ID:11757

NAME: SALMAN KHAN

SUBJECT: OBJECT ORIENTED PROGRAMMING

TEACHER: M. AYUB KHAN

DATE: 30/09/2020

## ANS 1: VAIABLES:

A variable provides us with named storage that our programs can manipulate. Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used. Following is the basic form of a variable declaration −

data type variable [ = value][, variable [ = value] ...] ;

Here data type is one of Java's datatypes and variable is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java −

### Example:

```
int a, b, c;              // Declares three ints, a, b, and c.

int a = 10, b = 10;    // Example of initialization

byte B = 22;                // initializes a byte type variable B.

double pi = 3.14159; // declares and assigns a value of PI.

char a = 'a';              // the char variable a iis initialized with value 'a'
```

### EXAMPLE:

```
{
    public static void main(String args[])
     {
           byte b;
           short s;
        int i;
           float f;
           char c;
           double d;
           boolean boo;
           long l;
           System.out.println("Examples created succesfully");
    }
}
```

## Output:

```
Examples created successfully
```

## TYPES OF VARIABLES

Local variables

instant avriables

class/static variables

**LOCAL VARIABLES:** These variables are declared inside method of the class. Their scope is limited to the method which means that You can’t change their values and access them outside of the method.

In this example, I have declared the instance variable with the same name as local variable, this is to demonstrate the scope of local variables.

## For Example:

```
class DataFlair {

   int a = 9,

   b = 10;

   void LearnJava {

      int local_j = 45; // A local variable

      String s = ”DataFlair Training”; //A local variable

   }

}
```

## EXAMLE OF LOCAL VARIABLES:

```
public class VariableExample {

    // instance variable

    public String myVar="instance variable";


    public void myMethod(){

        // local variable

        String myVar = "Inside Method";

        System.out.println(myVar);

    }

    public static void main(String args[]){

        // Creating object

        VariableExample obj = new VariableExample();
        /* We are calling the method, that changes the
         * value of myVar. We are displaying myVar again after
         * the method call, to demonstrate that the local
         * variable scope is limited to the method itself.
         */
        System.out.println("Calling Method");
        obj.myMethod();
        System.out.println(obj.myVar);
    }
}
```

## Output:

```
Calling Method
Inside Method
instance variable
```

**INSTANT VARIABLES:** Each instance(objects) of class has its own copy of instance variable. Unlike static variable, instance variables have their own separate copy of instance variable. We have changed the instance variable value using object obj2 in the following program and when we displayed the variable using all three objects, only the obj2 value got changed, others remain unchanged. This shows that they have their own copy of instance variable.

## EXAMLE O INSTANT VARIABLES:

```
public class InstanceVarExample {
    String myInstanceVar="instance variable";
    public static void main(String args[]){
        InstanceVarExample obj = new InstanceVarExample();
        InstanceVarExample obj2 = new InstanceVarExample();
        InstanceVarExample obj3 = new InstanceVarExample();
        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
        System.out.println(obj3.myInstanceVar);
        obj2.myInstanceVar = "Changed Text";
        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
        System.out.println(obj3.myInstanceVar);
    }
}
```

## Output:

```
instance variable
instance variable
instance variable
instance variable
Changed Text
instance variable
```

## CLASS/STATIC VARIABLES:

Static variables are also known as class variable because they are associated with the class and common for all the instances of class. For example, If I create three objects of a class and access this static variable, it would be common for all, the changes made to the variable using one of the object would reflect when you access it through other objects. .

## EXAMPLE OF CLASS/ STATIC VARIABLES:

```
public class StaticVarExample {
    public static String myClassVar="class or static variable";

    public static void main(String args[]){

        StaticVarExample obj = new StaticVarExample();

        StaticVarExample obj2 = new StaticVarExample();

        StaticVarExample obj3 = new StaticVarExample();

        //All three will display "class or static variable"

        System.out.println(obj.myClassVar);

        System.out.println(obj2.myClassVar);

        System.out.println(obj3.myClassVar);

        //changing the value of static variable using obj2

        obj2.myClassVar = "Changed Text";

        //All three will display "Changed Text"

        System.out.println(obj.myClassVar);

        System.out.println(obj2.myClassVar);

        System.out.println(obj3.myClassVar);

    }

}
```

## Output:

```
class or static variable

class or static variable

class or static variable

Changed Text

Changed Text

Changed Text
```

## ANS 2: IF STATEMENT:
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. Control falls into the if block.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.An **if statement** consists of a boolean expression followed by one or more statements.

