(String args[])
    {
        Record r = new Record("Mehran Shah");
        r.setAge(22);
        r.printRec();
    }
}

```

Output:

name : Mehran Shah

age :22

The screenshot shows an IDE window with two panes. The left pane contains the source code for a Java class named 'Record'. The code defines a class with a public 'name' field, a private 'age' field, and methods for setting age and printing the record details. The main method creates a Record object, sets its age to 22, and calls the printRec method. The right pane shows the command prompt output, which matches the expected output: 'name : Mehran Shah' followed by 'age :22' on the next line.

```

1 public class Record{
2
3 // this instance variable is visible for any child class.
4 public String name;
5
6 // this instance age variable is visible in Record class only.
7
8 private int age;
9 public Record (String RecName)
10 {
11     name = RecName;
12 }
13
14 public void setAge(int RecSal)
15 {
16     age = RecSal;
17 }
18
19 public void printRec()
20 {
21     // print the value for "name"
22     System.out.println("name : " + name );
23
24     //prints the value for "age"
25     System.out.println("age :" + age);
26 }
27
28 public static void main(String args[])
29 {
30     Record r = new Record("Mehran Shah");
31     r.setAge(22);
32     r.printRec();
33 }
34 }

```

```

$javac Record.java
$java -Xmx128M -Xms16M Record
name : Mehran Shah
age :22

```

Static Variables:

Static variables are also known as Class variables.

These variables are declared similarly as instance variables, the difference is that static variables are declared using the static keyword within a class outside any method constructor or block. Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create. Static variables are created at the start of program execution and destroyed automatically when execution ends. Initialisation of Static Variable is not mandatory. Its default value is 0. If we access the static variable like Instance variable (through an object), the compiler will show the warning message and it won't halt the program. The compiler will replace the object name to class name automatically. If we access the static variable without the class name, compiler will automatically append the class name.

Program:

```
import java.io.*;

class Emp {

    // static variable salary
    public static double salary;
    public static String name = "Mehran Shah";
}

public class EmpDemo {
    public static void main(String args[])
    {

        // accessing static variable without object
```

```

Emp.salary = 50000;

System.out.println(Emp.name + "'s average salary:"
                    + Emp.salary);

    }
}

```

Output:

Mehran Shah's average salary:50000

The screenshot shows an IDE window with two panes. The left pane displays the source code for a Java program. The right pane shows the execution result.

```

1 import java.io.*;
2 class Emp {
3
4     // static variable salary
5     public static double salary;
6     public static String name = "Mehran Shah";
7 }
8
9 public class EmpDemo {
10     public static void main(String args[])
11     {
12
13         // accessing static variable without object
14         Emp.salary = 50000;
15         System.out.println(Emp.name + "'s average salary:"
16                             + Emp.salary);
17     }
18 }
19

```

The right pane, titled "Result", shows the following output:

```

$javac EmpDemo.java
$java -Xmx128M -Xms16M EmpDemo
Mehran Shah's average salary:50000.0

```

Question No 2:

Answer:

if:

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.

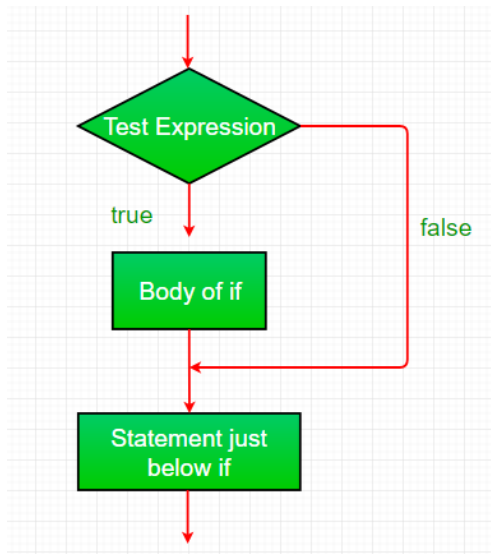
Syntax:

Here, condition after evaluation will be either true or false. if statement accepts boolean values if the value is true then it will execute the block of statements under it.

if(condition)

```
{  
  // Statements to execute if  
  // condition is true  
}
```

Flowchart:



Program:

```
public class IfStatementExample {  
  
  public static void main(String args[]){  
    int num=70;  
    if( num < 100 ){  
      /* This println statement will only execute,  
      * if the above condition is true
```



