Part 2

Methods and dividing the program into smaller parts

Learning Objectives

- You are familiar with the concepts of a method parameter, a method's return value, and a program's call stack.
- You know how to create methods and how to call them from both the main program (the `main` method) as well as from inside other methods.
- You can create parameterized and non-parameterized methods, and you can create methods that return a value.

So far, we've used various commands: value assignment, calculations, conditional statements, and loops.

Printing to the screen has been done with the statement `System.out.println()`, and the reading of values with `Integer.valueOf(scanner.nextLine())`. `if` has been used in conditional statements, and `while` and `for` in loops. We notice that printing and reading operations somewhat differ from `if`, `while`, and `for` in that the print and read commands are followed by parentheses, which may include parameters passed to the command. The ones that "end in parentheses" are not actually commands, but methods.

Technically speaking, **a method** is a named set of statements. It's a piece of a program that can be called from elsewhere in the code by the name given to the method. For instance `System.out.println("I am a parameter given to the method!")` calls a methods that performs printing to the screen. The internal implementation of the method — meaning the set of statements to be executed — is hidden, and the programmer does not need to concern themselves with it when using the method.

So far all the methods we have used have been ready-made Java methods. Next we will learn to create our own methods.

Custom Methods

A method means a named set consisting of statements that can be called from elsewhere in the program code by its name. Programming languages offer pre-made methods, but programmers can also write their own ones. It would, in fact, be quite exceptional if a program used no methods written by the programmer, because methods help in structuring the program. From this point onward nearly every program on the course will therefore contain custom-created methods.

In the code boilerplate, methods are written outside of the curly braces of the `main`, yet inside out the "outermost" curly braces. They can be located above or below the main.

```
import java.util.Scanner;

public class Example {
    public static void main(String[] args) {
        Scanner scanned = new Scanner(System.in);
        // program code
    }

    // your own methods here
}
```

Let's observe how to create a new method. We'll create the method `greet`.

```
public static void greet() {
    System.out.println("Greetings from the method world!");
}
```

And then we'll insert it into a suitable place for a method.

```
import java.util.Scanner;

public class Example {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // program code
    }

    // your own methods here
    public static void greet() {
        System.out.println("Greetings from the method world!");
    }
}
```

The definition of the method consists of two parts. The first line of the definition includes the name of the method, i.e. `greet`. On the left side of the name are the keywords `public static void`. Beneath the line containing the name of the method is a code block surrounded by curly brackets, inside of which is the code of the

method — the commands that are executed when the method is called. The only thing our method `greet` does is write a line of text on the screen.

Calling a custom method is simple: write the name of the methods followed by a set of parentheses and the semicolon. In the following snippet the main program (main) calls the greet method four times in total.

```
import java.util.Scanner;

public class ProgramStructure {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // program code
        System.out.println("Let's try if we can travel to the method world:");
        greet();

        System.out.println("Looks like we can, let's try again:");
        greet();
        greet();
        greet();
    }

    // own methods
    public static void greet() {
        System.out.println("Greetings from the method world!");
    }
}
```

The execution of the program produces the following output:

Sample output

```
Let's try if we can travel to the method world:
Greetings from the method world!
Looks like we can, let's try again:
Greetings from the method world!
Greetings from the method world!
Greetings from the method world!
```

The order of execution is worth noticing. The execution of the program happens by executing the lines of the main method (`main`) in order from top to bottom, one at a time. When the encountered statement is a method call, the execution of the program moves inside the method in question. The statements of the method are executed one at a time from top to bottom. After this the execution returns to the place where the method call occured, and then proceeds to the next statement in the program.

```java
import java.util.Scanner;

public class ProgramStructure {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // program code
        System.out.println("Let`s try if we can travel to the method world:");
        greet();

        System.out.println("Looks like we can, let`s try again::");
        greet();
        greet();
        greet();
    }

    // omat metodit
    public static void greet() {
        System.out.println("Greetings from the method world!");
    }
}
```

Name

Output

Prev 1 / 25 Next

Strictly speaking, the main program (`main`) itself is a method. When the program starts, the operating system calls `main`. The main method is the starting point for the program, since the execution begins from its first line. The execution of a program ends at the end of the main method.