Java Conditions and If Statements

Java supports the usual logical conditions from mathematics:

Less than: a < b

Less than or equal to: a <= b

Greater than: a > b

Greater than or equal to: a >= b

Equal to a == b

Not Equal to: a != b

You can use these conditions to perform different actions for different decisions.

Java has the following conditional statements:

Use if to specify a block of code to be executed, if a specified condition is true

Use else to specify a block of code to be executed, if the same condition is false

Use else if to specify a new condition to test, if the first condition is false

Use switch to specify many alternative blocks of code to be executed

The if Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

## Syntax:

```
if (condition) { // block of code to be executed if the condition is true }
```

## Example:

```
if (20 > 18) { System.out.println("20 is greater than 18"); }
```

## Output:

```
20 is greater than 18
```

## EXAMPLE: 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
```

## ANS 3: IF-ELSE-IF:

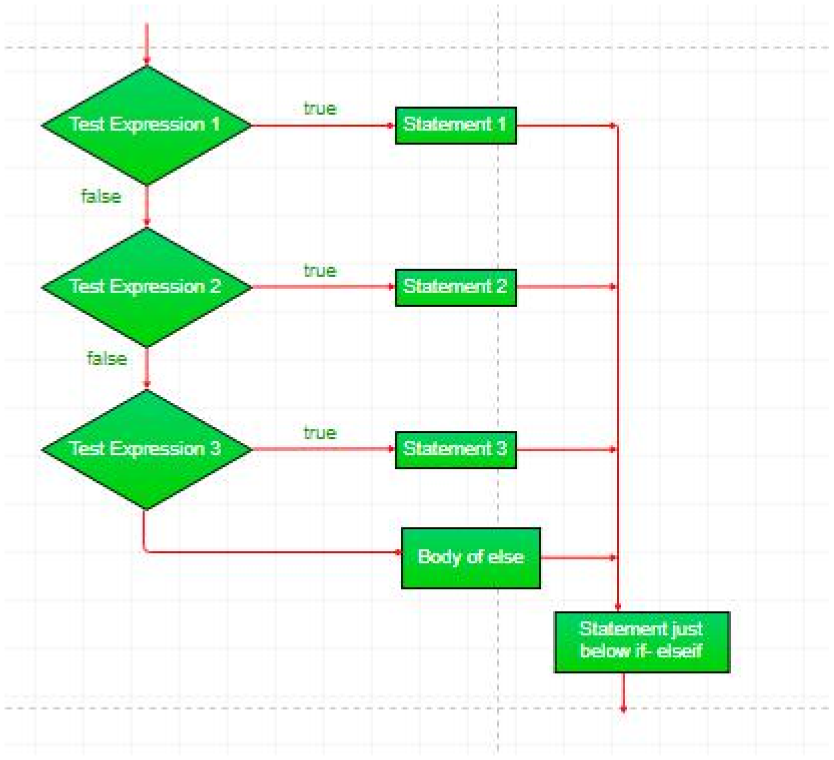
In Java, we have an if...else...if ladder, that can be used to execute one block of code among multiple other blocks.

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

Here, if statements are executed from the top towards the bottom. As soon as the test expression is true, codes inside the body of that the if statement is executed. Then, the control of the program jumps outside the if-else-if ladder.

If all test expressions are false, codes inside the body of else is executed**.**Here, a user can 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.Java has the following conditional statements: Use if to specify a block of code to be executed, if a specified condition is true. Use else to specify a block of code to be executed, if the same condition is false. Use else if to specify a new condition to test, if the first condition is false.

```
if (condition)

      statement;

else if (condition)

      statement;

.

