Lesson:

Java Operators

PW SKILLS

Pre-Requisites:

- Java Basic syntax for input and output
- Data types
- Variables
- Identifiers
- Keywords

List of Concepts Involved:

- Operators
- Operators precedence and associativity.

Topic 1: Java Operators

Operators are the symbols that are used to perform pre-defined operations on variables and values (commonly referred to as operands). As soon as the compiler encounters an operator, it performs the specific mathematical or logical operation and returns the result.

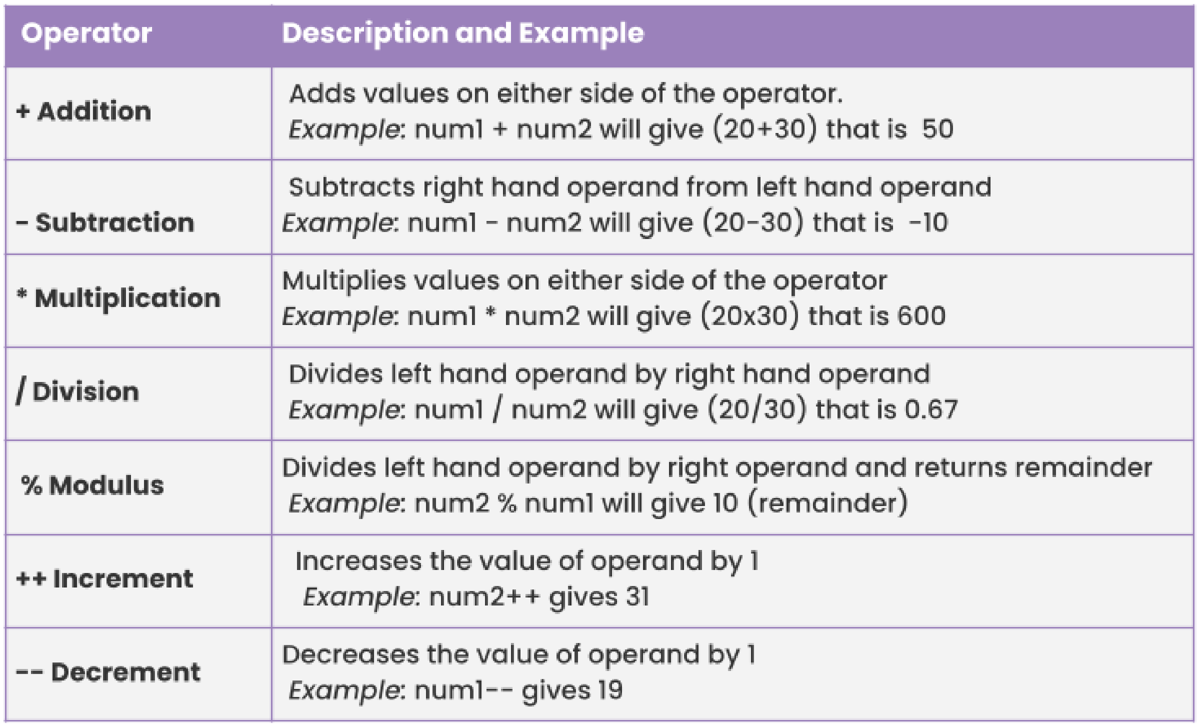
Let us learn and understand about the relevant operators in detail.

Operators in Java can be classified into 6 types:

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

1.Java Arithmetic operators:

Arithmetic operators are used in mathematical expressions in a program. They function in the same way as they do in algebra.
The following table lists the arithmetic operators:
Assume integer variable *num1* holds a value of 20, variable *num2* holds a value of 30 and *num3* holds a value of 30, then:



Operator	Description and Example
+ Addition	Adds values on either side of the operator. *Example*: num1 + num2 will give (20+30) that is 50
- Subtraction	Subtracts right hand operand from left hand operand *Example*: num1 - num2 will give (20-30) that is -10
*** Multiplication**	Multiplies values on either side of the operator *Example*: num1 * num2 will give (20x30) that is 600
/ Division	Divides left hand operand by right hand operand *Example*: num1 / num2 will give (20/30) that is 0.67
% Modulus	Divides left hand operand by right operand and returns remainder *Example*: num2 % num1 will give 10 (remainder)
++ Increment	Increases the value of operand by 1 *Example*: num2++ gives 31
-- Decrement	Decreases the value of operand by 1 *Example*: num1-- gives 19

Let us now write a simple program to implement all the operators.You may try it yourself too !

