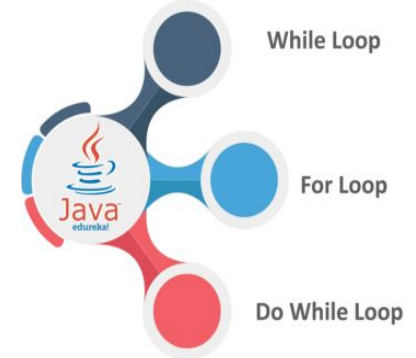
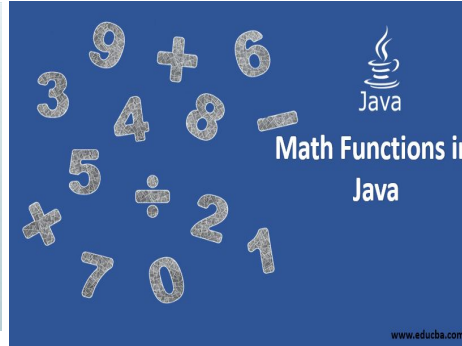
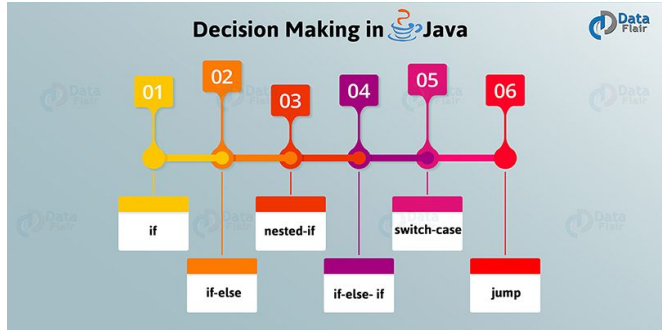


