OPERATORS IN JAVA

* What are operators?
- This indicates what action to perform

* What is operand?
- This indicates the items, on which action has to apply on.

* What is expression?
- It consists of 1 or more operand and 0 or more operators.

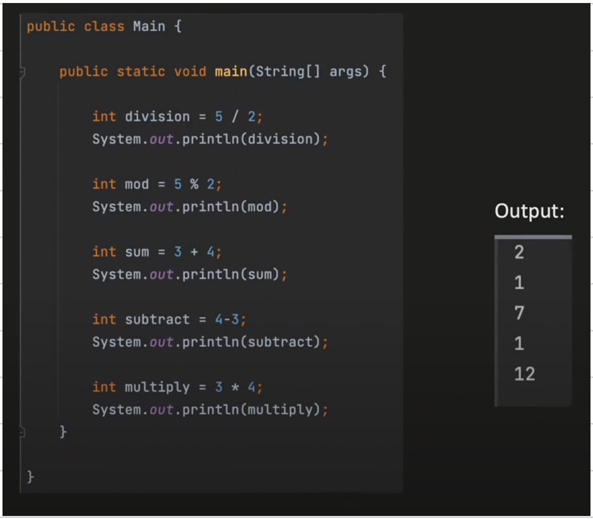
Eg: 5 + 3
- 5 & 3 are operands
- + is operator
- 5+3 as a whole is kla expression

⇒ Categories Of Operators
There are 7 categories of operators

1) Arithmetic Operators
- / (Division)
- − (Subtraction)
- + (Addition)
- % (Modulus)
- * (Multiplication)

We already have information from mathematics of how arithmetic operators work.

For eg:-



```
public class Main {

    public static void main(String[] args) {

        int division = 5 / 2;
        System.out.println(division);

        int mod = 5 % 2;
        System.out.println(mod);

        int sum = 3 + 4;
        System.out.println(sum);

        int subtract = 4-3;
        System.out.println(subtract);

        int multiply = 3 * 4;
        System.out.println(multiply);
    }

}
```

Output:

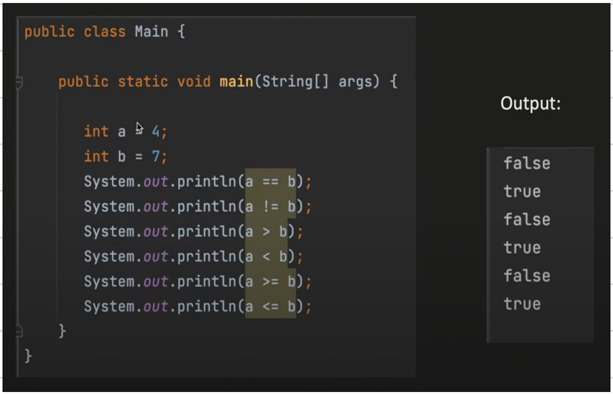
```
2
1
7
1
12
```

2) Relational Operators

– These compares two operand relation and return true or false .

- == (Equals to)
- != (Not equals to)
- > (Greater than)
- < (Less than)
- >= (Greater than or equals to)
- <= (Less than or equals to)

For Eg:-



```
public class Main {

    public static void main(String[] args) {

        int a = 4;
        int b = 7;
        System.out.println(a == b);
        System.out.println(a != b);
        System.out.println(a > b);
        System.out.println(a < b);
        System.out.println(a >= b);
        System.out.println(a <= b);
    }
}
```

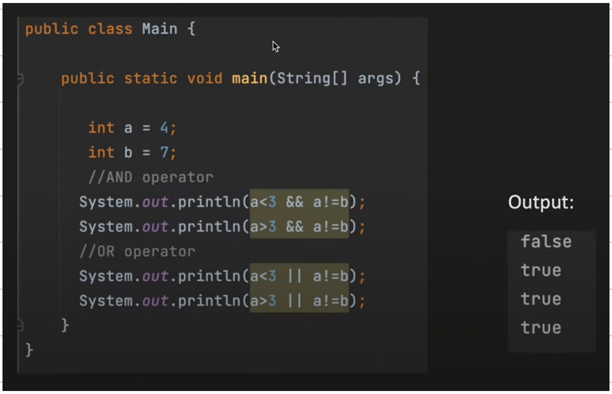
Output:

```
false
true
false
true
false
true
```

3.) Logical Operators

- These combines two or more conditions and return true or false.
  - && (Logical AND) — Returns true only if all conditions are true.
  - || (Logical OR) - Returns true if atleast one condition is true

For Eg:-



```
public class Main {

    public static void main(String[] args) {

        int a = 4;
        int b = 7;
        //AND operator
       System.out.println(a<3 && a!=b);
       System.out.println(a>3 && a!=b);
       //OR operator
       System.out.println(a<3 || a!=b);
       System.out.println(a>3 || a!=b);
    }
}
```

Output:

```
false
true
true
true
```

* As seen if one condition is false, && will not even evaluate further conditions.