Example:

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

     // declare variables p and q
    int p = 20, q = 10;
    int result;

     // addition operator
    result=p+q;
    System.out.println(result);

     // subtraction operator
    System.out.println(p - q);
     // we can directly perform subtraction in print statement,no need
     // to use result variable here
     // multiplication operator
    System.out.println(p * q);

     // division operator
    System.out.println(p / q);

     // modulo operator
    System.out.println(p % q);
  }
}
```

Output

```
30
10
200
2
0
```

2.Java Relational Operators:

Relational operators compare numeric, character string, or logical data. The result of the comparison, either true (1) or false (0), can be used to make a decision regarding program flow. The table below enlists relational operators used in Java.

Operator	Description	Example
==	Is Equal To	4 == 5 returns false
!=	Not Equal To	1!= 5 returns true
>	Greater Than	1 > 5 returns false
<	Less Than	2 < 5 returns true
>=	Greater Than or Equal To	4 >= 5 returns false
<=	Less Than or Equal To	1 <= 5 returns true

Let us write a program to implement all of these.

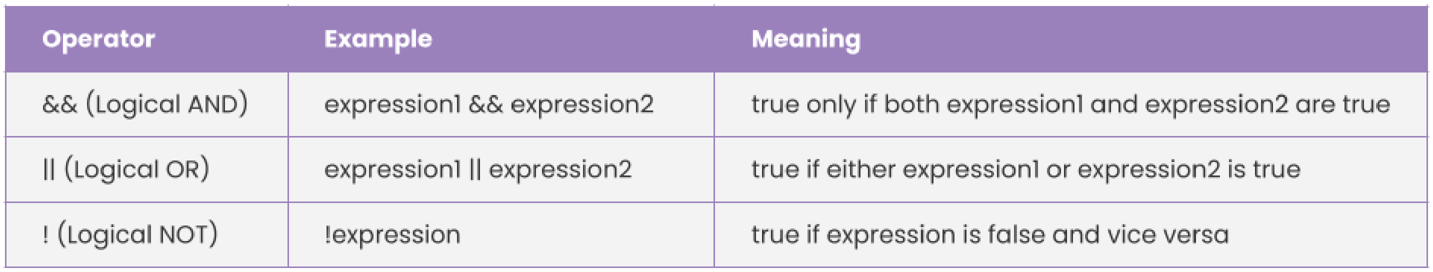
```
class Main {

 public static void main(String[] args) {
       // create variables
       int p = 10, q = 15;
      // == operator
    System.out.println(p == q);  // false
      // != operator
    System.out.println(p != q);  // true
      // > operator
    System.out.println(p > q);  // false
      // < operator
    System.out.println(p < q);  // true
      // >= operator
    System.out.println(p >= q);  // false
      // <= operator
    System.out.println(p <= q);  // true
  }
}
```

Output

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

3.Java Logical Operators

Logical operators are used for decision making. This class of operators is used to check whether an expression is true or false. Some of the commonly used logical operators are mentioned in the table below.



Operator	Example	Meaning
&& (Logical AND)	expression1 && expression2	true only if both expression1 and expression2 are true
|| (Logical OR)	expression1 || expression2	true if either expression1 or expression2 is true
! (Logical NOT)	!expression	true if expression is false and vice versa

Let us look at the following program to understand it better.

Example code:

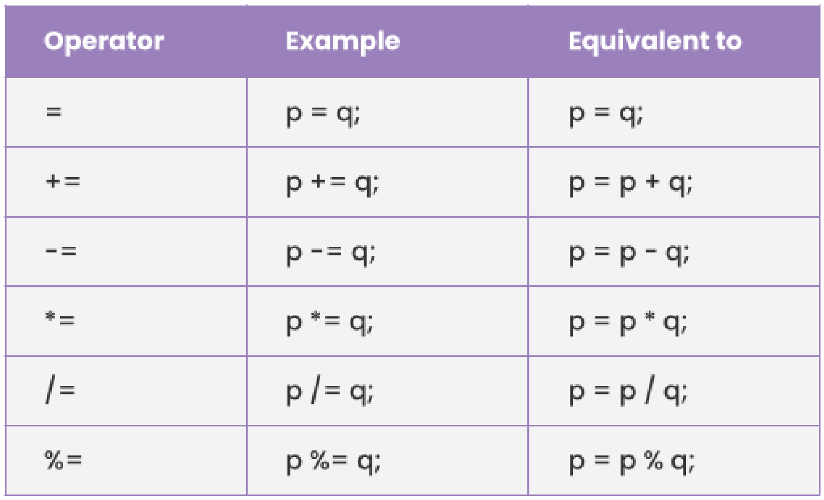
```java
class Main {
  public static void main(String[] args) {

