Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and --

```
public class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
}}
```

Output:

```
10
12
12
10
```

Java Unary Operator Example 2: ++ and --

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=10;
System.out.println(a++ + ++a);//10+12=22
System.out.println(b++ + b++);//10+11=21

}}
```

Output:

```
22
21
```

Java Unary Operator Example: Complement (~) and ! (NOT)

```java
public class OperatorExample{
public static void main(String args[]){
int a=5;
int b=-5;
boolean c=true;
boolean d=false;
System.out.println(~a);//-11 (minus of total positive value which starts from 0)
System.out.println(~b);//4 (positive of total minus, positive starts from 0)
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true
}}
```

Output:

```
-6
4
false
true
```

Illustration:
a = 5 [0101 in Binary]
result = ~5
This performs a bitwise complement of 5
~0101 = 1010 = 10 (in decimal)
Then the compiler will give 2’s complement of that number.
2’s complement of 10 will be -6.
result = -6

Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Arithmetic Operator Example

```java
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

```
15
5
50
2
0
```

Java Arithmetic Operator Example: Expression

```java
public class OperatorExample{
public static void main(String args[]){
System.out.println(10*10/5+3-1*4/2);
}}
```

Output:

```
21
```

Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Output:

```
40
80
80
240
```

Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

```
public OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Output:

```
2
5
2
```

Java Shift Operator Example: >> vs >>>

```
public class OperatorExample{
public static void main(String args[]){
    //For positive number, >> and >>> works same
    System.out.println(20>>2);
    System.out.println(20>>>2);
    //For negative number, >>> changes parity bit (MSB) to 0
    System.out.println(-20>>2);
    System.out.println(-20>>>2);
}}
```

Output:

```
5
5
-5
1073741819
```

Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

Output:

```
false
false
```

Java AND Operator Example: Logical && vs Bitwise &

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a++<c);//false && true = false
System.out.println(a);//10 because second condition is not checked
System.out.println(a<b&a++<c);//false && true = false
System.out.println(a);//11 because second condition is checked
}}
```

Output:

```
false
10
false
11
```

Java OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
//|| vs |
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

Output:

```
true
true
true
10
true
```

```
11
```

Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Java Ternary Operator Example

```java
public class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

```
2
```

Another Example:

```java
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output:

```
5
```

Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Java Assignment Operator Example

```java
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:

```
14
16
```

Java Assignment Operator Example

```java
public class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

Output:

```
13
9
18
9
```

Java Assignment Operator Example: Adding short

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

```
short a=10;
short b=10;
//a+=b;//a=a+b internally so fine
a=a+b;//Compile time error because 10+10=20 now int
System.out.println(a);
}}
```

Output:

```
Compile time error
```

After type cast:

```
public class OperatorExample{
public static void main(String args[]){
short a=10;
short b=10;
a=(short)(a+b);//20 which is int now converted to short
System.out.println(a);
}}
```

Output:

```
20
```

Operator Shifting

Bitwise Left Shift Operator (<<)

Example

If **x=10**, then calculate **x<<2** value.

Shifting the value of x towards the left two positions will make the leftmost 2 bits to be lost. The value of x is 10. The binary representation of 10 is **00001010**. The procedure to do left shift explained in the following example:

Observe the above example, after shifting the bits to the left the binary number **00001010** (in decimal 10) becomes **00101000** (in decimal 40).

Bitwise Right Shift Operator

Example

If **x=10**, then calculate **x>>2** value.

Shifting the value of x towards the right two positions will make the rightmost 2 bits to be lost. The value of x is 10. The binary representation of **10** is **00001010**. The procedure to do right shift explained in the following example:

Observe the above example, after shifting the bits to the right the binary number **00001010** (in decimal 10) becomes **00000010** (in decimal 2).

Bitwise Zero Fill Right Shift Operator (>>>)

Bitwise Zero Fill Right Shift Operator shifts the bits of the number towards the **right** a specified n number of positions. The sign bit filled with 0's. The symbol >>> represents the Bitwise Zero Fill Right Shift Operator.

When we apply **>>>** on **a positive number**, it gives the same output as that of >>. It gives a positive number when we apply >>> on a negative number. MSB is replaced by a 0.

