OPERATORS:

Operators are keywords used to perform some operations on the java code.

1. **Arithmetic Operators:** They are used to perform simple arithmetic operations on primitive data types.

   * : Multiplication

   / : Division

   % : Modulo

   + : Addition

   – : Subtraction

2**. Unary Operators:** Unary operators need only one operand. They are used to increment, decrement or negate a value.

- ++ : Increment operator, used for incrementing the value by 1. There are two varieties of increment operators.
  - Post-Increment: Value is first used for computing the result and then incremented.
  - Pre-Increment: Value is incremented first, and then the result is computed.
- -- : Decrement operator, used for decrementing the value by 1. There are two varieties of decrement operators.
  - Post-decrement: Value is first used for computing the result and then decremented.
  - Pre-Decrement: Value is decremented first, and then the result is computed.
- ! : Logical not operator, used for inverting a boolean value.

3. **Assignment Operator:** '=' Assignment operator is used to assigning a value to any variable. It has a right to left associativity, i.e. value given on the right-hand side of the operator is assigned to the variable on the left, and therefore right-hand side value must be declared before using it or should be a constant.

The general format of the assignment operator is:

variable = value;

In many cases, the assignment operator can be combined with other operators to build a shorter version of the statement called a Compound Statement. For example, instead of a = a+5, we can write a += 5.

+=, for adding left operand with right operand and then assigning it to the variable on the left.

-=, for subtracting right operand from left operand and then assigning it to the variable on the left.

*=, for multiplying left operand with right operand and then assigning it to the variable on the left.

/=, for dividing left operand by right operand and then assigning it to the variable on the left.

%=, for assigning modulo of left operand by right operand and then assigning it to the variable on the left.

4**. Relational Operators**: These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements.

Some of the relational operators are-

==, Equal to returns true if the left-hand side is equal to the right-hand side.

!=, Not Equal to returns true if the left-hand side is not equal to the right-hand side.

<, less than: returns true if the left-hand side is less than the right-hand side.

<=, less than or equal to returns true if the left-hand side is less than or equal to the right-hand side.

>, Greater than: returns true if the left-hand side is greater than the right-hand side.

>=, Greater than or equal to returns true if the left-hand side is greater than or equal to the right-hand side.

5. **Logical Operators:** These operators are used to perform “logical AND” and “logical OR” operations

&&, Logical AND: returns true when both conditions are true.

||, Logical OR: returns true if at least one condition is true.

!, Logical NOT: returns true when a condition is false and vice-versa

6. **Ternary operator:** Ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name ternary.

The general format is:

condition ? if true : if false

The above statement means that if the condition evaluates to true, then execute the statements after the ‘?’ else execute the statements after the ‘:’.

7. **Bitwise Operators:** This operator will perform the operation on the bits of a number. The various bit wise operators are ~ & | ^ << >> >>>

Negation Operator(~)(tilde): This operator will convert the bits from $0^{'s}$ to $1^{'s}$ and $1^{'s}$ to $0^{'s}$

x = 5	0	0	0	0	0	1	0	1
~x	1	1	1	1	1	0	1	0

If the first bit is '1' then it represents a negative number. The value of the negative number is calculated by performing 2’s compliment.

2’s compliment = 1’s compliment + 1

```
1   1   1   1   1   0   1   0

0   0   0   0   0   1   0   1
                            1
------------------------------
0   0   0   0   0   1   1   0      =   -6
------------------------------
```

Note: ~x = x + 1

Bitwise AND Operator(&): This operator will perform AND operation on the bits of a number.

0	0	0
0	1	0
1	1	0
1	1	1

```
x=5   0   0   0   0   0   1   0   1
y=6   0   0   0   0   0   1   1   0
------------------------------------
x&y   0   0   0   0   0   1   0   0   =  4
```

Bitwise OR Operator(|): This operator will perform OR operation on the bits of a number.

0	0	0
0	1	1
1	1	1
1	1	1

```
x=5   0   0   0   0   0   1   0   1
y=6   0   0   0   0   0   1   1   0
```

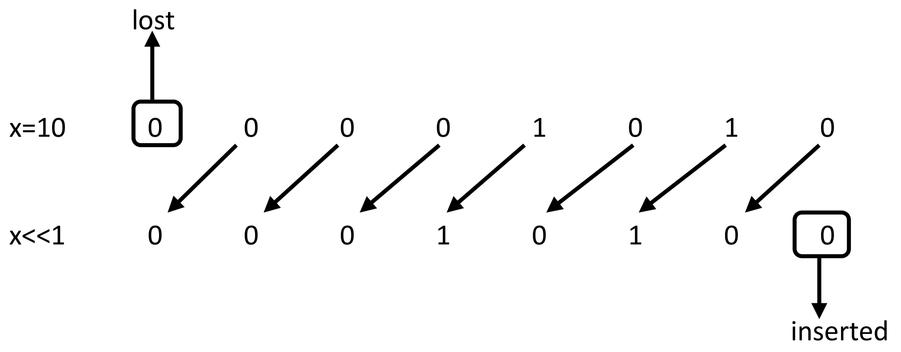
```
x|y   0   0   0   0   0   1   1   1  =  7
```

Bitwise X-OR Operator(^): This operator will perform X-OR operation on the bits of a number.

0	0	0
0	1	1
1	1	1
1	1	0

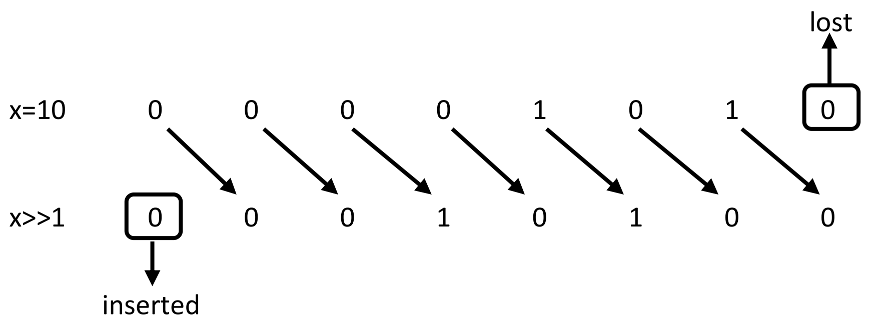
```
x=5   0   0   0   0   0   1   0   1
y=6   0   0   0   0   0   1   1   0
-------------------------------------
x^y   0   0   0   0   0   0   1   1  =  3
```

Left Shift Operator(<<): This operator will shift the bits of a number towards left side by the specified number of positions.



Shifting the bits of a number towards left side by one position is equivalent to multiplying a number by 2.

Right Shift Operator(>>): This operator will shift the bits of a number towards right side by the specified number of positions.



Shifting the bits of a number towards right side by one position is equivalent to dividing a number by 2.

Unsigned Right Shift Operator(>>>): This operator will shift the bits of a number towards right side by the specified number of positions.

Difference between >> & >>>:
When we shift the bits of a number towards right side by using >> operator, the sign bit of the input number will be copied as it is into the resultant i.e. positive input will lead to positive output and negative input will lead to negative output.

When we shift the bits of a number towards right side by using >>> operator, the sign bit will be never copied into the resultant. We will always insert a bit 0 into the sign bit of the result. i.e. the result will be always positive, whether the input is positive or negative.

Rule: Bitwise operators can be applied to any combination of numbers without decimal point(byte, short, int, long, char).

Note**:** The & | ^ operators can be applied to any combination of boolean type or any combination of numbers without decimal point.