Op-VIII: Bitwise Operators in Java

**Introduction:**

The Java programming language also provides operators that perform operations on **bits** and only on **integral types**. These operators are less commonly used but an important optimization skill. Therefore, in brief; the intent is to simply make you aware that these operators exist.

**What is bit manipulation?**

[Bit manipulation](https://www.educative.io/blog/bit-manipulation-algorithm) is the process of applying logical operations on a sequence of bits, the smallest form of data in a computer, to achieve a required result. Bit manipulation has constant time complexity and process in parallel, meaning it is very efficient on all systems.Most programming languages will have you work with abstractions, like [**objects**](https://www.educative.io/blog/object-oriented-programming-concepts-java) or **variables**, rather than the bits they represent. However, direct bit manipulation is needed to improve performance and reduce error in certain situations.Bit manipulation requires a strong knowledge of [binary and binary conversion](https://www.educative.io/blog/computer-number-systems-binary-hexadecimal-conversions).

**When we require bit manipulation?**

We require bit manipulation:

* Low-level device control
* Error detection and correction algorithms
* Data compression
* Encryption algorithms
* Optimization

**What are the Advantages of Bitwise Operators?**

* They are fast and simple actions.
* They are directly supported by the processor.
* They are used to manipulate values for comparisons and calculations.
* Bitwise operations are incredibly simple and faster than arithmetic operations.

**What are Bitwise Operators?**

**Bitwise operators work on**[binary digits](https://medium.com/coderscorner/number-systems-decimal-binary-octal-and-hexadecimal-5e567e55ab28)**or bits of values.** We can apply these to the

integer types – ***byte, short, int, long*** and also oncharacter type -***char***

**how they work?**

**Bitwise operators work on a binary equivalent of decimal numbers and perform operations on them bit by bit as per the given operator:**

* **First,** the operands are converted to their binary representation
* **Next,** the operator is applied to each binary number and the result is calculated
* **Finally,** the result is converted back to its decimal representation

## What are the Categories of Bitwise Operators?

## Bitwise operators are further classified as bitwise logical and bitwise shift operators. Let's now go through each type

## I)Bitwise Logical Operators

Theyare:AND (**&**), OR (**|**), NOT(**~**) and XOR (**^**).

## II)Bitwise Shift Operators

Theyare:Left Shift (**<<**), Right Shift (**>>**) and Unsigned Right Shift (**>>>**).

## Bitwise Logical Operators

### Bitwise *AND* (&)

### Compares each binary digit of two integers andreturns back resultant value as follows...

**“If both thebits are 1, result is 1, otherwise it returns 0”**

NOTE: This is similar to the Logical AND ‘&&’

int a=5; //+ve // 0000 0101

int b=6; //+ve // **&** 0000 0110

// weights->signbit8421 (0 \* 8 = 0)

// ============

System.out.println(a&b);// //4 0000 0100

### Bitwise *OR* (|)

### Compares each binary digit of two integers and returns back resultant value as follows...

**“If both the bits are 0, result is 0, otherwise it returns 1”**

NOTE: This is similar to the Logical OR ‘||’

int a =5; //+ve // 0000 0101

int b=6; //+ve // **|**0000 0110

// ============

System.out.println( a**|** b); // 7 0000 0111

// ============

// weights->**signbit**0421 (0 \* 8 = 0)

### Bitwise *NOT/COMPLEMENT* (~)

### It takes only binary digit of ONE integer and returns back resultant value as follows...

**“If a bitis 0 result is 1, and vice versa”**

NOTE: This is similar to the Logical NOT ‘! ’

int a =5; //+ve // 0101

// ============

System.out.println(~a); // -6🡪1010

// ============

// weights->**signbit**8421 (-1 \* 8 = **-**8)

**FORMULA :** bitwise complement of any integer **N** is equal to **- (N + 1)**.i.e., -(5+1)=-6

### Bitwise *XOR* (^)

### Compares each binary digit of two integers and returns back resultant value as follows...

**“If both the bits are SIMILAR, result is 0, otherwise it returns 1”**

NOTE: Such Logical Operator is NOT existed.

int a =5; //+ve // 0101

int b=6; //+ve // **^** 0110

// ============

System.out.println( a**^**b); // 3🡪0011

// ============

// weights->**signbit**8421 (0 \* 8 = 0)

### Bitwise Logical Operators Table

**A B A|B A&B ~A A^B**

0 0 0 0 1 0

1 0 1 0 0 1

0 1 1 0 1 1

1 1 1 1 0 0

## Bitwise Shift Operators

These operators shift all the bits of a value either to the left or right based on the Operator.