Programming exercise:

In a hole in the ground

Points 1/1

Create a method called `printText` which prints the phrase "In a hole in the ground there lived a method" and a newline.

```java
public static void main(String[] args) {
    printText();
}
```

```
public static void printText() {
    // Write some code in here
}
```

The output of the program:

Sample output

```
In a hole in the ground there lived a method
```

Exercise submission instructions

How to see the solution

Programming exercise:

Reprint

Points

1/1

Expand the previous program so that the main program asks the user for the number of times the phrase will be printed (i.e. how many times the method will be called).

```
public static void main(String[] args) {
    // ask the user for the number of times that the phrase will be printed
    // use the while command to call the method a suitable number of times
}

public static void printText() {
    // write some code in here
}
```

Sample output:

Sample output

```
How many times?
7
In a hole in the ground there lived a method
In a hole in the ground there lived a method
In a hole in the ground there lived a method
In a hole in the ground there lived a method
In a hole in the ground there lived a method
```

```
In a hole in the ground there lived a method
In a hole in the ground there lived a method
```

NB: print the prompt `How many times?` on its own separate line!

Exercise submission instructions

How to see the solution

From here on out, when introducing methods, we will not explicitly mention that they must be located in the correct place. Methods cannot be defined e.g. inside other methods.

On Naming Methods

The names of methods begin with a word written entirely with lower-case letters, and the rest of the words begin with an upper-case letter - this style of writing is known as camelCase. Additionally, the code inside methods is indented by four characters.

In the code example below the method is poorly named. It begins with an upper-case letter and the words are separated by _ characters. The parentheses after the method name have a space between and indentation in the code block is incorrect.

```
public static void This_method_says_woof ( ) {
System.out.println("woof");
}
```

In contrast the method below is correctly named: The name begins with a lower-case letter and the words are joined together with the camelCase style, meaning that each word after the first begins with an upper-case letter. The parentheses sit next to one another and the contents are correctly indented (the method has its own code block, so the indentation of the code is four characters).

```
public static void thisMethodSaysWoof() {
    System.out.println("woof");
}
```

Quiz:
Method name

Points:
1/1

The code below contains a method that has been named with an incorrect style.

```
public static void printcodingis_cool() {
    System.out.println("Coding is cool!");
}
```

Write here the name of the method corrected to the right style.

Your answer:

Answer

printCodingIsCool

Correct, the name of the method should be "printCodingIsCool".

Submitted Tries remaining: 1

Method Parameters

Parameters are values given to a method that can be used in its execution. The parameters of a method are defined on the uppermost line of the method within the parentheses following its name. The values of the parameters that the method can use are copied from the values given to the method when it is executed.

In the following example a parameterized method `greet` is defined. It has an `int` type parameter called `numOfTimes`.

```
public static void greet(int numOfTimes) {
    int i = 0;
    while (i < numOfTimes) {
        System.out.println("Greetings!");
        i++;
    }
}
```

We will call the method `greet` with different values. The parameter `numOfTimes` is assigned the value `1` on the first call, and `3` on the second.

```
public static void main(String[] args) {
    greet(1);
    System.out.println("");
    greet(3);
}
```

Sample output

```
Greetings!

Greetings!
Greetings!
Greetings!
```

Just like when calling the predefined method `System.out.println`, you can pass an expression as a parameter.

```
public static void main(String[] args) {
    greet(1 + 2);
}
```

Sample output

```
Greetings!
Greetings!
Greetings!
```

If an expression is used as a parameter for a method, the expression is evaluated prior to the method call. Above, the expression evaluates to `3` and the final method call is of the form `greet(3);`.

Programming exercise:

From one to parameter

Points
1/1

Create the following method in the exercise template: `public static void printUntilNumber(int number)`. It should print the numbers from one to the number passed as a parameter. Two examples of the method's usage are given below.

```
public static void main(String[] args) {
    printUntilNumber(5);
}
```

Sample output

```
1
2
3
```

```
4
5
```

```
public static void main(String[] args) {
    printUntilNumber(2);
}
```

Sample output

```
1
2
```

Exercise submission instructions

How to see the solution

Programming exercise:

From parameter to one

Points 1/1

Create the following method in the exercise template: `public static void printFromNumberToOne(int number)`. It should print the numbers from the number passed as a parameter down to one. Two examples of the method's usage are given below.

```
public static void main(String[] args) {
    printFromNumberToOne(5);
}
```

Sample output

```
5
4
3
2
1
```

```
public static void main(String[] args) {
```

```
    printFromNumberToOne(2);
}
```

Sample output

```
2
1
```

Exercise submission instructions

How to see the solution

Multiple Parameters

A method can be defined with multiple parameters. When calling such a method, the parameters are passed in the same order.

```
public static void sum(int first, int second) {
    System.out.println("The sum of numbers " + first + " and " + second + " is " + (first +
}
```

```
sum(3, 5);

int number1 = 2;
int number2 = 4;

sum(number1, number2);
```

