Made by :- Amrit Agrawal

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**

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

1. *** :** Multiplication
2. **/ :** Division
3. **% :** Modulo
4. **+ :** Addition
5. **– :** Subtraction

```
package Amrit;
public class operator1
{
	public static void main(String[] args)
	{
		int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
		String x = "Thank", y = "You";

		System.out.println("a + b = " + (a + b));
		System.out.println("a - b = " + (a - b));
		System.out.println("x + y = " + x + y);
		System.out.println("a * b = " + (a * b));
		System.out.println("a / b = " + (a / b));
		System.out.println("a % b = " + (a % b));
	}
}
```

Output:-

```
a + b = 30
a - b = 10
x + y = ThankYou
a * b = 200
a / b = 2
a % b = 0
```

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

A) – :Unary minus used for negating the values.

B) + :Unary plus, used for giving positive values. Only used when deliberately converting a negative value to positive.

C) ++ :Increment operator, used for incrementing the value by 1. There are two varieties of increment operator.

- **Post-Increment :** Value is first used for computing the result and then incremented. **ex a++;**
- **Pre-Increment :** Value is incremented first and then result is computed. **ex ++a;**

D) -- : Decrement operator used for decrementing the value by 1. There are two varieties of decrement operator.

- **Post-decrement :** Value is first used for computing the result and then decremented. **ex a--;**
- **Pre-Decrement :** Value is decremented first and then result is computed. Ex:**--a;**

E) ! : Logical not operator, used for inverting a boolean value.

```
package Amrit;
public class operator1
{
	public static void main(String[] args)
	{
			int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;

			boolean condition = true;


			c = ++a;
			System.out.println("Value of c (++a) = " + c);
			c = +a;
			System.out.println("Value of c (+a) = " + c);
			c = a++;
			System.out.println("Value of c (a++) = " + c);
			//c = a+;
			System.out.println("Value of c (a+) = not possible " );

			c = ++b;
			System.out.println("Value of c (++b) = " + c);
			c = +b;
			System.out.println("Value of c (+b) = " + c);
			c = b++;
			System.out.println("Value of c (b++) = " + c);
			//c = b+;
			System.out.println("Value of c (b+) = not possible");


			c = --d;
			System.out.println("Value of c (--d) = " + c);
			c = -d;
			System.out.println("Value of c (-d) = " + c);
			c = d--;
			System.out.println("Value of c (d--) = " + c);
			System.out.println("Value of c (d- ) = not possible");

			c = e--;
			System.out.println("Value of c (e--) = " + c);


			System.out.println("Value of !condition ="+ !condition);
		}
	}
```

Output:-

```
/*Value of c (++a) = 21
Value of c (+a) = 21
Value of c (a++) = 21
Value of c (a+) = not possible
Value of c (++b) = 11
Value of c (+b) = 11
```

```
Value of c (b++) = 11
Value of c (b+) = not possible
Value of c (--d) = 19
Value of c (-d) = -19
Value of c (d--) = 19
Value of c (d- ) = not possible
Value of c (e--) = 40
Value of !condition =false
*/
```

3) Assignment Operator : '='

For example, instead of a **=** a+5, we can write a **+=** 5.

1. **+=,** for adding left operand with right operand and then assigning it to variable on the left.
2. **-=,** for subtracting left operand with right operand and then assigning it to variable on the left.
3. ***=**, for multiplying left operand with right operand and then assigning it to variable on the left.
4. **/=**, for dividing left operand with right operand and then assigning it to variable on the left.
5. **%=**, for assigning modulo of left operand with right operand and then assigning it to variable on the left.

For example:-

int a = 5;

a += 5; or a = a+5;

```
package Amrit;
public class operator1
{
	public static void main(String[] args)
	{
		int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;
		c = b;
		System.out.println("Value of c = " + c);
		a = a + 1;
		b = b - 1;
		e = e * 2;
		f = f / 2;
		System.out.println("a, b, e, f = " + a + ", "+ b + ", " + e + ", " + f);


		a = a - 1;
		b = b + 1;
		e = e / 2;
```

```
            f = f * 2;

            a += 1;
            b -= 1;
            e *= 2;
            f /= 2;
            System.out.println("a, b, e, f ("+ "using shorthand operators)= "+ a +
", " + b + ", "+ e + ", " + f);
      }
}
```

Output:-

```
/*Value of c = 10
a, b, e, f = 21, 9, 20, 2
a, b, e, f (using shorthand operators)= 21, 9, 20, 28*/
```

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

1. **==, Equal to :** returns true of left hand side is equal to right hand side.
2. **!=, Not Equal to :** returns true of left hand side is not equal to right hand side.
3. **<, less than :** returns true of left hand side is less than right hand side.
4. **<=, less than or equal to :** returns true of left hand side is less than or equal to right hand side.
5. **>, Greater than :** returns true of left hand side is greater than right hand side.
6. **>=, Greater than or equal to:** returns true of left hand side is greater than or equal to right hand side.

```
package Amrit;
public class operator1
{
      public static void main(String[] args)
      {
            int a = 20, b = 10;
            String x = "Thank", y = "Thank";
            int ar[] = { 1, 2, 3 };
            int br[] = { 1, 2, 3 };
            boolean condition = true;
            System.out.println("a == b :" + (a == b));
            System.out.println("a < b :" + (a < b));
            System.out.println("a <= b :" + (a <= b));
            System.out.println("a > b :" + (a > b));
            System.out.println("a >= b :" + (a >= b));
            System.out.println("a != b :" + (a != b));
            System.out.println("x == y : " + (ar == br));
```

```
                System.out.println("condition==true :"+ (condition == true));
        }
}
```

Output:-

```
/*a == b :false
a < b :false
a <= b :false
a > b :true
a >= b :true
a != b :true
x == y : false
condition==true :true*/
```

5) Logical Operators :- These operators are used to perform "logical AND" and "logical OR" operation, i.e. the 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.