## CHAPTER 2

### SELECTION, MATHEMATICAL FUNCTIONS AND LOOPS

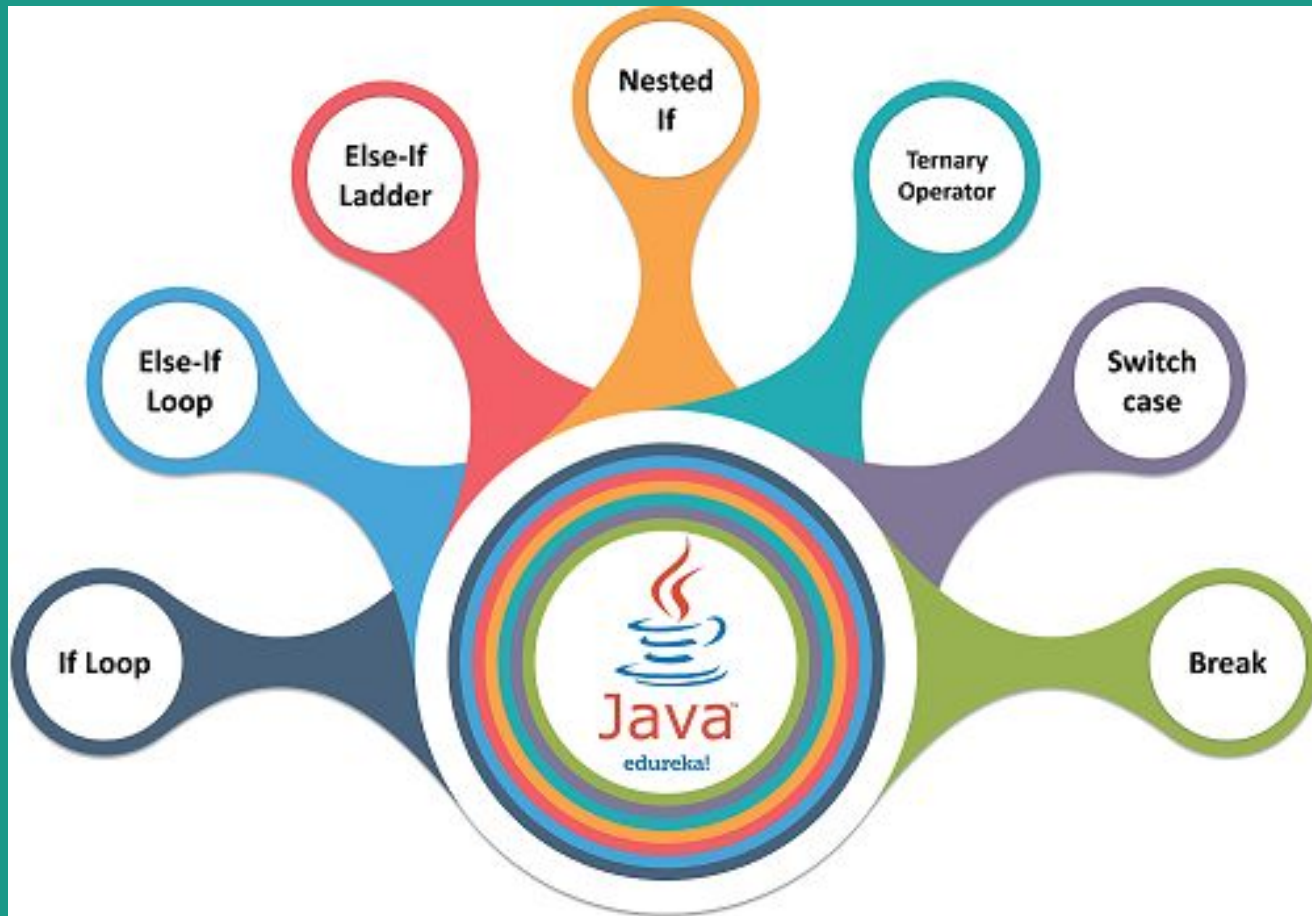


**SUBJECT: OOP-I**  
**CODE: 3140705**

**PREPARED BY:**  
**ASST. PROF. NENSI KANSAGARA**  
**(CSE DEPARTMENT, ACET)**

---

# SELECTIONS

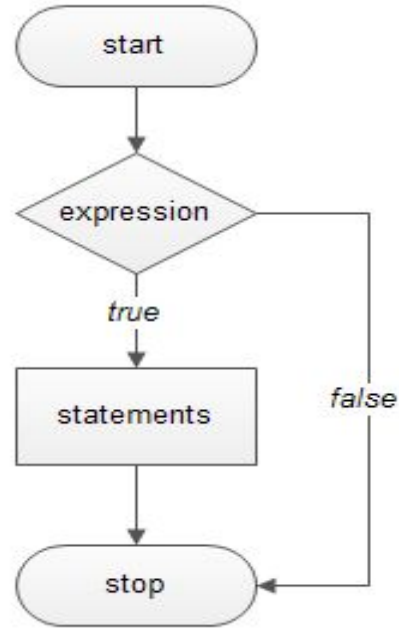


# IF STATEMENTS

The Java if statement tests the condition. It executes the *if block* if condition is true.

## Syntax:

```
if(condition){  
    //code to be executed  
}
```






# WORKING OF IF STATEMENT

Expression is true.

```
int test = 5;

if (test < 10)
{
  // codes
}

// codes after if
```



Expression is false.

```
int test = 5;

if (test > 10)
{
  // codes
}

// codes after if
```



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

        int number = 10;

        // checks if number is greater than 0
        if (number > 0) {
            System.out.println("The number is positive.");
        }
        System.out.println("This statement is always executed.");
    }
}
```

### Output:

```
The number is positive.
This statement is always executed.
```

# TWO WAY IF-ELSE STATEMENT

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

## Syntax:

```
if(condition)
```

```
{
```

```
//code if condition is true
```

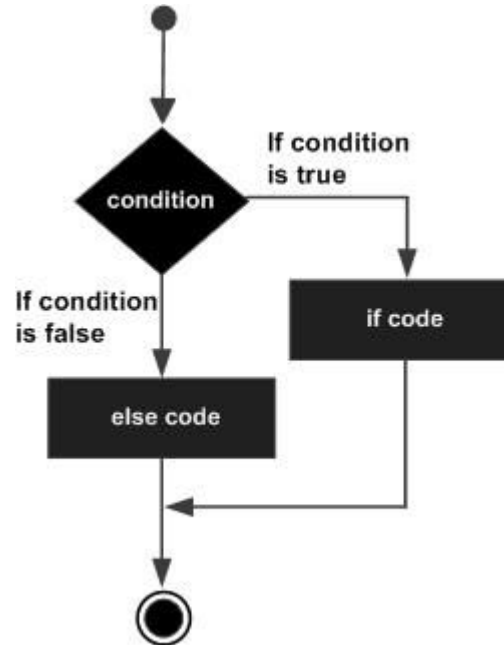
```
}
```

```
Else
```

```
{
```

```
//code if condition is false
```

```
}
```



# WORKING OF IF-ELSE STATEMENT

Expression is true.

