**Guided Lab 303.2.1 - Core Java: Operators**

# Objective:

In this lab, we will discuss and demonstrate the different types of operators in Java, their syntax, and how to use them with the help of examples.

# **Learning Objectives:**

By the end of this lab, learners will be able to use Operators in Java.

# 

# Introduction

The list given below informs us about the entire range of operators provided in the Java programming language.

1. Unary Operator.
2. Relational Operator.
3. Assignment Operator.
4. Java instanceof Operator.
5. Ternary Operator or Conditional Operators.
6. Arithmetic Operator.
7. Logical Operator.
8. Bitwise Operator.

## 

## **Example: Assignment Operators**

Java Assignment Operator Syntax:

**<variable> = <expression>**

| **Operator** | **Example** | **Equivalent to** |
| --- | --- | --- |
| = | x=y | x=y |
| += | x+=y | x=x+y |
| -= | x-=y | x=x-y |
| \*= | x\*=y | x=x\*y |
| /= | x/=y | x=x/y |
| %= | x%=y | x=x%y |

Java Assignment Operators are used for modifying a value, performing some mathematical calculations, and assigning values to the operands.

If the value already exists in the variable, it is overwritten by the assignment operator (=).

Create a Java class named **AssignmentOperatorsDemo** and write the code below in that class.

| public class AssignmentOperatorsDemo {  public static void main(String args[]) {  *// Assigning Primitive Values*  int j, k;  j = 10; *// j gets the value 10.*  j = 5; *// j gets the value 5. The previous value is overwritten.*  k = j; *// k gets the value 5.*  System.out.println("j is : " + j);  System.out.println("k is : " + k);   *// Multiple Assignments*  k = j = 10; *// (k = (j = 10))*  System.out.println("j is : " + j);  System.out.println("k is : " + k);  } } |
| --- |

**Output**

| j is : 5  k is : 5  j is : 10  k is : 10 |
| --- |

## **Example: Arithmetic Operators**

Eight arithmetic operators are available in Java: addition, subtraction, multiplication, division, modulo (or remainder), increment (or add 1), decrement (or subtract 1), and negation. The program below demonstrates the different arithmetic operators in Java:

| **Operator** | **Name** | **Function** | **Syntax** |
| --- | --- | --- | --- |
| + | Addition | Addition of two values | x+y |
| - | Subtraction | Subtraction of two values | x-y |
| \* | Multiplication | Multiplies two values | x\*y |
| / | Division | Divides two values. If we divide a number by 0, it does not have any definitive meaning in arithmetic operations; hence, we will get *undefined* as a result. | x/y |
| % | Modulus | Returns the divisor or the reminder | x%y |

Create a Java class named **ArithmeticOperatorsDemo**  and write the code below in that class.

| public class ArithmeticOperatorsDemo {   public static void main(String[] args) {  *// TODO Auto-generated method stub*  int x, y = 10, z = 5;  x = y + z;  System.out.println("+ operator resulted in " + x);  x = y - z;  System.out.println("- operator resulted in " + x);  x = y \* z;  System.out.println("\* operator resulted in " + x);  x = y / z;  System.out.println("/ operator resulted in " + x);  x = y % z;  System.out.println("% operator resulted in " + x);  x = y++;  System.out.println("Postfix ++ operator resulted in " + x);  x = ++z;  System.out.println("Prefix ++ operator resulted in " + x);  x = -y;  System.out.println("Unary operator resulted in " + x);  *// Some examples of special Cases*  int tooBig = Integer.MAX\_VALUE + 1; *// -2147483648 which is*  *// Integer.MIN\_VALUE.*  int tooSmall = Integer.MIN\_VALUE - 1; *// 2147483647 which is*  *// Integer.MAX\_VALUE.*  System.out.println("tooBig becomes " + tooBig);  System.out.println("tooSmall becomes " + tooSmall);  System.out.println(4.0 / 0.0); *// Prints: Infinity*  System.out.println(-4.0 / 0.0); *// Prints: -Infinity*  System.out.println(0.0 / 0.0); *// Prints: NaN*  double d1 = 12 / 8; *// result: 1 by integer division. The value of d1 becomes 1.0.*   double d2 = 12.0F / 8; *// result: 1.5*  System.out.println("d1 is " + d1);  System.out.println("d2 is " + d2);  } } |
| --- |