    // && operator
    int p=15,q=10,r=5;
    System.out.println((p > q) && (p > r));  // true
    System.out.println((p > q) && (p < r));  // false

    // || operator
    System.out.println((r < q) || (p < q));  // true
    System.out.println((p > q) || (q > r));  // true
    System.out.println((p < q) || (p < r));  // false

    // ! operator
    System.out.println(!(p == q));  // true
    System.out.println(!(p > q));  // false
  }
}
```

Output:

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

4.Java assignment operators:

The assignment operator = assigns the value of its right-hand operand to a variable, a property, or an indexer element given by its left-hand operand. Confused? Don't worry, we have you covered here.
Let's have a look at some of the commonly used assignment operators available in Java with examples.



Operator	Example	Equivalent to
=	p = q;	p = q;
+=	p += q;	p = p + q;
-=	p -= q;	p = p - q;
*=	p *= q;	p = p * q;
/=	p /= q;	p = p / q;
%=	p %= q;	p = p % q;

Try out the following example for better understanding.

Example code:

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

    int p = 10;
    int q;

    // assign value using =
    q = p;
    System.out.println(q);  // value of q is 10

    // assign value using =+
    q += p;
    System.out.println(q);  // value of q is 20

    // assign value using =*
    q *= p;
    System.out.println(q); // value of q is 200
  }
}
```

Output:

```
10
20
200
```

Question: Is there any performance difference between x+=y and x=x+y?
Solution: Yes, x+=y performs better than x=x+y.
Eg. x += 10 means

- Find the place identified by x
- Add 10 to it

And x = x + 10 means:

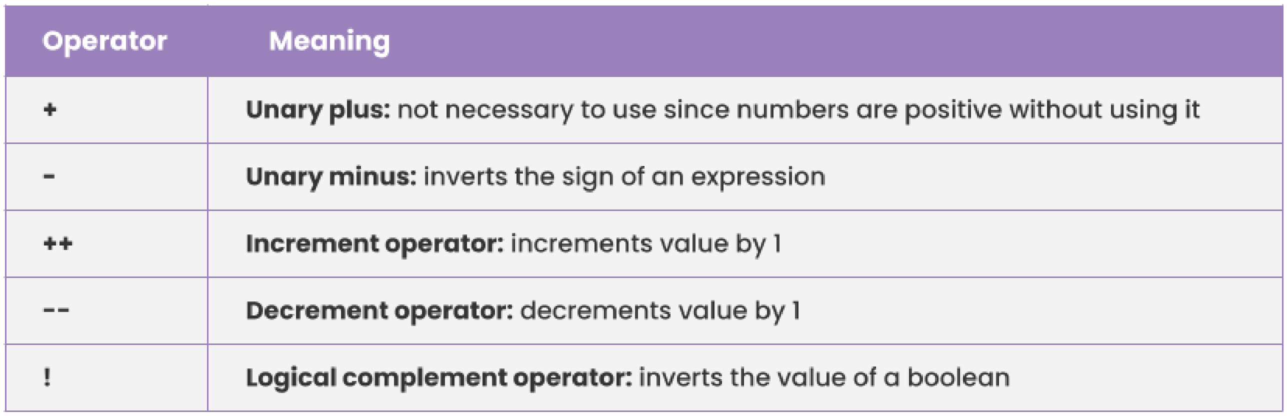
- Evaluate x+10
- Find the place(memory) identified by the variable x
- Copy x into an accumulator (accumulator is a part of CPU for temporary storage)
- Add 10 to the accumulator
- Store the result in variable x
- Find the place (memory) identified by x
- Copy the accumulator(result) to it

Hence, clearly x=+10 is better than x=x+10 since evaluation is direct with no intermediate steps for CPU to perform.

Let us look at the next class of operators.

5.Java Unary Operators:

Interestingly, unlike the normal operators seen so far, java unary operators need only one operand to perform operations like increment, decrement, negation, etc.
The table below will help you understand it in a better way.



