Java Operators: Comprehensive Guide

1. Arithmetic Operators

Theory

Arithmetic operators perform mathematical operations on numeric values.

Example

```
public class ArithmeticOperatorsDemo {
    public static void main(String[] args) {
        int a = 10, b = 5;

        // Addition
        System.out.println("a + b = " + (a + b));  // 15

        // Subtraction
        System.out.println("a - b = " + (a - b));  // 5

        // Multiplication
        System.out.println("a * b = " + (a * b));  // 50

        // Division
        System.out.println("a / b = " + (a / b));  // 2

        // Modulus (Remainder)
        System.out.println("a % b = " + (a % b));  // 0

        // Unary plus and minus
        int x = +5;   // Positive 5
        int y = -5;   // Negative 5

        // Increment and Decrement
        int z = 5;
        System.out.println("Pre-increment: " + (++z));    // 6
        System.out.println("Post-increment: " + (z++));   // 6 (increments after
use)
        System.out.println("Pre-decrement: " + (--z));    // 5
        System.out.println("Post-decrement: " + (z--));   // 5 (decrements after
use)
    }
}
```

2. Relational Operators

Theory

Relational operators compare two values and return a boolean result.

Example

```
public class RelationalOperatorsDemo {
    public static void main(String[] args) {
        int a = 10, b = 5;

        // Equal to
        System.out.println("a == b: " + (a == b));  // false

        // Not equal to
        System.out.println("a != b: " + (a != b));  // true

        // Greater than
        System.out.println("a > b: " + (a > b));    // true

        // Less than
        System.out.println("a < b: " + (a < b));    // false

        // Greater than or equal to
        System.out.println("a >= b: " + (a >= b));  // true

        // Less than or equal to
        System.out.println("a <= b: " + (a <= b));  // false
    }
}
```

3. Logical Operators

Theory

Logical operators perform logical operations on boolean values.

Example

```
public class LogicalOperatorsDemo {
    public static void main(String[] args) {
        boolean x = true, y = false;

        // Logical AND
        System.out.println("x && y: " + (x && y));  // false

        // Logical OR
        System.out.println("x || y: " + (x || y));  // true

        // Logical NOT
        System.out.println("!x: " + (!x));          // false

        // Complex logical expression
        int age = 25;
        boolean isStudent = true;
```

```
        boolean canVote = (age >= 18) && isStudent;
        System.out.println("Can vote: " + canVote);  // true
    }
}
```

4. Bitwise Operators

Theory

Bitwise operators perform operations on individual bits of integers.

Example

```
public class BitwiseOperatorsDemo {
    public static void main(String[] args) {
        int a = 5;  // 0101 in binary
        int b = 3;  // 0011 in binary

        // Bitwise AND
        System.out.println("a & b: " + (a & b));  // 1 (0001 in binary)

        // Bitwise OR
        System.out.println("a | b: " + (a | b));  // 7 (0111 in binary)

        // Bitwise XOR
        System.out.println("a ^ b: " + (a ^ b));  // 6 (0110 in binary)

        // Bitwise NOT
        System.out.println("~a: " + (~a));  // -6

        // Left shift
        System.out.println("a << 1: " + (a << 1));  // 10 (shifts bits left)

        // Right shift
        System.out.println("a >> 1: " + (a >> 1));  // 2 (shifts bits right)
    }
}
```

5. Assignment Operators

Theory

Assignment operators are used to assign values to variables.

Example

```
public class AssignmentOperatorsDemo {
    public static void main(String[] args) {
        int a = 10;
```

```
        // Simple assignment
        a = 20;

        // Addition assignment
        a += 5;  // Equivalent to a = a + 5

        // Subtraction assignment
        a -= 3;  // Equivalent to a = a - 3

        // Multiplication assignment
        a *= 2;  // Equivalent to a = a * 2

        // Division assignment
        a /= 2;  // Equivalent to a = a / 2

        // Modulus assignment
        a %= 3;  // Equivalent to a = a % 3
    }
}
```

6. Ternary Operator

Theory

Ternary operator is a shorthand for if-else statement.

Example

```
public class TernaryOperatorDemo {
    public static void main(String[] args) {
        int age = 20;

        // Basic ternary operator
        String status = (age >= 18) ? "Adult" : "Minor";
        System.out.println(status);

        // Nested ternary operator
        int x = 10, y = 20, z = 30;
        int max = (x > y) ? ((x > z) ? x : z) : ((y > z) ? y : z);
        System.out.println("Maximum value: " + max);
    }
}
```

7. Operator Precedence Table

Precedence	Operator Type	Operators	Associativity
1 (Highest)	Postfix	`() [] .`	Left to Right

Precedence	Operator Type	Operators	Associativity
2	Unary	`! + - ++ --`	Right to Left
3	Multiplicative	`* / %`	Left to Right
4	Additive	`+ -`	Left to Right
5	Shift	`<< >> >>>`	Left to Right
6	Relational	`< > <= >= instanceof`	Left to Right
7	Equality	`== !=`	Left to Right
8	Bitwise AND	`&`	Left to Right
9	Bitwise XOR	`^`	Left to Right
10	Bitwise OR	`|`	Left to Right
11	Logical AND	`&&`	Left to Right
12	Logical OR	`||`	Left to Right
13	Ternary	`?:`	Right to Left
14 (Lowest)	Assignment	`= += -= *= /= %=`	Right to Left

Best Practices

Operator Usage Guidelines

1. Use parentheses to clarify complex expressions
2. Be cautious with bitwise operators
3. Understand operator precedence
4. Avoid overly complex expressions
5. Use type casting carefully

Common Pitfalls

- Integer division truncation
- Floating-point precision issues
- Unexpected results with bitwise operations
- Operator precedence misunderstandings

Practice Exercises

1. Create a calculator program using various operators
2. Implement a bitwise operation demonstrator
3. Write a program to swap two numbers without using a third variable
4. Develop a grade classification system using relational and logical operators

Recommended Learning Path

1. Master basic arithmetic and assignment operators
2. Understand relational and logical operators
3. Learn bitwise operators
4. Practice complex expressions
5. Study operator precedence
6. Solve coding challenges

Conclusion

Operators are fundamental to Java programming. They allow you to perform computations, make comparisons, and manipulate data. Understanding their behavior and precedence is crucial for writing efficient and correct code.