```
        */
        System.out.println("number is less than 100");
    }
}
}
```

Output:

Number is less than 100

Question No 3:

Answer:

if-else-if Statement:

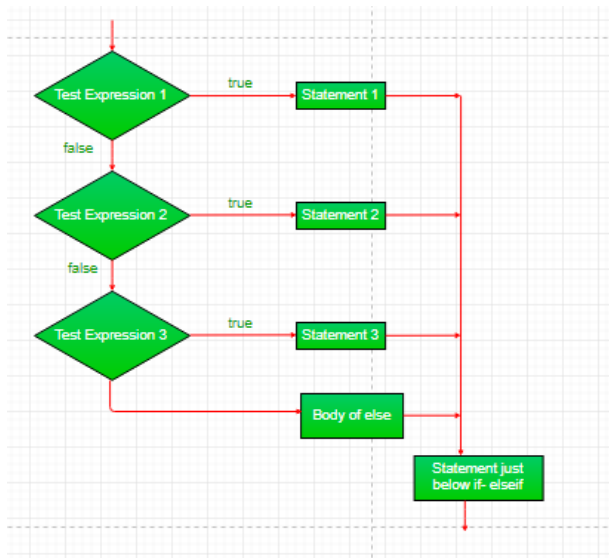
if-else-if statement is used when we need to check multiple conditions. In this statement we have only one “if” and one “else”, however we can have multiple “else if”. It is also known as if else if ladder.

Syntax:

```
if (expression1) {
    // codes
}
else if(expression2) {
    // codes
}
else if (expression3) {
    // codes
```

```
}  
.  
.  
else {  
    // codes  
}
```

Flowchart:



Program:

```
public class IfElseIfExample {  
  
    public static void main(String args[]){  
        int num=124;  
        if(num <100 && num>=1) {  
            System.out.println("Its a two digit number");  
        }  
    }  
}
```

```
else if(num <1000 && num>=100) {  
    System.out.println("Its a three digit number");  
}  
else if(num <10000 && num>=1000) {  
    System.out.println("Its a four digit number");  
}  
else if(num <100000 && num>=10000) {  
    System.out.println("Its a five digit number");  
}  
else {  
    System.out.println("number is not between 1 & 99999");  
}  
}  
}
```

Output:

```
Execute | > Share | Source File | STDIN | Result
1 public class IfElseIfExample {
2
3     public static void main(String args[]){
4         int num=155554;
5         if(num <100 && num>=1) {
6             System.out.println("Its a two digit number");
7         }
8         else if(num <1000 && num>=100) {
9             System.out.println("Its a three digit number");
10        }
11       else if(num <10000 && num>=1000) {
12           System.out.println("Its a four digit number");
13       }
14       else if(num <100000 && num>=10000) {
15           System.out.println("Its a five digit number");
16       }
17       else {
18           System.out.println("number is not between 1 & 99999");
19       }
20   }
21 }
22
```

\$javac IfElseIfExample.java
\$java -Xmx128M -Xms16M IfElseIfExample
number is not between 1 & 99999

```
Execute | > Share | Source File | STDIN | Result
1 public class IfElseIfExample {
2
3     public static void main(String args[]){
4         int num=14;
5         if(num <100 && num>=1) {
6             System.out.println("Its a two digit number");
7         }
8         else if(num <1000 && num>=100) {
9             System.out.println("Its a three digit number");
10        }
11       else if(num <10000 && num>=1000) {
12           System.out.println("Its a four digit number");
13       }
14       else if(num <100000 && num>=10000) {
15           System.out.println("Its a five digit number");
16       }
17       else {
18           System.out.println("number is not between 1 & 99999");
19       }
20   }
21 }
22
```

\$javac IfElseIfExample.java
\$java -Xmx128M -Xms16M IfElseIfExample
Its a two digit number

