Part 1

Printing

Learning Objectives

- Learn to write a program that prints text.
- Become familiar with executing programs.
- Know what the term "parameter" means.

The print command `System.out.println("Hello world");` prints the text "Hello world".

```
System.out.println("Hello world!");
```

Sample output

```
Hello world!
```

In this material, text boxes like the one above demonstrate an output produced by the example code. Accordingly, the above program would produce the print output "Hello World!". You can try any of these examples in the exercise template named "Sandbox", which you will find in the programming environment.

You can print any text you want with the command, as long as the command `System.out.println("arbitrary text");` — i.e., `System` dot `out` dot `println` open parenthesis `(` "the text" close parenthesis `)` and semicolon `;` remains unchanged. The command below will print the te
"Hello there!".

```
System.out.println("Hello there!");
```

Sample output

```
Hello there!
```

Program Boilerplate

In Java, our programs have to include some boilerplate code to function. This boilerplate, an example of which is shown below, for example tells the computer what your program is called. Below, the name of the program is `Example`. This name has to correspond to the name of the file that contains the source code (e.g. `Example.java`).

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

        System.out.println("Text to be printed");

    }
}
```

Execution of the program starts from the line that follows `public static void main(string[] args) {`, and ends at the closing curly bracket `}`. Commands are executed one line at a time. We'll learn what the terms `public class` and `public static void` mean later on. In the above example, `System.out.println("Text to be printed")` is the only command to be executed. Its output is:

Sample output

```
Text to be printed
```

Examples in the Material and Code Templates

The examples in the material will not always show the template, but you can assume that your program file always needs one. As such, the

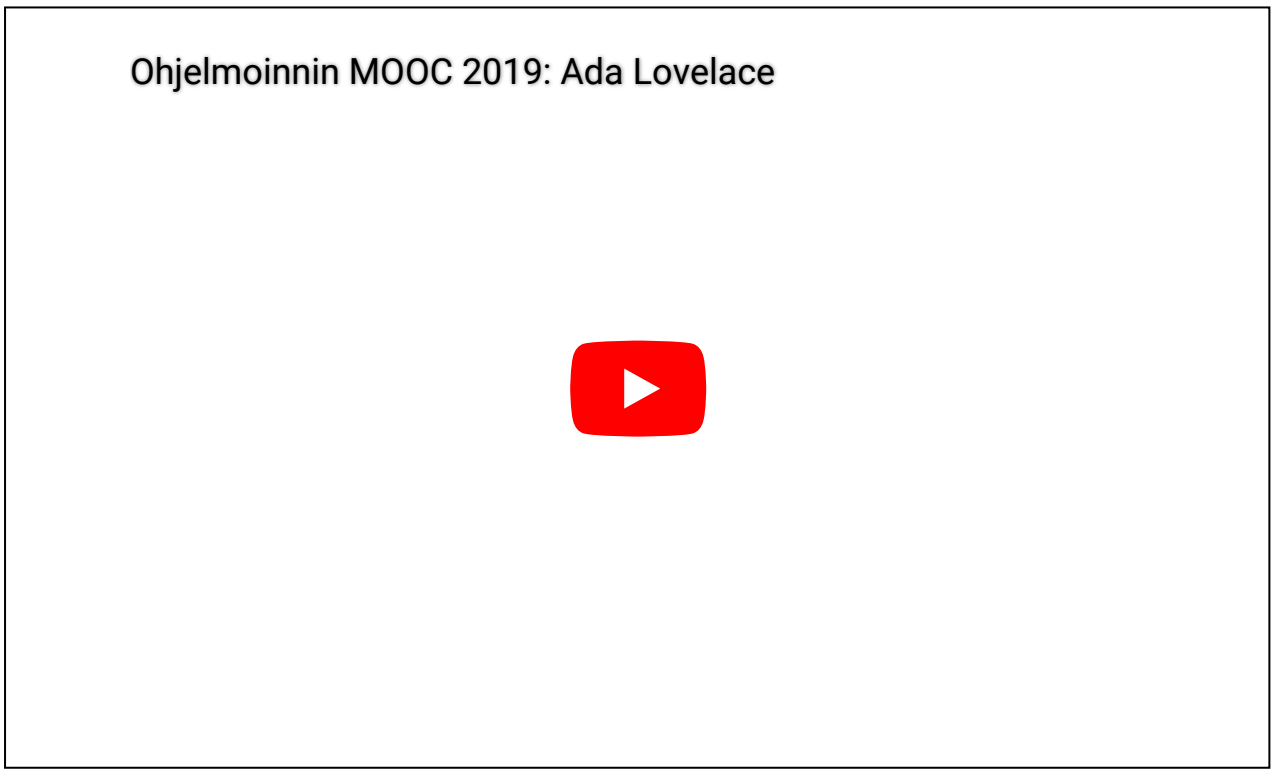
examples might be as short as a single line, such as the example below that illustrates the print command.

