Operators

1. Arithmetic Operators

These are like the basic math you do in school. They help you add, subtract, multiply, and divide numbers.

- **Addition (+)**: Adds two numbers together.

```
int a = 10;
int b = 30;
int sum = a + b;
System.out.println("The sum is: " + sum); // The sum is: 40
```

Example: If you have 10 apples and get 30 more, you'll have 40 apples in total.

- **Subtraction (-)**: Takes one number away from another.

```
int diff = a - b;
System.out.println("The difference is: " + diff); // The difference is: -20
```

Example: If you start with 10 apples and give away 30, you'll be short by 20 apples.

- **Multiplication (*)**: Multiplies two numbers.

```
int prod = a * b;
System.out.println("The product is: " + prod); // The product is: 300
```

Example: If you have 10 baskets, each with 30 apples, you'll have 300 apples in total.

- **Division (/)**: Splits one number by another.

```
int quot = a / b;
System.out.println("The quotient is: " + quot); // The quotient is: 0
```

Example: If you try to divide 10 apples into 30 parts, each part gets 0 apples (since 10 is smaller than 30).

- **Modulo (%)**: Shows what's left after dividing.

```
int rem = a % b;
System.out.println("The remainder is: " + rem); // The remainder is: 10
```

Example: If you have 10 apples and try to divide them into 30 baskets, you'll have 10 apples left over.

2. Unary Operators

These work with just one number and can change its value or give you true/false answers.

- **Unary Minus (`-`)**: Makes a number negative.

```
int k = 40;
System.out.println("Unary minus of k is: " + (-k)); // Unary minus of k is:
-40
```

Example: If you have 40 apples and you make it negative, you get -40.

- **Unary Negation (`!`)**: Turns true to false, and false to true.

```
boolean bool = false;
System.out.println("Unary negation of bool is: " + (!bool)); // Unary
negation of bool is: true
```

Example: If it's not raining, but you say it's raining, you change false to true.

- **Pre-Increment (`++k`)**: Increases a number by 1 before using it.

```
System.out.println("Pre-increment of k is: " + (++k)); // Pre-increment of k
is: 41
```

Example: If you have 40 apples and you add 1 before you count them, you'll have 41 apples.

- **Post-Increment (`k++`)**: Increases a number by 1 after using it.

```
System.out.println("Post-increment of k is: " + (k++) + " (k after post-
increment: " + k + ")");
// Post-increment of k is: 41 (k after post-increment: 42)
```

Example: If you have 41 apples and you add 1 after counting, you will count 41, but have 42 apples.

- **Pre-Decrement (`--k`)**: Decreases a number by 1 before using it.

```
System.out.println("Pre-decrement of k is: " + (--k)); // Pre-decrement of k
is: 41
```

Example: If you have 42 apples and you remove 1 before counting, you will have 41 apples.

- **Post-Decrement (`k--`)**: Decreases a number by 1 after using it.

```
System.out.println("Post-decrement of k is: " + (k--) + " (k after post-
decrement: " + k + ")");
// Post-decrement of k is: 41 (k after post-decrement: 40)
```

Example: If you have 41 apples and you remove 1 after counting, you will count 41, but have 40 apples.

3. Assignment Operators

These are used to give values to variables.

- **Simple Assignment (`=`)**: Just gives a variable a value.

```
int d = a;
System.out.println("Simple assignment d = a, d is: " + d); // Simple
assignment d = a, d is: 10
```

Example: If you have 10 apples and you put them in another basket, that basket also has 10 apples.

- **Compound Assignment (`+=`, `-=`, `*=`, `/=`, `%=`)**: Combines an operation with assignment.

```
d += 5; // d = d + 5
System.out.println("d += 5, d is: " + d); // d += 5, d is: 15
```

Example: If you have 10 apples and you get 5 more, you now have 15 apples.

4. Bitwise Operators

These work with the bits of numbers, like working with the smallest pieces of information in a computer.

- **Bitwise OR (`|`)**: Combines bits where at least one bit is 1.

```
int n = 10; // 1010 in binary
int z = 5;  // 0101 in binary
System.out.println("Bitwise OR of n and z is: " + (n | z)); // Bitwise OR of
n and z is: 15
```

Example: Combining 1010 and 0101 gives 1111, which is 15 in decimal.

- **Bitwise AND (`&`)**: Combines bits where both bits are 1.

```
System.out.println("Bitwise AND of n and z is: " + (n & z)); // Bitwise AND
of n and z is: 0
```

Example: Combining 1010 and 0101 gives 0000, which is 0 in decimal.

- **Bitwise XOR (^)**: Combines bits where only one bit is 1.

```
System.out.println("Bitwise XOR of n and z is: " + (n ^ z)); // Bitwise XOR
of n and z is: 15
```

Example: Combining 1010 and 0101 gives 1111, which is 15 in decimal.

- **Bitwise Complement (~)**: Flips all bits.

```
System.out.println("Bitwise Complement of z is: " + (~z)); // Bitwise
Complement of z is: -6
```

Example: Flipping 0101 gives 1010 (in 2's complement form, it's -6).