```
Execute | > Share | Source File | STDIN | Result
1 public class IfElseIfExample {
2
3     public static void main(String args[]){
4         int num=124;
5         if(num <100 && num>=1) {
6             System.out.println("Its a two digit number");
7         }
8         else if(num <1000 && num>=100) {
9             System.out.println("Its a three digit number");
10        }
11       else if(num <10000 && num>=1000) {
12           System.out.println("Its a four digit number");
13       }
14       else if(num <100000 && num>=10000) {
15           System.out.println("Its a five digit number");
16       }
17       else {
18           System.out.println("number is not between 1 & 99999");
19       }
20   }
21 }
22
```

\$javac IfElseIfExample.java
\$java -Xmx128M -Xms16M IfElseIfExample
Its a three digit number

Question No 4:

Answer:

Loops in Java:

Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true. Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

While Loop

Do-while Loop

For Loop

While Loop:

A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement. While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called Entry control loop. Once the condition is evaluated to true, the statements in the loop body are executed. Normally the statements contain an update value for the variable being processed for the next iteration. When the condition becomes false, the loop terminates which marks the end of its life cycle.

Syntax :

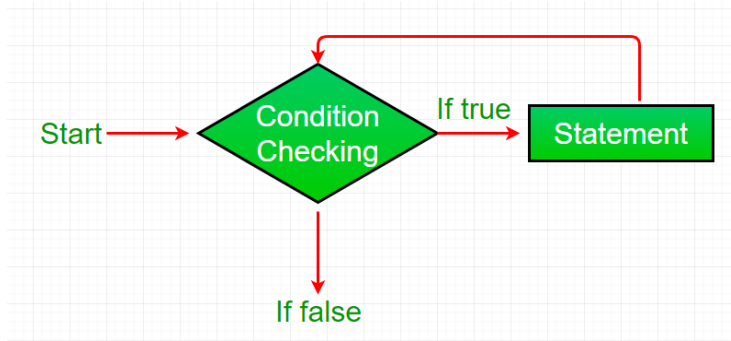
while (boolean condition)

{

 loop statements...

}

Flowchart:



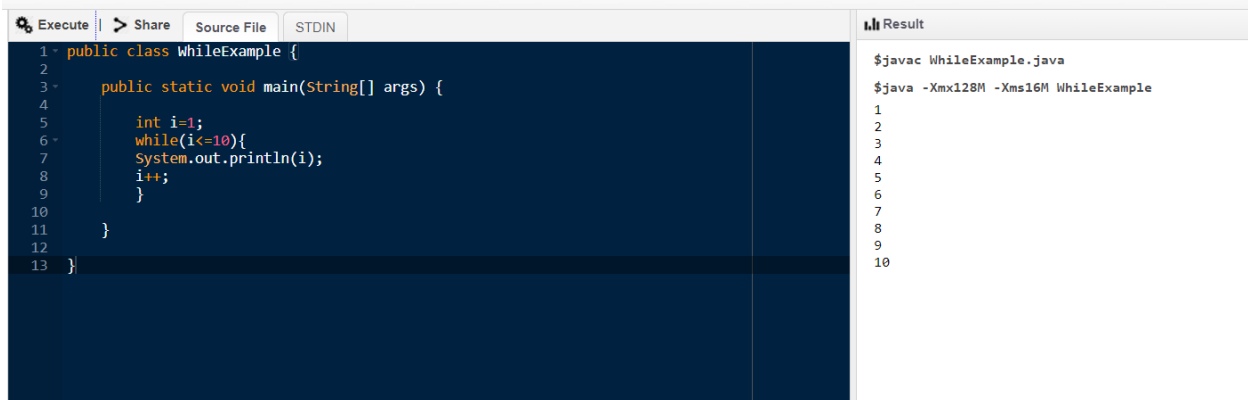
Program:

```
public class WhileExample {  
  
    public static void main(String[] args) {  
  
        int i=1;  
        while(i<=10){  
            System.out.println(i);  
            i++;  
        }  
    }  
}
```

Output:

1
2
3
4
5

6
7
8
9
10



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

Result

```
$javac WhileExample.java
$java -Xmx128M -Xms16M WhileExample
1
2
3
4
5
6
7
8
9
10
```

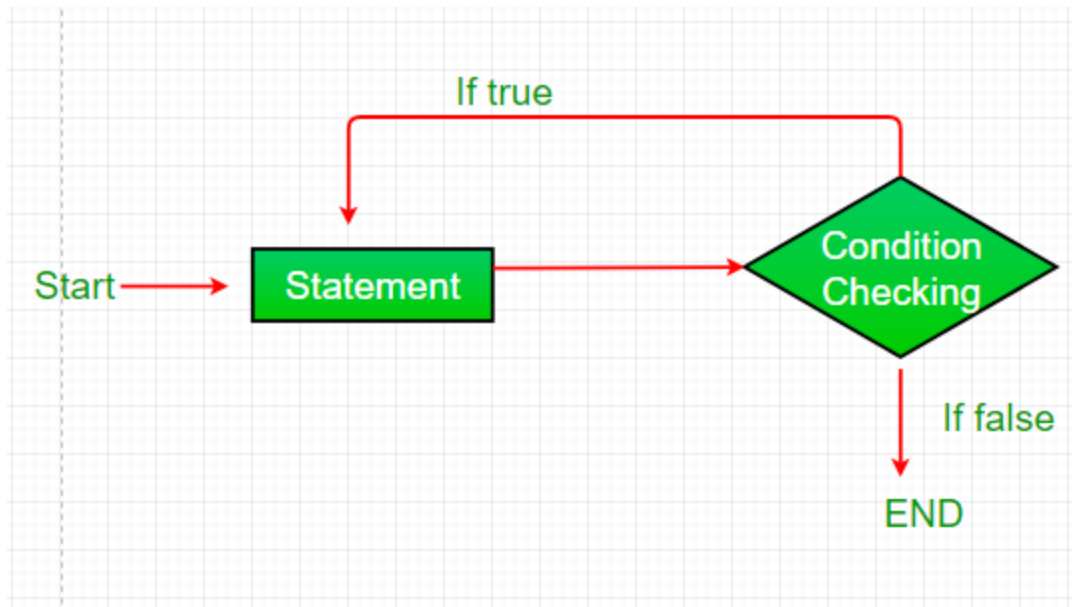
Do-while:

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

Syntax:

```
do
{
    statements..
}
while (condition);
```

Flowchart:



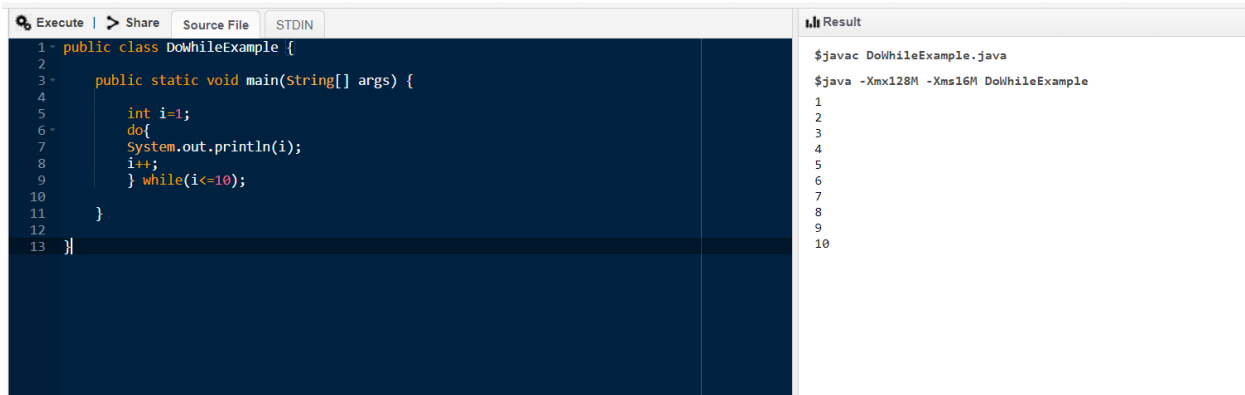
Program:

```
public class DoWhileExample {  
  
    public static void main(String[] args) {  
        int i=1;  
        do{  
            System.out.println(i);  
            i++;  
        } while(i<=10);  
    }  
}
```

Output:

1
2

3
4
5
6
7
8
9
10



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

Result

```
$javac DoWhileExample.java  
$java -Xmx128M -Xms16M DoWhileExample  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

For Loop:

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times. A for loop is useful when you know how many times a task is to be repeated.

Syntax:

```
for (initialization condition; testing condition;  
      increment/decrement)  
{  
    statement(s)
```

}

Initialization condition: Here, we initialize the variable in use. It marks the start of a for loop. An already declared variable can be used or a variable can be declared, local to loop only.

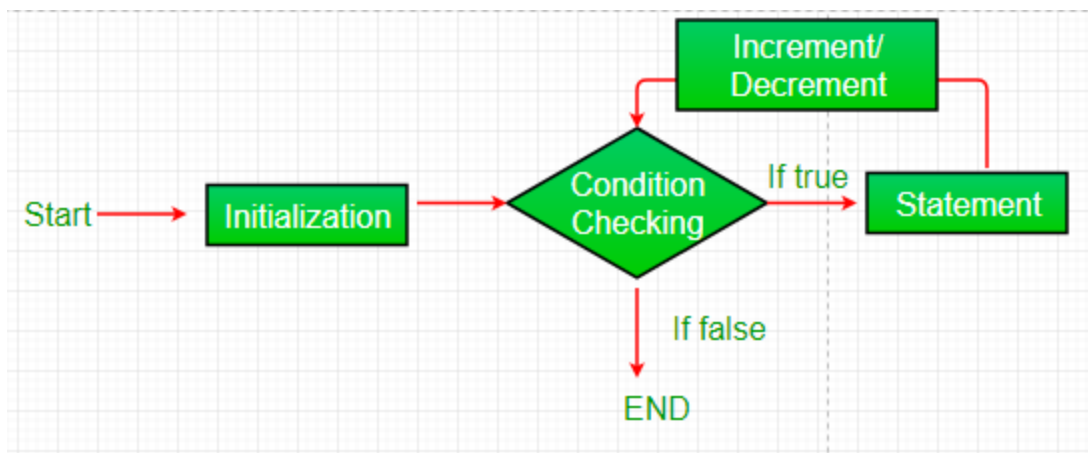
Testing Condition: It is used for testing the exit condition for a loop. It must return a boolean value. It is also an Entry Control Loop as the condition is checked prior to the execution of the loop statements.