.

else

      statement;
```



EXAMPLE: // Java program to illustrate if-else-if ladder

```
class ifelseifDemo
{
    public static void main(String args[])
    {
        int i = 20;

        if (i == 10)
            System.out.println("i is 10");
        else if (i == 15)
            System.out.println("i is 15");
        else if (i == 20)
            System.out.println("i is 20");
        else
```

```
                    System.out.println("i is not present");
        }
}
```

## Output:

```
i is 20
```

## ANS 4: lOOPS: 
Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.

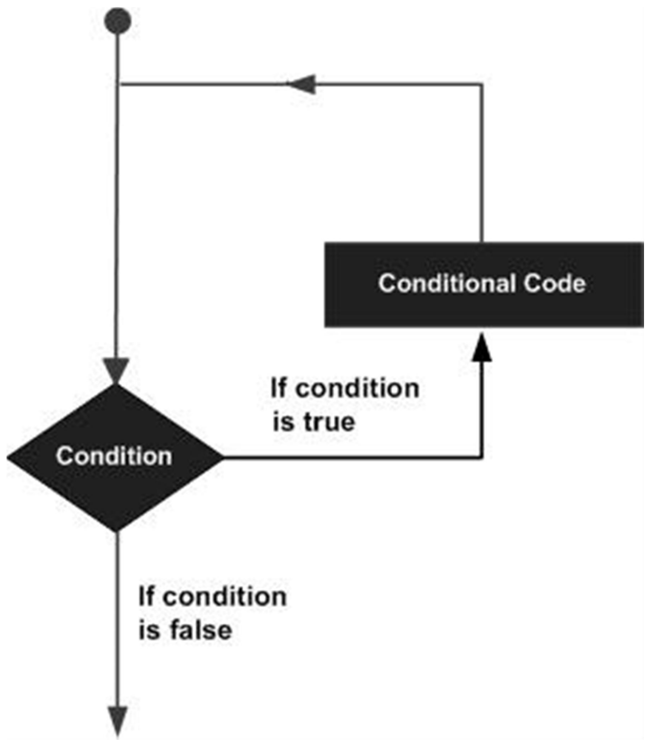
Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

Loops are used to execute a set of statements repeatedly until a particular condition is satisfied. In Java we have three types of basic loops: for, while and do-while. In this tutorial we will learn how to use “for loop” in Java

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

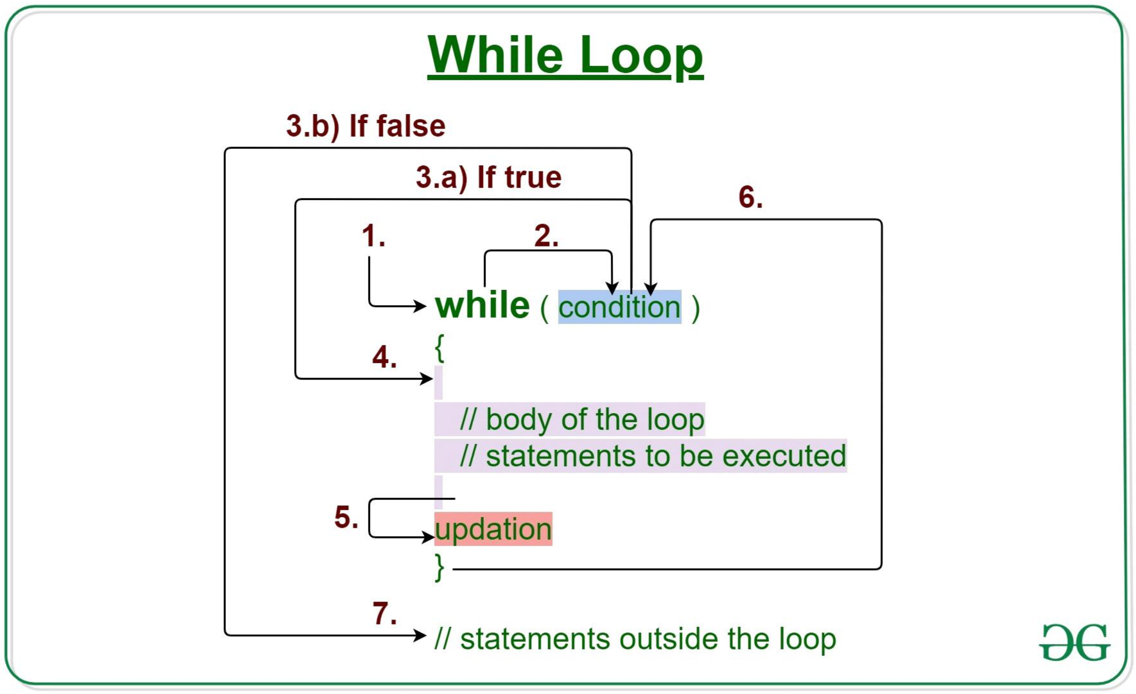
Programming languages provide various control structures that allow for more complicated execution paths.

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 −



## TYPES OF LOOPS:

WHILE LOOP: Java 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 ( condition )
{

    // body of the loop
    // statements to be executed

updation
}
// statements outside the loop
```

## Syntax:

```
while (test_expression)

{

      // statements



    update_expression;

}
```

The various parts of the While loop are:

**Test Expression:** In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of the loop and go to update expression. Otherwise, we will exit from the while loop.