Sample output

```
The sum of numbers 3 and 5 is 8
The sum of numbers 2 and 4 is 6
```

```
1  public class Example {
2      public static void main(String[] args) {
3          sum(3, 5);
4
5          int number1 = 2;
6          int number2 = 4;
7
8          sum(number1, number2);
9      }
```

```
10
11      public static void sum(int first, int second) {
12          System.out.println("" + first + " + " + second + " = " + (first+ second));
13      }
14  }
```

Output

Prev 1 / 14 Next

Programming exercise:

Division

Points 1/1

Write a method `public static void division(int numerator, int denominator)` that prints the result of the division of the numerator by the denominator. Keep in mind that the result of the division of the integers is an integer — in this case we want the result to be a floating point number.

```
public static void main(String[] args) {
    division(3, 5);
}
```

Sample output

```
0.6
```

Exercise submission instructions

How to see the solution

Programming exercise:

Divisible by three

Points
1/1

Write a method `public static void divisibleByThreeInRange(int beginning, int end)` that prints all the numbers divisible by three in the given range. The numbers are to be printed in order from the smallest to the greatest.

```
public static void main(String[] args) {
    divisibleByThreeInRange(3, 6);
}
```

Sample output

```
3
6
```

```
public static void main(String[] args) {
    divisibleByThreeInRange(2, 10);
}
```

Sample output

```
3
6
9
```

Exercise submission instructions

How to see the solution

Parameter Values Are Copied in a Method Call

As a method is called **the values of its parameters are copied**. In practice, this means that both the main method and the method to be called can use variables with the same name. However, changing the value of the variables inside the method does not affect the value of the variable in the main method that has the same name. Let's examine this behavior with the following program.

```
public class Example {
    public static void main(String[] args) {
        int min = 5;
        int max = 10;

        printNumbers(min, max);
        System.out.println();

        min = 8;

        printNumbers(min, max);
    }

    public static void printNumbers(int min, int max) {
        while (min < max) {
            System.out.println(min);
            min++;
        }
    }
}
```

The output of the program is:

Sample output

```
5
6
7
8
9

8
9
```

Beneath, you'll find the same program visualized step-by-step. Changing the values of the variables in the method printNumbers does not affect the values in the main method, even though they have the same names.

```
1  public class Example {
2      public static void main(String[] args) {
3          int min = 5;
4          int max = 10;
5
6          printNumbers(min, max);
7
8          min = 8;
9
10         printNumbers(min, max);
11     }
12
13     public static void printNumbers(int min, int max) {
```

main:3

Name	Value

```
14          while (min < max) {
15              System.out.println(min);
16              min++;
17          }
18      }
19  }
```

Output

So, method parameters are distinct from the variables (or parameters) of other methods, even if they had the same name. As a variable is passed to a method during a method call, the value of that variable gets copied to be used as the value of the parameter variable declared in the method definition. Variables in two separate methods are independent of one another.

To further demonstrate this point, let's consider the following example. We define a variable called `number` in the main method. That variable is passed as a parameter to the method `incrementByThree`.

```
// main program
public static void main(String[] args) {
    int number = 1;
    System.out.println("The value of the variable 'number' in the main program: " + number);
    incrementByThree(number);
    System.out.println("The value of the variable 'number' in the main program: " + number);
}

// method
public static void incrementByThree(int number) {
    System.out.println("The value of the method parameter 'number': " + number);
    number = number + 3;
    System.out.println("The value of the method parameter 'number': " + number);
}
```

The execution of the program produces the following output.

Sample output

```
The value of the variable 'number' in the main program: 1
The value of the method parameter 'number': 1
```

```
The value of the method parameter 'number': 4
The value of the variable 'number' in the main program: 1
```

When the variable `number` is incremented inside the method, there's no issue. This, however, is not reflected in the `number` variable of the main program. The `number` variable living in the main program is different from the `number` variable of the method.

```java
public class Example {

   public static void main(String[] args) {
      int number = 1;
      System.out.println("The value of the number in the main program: " + number);
      incrementByThree(number);
      System.out.println("The value of the number in the main program: " + number);
   }

   public static void incrementByThree(int number) {
      System.out.println("The value of the number in the  method: " + number);
      number = number + 3;
      System.out.println("The value of the number in the  method: " + number);
   }
}
```

Output

The parameter `number` is copied for the method's use, i.e., a new variable called `number` is created for `incrementByThree` method, to which the value of the variable `number` in the main program is copied during the method call. The variable `number` inside the method `incrementByThree` exists only for the duration of the method's execution and has no relation to the variable of the same name in the main program.