```
int test = 5;

if (test < 10)
{
    // body of if
}
else
{
    // body of else
}
```

Expression is false.

```
int test = 5;

if (test > 10)
{
    // body of if
}
else
{
    // body of else
}
```



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

        // checks if number is greater than 0
        if (number > 0) {
            System.out.println("The number is positive.");
        }
        else {
            System.out.println("The number is not positive.");
        }

        System.out.println("This statement is always executed.");
    }
}
```

### Output:

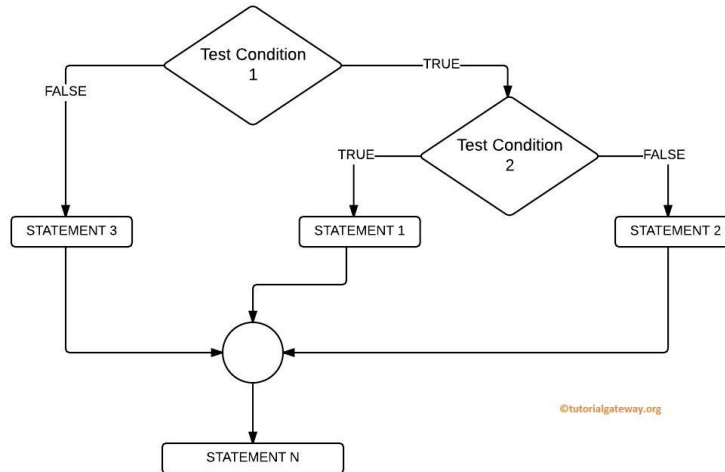
```
The number is positive.
This statement is always executed.
```

# NESTED IF AND MULTI-WAY IF STATEMENTS

The nested if statement represents the *if block within another if block*. Here, the inner if block condition executes only when outer if block condition is true.

**Syntax:**

```
if(condition){  
    //code to be executed  
    if(condition){  
        //code to be executed  
    }  
}
```





