First program

Traditionally, the exemplary text "Hello, world!" is presented as the first program. Its task is to display the following text on the screen:

```
Hello, World!
```

Below, the (source) code of this program saved in Java is presented.

```
// MyFirstJavaProgram.java

public class MyFirstJavaProgram {            // (1)

  public static void main(String[] args) {   // (2)
    System.out.println("Hello, World!");       // (3)
  }
}
```

Does it look complicated? Below we will discuss step by step what is happening in the individual lines.

- *(1)* `MyFirstJavaProgram` is the name of our program. Besides, it is also the name of the file where the program, i.e. `MyFirstJavaProgram.java` (a file with `.java` extension), is located. At the moment we will skip the explanation of the words `public` and `class`; I will only mention that `MyFirstJavaProgram` is also the name of a class. Classes are basic entities in Java, and each program consists of one or more classes (later on, you will learn more on what the class is).
- *(2)* `public static void main(String[] args)` – this fragment has a special "power". It indicates the declaration of the `main` method that is responsible for running the entire program; everything starts here. For now, treat this line as a mantra. In our future programs, in this very method we will be placing the Java code that we are going to run.
- *(3)* `System.out.println()` is responsible for writing the text on

the screen, which in this case is: **Hello, World!**.

> ***Note:*** CamelCase notation — if a name consists of several words, we type it without space, and each word begins with a capital letter, for instance MyFirstJavaProgram, AccountService or BankAccount. It has also been adopted that all elements such as classes, methods or variables are named in English.

Let us go back to our example again.

```
// MyFirstJavaProgram.java

public class MyFirstJavaProgram {           // (1)

  public static void main(String[] args) {  // (2)
    System.out.println("Hello, World!");    // (3)
  }                                         // (4)
}                                           // (5)
```

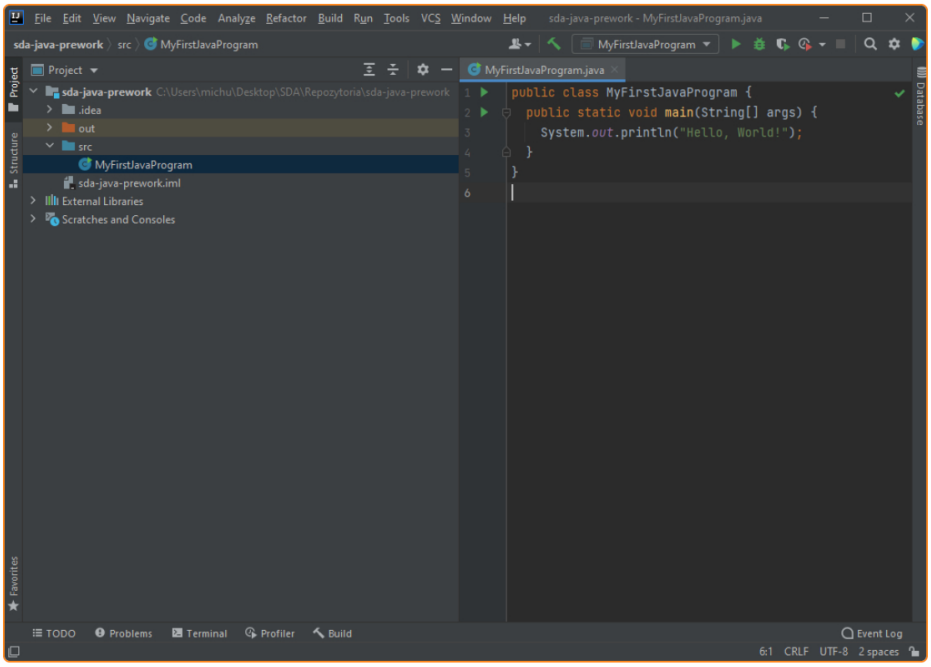
- *(1)* We already know that `MyFirstJavaProgram` is the name of the file and class that we have created. Please notice the opening brace ({) that defines the beginning of the **class body**. The end of the class body is defined by the final brace (}).
- *(2)* Notice the opening brace at the end of the line—it defines the beginning of the **method body**.
- *(3)* `System.out.println()` is an instruction. Is means invoking the `println()` method on the out object (more information on this topic can be found in further part of this Handbook).
  The `println()` method displays the content included between the quotes. Please also note the semicolon at the end of the instruction.
- *(4)* The brace ending the `main` method body
- *(5)* The brace ending the `MyFirstJavaProgram` class body

> ***Note:*** Please notice the way of formatting the code and indentation that are used for fast identification of blocks and analysis of control flow. They have no meaning for the computer, but they make the program more legible for the programmer.

Working with IntelliJ IDEA

Now let us see how to save and run this program in a Java environment. We will use the IntelliJ IDEA program.

