CODEGYM

# Java Developer in 12 months

Mentor-supported training program from CodeGym

**MODULE 1. JAVA SYNTAX**

Lesson 8 Methods

# Lesson plan

- Methods in Java

- Declaring and calling a method in Java

- Method parameters

- Passing references to methods

- Method overloading

- The return statement

- Access modifiers

- static keyword

- Local variables

# Methods in Java

**A method is a group of commands that has a unique name.**

| Without a method | With a method |
|---|---|
| ```java
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");
    }
}
``` | ```java
class Solution {
    public static void main(String[] args) {
        printWiFi();
        printWiFi();
        printWiFi();
    }

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

Any code can be split into separate methods. This is done to make things easier to understand: the idea is that it is better to have many small methods than one large one.

# Declaring and calling a method in Java

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

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

**A method call looks like this:** `name();`

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.

`System.out.println()` → **This is also a method that we already know**

# Method parameters

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

```
public static void name(type1 name1, type2 name2,
type3, name3) {
    method body
}
```

| Code | Explanation |
|------|-------------|
| ```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("Hello", 10);         printLines("Bye", 20);     } }``` | We declared the printLines method with the following parameters: String text, int count The method displays String text count times   We call the printLines method with various parameters |

# Passing references to methods

Some Java variables (string variables and array variables) store not the values themselves, but instead a reference, i.e. the address of the block of memory where the values are located. Such variables are called reference variables.

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | | int[] a | | | | int array | | |
| 2 | | F2 | | | | 0 | | |
| 3 | | | | | | 2 | | |
| 4 | | | | | | 44 | | |
| 5 | | | | | | 2 | | |
| 6 | | | | | | | | |

```java
int[] a = {0,2,44,2};
int[] b = a;
```

When you assign an array variable to another array variable, the two variables start to refer to the same space in memory:

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | | int[] a | | | | int array | | |
| 2 | | F2 | | | | 0 | | |
| 3 | | | | | | 2 | | |
| 4 | | int[] b | | | | 44 | | |
| 5 | | F2 | | | | 2 | | |
| 6 | | | | | | | | |

# Passing references to methods

When you pass a reference to a method as an argument, what happens is the same as when assigning one reference to another.

```
int[] a = {0,2,44,2};
int[] b = a;
```

```java
class Solution {
 public static void main(String[] args) {
   int[][] array = {{31, 28}, {31, 30, 31}, {30, 31, 31}};
   fill (array, 8);
 }

 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;
   }
  }
 }
}
```

We create a two-dimensional array.
We fill the entire array with the number 8.

The fill method iterates over every cell in the passed two-dimensional array and assigns value to them .

The array variable (in the main method) and the data variable (in the fill method) refer to the same array in memory

# Method overloading

The name of a method and the types of its parameters are called the **method signature.**

For example, `sum(int, int)`

It isn't that each class must have unique method names. Each class must have **methods with unique signatures.**

| Code | Explanation |
|---|---|
| ```void fill(int[] data, int value) {```<br>```}```<br><br>```void fill(int[][] data, int value) {```<br>```}```<br><br>```void fill(int[][][] data, int value) {```<br>```}``` | These three methods are different methods. They can be declared in the same class. |
| ```void sum(int x, int y) {```<br>```}```<br><br>```void sum(int data, int value) {```<br>```}``` | These two methods are considered the same, meaning they cannot be declared in the same class. |

# The return statement

The **return** statement allows you to instantly terminate a method in which it is called.

For example, if **return** is called in the **main** method, then the **main** method will immediately end, and with it the entire program.

```java
import java.util.Arrays;
class Solution {
    public static void main(String[] args) {
        int[] array = new int[10];

        for (int i = 0; i < 10; i++) {
            array[i] = i;

            if (i == 3) {
                return;
            }
            System.out.println(Arrays.toString(array));
        }
    }
}
```

In the main method, we declare an array of 10 elements.

We begin to fill the array in the loop.

As soon as the loop counter reaches 3, the main method exits and the program ends.

Display the current state of the array

Result of running the program:
**[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]**
**[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]**
**[0, 1, 2, 0, 0, 0, 0, 0, 0, 0]**

# Methods with a result, void

In Java, methods can have a value. And this is very good news: methods are not only able to do something based on the input parameters, but also, for example, to evaluate something and return the result of the calculation.

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

Where **name** is the name of the method, parameters is the list of method parameters, and type is the type of the result that the method returns.

For methods that return nothing, there is a special placeholder type: void.

The return keyword is used to return the result of a method. For methods declared as void, there is no return value. Accordingly, the return keyword is used like this:

```
return;
```

For methods whose result type is not void, we return the method's result by using the return keyword like this:

```
return value;
```

Where **return** is a statement that terminates the method immediately. And **value** is the value that the method returns to the calling method when it exits.
The type of value must match the type specified in the method declaration.

# Access modifiers

Before each method, you can indicate access modifiers:

| public | | protected | | private |

There are a total of 3 such modifiers, but there are 4 types of access to a method. This is because the absence of an access modifier also means something.

| | Access from... | | | |
|---|---|---|---|---|
| Modifiers | Any class | Child class | Its package | Its class |
| public | Yes | Yes | Yes | Yes |
| protected | No | Yes | Yes | Yes |
| no modifier | No | No | Yes | Yes |
| private | No | No | No | Yes |

# Static keyword

The static keyword is used to declare static class members: static methods and static variables.

```
ClassName.MethodName()
```

A static method is not attached to any object, but instead belongs to the class in which it is declared.

Examples of static methods:

| Class name | Static method name | |
|---|---|---|
| Thread.sleep() | Thread | sleep() |
| Math.abs() | Math | abs() |
| Arrays.sort() | Arrays | sort() |

A static method cannot access the non-static methods and fields of its own class. A static method can only access static methods and fields.

# Local variables

All variables that are declared inside methods are called local variables. A local variable exists only in the block of code in which it is declared.

```java
public static void main(String[] args) {
    int a = 5;                          a
    if (a < 10) {                       a
        int b = 10;                     a, b
        while (true) {                  a, b
            int x = a + b;              a, b, x
            System.out.println(x);      a, b, x
        }                               a, b
        System.out.println(b);          a, b
    }                                   a
}
```
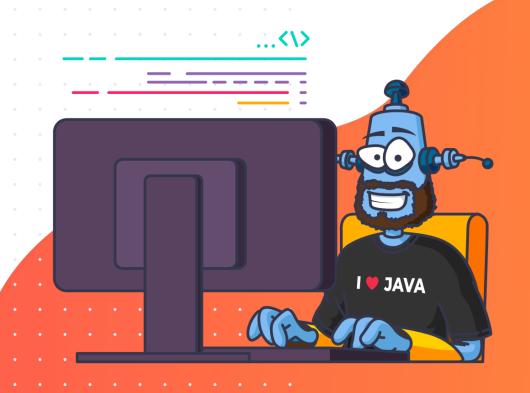
You cannot declare two local variables with the same name in one code block – the program will not compile. But you can do this if the blocks of code where the variables are declared do not overlap.

# Homework

Complete Level 9