11. Operators in Java

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are:

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators
6. Ternary Operator
7. Bitwise Operators
8. Shift Operators
9. instance of operator

11.1 Arithmetic Operators

They are used to perform simple arithmetic operations on primitive data types.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from other	x – y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by other	x / y
%	Modulus	Returns the division remainder	x % y

Arithmetic Example 1:

```
public class ArithmeticExample1
{
	public static void main(String[] args)
	{
		System.out.println(2+3);
		System.out.println(2-3);
		System.out.println(4*6);
		System.out.println(33/2);
		System.out.println(33%2);
	}
}
```

Output:

```
5
-1
24
16
1
```

Note: The result is integer when integer is divided by integer.

Arithmetic Example 2:

```
public class ArithmeticExample2
{
	public static void main(String[] args)
	{
		System.out.println(33/2);
	}
}
```

Output:

```
16
```

Arithmetic Example 3:

```
public class ArithmeticExample3
{
	public static void main(String[] args)
	{
		int a=27;
		int b=2;
		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:

```
29
25
54
13
1
```

Arithmetic Example 4:

```
public class ArithmeticExample4
{
	public static void main(String[] args)
	{
		int a=27;
		int b=2;
		double c=a/b;
		System.out.println(c);
		double d=33.0/2;
		System.out.println(d);
		double f=33.0/b;
```

```
            System.out.println(f);
            double g=33/2.0;
            System.out.println(g);
            double h=39.2;
            double i=2.23;
            double j=h/i;
            System.out.println(j);
            System.out.println(h/i);
      }
}
```

Output:

```
13.0
16.5
16.5
16.5
17.57847533632287
17.57847533632287
```

11.2 Unary Operators:

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

Operator	Name	Description
-	Unary minus	Used for negating the values.
+	Unary plus	Indicates the positive value (numbers are positive without this, however). It performs an automatic conversion to int when the type of its operand is the byte, char, or short. This is called unary numeric promotion.
++	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.

Unary Example 1:

```
public class UnaryExample1
{
      public static void main(String[] args)
      {
            int i=1;
            ++i;
            System.out.println(i);
      }
}
```

Output:

```
2
```

Unary Example 2:

```
public class UnaryExample2
{
      public static void main(String[] args)
      {
            int i=1;
            i++;
            System.out.println(i);
      }
}
```

Output:

```
2
```

Unary Example 3:

```
public class UnaryExample3
{
      public static void main(String[] args)
      {
            int i=5;
            --i;
            System.out.println(i);
      }
}
```

Output:

```
4
```

Unary Example 4:

```
public class UnaryExample4
{
      public static void main(String[] args)
      {
```

```
            int i=5;
            i--;
            System.out.println(i);
      }
}
```

Output:

```
4
```

Unary Example 5:

```
public class UnaryExample5
{
      public static void main(String[] args)
      {
            int i=5;
            ++i; // i=6
            int var=i + ++i + ++i + i;
            //  var=6 + 7 + 8 + 8 = 29
            System.out.println(var);
            System.out.println(i);
      }
}
```

Output:

```
29
8
```

Unary Example 6:

```
public class UnaryExample6
{
      public static void main(String[] args)
      {
            int i=5;
            i++;  // i=6
            int var= i + i++ + i++ + i;
            //  var= 6 + 6 + 7 + 8 = 27
            System.out.println(var);
            System.out.println(i);
      }
}
```

Output:

```
27
8
```

Unary Example 7:

```
public class UnaryExample7
{
	public static void main(String[] args)
	{
		int i=5;
		--i;  // i= 4
		int var= i + --i - i + --i - i;
		//  var= 4 + 3 - 3 + 2 - 2 = 4
		System.out.println(var);
		System.out.println(i);
	}
}
```

Output:

```
4
2
```

Unary Example 8:

```
public class UnaryExample8
{
	public static void main(String[] args)
	{
		int i=5;
		i--;  // i= 4
		int var= i + i-- - i + i-- + i;
		//  var= 4 + 4 - 3 + 3 + 2 = 10
		System.out.println(var);
		System.out.println(i);
	}
}
```

Output:

```
10
2
```

Unary Example 9:

```
public class UnaryExample9
{
	public static void main(String[] args)
	{
		int i=5;
		i--;  // i= 4
		int var= i + i-- + ++i - --i + i++ - i;
		//  var= 4 + 4 + 4 - 3 + 3 -4 = 8
		System.out.println(var);
		System.out.println(i);
```

```
    }
}
```

Output:

```
8
4
```

Unary Example 10:

```
public class UnaryExample10
{
    public static void main(String[] args)
    {
        int i=10;
        int j=-11;
        System.out.println(~i); // -(var + 1)
        // -(10 + 1) = -11
        System.out.println(~j); // -(var + 1)
        // -(-11 + 1) = -(-10) = 10
        System.out.println(!false);
        System.out.println(!true);
    }
}
```

Output:

```
-11
10
true
false
```

11.3 Assignment Operators

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.

Syntax:
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**.

Operator	Example	Same as	Description
+=	a+=5	a=a+5	for adding left operand with right operand and then assigning it to the variable on the left.

-=	a-=5	a=a-5	for subtracting right operand from left operand and then assigning it to the variable on the left.
*=	a*=5	a=a*5	for multiplying left operand with right operand and then assigning it to the variable on the left.
/=	a/=5	a=a/5	for dividing left operand by right operand and then assigning it to the variable on the left.
%=	a%=5	a=a%5	for assigning modulo of left operand by right operand and then assigning it to the variable on the left.

Assignment Example:

```
public class AssignmentExample
{
	public static void main(String[] args)
	{
		int a=4;
		a+=10; // a=a+10 = 4+10 = 14
		System.out.println(a);
		int b=3;
		b-=4; // b=b-4 = 3-4 = -1
		System.out.println(b);
		int c=7;
		c*=2; // c=c*2 = 7*2 = 14
		System.out.println(c);
		int d=6;
		d/=3; // d=d/3 = 6/3 =2
		System.out.println(d);
		int e=23;
		e%=2; // e=e%2 = 23%2 = 1
		System.out.println(e);
	}
}
```

Output:

```
14
-1
14
2
1
```

11.4 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.

Operator	Name	Example	Description
==	Equal to	x==y	returns true if the left-hand side is equal to the right-hand side.
!=	Not equal	x!=y	returns true if the left-hand side is not equal to the right-hand side.
<	Less than	x<y	returns true if the left-hand side is less than the right-hand side.
<=	Less that or equal to	x<=y	returns true if the left-hand side is less than or equal to the right-hand side.
>	Greater than	x>y	returns true if the left-hand side is greater than the right-hand side.
>=	Greater than or equal to	x>=y	returns true if the left-hand side is greater than or equal to the right-hand side.

Relational Example:

```
public class RelationalExample
{
	public static void main(String[] args)
	{
		System.out.println(2.0==2.0);
		System.out.println(4!=4);
		System.out.println(3<4);
		System.out.println(3<=3);
		System.out.println(3>4);
		System.out.println(4>=4);
	}
}
```

Output:

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

11.5 Logical Operators

These operators are used to perform “logical AND” and “logical OR” operations, i.e., a function similar to AND gate and OR gate in digital electronics. One thing to keep in mind is the second condition is not evaluated if the first one is false, i.e., it has a short-circuiting effect. Used extensively to test for several conditions for making a decision. Java also has “Logical NOT”, which returns true when the condition is false and vice-versa.

Operator	Name	Example	Description
&&	Logical AND	x>5 && x<10	Returns true if both statements are true
||	Logical OR	x>9 || x<1	Returns true if one of the statements is true
!	Logical NOT	!(x>5 && x<10)	Reverse the result, returns false if the result is true

Logical Example:

```
public class LogicalExample
{
	public static void main(String[] args)
	{
		int x=4;
		System.out.println(x>5 && x<10);
		System.out.println(x>9 || x<1);
		System.out.println(!(x>5 && x<10));
	}
}
```

Output:

```
false
false
true
```

11.6 Ternary Operators

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

Syntax:

DataType Variable_name = condition ? expression1 : expression2 ;

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

Ternary Example:

```
public class TernaryExample
{
	public static void main(String[] args)
	{
		int x = (4>2) ? 6 : 5;
		System.out.println(x);
		int y = (6<3) ? 2 :10;
		System.out.println(y);
	}
}
```

Output:

```
6
10
```

11.7 Bitwise Operators

These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.

Name	Operator	Description
&	Bitwise AND operator	The result is true i.e., 1 when both the bits are true i.e., 1
|	Bitwise OR operator	The result is true i.e., 1 when any one the bits is true i.e., 1
^	Bitwise XOR operator	The result is true i.e., 1 when the bits are in the combination of 0 to 1

Bitwise Example

```
public class BitwiseExample1
{
      public static void main(String[] args)
      {
            int x=10;
            int y=3;
            System.out.println(x&y);
            /*
             Truth table of AND or & in Digital Electronics
             ----------------------------------------------
             case 1      case 2      result
                1           1           1
                1           0           0
                0           1           0
                0           0           0
             ----------------------------------------------

             Decimal x=10 ----> Binary 1 0 1 0
                   operation &
             Decimal y=3 -----> Binary 0 0 1 1
             ---------------------------------
             Result ----------> Binary 0 0 1 0
             ---------------------------------
             Convert the above result into Decimal
             0010 Binary -------> 2 Decimal
             */
      }
}
```

Output:

```
2
```

Bitwise Example2

```
public class BitwiseExample2
{
	public static void main(String[] args)
	{
		int x=10;
		int y=3;
		System.out.println(x|y);
		/*
		 Truth table of OR or | in Digital Electronics
		 ---------------------------------------------
		 case 1		case 2		result
		    1		  1		   1
		    1		  0		   1
		    0		  1		   1
		    0		  0		   0
		 ---------------------------------------------

		 Decimal x=10 ----> Binary 1 0 1 0
		       operation |
		 Decimal y=3 -----> Binary 0 0 1 1
		 ---------------------------------
		 Result ----------> Binary 1 0 1 1
		 ---------------------------------
		 Convert the above result into Decimal
		 1011 Binary -------> 11 Decimal
		 */
	}
}
```

Output:

```
11
```

Bitwise Example 3:

```
public class BitwiseExample3
{
	public static void main(String[] args)
	{
		int x=10;
		int y=3;
		System.out.println(x^y);
```

```
        /*
         Truth table of XOR or ^ in Digital Electronics
         ----------------------------------------------
         case 1      case 2      result
            1           1           0
            1           0           1
            0           1           1
            0           0           0
         ----------------------------------------------

         Decimal x=10 ----> Binary 1 0 1 0
               operation ^
         Decimal y=3 -----> Binary 0 0 1 1
         ---------------------------------
         Result ----------> Binary 1 0 0 1
         ---------------------------------
         Convert the above result into Decimal
         1001 Binary -------> 9 Decimal
         */
    }
}
```

Output:

```
9
```

11.8 Shift Operator

These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two.

Name	Operator	Description
<<	Left Shift operator	Shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as multiplying the number with some power of two.
>>	Signed Right Shift operator	Shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit depends on the sign of the initial number. Similar effect as dividing the number with some power of two.
>>>	Unsigned Right Shift operator	shifts the bits of the number to the right and fills 0 on voids left as a result. The leftmost bit is set to 0.

Shift Example 1:

```
public class ShiftExample1
{
	public static void main(String[] args)
	{
		System.out.println(10<<2);//10*2^2=10*4=40
		System.out.println(10<<3);//10*2^3=10*8=80
		System.out.println(-20<<2);//20*2^2=20*4=-80
		System.out.println(-15<<4);//15*2^4=15*16=-240
	}
}
```

Output:

```
40
80
-80
-240
```

Shift Example 2:

```
public class ShiftExample2
{
	public static void main(String[] args)
	{
		System.out.println(10>>2);//10/2^2=10/4=2
		System.out.println(20>>>2);//20/2^2=20/4=5
		System.out.println(-20>>2);//-20/2^2=-20/4=-5
		System.out.println(-20>>3);//-20/2^3=-20/8=-3
	}
}
```

Output:

```
2
5
-5
-3
```

Shift Example 3:

```
public class ShiftExample3
{
	public static void main(String[] args)
	{
		 //For positive number, >> and >>> works same
	    System.out.println(20>>2);
	    System.out.println(20>>>2);
	    //For negative number,
	    //>>> changes parity bit (MSB) to 0
	    System.out.println(-20>>2);
	    System.out.println(-20>>>2);
```

```
      }
}
```

Output:

```
5
5
-5
1073741819
```

11.9 instanceof Operator

The instance of the operator is used for type checking. It can be used to test if an object is an instance of a class, a subclass, or an interface.

instanceof Example

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

        Person obj1 = new Person();
        Person obj2 = new Boy();

        // As obj is of type person, it is not an
        // instance of Boy or interface
        System.out.println("obj1 instanceof Person: "
                           + (obj1 instanceof Person));
        System.out.println("obj1 instanceof Boy: "
                           + (obj1 instanceof Boy));
        System.out.println("obj1 instanceof MyInterface: "
                          + (obj1 instanceof MyInterface));

        // Since obj2 is of type boy,
        // whose parent class is person
        // and it implements the interface Myinterface
        // it is instance of all of these classes
        System.out.println("obj2 instanceof Person: "
                           + (obj2 instanceof Person));
        System.out.println("obj2 instanceof Boy: "
                           + (obj2 instanceof Boy));
        System.out.println("obj2 instanceof MyInterface: "
                          + (obj2 instanceof MyInterface));
    }
}

class Person
{
}
```

```
class Boy extends Person implements MyInterface
{
}

interface MyInterface
{
}
```

Output:

```
obj1 instanceof Person: true
obj1 instanceof Boy: false
obj1 instanceof MyInterface: false
obj2 instanceof Person: true
obj2 instanceof Boy: true
obj2 instanceof MyInterface: true
```

11.10 Interesting Questions about Operators

1. **Precedence and Associativity:**
   There is often confusion when it comes to hybrid equations which are equations having multiple operators. The problem is which part to solve first. There is a golden rule to follow in these situations. If the operators have different precedence, solve the higher precedence first. If they have the same precedence, solve according to associativity, that is, either from right to left or from left to right. The explanation of the below program is well written in comments within the program itself.

2. **Be a Compiler:**
   Compiler in our systems uses a lex tool to match the greatest match when generating tokens. This creates a bit of a problem if overlooked. For example, consider the statement a=b+++c; too many of the readers might seem to create a compiler error. But this statement is absolutely correct as the token created by lex are a, =, b, ++, +, c. Therefore, this statement has a similar effect of first assigning b+c to a and then incrementing b. Similarly, a=b+++++c; would generate an error as tokens generated are a, =, b, ++, ++, +, c. which is actually an error as there is no operand after the second unary operand.

3. **Using + over():**
   When using + operator inside system.out.println() make sure to do addition using parenthesis. If we write something before doing addition, then string addition takes place, that is, associativity of addition is left to right, and hence integers are added to a string first producing a string, and string objects concatenate when using +. Therefore it can create unwanted results.

Example Program:

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

     int x = 5, y = 8;

     // concatenates x and y as
     // first x is added to "concatenation (x+y) = "
     // producing "concatenation (x+y) = 5"
     // and then 8 is further concatenated.
     System.out.println("Concatenation (x+y)= " + x + y);

     // addition of x and y
     System.out.println("Addition (x+y) = " + (x + y));
    }
}
```

Output:

```
Concatenation (x+y)= 58
Addition (x+y) = 13
```