Quiz:
Variable and method

Points:
1/1

What does the program below print?

```
public static void main(String[] args) {
    int number = 10;
    modifyNumber(number);
    System.out.println(number);
}

public static void modifyNumber(int number) {
    number = number - 4;
}
```

Select the correct answer

- ✓ 10
- 6
- 4
- -6
- 14

The answer is correct

Submitted Tries remaining: 1

Methods Can Return Values

The definition of a method tells whether that method returns a value or not. If it does, the method definition has to include the type of the returned value. Otherwise the keyword `void` is used in the definition. The methods we've created so far have been defined with the keyword `void`, meaning that they have not returned values.

```
public static **void** incrementByThree() {
    ...
}
```

The keyword `void` means that the method returns nothing. If we want the method to return a value, the keyword must be replaced with the type of the return variable. In the following example, there is a method called `alwaysReturnsTen` which returns an integer-type (`int`) variable (in this case the value 10).

To actually return a value, we use the command `return` followed by the value to be returned (or the name of the variable whose value is to be returned).

```
public static int alwaysReturnsTen() {
    return 10;
}
```

The method defined above returns an `int`-type value of `10` when called. For the return value to be used, it must be stored in a variable. This is done the same way as regular value assignment to a variable, by using an equals sign.

```
public static void main(String[] args) {
    int number = alwaysReturnsTen();

    System.out.println("the method returned the number " + number);
}
```

The return value of the method is placed in an `int` type variable as with any other `int` value. The return value can also be used in any other expression.

```
double number = 4 * alwaysReturnsTen() + (alwaysReturnsTen() / 2) - 8;

System.out.println("the result of the calculation " + number);
```

All the variable types we've encountered so far can be returned from a method.

Type of return value	Example

Method returns no value	`public static void methodThatReturnsNothing() {`
	`    // method body`
	`}`

Method returns `int` type variable	`public static int methodThatReturnsInteger() {`
	`    //method body, return statement must be included`
	`}`

Method returns `double` type variable	`public static double methodThatReturnsFloatingPointNumber() {`
	`    // method body, return statement must be included`
	`}`

Programming exercise:
Number uno

Points
1/1

Write a method `public static int numberUno()` that returns the value 1.

Exercise submission instructions

How to see the solution

Programming exercise:
Word

Points
1/1

Write a method `public static String word()`. The method must return a string of your choice.

Exercise submission instructions

How to see the solution

When execution inside a method reaches the command `return`, the execution of that method ends and the value is returned to the calling method.

The lines of source code following the command `return` are never executed. If a programmer adds source code after the return to a place which can never be reached during the method's execution, the IDE will produce an error message.

For the IDE, a method such as the following is faulty.

```
public static int faultyMethod() {
    return 10;
    System.out.println("I claim to return an integer, but I don't.");
}
```

The next method works since it is possible to reach every statement in it — even though there is source code below the return command.

```
public static int functioningMethod(int parameter) {
    if (parameter > 10) {
        return 10;
    }

    System.out.println("The number received as parameter is ten or less.");

    return parameter;
}
```

If a method has the form `public static void nameOfMethod()` it is possible to return from it — in other words, to stop its execution in that place — with the `return` command that is not followed by a value. For instance:

```
public static void division(int numerator, int denominator) {
    if (denominator == 0) {
        System.out.println("Can not divide by 0!");
        return;
    }

    System.out.println("" + numerator + " / " + denominator + " = " + (1.0 * numerator / der
}
```

Defining Variables Inside Methods

Defining variables inside methods is done in the same manner as in the "main program". The following method calculates the average of the numbers it receives as parameters. Variables `sum` and `avg` are used to help in the calculation.

```
public static double average(int number1, int number2, int number3) {
    int sum = number1 + number2 + number3;
    double avg = sum / 3.0;

    return avg;
}
```

One way to call the method is as follows.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the first number: ");
    int first = Integer.valueOf(scanner.nextLine());

    System.out.print("Enter the second number: ");
    int second = Integer.valueOf(scanner.nextLine());

    System.out.print("Enter the third number: ");
    int third = Integer.valueOf(scanner.nextLine());

    double averageResult = average(first, second, third);

    System.out.print("The average of the numbers: " + averageResult);
}
```

Variables defined in a method are only visible inside that method. In the example above, this means that the variables `sum` and `avg` defined inside the method `average` are not visible in the main program. A typical mistake while learning programming is to try and use a method in the following way.

