Operators

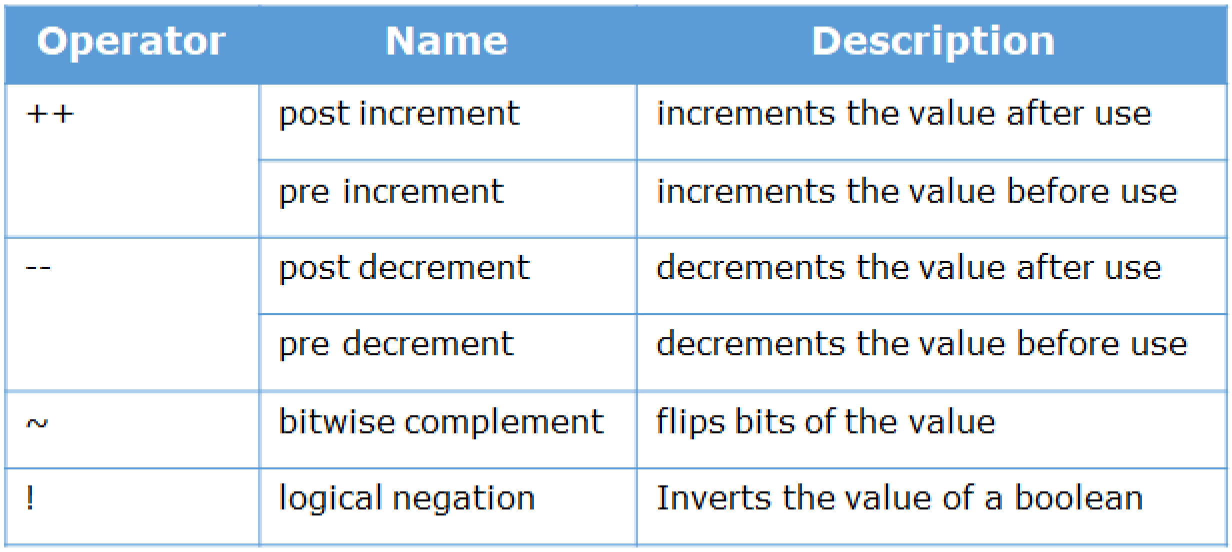
Definition

Operators are the symbols used to perform specific operations. There are various operators that can be used for different purposes.
The operators are categorized as:

1. Unary
2. Arithmetic
3. Relational
4. Logical
5. Ternary
6. Assignment
7. Bitwise

Unary Operators

Unary operators act upon only one operand and perform operations such as increment, decrement, negating an expression or inverting a boolean value.

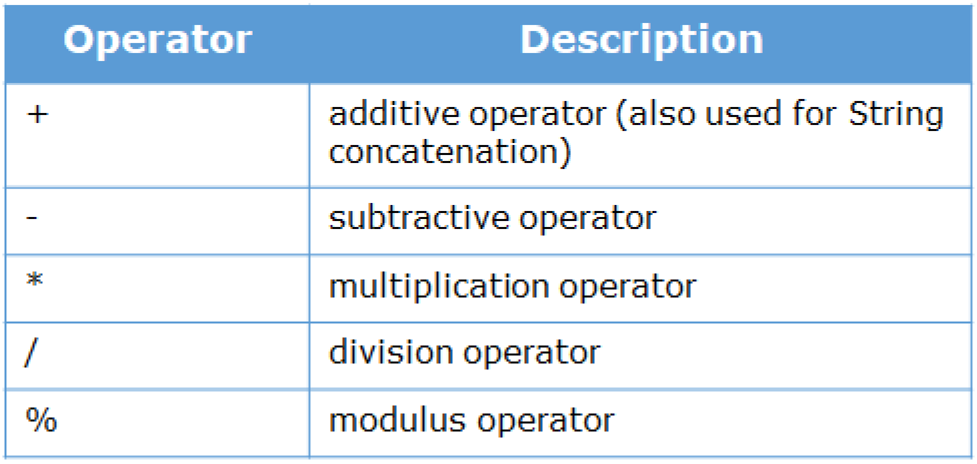


Operator	Name	Description
++	post increment	increments the value after use
	pre increment	increments the value before use
--	post decrement	decrements the value after use
	pre decrement	decrements the value before use
~	bitwise complement	flips bits of the value
!	logical negation	Inverts the value of a boolean

```
public static void main(String args[]) {
    int numOne = 10;
    int numTwo = 5;
    boolean isTrue = true;
    System.out.println(numOne++ + " " + ++numOne);     //Output will be 10 12
    System.out.println(numTwo-- + " " + --numTwo);     //Output will be 5 3
    System.out.println(!isTrue + " " + ~numOne);     //Output will be false -13
}
```

Arithmetic Operators

Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication and division.