4.) Unary Operators

- These require only a single operand
  - ++ (Increment)
  - -- (Decrement)
  - − (Unary Minus)
  - + (Unary Plus)
  - ! (Logical NOT)

* ++ & -- can be used before or after operand.

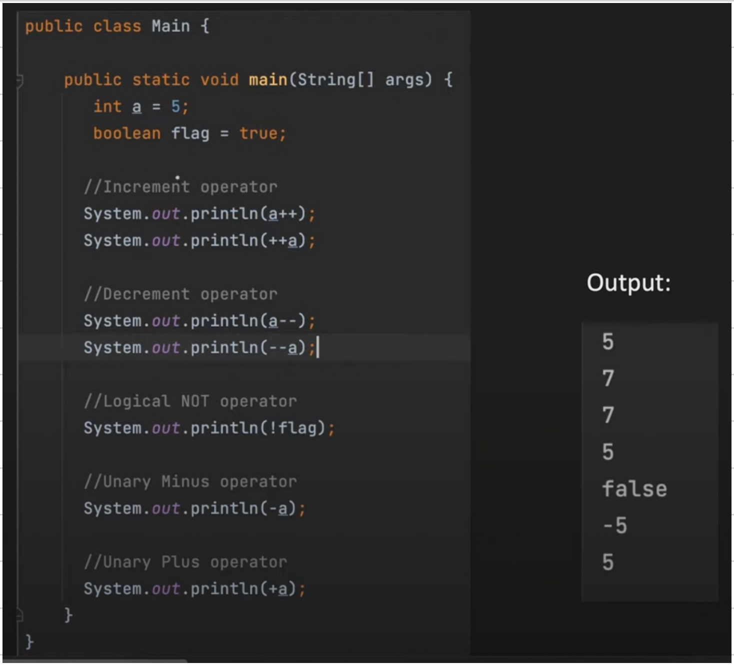
So Operand ++ ⇒ Postfix Increment , ++ operand ⇒ Prefix Increment
Operand -- ⇒ Postfix Decrement , -- operand ⇒ Prefix Decrement

- Postfix will return whatever value is first & then increments / decrements the value
- Prefix will first increment / decrement the value & then returns the value.

- Logical NOT operator will reverse the value i.e if current value is true, it'll change it to false & if the current value is false, it'll change the value to true.

- Uniry + and - makes the value positive or negative.

For Eg :-



```
public class Main {

    public static void main(String[] args) {
        int a = 5;
        boolean flag = true;

       //Increment operator
       System.out.println(a++);
       System.out.println(++a);

       //Decrement operator
       System.out.println(a--);
       System.out.println(--a);

       //Logical NOT operator
       System.out.println(!flag);

       //Unary Minus operator
       System.out.println(-a);

       //Unary Plus operator
       System.out.println(+a);
    }
}
```

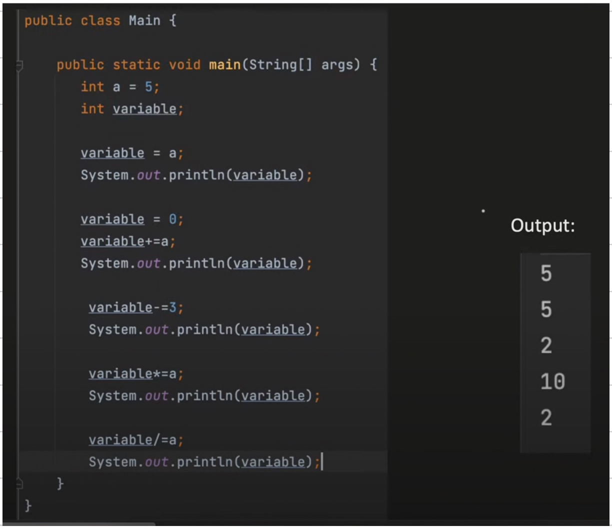
Output:

```
5
7
7
5
false
-5
5
```

5.) Assignment Operators

- These are used to assign new value to the variable.
  - =
  - +=
  - -=
  - *=
  - /=
  - %=
- Assignment Operator assigns a value (on right side of operator) to a variable (on left side of the operator)