Observe the above example, after shifting the bits to the right the binary number **00100000** (in decimal 32) becomes **00000100** (in decimal 4). The last three bits shifted out and lost.

Difference between >> and >>> operator

Both >> and >>> are used to shift the bits towards the right. The difference is that the >> preserve the sign bit while the operator >>> does not preserve the sign bit. To preserve the sign bit, you need to add 0 in the MSB.

Example

Let's see the left and right shifting through example:

```
public class OperatorShifting {

    public static void main ( String[] args ) {
        byte l, r,b;
        l=10; //00001010
        r=10; //00001010
        System.out.println("Bitwise Left Shift: l<<2 = "+(l<<2));
//00101000
        System.out.println("Bitwise Right Shift: r>>2 = "+(r>>2));
//00000010
        b=32; //00100000
        System.out.println("Bitwise Zero Fill Right Shift: b>>>2 =
"+(b>>>2)); //00001000
       System.out.println("Bitwise Zero Fill Right Shift: b>>>3 =
       "+(b>>>3)); //00000100
       System.out.println("Bitwise Zero Fill Right Shift: b>>>4 =
       "+(b>>>4)); //00000010

    }
}
```

Output:

```
Bitwise Left Shift: l<<2 = 40
Bitwise Right Shift: r>>2 = 2
Bitwise Zero Fill Right Shift: b>>>2 = 8
Bitwise Zero Fill Right Shift: b>>>3 = 4
Bitwise Zero Fill Right Shift: b>>>4 = 2
```

Example 1

```
public class Demo{
  public static void main(String[] args){
    System.out.println("Is the argument an instance of super class Object? ");
    System.out.println(args instanceof Object);
    int[] my_arr = new int[4];
    System.out.println("Is the array my_arr an instance of super class Object? ");
    System.out.println(my_arr instanceof Object);
  }
}
```

Output

```
Is the argument an instance of super class Object?
true
Is the array my_arr an instance of super class Object?
true
```

Arrays of Primitive Data Types

An array is a collection of similar data types. Array is a container object that hold values of homogenous type. It is also known as static data structure because size of an array must be specified at the time of its declaration.

An array can be either primitive or reference type. It gets memory in heap area. Index of array starts from zero to size-1.

Array Declaration

Syntax :

```
datatype[] identifier;
or
datatype identifier[];
```

Both are valid syntax for array declaration. But the former is more readable.

Example :

```
int[] arr;
char[] arr;
short[] arr;
long[] arr;
int[][] arr; //two dimensional array.
```

Initialization of Array

new operator is used to initialize an array.

Example :

```
int[] arr = new int[10];    //10 is the size of array.
or
int[] arr = {10,20,30,40,50};
```

Accessing array element

As mention ealier array index starts from 0. To access nth element of an array. Syntax

```
arrayname[n-1];
```

Example : To access 4th element of a given array

```
int[] arr={10,20,30,40};
System.out.println("Element at 4th place"+arr[3]);
```

The above code will print the 4th element of array arr on console.

foreach or enhanced for loop

J2SE 5 introduces special type of for loop called foreach loop to access elements of array. Using foreach loop you can access complete array sequentially without using index of array. Let us see an exapmle of foreach loop.

```
class Arraydemo
{
public static void main(String[] args)
  {
    int[] arr={10,20,30,40};
       for(int x:arr)
       {
       System.out.println(x);
       }
   }
}
```

```
Output:
10
20
30
40
```

Java Control Statements

Java provides three types of control flow statements.

1. Decision Making statements
   - if statements
   - switch statement
2. Loop statements
   - do while loop
   - while loop
   - for loop
   - for-each loop
3. Jump statements
   - break statement
   - continue statement

Control Statements

If, If..else Statement in Java with Examples

When we need to execute a set of statements based on a condition then we need to use **control flow statements**. For example, if a number is greater than zero then we want to print "Positive Number" but if it is less than zero then we want to print "Negative Number". In this case we have two print statements in the program, but only one print statement executes at a time based on the input value. We will see how to write such type of conditions in the java program using control statements.
In this tutorial, we will see four types of control statements that you can use in java programs based on the requirement: In this tutorial we will cover following conditional statements:

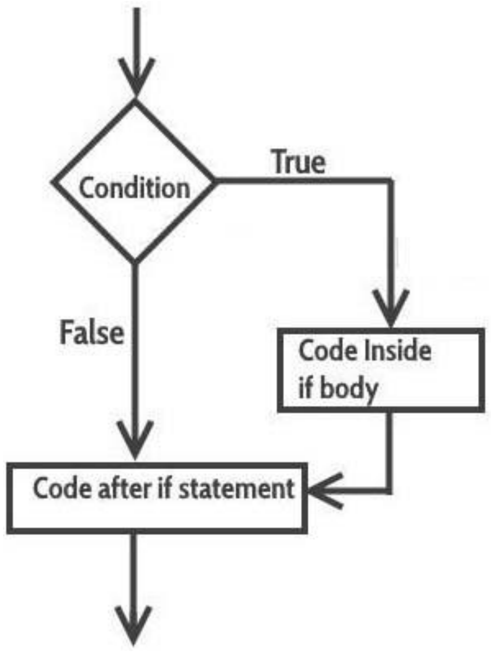
a) if statement
b) nested if statement
c) if-else statement
d) if-else-if statement

If statement

If statement consists a condition, followed by statement or a set of statements as shown below:

```
if(condition){
  Statement(s);
}
```

The statements gets executed only when the given condition is true. If the condition is false then the statements inside if statement body are completely ignored.



Example of if statement

```
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
```

Nested if statement in Java

When there is an if statement inside another if statement then it is called the **nested if statement**.
The structure of nested if looks like this:

```
if(condition_1) {
   Statement1(s);

   if(condition_2) {
      Statement2(s);
   }
}
```

Statement1 would execute if the condition_1 is true. Statement2 would only execute if both the conditions(condition_1 and condition_2) are true.

Example of Nested if statement

```
public class NestedIfExample {

   public static void main(String args[]){
        int num=70;
             if( num < 100 ){
           System.out.println("number is less than 100");
           if(num > 50){
                  System.out.println("number is greater than 50");
                }
             }
   }
}
```

Output:

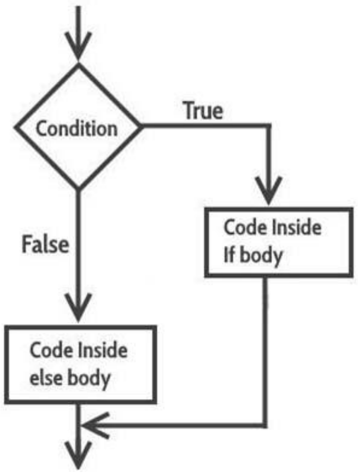
```
number is less than 100
number is greater than 50
```

If else statement in Java

This is how an if-else statement looks:

```
if(condition) {
   Statement(s);
}
else {
   Statement(s);
}
```

The statements inside “if” would execute if the condition is true, and the statements inside “else” would execute if the condition is false.



Example of if-else statement

```
public class IfElseExample {

  public static void main(String args[]){
    int num=120;
    if( num < 50 ){
            System.out.println("num is less than 50");
    }
    else {
            System.out.println("num is greater than or equal 50");
    }
  }
}
```

Output:

```
num is greater than or equal 50
```

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**. This is how it looks:

```
if(condition_1) {
   /*if condition_1 is true execute this*/
   statement(s);
}
else if(condition_2) {
   /* execute this if condition_1 is not met and
    * condition_2 is met
    */
   statement(s);
}
else if(condition_3) {
   /* execute this if condition_1 & condition_2 are
    * not met and condition_3 is met
    */
   statement(s);
}
.
.
.
else {
   /* if none of the condition is true
    * then these statements gets executed
    */
   statement(s);
}
```

Note: The most important point to note here is that in if-else-if statement, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored. If none of the condition is met then the statements inside "else" gets executed.