Operator	Meaning
+	**Unary plus:** not necessary to use since numbers are positive without using it
-	**Unary minus:** inverts the sign of an expression
++	**Increment operator:** increments value by 1
--	**Decrement operator:** decrements value by 1
!	**Logical complement operator:** inverts the value of a boolean

For example:

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

    // initialize p
    int p = 5;
    System.out.println("Post-Increment Operator");
    System.out.println(p++); // 5
    // / p's value is incremented to 6 after returning current value i.e; 5
```

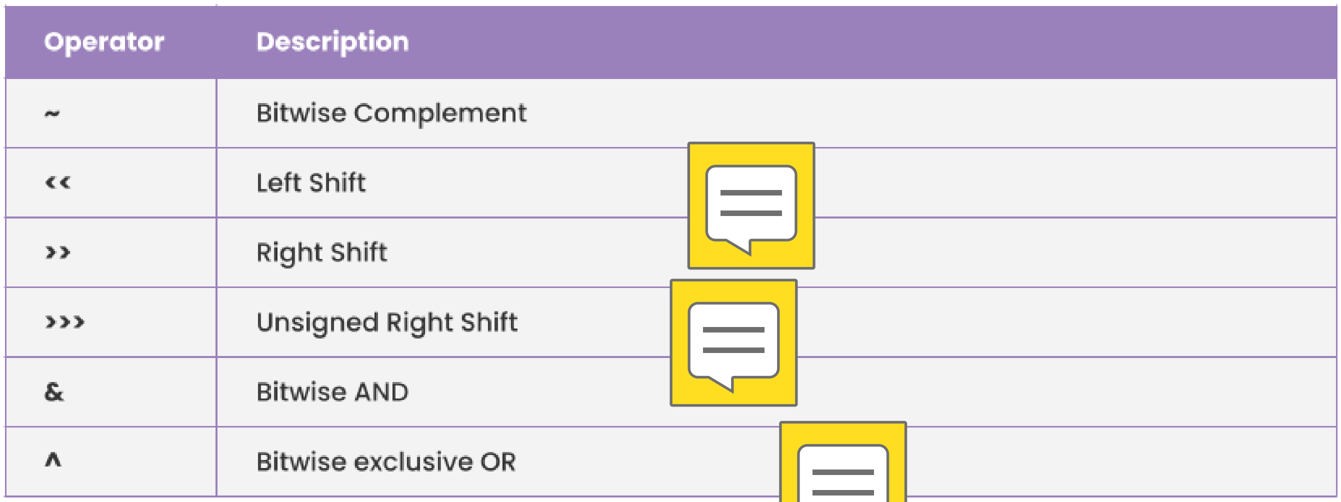
```
      // initialized to 5
      int q = 5;
      System.out.println("Pre-Increment Operator");
      System.out.println(++q);   //6
     // q is incremented to 6 and then it's value is returned
  }
}
```

Output:

```
Post-Increment Operator
5
Pre-Increment Operator
6
```

6.Java Bitwise operators:

Interestingly, unlike the normal operators seen so far, java unary operators need only one operand to perform operations like increment, decrement, negation, etc.
The table below will help you understand it in a better way.



Operator	Description
~	Bitwise Complement
<<	Left Shift
>>	Right Shift
>>>	Unsigned Right Shift
&	Bitwise AND
^	Bitwise exclusive OR

Let's look at a simple code to understand the working of these operators.

Example code:

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

    // initialize p
        int p = 5;
        System.out.println(p<<2);
// Shifting the value of p towards the left two positions
        }
      }
```

Output 20
Explanation:
Shifting the value of p towards the left (two positions) will make the leftmost 2 bits to be lost. The value of p is 5. The binary representation of 5 is 00000101(we will learn about this conversion in the forthcoming lecture).
After 2 left shifts, binary representation of p will be 00010100 which is equivalent to 20.

Now that we have learnt about all types of operators, its a good time to explore the precedence/priority of each of these wrt each other in different scenarios.

Topic : Java Operator Precedence and Associativity

Operator precedence determines the order/sequence of evaluation of operators in an expression (analogous to BODMAS concept in maths).