For Eg:-



```
public class Main {

    public static void main(String[] args) {
        int a = 5;
        int variable;

        variable = a;
        System.out.println(variable);

        variable = 0;
        variable+=a;
        System.out.println(variable);

         variable-=3;
         System.out.println(variable);

         variable*=a;
         System.out.println(variable);

         variable/=a;
         System.out.println(variable);
    }
}
```

Output:

```
5
5
2
10
2
```

6) Bitwise Operators

- These works on bits i.e 1 and 0 and are very fast.
  - & (Bitwise AND)
  - | (Bitwise OR)
  - ^ (Bitwise XOR)
  - ~ (Bitwise NOT)

    (It can come under Unary too)

* Bitwise AND (&)

a	b	
0	0	⇒ 0
0	1	⇒ 0
1	0	⇒ 0
1	1	⇒ 1

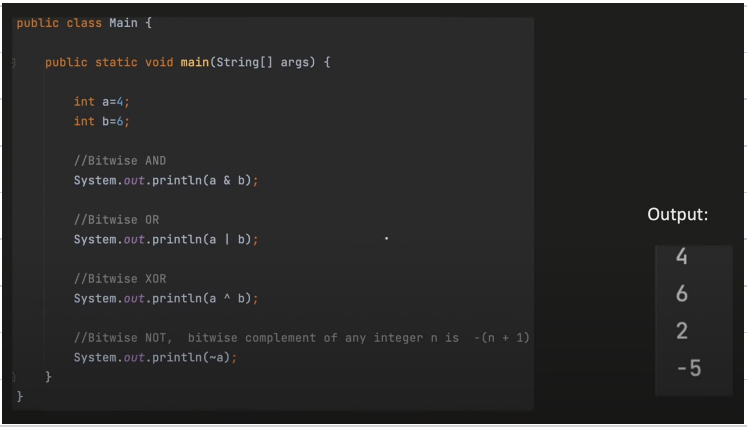
* Bitwise OR (|)

a	b	
0	0	⇒ 0
0	1	⇒ 1
1	0	⇒ 1
1	1	⇒ 1

* Bitwise XOR (^)

a	b	
0	0	⇒ 0
0	1	⇒ 1
1	0	⇒ 1
1	1	⇒ 0

For Eg:-



```
public class Main {

    public static void main(String[] args) {

        int a=4;
        int b=6;

        //Bitwise AND
        System.out.println(a & b);

        //Bitwise OR
        System.out.println(a | b);

        //Bitwise XOR
        System.out.println(a ^ b);

        //Bitwise NOT,  bitwise complement of any integer n is  -(n + 1)
        System.out.println(~a);
    }
}
```

Output:

```
4
6
2
-5
```

* How does bitwise NOT work?

— It basically reverses the bit

So 0 ⇒ 1

1 ⇒ 0

Now how does it works on numbers.

Let's compute ~4 (NOT of 4)

As we know in Java numbers are signed i.e Most Significant Bit (MSB) tells the sign of the number

So binary of 4 will be

2^3	2^2	2^1	2^0
0	1	0	0

(MSB) – Since MSB is 0 so it is positive.

Now NOT will reverse all the bits, so binary of ~4 will be

	2^3	2^2	2^1	2^0
(-)	1	0	1	1
	(MSB)			

Since MSB is 1 ie result is -ve, so the value of MSB will be negative
Now converting it to decimal gives us

$-8 + 0 + 2 + 1$

i.e -5

So 0100 ---> ~0100 = 1011

We can calculate it directly using the formula $-(N+1)$. So in our case it'll be $-(4+1)$ ie $\underline{-5}$.

* How can we confirm if -5 is 1011

- To get -5, we know that we have to find its 2nd Complement

So for 4 ie 0101 ⇒

1st Complement = 1010

2nd Complement = 1st complement + 1

i.e 1010 + 1 ⇒ 1011

So bitwise NOT, bitwise complement of any integer n is $-(n+1)$

7) Bitwise Shift Operators

→ These are used to shift the bits of a number left or right.

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

- There is no unsigned left shift as << and <<< are equal.

