BITWISE OPERATOR’S

In Java Bitwise Operators allow access and modification of a particular bit inside a section of the data. It can be applied to *integer* types **int,short,char** or *bytes*, and cannot be applied to **float** and **double**.

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

1.Bitwise OR

2.Bitwise AND

3.Bitwise Exclusive OR(XOR)

4. Bitwise Unary NOT

5. Left Shift

6.Signed right shift

7.Unsigned Right Shift

Operator	Meaning	Work
&	Binary AND Operator	There are two types of AND operators in Java: the logical && and the binary &. Binary & operator work very much the same as logical && operators works, except it works with two bits instead of two expressions. The "Binary AND operator" returns 1 if both operands are equal to 1.
|	Binary OR Operator	Like "AND operators ", Java has two different "OR" operators: the logical || and the binary |. Binary | Operator work similar to logical || operators works, except it, works with two bits instead of two expressions. The "Binary OR operator" returns 1 if one of its operands evaluates as 1. if either or both operands evaluate to 1, the result is 1.
^	Binary XOR Operator	It stands for "exclusive OR" and means "one or the other", but not both. The "Binary XOR operator" returns 1 if and only if exactly one of its operands is 1. If both operands are 1, or both are 0, then the result is 0.
~	Binary Complement Operator	In binary arithmetic, we can calculate the binary negative of an integer using 2's complement. 1's complement changes **0** to **1** and **1** to **0**. And, if we add **1** to the result of the 1's complement, we get the 2's complement of the original number.
<<	Binary Left Shift Operator	As we can see from the image above, we have a 4-digit number. When we perform a 1 bit left shift operation on it, each individual bit is shifted to the left by **1** bit.
>>	Binary Right Shift Operator	The signed right shift operator shifts all bits towards the right by a certain number of specified bits. It is denoted by `>>`. When we shift any number to the right, the least significant bits (rightmost) are discarded and the most significant position (leftmost) is filled with the sign bit.
>>>	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

1. Java Bitwise OR Operator

The bitwise OR `|` operator returns 1 if at least one of the operands is 1. Otherwise, it returns 0.

The following truth table demonstrates the working of the bitwise OR operator. Let a and b be two operands that can only take binary values i.e. 1 or 0.

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

The above table is known as the "Truth Table" for the bitwise OR operator.

Let's look at the bitwise OR operation of two integers 12 and 25.

```
12  = 00001100(In Binary)

25 =  00011001(In Binary)

Bitwise OR Operation of 12 and 25

00001100

00011001

00011101 =29 (In Decimal)
```

```
EX: class Main {
  public static void main(String[] args) {

    int number1 = 12, number2 = 25, result;

    // bitwise OR between 12 and 25
    result = number1 | number2;
    System.out.println(result);    // prints 29
  }
}
```

2. Java Bitwise AND Operator

The bitwise AND `&` operator returns 1 if and only if both the operands are 1. Otherwise, it returns 0. The following table demonstrates the working of the bitwise AND operator. Let a and b be two operands that can only take binary values i.e. 1 and 0.

a	b	a|b
0	0	0

0	1	0
1	0	0
1	1	1

Let's take a look at the bitwise AND operation of two integers 12 and 25.

```
12  = 00001100(In Binary)

25 =  00011001(In Binary)

//Bitwise OR Operation of 12 and 25

00001100

00011001

00001000 = 8 (In Decimal)
```

EX:
```
class Main {
  public static void main(String[] args) {

    int number1 = 12, number2 = 25, result;

    // bitwise AND between 12 and 25
    result = number1 & number2;
    System.out.println(result);    // prints 8
  }
}
```

3. Java Bitwise XOR Operator

The bitwise XOR `^` operator returns 1 if and only if one of the operands is 1. However, if both the operands are 0 or if both are 1, then the result is 0.
The following truth table demonstrates the working of the bitwise XOR operator. Let a and b be two operands that can only take binary values i.e. 1 or 0.

a	b	a|b
0	0	0
0	1	1
1	0	1

1	1	0

Let's look at the bitwise XOR operation of two integers 12 and 25.

```
12  = 00001100(In Binary)

25 =  00011001(In Binary)

//Bitwise OR Operation of 12 and 25

00001100

00011001

00010101 = 21 (In Decimal)
```

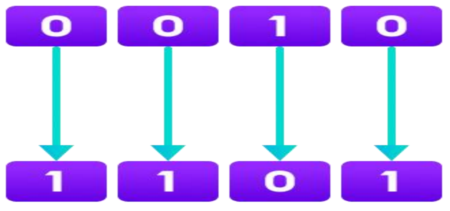
```
EX: class Main {
  public static void main(String[] args) {

    int number1 = 12, number2 = 25, result;

    // bitwise XOR between 12 and 25
    result = number1 ^ number2;
    System.out.println(result);    // prints 21
  }
}
```

4. Java Bitwise Complement Operator

The bitwise complement operator is a unary operator (works with only one operand). It is denoted by ~. It changes binary digits **1** to **0** and **0** to **1**.



