Operators in Java

Introduction

Operators are specific symbols in Java that are the basic building blocks of any language used to perform operations on variables and values. Java provides different operators that can be categorised into other groups based on functionality. Here are some operators discussed below.

1. Arithmetic Operators

Arithmetic operators in Java perform various mathematical operations on numeric values and variables representing such values as integers, float-point numbers, etc.

Types:

- **'+' Addition**: adds two numbers and concatenates two string literals.

```
  int x = 4+5; //assigns value 9 to x.

  String a= "hello";
  String b ="world";
  String sentence =a+b   //Assigns value "helloworld" to sentence

Note - '+' operator works in two ways while using numbers it performs
addition while with Strings it performs concatenation.
```

- **'-' Subtraction**: subtraction of numbers.

```
 int z = 10/2; //assigns value 5 to z.
```

- **'*' Multiplication**: Multiplication of numbers.

```
int z = 3 * 4;     //assigns value 12 to z
```

- **'/' Division**: gives quotient after the division of numbers.

```
 int z = 10/2; //assigns value 5 to z.
```

- **%' Modulo**: gives remainder after division of numbers.

```
System.out.println(10/3);   //prints value 1.
```

2. Assignment Operators

As the name suggests, assignment operators assign values to variables that are fundamental to programming. The right side operand of the assignment operator is a value, and the left side operand is a variable. Assignment operators follow right-side associativity, i.e., expressions are evaluated from right to left.

Key points to remember:

- Datatypes of value and variable should be the same.
- The value must be declared before using it or should be a constant.
- **'=' operator**: assigns the value on the right to the variable on the left.
  int x = 5; //assigns value 5 to the variable x.
  int y = x; //assigns value of x (i.e. 5) to the variable y.
- **Augmented Operators:** Expressions involving the same variable on both sides of the operand to manipulate the variable's value are augmented operators. For example,

```
int x = 4;
int x = x + 5;  //value of x will change from  4 to 10.

//the above code can also be written as:
x += 5;   //addition assignment operator.

The same can be done with other arithmetic operators.

x -= 2; // x = 2
x *= 4; // x = 16
x /= 2; // x = 2
```

3. Precedence:

Precedence determines the order in which operators are evaluated in an expression. When an expression contains multiple operators, Java follows a specific set of rules to determine which operation should first be performed. Below is the table with decreasing order of precedence, i.e. the operator above the other operators is given a higher preference than the one below.

Operators	Precedence
Postfix	expr++ expr–
Unary	++expr –expr +expr -expr ~!
Multiplicative	* / %

Additive	+ -
Shift	<< >> >>>
Relational	< > <= >= instanceOf
Equality	== !=
Bitwise And	&
Bitwise Exclusive OR	^
Bitwise Inclusive OR	|
Logical AND	&&
Logical OR	||
Ternary	? :
Assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Will learn about the other operators ahead.

```
System.out.println(2+5*4)  //prints 22, as multiplication is executed
first.
```

4. Implicit Conversions:

Implicit conversion, also known as type promotion, is the automatic conversion of one data type to another by the Java compiler without writing any extra instructing code. We'll now see some string operations depicting conversion.

- Implicit conversion with strings occurs when we combine strings with other data types, like numbers.

- Java can automatically convert these non-string data types into strings, so they play nicely together.

```
System.out.println(2+5*4)  //prints 22, as multiplication is executed
first.
```

Here, the Java compiler identified that we were mixing the variables of different data types; therefore, it converted int to string implicitly, concatenated with the string literals and stored it in 'sentence'.

```
int n1 = 10;
double n2 = 3.5;

Double result = n1 + n2;

//int n1 implicitly got promoted to double 10.0 and added to n2.
```

Note - Implicit conversion is only allowed while converting shorter datatypes to larger data types as it is possible to accommodate smaller data types to larger ones, which is also known as widening.

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

        long a = 2.5;
        int b = 4;
        int c = a+b;

//This code will throw a compilation error.

    }
}
```

5. Type Casting:

Typecasting refers to converting one data type to another. Two common types of typecasting are implicit and explicit casting.

- **Implicit:** Implicit type casting refers to converting a small data type into a larger one by Java to prevent data loss. It is also known as widening or automatic type conversion.

```
int a= 5;
double b= smaller; // Implicit casting from int to double
```

- **Explicit:** Explicit type casting refers to converting a large data type into a small one. Explicit type casting may result in data loss and requires the programmer to specify the type conversion explicitly. It is also known as narrowing or manual type conversion.

```
double a= 10.5;
int b= (int) larger; // Explicit casting from double to int
```

Key points to remember:

- Use implicit casting when converting to a larger data type to prevent data loss or perform operations involving mixed data types.
- Use explicit casting when converting to a smaller data type, but be cautious about potential data loss. Explicit casting allows you to control the conversion.
- Explicit casting requires manual intervention and may result in data loss.
- Choose the appropriate casting method based on your data type conversion needs.

6. Unary Operators:

In Java, unary operators are those operators that perform operations on a single operand used to manipulate values in different ways.

- **Unary Plus (+) Operator:** This operator doesn't change any value; it just states that the number is positive.

  int x = +5;

- **Unary Minus (-) Operator**: The minus operator negates the value. It turns positive numbers into negative ones and vice versa.

  int x = -5;

- **Increment Operator (++)**: The operator increases the value of a variable by 1. It can be used in two ways: pre-increment and post-increment.