```
public static void main(String[] args) {
    int first = 3;
    int second = 8;
    int third = 4;

    average(first, second, third);

    // trying to use a method's internal variable, DOES NOT WORK!
    System.out.print("The average of the numbers: " + avg);
}
```

In the above example, an attempt is made to use the variable `avg` that has been defined inside the method `average` and print its value. However, the variable `avg` only exists inside the method `average`, and it cannot be accessed outside of it.

The following mistakes are also commonplace.

```
public static void main(String[] args) {
    int first = 3;
    int second = 8;
    int third = 4;

    // trying to use the method name only, DOES NOT WORK!
    System.out.print("The average of the numbers: " + average);
}
```

Above, there is an attempt to use the name of the method `average` as if it were a variable. However, a method has to be called.

As well as placing the method result into a helper variable, another way that works is to execute the method call directly inside the print statement:

```
public static void main(String[] args) {
    int first = 3;
    int second = 8;
    int third = 4;

    // calling the method inside the print statement, DOES WORK!
    System.out.print("The average of the numbers: " + average(first, second, third));
}
```

Here, the method call occurs first returning the value 5.0, which is then printed with the help of the print statement.

Quiz:
Variables in method

Points:
1/1

What does the following program print as the last line?

```
public static void main(String[] args) {
    int first = 5;
    int second = 10;

    beginningToMiddle(first, second);

    System.out.println(first);
}
```

```
public static void beginningToMiddle (int start, int end) {
    int middle = (start + end)/2;
    while (start < middle) {
        System.out.println("step");
        start++;
    }
}
```

Select the correct answer

2

10

7

✓ 5

The answer is correct

Submitted Tries remaining: 1

Calculating the Return Value Inside a Method

The return value does not need to be entirely pre-defined - it can also be calculated. The return command that returns a value from the method can also be given an expression that is evaluated before the value is returned.

In the following example, we'll define a method sum that adds the values of two variables and returns their sum. The values of the variables to be summed are received as method parameters.

```
public static int sum(int first, int second) {
    return first + second;
}
```

When the execution of the method reaches the statement `return first + second;`, the expression `first + second` is evaluated, and then its value is returned.

The method is called in the following way. Below, the method is used to add the numbers 2 and 7 together. The value resulting from the method call is placed into the

variable `sumOfNumbers`.

```
int sumOfNumbers = sum(2, 7);
// sumOfNumbers is now 9
```

Let's expand the previous example so that the numbers are entered by a user.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the first number: ");
    int first = Integer.valueOf(scanner.nextLine());

    System.out.print("Enter the second number: ");
    int second = Integer.valueOf(scanner.nextLine());

    System.out.print("The combined sum of the numbers is: " + sum(first, second));
}

public static int sum(int first, int second) {
    return first + second;
}
```

In the example above, the method's return value is not stored in a variable but is instead directly used as part of the print operation. The print command's execution is done by the computer first evaluating the string `"The combined sum of the numbers is: "+ sum(first, second)`. The computer first looks for the variables `first` and `second` and copies their values as the values of the method `sum`'s parameters. The method then adds the values of the parameters together, after which it returns a value. This value takes the place of the `sum` method call, whereby the sum is appended to the string `"The combined sum of the numbers is: "`.

Since the values passed to a method are copied to its parameters, the names of the parameters and the names of the variables defined on the side of the caller have, in fact, nothing to do with each other. In the previous example, both the variables of the main program and the method parameters were named the same (`first` and `second`) "by accident". The code below will function in precisely the same manner even though the variables are named differently:

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the first number: ");
    int number1 = Integer.valueOf(scanner.nextLine());

    System.out.print("Enter the second number: ");
    int number2 = Integer.valueOf(scanner.nextLine());

    System.out.print("The total sum of the numbers is: " + sum(number1, number2));
```

```
}

public static int sum(int first, int second) {
    return first + second;
}
```

Now the value of the variable `number1` is copied as the value of the method parameter `first`, and the value of the variable `number2` is copied as the value of the parameter `second`.

Quiz:
Variable and method 2

Points:
1/1

What does the following program print?

