# Bitwise and BitShift Operators in Java - AND, OR, XOR, Signed Left and Right shift Operator Examples

Bitwise and Bit Shift operators in Java are powerful set of operators which allows you to manipulate bits on integral types like int, long, short, bytes and boolean data types in Java. Bitwise and Bit shift operators are among the fastest operator in Java

**Basics**

1. Binary format has just two bit,  0 and 1 and that's why is called binary.
2. In binary arithmetic :

0 + 0 =  0

0 + 1 =  1

1 + 1 = 10

1. Negation bitwise operation will change 0 to 1 and 1 to 0 and it’s denoted by ~.
2. OR bitwise operation will return 1 if any of operand is 1 and zero only if both operands are zeros.
3. AND bitwise operation will return 1 only if both operands are 1, otherwise zero.
4. XOR bitwise operation will return 1 if both operands are different e.g. one of them is 1 and other is zero and returns zero if both operands are same e.g. 1,1 or 0,0
5. Integral types in Java (int, long, short and byte) are signed numbers in Java where most significant bit (MSB) represent sign of number. 1 will denote a negative number and 0 as MSB will denote a positive numbers.
6. Negative numbers in Java are represented as 2's complement of number. 2's complement of a number is 1's complement + 1 and 1's complement means all zero will turn to 1 and all 1 turned to zero in a binary number e.g. 1's complement of 10 would be 01. and 2's complement of 10 would be 10+1 = 11 (all in binary).

## Bitwise operators in Java

Java provides several bitwise operator e.g. ~ for complement, & for AND bitwise operation, | for OR bitwise operation and ^ for bitwise XOR operation. All of these operator implements there respective operation and operates on bit level. By the way bitwise AND and OP operators are entirely different than logical AND && and logical OR operators ||, which operates on boolean variables and also implements AND and OR operation. Bitwise operator compares each bits of two operands, for example if you apply bitwise AND operator & on two integers ( which is a 32 bit data type in  Java), bitwise AND  will apply AND operation on each bits of both operands.

e.g.

int a = 2; //0010

int b = 4; //0100

System.out.println(" value of a XOR B in Java : " + (a^b) );

Output:

value of a XOR B in Java : 6

## Bit Shift operator in Java- Example

Apart from bitwise operators, Java also provides bit shift operators, which can be used to shift bit from one position to another on both left and right side in a number. Java provides three bit shift operator signed left shift operator "<<", signed right shift operator ">>" and unsigned right shift operator ">>>". Left shift operator with sign, shifts bit into left side and fill the empty place with zero, while right shift operator with sign, shifts the bit on right side and fills the empty place with value of sign bit.  For positive number it fills with zero and for negative numbers it fills with 1. On the other hand ">>>" right shift without sign operator will only shift the bit towards right without preserving sign of number. As per syntax of bitshift operator, left hand side of operand is the number whose bit needs to be shift and right hand side of operator is how many bits needs to shift. This will be much clear with following example of signed left shift, signed right shift and right shift without sign operators:

public class BitShiftTest {

    public static void main(String args[]) {

     int number = 8; *//0000 1000*

System.out.println("Original number : " + number);

*//left shifting bytes with 1 position*

     number = number<<1; *//should be 16 i.e. 0001 0000*

*//equivalent of multiplication of 2*

     System.out.println("value of number after left shift: " + number);

     number = -8;

*//right shifting bytes with sign 1 position*

     number = number>>1; *//should be 16 i.e. 0001 0000*

*//equivalent of division of 2*

     System.out.println("value of no after right shift with

sign: " + number);

     number = -8;

*//right shifting bytes without sign 1 position*

     number = number>>>1; *//should be 16 i.e. 0001 0000*

*//equivalent of division of 2*

     System.out.println("value of number after right shift with

sign: " + number);

    }

}

Output:

Original number : 8

value of number after left shift: 16

value of number after right shift with sign: -4

value of number after right shift with sign: 2147483644

From above example of bit shift operators in Java, you can imply that signed left shift operators are equivalent of multiplying by 2 and signed r  
  
**Important points to remember while using bit shift operators in Java**

1. Smaller types like byte, short are promoted to int before applying bit shift operator in Java. This requires explicit [casting](http://javarevisited.blogspot.sg/2012/12/what-is-type-casting-in-java-class-interface-example.html) on lower type of result.

2. If number of shift positions exceeds with number of bits in a variable, then remainder operator is used to calculate effective bit movement. For example int variables contains 32 bit, and if you shift bits to 33 times, then its equivalent of shifting bits 33%32 or just 1 time. e.g. 8 >> 33 is equal to 8 >> 1 and this is true for all bit shift operators.

3. Since bit shift operators can simulate divide by 2 and multiplied by 2 operation they can be used to implement fast multiplication and division but on the cost of readability as its difficult to comprehend code written using bit shift operators in Java.