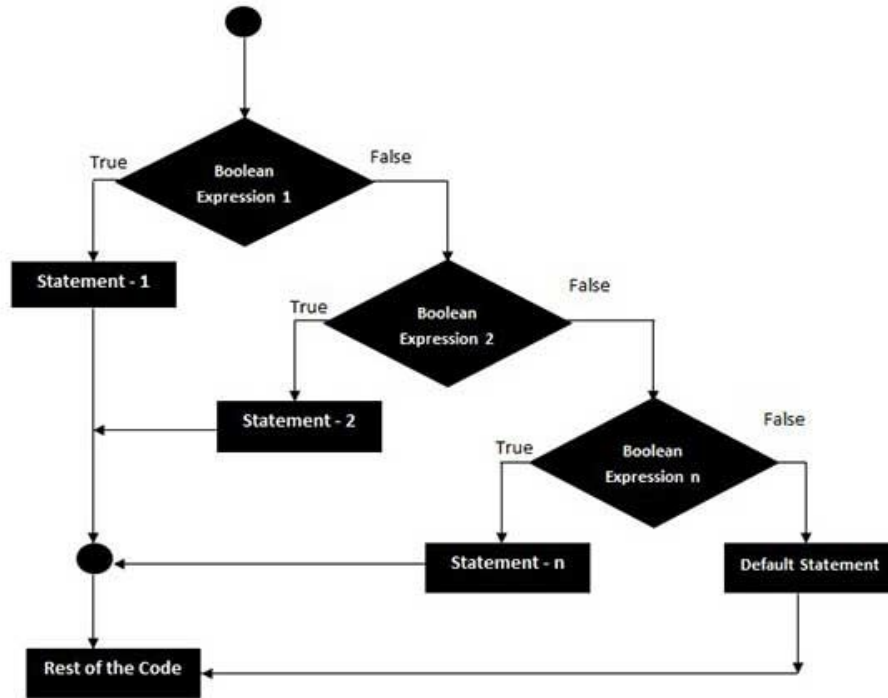
# IF-ELSE STATEMENT

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  
}
```



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

        int number = 0;

        // checks if number is greater than 0
        if (number > 0) {
            System.out.println("The number is positive.");
        }

        // checks if number is less than 0
        else if (number < 0) {
            System.out.println("The number is negative.");
        }
        else {
            System.out.println("The number is 0.");
        }
    }
}
```

### Output:

The number is 0.



# SWITCH STATEMENT

- ❖ The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement. The switch statement works with byte, short, int, long, enum types, String and some wrapper types like Byte, Short, Int, and Long. Since Java 7, you can use strings in the switch statement.
- ❖ In other words, the switch statement tests the equality of a variable against multiple values.
- ❖ **Points to Remember**
  - There can be *one or N number of case values* for a switch expression.
  - The case value must be of switch expression type only. The case value must be *literal or constant*. It doesn't allow variables.



# SWITCH STATEMENT

- The case values must be *unique*. In case of duplicate value, it renders compile-time error.
- The Java switch expression must be of *byte, short, int, long (with its Wrapper type), enums and string*.
- Each case statement can have a *break statement* which is optional. When control reaches to the break statement, it jumps the control after the switch expression. If a break statement is not found, it executes the next case.
- The case value can have a *default label* which is optional.