Operator	Description
+	additive operator (also used for String concatenation)
-	subtractive operator
*	multiplication operator
/	division operator
%	modulus operator

```
public static void main(String args[]) {
    int numOne = 10;
    int numTwo = 5;
    System.out.println(numOne + numTwo);            //Output will be 15
    System.out.println(numOne - numTwo);            //Output will be 5
    System.out.println(numOne * numTwo);            //Output will be 50
    System.out.println(numOne / numTwo);            //Output will be 2
    System.out.println(numOne % numTwo);            //Output will be 0
}
```

Relational Operators

Relational operators are used to compare two values. The result of all the relational operations is either true or false.

Operator	Description
==	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
!=	not equal to

```
public static void main(String args[]) {
    int numOne = 10;
    int numTwo = 5;
    System.out.println(numOne > numTwo); //Output will be true
}
```

Logical Operators

Logical operators are used to combine two or more relational expressions or to negate the result of a relational expression.

Operator	Name	Description
&&	AND	result will be true only if both the expressions are true
||	OR	result will be true if any one of the expressions is true
!	NOT	result will be false if the expression is true and vice versa

Assume A and B to be two relational expressions. The below tables show the result for various logical operators based on the value of expressions, A and B.

A	B	A&&B	A||B
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

A	!A
true	false
false	true

```
public static void main(String args[]) {
    int numOne = 100;
    int numTwo = 20;
    int numThree = 30;
    System.out.println(numOne > numTwo && numOne > numThree);        //Output will be true
}
```

Ternary Operator

Ternary operator is used as a single line replacement for if-then-else statements and acts upon three operands.

Syntax:

<condition> ? <value if condition is true> : < value if condition is false>

```
public static void main(String args[]) {
    int numOne = 10;
    int numTwo = 5;
    int min = (numOne < numTwo) ? numOne : numTwo;
    System.out.println(min);                                          //Output will be 5
}
```

Here, first the condition (numOne < numTwo) is evaluated. The result is false and hence, min will be assigned the value numTwo.

Assignment Operators

Assignment operator is used to assign the value on the right hand side to the variable on the left hand side of the operator.

Operator	Description
=	assigns the value on the right to the variable on the left
+=	adds the current value of the variable on left to the value on the right and then assigns the result to the variable on the left
-=	subtracts the value of the variable on the right from the current value of the variable on left and then assigns the result to the variable on the left
*=	multiplies the current value of the variable on left to the value on the right and then assigns the result to the variable on the left
/=	divides the current value of the variable on left by the value on the right and then assigns the result to the variable on the left

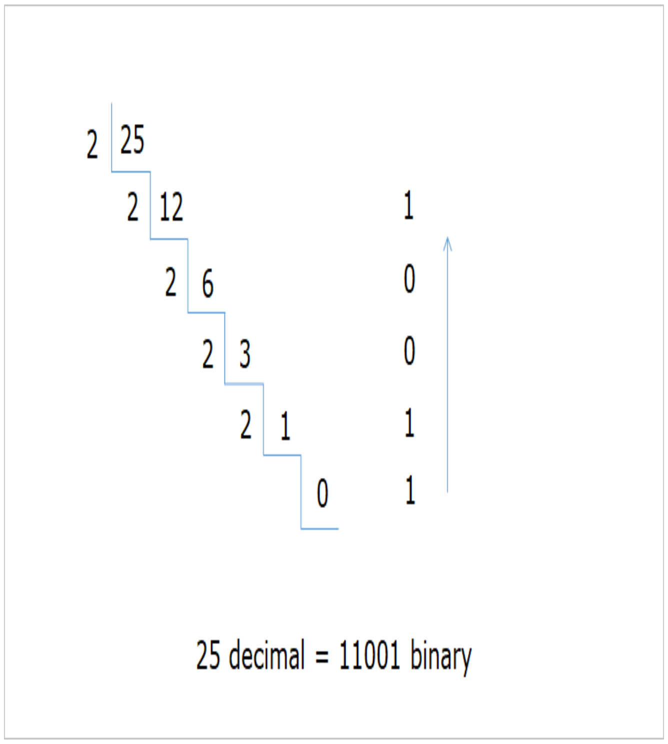
```
public static void main(String args[]) {
    int numOne = 10; //The value 10 is assigned to numOne
    System.out.println(numOne);                   //Output will be 10
    numOne += 5;
    System.out.println(numOne);                    //Output will be 15
    numOne -= 5;
    System.out.println(numOne);                   //Output will be 10
    numOne *= 5;
    System.out.println(numOne);                   //Output will be 50
    numOne /= 5;
    System.out.println(numOne);                   //Output will be 10
}
```