## Example:

i <= 10

**Update Expression:** After executing the loop body, this expression increments/decrements the loop variable by some value.

Example:

i++;

How does a While loop executes?

Control falls into the while loop.

The flow jumps to Condition

Condition is tested.

If Condition yields true, the flow goes into the Body.

If Condition yields false, the flow goes outside the loop

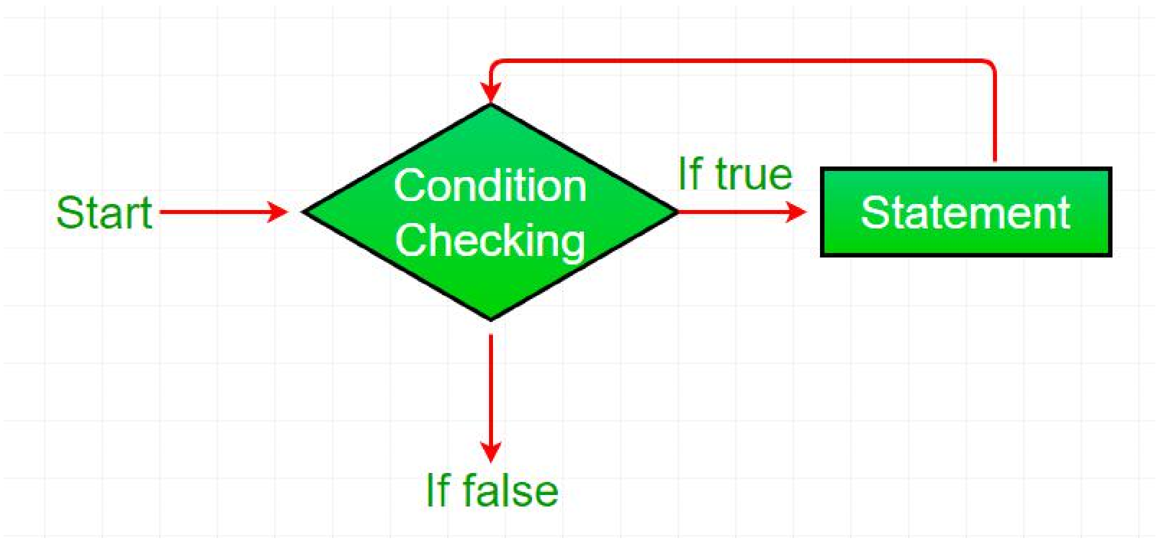
The statements inside the body of the loop get executed.

Updation takes place.

Control flows back to Step 2.

The do-while loop has ended and the flow has gone outside.

Flow chart while loop (for Control Flow):



Example : This program will try to print “Hello World” 5 times.