1. **&&, Logical AND :** returns true when both conditions are true.
2. **||, Logical OR :** returns true if at least one condition is true.

```
package Amrit;
import java.util.Scanner;
public class operator1
{
        public static void main(String[] args)
        {
                String x = "Sher";
                String y = "Locked";

                Scanner s = new Scanner(System.in);
                System.out.print("Enter username:");
                String uuid = s.next();
                System.out.print("Enter password:");
                String upwd = s.next();
                if ((uuid.equals(x) && upwd.equals(y)) || (uuid.equals(y) &&
upwd.equals(x)))
                {
                        System.out.println("Welcome user.");
                }
                else
                {
                        System.out.println("Wrong uid or password");
                }
        }
}
```

Output :-

```
/*Enter username:Sher
Enter password:Locked
Welcome user.*/
```

6) Ternary operator : Ternary operator is a shorthand version of if-else statement. ?

condition ? if true : if false

```
package Amrit;
import java.util.Scanner;
public class operator1
{
	public static void main(String[] args)
	{
			int a = 20, b = 10, c = 30, result;
			result = ((a > b) ? (a > c) ? a : c : (b > c) ? b : c);
			System.out.println("Max of three numbers = "+ result);
		}
	}

	/*Max of three numbers = 30*/
```

7) Bitwise Operators : These operators are used to perform 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 Binary indexed tree.

1. **&, Bitwise AND operator:** returns bit by bit AND of input values.
2. **|, Bitwise OR operator:** returns bit by bit OR of input values.
3. **^, Bitwise XOR operator:** returns bit by bit XOR of input values.
4. **~, Bitwise Complement Operator:** This is a unary operator which returns the one's compliment representation of the input value, i.e. with all bits inversed.

```
package Amrit;
import java.util.Scanner;
public class operator1
{
	public static void main(String[] args)
	{
		int a = 0x0005;
		int b = 0x0007;
		System.out.println("a&b = " + (a & b));
		System.out.println("a|b = " + (a | b));
		System.out.println("a^b = " + (a ^ b));
		System.out.println("~a = " + ~a);
```

```
                a &= b;
                System.out.println("a= " + a);
        }
}
```

Output:-

```
/*
a&b = 5
a|b = 7
a^b = 2
~a = -6
a= 5
     */
```

8) Shift Operators : 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. General format-

1. **<<, Left shift operator:** shifts the bits of the number to the left and fills 0 on voids left as a result. Similar effect as of multiplying the number with some power of two.
2. **>>, 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 initial number. Similar effect as of dividing the number with some power of two.
3. **>>>, 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

```
package Amrit;
import java.util.Scanner;
public class operator1
{
        public static void main(String[] args)
        {
                int a = 0x0005;
                int b = -10;
                System.out.println("a<<2 = " + (a << 2));
                System.out.println("a>>2 = " + (a >> 2));
                System.out.println("b>>>2 = " + (b >>> 2));
        }
        }
```

Output:-

```
/*
a<<2 = 20
a>>2 = 1
```

```
b>>>2 = 1073741821
 */
```

9) instance of operator : Instance of 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. General format-

object instance of class/subclass/interface

```
package Amrit;
import java.util.Scanner;
class operator1
{
	public static void main(String[] args)
	{

		Person obj1 = new Person();
		Person obj2 = new Boy();
		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));
		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
 */
```

Importants and hints:-

S.no	Operators	
1)	Arithmetic Operators	1. * : Multiplication 2. / : Division 3. % : Modulo 4. + : Addition 5. – : Subtraction
2)	Unary Operators	A) – :Unary minus B) + :Unary plus, C) ++ :Increment operator • Post-Increment : ex a++; • Pre-Increment : ex ++a; D) -- : Decrement operator. • Post-decrement : ex a--; • Pre-Decrement :ex --a; E) ! : Logical not operator,
3)	Assignment Operator	a = a+5, we can write a += 5. 1. +=,. 2. -=, 3. *=, 4. /=, 5. %=,
4)	Relational Operators	1. ==, Equal to 2. !=, Not Equal to 3. <, less than. 4. <=, less than or equal to 5. >, Greater than 6. >=, Greater than or equal to:

5)	Logical Operators	1. &&, Logical AND 2. ||, Logical OR
6)	Ternary Operator	condition ? if true : if false
7)	Bitwise Operators	1. &, Bitwise AND operator 2. |, Bitwise OR operator 3. ^, Bitwise XOR operator 4. ~, Bitwise Complement Operator:
8)	Shift Operators	1. <<, Left shift operator 2. >>, Signed Right shift operator 3. >>>, Unsigned Right shift operator
9)	instance of operator	instanceof