do-while Statement

Click to Continue

It is **similar** to **while** loop except that the do-while **execute** the block **once**, and then **checks** the **while** condition.

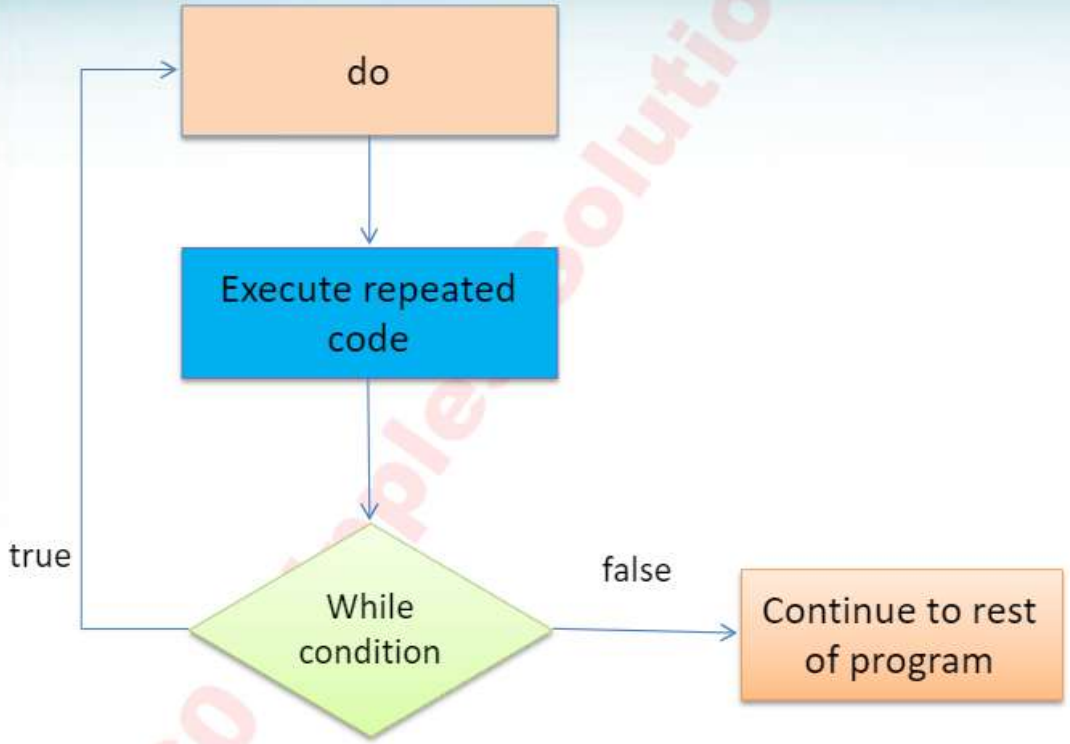
Syntax:

```
do {
    statement1;
    statement2;
} while(boolean_expression);
```

Do not forget to use semicolon after the while statement

Illustration of a do while statement

Click to Continue



Example of a do while statement

```
public class DoWhileExample {

    public static void main(String[] args) {
        int i = 6;
        do {
            System.out.println("i is : " + i);
            i++;
        } while (i < 5);
    }
}
```

The value of i is printed for the first time, even though it does not match the condition i< 5

Output:

```
i is : 6
```

For Statement

Click to Continue

For statement is similar to while loop is used to repeat the execution of the code till a condition is met.

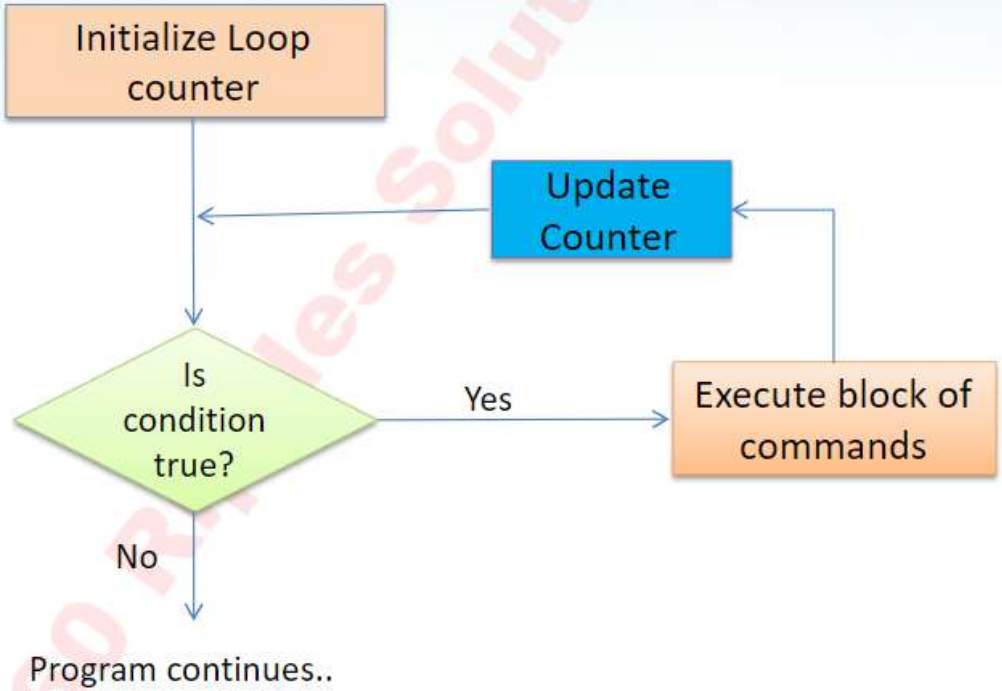
Syntax:

```
for(initialization; loopCondition; iteration) {
    statements;
}
```