* >> : It's signed right shift

- It fills the most significant bit with the sign of the number

eg:- 1) 11000110's >> will be

$1\underbrace{1100011}_{\text{Shifted right}}$

MSB added as same sign of the original number

2) 01000110's >> will be

$0\underbrace{0100011}_{\text{Shifted right}}$

MSB added as same sign of the original number

* >>> : It's unsigned right shift

- It fills the MSB with the 0;

eg: 1) 11000110's >>> will be

$0\underbrace{1100011}_{\text{Shifted right}}$

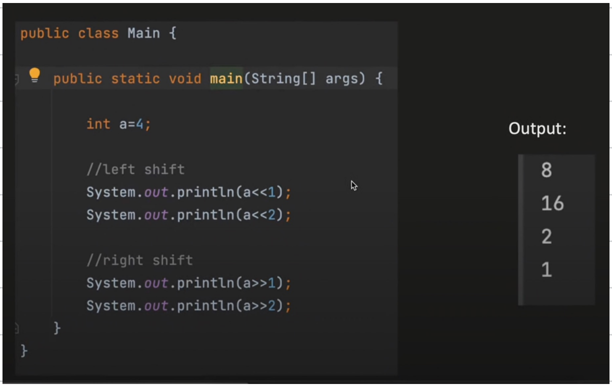
MSB filled with 0

2) 01000110's >>> will be

$0\underbrace{0100011}_{\text{Shifted right}}$

MSB added as 0

Demo for both left & right shift :-



```java
public class Main {

    public static void main(String[] args) {

        int a=4;

        //left shift
        System.out.println(a<<1);
        System.out.println(a<<2);

        //right shift
        System.out.println(a>>1);
        System.out.println(a>>2);
    }
}
```

Output:

```
8
16
2
1
```

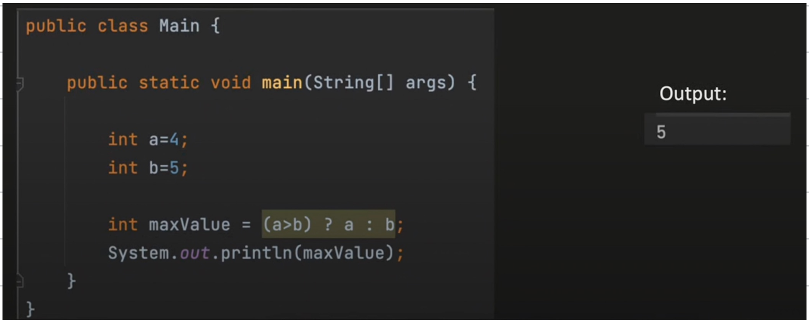
- * For Left Shift LSB will always filled with 0.
- * Left Shift once doubles the number
- * Right Shift once halves the number
- * There is no sense of <<< as LSB don't have any value

8) Ternary Operators

- It mimics the if else condition
- So, it evaluates the condition, it'll execute first expression otherwise it'll execute second expression.

- (Condition) ? Expression 1 : Expression 2

For Eg:-



```java
public class Main {

    public static void main(String[] args) {

        int a=4;
        int b=5;

        int maxValue = (a>b) ? a : b;
        System.out.println(maxValue);
    }
}
```

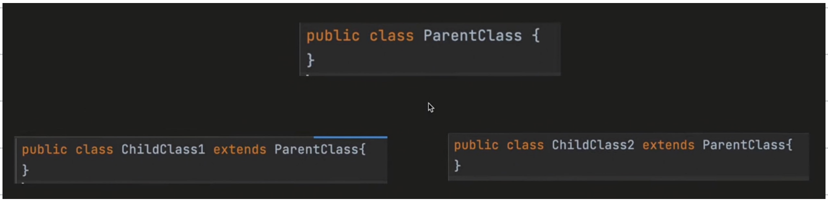
Output:

```
5
```

9) Type Comparison Operator

- It is used to do the type check, whether particular object is of a certain class or not.
- instance Of

