OPERATOR IN JAVA

Operators in Java are the symbols used for performing specific operations in Java. Operators make tasks like addition, multiplication, etc which look easy although the implementation of these tasks is quite complex.

Types of Operators in Java

There are multiple types of operators in Java all are mentioned below:

1. **Arithmetic Operators**
2. **Unary Operators**
3. **Assignment Operator**
4. **Relational Operators**
5. **Logical Operators**
6. **Ternary Operator**
7. **Bitwise Operators**
8. **Shift Operators**
9. **instance of operator**

1. Arithmetic Operators

They are used to perform simple arithmetic operations on primitive data types.

*** : Multiplication** **+ : Addition**

/ : Division **- : Substraction**

% : Modulo

Example:

```java
// Java Program to implement
// Arithmetic Operators
import java.io.*;

// Drive Class
public class Topperworld {
    // Main Function
    public static void main (String[] args) {

        // Arithmetic operators
        int a = 10;
        int b = 3;

        System.out.println("a + b = " + (a + b));
        System.out.println("a - b = " + (a - b));
        System.out.println("a * b = " + (a * b));
        System.out.println("a / b = " + (a / b));
        System.out.println("a % b = " + (a % b));

    }
}
```

Output:

```
a + b = 13
a - b = 7
a * b = 30
a / b = 3
a % b = 1
```

2. Unary Operators

Unary operators need only one operand. They are used to increment, decrement, or negate a value.

- : Unary minus, used for negating the values.
- + : Unary plus indicates the positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.

- ++ : Increment operator, used for incrementing the value by 1. There are two varieties of increment operators.
  - Post-Increment: Value is first used for computing the result and then incremented.
  - Pre-Increment: Value is incremented first, and then the result is computed.
- – : Decrement operator, used for decrementing the value by 1. There are two varieties of decrement operators.
  - Post-decrement: Value is first used for computing the result and then decremented.
  - Pre-Decrement: The value is decremented first, and then the result is computed.
- ! : Logical not operator, used for inverting a boolean value.

Example:

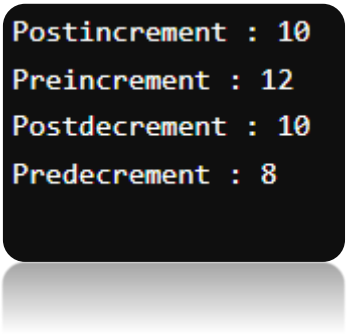
```java
// Java Program to implement
// Uniary Operators
import java.io.*;

// Driver Class
public class Topperworld {
    // main function
    public static void main(String[] args)
    {
        // Interger declared
        int a = 10;
        int b = 10;

        // Using uniary operators
        System.out.println("Postincrement : " + (a++));
        System.out.println("Preincrement : " + (++a));

        System.out.println("Postdecrement : " + (b--));
        System.out.println("Predecrement : " + (--b));
    }
}
```

Output:



3. Assignment Operator

'=' Assignment operator is used to assign a value to any variable. It has right-to-left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.

The general format of the assignment operator is:

variable = value;

In many cases, the assignment operator can be combined with other operators to build a shorter version of the statement called a Compound Statement. For example, instead of a = a+5, we can write a += 5.

- **+=**, for adding the left operand with the right operand and then assigning it to the variable on the left.
- **-=,** for subtracting the right operand from the left operand and then assigning it to the variable on the left.
- ***=**, for multiplying the left operand with the right operand and then assigning it to the variable on the left.
- **/=,** for dividing the left operand by the right operand and then assigning it to the variable on the left.
- **%=**, for assigning the modulo of the left operand by the right operand and then assigning it to the variable on the left.

Example:

```java
// Java Program to implement
// Assignment Operators
import java.io.*;

// Driver Class
public class Topperworld {
    // Main Function
    public static void main(String[] args)
    {

        // Assignment operators
        int f = 7;
        System.out.println("f += 3: " + (f += 3));
        System.out.println("f -= 2: " + (f -= 2));
        System.out.println("f *= 4: " + (f *= 4));
        System.out.println("f /= 3: " + (f /= 3));
        System.out.println("f %= 2: " + (f %= 2));
        System.out.println("f &= 0b1010: " + (f &= 0b1010));
        System.out.println("f |= 0b1100: " + (f |= 0b1100));
        System.out.println("f ^= 0b1010: " + (f ^= 0b1010));
        System.out.println("f <<= 2: " + (f <<= 2));
        System.out.println("f >>= 1: " + (f >>= 1));
        System.out.println("f >>>= 1: " + (f >>>= 1));
    }
}
```

Output:

```
f += 3: 10
f -= 2: 8
f *= 4: 32
f /= 3: 10
f %= 2: 0
f &= 0b1010: 0
f |= 0b1100: 12
f ^= 0b1010: 6
f <<= 2: 24
f >>= 1: 12
f >>>= 1: 6
```

4. Relational Operators

These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements.

