

Method parameters in Java

1. Passing arguments

And now the fun begins. You probably already know from methods like `System.out.println()` that we can pass arguments to methods. Once we are inside the method, we refer to them as parameters. In fact, parameters greatly enhance the benefits we get from creating and using methods.

How do we declare a method with parameters? It's actually quite simple:

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

Where **name** is the unique name of the method and **method body** represents the commands that make up the method.

And **parameters** is a placeholder for the method parameters, separated by commas. Let's describe this template in more detail:

```
public static void name(Type1 name1, Type2 name2, Type3 name3)
{
    method body
}
```

Examples:

Code	Explanation
<pre>public static void print(String str) { }</pre>	The <code>print</code> method is declared with a parameter: <code>String str</code>
<pre>public static void print(String str, int count) { }</pre>	The <code>print</code> method is declared with two parameters: <code>String str</code> <code>int count</code>
<pre>public static void write(int x, int y) { }</pre>	The <code>write</code> method is declared with two parameters: <code>int x</code> <code>int y</code>

If we don't want the method to have parameters, then we just leave the parentheses empty.

Parameters are special variables within a method. With their help, you can pass various values to the method when it is called.

For example, let's write a method that displays a string of text a given number of times.

You already know how to write code to display a string on the screen several times. But which string should you print? And how many times? That's what we need the parameters for.

The code that does this would look like this:

Code	Explanation
<pre>class Solution { public static void printLines(String text, int count) { for (int i = 0; i < count; i++) System.out.println(text); } public static void main(String[] args) { printLines("Hi", 10); printLines("Bye", 20); } }</pre>	<p>We declared the <code>printLines</code> method with the following parameters:</p> <p><code>String text</code>, <code>int count</code></p> <p>The method displays String <code>text</code> <code>count</code> times</p> <p>We call the <code>printLines</code> method with various parameters</p>

Each time a method is called, its parameters are assigned the passed values, and only then do we start to execute the commands inside the method.

2. Arguments

I would like to draw your attention a little more to calling a method with parameters.

The values passed to the method are usually called arguments when they are passed to the method.

Let's look at another example:

Code	Explanation
<pre>class Solution { public static void printLines(String text, int count) { for (int i = 0; i < count; i++) System.out.println(text); } public static void main(String[] args) { printLines("Hi", 10); printLines("Bye", 20); } }</pre>	<p>We declared the <code>printLines</code> method with the following parameters:</p> <p><code>String text</code>, <code>int count</code></p> <p>The method displays String <code>text</code> <code>count</code> times</p> <p>We call the <code>printLines</code> method with the following arguments:</p> <p><code>text = "Hi"; count = 10;</code> <code>text = "Bye"; count = 20;</code></p>

When the `printLines` method was called for the first time, its parameters were assigned the following values:
String text = "Hi", int count = 10.

When the `printLines` method was called the second time, its parameters were assigned different values:
String text = "Bye", int count = 20.

Parameters are no more and no less than variables that are assigned certain values when a method is called. The values "Hi", "Bye", 10, and 20 are themselves called arguments."

3. Conflicting variable names when calling a method

Variables can be used as method arguments. This is simple and understandable, but it can potentially produce some difficulties. Let's go back to that same example, but this time we'll move the arguments into separate variables:

Code	Explanation
<pre> class Solution { public static void printLines(String text, int count) { for (int i = 0; i < count; i++) System.out.print(text); } public static void main(String[] args) { String str = "Hi"; int n = 10; printLines(str, n); } } </pre>	<p>We declared the <code>printLines</code> method with the following parameters:</p> <p><code>String text</code>, <code>int count</code></p> <p>The method displays String <code>text</code> <code>count</code> times</p> <p>We call the <code>printLines</code> method with the following arguments:</p> <p><code>text = str;</code> <code>count = n;</code></p>

So far, so good: we have a `str` variable. Its value is assigned to the `text` parameter when the method is called. We have an `n` variable. Its value is assigned to the `count` parameter when the method is called." So far, everything is clear.