filter_none

edit

play_arrow

brightness_4

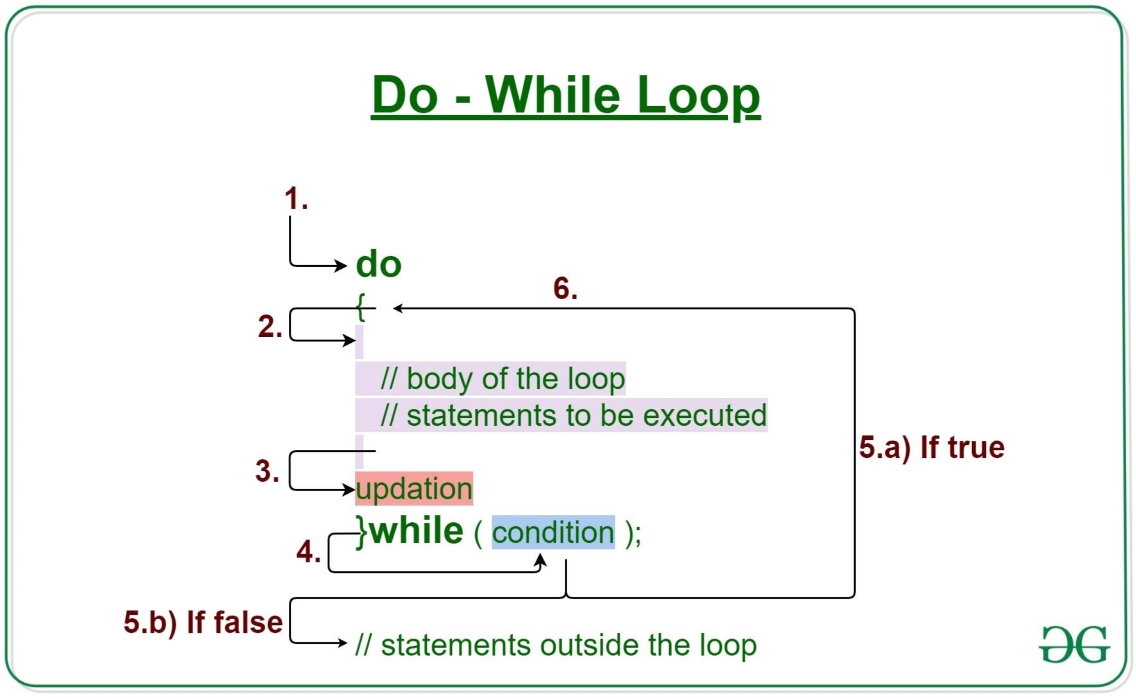
```
// Java program to illustrate while loop.

class whileLoopDemo {

    public static void main(String args[])

    {

        // initialization expression

        int i = 1;

        // test expression

        while (i < 6) {

            System.out.println("Hello World");
```

```
                    // update expression

                    i++;

                }

        }

}
```

## Output:

```
Hello World

Hello World

Hello World

Hello World

Hello World
```

## DO WHILE LOOP:
Java do-while loop is an Exit control loop. Therefore, unlike for or while loop, a do-while check for the condition after executing the statements or the loop body.



## Syntax:

```
do

{

        // loop body

        update_expression

}

while (test_expression);
```

The various parts of the do-while loop are:

**Test Expression:** In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of the loop and go to update expression. Otherwise, we will exit from the while loop.

## Example:

i <= 10

**Update Expression:** After executing the loop body, this expression increments/decrements the loop variable by some value.

## Example:

i++;

How does a do-While loop executes?

Control falls into the do-while loop.

The statements inside the body of the loop get executed.

Updation takes place.

The flow jumps to Condition

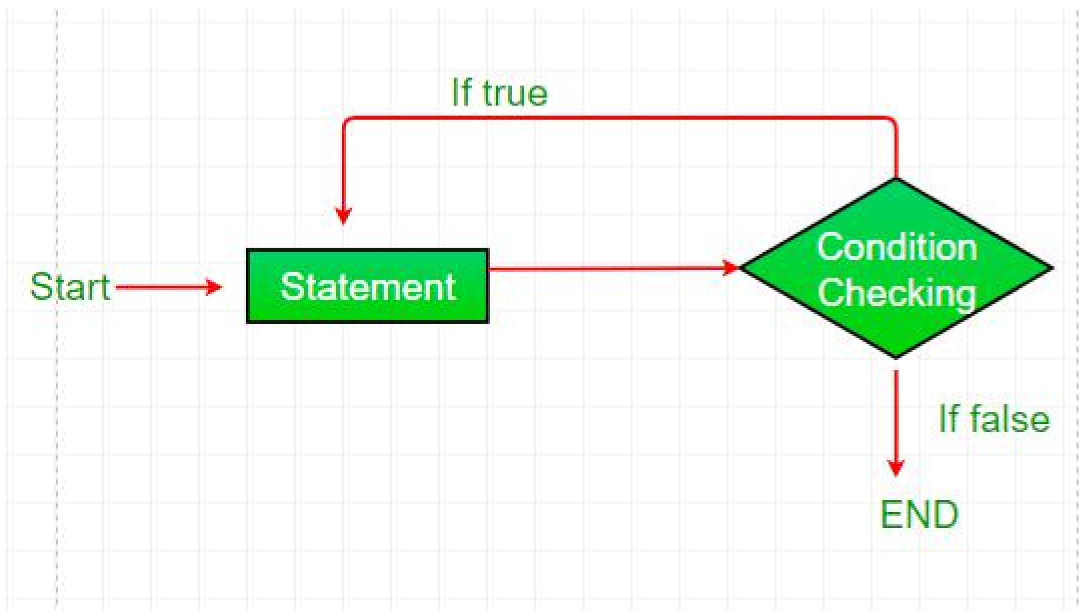
Condition is tested.