The general format is,

variable relation_operator value

Some of the relational operators are-

- **==**, Equal to returns true if the left-hand side is equal to the right-hand side.
- **!=**, Not Equal to returns true if the left-hand side is not equal to the right-hand side.
- **<**, less than: returns true if the left-hand side is less than the right-hand side.
- **<=**, less than or equal to returns true if the left-hand side is less than or equal to the right-hand side.
- **>**, Greater than: returns true if the left-hand side is greater than the right-hand side.
- **>=**, Greater than or equal to returns true if the left-hand side is greater than or equal to the right-hand side.

Example:

```java
// Java Program to implement
// Relational Operators
import java.io.*;

// Driver Class
public class Topperworld {
    // main function
    public static void main(String[] args)
    {
        // Comparison operators
        int a = 10;
        int b = 3;
        int c = 5;

        System.out.println("a > b: " + (a > b));
        System.out.println("a < b: " + (a < b));
        System.out.println("a >= b: " + (a >= b));
        System.out.println("a <= b: " + (a <= b));
        System.out.println("a == c: " + (a == c));
        System.out.println("a != c: " + (a != c));
    }
}
```

Output:

```
a > b: true
a < b: false
a >= b: true
a <= b: false
a == c: false
a != c: true
```

5. Logical Operators

These operators are used to perform "logical AND" and "logical OR" operations, i.e., a function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e., it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Java also has "Logical NOT", which returns true when the condition is false and vice-versa

Conditional operators are:

- **&&**, **Logical AND**: returns true when both conditions are true.
- **||**, **Logical OR**: returns true if at least one condition is true.
- **!, Logical NOT**: returns true when a condition is false and vice-versa

Example:

```java
// Java Program to implemenet
// Logical operators
import java.io.*;

// Driver Class
public class Topperworld {
    // Main Function
    public static void main (String[] args) {
        // Logical operators
        boolean x = true;
        boolean y = false;

        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: true
!x: false
```

6.Ternary operator

The ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name Ternary.

The general format is:

condition ? if true : if false

The above statement means that if the condition evaluates to true, then execute the statements after the '?' else execute the statements after the ':'.

Example:

```
// Java program to illustrate
// max of three numbers using
// ternary operator.
public class operators {
    public static void main(String[] args)
    {
        int a = 20, b = 10, c = 30, result;

        // result holds max of three
        // numbers
        result
            = ((a > b) ? (a > c) ? a : c : (b > c) ? b : c);
        System.out.println("Max of three numbers = "
                           + result);
    }
}
```

Output:

```
Max of three numbers = 30
```

7. Bitwise Operators

These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.

- **&, Bitwise AND operator:** returns bit by bit AND of input values.
- **|, Bitwise OR operator:** returns bit by bit OR of input values.
- **^, Bitwise XOR operator:** returns bit-by-bit XOR of input values.
- **~, Bitwise Complement Operator:** This is a unary operator which returns the one's complement representation of the input value, i.e., with all bits inverted.

```java
\// Java Program to implement
// bitwise operators
import java.io.*;

// Driver class
 public class Topperworld {
    // main function
    public static void main(String[] args)
    {
        // Bitwise operators
        int d = 0b1010;
        int e = 0b1100;
        System.out.println("d & e: " + (d & e));
        System.out.println("d | e: " + (d | e));
        System.out.println("d ^ e: " + (d ^ e));
        System.out.println("~d: " + (~d));
        System.out.println("d << 2: " + (d << 2));
        System.out.println("e >> 1: " + (e >> 1));
        System.out.println("e >>> 1: " + (e >>> 1));
    }
}
```

Output:

```
d & e: 8
d | e: 14
d ^ e: 6
~d: -11
d << 2: 40
e >> 1: 6
e >>> 1: 6
```

Advantages of Operators in Java

The advantages of using operators in Java are mentioned below:

- **Expressiveness:** Operators in Java provide a concise and readable way to perform complex calculations and logical operations.
- **Time-Saving:** Operators in Java save time by reducing the amount of code required to perform certain tasks.
- **Improved Performance:** Using operators can improve performance because they are often implemented at the hardware level, making them faster than equivalent Java code.

Disadvantages of Operators in Java

The disadvantages of Operators in Java are mentioned below:

- **Operator Precedence:** Operators in Java have a defined precedence, which can lead to unexpected results if not used properly.
- **Type Coercion:** Java performs implicit type conversions when using operators, which can lead to unexpected results or errors if not used properly.
- **Overloading:** Java allows for operator overloading, which can lead to confusion and errors if different classes define the same operator with different behavior.