Theory: Defining methods

23 minutes 0 / 5 problems solved

Skip this topic Start practicing

The program decomposition

A method is a sequence of statements grouped together to perform an operation. In Java, a method is always located inside a class. The relation between methods and classes will be learned further. In this topic, all methods will be located in the same class which contains the `main` method.

The main reason to write methods is to decompose a program into small reusable subroutines. These subroutines can be used many times instead of always re-writing the code. A decomposed program has a modular structure and it is much easier to modify and maintain it than a program consisting of one single big main method. It is an important idea of procedural programming.

In this topic, you will learn how to define new methods. It is assumed you've already known how to invoke existing methods.

The base syntax of methods

In general case, a method has the following six components:

1. **a set of modifiers** (`public` , `static` , etc);
2. a type of the return value;
3. **a name;**
4. a list of parameters (as well known as formal parameters) in parenthesis `()` ;
5. a list of exceptions;
6. **a body** containing statements to perform the operation.

Note that the components are listed here in the correct order for the method declaration in Java. Let's say for now that some of these components are always required (those marked as bold) and others are optional. Pair of parentheses `()` is also required, as well as `{}` for enclosing the method's body.

Now, we will focus on 2, 3, 4 and 6 components. Modifiers will be learned in topics related to object-oriented programming.

Defining a simple method

Here is an example of a simple method that calculates the sum of two given numbers:

```
public static int sum(int a, int b) {
    return a + b;
}
```

The `sum` is a typical method written in Java. It returns the sum of two of its parameters. The parameters are written in the parenthesis `"(...)"`. To return the integer result the keyword `return` is written.

In the general case, returning value and parameters can have any type, including non-primitive types.

Also, the method has two modifiers: `public` and `static`. Keep in mind, that there is a recommended order for the modifiers that you can find in Java Language Specification. Although it is technically possible to write `static public` instead, not following the suggested order will reduce the readability of your code and considered as bad practice. All our examples are compliant with the recommendations.

Signatures

The combination of the name of a method and its parameters is called the **signature**. It doesn't include the returning type, modifiers, and names of parameters.

The considered method `sum` has the following signature `sum(int, int)`.

Here are some examples of other signatures:

- `sum(double, double)`
- `min(long, long, long)`
- `getValue()`

We will learn later when the concept of signature is helpful.

Naming methods

There are two kinds of restrictions for the name of a method: the compiler (required) and the naming convention (optional, but desired).

The Java compiler requires that a method name can be a **legal identifier**. The rules for legal identifiers are the following:

- identifiers are case-sensitive;
- an identifier can include Unicode letters, digits, and two special characters (`$`, `_`);
- an identifier can't start with a digit;
- identifiers must not be a keyword.

In addition, there is a naming convention that restricts possible method names. It's optional but desired for developers.

By the convention, method names should be a verb in lowercase or a multi-word name that begins with a verb in lowercase, followed by adjectives, nouns, etc. In multi-word names, the first letter of the second and the following words should be capitalized. Here are some correct examples:

```
sum
getValue
calculateNumberOfOranges
findUserByName
printArray
```

The listed methods satisfy the convention.

The type of a returning value and parameters

A method can return a single value or nothing. To declare a method that returns nothing you should write the special keyword void as the type of a result value.

The following method prints the sum of two given numbers and returns no value.

```
public static void printSum(int a, int b) {
    System.out.println(a + b);
}
```

A method can take one or multiple parameters of the same or different types. Also, it's possible to declare a method without any parameters, but "()" are still required.

```
/**
 * The method has an int parameter
 */
public static void method1(int a) {
    // do something
}

/**
 * The method has long and double parameters
 */
public static void method2(long a, double b) {
    // do something
}

/**
 * The method has no parameters and returns a value
 */
public static int method3() {
    return 3;
}

/**
 * The method has an int parameter and returns an array of Strings
 */
public static String[] createArray(int lengthOfArray) {
    return new String[lengthOfArray];
}
```

When you call a method with a value of a primitive type then a copy of the value is created. Inside a method, you can process this copy. If you change it, the passed argument is not changed.

```
public static void main(String[] args) {
    int val = 100; // 100
    change(val); // try to change val
    System.out.println(val); // it prints "100", because the method changed a copy of the val
}

/**
 * The method changes a given value
 */
public static void change(int val) {
    val = 400; // now, the copy is 400
}
```

As you can see, the method changed a copy of the given integer value 100.

Method's body

In a method's body, you can write any statements including the conditional statement, any loops, invoking methods and declaring local variables. The declared variables are visible only in this method.

If a method returns a value, the method's body must contain the `return` keyword. Moreover, a method may have multiple returns. But each state can return only a single value.

Let's see an example. The following method performs the integer division on a given value the specified number of times.

```
public static int divideBy2(int number, int times) {
    if (times <= 0) {
        return number;
    }

    for (int i = 0; i < times; i++) {
        number /= 2;
    }

    return number;
}
```

The method `divideBy2` takes two integers and returns another integer value. If the specified parameter `times` is less than or equal to zero, the result is the given `number`, otherwise, the method performs the integer division by two in a loop.

If a method doesn't return a value (it has the keyword `void`), the method body may contain the `return` keyword without returning value. It allows finishing the method ahead of schedule, for example, depending on a condition.

For example, the following method prints its arguments if given numbers are positive, otherwise, it performs the return statement.

```
public static void returnNothingOrPrintNumbers(int a, int b) {
    if (a <= 0 || b <= 0) {
        return;
    }

    System.out.println(a + " " + b);
}
```

So, you can write any calculations in the body of a method.

Conclusion

Define a new method if you'd like to re-use a part of your code, or if the part is a well-separated code that can be changed independently. Methods allow you to decompose a program into some well-understood subroutines and manage them conveniently.