3-Operators

Author: Vincent Lau

Learning Objectives

> Understand different types of Operators & Precedence

Operators

Summary

Operator type	Operators	Purpose
Assignment	`=, +=, -=, *=, /=`	Assign value to a variable
Arithmetic	`+, -, *, /, %, ++, --`	Add, subtract, multiply, divide, and modulus primitives
Relational	`<, <=, >, >=, ==, !=`	Compare primitives
Logical	`!, &&, ||`	Apply NOT, AND, and OR logic to primitives

Assignment Operator

Assignment operators are used to assign values to variables. The basic assignment operator is "=" which assigns the value on the right side to the variable on the left side. For example:

```
1 int x = 5; // Assigns the value 5 to the variable x
```

Arithmetic Operators

Operator	Result
+	Addition (also unary plus)
–	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
– =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
– –	Decrement

Basic Arithmetic Operators

Arithmetic operators are used to perform basic mathematical calculations. Java supports operators such as addition (+), subtraction (-), multiplication (*), division (/), and modulus (%). For example:

```
int x = 10 + 5; // 5, Addition
int y = 10 - 3; // 7, Subtraction
int z = 10 * 5; // 50, Multiplication
int w = 10 / 5; // 2, Division
int remainder = 10 % 3; // 1, Modulus (returns the Remainder)
```

Compound Assignment Operators

Compound assignment operators combine assignment with another operation. They perform the operation and then assign the result to the variable. Here are a few examples:

```
int x = 5;
x += 3; // Equivalent to x = x + 3;  Result: x = 8

int y = 10;
y -= 4; // Equivalent to y = y - 4;  Result: y = 6

int z = 7;
z *= 2; // Equivalent to z = z * 2;  Result: z = 14

int w = 15;
w /= 5; // Equivalent to w = w / 5;  Result: w = 3

int a = 10;
a %= 3; // Equivalent to a = a % 3;  Result: a = 1
```

Increment Operator (++)

The increment operator "++" is used to increase the value of a variable by 1. It can be used in two ways:

- Pre-increment: The value is incremented before any other operation.
- Post-increment: The value is incremented after any other operation.

Here's an example to illustrate both types of increment operators:

```
int x = 5;
x++; // 6
++x; // 7
x = x + 1; // 8
x += 1 // 9
```

But if ...

```
int x = 5;
int preIncrement = ++x;
// Pre-increment: The value of x is incremented first, and then assigned to preIncrement.
// Result: x = 6, preIncrement = 6
int y = 5;
int postIncrement = y++;
// Post-increment: The value of y is assigned to postIncrement first and then incremented.
// Result: y = 6, postIncrement = 5
```

Decrement Operator (--)

The decrement operator "--" is used to decrease the value of a variable by 1. Similar to the increment operator, it can be used in two ways:

- Pre-decrement: The value is decremented before any other operation.
- Post-decrement: The value is decremented after any other operation.

Here's an example to illustrate both types of decrement operators:

```
int a = 8;
a--; // 7
--a; // 6
a = a - 1; // 5
a -= 1 // 4
```

But if ...

```
int a = 8;
int preDecrement = --a;
// Pre-decrement: The value of a is decremented first, and then assigned to preDecrement.
// Result: a = 7, preDecrement = 7
int b = 8;
int postDecrement = b--;
// Post-decrement: The value of b is assigned to postDecrement first, and then decremented.
// Result: b = 7, postDecrement = 8
```

String Concatenation Assignment

The "+=" operator can also be used for string concatenation, where the right-hand side is appended to the existing string on the left-hand side.

```
String message = "Hello";
message += " World!";  // Equivalent to message = message + " World!";
// Result: "Hello World!"
```

Logical Operators

Operators && (AND)	Operator || (OR)	Operator ! (NOT)
true && true → true true && false → false false && true → false false && false → false true && true && false → false	true || true → true true || false → true false || true → true false || false → false false || false || true → true	!true → false !false → true

Logical operators are used to perform logical operations on boolean expressions. The commonly used logical operators in Java are "&&" (logical AND), "||" (logical OR), and "!" (logical NOT). For example:

```
boolean a = true;
boolean b = false;
boolean result1 = a && b; // false, Logical AND (asking if a and b are true)
boolean result2 = a || b; // true, Logical OR (asking if either a or b is true)
boolean result3 = !a; // false, Logical NOT (asking if a is false)
```

Comparison Operators

op	meaning	true	false
==	equal	2 == 2	2 == 3
!=	not equal	3 != 2	2 != 2
<	less than	2 < 13	2 < 2
<=	less than or equal	2 <= 2	3 <= 2
>	greater than	13 > 2	2 > 13
>=	greater than or equal	3 >= 2	2 >= 3

Comparison operators are used to compare values. They return a boolean result (true or false) based on the comparison. Common comparison operators in Java include "==", "!=", "<", ">", "<=", ">=", etc.

