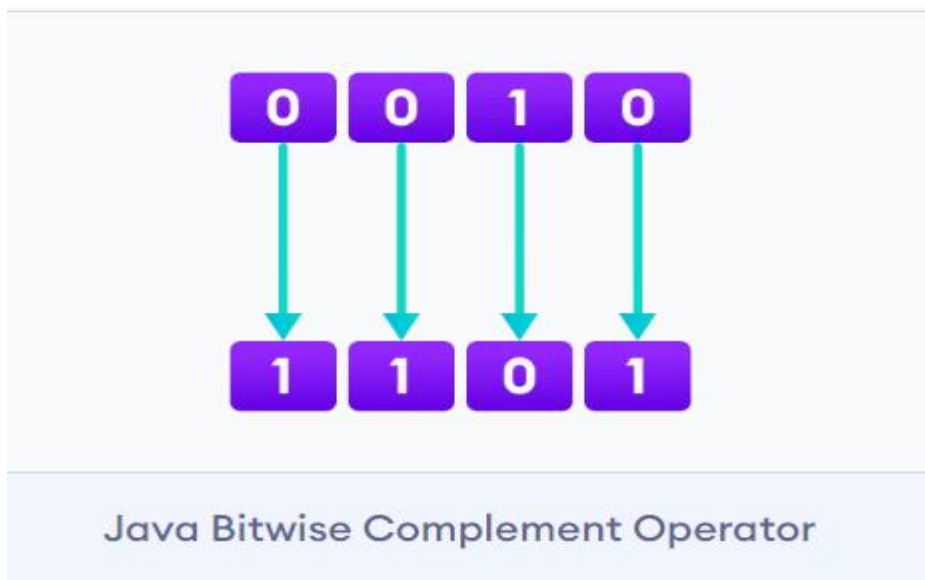


## BITWISE COMPLEMENT OPERATOR:

The unary bitwise complement operator "~" inverts a bit pattern; it can be applied to any of the integral types, making every "0" a "1" and every "1" a "0".

For example, a byte contains 8 bits; applying this operator to a value whose bit pattern is "00000000" would change its pattern to "11111111".

It changes binary digits **1** to **0** and **0** to **1**.



**NOTE:** It is important to note that the bitwise complement of any integer **N** is equal to **-(N + 1)**.

### 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**.

## 2's Complement

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. For example,

```
// compute the 2's complement of 36
```

```
36 = 00100100 (In Binary)
```

```
1's complement = 11011011
```

```
2's complement:
```

```
11011011
```


```
+ 1
```

```
-----
```

```
11011100
```

Here, we can see the 2's complement of **36** (i.e. **-36**) is **11011100**. This value is equivalent to the bitwise complement of **35**.

Hence, we can say that the bitwise complement of **35** is **-(35 + 1) = -36**.

Main.java	  	Output
<pre>1 class Main { 2     public static void main(String[] args) { 3 4         int number = 35, result; 5 6         // bitwise complement of 35 7         result = ~number; 8         System.out.println(result);    // prints -36 9     } 10 }</pre>		<pre>java -cp /tmp/WItIWGwweY Main -36</pre>

### Bitwise Not:

Bitwise Not (!) is a Boolean Operator which is similar to Bitwise Complement.

**EXAMPLE:** !True the output is False

### BigInteger:

Java provides a custom class named BigInteger for handling very large integers (which is bigger than 64 bit values). This class can handle very large integers and the size of the integer is only limited by the available memory of the JVM. However BigInteger should only be used if it is absolutely necessary as using BigInteger is less intuitive compared to built-in types (since Java doesn't support Operator overloading) and there is always a performance hit associated with its use.

BigInteger operations are substantially slower than built-in integer types. Also the memory space taken per BigInteger is substantially high (averages about 80 bytes on a 64-bit JVM) compared to built-in types.

Computation of factorial is a good example of numbers getting very large even for small inputs. We can use BigInteger to calculate factorial even for large numbers!

```
import java.math.BigInteger;
public class BigIntegerFactorial {

    public static void main(String[] args) {
        calculateFactorial(50);
    }
    public static void calculateFactorial(int n) {

        BigInteger result = BigInteger.ONE;
        for (int i=1; i<=n; i++) {
            result = result.multiply(BigInteger.valueOf(i));
        }
        System.out.println(n + "! = " + result);
    }
}
```

The factorial output for an input value of 50 is,

50!=30414093201713378043612608166064768844377641568960512  
000000000000