Example of if-else-if

```
public class IfElseIfExample {

  public static void main(String args[]){
          int num=1234;
          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:

```
Its a four digit number
```

Check out these related java examples:

1. Java Program to find the largest of three numbers using if..else..if
2. Java Program to check if number is positive or negative
3. Java Program to check if number is even or odd

Switch Case statement in Java with example

Switch case statement is used when we have number of options (or choices) and we may need to perform a different task for each choice.

The syntax of Switch case statement looks like this –

```
switch (variable or an integer expression)
{
     case constant:
     //Java code
     ;
     case constant:
     //Java code
     ;
     default:
     //Java code
     ;
}
```

Switch Case statement is mostly used with break statement even though it is optional. We will first see an example without break statement and then we will discuss switch case with break

A Simple Switch Case Example

```
public class SwitchCaseExample1 {

   public static void main(String args[]){
     int num=2;
     switch(num+2)
     {
        case 1:
               System.out.println("Case1: Value is: "+num);
             case 2:
               System.out.println("Case2: Value is: "+num);
             case 3:
               System.out.println("Case3: Value is: "+num);
        default:
               System.out.println("Default: Value is: "+num);
      }
   }
}
```

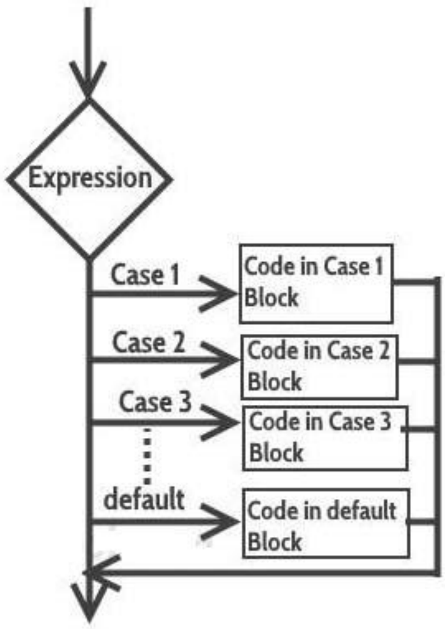
Output:

```
Default: Value is: 2
```

Explanation: In switch I gave an expression, you can give variable also. I gave num+2, where num value is 2 and after addition the expression resulted 4. Since there is no case defined with value 4 the default case got executed. This is why we should use default in switch case, so that if there is no catch that matches the condition, the default block gets executed.

Switch Case Flow Diagram

First the variable, value or expression which is provided in the switch parenthesis is evaluated and then based on the result, the corresponding case block is executed that matches the result.



Break statement in Switch Case

Break statement is optional in switch case but you would use it almost every time you deal with switch case. Before we discuss about break statement, Let’s have a look at the example below where I am not using the break statement:

```
public class SwitchCaseExample2 {

  public static void main(String args[]){
    int i=2;
    switch(i)
    {
           case 1:
           System.out.println("Case1 ");
           case 2:
           System.out.println("Case2 ");
           case 3:
           System.out.println("Case3 ");
           case 4:
       System.out.println("Case4 ");
           default:
           System.out.println("Default ");
    }
  }
}
```

Output:

```
Case2
Case3
Case4
Default
```

In the above program, we have passed integer value 2 to the switch, so the control switched to the case 2, however we don’t have break statement after the case 2 that caused the flow to pass to the subsequent cases till the end. The solution to this problem is break statement

Break statements are used when you want your program-flow to come out of the switch body. Whenever a break statement is encountered in the switch body, the execution flow would directly come out of the switch, ignoring rest of the cases

Let’s take the same example but this time with break statement.

Example with break statement

```
public class SwitchCaseExample2 {

  public static void main(String args[]){
    int i=2;
    switch(i)
    {
           case 1:
             System.out.println("Case1 ");
             break;
           case 2:
             System.out.println("Case2 ");
             break;
           case 3:
             System.out.println("Case3 ");
             break;
           case 4:
       System.out.println("Case4 ");
       break;
           default:
             System.out.println("Default ");
    }
  }
}
```

Output:

```
Case2
```

Now you can see that only case 2 had been executed, rest of the cases were ignored.

Why didn’t I use break statement after default?

The control would itself come out of the switch after default so I didn’t use it, however if you still want to use the break after default then you can use it, there is no harm in doing that.

Few points about Switch Case

1) Case doesn’t always need to have order 1, 2, 3 and so on. It can have any integer value after case keyword. Also, case doesn’t need to be in an ascending order always, you can specify them in any order based on the requirement.

2) You can also use characters in switch case. for example –