```
public static void main(String[] args) {
    int number = 3;
    modifyNumber(number);
    System.out.println(addAndReturn(number));
}

public static void modifyNumber(int number) {
    number = number + 2;
}

public static int addAndReturn(int number) {
    return number + 10;
}
```

- 3
- 1
- 5
- 15
- -7

✓ 13

-9

The answer is correct

The answer is correct

MOOC opetusvideo: metodit jotka palauttavat arvon

Programming exercise:

Summation

Points

1/1

Expand the method `sum` in the exercise template so that it calculates and returns the sum of the numbers that are given as the parameters.

Create the method using the following structure:

```
public static int sum(int number1, int number2, int number3, int number4) {
    // write your code here
    // remember to include return (at the end)!
}

public static void main(String[] args) {
    int answer = sum(4, 3, 6, 1);
```

```
    System.out.println("Sum: " + answer);
}
```

The output of the program:

Sample output

```
Sum: 14
```

NB: when an exercise describes a method that should *return* something, this means that the type of the return value must be declared in the method definition, and that the method contains a `return` command that returns the wanted data. The method itself will print nothing (i.e. will not use the command `System.out.println`) - that task is left to the method caller, which in this case is the main program.

Exercise submission instructions

How to see the solution

Programming exercise:

Smallest

Points 1/1

Define a two-parameter method `smallest` that returns the smaller of the two numbers passed to it as parameters.

```
public static int smallest(int number1, int number2) {
    // write your code here
    // do not print anything inside the method

    // there must be a return command at the end
}

public static void main(String[] args) {
    int answer =  smallest(2, 7);
    System.out.println("Smallest: " + answer);
}
```

The output of the program:

Sample output

Smallest: 2

Exercise submission instructions

How to see the solution

Programming exercise:
Greatest

Points
1/1

Define a method called `greatest` that takes three numbers and returns the greatest of them. If there are multiple greatest values, returning one of them is enough. Printing will take place in the main program.

```
public static int greatest(int number1, int number2, int number3) {
  // write some code here
}

public static void main(String[] args) {
  int answer =  greatest(2, 7, 3);
  System.out.println("Greatest: " + answer);
}
```

The output of the program:

Sample output

Greatest: 7

Exercise submission instructions

How to see the solution

Programming exercise:
Averaging

Points
1/1

Create a method called `average` that calculates the average of the numbers passed as parameters. The previously created method `sum` must be used inside this method!

Define the method in the following template:

```
public static int sum(int number1, int number2, int number3, int number4) {
    // you can copy your implementation of the method sum here
}

public static double average(int number1, int number2, int number3, int number4) {
    // write your code here
    // calculate the sum of the elements by calling the method sum
}

public static void main(String[] args) {
    double result = average(4, 3, 6, 1);
    System.out.println("Average: " + result);
}
```

The output of the program:

Sample output

```
Average: 3.5
```

Make sure to remember how to convert an integer (`int`) into a decimal number (`double`)!

Exercise submission instructions

How to see the solution

Execution of Method Calls and the Call Stack

How does the computer remember where to return after the execution of a method?

The environment that executes Java source code keeps track of the method being executed in the call stack. **The call stack** contains frames, each of which includes information about a specific method's internal variables and their values. When a method is called, a new frame containing its variables is created in the call stack. When the execution of a method ends, the frame relating to a method is removed from the call stack, which leads to execution resuming at the previous method of the stack.

The right side of the visualization below displays the functioning of the call stack. When a method is called, a new frame is created in the stack, which is removed upon exit from the method call.

```
1  public class Example {
2     public static void main(String[] args) {
3        // program code
4        System.out.println("Let`s try if we can travel to the method world:");
5        greet();
6
7        System.out.println("Looks like we can, let`s try again:");
8        greet();
9        greet();
10       greet();
11    }
12
13    // omat metodit
14    public static void greet() {
15       System.out.println("Greetings from the method world!");
16    }
17 }
```

main:4
Name V

Output

Prev 1 / 24 Next

When a method is called, the execution of the calling method is left waiting for the execution of the called method to end. This can be visualized with the help of a call stack. The call stack refers to the stack formed by the method calls — the method currently being executed is always on the top of the stack, and when that method has finished executing the execution moves on to the method that is next on the stack. Let's examine the following program:

```
public static void main(String[] args) {
    System.out.println("Hello world!");
    printNumber();
    System.out.println("Bye bye world!");
}

public static void printNumber() {
```

```
    System.out.println("Number");
}
```

The execution begins from the first line of the `main` method when the program is run. The command on this line prints the text `"Hello world!"`. The call stack of the program looks as follows:

Sample output

```
main
```

Once the print command has been executed, we move on to the next command, which calls the method `printNumber`. Calling this method moves the execution of the program to the beginning of the method `printNumber`. Meanwhile, the `main` method will await for the execution of the method `printNumber` to end. While inside the method `printNumber`, the call stack looks like this:

Sample output

```
printNumber
main
```

Once the method `printNumber` completes, we return to the method that is immediately below the method `printNumber` in the call stack — which in this case is the method `main`. `printNumber` is removed from the call stack, and the execution continues from the line after the `printNumber` method call in the `main` method. The state of the call stack is now the following:

Sample output

```
main
```

Call Stack and Method Parameters

Let's examine the call stack in a situation where parameters have been defined for the method.