Take a look at the statement below:
int ans = 10 - 4 * 2;

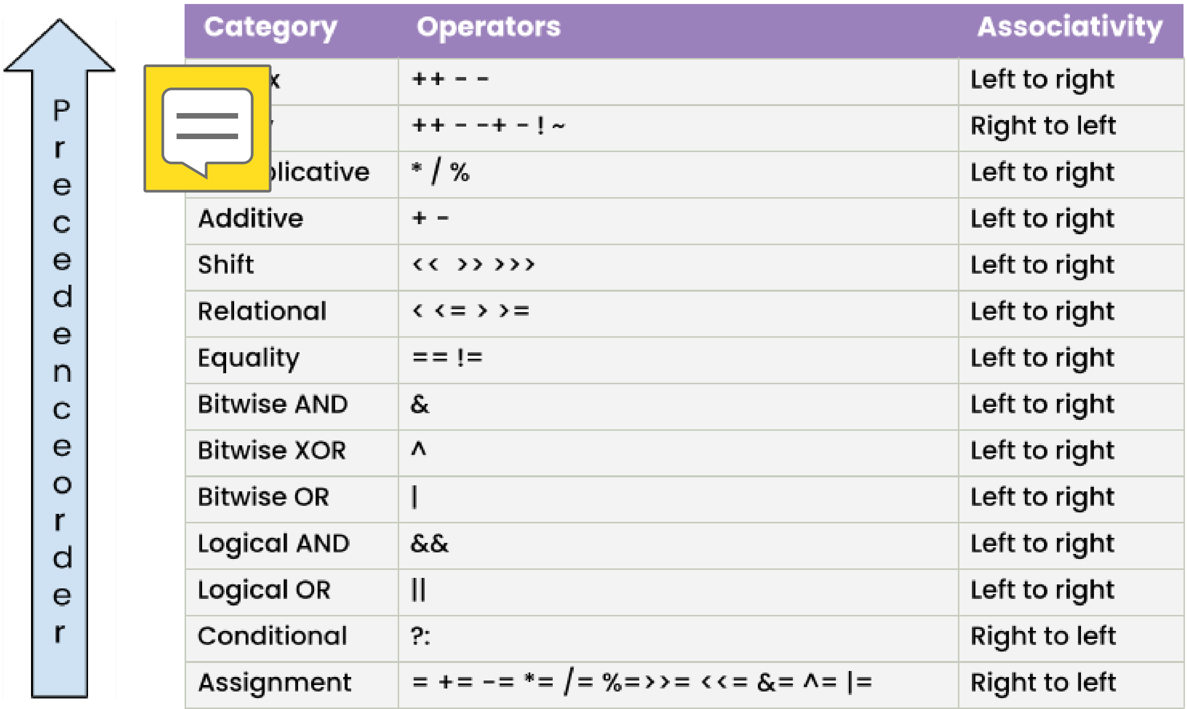
What will be the value of variable ans? Will it be (10 - 4)*2, that is, 12? Or it will be 10 - (4 * 2), that is, 2?

When two operators share a common operand (4 in this case), the operator with the highest precedence is operated upon first. In java, the precedence of * is higher than that of -. Hence, multiplication is performed before subtraction, and hence the final value of the variable ans will be 2.

Associativity of Operators in Java
A question arises here, what if the precedence of all operators in an expression is same? In that case, the concept of associativity comes into picture.

Associativity specifies the order in which operators are evaluated by the compiler, which can be left to right or right to left. For example, in the phrase p = q = r = 10, the assignment operator is used from right to left. It means that the value 10 is assigned to r, then r is assigned to q, and at last, q is assigned to p. This phrase can be parenthesized as (p = (q = (r = 10))).

Operator Precedence in Java (Highest to Lowest)


Precedence order

Category	Operators	Associativity
x	++ - -	Left to right
y	++ - -+ - ! ~	Right to left
plicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >> >>>	Left to right
Relational	< <= > >=	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
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left

Let us try a few questions on precedence and associativity to clear the air of confusion.

Example 1.
Let say we have 4 variables of type int ; p,q,r,s

s = p-++r-++q;
is equivalent to

Answer: s = p-(++r)-(++q);

Explanation: The operator precedence of prefix ++ is higher than that of - subtraction operator.

Example 2. What does the following code fragment print?

```
    System.out.println(4 + 2 + "pqr");
    System.out.println("pqr" + 4 + 2);
```

Answer: 6pqr and pqr42, respectively.

Explanation: The + operator is left-to-right associative, whether it is string concatenation or simple number addition.

Example 3: What is the result of the following code fragment? Explain.

```
boolean p = false;
boolean q = false;
boolean r = true;
System.out.println(p == q == r);
```

Answer: It prints true.

Explanation: The equality operator is left-to-right associative, so p == q evaluates to true and this result is compared to r, which yields true.

This brings us to the end of the lecture !
Keep Learning ! Keep Exploring !!

Upcoming Class Teaser

- Java conditionals
- if/else
- switch