**Output**

| + operator resulted in 15  - operator resulted in 5  \* operator resulted in 50  / operator resulted in 2  % operator resulted in 0  Postfix ++ operator resulted in 10  Prefix ++ operator resulted in 6  Unary operator resulted in -11  tooBig becomes -2147483648  tooSmall becomes 2147483647  Infinity  -Infinity  NaN  d1 is 1.0  d2 is 1.5 |
| --- |

The binary operator “+” is overloaded in the sense that the operation performed is determined by the type of the operands. When one of the operands is a String object, the other operand is implicitly converted to its string representation, and string concatenation is performed.

*String message = 100 + "Messages"; //"100 Messages"*

## 

## 

## 

## Example: Relational Operators

Relational operators in Java are used to compare two or more objects. Java provides six relational operators. All relational operators are binary operators, and their operands are numeric expressions.

| **Symbol** | **Name** | **Function** |
| --- | --- | --- |
| < | Less than | The value returned will be true if the left-hand side (LHS) is less than the right-hand side (RHS). |
| > | Greater than | The value returned will be true if the left-hand side (LHS) is greater than the right-hand side (RHS). |
| == | Equal To | The value returned will be true if the LHS is equal to the RHS. |
| != | Not Equal To | The value returned will be true if the LHS is not equal to the RHS. |
| <= | Less than or Equal To | The value returned will be true if the LHS is less than or equal to the RHS. |
| >= | Greater than or Equal To | The value returned will be true if the LHS is greater than or equal to the RHS. |

Binary numeric promotion is applied to the operands of these operators. The evaluation results in a *boolean* value. Relational operators have a lower precedence than arithmetic operators, but a higher precedence than the assignment operators. The program below demonstrates the different relational operators in Java.

Create a Java class named **RelationalOperatorsDemo**  and write the code below in that class.

| public class RelationalOperatorsDemo {  public static void main(String[] args)  {  int x = 10, y = 5;  System.out.println("x > y : "+(x > y));  System.out.println("x < y : "+(x < y));  System.out.println("x >= y : "+(x >= y));  System.out.println("x <= y : "+(x <= y));  System.out.println("x == y : "+(x == y));  System.out.println("x != y : "+(x != y));   int variable1 = 50, variable2 = 100, variable3 = 50;  System.out.println("variable1 = " + variable1);  System.out.println("variable2 = " + variable2);  System.out.println("variable3 = " + variable3);  System.out.println("variable1 == variable2: "  + (variable1 == variable2));   System.out.println("variable1 == variable3: "  + (variable1 == variable3));  } } |
| --- |

**Output**

| x > y : true  x < y : false  x >= y : true  x <= y : false  x == y : false  x != y : true  variable1 = 50  variable2 = 100  variable3 = 50  variable1 == variable2: false  variable1 == variable3: true |
| --- |

## Example: Logical Operators

Logical operators return a true or false value based on the state of the variables. The result is always a Boolean data type.

| **Name** | **Function** |
| --- | --- |
| Logical AND | Value returned is true when both given conditions are true. |
| Logical OR | Value returned is true when at least one given condition is true. |
| Logical NOT | Has the power to reverse the overall result. For example, if the value returned is true, it gives out false. |

An example program is shown below that demonstrates the different Logical Operators in Java.

Create a Java class named **LogicalOperatorsDemo**  and write the code below in that class.

| public class LogicalOperatorsDemo {  public static void main(String[] args)  {  boolean x = true;  boolean y = false;  System.out.println("x & y : " + (x & y));  System.out.println("x && y : " + (x && y));  System.out.println("x | y : " + (x | y));  System.out.println("x || y: " + (x || y));  System.out.println("x ^ y : " + (x ^ y));  System.out.println("!x : " + (!x));  } } |
| --- |

**Output**

| x & y : false  x && y : false  x | y : true  x || y: true  x ^ y : true  !x : false |
| --- |

## 

## Example: Bitwise Operators

Java provides Bitwise operators to manipulate the content of variables at the bit level.