```
public static void main(String[] args) {
    int beginning = 1;
    int end = 5;

    printStars(beginning, end);
}

public static void printStars(int beginning, int end) {
    while (beginning < end) {
        System.out.print("*");
        beginning++; // same as beginning = beginning + 1
```

```
    }
}
```

The execution of the program begins on the first line of the `main` method. The next two lines create the variables `beginning` and `end`, and also assign values to them. The state of the program prior to calling the method `printStars`:

Sample output

```
main
  beginning = 1
  end = 5
```

When `printStars` is called, the `main` method enters a waiting state. The method call causes new variables `beginning` and `end` to be created for the method `printStars`, to which the values passed as parameters are assigned to. These values are copied from the variables `beginning` and `end` of the `main` method. The state of the program on the first line of the execution of the method `printStars` is illustrated below.

Sample output

```
printStars
  beginning = 1
  end = 5
main
  beginning = 1
  end = 5
```

When the command `beginning++` is executed within the loop, the value of the variable `beginning` that belongs to the method currently being executed changes.

Sample output

```
printStars
  beginning = 2
  end = 5
main
  beginning = 1
  end = 5
```

As such, the values of the variables in the method `main` remain unchanged. The execution of the method `printStart` would continue for some time after this. When the execution of that method ends, the execution resumes inside the `main` method.

Sample output

```
main
```

```
beginning = 1
end = 5
```

Let's observe the same program by visualizing its execution step-by-step. The application used for visualization grows the call stack downwards — on the right side, the method on top is always `main`, under which go the methods being called.

```
1  public class Example {
2
3     public static void main(String[] args) {
4        int beginning = 1;
5        int end = 5;
6
7        printStars(beginning, end);
8     }
9
10    public static void printStars(int beginning, int end) {
11       while (beginning < end) {
12          System.out.print("*");
13          beginning++;
14       }
15    }
16 }
```

main:4

Name	Value

Output

Call Stack and Returning a Value from a Method

Let's now study an example where the method returns a value. The `main` method of the program calls a separate `start` method, inside of which two variables are created, the `sum` method is called, and the the value returned by the `sum` method is printed.

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

```
public static void start() {
    int first = 5;
    int second = 6;

    int sum = sum(first, second);

    System.out.println("Sum: " + sum);
}

public static int sum(int number1, int number2) {
    return number1 + number2;
}
```

At the beginning of the `start` method's execution the call stack looks as in the following illustration since it was called from the `main` method. The method `main` has no variables of its own in this example:

Sample output

```
start
main
```

When the variables `first` and `second` have been created in the `start` method (i.e., the first two rows of that method have been executed), the situation is the following:

Sample output

```
start
  first = 5
  second = 6
main
```

The command `int sum = sum(first, second);` creates the variable `sum` in the method `start` and calls the method `sum`. The method `start` enters a waiting state. Since the parameters `number1` and `number2` are defined in the method `sum`, they are created right at the beginning of the method's execution, after which the values of the variables given as parametes are copied into them.

Sample output

```
sum
  number1 = 5
  number2 = 6
start
  first = 5
  second = 6
  sum // no value
main
```

The execution of the method `sum` adds together the values of the variables `number1` and `number2`. The command `return` returns the sum of the numbers to the method that is one beneath it in the call stack - the method `start` in this case. The returned value is set as the value of the variable `sum`.

Sample output

```
start
  first = 5
  second = 6
  sum = 11
main
```

After that, the print command is executed, and then we return to the `main` method. Once the execution reaches the end of the `main` method, the execution of the program ends.

```
1  public class Example {
2     public static void main(String[] args) {
3        start();
4     }
5
6     public static void start() {
7        int first = 5;
8        int second = 6;
9
10       int sum = sum(first, second);
11
12       System.out.println("Sum: " + sum);
13    }
14
15    public static int sum(int number1, int number2) {
16       return number1 + number2;
17    }
18 }
```

main:3

Name	Value

Output

Method Calling Another Method

As we noticed earlier, other methods can be called from within methods. An additional example of this technique is given below. We'll create the method `multiplicationTable` that prints the multiplication table of the given number. The multiplication table prints the rows with the help of the `printMultiplicationTableRow` method.