Statement execution: Once the condition is evaluated to true, the statements in the loop body are executed.

Increment/ Decrement: It is used for updating the variable for next iteration.

Loop termination: When the condition becomes false, the loop terminates marking the end of its life cycle.

Flowchart:



Program:

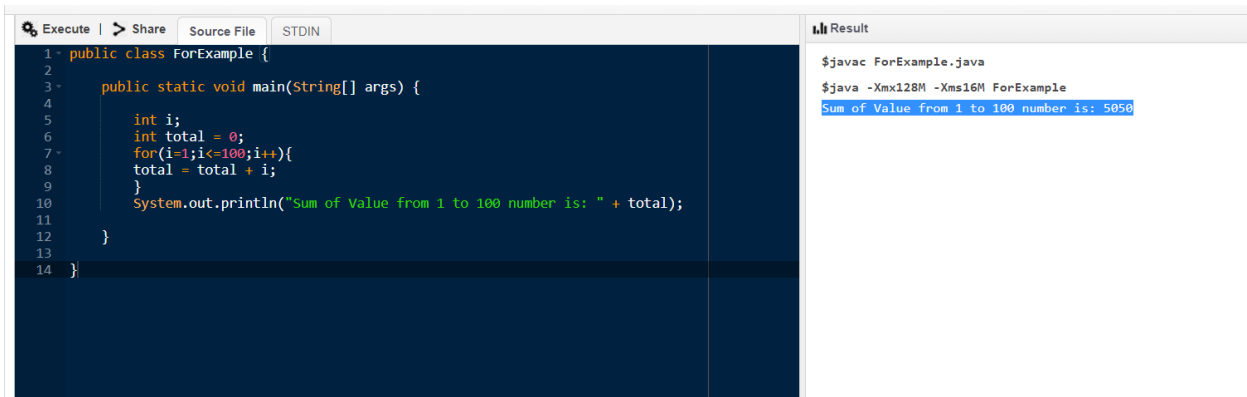
A program to find the sum of 1 to 100 number using for Loop:

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

```
        int i;
        int total = 0;
        for(i=1;i<=100;i++){
            total = total + i;
        }
        System.out.println("Sum of Value from 1 to 100 number is: "
+ total);
    }
}
```

Output:

Sum of Value from 1 to 100 number is: 5050



The screenshot shows an IDE window with two panes. The left pane displays the source code for a Java class named 'ForExample'. The code is as follows:

```
1 public class ForExample {
2
3     public static void main(String[] args) {
4
5         int i;
6         int total = 0;
7         for(i=1;i<=100;i++){
8             total = total + i;
9         }
10        System.out.println("Sum of Value from 1 to 100 number is: " + total);
11    }
12 }
13
14 }
```

The right pane, titled 'Result', shows the command prompt output for the execution of the code:

```
$javac ForExample.java
$java -Xmx128M -Xms16M ForExample
Sum of Value from 1 to 100 number is: 5050
```

Question No 5:

Answer:

package strings;

import java.util.Scanner;

```
public class MultiplicationTables{  
    public static void main (String[] args) {  
        Integer number,i;  
        Scanner sc= new Scanner(System.in);  
        System.out.println("Enter the number to print the table");  
        number=sc.nextInt();  
        for (i=10;i>=1 ;i++){  
            System.out.println(number+"*"+i+"="+number*i);  
        }  
    }  
}
```

Output:

Enter the number to print the table:7

7 * 10 = 70

7 * 9 = 63

7 * 8 = 56

7 * 7 = 49

7 * 6 = 42

7 * 5 = 35

7 * 4 = 28

7 * 3 = 21

$$7 * 2 = 14$$

$$7 * 1 = 7$$