These variables must be of numeric data type (char, short, int, or long). Java provides seven bitwise operators, as shown in the list below:

| **Left Operand** | **Right Operand** | **Operator** | **Result** |
| --- | --- | --- | --- |
| **11001100** | **01010101** | & (AND) | **01000100** |
| **11001100** | **01010101** | | (OR) | **11011101** |
| **11001100** | **01010101** | ^ (XOR) | **10011001** |
|  | **01010101** | ~ (NOT) | **10101010** |
| **11001100** | **2** | << (shift left) | **00110000** |
| **11001100** | **2** | >> (shift right) | **00110011** |

The example program below demonstrates the different Bitwise operators in Java.

Create a Java class named **BitwiseOperatorsDemo** and write the code below in that class.

| public class BitwiseOperatorsDemo {  public static void main(String[] args)  {  int x = 58; *//111010*  int y =13; *//1101*  System.out.println("x & y : " + (x & y)); *//returns 8 = 1000*  System.out.println("x | y : " + (x | y)); *//63=111111*  System.out.println("x ^ y : " + (x ^ y)); *//55=11011*  System.out.println("~x : " + (~x)); *//-59*  System.out.println("x << y : " + (x << y));  System.out.println("x >> y : " + (x >> y));  } } |
| --- |

**Output**

| x & y : 8  x | y : 63  x ^ y : 55  ~x : -59  x << y : 475136  x >> y : 0 |
| --- |

## 

## Example: Ternary Operator

Create a Java class named **TernaryOperatorDemo** and write the code below in that class.

| public class TernaryOperatorDemo {  public static void main(String[] args) {  int age = 18;  String result = age < 100 ?  "Less than 100" : "Greater than 100";  System.out.println(result); *//Less than 100*  } } |
| --- |

**Output**

| Less than 100 |
| --- |

## 

## Example: Unary Operators

Unary Operators only need a single operand, and they are used to increment, decrement, or eliminate a value.

Unary Operators:

| **Operator** | **Description** |
| --- | --- |
| ++ | Increment Operator - It will increment the value by 1. This operator can be used as a post-increment and as a pre-increment. Post increment means that we want to use the current value, and then perform the incrementation. Pre-increment means to perform the incrementation, and then use the value. |
| - - | Decrement Operator - It will decrement the value by 1 (opposite of increment operator). As we have seen in the above increment operator, this operator also has two use cases. The first is post-decrement, the second is pre-increment. Post-decrement means that the value is first used, and then the computation is performed. Pre-decrement means that the computation or the value is first decremented, and then used. |
| + | Unary Plus Operator - Presents the positive value. |
| - | Unary Minus Operator- Normally used for eliminating or negating values. |
| ! | Logical not Operator- It is used for inversion of the Boolean value. This operator reverses the value of a Boolean expression. For example, let’s say that a Boolean value is *true.* If we use the “**!”** operator with a true expression, it will give us the result as *false*. It inverses the Boolean value and returns the result. |

Let's see how they perform in real-world situations.

Create a Java class named **UnaryOperator** and write the code below in that class.

| public class UnaryOperator {  public static void main(String[] args)  {  int sum = +1;  *// sum is now 1*  System.out.println(sum);   sum--;  *// sum is now 0*  System.out.println(sum);   sum++;  *// sum is now 1*  System.out.println(sum);   sum = -sum;  *// sum is now -1*  System.out.println(sum);   boolean result = false;  *// false*  System.out.println(result);  *// true*  System.out.println(!result);  } } |
| --- |

**Output**

| 1  0  1  -1  false  true |
| --- |

**Submission Instructions:**

Include the following deliverables in your submission -

* + Submit your source code using the Start Assignment button in the top-right corner of the assignment page in Canvas.

**CANVAS STAFF USE ONLY: Canvas Submission Guideline:**

| **Instructions for Canvas Assignment Creation** |
| --- |
| **Assignment Name: GLAB - 303.2.1 - Core Java: Operators**  **Points:** **100**  **Assignment Group: Module 303: Java SE Review (Not Graded)**  **Display Grade As: Non-graded (This assignment does not count toward the final grade.)**.  **Complete/Incomplete**  **Do not count this assignment towards the final grade: Checked**  **Submission Types: File Uploads**  **Everything else is the default.** |

## 