Bitwise Operators

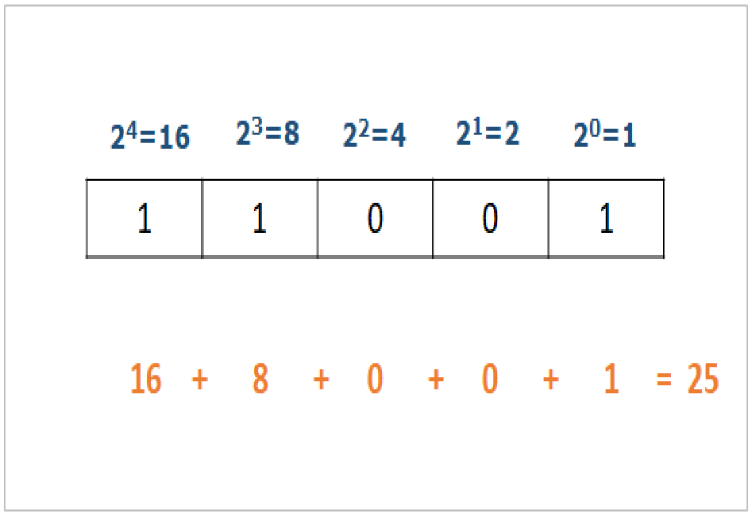
Bitwise operators are used to perform manipulation of individual bits of a number.

Conversion of decimal numbers to binary:

The decimal or the base 10 number system is used in everyday life but the binary number system is the basis for representing data in computing systems.



1. Divide the decimal number by 2.
2. Write the number on the right hand side. This will be either 1 or 0.
3. Divide the result of the division and again by 2 and write the remainder.
4. Continue this process until the result of the division is 0.
5. The first remainder that you received is the least significant bit and the last remainder is the most significant bit.



how to convert the binary number back to decimal number?

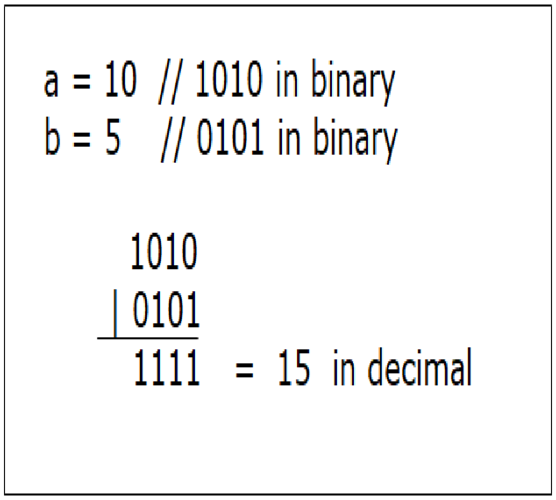
The decimal number is equal to the sum of binary digits (d_n) times their power of 2 (2^n).

For example: 11001.

11001 =
$1*2^4+1*2^3+0*2^2+0*2^1+1*2^0=16+8+0+0+1=25$

Types of Bitwise Operators:

- Bitwise OR (|) :

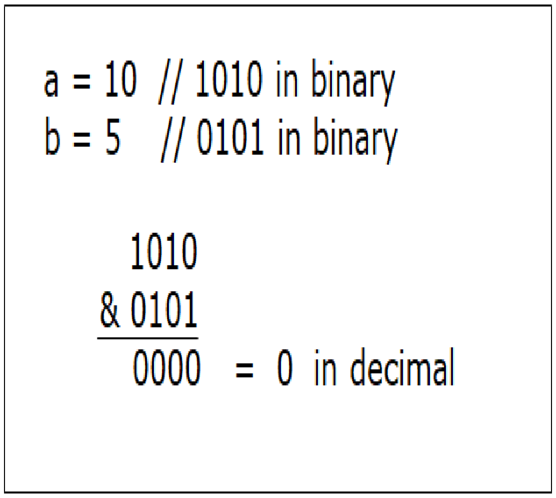


```
a = 10  // 1010 in binary
b = 5   // 0101 in binary

     1010
   | 0101
     1111  =  15  in decimal
```

It returns bit by bit OR of the input values. If either of the bits is 1, then it gives 1, else it gives 0.

