Java Unary Operator with Examples

Difficulty Level : Basic Last Updated : 27 Dec, 2021

Operators constitute the basic building block to any programming language. Java too provides many types of operators which can be used according to the need to perform various calculations and functions be it logical, arithmetic, relational, etc. They are classified based on the functionality they provide. Here are a few types:

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators

Unary Operators in Java

Java unary operators are the types that need only one operand to perform any operation like increment, decrement, negation, etc. It consists of various arithmetic, logical and other operators that operate on a single operand. Let’s look at the various unary operators in detail and see how they operate.

Operator 1: Unary minus(-)

This operator can be used to convert a negative value to a positive one.

Syntax:

```
~(operand)
```

Illustration:

```
a = -10
```

Example:

```java
// Java Program to Illustrate Unary - Operator

// Importing required classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Declaring a custom variable
        int n1 = 20;

        // Printing the above variable
        System.out.println("Number = " + n1);

        // Performing unary operation
        n1 = -n1;

        // Printing the above result number
        // after unary operation
        System.out.println("Result = " + n1);
    }
}
```

Output

```
Number = 20
Result = -20
```

Operator 2: 'NOT' Operator(!)

This is used to convert true to false or vice versa. Basically, it reverses the logical state of an operand.

Syntax:

```
!(operand)
```

Illustration:

```
cond = !true;
// cond < false
```

Example:

```
// Java Program to Illustrate Unary NOT Operator

// Importing required classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Initializing variables
        boolean cond = true;
        int a = 10, b = 1;

        // Displaying values stored in above variables
        System.out.println("Cond is: " + cond);
        System.out.println("Var1 = " + a);
        System.out.println("Var2 = " + b);

        // Displaying values stored in above variables
        // after applying unary NOT operator
        System.out.println("Now cond is: " + !cond);
        System.out.println("!(a < b) = " + !(a < b));
        System.out.println("!(a > b) = " + !(a > b));
    }
}
```

Output:

```
Cond is: true
Var1 = 10
Var2 = 1
Now cond is: false
```

```
!(a < b) = true
!(a > b) = false
```

Operator 3: Increment(++)

It is used to increment the value of an integer. It can be used in two separate ways:

3.1: Post-increment operator

When placed after the variable name, the value of the operand is incremented but the previous value is retained temporarily until the execution of this statement and it gets updated before the execution of the next statement.

Syntax:

```
num++
```

Illustration:

```
num = 5
num++ = 6
```

3.2: Pre-increment operator

When placed before the variable name, the operand's value is incremented instantly.

Syntax:

```
++num
```

Illustration:

```
num = 5
++num = 6
```

Operator 4: Decrement(–)

It is used to decrement the value of an integer. It can be used in two separate ways:

4.1: Post-decrement operator

When placed after the variable name, the value of the operand is decremented but the previous values is retained temporarily until the execution of this statement and it gets updated before the execution of the next statement.

Syntax:

```
num--
```

Illustration:

```
num = 5
num-- = 4
```

4.2: Pre-decrement operator

When placed before the variable name, the operand's value is decremented instantly.

Syntax:

```
--num
```

Illustration:

```
num = 5
--num = 4
```

Operator 5: Bitwise Complement(~)

This unary operator returns the one's complement representation of the input value or operand, i.e, with all bits inverted, which means it makes every 0 to 1, and every 1 to 0.

Syntax:

```
~(operand)
```

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
```

Example:

```
// Java program to Illustrate Unary
// Bitwise Complement Operator

// Importing required classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Declaring a variable
        int n1 = 6, n2 = -2;

        // Printing numbers on console
        System.out.println("First Number = " + n1);
        System.out.println("Second Number = " + n2);

        // Printing numbers on console after
        // performing bitwise complement
        System.out.println(
            n1 + "'s bitwise complement = " + ~n1);
        System.out.println(
            n2 + "'s bitwise complement = " + ~n2);
    }
}
```

Output:

```
First Number = 6
Second Number = -2
```

```
6's bitwise complement = -7
-2's bitwise complement = 1
```