```
public class SwitchCaseExample2 {

   public static void main(String args[]){
      char ch='b';
      switch(ch)
      {
            case 'd':
              System.out.println("Case1 ");
              break;
            case 'b':
              System.out.println("Case2 ");
              break;
            case 'x':
              System.out.println("Case3 ");
              break;
            case 'y':
         System.out.println("Case4 ");
         break;
            default:
              System.out.println("Default ");
      }
   }
}
```

3) The expression given inside switch should result in a constant value otherwise it would not be valid. For example:

Valid expressions for switch:

```
switch(1+2+23)
switch(1*2+3%4)
```

Invalid switch expressions:

```
switch(ab+cd)
switch(a+b+c)
```

4) Nesting of switch statements are allowed, which means you can have switch statements inside another switch. However nested switch statements should be avoided as it makes program more complex and less readable.

Check out these related java programs:

1. Java Program to check whether a char is vowel or Consonant using Switch Case
2. Java Program to make a Simple Calculator using Switch Case

For loop in Java with example

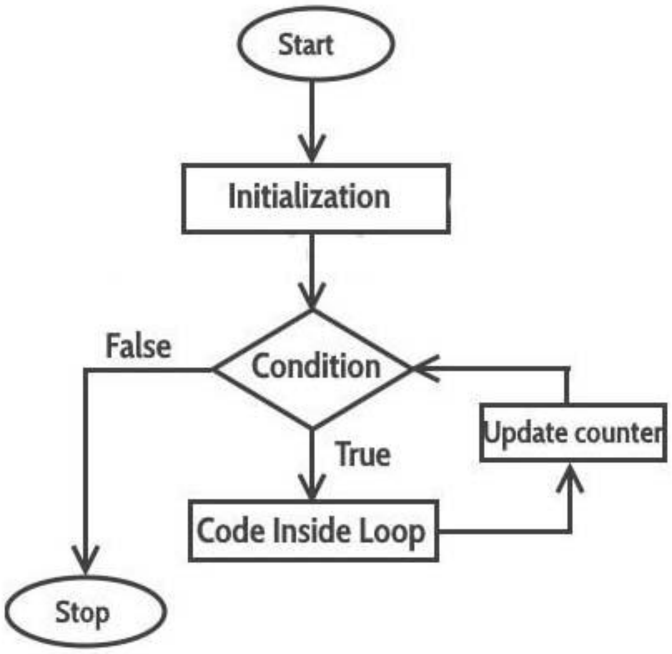
Loops are used to execute a set of statements repeatedly until a particular condition is satisfied. In Java we have three types of basic loops: for, while and do-while. In this tutorial we will learn how to use "**for loop**" in Java.

Syntax of for loop:

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

Flow of Execution of the for Loop

As a program executes, the interpreter always keeps track of which statement is about to be executed. We call this the control flow, or the flow of execution of the program.



First step: In for loop, initialization happens first and only one time, which means that the initialization part of for loop only executes once.
Second step: Condition in for loop is evaluated on each iteration, if the condition is true then the statements inside for loop body gets executed. Once the condition returns false, the statements in for loop does not execute and the control gets transferred to the next statement in the program after for loop.
Third step: After every execution of for loop’s body, the increment/decrement part of for loop executes that updates the **loop counter**.
Fourth step: After third step, the control jumps to second step and condition is re-evaluated.

Example of Simple For loop

```
class ForLoopExample {
    public static void main(String args[]){
        for(int i=10; i>1; i--){
            System.out.println("The value of i is: "+i);
        }
    }
}
```

The output of this program is:

```
The value of i is: 10
The value of i is: 9
The value of i is: 8
The value of i is: 7
The value of i is: 6
The value of i is: 5
The value of i is: 4
The value of i is: 3
The value of i is: 2
```

In the above program:
int i=1 is initialization expression
i>1 is condition(Boolean expression)
i– Decrement operation

Infinite for loop

The importance of Boolean expression and increment/decrement operation co-ordination:

```
class ForLoopExample2 {
    public static void main(String args[]){
         for(int i=1; i>=1; i++){
              System.out.println("The value of i is: "+i);
         }
    }
}
```

This is an infinite loop as the condition would never return false. The initialization step is setting up the value of variable i to 1, since we are incrementing the value of i, it would always be greater than 1 (the Boolean expression: i>1) so it would never return false. This would eventually lead to the infinite loop condition. Thus it is important to see the co-ordination between Boolean expression and increment/decrement operation to determine whether the loop would terminate at some point of time or not.
Here is another example of infinite for loop:

```
// infinite loop
for ( ; ; ) {
    // statement(s)
}
```

For loop example to iterate an array:

Here we are iterating and displaying array elements using the for loop.

```
class ForLoopExample3 {
    public static void main(String args[]){
         int arr[]={2,11,45,9};
         //i starts with 0 as array index starts with 0 too
         for(int i=0; i<arr.length; i++){
              System.out.println(arr[i]);
         }
    }
}
```

Output:

```
2
11
45
9
```

Enhanced For loop

Enhanced for loop is useful when you want to iterate Array/Collections, it is easy to write and understand.
Let’s take the same example that we have written above and rewrite it using **enhanced for loop**.

```
class ForLoopExample3 {
   public static void main(String args[]){
      int arr[]={2,11,45,9};
      for (int num : arr) {
         System.out.println(num);
      }
   }
}
```

Output:

```
2
11
45
9
```

Note: In the above example, I have declared the num as int in the enhanced for loop. This will change depending on the data type of array. For example, the enhanced for loop for string type would look like this:

```
String arr[]={"hi","hello","bye"};

for (String str : arr) {
        System.out.println(str);
}
```

Check out these java programming examples related to for loop:

1. Java Program to find sum of natural numbers using for loop
2. Java Program to find factorial of a number using loops
3. Java Program to print Fibonacci Series using for loop

While loop in Java with examples

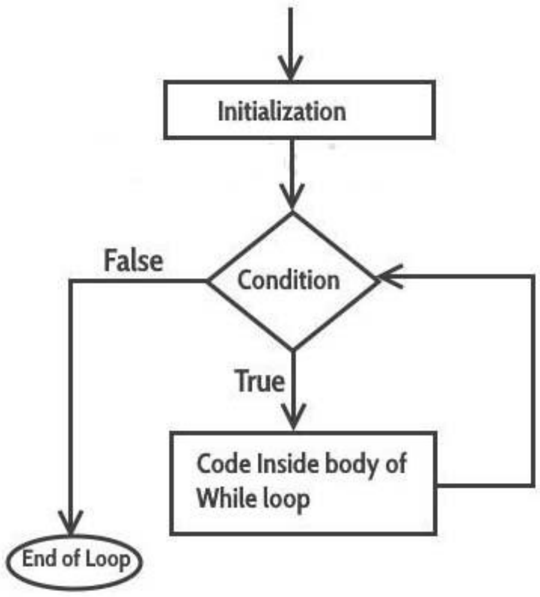
In the last tutorial, we discussed for loop. In this tutorial we will discuss while loop. As discussed in previous tutorial, loops are used to execute a set of statements repeatedly until a particular condition is satisfied.

Syntax of while loop

```
while(condition)
{
   statement(s);
}
```

How while Loop works?

In while loop, condition is evaluated first and if it returns true then the statements inside while loop execute. When condition returns false, the control comes out of loop and jumps to the next statement after while loop. Note: The important point to note when using while loop is that we need to use increment or decrement statement inside while loop so that the loop variable gets changed on each iteration, and at some point condition returns false. This way we can end the execution of while loop otherwise the loop would execute indefinitely.



Simple while loop example

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

Output:

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

Infinite while loop

```
class WhileLoopExample2 {
    public static void main(String args[]){
         int i=10;
         while(i>1)
         {
             System.out.println(i);
              i++;
         }
    }
}
```

This loop would never end, its an infinite while loop. This is because condition is i>1 which would always be true as we are incrementing the value of i inside while loop.
Here is another example of infinite while loop:

```
while (true){
    statement(s);
}
```

Example: Iterating an array using while loop

Here we are iterating and displaying array elements using while loop.

