# Bitwise Operator in Java

In Java, an **operator** is a symbol that performs the specified operations. In this section, we will discuss only the **bitwise operator and its types** with proper examples.

## Types of Bitwise Operator

There are six types of the bitwise operator in Java:

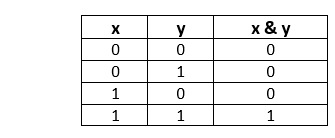
* Bitwise AND
* Bitwise exclusive OR
* Bitwise inclusive OR
* Bitwise Compliment
* Bit Shift Operators

| **Operators** | **Symbol** | **Uses** |
| --- | --- | --- |
| Bitwise AND | & | op1 & op2 |
| Bitwise exclusive OR | ^ | op1 ^ op2 |
| Bitwise inclusive OR | | | op1 | op2 |
| Bitwise Compliment | ~ | ~ op |
| Bitwise left shift | << | op1 << op2 |
| Bitwise right shift | >> | op1 >> op2 |
| Unsigned Right Shift Operator | >>> op >>> | number of places to shift |

Let's explain the bitwise operator in detail.

### Bitwise AND (&)

It is a binary operator denoted by the symbol **&**. It returns 1 if and only if both bits are 1, else returns 0.



Let's use the bitwise AND operator in a Java program.

**BitwiseAndExample.java**

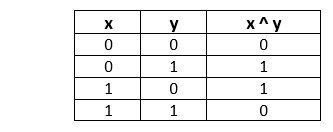
1. **public** **class** BitwiseAndExample
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. **int** x = 9, y = 8;
6. // bitwise and
7. // 1001 & 1000 = 1000 = 8
8. System.out.println("x & y = " + (x & y));
9. }
10. }

**Output**

x & y = 8

### Bitwise exclusive OR (^)

It is a binary operator denoted by the symbol **^** (pronounced as caret). It returns 0 if both bits are the same, else returns 1.



Let's use the bitwise exclusive OR operator in a Java program.

**BitwiseXorExample.java**

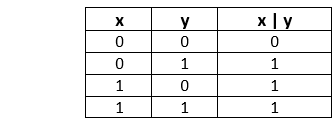
1. **public** **class** BitwiseXorExample
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. **int** x = 9, y = 8;
6. // bitwise XOR
7. // 1001 ^ 1000 = 0001 = 1
8. System.out.println("x ^ y = " + (x ^ y));
9. }
10. }

**Output**

x ^ y = 1

### Bitwise inclusive OR (|)

It is a binary operator denoted by the symbol **|** (pronounced as a pipe). It returns 1 if either of the bit is 1, else returns 0.



Let's use the bitwise inclusive OR operator in a Java program.

**BitwiseInclusiveOrExample.java**

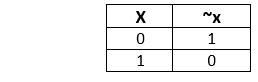
1. **public** **class** BitwiseInclusiveOrExample
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. **int** x = 9, y = 8;
6. // bitwise inclusive OR
7. // 1001 | 1000 = 1001 = 9
8. System.out.println("x | y = " + (x | y));
9. }
10. }

**Output**

x | y = 9

### Bitwise Complement (~)

It is a unary operator denoted by the symbol **~** (pronounced as the tilde). It returns the inverse or complement of the bit. It makes every 0 a 1 and every 1 a 0.



Let's use the bitwise complement operator in a Java program.

**BitwiseComplimentExample.java**

1. **public** **class** BitwiseComplimentExample
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. **int** x = 2;
6. // bitwise compliment
7. // ~0010= 1101 = -3
8. System.out.println("~x = " + (~x));
9. }
10. }

**Output**

~x = -3

### Bit Shift Operators

Shift operator is used in shifting the bits either right or left. We can use shift operators if we divide or multiply any number by 2. The general format to shift the bit is as follows:

1. variable << or >> number of places to shift;

For example, if a=10

1. a>>2; //shifts two bits
2. a>>4; //shifts 4 bits

Java provides the following types of shift operators:

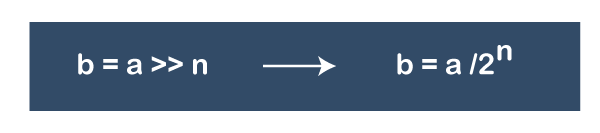
* Signed Right Shift Operator or Bitwise Right Shift Operator
* Unsigned Right Shift Operator
* Signed Left Shift Operator or Bitwise Left Shift Operator