```
System.out.println("Hello world");
```

In reality, the above example, when written as a full Java program, looks like so:

```
public class Example {
    public static void main(String[] args) {
        // Here goes the statements used by the program
        System.out.println("Hello world!");
    }
}
```

Here's the second programming exercise of this course. If you'd like, you can watch this video on how to solve the exercise first:



Programming exercise:

Ada Lovelace

Points
1/1

The exercise template has the following boilerplate code:

```
public class AdaLovelace {
    public static void main(String[] args) {
        // Write your program here

    }
}
```

The line "// Write your program here" is a *line comment*, and the computer will ignore it when executing the program. Add a new line below the line comment that prints the string "Ada Lovelace" and run the program. The output of the program should be:

Sample output

```
Ada Lovelace
```

Once you've finished the exercise and see that it prints the correct string, return the exercise to the TMC server. After that, you can read more about Ada Lovelace, who was one of the first programmers.

Exercise submission instructions

How to see the solution

Running the Program

You can run a program in TMC by pressing the green play button, or by selecting "Run project" from the TMC menu.

Even though running the program is straightforward, a lot is happenings behind the scenes. When a program is run, the source code is first compiled into Java bytecode. This compilation process is done by Java's compiler, which itself is a program. Following that, the program gets executed, meaning the commands are executed one-by-one by a Java-interpreter that is able to read Java bytecode.

This compile process affects how and when errors occur. When a program is compiled before execution, the compiler can search for errors in it. This also affects the hints provided by the IDE, and in this way, the programmer can receive immediate feedback on any errors.

The IDE both compiles and executes the program with just one press of a button. However, the programming environment compiles the program continuously, so it can report errors. You can, for example, try to change above Ada Lovelace exercise print command to `Systemoutprintln("hi!")` — what you'll notice is that the line will be underlined and you'll be notified of an error on the left-hand side.

Printing Multiple Lines

Programs are constructed command-by-command, where each command is placed on a new line. In the example below, the command `System.out.println` appears twice, which means that two print commands are being executed in the program.

```
public class Ohjelma {
    public static void main(String[] args) {
        System.out.println("Hello world!");
        System.out.println("... and the universe!");
    }
}
```

The program above will print:

Sample output

```
Hello world!
... and the universe!
```

Exact Inspector

The programming exercises will be checked by TMC Henrik, who is meticulous. The guidelines in the assignments regarding the print format are very precise. If the assignment expects you to print a parenthesis, you must print the parenthesis.

This preciseness with regard to the output is relevant in programming in a more general sense. Missing a single character may cause an error. Novice programmers often enter a comma instead of a dot, and write, for instance `printin` instead of `println`, leave out apostrophes, or forget the semicolon after a command. Any one of these would cause an error and cause the program execution to fail.

Learning programming is, in fact, a path full of mistakes — and every error message is a chance to learn. Keep a look out for any red signs in the IDE and try to read the test errors!

Programming exercise:

Once Upon a Time

Points 1/1

The exercise template comes with the following template:

```
public class OnceUponATime {
    public static void main(String[] args) {
        // Write your program here