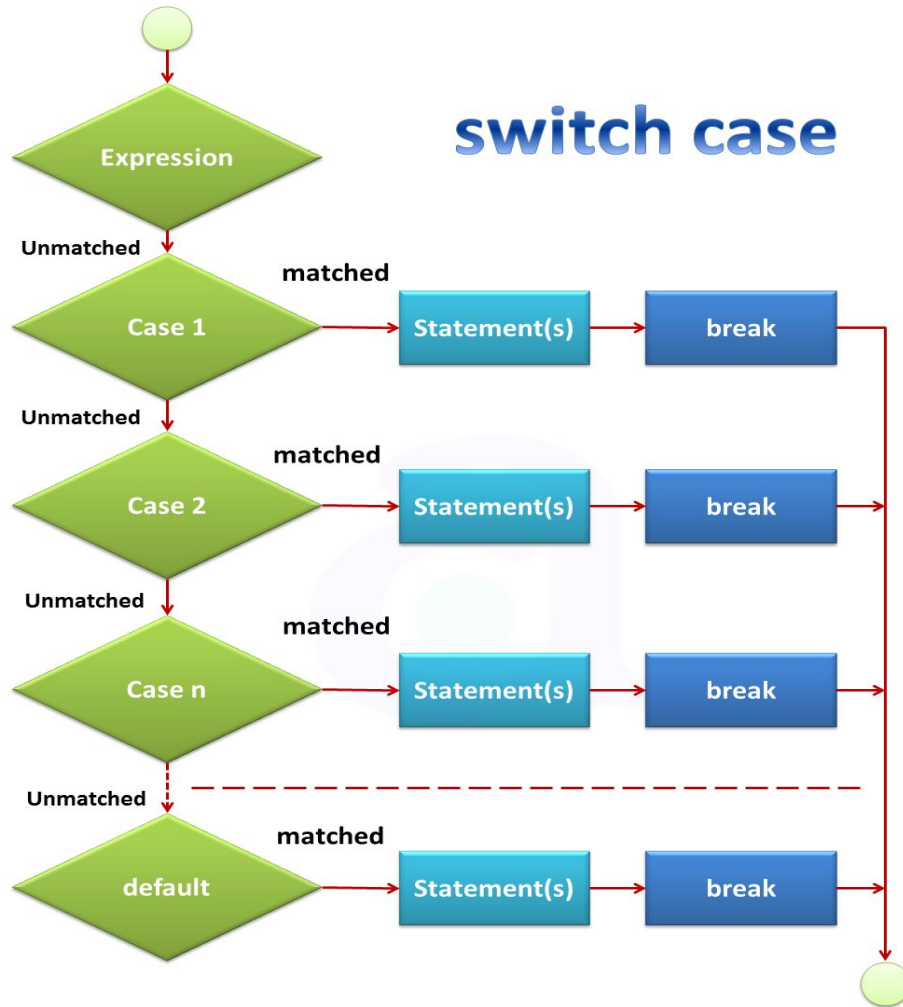


# SYNTAX OF SWITCH

```
switch(expression){  
  case value1:  
    //code to be executed;  
    break; //optional  
  case value2:  
    //code to be executed;  
    break; //optional  
  .....  
  
  default:  
    code to be executed if all cases are not matched;  
}
```



# switch case



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

        int week = 4;
        String day;

        // switch statement to check day
        switch (week) {
            case 1:
                day = "Sunday";
                break;
            case 2:
                day = "Monday";
                break;
            case 3:
                day = "Tuesday";
                break;

            // match the value of week
            case 4:
                day = "Wednesday";
                break;
            case 5:
                day = "Thursday";
                break;
        }
    }
}
```

```
        break;
    case 6:
        day = "Friday";
        break;
    case 7:
        day = "Saturday";
        break;
    default:
        day = "Invalid day";
        break;
    }
    System.out.println("The day is " + day);
}
}
```

### Output:

The day is Wednesday

---

# COMMON MATHEMATICAL FUNCTIONS



# COMMON MATHEMATICAL FUNCTIONS

There are three types of mathematical functions that are commonly used

1. Trigonometric Function
2. Exponent Method
3. Service Method



# TRIGONOMETRIC FUNCTION

METHOD	PURPOSE
$\sin(\text{radians})$	This function returns sine of an angle. The angle should be in radian.
$\cos(\text{radians})$	This function returns cosine of an angle. The angle should be in radian.
$\tan(\text{radians})$	This function returns tan of an angle. The angle should be in radian.
$\text{asin}(\text{radians})$	This function returns inverse of sine angle. The angle should be in radian.



<code>acos(radians)</code>	This function returns inverse of cosine angle. The angle should be in radian.
<code>atan(radians)</code>	This function returns inverse of tan angle. The angle should be in radian.
<code>toRadians(degree)</code>	This function converts the angle in degrees to angle in radian
<code>toDegree(radian)</code>	This function converts the angle in radian to angle in degrees



# EXPONENT METHOD

METHOD	DESCRIPTION	EXAMPLE
<code>exp(x)</code>	It returns value of $e^x$	<code>Math.exp(1)</code> returns 2.71828
<code>log(x)</code>	It returns natural logarithm of x i.e $\log(x)$	<code>Math.log(Math.E)</code> returns 1.0
<code>log10(x)</code>	It returns logarithm of x i.e to the base 10	<code>Math.log10(10)</code> returns 1.0
<code>pow(x,y)</code>	It returns $x^y$	<code>Math.pow(2,3)</code> returns 8
<code>sqrt(x)</code>	It returns the square root of number x	<code>Math.sqrt(25)</code> returns 5



# SERVICE METHOD

<b>METHOD</b>	<b>DESCRIPTION</b>	<b>EXAMPLE</b>
ceil(x)	Returns the smallest integer that is greater than or equal to the argument	ceil(84.6)=85.0 or ceil(0.45)=1.0
floor(x)	Returns the smallest integer that is less than or equal to the argument	floor(84.6)=84.0 or floor(0.45)=0.0
round(x)	Returns the closed int or long	round(84.6)=85
Max(x,y)	Returns the maximum two numbers x and y	max(2,3)=returns 3
min(x,y)	Returns the minimum two numbers x and y	min(2,3)=returns 2
abs(x)	Returns the absolute value	abs(-2) gives 2

---

# LOOPS

## for loop

The Java for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

## while loop

The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

## do-while loop

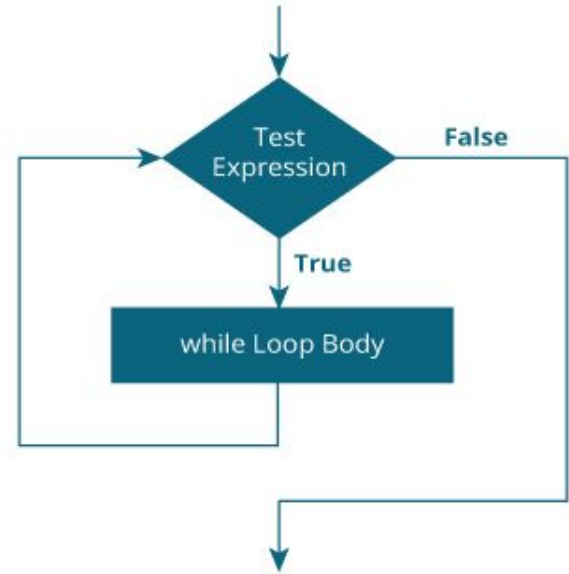
The Java do-while loop is used to iterate a part of the program several times. Use it if the number of iteration is not fixed and you must have to execute the loop at least once.

# WHILE LOOP

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

## Syntax:

```
while(condition){  
//code to be executed  
}
```





## HOW WHILE LOOP WORKS?

- ❖ In the above syntax, the test expression inside parenthesis is a boolean expression. If the test expression is evaluated to true,
  - statements inside the while loop are executed.
  - then, the test expression is evaluated again. This process goes on until the test expression is evaluated to false. If the test expression is evaluated to false,
  - the while loop is terminated.

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

### Output:

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

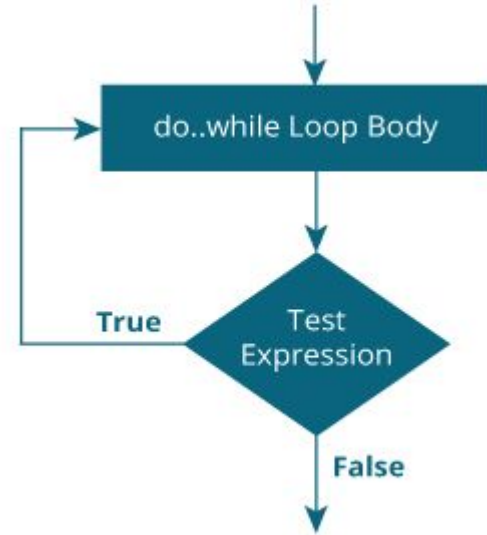
# DO-WHILE LOOP

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java *do-while loop* is executed at least once because condition is checked after loop body.

## Syntax:

```
do{  
    //code to be executed  
}while(condition);
```





## How do...while loop works?

- ❖ The body of do...while loop is executed once (before checking the test expression). Only then, the test expression is checked.
- ❖ If the test expression is evaluated to `true`, codes inside the body of the loop are executed, and the test expression is evaluated again. This process goes on until the test expression is evaluated to `false`.
- ❖ When the test expression is `false`, the do..while loop terminates.



```
import java.util.Scanner;

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

        Double number, sum = 0.0;
        // creates an object of Scanner class
        Scanner input = new Scanner(System.in);

        do {

            // takes input from the user
            System.out.print("Enter a number: ");
            number = input.nextDouble();
            sum += number;
        } while (number != 0.0); // test expression

        System.out.println("Sum = " + sum);
    }
}
```

### Output:

```
Enter a number: 2.5
Enter a number: 23.3
Enter a number: -4.2
Enter a number: 3.4
Enter a number: 0
Sum = 25.0
```



# FOR LOOP

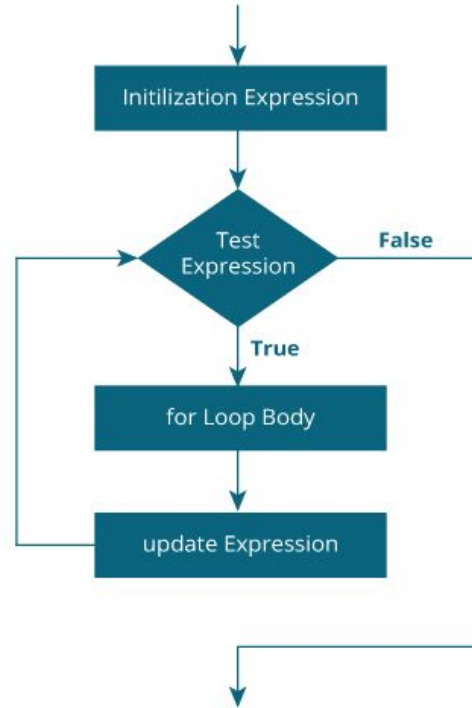
A simple for loop is the same as C/C++. We can initialize the variable, check condition and increment/decrement value. It consists of four parts:

1. **Initialization:** It is the initial condition which is executed once when the loop starts. Here, we can initialize the variable, or we can use an already initialized variable. It is an optional condition.
2. **Condition:** It is the second condition which is executed each time to test the condition of the loop. It continues execution until the condition is false. It must return boolean value either true or false. It is an optional condition.
3. **Statement:** The statement of the loop is executed each time until the second condition is false.
4. **Increment/Decrement:** It increments or decrements the variable value. It is an optional condition.



## Syntax:

```
for(initialization;condition;incr/decr){  
//statement or code to be executed  
}
```



```
// Program to print a sentence 10 times

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

        for (int i = 1; i <= 10; ++i) {
            System.out.println("Line " + i);
        }
    }
}
```

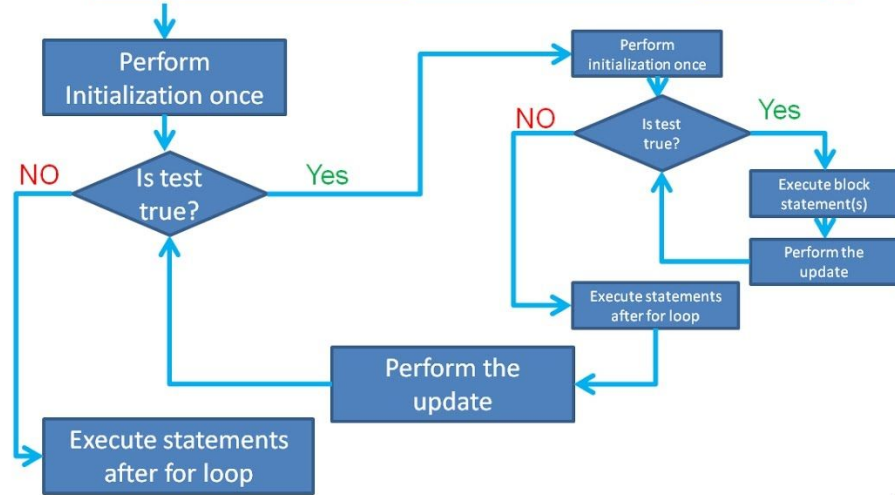
### Output:

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

# NESTED LOOP

If we have a for loop inside the another loop, it is known as nested for loop. The inner loop executes completely whenever outer loop executes.

## Flow Chart Nested for Loop



Malik AB

```
public class NestedForExample {  
public static void main(String[] args) {  
    //loop of i  
    for(int i=1;i<=3;i++){  
        //loop of j  
        for(int j=1;j<=3;j++){  
            System.out.println(i+" "+j);  
        }  
    }  
}
```

Output:

```
1 1  
1 2  
1 3  
2 1  
2 2  
2 3  
3 1  
3 2  
3 3
```

---

# THE BREAK AND CONTINUE



# BREAK STATEMENT

The break statement in Java terminates the loop immediately, and the control of the program moves to the next statement following the loop.

It is almost always used with decision-making statements ([Java if..else Statement](#)).


Here is the syntax of the break statement in Java:

```
break;
```

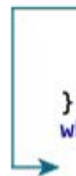


# HOW BREAK STATEMENT WORK?