```
public static void multiplicationTable(int max) {
    int number = 1;

    while (number <= max) {
        printMultiplicationTableRow(number, max);
        number++;
    }
}

public static void printMultiplicationTableRow(int number, int coefficient) {

    int printable = number;
    while (printable <= number * coefficient) {
        System.out.print("  " + printable);
        printable += number;
    }

    System.out.println("");
}
```

The output of the method call `multiplicationTable(3)`, for instance, looks like this.

Sample output

```
  1  2  3
  2  4  6
  3  6  9
```

Below is a visualization of the method call `multiplicationTable(3)`. Notice how the information about the internal state of the calling method is stored in the call stack.

```
 1 public class Example {
 2     public static void main(String[] args) {
 3         multiplicationTable(3);
 4     }
 5
 6     public static void multiplicationTable(int max) {
 7         int number = 1;
 8
 9         while (number <= max) {
10             printMultiplicationTableRow(number, max);
11             number++;
```

main:
Name

```
        }
    }

    public static void printMultiplicationTableRow(int number, int coefficient) {

        int printable = number;
        while (printable <= number * coefficient) {
            System.out.print("  " + printable);
            printable += number;
        }

        System.out.println("");
    }
}
```

Output

Prev 1 / 74 Next

Programming exercise:

Star sign (4 parts)

Points 4/4

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

Part 1: Printing stars

Define a method called `printStars` that prints the given number of stars and a line break.

Write the method in the following template:

```
public static void printStars(int number) {
    // you can print one star with the command
    // System.out.print("*");
    // call the print command n times
    // in the end print a line break with the comand
    // System.out.println("");
}
```

```
public static void main(String[] args) {
    printStars(5);
    printStars(3);
    printStars(9);
}
```

The output of the program:

Sample output

```
*****
***
*********
```

N.B multipart exercises can be uploaded to the server (click the button to the right of the testing button) even if some parts are unfinished. In this case the server will complain about the tests for the parts that haven't been completed, but it will mark down the finished parts.

Part 2: Printing a square

Define a method called `printSquare(int size)` that prints a suitable square with the help of the `printStars` method. So the method call `printSquare(4)` results in the following output:

Sample output

```
****
****
****
****
```

N.B.: producing the correct output is not enough; the rows of the square must be produced by calling the `printStars` method inside the `printSquare`method.

When creating the program, you can use the code in the main to test that the methods behave as required.

Part 3: Printing a rectangle

Write a method called `printRectangle(int width, int height)` that prints the correct rectangle by using the `printStars` method. So the method call `printRectangle(17, 3)` should produce the following output:

Sample output

```
*****************
```

```
*****************
*****************
```

Part 4: Printing a triangle

Create a method called `printTriangle(int size)` that prints a triangle by using the `printStars` method. So the call `printTriangle(4)` should print the following:

Sample output

```
*
**
***
****
```

Exercise submission instructions

How to see the solution

Programming exercise:
Advanced astrology (3 parts)

Points
3/3

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

Part 1: Printing stars and spaces

Define a method called `printSpaces(int number)` that produces the number of spaces specified by `number`. The method does not print the line break.

You will also have to either copy the `printStars` method from your previous exercise or reimplement it in this exercise template.

Part 2: Printing a right-leaning triangle

Create a method called `printTriangle(int size)` that uses `printSpaces` and `printStars` to print the correct triangle. So the method call `printTriangle(4)` should print the following:

Sample output

```
   *
  **
 ***
****
```

Part 3: Printing a Christmas tree

Define a method called `christmasTree(int height)` that prints the correct Christmas tree. The Christmas tree consists of a triangle with the specified height as well as the base. The base is two stars high and three stars wide, and is placed at the center of the triangle's bottom. The tree is to be constructed by using the methods `printSpaces` and `printStars`.

For example, the call `christmasTree(4)` should print the following:

Sample output

```
   *
  ***
 *****
*******
  ***
  ***
```

The call `christmasTree(10)` should print:

Sample output

```
         *
        ***
       *****
      *******
     *********
    ***********
   *************
  ***************
 *****************
*******************
        ***
        ***
```

NB: heights shorter than 3 don't have to work correctly!

Exercise submission instructions

How to see the solution

You have reached the end of this section! Continue to the next section:

→ 5. End questionnaire

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

Source code of the material

This course is created by the Agile Education Research -research group of the University of Helsinki.

Credits and about the material.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI