Theory: Calling a method

10 minutes 5 / 8 problems solved

Start practicing

2719 users solved this topic. Latest completion was **about 2 hours ago**.

You may remember that a method is a sequence of statements that can be invoked or referred to by its name. Nothing in particular, huh? But what if we tell you that you can use it in a program anytime you need and anywhere you want just simply invoking it with one line of code? Hope that we caught your attention now!

To get started, we will call some method:

```java
getVolume(a, b, h);
```

Here, `getVolume` is the name of the method that, for example, calculates a volume of some geometric shape, and variables `a`, `b` and `h` are its parameters. Technically, the parameter of a method is some value that is used inside the method. The combination of the name and method's parameters in parentheses is the way we call or invoke the method. Let's focus on it in more detail.

§1. Name of the method

Each method has a name that is used to call it. Generally, it reflects what the method does – prints, finds, calculates, provides you with some information.

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

- identifiers are case-sensitive;
- an identifier can include Unicode letters, digits, underscore `_` or currency characters, such as `$`;
- 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 convention, the one-word name should be a verb in lowercase: `sum`, `multiply`, or `round`.

If a method has a multi-word name, the first letter of the second and the following words should be capitalized: `getValue`, `calculateNumberOfOranges`, or `findLetter`.

OK, now we know how the methods are named. Let's talk about how we actually call it!

§2. Calling a method

If you want to call (or invoke) a method in your program, you should write its name and pass the values of its parameters in parentheses. That's how it can be done:

```java
printNumber(7); // this method prints 7

convertDoubleToInt(1.25) // this method converts double value to int

findUserByName("Kate"); // this method finds a user whose name is Kate
```

Let's take a look at the `findUserByName` example.

Here we pass a String value with a name to make the `findUserByName` do its job. To call a method from the outside of the class it belongs to you need to indicate a class as a prefix. Check these methods:

```java
Math.round(79.378); // method with Math class name
Character.isLetter('a'); // method with Character class name
```

Current topic:

✓ Calling a method Stage 2

Topic depends on:

✓ Branching statements Stage 2

✓ Type casting Stage 1

Topic is required for:

✓ Declaring a method Stage 3

Array Stage 2

String Stage 2

Math library Stage 5

Table of contents:

Let's try to perform a small task by invoking a method:

```
double weight = 63.85;
weight = Math.round(weight); // now weight equals 64.0
```

Here, we have methods that take parameters, but what if we tell you that some methods don't accept parameters at all? In that case, just leave the parentheses empty.

Some methods are called in a slightly different way. Take a look:

```
// this is how you call an instance method

String name = new String("Anya"); // created an instance (1)
name = name.toLowerCase() // anya (2)
```

Methods like this require an object of a certain class to be invoked. Before we called `toLowerCase()` method, we created an object of a `String` class called `name`, since the method in question deals with strings. Now we can call the method for this particular **instance** *(2),* which results in decapitalizing all the letters from our string.

As you see, this method requires an instance to be created before it can be called, that's why it is known as an **instance method**. We will talk about this type of method later.

§3. Built-in methods

Why do you need to rewrite algorithms that have already been written? Of course, we're not talking about some special cases like educational tasks. Still, it is more efficient to use pre-defined methods that are always available to the user. That is why there are two types of methods in Java: **built-in** and **user-defined** methods.

Built-in methods belong to the Standard Java library. Now there are a lot of built-in methods that convert or compare the values, round doubles, find the maximum or the minimum value, and do a lot of useful operations. We've already dealt with `round()`, `isLetter()`, `compareTo()`, `hasNext()` methods, but the number of built-in methods is huge and constantly growing. You can find the method you need in Oracle documentation. For example, check the link to the Math library.

By contrast with built-in methods, user-defined methods are created by the programmer. It is a common practice to create a customized subprogram for a specific purpose. You may create your own method and even put it in your own class. Later on, we will learn why it is a good practice to use user-defined methods and how to actually create them.

§4. Conclusion

Generally, a method is a necessary tool for a programmer who is aimed at a neat and reusable code style. With the help of methods, you can perform any specific task you need. They make the program look more readable, and you don't need to repeat routine code lines over and over. Some tasks are wrapped in special built-in methods that are parts of the standard Java library. Also, there are user-defined methods that are created by the programmer. We will discuss these methods in other topics.

Report a typo

229 users liked this theory. 5 didn't like it. **What about you?**

Start practicing

This content was created 4 months ago and updated 8 days ago. Share your feedback below in comments to help us improve it!

Comments (4) Hints (0) Useful links (0) Show discussion