```
class WhileLoopExample3 {
    public static void main(String args[]){
         int arr[]={2,11,45,9};
         //i starts with 0 as array index starts with 0 too
         int i=0;
         while(i<4){
              System.out.println(arr[i]);
              i++;
         }
    }
}
```

Output:

```
2
11
45
9
```

Check out these related programs:

1. Java Program to display Fibonacci Series using while loop
2. Java Program to find factorial using while loop

do-while loop in Java with example

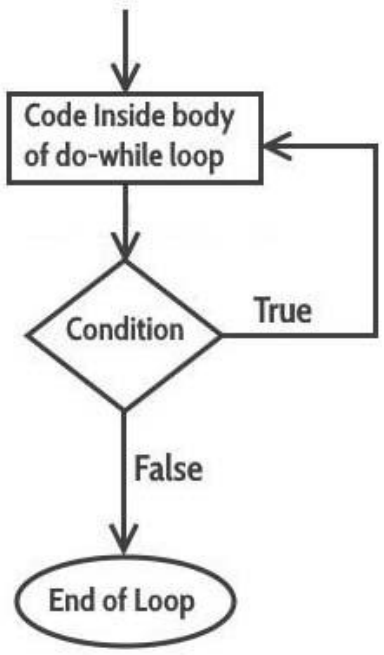
In the last tutorial, we discussed while loop. In this tutorial we will discuss do-while loop in java. do-while loop is similar to while loop, however there is a difference between them: In while loop, condition is evaluated before the execution of loop's body but in do-while loop condition is evaluated after the execution of loop's body.

Syntax of do-while loop:

```
do
{
  statement(s);
} while(condition);
```

How do-while loop works?

First, the statements inside loop execute and then the condition gets evaluated, if the condition returns true then the control gets transferred to the "do" else it jumps to the next statement after do-while.



do-while loop example

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

Output:

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

Example: Iterating array using do-while loop

Here we have an integer array and we are iterating the array and displaying each element using do-while loop.

```
class DoWhileLoopExample2 {
    public static void main(String args[]){
         int arr[]={2,11,45,9};
         //i starts with 0 as array index starts with 0
         int i=0;
         do{
             System.out.println(arr[i]);
             i++;
         }while(i<4);
    }
}
```

Output:

```
2
11
45
9
```

Continue Statement in Java with example

Continue statement is mostly used inside loops. Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop's body for the current iteration. This is particularly useful when you want to continue the loop but do not want the rest of the statements(after continue statement) in loop body to execute for that particular iteration.

Syntax:

continue word followed by semi colon.

```
continue;
```

Example: continue statement inside for loop

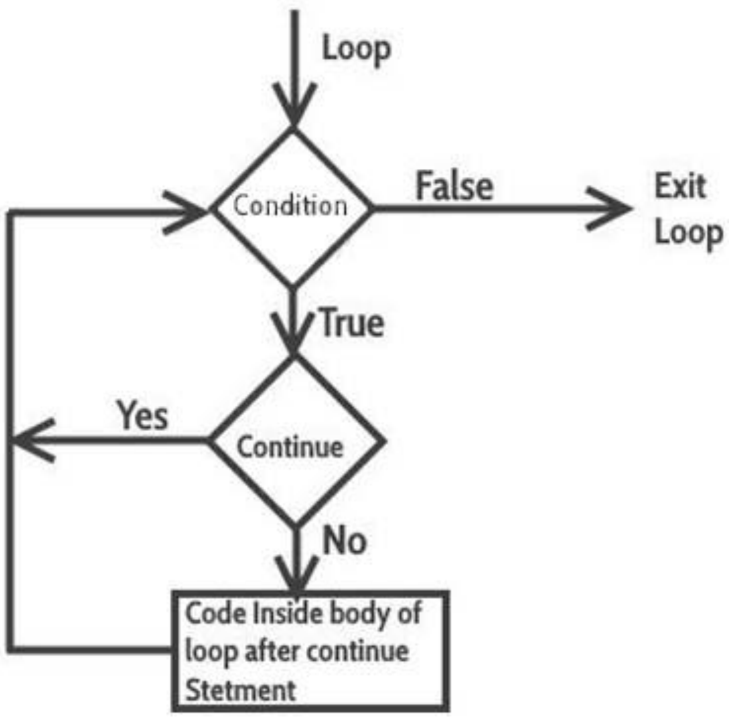
```
public class ContinueExample {

   public static void main(String args[]){
            for (int j=0; j<=6; j++)
            {
        if (j==4)
        {
                continue;
              }

