CHAPTER 3: OPERATORS

4.1 What are Operators?

Operators are symbols that are used to perform certain operations and changes to one or more operands. For example, 5+6 here 5 and 6 are operands and “+” is an operator. Operators in Java can be categorized as follows:

- Arithmetic
- Assignment
- Relational
- Logical
- Bitwise
- Unary and more

We shall only be focusing on the above six types of operators in this course.

4.2 Arithmetic Operators

Used to perform arithmetic operations between two operands.

Category	Name	Symbol	Operation	Example
Additive	Addition	+	Adds two numbers	5+6 is 11
	Subtraction	-	Subtracts the second number from the first	5-6 is -1
Multiplicative	Multiplication	*	Multiplies the first number with the second	5*6 is 30
	Division	/	Divides the first number with the second	12/5 is 2
	Modulus	%	Finds the remainder after dividing the first number with the second	12%6 is 0

To perform any of the arithmetic operations both operands must be of the same type. Which means, the operands must be both int, both float or both double.

While dividing an int with another int, the whole part is taken. On the other hand, while dividing a float/double with a float/double, the decimal part is also taken into account. Therefore, float/double datatypes provide more accurate results in some arithmetic operations. For example, 5/2 is 2 whereas 5.0/2.0 is 2.5.

4.3 Assignment operators

Used to assign a changed or unchanged value from the operand on its right to the operand on its left.

Name	Symbol	Identicality	Operation
Assignment	=	x= 5;	Assigns the value of the operand on its right to its left

Addition Equals	+=	x+= 5;
		x= x+5;
			Adds both of the operands on its right and left and assigns the addition to the operand on its left

Subtraction Equals	-=	x-= 2;
		x= x-2;
			Subtracts the operand on its right from the operand on its left and assigns the subtraction to the operand on its left

Multiplication Equals	*=	x*= 2;
		x= x*2;
			Multiplies both of the operands and assigns the multiplication to the operand on its left

Division Equals	/=	x/= 2;
		x= x/2;
			Divides the operand on its left with the operand on its right and assigns the subtraction to the operand on its left

Modulus Equals	%=	x%= 2;
		x= x%2;
			Assigns the remainder to the operator on its left after performing modulus operation on the operator on its left with the operator on its right.

4.4 Relational Operators

Used to compare between two operands. Always results in either true or false.

Category	Name	Symbol	Examples
Comparison	Greater Than	>	System.out.println(5>3); //Output: true
	Less Than	<	System.out.println(5<3); //Output: false
	Greater Than or Equals	>=	System.out.println(5>=5); //Output: true
	Less Than or Equals	<=	System.out.println(5<=3); //Output: false
Equality	Equals	==	System.out.println(5==3); //Output: false
	Not Equals	!=	System.out.println(5!=3); //Output: true

4.5 Logical Operators

Used to determine the logic between two operands. Always results in either true or false.

Name	Symbol	Operation	Examples
Logical AND	&&	Denotes true if both operands before and after are true, otherwise denotes false.	System.out.println(5>3 && 2<3); //Output: true
Logical OR	| |	Denotes true if at least one of the operands before or after is true, otherwise denotes false.	System.out.println(5<3 | | 2!=2); //Output: false
Logical NOT	!	Reverses the logical value of the operand on its right.	System.out.println(!(2==3)); //Output: true

4.6 Bitwise Operators

Bitwise operators are similar to logical operators but can be used on both int and Boolean datatypes (except bitwise complement). If the operands are int, the output is also int; if the operands are Boolean, the output is also Boolean.

Moreover, the bitwise operator checks both the operands regardless of the value of the first operand. On the other hand, logical AND (&&) does not check the value of the second operand if the first operand is false. Logical OR (||) does not check the value of the second operand if the first operand is true. Bitwise Exclusive OR provides true if one of the operands is true and the other one is false, otherwise, it provides false. Bitwise Complement only works for Boolean and is also a unary operator as it needs only one operand.