If Condition yields true, goto Step 6.

If Condition yields false, the flow goes outside the loop

Flow goes back to Step 2.

## Flowchart do-while loop:



## EXAMPLE:

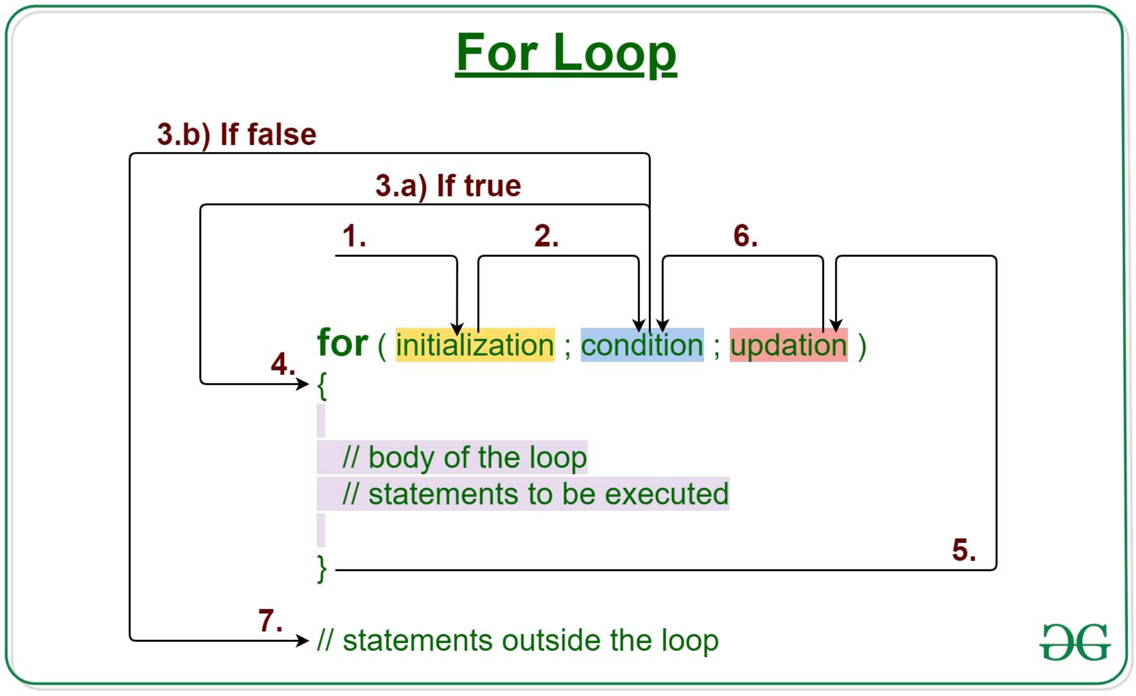
Java program to illustrate the do-while loop

```
class dowhileloopDemo {
    public static void main(String args[])
    {
        int x = 21, sum = 0;
        do {
            // The line will be printed even
            // if the condition is false
            sum += x;
            x--;
        } while (x > 10);
        System.out.println("Summation: " + sum);
    }
}
```

## Output:

Summation: 176

## FOR LOOP: 
Java for loop provides a concise way of writing the loop structure. The for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.



For Loop
3.b) If false
3.a) If true
1.
2.
6.
for ( initialization ; condition ; updation )
4.
{
// body of the loop
// statements to be executed
}
5.
7.
// statements outside the loop

## Syntax:

```
for (initialization expr; test expr; update exp
{
        // body of the loop
        // statements we want to execute
}
```

The various parts of the For loop are:

## Initialization Expression: 
In this expression we have to initialize the loop counter to some value.

Example:

int i=1;

## Test Expression: In this expression we have to test the condition. If the condition evaluates to true then we will execute the body of the loop and go to update expression. Otherwise, we will exit from the for loop.

