Theory: Arithmetic operations

5 minutes 5 / 9 problems solved

Start practicing

In real life, we often perform arithmetic operations. They help us to determine the change from a purchase, calculate the area of a room, count the number of people in a queue, and so on. The same operations are used in programs.

§1. Binary arithmetic operators

The Java programming language provides operators to perform arithmetic operations:

- addition `+`
- subtraction `-`
- multiplication `*`
- integer division `/`
- remainder `%`

The operators are called binary because they take two values as operands.

The following example prints results of **addition**, **subtraction**, and **multiplication**.

```
System.out.println(13 + 25); // prints 38
System.out.println(20 + 70); // prints 90

System.out.println(70 - 30); // prints 40
System.out.println(30 - 70); // prints -40

System.out.println(21 * 3);  // prints 63
System.out.println(20 * 10); // prints 200
```

The `/` operator returns the integer part of the division of two integer numbers, and any fractional part is discarded.

```
System.out.println(8 / 3); // prints 2
System.out.println(41 / 5); // prints 8
```

The `%` in Java is the modulus or remainder operator. It returns the remainder of the division of two numbers. Note, that when the dividend is less than the divisor, the quotient is zero and the remainder equals the dividend. If you still feel uneasy about modulo operation, check out the wiki.

```
System.out.println(10 % 3) // prints 1, because 10 divided by 3 leaves a remainder of 1
System.out.println(12 % 4) // prints 0, because 12 divided by 4 leaves no remainder
System.out.println(5 % 9) // prints 5, because 5 divided by 9 leaves a remainder of 5
```

§2. Writing complex expressions

The operations can be combined to write more complex expressions:

```
System.out.println(1 + 3 * 4 - 2); // prints 11
```

The calculation order coincides with arithmetic rules. Multiplication has a higher priority level than addition and subtraction, so the operation `3 * 4` is calculated first.

To specify the order of execution we can use **parentheses** as in the following:

Current topic:

Arithmetic operations — Stage 1

Topic depends on:

Printing data — Stage 1

Topic is required for:

Integer types and operations — Stage 1

Table of contents:

```
System.out.println((1 + 3) * (4 - 2)); // prints 8
```

As in arithmetic, **parentheses** can be nested. You can also use them for clarity.

§3. Unary operators

A unary operator takes a single value as the operand.

- The **unary plus** operator indicates a positive value. It's an optional operator.

```
System.out.println(+5); // prints 5
```

- The **unary minus** operator negates a value or an expression.

```
System.out.println(-8);  // prints -8
System.out.println(-(100 + 4)); // prints -104
```

They both have a higher level of precedence than the **multiplication** and **division** operators.

§4. The precedence order

There is a precedence order of all arithmetic operators, including parentheses. The list below is sorted from the highest to the lowest precedence level.

- parentheses
- unary plus/minus
- multiplication, division
- addition, subtraction