**Bitwise operator in java**

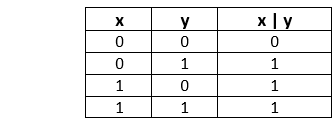
Bitwise operators are used to performing the manipulation of individual bits of a number. They can be used with any integral type (char, short, int, etc.).

There are 7 operators to perform bit-level operations in Java.

|  |  |  |
| --- | --- | --- |
| **Sr No** | **Operator** | **Description** |
| 1 | | | [Bitwise OR](https://www.programiz.com/java-programming/bitwise-operators" \l "or) |
| 2 | & | [Bitwise AND](https://www.programiz.com/java-programming/bitwise-operators" \l "and) |
| 3 | ^ | [Bitwise XOR](https://www.programiz.com/java-programming/bitwise-operators" \l "xor) |
| 4 | ~ | [Bitwise Complement](https://www.programiz.com/java-programming/bitwise-operators" \l "complement) |
| 5 | << | [Left Shift](https://www.programiz.com/java-programming/bitwise-operators" \l "left-shift) |
| 6 | >> | [Signed Right Shift](https://www.programiz.com/java-programming/bitwise-operators" \l "signed-right-shift) |
| 7 | >>> | [Unsigned Right Shift](https://www.programiz.com/java-programming/bitwise-operators" \l "unsigned-right-shift) |

**1. Java Bitwise OR Operator**

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.



Example :

int x = 9, y = 8;

// bitwise inclusive OR

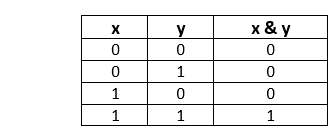
// 9 represented as 00001001 and 8 represented as 00001000 in binary

// 1001 | 1000 = 1001 = 9

System.out.println("x | y = " + (x | y));

1. **Java Bitwise AND Operator**

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



Example :

int x = 9, y = 8;

// bitwise and

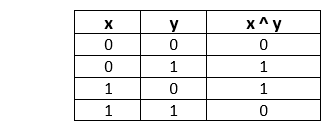
// 9 represented as 00001001 and 8 represented as 00001000 in binary

// 1001 & 1000 = 1000 = 8

System.out.println("x & y = " + (x & y));

1. **Java Bitwise XOR Operator**

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



Example :

int x = 9, y = 8;

// bitwise XOR

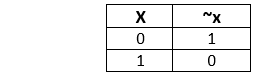
// 9 represented as 00001001 and 8 represented as 00001000 in binary

// 1001 ^ 1000 = 0001 = 1

System.out.println("x ^ y = " + (x ^ y));

1. **Java Bitwise Complement Operator**

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.



Example :

int x = 2;

// bitwise compliment

// ~0010= 1101 = -3

System.out.println("~x = " + (~x));

It is important to note that the bitwise complement of any integer N is equal to - (N + 1). For example, Consider an integer 35. As per the rule, the bitwise complement of 35 should be -(35 + 1) = -36.

1. **Java Left Shift Operator**

The left shift operator shifts all bits towards the left by a certain number of specified bits. It is denoted by <<. When we perform a 1 bit left shift operation on it, each individual bit is shifted to the left by 1 bit.

As a result, the left-most bit (most-significant) is discarded and the right-most position(least-significant) remains vacant. This vacancy is filled with 0s.

In the terms of mathematics, we can represent the signed right shift operator as follows:

b = a << n ------> b = a \* 2n

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

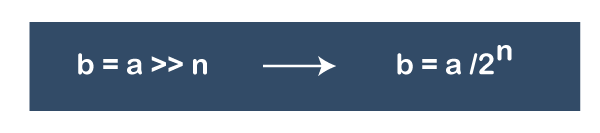
-10\*22 = -10\*4 = -40

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

1. **Java 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 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:



Example 1 :

int number1 = 8;

int number2 = -8;

// 2 bit signed right shift

System.out.println(number1 >> 2); // prints 2

System.out.println(number2 >> 2); // prints -2

Example 2 :

If x = 256

256 >> 4

256/24 = 16

If x = -256

-256 >> 4

-256/24 = -16

1. **Java Unsigned Right Shift Operator**

Java also provides an unsigned right shift. It is denoted by >>>. Here, the vacant leftmost position is filled with 0 instead of the sign bit.

Example 1 :

int number1 = 8;

int number2 = -8;

// 2 bit signed right shift

System.out.println(number1 >>> 2); // prints 2

System.out.println(number2 >>> 2); // prints 1073741822

Example 2:

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.