Creating your own methods and passing arguments

1. Functions/methods in Java

You've already learned lots of Java commands, which means you can write some pretty complex programs. 10, 20, 30 lines of code in a program is not a very big program, right?

But a program of 100+, now that's big, and it can be rather difficult to understand the code. Is there anything you can do to somehow simplify writing and reading programs that have a lot of code?

Yes, and methods (or functions) will help us with this.

What is a method? Putting it very simply, a method is a group of commands that has a unique name. In other words, we just put several commands into one group and give it a unique name. And that's it — the method is ready.

Example:

Without a method

```
class Solution
{
   public static void main(String[] args)
   {
     System.out.print("Wi-");
     System.out.println("Fi");
     System.out.print("Wi-");
     System.out.println("Fi");

     System.out.print("Wi-");
     System.out.println("Fi");
   }
}
```

With a method

```
class Solution
{
   public static void main(String[] args)
   {
     printWiFi();
     printWiFi();
     printWiFi();
   }
   public static void printWiFi()
   {
     System.out.print("Wi-");
     System.out.println("Fi");
   }
}
```

In the program in the left column, we repeat the same code three times — we did this intentionally to illustrate a point. But in the program on the right, we moved the repeated code into a separate method and gave it a unique name — `printWiFi`.

And instead of the relocated code, we call the `printWiFi()` method 3 times.

When the program in the column on the right is run, each time the `printWiFi()` method is executed, all the commands inside the `printWiFi()` method are executed. We just created a new command (method), combining several commands into a single group.

Any code can be split into separate methods. This is done to simplify things: the idea is that it is better to have many small methods than one large one. You'll soon be surprised that there was ever a time when you wrote your own programs without writing your own methods.

---

2. Declaring a method in Java

So how do you write your method correctly?

```
public static void name()
{
  method body
}
```

There are plenty of nuances to consider when declaring (creating) a method, but we'll start with the basics. How can we declare the simplest method? A simple method declaration looks like this:

Where `name` is the unique name of the method and `method body` represents the commands that make up the method. The meaning of the words `public`, `static`, and `void` will be discussed later.

After we've created a method, we can call it in our other methods. A method call looks like this:

```
name();
```

Where `name` is the unique name of the method we want to call, i.e. the method whose commands we want to execute when we arrive at the method call.

When the program reaches the method call, it will simply step into the method, execute all its commands, return to the original method, and continue execution.

As you probably guessed by now, most of the commands we've learned so far are just methods written by other programmers to make our life easier: `System.out.println()`, `Thread.sleep()`, etc.

A method can contain calls to other methods:

Code	Note

`class Solution`
`{`
`  public static void main(String[] args)`
`  {`
`  printWiFi10Times();`
`  }`
 	
`  public static void printWiFi10Times()`
`  {`
`  for (int i = 0; i < 10; i++)`
`  printWiFi();`
`  }`
 	
`  public static void printWiFi()`
`  {`
`  System.out.print("Wi-");`
`  System.out.println("Fi");`
`  }`
`}`
	Call the `printWiFi10Times()` method
	
	Declare the `printWiFi10Times` method
	
	Call the `printWiFi()` method 10 times in a loop
	
	Declare the `printWiFi` method
	
	Display text on the screen:
	`Wi-Fi`

3. Facts about methods

Here are some more facts about methods:

Fact 1. A method is always part of a class.

A method can only be declared in a class. A method cannot be declared inside another method. A method cannot be declared outside a class.

Fact 2. A method's name has no sacred meaning.

It doesn't matter what methods are called — that doesn't affect anything. The main method is a method just like all the others. It's just that this name was chosen for the method from which the Java machine will start execution of the program. There is nothing magical about it.

Fact 3. The order of methods in a class does not matter.

You can write your methods in a class in any order — this will not affect the program's execution in any way. Example:

Code

```
class Solution
{
   public static void printWiFi10Times()
   {
     for (int i = 0; i < 10; i++)
       printWiFi();
   }

   public static void main(String[] args)
   {
     printWiFi10Times();
   }

   public static void printWiFi()
   {
     System.out.print("Wi-");
     System.out.println("Fi");
   }
}
```

```
class Solution
{
   public static void printWiFi()
   {
     System.out.print("Wi-");
     System.out.println("Fi");
   }

   public static void printWiFi10Times()
   {
     for (int i = 0; i < 10; i++)
       printWiFi();
   }
   public static void main(String[] args)
   {
     printWiFi10Times();
   }
}
```

Fact 4. The variables inside one method are not related in any way to the variables of other methods.

What happens in Vegas, stays in Vegas. And the variables declared inside a method stay inside the method.

Variables with the same names can be declared in two adjacent methods, and these variables are not related to each other in any way.

---

4. Method names

It has long been known that the two most difficult problems in programming are choosing the right names for methods and choosing the right names for variables.

In fact, almost an entire science has emerged regarding how to correctly name methods. And each programming language has its own standards. In Java, it is customary to follow these principles:

Principle 1. A method name should briefly describe what the method does.

Then another programmer reading your code can rely on the name of the method to guess what the code does. He or she won't need to look at the code of called methods every time. And the purpose of the methods is easier to remember.

Recall that `Thread.sleep()` is used to 'put the program to sleep' and `Scanner.nextInt()` is used to 'read the next integer'. Convenient, right?

Principle 2. A method name can be multiple words.

However, there are several limitations when doing this:

- You cannot have spaces in a method name: all words are written together.
- Each word is capitalized, except for the first.
- A method name always starts with a lowercase letter

Recall the `printWiFi10Times` method. What does that name mean? "Display the word 'WiFi' 10 times". You shouldn't include lots of words in the name of a method: the name should reflect its very essence.

This standard for naming methods is called CamelCase (The uppercase letters are like the humps of a camel).

Principle 3. A method name starts with a verb.

A method always does something, so the first word in a method name is always an action.

Here are some bad names for methods: `home`, `cat`, `car`, `train`, ...;

Some good names are: `run`, `execute`, `print`, `read`, `write`, ...

Principle 4. Use only Latin letters and numbers.

Java has excellent support for different languages. You can write the names of variables, methods and classes in Russian as well as Chinese — everything will work!

But! How long would you have to study Java, if the `System.out.println()` method was written in Chinese? Much longer than now, right? That's the first point.

Second, many software development teams are international. A very large number of Java libraries are used by programmers from all over the world.

Therefore, it is recommended to use only Latin letters and numbers in method names.

Important:

The name of a method must begin with a letter (it cannot begin with a number).