## Example:

i <= 10

## Update Expression: After executing the loop body, this expression increments/decrements the loop variable by some value.

## Example:

i++;

How does a For loop executes?

Control falls into the for loop. Initialization is done

The flow jumps to Condition

Condition is tested.

If Condition yields true, the flow goes into the Body

If Condition yields false, the flow goes outside the loop

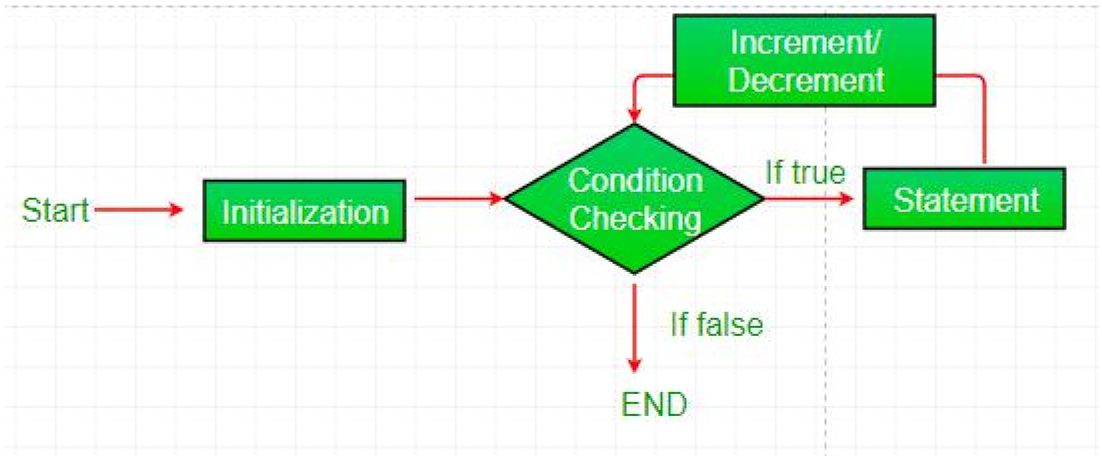
The statements inside the body of the loop get executed.

The flow goes to the Updation

Updation takes place and the flow goes to Step 3 again

The for loop has ended and the flow has gone outside.