- The initialization allows to declare and initialize loop variables this is executed only once.
- The loopCondition compares the loop variable to some limit value. If the loop condition is not met iteration is broken.
- The iteration usually increments or decrements the values of the loop variables

Illustration of a for statement

Click to Continue



For Statement Example

Click to Continue

```
class Example{
        public static void main(String []args){
          for(int  i=1; i< 10; i++){
            Syst m.out.print (“The Numb r is “+i);
          }
    }
```

Loop Value Initialized to 1.

Condition for loop, Loop executed till value is < 10

Loop value incremented.

Statement Executed in loop

Transfer Statement

Click to Continue

The **transfer** Statements in Java alter the normal control flow of the statements. They allow you to redirect the flow of program execution.

Transfer Statements are used to quit either the current iteration of a loop or the entire loop.

1. break.
2. continue.
3. return.

Break Statement

Click to Continue

Break statement used for,

1. Used to terminates a statement sequence in a switch statement.
2. Used to exit loops in Iteration Statement.

Let us see how it is implemented

Problem statement:
This program iterates through the 100 employees and calculate salary . If one employee is minor age, i.e. age < 18 it should break the loop and stop the execution.

```
while (employeecount<=100) {
        if(employeeAge <18)
        {
            break;
        }
        calculateSalary();
}
```

The loop breaks when the age < 18

Continue Statement

Click to Continue

Continue Statements stops the processing the remaining code in the body of the particular iteration Statement and continue with the next loop.

Problem statement:
This program iterates through the 100 employees and calculate salary . If one employee is minor age, i.e. age < 18 it should SKIP the salary calculation logic for the employee and proceed with other employees.

```
while (employeecount<100) {
        if(employeeAge <18)
        {
            continue;
        }
        calculateSalary();
}
```

The loop will not execute calculate salary if age < 18

Let us see how it is implemented

Return Statement

Click to Continue

The return statement stop execution and exits from the current method, and the control flow returns to where the method was invoked.

The return statement has two forms:

- One that returns a value
- One that doesn't.

Illustration 1: To return a value, simply put the value that needs to be returned after the return keyword.

```
return <value/expression>;
```

Illustration 2: When the return type of method is void, use th form of return with no value In this case, the execution of the method is stopped.

```
return;
```

Try it out - Return statement

Click to Continue

```
public class TestProgram {

    public static void main(String[] args) {
        int count = 5;
        int i;
        WelcomeMessage welcome = new WelcomeMessage();
        for (i=1;i<=count;i++){
            welcome.printMessage();
            if(i==3){
                return;
            }
            System.out.println("After if loop "+i);
        }
        System.out.println("Final returned value of i is "+i);
    }
}
```

Let us use the same WelcomeMessage class that we developed for the previous example

Try it out - Return statement

Click to Continue

```
public class TestProgram {

    public static void main(String[] args) {
        int count = 5;
        int i;
        WelcomeMessage welcome = new WelcomeMessage();
        for (i=1;i<=count;i++){
            welcome.printMessage();
            if(i==3){
                return;
            }
            System.out.println("After if loop "+i);
        }
        System.out.println("Final returned value of i is "+i);
    }
}
```

Let us use the same WelcomeMessage class that we developed for the previous example

When i is equal to 3, the return statement is executed and the execution of the method is stopped

Output:

```
Welcome all
After if loop 1
Welcome all
After if loop 2
Welcome all
```

Try the same example with break and continue statement and see how the program behaves.