- Bitwise AND (&):



```
a = 10  // 1010 in binary
b = 5   // 0101 in binary

     1010
   & 0101
     0000  =  0  in decimal
```

It returns bit by bit AND of the input values. If both the bits are 1, then it gives 1, else it gives 0.

- Left shift operators (<<)
  It takes two operators and left shifts the bits of the first operand. The second operand decides the number of places to shift. It fills 0 on voids left as a result.
  E.g   The output of 10<<1 is 20 if the numbers are stored in 32 bit system.
  10 is represented as 00000000 00000000 00000000 00001010.
  After left shifting by 1 bit, the result becomes 00000000 00000000 00000000 00010100 which is 20.

The 0 that is highlighted is present because of the void.

- Signed right shift operator (>>)
  It takes two operators and right shifts the bits of the first operand. The second operand decides the number of places to shift. It fills 0 on voids left as a result if the first operand is positive else it fills 1.

  For positive numbers:
  The output of 10>>1 is 5.
  10 is represented as 00000000 00000000 00000000 00001010.
  After right shifting by 1 bit, the result becomes 00000000 00000000 00000000 00000101 which is 5.

  For negative numbers:
  The output of -10>>1 is -5.
  -10 is represented as 11111111 11111111 11111111 11110110.
  After right shifting by 1 bit, the result becomes 11111111 11111111 11111111 11111011 which is -5.

- Unsigned right shift operator (>>>)
  It takes two operators and right shifts the bits of the first operand. The second operand decides the number of places to shift. It fills 0 on voids left as a result.

  For positive numbers:
  The output of 10>>>1 is 5.
  10 is represented as 00000000 00000000 00000000 00001010.
  After right shifting by 1 bit, the result becomes 00000000 00000000 00000000 00000101 which is 5.

  For negative numbers:
  The output of -10>>>1 is 214783643.
  -10 is represented as 11111111 11111111 11111111 11110110.
  After right shifting by 1 bit, the result becomes 01111111 11111111 11111111 11111011 which is 214783643.

Operator precedence

Operators also have precedence. Operators with higher precedence are evaluated before the operators with lower precedence. When an expression has two operators with the same precedence one after the other, the expression is evaluated according to their associativity. The associativity of operators can be left-to-right, right-to-left or no associativity.

Eg : num1 = num2 = num3 = 39 is treated as (num1 = (num2 = (num3 = 39))), leaving all three variables with the value 39, since the = operator has right-to-left associativity. On the other hand, 84 / 4 / 3 is treated as ((84 / 4) / 3) since the / operator has left-to-right associativity. Some operators, like the postfix and prefix operators, are *not* associative: for example, the expression num++-- is invalid.

Operator Name	Operator
parenthesis/ brackets	()
postfix	++, --
unary, prefix	++, --, +, -, ~, !
multiplicative	*, /, %
additive	+, -
shift	<<, >>, >>>
relational	<, >, <=, >=
equality	==, !=
logical AND	&&
logical OR	||
ternary	?:
assignment	=, +=, -=, *=, <<=, >>=, >>>=

```
5 + 4 * 9 % (3 + 1) / 6 - 1
   5 + 4 * 9 % 4 / 6 - 1
     5 + 36 % 4 / 6 - 1
        5 + 0 / 6 - 1
          5 + 0 - 1
            5 - 1
              4
```

Practice Programs

1.

```
class Tester {

   public static void main(String args[]) {

           int a = 10;
           int b = 20;
           System.out.println(a << 2);                    // 40
           System.out.println(b >> 3);                    // 2

           System.out.println(a >> 2);                    // 2
           System.out.println(a >>> 2);                   // 2

           int c = -1;
           System.out.println(c >> 2);                    // -1
           System.out.println(c >>> 2);                   // 1073741823

           // There is no unsigned left shift operator(<<<). The below line leads
           // to an error
           // System.out.println(a<<<2);


          // Logical OR(||) does not check second condition if first is true
           // Bitwise OR(|) always checks both conditions even if first condition
           // is true or false
           System.out.println(a | b);                     // 30

           // Logical AND(&&) does not check second condition if first is false
           // Bitwise AND(&) checks both conditions even if first condition is true
           // or false
           System.out.println(a & b);                     // 0
   }
}
```

2. Implement a program to calculate the Simple Interest by using the formula given below and display the calculated Simple Interest.

Simple Interest = (principal*rate of interest*time)/100

```
class SimpleInterest {

   public static void main(String[] args) {
         // Implement your code here
         double principal = 3250d, roi = 7d, time = 3d;
         double si = (principal * roi * time) / 100;
         System.out.println(si);
   }
}
```