Name	Symbol	Operation	Examples

Bitwise AND	&	Performs binary AND	System.out.println(8 & 7); //Output: 0
			System.out.println(false & true); //Output: false

Bitwise Exclusive OR	^	Performs binary XOR	System.out.println(8 ^ 7); //Output: 15
			System.out.println(true ^ false); //Output: true

Bitwise Inclusive OR	|	Performs binary OR	System.out.println(9 | 7); //Output: 15
			System.out.println(false | true); //Output: false

Bitwise Complement	~	Returns the two's complement representation of the input value	System.out.println(~5); //Output: -6

4.7 Unary Operators

Unary operators only need one operand. These types of operators are mainly used for counting purposes. There are two variations of unary operators:

- Prefix: The value is changed first and then assigned
- Postfix: the value is assigned first and then changed

Type	Name	Symbol	Operation	Example

Prefix	Negation	-	Negates an operand	x=5;
				y= -x;
				Here y is -5

	Not	!	Reverses the logical value of an operand	x= !true; Here x is false

	Increment	++	Increments the operand by 1	x= 5;
				y= ++x;
				Here x and y both are 6

	Decrement	--	Decrements the operand by 1	x= 5;
				y= --x;
				Here x and y both are 4

Postfix	Increment	++	Increments the operand by 1	x= 5;
				y= x++;
				Here x is 6 but y is 5

	Decrement	--	Decrements the operand by 1	x= 5;
				y= x--;
				Here x is 4 but y is 5

4.8 Operator Precedence

The precedence is higher at the top and decreases as we move towards the bottom. If multiple operators with the same precedence are placed adjacently, operators are executed from left to right. If the first brackets "( )" are present, we must go to the innermost first bracket and gradually move towards the outer ones.

Operator Type	Operator Category/Name	Symbol
Unary	Postfix	++, --
	Prefix	++, --, !, -

Arithmetic	Multiplicative	*, / , %
	Additive	+, -
Relational	Comparison	<, >, <=, >=
	Equality	==, !=
Bitwise	Bitwise AND	&
	Bitwise Exclusive OR	^
	Bitwise Inclusive OR	|
Logical	Logical AND	&&
	Logical OR	| |
Assignment	All assignment operators	=, +=, -=, *=, /=, %=

Now, let us find the output of the following Java code lines:

```
System.out.println(5*(5+5/5%(5*5)));          //Output: 30; Here 5*5 was executed first, then 5/5.
System.out.println(5+1==(5+5/5%(5*5)));               //Output: true.
System.out.println(-(-5-1)==(5+5/5%(5*5)));   //Output: true; Here -5-1 and 5*5 were executed first.
System.out.println(5!=5 && 5%1!=1 || 1==(5/2));        //Output: true.
```

From the above lines of codes, we can see that if operators with same precedence are placed adjacently such as: 5/5*5, we must calculate from left to right. Which means performing the division first and then the multiplication.

4.9 WORKSHEET

A. Trace the following lines of codes and write the outputs.

Code	Output

class A {
   public static void main(String[] args) {
      int x= 5; //Value of x is 5
      int y= 7; //Value of y is 7
      System.out.println(x); //Print current value of x
      System.out.println(y); //Print current value of y
      x-=1;
      y+=1;
      System.out.println(x);
      System.out.println(y);
      x= ++y;
      y= x--;
      System.out.println(x);
      System.out.println(y);
      System.out.println(x==y);
      System.out.println(x%y==4*2);
      System.out.println(x&y);
      System.out.println(x|y);
      x= -x;
      System.out.println(x);
      x= ~x;
      System.out.println(x);
      y*=x;
      System.out.println(y);
      System.out.println(x%2==1 && y%2!=0);
   }
}	

B. Answer the following questions.

- ❖ What is the difference between bitwise operators and logical operators?
- ❖ What is the output of the following lines of codes? Why do the values of x and y differ?

```
int x= 5;
int y= 7;
y= ++x;
x= y--;
System.out.println(x);
System.out.println(y)
```