Java Bitwise Complement Operator

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**. Now let's see if we get the correct answer or not.\

```
35 = 00100011 (In Binary)

// using bitwise complement operator

 ~ 00100011

   11011100
```

In the above example, we get that the bitwise complement of **00100011** (**35**) is **11011100**. Here, if we convert the result into decimal we get **220**.
However, it is important to note that we cannot directly convert the result into decimal and get the desired output. This is because the binary result **11011100** is also equivalent to **-36**.
To understand this we first need to calculate the binary output of **-36**.

```
EX: class Main {
  public static void main(String[] args) {

    int number = 35, result;

    // bitwise complement of 35
    result = ~number;
    System.out.println(result);    // prints -36
  }
}
```

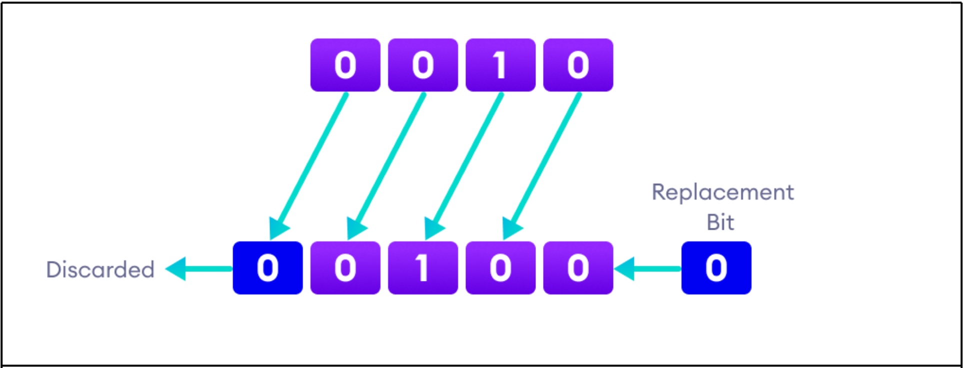
Java Shift Operators

There are three types of shift operators in Java:

- Signed Left Shift (<<)
- Signed Right Shift (>>)
- Unsigned Right Shift (>>>)

5. 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 `<<`.



Java 1 bit Left Shift Operator

As we can see from the image above, we have a 4-digit number. 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**.

```
EX: class Main {
  public static void main(String[] args) {

    int number = 2;

    // 2 bit left shift operation
    int result = number << 2;
    System.out.println(result);    // prints 8
  }
}
```

6. Java Signed Right Shift Operator

The signed right shift operator shifts all bits towards the right by a certain number of specified bits. It is denoted by `>>`.
When we shift any number to the right, the least significant bits (rightmost) are discarded and the most significant position (leftmost) is filled with the sign bit.

```
// right shift of 8

 8 = 1000 (In Binary)

// perform 2 bit right shift

 8 >> 2:
```

```
1000 >> 2 = 0010 (equivalent to 2)
```

Here, we are performing the right shift of **8** (i.e. sign is positive). Hence, there no sign bit. So the leftmost bits are filled with **0** (represents positive sign).

```
// right shift of -8

8 = 1000 (In Binary)

 1's complement = 0111

 2's complement:

     0111

        + 1

_______

     1000

Signed bit = 1

// perform 2 bit right shift

 8 >> 2:

1000 >> 2 = 1110 (equivalent to -2)
```

Here, we have used the signed bit **1** to fill the leftmost bits.

EX:
```
class Main {
  public static void main(String[] args) {

    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
  }
}
```

7. 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.

For example,

```
// unsigned right shift of 8
 8 = 1000
 8 >>> 2 = 0010
// unsigned right shift of -8
 -8 = 1000 (see calculation above)
-8 >>> 2 = 0010
```

EX:

```
class Main {
  public static void main(String[] args) {

    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
  }
}
```