COMMON SYNTAX : <Value><SHIFTop><Number\_Of\_Times>

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

This shifts all the **bits to the LEFT**by the number of times specified on RHS **operand.** After the LEFT shift, the **empty space** on the**RIGHT is filled with 0s.**

**FORMULA:**This is**equivalent toMultiplication by** 2**^n**.

Example for 5 is**=>5 \* (2^2) = 20**

int a =5; //+ve // **00**00 0101

//left side 2 bits 00 are dropped space **--**on RHS0001 01**--**

// ============

System.out.println(a<<2); // 20🡪000101**00** //spaces filled by **00**

// ============

// weights->**signbit**168421 (1 \* 16 = 16)

### Signed *Right Shift*( >> )

This shifts all the **bits to the RIGHT**by the number of times specified on RHS **operand.** After the RIGHT shift, the **empty space** on the**RIGHT is filled depending on the given number Positive/ Negative.** It also **preserves the** leftmost bit **(sign bit-MSB=>MostSignificantBit).**

**FORMULA:**This is **equivalent toDivisionby** 2**^n**.

Example for 5 is**=>20 / (2^2) = 5**

**RULE:When an input number is NEGATIVE, then the empty spaces will be filled with 1 otherwise Like Left SHIFT they are filled by 0.**

int a =20; //+ve // 0001 0100

//right side 2 bits 00 are dropped space **--**on LHS **--**000101

// ============

System.out.println(a>>2 ); // 5🡪**00**000101//spaces filled by**00**

// ============

// weights->**signbit16**8421 (0 \* 16 = 0)

### UnSigned*Right Shift*( >>> )

This shifts all the **bits to the RIGHT**by the number of times specified on RHS **operand,very similar to the signed Right shift operator(>>),** the only **difference** is that the **empty spaces** in the left are filled with **0** irrespective of whether the **number is Positive or Negative**, therefore, the**result will always be a positive integer**

int a = 12; // 0000 1100

// -------------

int ls = a>>>2; // 0000 0011 // 3

**Is there unsigned Left Shift in Java?**

Unlike unsigned Right Shift, there is no “<<<” operator in Java because the logical (<<) and arithmetic left-shift (<<<) operations are identical

[**How do I perform an unsigned right shift (>>> in Java) in C/C++?**](https://stackoverflow.com/questions/2429479/how-do-i-perform-an-unsigned-right-shift-in-java-in-c-c)

In C, to get an unsigned shift, you just do a shift on an unsigned type.

unsigned int result = (unsigned int)valueToBeShifted>>shiftAmount;

Note that there is no guarantee that >> on a signed type gives you a signed shift in C -- this is implementation defined behavior. Most common implementations produce a signed shift if the type is signed, howeve

**Use Cases**

Some potential use cases of bitwise operators are:

* Communication stacks where the individual bits in the header attached to the data signify important information
* In embedded systems to set/clear/toggle just one single bit of a specific register without modifying the remaining bits
* To encrypt data for safety issues using the XOR operator
* In data compression by converting data from one representation to another, to reduce the amount of space used

**NOTE :** Bit manipulation is also a common topic in coding interviews, especially with [FAANG companies](https://www.educative.io/blog/why-a-faang-company-may-not-be-right-for-you). These interviewers expect you to have a basic understanding of bits, fundamental bit operators, and generally understand the thought process behind bit manipulation.

Having this knowledge demonstrates that you’re a well-rounded developer who understands both the specific tools and the foundation of computer science.

If you’re applying for a role that will work with embedded systems or other low-level systems, you’ll encounter more bit questions. In short, the closer your role is to machine level, the more bit manipulation questions you’ll encounter.

The best way to prepare for bit manipulation questions is to practice using each bitwise operator and brush up on your binary to decimal conversions.