```
int a = 10;
System.out.println(a++);

//prints initial value 10 and then increments value by 1.
```

```
int a = 10;
System.out.println(++a);

//increments the value of 'a' by 1 first and then prints 'a'  i.e. 11.
```

- **Decrement Operator (--)**: The decrement operator decrements the value by 1. Just like the increment operator, it can also be used in two ways, i.e. pre-decrement and post-decrement

```
int a = 10;
System.out.println(a--);

//prints initial value 10 and then decrements value by 1.
```

```
int a = 10;
System.out.println(--a);

//decrements the value of 'a' by 1 first and then prints 'a'  i.e. 9.
```

- **Logical Not Operator (!)**: The logical not operator reverses the value of a boolean expression. True becomes false and vice-versa.

```
boolean flag = true;
            System.out.print(!flag);

      //prints false
```

7. Bitwise Operators:

Bitwise operators are used in Java to manipulate individual data bits in integers or longs. These operators allow you to perform low-level operations on binary representations of data.

- **The Bitwise AND operator ('&'):** The bitwise AND operator (&) performs a bitwise AND operation between corresponding bits of two integers. It results in a 1 only if both bits being compared are 1. For example,

```
int a = 5;    // Binary: 0101
int b = 3;    // Binary: 0011
int result = a & b;  // Result: 0001 (Decimal 1)
```

- **Bitwise OR operator ('|'):** The bitwise OR operator (|) performs a bitwise OR operation between corresponding bits of two integers. It results in a 1 if at least one of the compared bits is 1. For example,

```
int a = 5;    // Binary: 0101
int b = 3;    // Binary: 0011
int result = a | b;  // Result: 0111 (Decimal 7)
```

- **Left shift operator (‘<<’):** The left shift operator (<<) shifts the bits of an integer to the left by a specified number of positions. It effectively multiplies the number by 2, raising the shift amount's power.

```
int a = 5;    // Binary: 0101
int result = a << 2;  // Result: 20
```

- **Right shift operator (‘<<’):** The right shift operator (>>) shifts the bits of an integer to the right by a specified number of positions. It effectively divides the number by 2, raised to the power of the shift amount.

```
int a = 16;   // Binary: 10000
int result = a >> 2;  // Result: 4
```

8. Relational Operators:

Relational operators in Java are used to compare two values and determine their relationship. These operators return a boolean result, indicating whether the specified condition is true or false.

- **Equality Operator (‘==’):** The equality operator compares two values for equality by checking if the values on both sides are equal and returns ‘true’ if the values are equal and ‘false’ otherwise.

```
int x = 5;
int y = 5;
boolean isEqual = (x == y); // isEqual = true
```

- **Inequality Operator(!=):** The inequality operator (!=) checks if two values are not equal and returns true if the values are different and false if they are equal.

```
int a = 10;
int b = 5;
boolean isNotEqual = (a != b); // isNotEqual = true
```

.

- **Greater Than Operator (>):** The greater than operator (>) checks if the value on the left is greater than the value on the right. Returns true if the condition is met; otherwise, it returns false.

```
int num1 = 8;
int num2 = 5;
boolean isGreaterThan = (num1 > num2); // isGreaterThan = true
```

- **Less Than Operator ('<'):** The less than operator checks if the value on the left is less than the value on the right and returns true if the condition is true; otherwise, it returns false.

```
int score1 = 75;
int score2 = 90;
boolean isLessThan = (score1 < score2); // isLessThan = true
```

- **Greater Than or Equal To Operator (>=):** The greater than or equal to operator checks if the value on the left is greater than or equal to the value on the right and returns true if the condition is true. Otherwise, it returns false.

```
int age1 = 20;
int age2 = 20;
boolean isGreaterOrEqual = (age1 >= age2); // isGreaterOrEqual = true
```

- **Less Than or Equal To Operator (<=):** The less than or equal operator checks if the value on the left is less than or equal to the right and returns true if the condition is true; otherwise, it returns false.

```
double price1 = 45.0;
double price2 = 50.0;
boolean isLessOrEqual = (price1 <= price2); // isLessOrEqual = true
```

9. Logical Operators:

Logical operators in Java are used to perform logical operations on boolean values or conditions. They help make decisions and control the flow of your program based on the truth values of expressions.

1. Logical AND (&&) Operator:

- The logical AND operator combines two boolean expressions and returns true if both expressions on its left and right sides are valid.
- If any of the expressions is false, the result will be false.

```
boolean isSunny = true;
boolean isWarm = true;
boolean isGoodWeather = isSunny && isWarm; // isGoodWeather = true
```

2. Logical OR Operator(||):

- The logical OR operator combines two boolean expressions and returns true if at least one of the expressions on its left and right sides is true.
- If both expressions are false, the result will be false.

```
boolean hasCoffee = true;
boolean hasMilk = false;
boolean canMakeCoffee = hasCoffee || hasMilk; // canMakeCoffee = true
```

3. Combining Logical Operators: We can combine logical operators to create complex conditions for decision-making in our Java programs.

- Example for using Logical AND and Logical OR:

```
int age = 25;
boolean isStudent = true;
boolean isTeenagerOrStudent = (age < 20) || isStudent; //
isTeenagerOrStudent = true
```

Conclusion

In conclusion, Java provides a variety of operators for performing different operations within Java programs. Here's a summary of the operators we studied:

- Arithmetic Operators: Used for basic mathematical calculations such as addition, subtraction, multiplication, division, and modulus.
- Assignment Operators: Used to assign values to variables and concisely perform operations.
- Relational Operators: Used to compare values and determine their relationship, including equality, inequality, greater than, less than, and combinations.
- Logical Operators: Employed to evaluate logical conditions, combining boolean values using AND (&&) and OR (||) operators and negating conditions with NOT (!).
- Bitwise Operators: Used for low-level bit manipulation operations on integers.
- Conditional Operator (Ternary): Allows conditional value assignment based on a given condition.

These operators are fundamental to Java programming and are essential for performing calculations, making decisions, and controlling the flow of your code. Mastery of these operators is crucial for developing efficient and functional Java applications.