#### Note: Java does not support the unsigned left shift operator (<<<).

### Signed Right Shift Operator (>>)

The signed right shift operator shifts a bit pattern of a number towards the **right** with a specified number of positions and fills 0. The operator is denoted by the symbol **>>.** It also preserves the leftmost bit (sign bit). If **0** is presented at the leftmost bit, it means the number is **positive**. If **1** is presented at the leftmost bit, it means the number is **negative**.

In general, if we write a>>n, it means to shift the bits of a number toward the right with a specified position (n). In the terms of mathematics, we can represent the signed right shift operator as follows:



#### Note: When we apply right shift operator on a positive number, we get the positive number in the result also. Similarly, when we apply right shift operator on a negative number, we get the negative number in the result also.

**Example: Apply the signed right shift operator with specified positions 4 if x = 256 and x = -256.**

**If x = 256**

256 >> 4

256/24 = **16**

If x = -256

-256 >> 4

-256/24 = **-16**

In the above example, we have observed that after shifting the operator 256 converted into 16 and -256 converted into -16.

Let's create a Java program and implement the left shift operator.

**SignedRightShiftOperatorExample.java**

1. **public** **class** SignedRightShiftOperatorExample
2. {
3. **public** **static** **void** main(String args[])
4. {
5. **int** x = 50;
6. System.out.println("x>>2 = " + (x >>2));
7. }
8. }

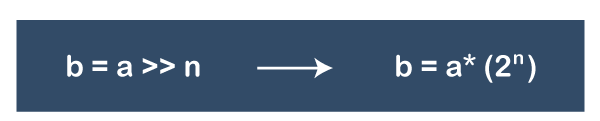
**Output**

x>>2 = 12

### **Signed Left Shift Operator (<<)**

The signed left shift operator (<<) shifts a bit pattern to the left. It is represented by the symbol **<<.** It also preserves the leftmost bit (sign bit). It does not preserve the sign bit.

In general, if we write a<<n, it means to shift the bits of a number toward the left with specified position (n). In the terms of mathematics, we can represent the signed right shift operator as follows:



**Example 1: What will be the result after shifting a<<3. The value of a is 20.**

Representation of 20 in binary is = 00010100

After performing the left shift operator, we get:

a << 3 = 10100000 (last three bits are the filled bits)

a << 3 = **160**

Let's check the result by using the formula.

20 << 3

20\*23 = 20\*8 = **160**

**Example 2: What will be the result after shifting a<<2. The value of a is -10.**

Representation of -10 in binary is = 11110110

a<<2 = 11011000 = **-40**

Let's check the result by using the formula.

-10 << 3

-10\*22 = -10\*4 = **-40**

Let's create a Java program and implement the signed left shift operator.

**SignedLeftShiftOperatorExample.java**

1. **public** **class** SignedLeftShiftOperatorExample
2. {
3. **public** **static** **void** main(String args[])
4. {
5. **int** x = 12;
6. System.out.println("x<<1 = " + (x << 1));
7. }
8. }

**Output**

x<<1 = 24

### Unsigned Right Shift Operator (>>>)

It shifts a zero at the leftmost position and fills 0. It is denoted by the symbol **>>>.** Note that the leftmost position after >> depends on the sign bit. It does not preserve the sign bit.

**Example: If a=11110000 and b=2, find a>>>b?**

a >>> b = 11110000 >>> 2 = **00111100**

The left operand value is moved right by the number of bits specified by the right operand and the shifted bits are filled up with zeros. Excess bits shifted off to the right are discarded.

Therefore, before shifting the bits the decimal value of a is 240, and after shifting the bits the decimal value of a is 60.

Let's create a Java program and use the unsigned right shift operator.

**UnsignedRightShiftOperatorExample.java**

1. **public** **class** UnsignedRightShiftOperatorExample
2. {
3. **public** **static** **void** main(String args[])
4. {
5. **int** x = 20;
6. System.out.println("x>>>2 = " + (x >>>2));
7. }
8. }