    }
}
```

Modify the program so that it will print the following text. Use three `System.out.println`-commands for printing.

Sample output

Once upon a time
there was
a program

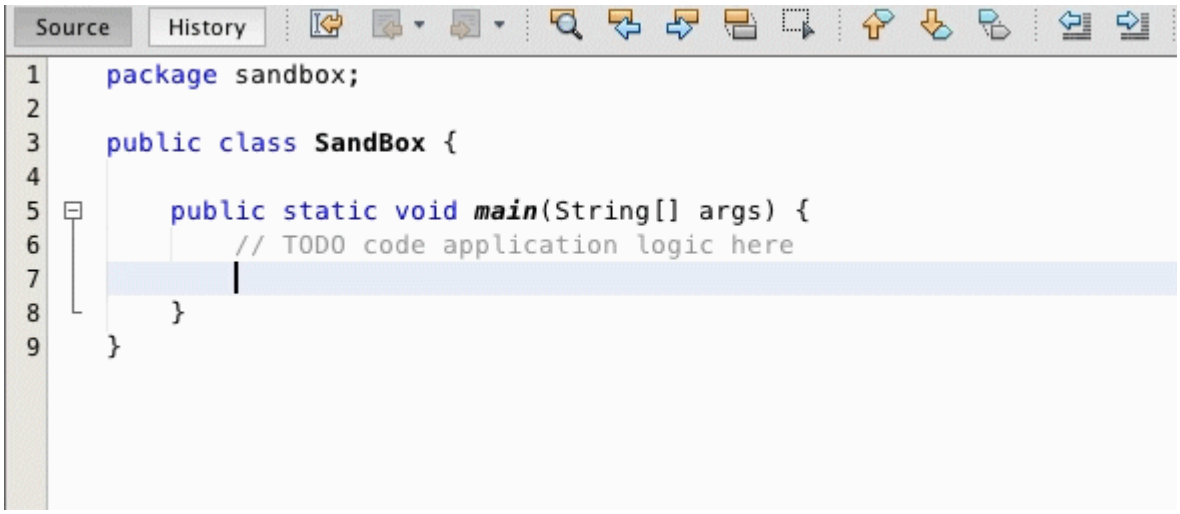
Exercise submission instructions

How to see the solution

"sout"

Writing the command `System.out.println("...") can be taxing. In NetBeans, try to write **sout** on blank line (within main) and press tab (the key left to q). What happens? This shortcut may save you a lot of time in the future.

The animation below illustrates the use of the sout command. The user first writes sout and then pressed tab. Magic!



Programming exercise:

Dinosaur

Points
1/1

The exercise comes ready with the following template:

```
public class Dinosaur {
    public static void main(String[] args) {
        // Write your program here
    }
}
```

Edit the program so that it will print the following text. Use the "sout"-shorthand for writing the print commands.

Sample output

```
Once upon a time
there was
a dinosaur
```

Exercise submission instructions

How to see the solution

Terminology and Code Comments

Command parameters

The information to be printed by the print command, i.e. its **parameters**, are passed to it by placing them inside the parentheses `()` that follow the command. For example, passing `Hi` as a parameter to the `System.out.println` command is done like this: `System.out.println("Hi")`.

Semicolon Separates Commands

Commands are separated with a semicolon `;`. We could, if we wanted to, write almost everything on a single line. However, that would be difficult to understand.

```
System.out.println("Hello "); System.out.println("world"); System.out.println("
```

Sample output

```
Hello
world
!
```

Although the previous example works, it's important to be considerate of other programmers (and your future self!) and to use line breaks. That way, anyone reading the program knows that each line does only a single concrete thing.

Comments

Source code can be commented to clarify it or to add notes. There are two ways to do this.

- Single-line comments are marked with two slashes `//`. Everything following them on the same line is interpreted as a comment.
- Multi-line comments are marked with a slash and an asterisk `/*`, and closed with an asterisk followed by a slash `*/`. Everything between them is interpreted as a comment.

Below is an example of a program where both are used.

```
public class Comments {
    public static void main(String[] args) {
        // Printing
        System.out.println("Text to print");
        System.out.println("More text to print!");
        /* Next:
        - more on printing
```

```
        - more practice
        - variables
        - ...
        */
        System.out.println("Some other text to print");
        // System.out.println("Trying stuff out")
    }
}
```

The last line of the example shows a particularly handy use-case for comments. Code that has been written does not need to be deleted to try out something else.

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

→ 3. Reading input

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

In this part:

1. Getting started with programming
2. Printing
3. Reading input
4. Variables
5. Calculating with numbers
6. Conditional statements and conditional operation
7. Programming in our society

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