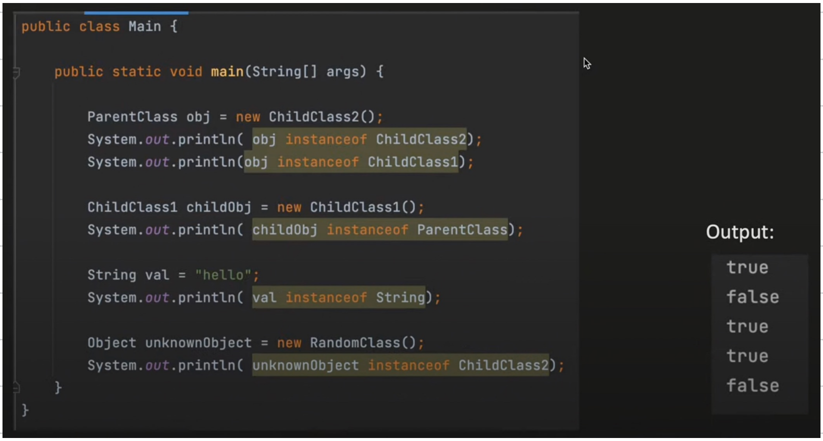
For Eg:-



```java
public class ParentClass {
}
```

```java
public class ChildClass1 extends ParentClass{
}
```

```java
public class ChildClass2 extends ParentClass{
}
```



```java
public class Main {

    public static void main(String[] args) {

        ParentClass obj = new ChildClass2();
        System.out.println( obj instanceof ChildClass2);
        System.out.println(obj instanceof ChildClass1);

        ChildClass1 childObj = new ChildClass1();
        System.out.println( childObj instanceof ParentClass);

        String val = "hello";
        System.out.println( val instanceof String);

        Object unknownObject = new RandomClass();
        System.out.println( unknownObject instanceof ChildClass2);
    }
}
```

Output:

```
true
false
true
true
false
```

* It'll return true when we perform the type check of an object of child class with its parent class.

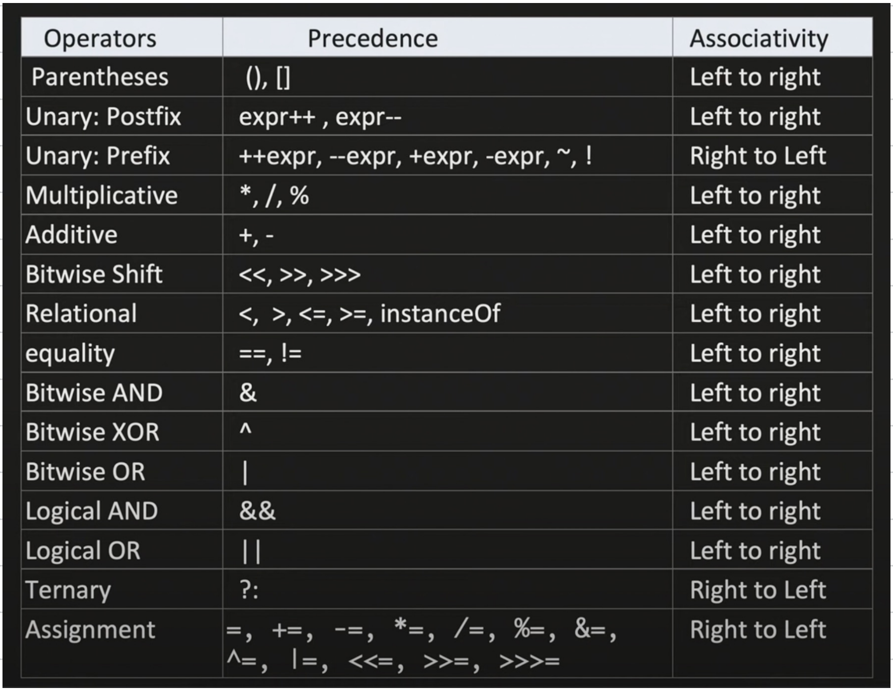
=> Operator Precedence :-

- Associativity: If 2 operators have the same precedence, then it is evaluated based on its associativity (Left to Right or Right to Left).

High

Priority

Low



Operators	Precedence	Associativity
Parentheses	(), []	Left to right
Unary: Postfix	expr++ , expr--	Left to right
Unary: Prefix	++expr, --expr, +expr, -expr, ~, !	Right to Left
Multiplicative	*, /, %	Left to right
Additive	+, -	Left to right
Bitwise Shift	<<, >>, >>>	Left to right
Relational	<, >, <=, >=, instanceOf	Left to right
equality	==, !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Ternary	?:	Right to Left
Assignment	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	Right to Left

* Let's solve an example expression

Int a = 4

Solve a = a + a++ + ++a * --a + a--

Now replace operands with values

a = ~~4~~ ~~5~~ ~~6~~ ~~5~~ 4

∴ a = 4 + 4 + 6 * 5 + 5

a = 4 + 4 + 30 + 5

a = 39