## Flow chart for loop (For Control Flow):



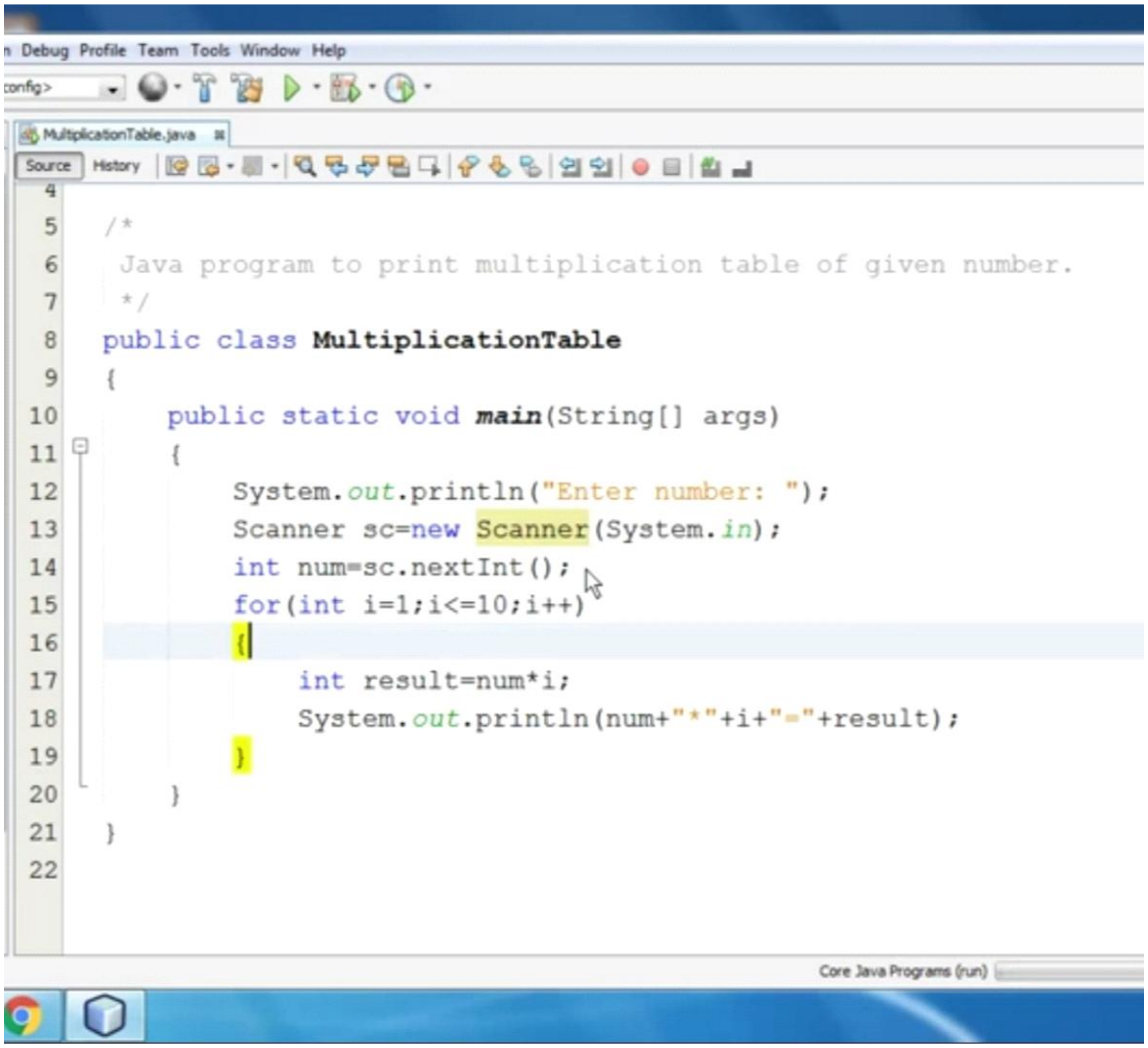
## EXAMPLE: // Java program to illustrate for loop.

```
class forLoopDemo {
    public static void main(String args[])
    {
        int sum = 0;
        // for loop begins
        // and runs till x <= 20
        for (int x = 1; x <= 20; x++) {
            sum = sum + x;
        }
        System.out.println("Sum: " + sum);
    }
}
```

## Output:

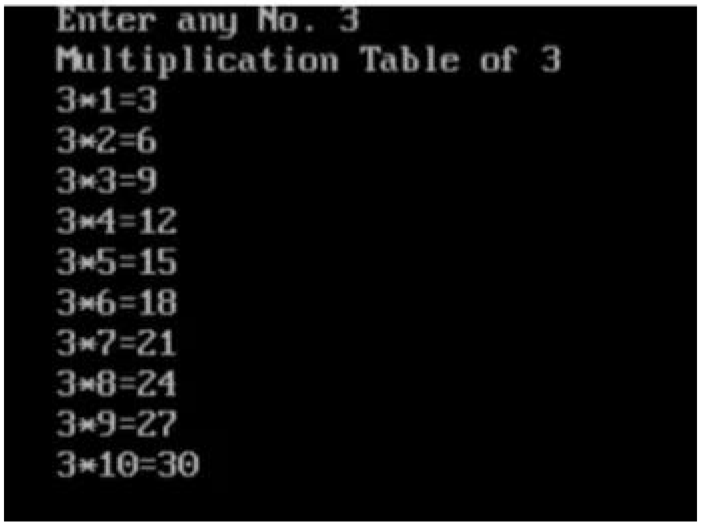
```
Sum: 210
```

## ANS 5:



```java

/*
 Java program to print multiplication table of given number.
 */
public class MultiplicationTable
{
    public static void main(String[] args)
    {
        System.out.println("Enter number: ");
        Scanner sc=new Scanner(System.in);
        int num=sc.nextInt();
        for(int i=1;i<=10;i++)
        {
            int result=num*i;
            System.out.println(num+"*"+i+"="+result);
        }
    }
}

```

OUTPUT:



```
Enter any No. 3
Multiplication Table of 3
3*1=3
3*2=6
3*3=9
3*4=12
3*5=15
3*6=18
3*7=21
3*8=24
3*9=27
3*10=30
```