        System.out.print(j+" ");
            }
   }
}
```

Output:

```
0 1 2 3 5 6
```

As you may have noticed, the value 4 is missing in the output, why? because when the value of variable j is 4, the program encountered a **continue statement**, which makes it to jump at the beginning of for loop for next iteration, skipping the statements for current iteration (that's the reason println didn't execute when the value of j was 4).

Flow Diagram of Continue Statement



Example: Use of continue in While loop

Same thing you can see here. We are iterating this loop from 10 to 0 for counter value and when the counter value is 7 the loop skipped the print statement and started next iteration of the while loop.

```
public class ContinueExample2 {

   public static void main(String args[]){
            int counter=10;
            while (counter >=0)
            {
        if (counter==7)
        {
                counter--;
                continue;
        }
        System.out.print(counter+" ");
        counter--;
            }
   }
}
```

Output:

```
10 9 8 6 5 4 3 2 1 0
```

Example of continue in do-While loop

```
public class ContinueExample3 {

  public static void main(String args[]){
          int j=0;
      do
          {
            if (j==7)
            {
                     j++;
                     continue;
            }
            System.out.print(j+ " ");
            j++;
     }while(j<10);

  }
}
```

Output:

```
0 1 2 3 4 5 6 8 9
```

Break statement in Java with example

The **break statement** is usually used in following two scenarios:
a) Use break statement to come out of the loop instantly. Whenever a break statement is encountered inside a loop, the control directly comes out of loop and the loop gets terminated for rest of the iterations. It is used along with if statement, whenever used inside loop so that the loop gets terminated for a particular condition. The important point to note here is that when a break statement is used inside a nested loop, then only the inner loop gets terminated.
b) It is also used in switch case control. Generally all cases in switch case are followed by a break statement so that whenever the program control jumps to a case, it doesn’t execute subsequent cases (see the example below). As soon as a break is encountered in switch-case block, the control comes out of the switch-case body.
Syntax of break statement:
“break” word followed by semi colon

```
break;
```

Example – Use of break in a while loop

In the example below, we have a while loop running from o to 100 but since we have a break statement that only occurs when the loop value reaches 2, the loop gets terminated and the control gets passed to the next statement in program after the loop body.

```
public class BreakExample1 {
  public static void main(String args[]){
    int num =0;
    while(num<=100)
    {
       System.out.println("Value of variable is: "+num);
       if (num==2)
       {
         break;
       }
       num++;
    }
    System.out.println("Out of while-loop");
  }
}
```

Output:

```
Value of variable is: 0
Value of variable is: 1
Value of variable is: 2
Out of while-loop
```

Example – Use of break in a for loop

The same thing you can see here. As soon as the var value hits 99, the for loop gets terminated.

```
public class BreakExample2 {

  public static void main(String args[]){
          int var;
          for (var =100; var>=10; var --)
          {
             System.out.println("var: "+var);
             if (var==99)
             {
                 break;
             }
           }
           System.out.println("Out of for-loop");
  }
}
```

Output:

```
var: 100
var: 99
Out of for-loop
```

Example – Use of break statement in switch-case

```
public class BreakExample3 {

  public static void main(String args[]){
           int num=2;

           switch (num)
           {
              case 1:
                System.out.println("Case 1 ");
                break;
              case 2:
                System.out.println("Case 2 ");
                break;
              case 3:
                System.out.println("Case 3 ");
                break;
              default:
                System.out.println("Default ");
           }
  }
}
```

Output:

```
Case 2
```

In this example, we have break statement after each Case block, this is because if we don't have it then the subsequent case block would also execute. The output of the same program without break would be Case 2 Case 3 Default.