```
while (testExpression) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```

A flowchart illustrating the execution of a while loop with a break statement. It starts with a box containing 'while (testExpression) {'. An arrow points down to a box containing '// codes'. Another arrow points down to a box containing 'if (condition to break) {'. From there, an arrow points down to a box containing 'break;'. A vertical line then extends down to a box containing '}', and an arrow points from this box back to the 'while (testExpression) {' box, indicating the loop's continuation.

```
do {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
} while (testExpression);
```

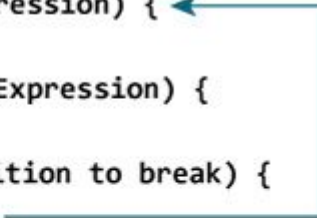
A flowchart illustrating the execution of a do-while loop with a break statement. It starts with a box containing 'do {'. An arrow points down to a box containing '// codes'. Another arrow points down to a box containing 'if (condition to break) {'. From there, an arrow points down to a box containing 'break;'. A vertical line then extends down to a box containing '}', and an arrow points from this box to a box containing 'while (testExpression);'. From this box, an arrow points back to the 'do {' box, indicating the loop's continuation.

```
for (init; testExpression; update) {  
    // codes  
    if (condition to break) {  
        break;  
    }  
    // codes  
}
```

A flowchart illustrating the execution of a for loop with a break statement. It starts with a box containing 'for (init; testExpression; update) {'. An arrow points down to a box containing '// codes'. Another arrow points down to a box containing 'if (condition to break) {'. From there, an arrow points down to a box containing 'break;'. A vertical line then extends down to a box containing '}', and an arrow points from this box back to the 'for (init; testExpression; update) {' box, indicating the loop's continuation.

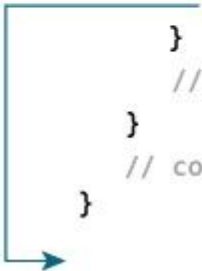
# NESTED BREAK & LABELED BREAK

```
while (testExpression) {  
    // codes  
    while (testExpression) {  
        // codes  
        if (condition to break) {  
            break;  
        }  
        // codes  
    }  
    // codes  
}
```



A blue line starts from the `break;` statement in the inner `while` loop, goes right, then up, then left, and finally down with an arrow pointing to the opening curly brace of the outer `while` loop, illustrating that the break statement exits the entire loop structure.

```
label:  
for (int; testExpresison, update) {  
    // codes  
    for (int; testExpression; update) {  
        // codes  
        if (condition to break) {  
            break label;  
        }  
        // codes  
    }  
    // codes  
}
```



A blue line starts from the `break label;` statement in the inner `for` loop, goes right, then down, then left, and finally down with an arrow pointing to the opening curly brace of the outer `for` loop, illustrating that the labeled break statement exits only the inner loop.

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

        // for loop
        for (int i = 1; i <= 10; ++i) {

            // if the value of i is 5 the loop terminates
            if (i == 5) {
                break;
            }
            System.out.println(i);
        }
    }
}
```

**Output:**

```
1
2
3
4
```



# CONTINUE STATEMENT

The continue statement in Java skips the current iteration of a loop (for, while, do...while, etc) and the control of the program moves to the end of the loop. And, the test expression of a loop is evaluated.

In the case of for loop, the update statement is executed before the test expression.

The continue statement is almost always used in decision-making statements (if...else Statement). It's syntax is:

```
continue;
```



# HOW CONTINUE STATEMENT WORK?

```
→ while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} → while (testExpression);
```

```
→ for (init; testExpression; update) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

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

        // for loop
        for (int i = 1; i <= 10; ++i) {

            // if value of i is between 4 and 9, continue is executed
            if (i > 4 && i < 9) {
                continue;
            }
            System.out.println(i);
        }
    }
}
```


### Output:

```
1
2
3
4
9
10
```

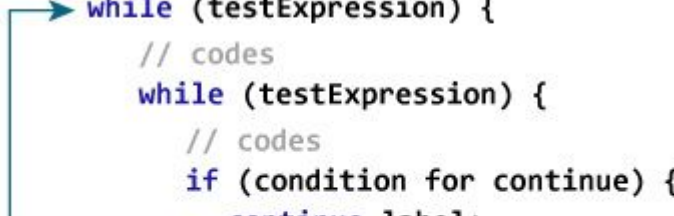


# NESTED CONTINUE & LABELED CONTINUE

```
while(testExpresison) {  
    // codes  
    while (testExpression) {  
        // codes  
        if (condition for continue) {  
            continue;  
        }  
        // codes  
    }  
    // codes  
}
```



```
label:  
while (testExpression) {  
    // codes  
    while (testExpression) {  
        // codes  
        if (condition for continue) {  
            continue label;  
        }  
        // codes  
    }  
    // codes  
}
```



Thank you!