For example:

```java
// Comparison Operators with int
int x = 5;
int y = 10;
boolean result1 = x == y; // false, asking if they are equal to
boolean result2 = x != y; // true, asking if they are not equal to
boolean result3 = x < y; // true, asking if x is less than y
boolean result4 = x > y; // false, asking if x is greater than y
boolean result5 = x <= y; // true, asking if x is less than or equal to y
boolean result6 = x >= y; // false, asking if x is greater than or equal to y

// Comparison Operators with chars
char c1 = 'A';
char c2 = 'B';
boolean result1 = c1 == c2; // false: c1 is not equal to c2
boolean result2 = c1 != c2; // true: c1 is not equal to c2
boolean result3 = c1 < c2;  // true: c1 comes before c2 in the character set
boolean result4 = c1 > c2;  // false: c1 does not come after c2 in the
character set
boolean result5 = c1 <= c2; // true: c1 comes before or is equal to c2
boolean result6 = c1 >= c2; // false: c1 does not come after or is equal to c2

// Comparison Operators with Booleans:
boolean b1 = true;
boolean b2 = false;
boolean result1 = b1 == b2; // false: b1 is not equal to b2
boolean result2 = b1 != b2; // true: b1 is not equal to b2

// Comparison Operators with Strings:
```

```
28 String s1 = "Hello";
29 String s2 = "World";
30 boolean result1 = s1.equals(s2);      // false: s1 is not equal to s2
31 boolean result2 = !s1.equals(s2);     // true: s1 is not equal to s2
32 boolean result3 = s1.compareTo(s2) < 0; // true: s1 comes before s2
   lexicographically
```

Operator Precedence

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Operator precedence in Java determines the order in which operators are evaluated within an expression. Here's an overview of operator precedence in Java, along with some examples:

Parentheses

Operators enclosed in parentheses have the highest precedence. Expressions within parentheses are evaluated first. For example:

```
int result = (5 + 3) * 2;
// The addition inside parentheses is evaluated first: 8 * 2 = 16
```

Unary Operators

Unary operators, such as increment (++), decrement (--), logical NOT (!), and negation (-), have the next highest precedence. They are evaluated before other operations. For example:

```
int x = 5;
int result = ++x * 2;
// x is incremented first, then multiplied by 2: 6 * 2 = 12
```

Multiplicative Operators

Multiplicative operators, including multiplication (*), division (/), and modulus (%), have higher precedence than additive operators. They are evaluated left to right. For example:

```
int result = 10 + 5 * 2;
// Multiplication is performed first: 5 * 2 = 10, then addition: 10 + 10 = 20
```

Additive Operators

Additive operators, such as addition (+) and subtraction (-), have lower precedence than multiplicative operators. They are evaluated left to right. For example:

```
int result = 10 - 5 + 2;
// Subtraction is performed first: 10 - 5 = 5, then addition: 5 + 2 = 7
```

Relational and Equality Operators

Relational operators (such as <, >, <=, >=) and equality operators (==, !=) have higher precedence than logical operators. They are evaluated left to right. For example:

```
boolean result = 5 + 3 < 10 - 2;
// Addition and subtraction are performed first: 8 < 8, which evaluates to false
```

Logical Operators

Logical operators, such as logical AND (&&) and logical OR (||), have lower precedence than relational and equality operators. They are evaluated left to right. For example:

```
boolean result = true && false || true;
// The logical AND (&&) is evaluated first, then the logical OR (||): false || true, which evaluates to true
```

Assignment Operators

Assignment operators, such as =, +=, -=, etc., have the lowest precedence. They are evaluated right to left. For example:

- If you find this code is

```
int x = 5;
int y = 10;
int z = x += 3 * y;
// z = 35
// First, 3 * y is performed: 3 * 10 = 30, then x is updated: x = x + 30 = 35, and finally, z is assigned the value of x: z = 35
```

Other Examples

Just like what we do in Math, use parentheses to override the default operator precedence.

```
3 + 4 * 4 > 5 * (4 + 3) - 1 // inside parentheses first
3 + 4 * 4 > 5 * 7 - 1 // multiplication
3 + 16 > 5 * 7 - 1 // multiplication
3 + 16 > 35 - 1 // addition
19 > 35 - 1 // subtraction
19 > 34 // greater than
false // result
```

Challenge

```
int x = 3;
int y = (x++ + 3) * x++;
// what is y?
```

Questions

- What are the types of operators?
- Make sure you are familiar with Compound Assignment, Increment & Decrement operators
- Are you able to write programs with combinations of Logical & Comparsion operators?
- Do you know arithmetic operators can be used in string?
- Can the substraction operator be used in String?
- What is the highest Operator Precedence?
- Take the Challenge