Now let's rename our variables in the `main` method:

Code	Explanation
<pre> class Solution { public static void printLines(String text, int count) { for (int i = 0; i < count; i++) System.out.print(text); } public static void main(String[] args) { String text = "Hi"; int count = 10; printLines(text, count); } } </pre>	<p>We declared the <code>printLines</code> method with the following parameters:</p> <p><code>String text</code>, <code>int count</code></p> <p>The method displays String <code>text</code> <code>count</code> times</p> <p>We call the <code>printLines</code> method with the following arguments:</p> <p><code>text = text;</code> <code>count = count;</code></p>

Pay attention to two things

[illegible]

And what happens if one of these variables is a method parameter?

Code	Explanation
<pre>class Solution { public static void printArraySum(int[] data) { int sum = 0; for (int i = 0; i < data.length; i++) sum = sum + data[i]; System.out.println(sum); } public static void main(String[] args) { int[] months = {31, 28, 31, 30, 31, 30, 31, 31, 30}; printArraySum(months); } }</pre>	<p>The <code>printArraySum</code> method calculates the sum of the numbers in the passed array and displays it on the screen</p>

Exactly the same thing happens: the `data` parameter will contain a reference to the same area of memory as the `months` variable. When the method is called, a simple assignment occurs: `data = months`.

And since both variables refer to the same area of memory storing an integer, then the `printArraySum` method can not only read values from the array, but also change them!

For example, we can write our own method that fills a two-dimensional array with the same value. This is how it might look:

Code	Explanation
<pre> class Solution { public static void fill(int[][] data, int value) { for (int i = 0; i < data.length; i++) { for (int j = 0; j < data[i].length; j++) data[i][j] = value; } } public static void main(String[] args) { int[][] months = {{31, 28}, {31, 30, 31}, {30, 31, 31}}; fill (months, 8); } } </pre>	<p>The <code>fill</code> method iterates over every cell in the passed two-dimensional array and assigns <code>value</code> to them.</p> <p>We create a two-dimensional array. We fill the entire array with the number <code>8</code>.</p>

5. Methods with the same name

Now let's return to method names once again.

Java standards require all methods in the same class to have unique names. In other words, it's impossible to declare two identically named methods in the same class.

When methods are compared for sameness, **not only are the names taken into account, but also the types of the parameters!** **Note that the names of the parameters are not taken into account.** Examples:

Code	Explanation
<pre> void fill(int[] data, int value) { } void fill(int[][] data, int value) { } void fill(int[][][] data, int value) { } </pre>	<p>These three methods are different methods. They can be declared in the same class.</p>
<pre> void print(String str) { } void print(String str, String str2) { } void print(int val) { } void print(double val) { } void print() { } </pre>	<p>Each of these five methods is considered different. They can be declared in the same class.</p>
<pre> void sum(int x, int y) { } void sum(int data, int value) { } </pre>	<p>These two methods are considered the same, meaning they cannot be declared in the same class.</p>

Why are some methods considered **the same**, while others are **different**? And why are parameter names not taken into account when determining the uniqueness of a method?

Why is uniqueness necessary at all? The thing is that when the compiler compiles a program, it must know exactly which method you intend to call in any given place.

For example, if you write `System.out.println("Hi")`, the compiler is smart and will easily conclude that you intend to call the `println()` method here with a `String` parameter.

But if you write `System.out.println(1.0)`, the compiler will see a call to the `println()` method with a `double` parameter.

When a method is called, the compiler ensures that the types of the arguments match the types of the parameters. It does not pay any attention to the name of the arguments. In Java, parameter names do not help the compiler determine which method to call. And that's why they are not taken into account when determining the uniqueness of a method.

The name of a method and the types of its parameters are called the **method signature**. For example, `sum(int, int)`

Each class must have **methods with unique signatures** rather than methods with unique names.