- **Compound Bitwise Assignment (&=, |=, ^=)**: Combines a bitwise operation with assignment.

```
int x = 10; // 1010 in binary
x &= 6;     // 0110 in binary
System.out.println("x &= 6, x is: " + x); // x &= 6, x is: 2
```

Example: Combining 1010 and 0110 with AND gives 0010, which is 2 in decimal.

5. Relational Operators

These compare two values and tell you if they are the same or different.

- **Equal to (==)**: Checks if two values are the same.

```
int f1 = 5;
int f2 = 3;
System.out.println("f1 == f2: " + (f1 == f2)); // f1 == f2: false
```

Example: 5 is not the same as 3, so it's false.

- **Not equal to (!=)**: Checks if two values are different.

```
System.out.println("f1 != f2: " + (f1 != f2)); // f1 != f2: true
```

Example: 5 is different from 3, so it's true.

- **Less than (<)**: Checks if one value is smaller than another.

```
System.out.println("f1 < f2: " + (f1 < f2)); // f1 < f2: false
```

Example: 5 is not smaller than 3, so it's false.

- **Less than or equal to (<=)**: Checks if one value is smaller or equal to another.

```
System.out.println("f1 <= f2: " + (f1 <= f2)); // f1 <= f2: false
```

- **Greater than (>)**: Checks if one value is larger than another.

```
System.out.println("f1 > f2: " + (f1 > f2)); // f1 > f2: true
```

Example: 5 is greater than 3, so it's true.

- **Greater than or equal to (>=)**: Checks if one value is larger or equal to another.

```
System.out.println("f1 >= f2: " + (f1 >= f2)); // f1 >= f2: true
```

Example: 5 is greater than 3, so it's true.

6. Logical Operators

These are used to combine or invert conditions, like when making decisions.

- **Logical AND (&&)**: Returns true if both conditions are true.

```
int l1 = 10;
int l2 = 20;
System.out.println("l1 > 2 && l2 < 3: " + (l1 > 2 && l2 < 3)); // l1 > 2 &&
l2 < 3: false
```

Example: If you have 10 apples and 20 bananas, the statement that you have more than 2 apples and less than 3 bananas is false.

- **Logical OR (||)**: Returns true if at least one condition is true.

```
System.out.println("l1 >= 2 || l2 < 3: " + (l1 >= 2 || l2 < 3)); // l1 >= 2
|| l2 < 3: true
```

Example: If you have 10 apples and 20 bananas, the statement that you have at least 2 apples or less than 3 bananas is true because you have more than 2 apples.

- **Logical NOT (`!`)**: Inverts the value of a condition.

```
System.out.println("!(l1 > 2 && l2 < 3): " + !(l1 > 2 && l2 < 3)); // !(l1 > 2 && l2 < 3): true
```

Example: If it's not true that you have more than 2 apples and less than 3 bananas, then it's true that the opposite is correct.

7. Ternary Operator (`?:`)

This is a shortcut for `if-else` statements. It chooses between two values based on a condition.

- **Syntax**: `variable = condition ? valueIfTrue : valueIfFalse;`

```
int var = 10;
boolean result = var >= 10;
System.out.println("Result of conditional operation var >= 10 is: [" + result + "]"); // Result of conditional operation var >= 10 is: [true]
```

Example: If `var` is greater than or equal to 10, the result will be true. Since `var` is 10, the result is true.

8. Instanceof Operator

This checks if an object is an instance of a certain class or type.

- **Syntax**: `object instanceof Class`

```
String str = "Hello";
boolean isString = str instanceof String;
System.out.println("str is instance of String: " + isString); // str is instance of String: true
```

Example: Since `str` is a String, the result is true.

9. Type Cast Operator

This is used to convert a value from one type to another.

- **Syntax**: `(type) variable`

```
double pi = 3.14;
int piInt = (int) pi; // Type casting from double to int
System.out.println("Casted pi to int: " + piInt); // Casted pi to int: 3
```

Example: If you convert 3.14 to an integer, you get 3 because it drops the decimal part.

10. Shift Operators

These shift the bits of a number left or right, changing its value.

- **Left Shift (<<)**: Moves bits to the left, filling with zeros.

```
int num = 8; // 00001000 in binary
System.out.println("Left shift num << 2: " + (num << 2)); // Left shift num
<< 2: 32
```

Example: Shifting 8 left by 2 places (from 00001000 to 00100000) gives 32.

- **Right Shift (>>)**: Moves bits to the right, preserving the sign bit (for negative numbers).

```
int negativeNum = -8; // 11110111 in binary (32-bit representation)
System.out.println("Right shift negativeNum >> 2: " + (negativeNum >> 2));
// Right shift negativeNum >> 2: -2
```

Example: Shifting -8 right by 2 places gives -2 in binary.

- **Unsigned Right Shift (>>>)**: Moves bits to the right, filling with zeros (even for negative numbers).

```
System.out.println("Unsigned right shift negativeNum >>> 2: " + (negativeNum
>>> 2)); // Unsigned right shift negativeNum >>> 2: 1073741822
```

Example: Shifting -8 right by 2 places with unsigned shift results in a large positive number because it fills with zeros.