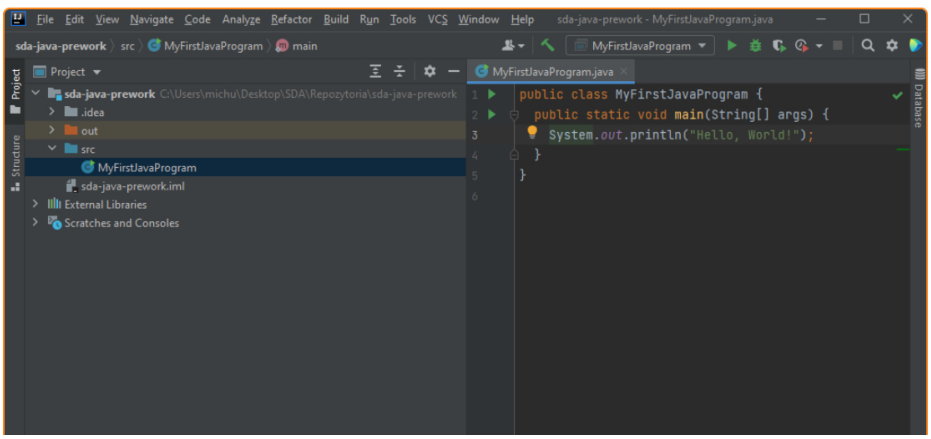
- click the `src` directory and then right-click to select **New › Java Class** from the menu
- in the *Name* field, enter `MyFirstJavaProgram` (without `.java`) and click *OK*
- rewrite the above code snippet



Then run the program using one of the following methods:

- Select **Run › Run...** from the top menu and select the `MyFirstJavaProgram`. You can also use a keyboard shortcut: *Alt* + *Shift* + *F10*
- Click the green triangle located on the left at the height of `public class MyFirstJavaProgram` or `public static void main(String[] args)` and select *Run MyFirstJavaProgram.main()*

In the window below the code, you should see the text `Hello, World!`



> ***Note:*** In order to improve the efficiency of your work, it is worth mastering at least basic keyboard shortcuts. I encourage you to download the full list of keyboard shortcuts from IntelliJ IDEA site. You can also use *Key Promoter X* plugin to learn and master keyboard shortcuts. More information can be found here.

Working on the console

This Handbook focuses on writing and running programs in the IDE – IntelliJ IDEA. However, many books, studies and websites provide examples of Java programs operating from the terminal level. Therefore, I decided to discuss this way here as well.

> ***Note:*** All examples presented in this Handbook can successfully run on the console.

If you want to run programs from the console level, you can write them using one of the following editors:

- Sublime Text
- Visual Studio Code
- jEdit

Creating a class

Create a directory on the hard drive to store your Java programs. Using the selected editor create the `MyFirstJavaProgram.java` file in this directory, with the following content:

```
// MyFirstJavaProgram.java

public class MyFirstJavaProgram {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

Compilation

Activate the terminal and navigate to the directory where `MyFirstJavaProgram.java` file is located. Using the `javac` command, compile the class.

```
javac MyFirstJavaProgram.java
```

As a result of this operation, the `MyFirstJavaProgram.class` file is created. There should be two files in the directory now:

- `MyFirstJavaProgram.java` – the source code
- `MyFirstJavaProgram.class` – the compiled class with the byte code (*bytecode*)

Activation

Now it is time to run the compiled code using the `java` command.

```
java MyFirstJavaProgram
```

The following text should appear in the terminal window:

```
Hello, World!
```

The `javac` command is used to compile. To do this, enter a filename, e.g. `javac App.java`. The java command is used to run a program, e.g. `java App`.

Developing your first program

Now we will try to display something else on the screen so that you can see how Java's instructions are invoked. Let us add text `Hello, Java!` to the program.

```
// MyFirstJavaProgram.java

public class MyFirstJavaProgram {

  public static void main(String[] args) {
    System.out.println("Hello, World!");
    System.out.println("Hello, Java!");
  }
}
```

The result of program execution should be as below:

```
Hello, World!
Hello, Java!
```

You have learned two things now. The first one is that Java executes instructions from top to bottom: the first message displayed was `Hello, World!`, followed by `Hello, Java!`. Besides, now you know more about the `println()` method operation. This method displays the text passed to it and "goes to a new line", i.e., adds a new line sign to the displayed text. Is there a method that can **only** display the text without going to a new line? Of course there is. Let us try to remove `ln` from the *print***ln()** name to invoke the `print()` method.

```
// MyFirstJavaProgram.java

public class MyFirstJavaProgram {

  public static void main(String[] args) {
    System.out.print("Hello, World!");
    System.out.print("Hello, Java!");
  }
}
```

When the program is run, the displayed result should be as below:

```
Hello, World!Hello, Java!
```

> ***Note:*** The `println()` and `print()` methods are very useful and offer more possibilities. Later you will learn that more than a text can be passed to them.

What if you would like to separate these two strings with an empty line? For example, you can invoke the `println()` method without passing any text to it. Then the method will have no text to display, but it will go to a new line anyway.

```
// MyFirstJavaProgram.java

public class MyFirstJavaProgram {

  public static void main(String[] args) {
    System.out.print("Hello, World!");
    System.out.println();               // (1)
```

```
    System.out.print("Hello, Java!");
  }
}
```

- *(1)* invoking the `println()` method without passing a text to display; the method does not display anything but only goes to a new line.

The result of this program will be as below:

```
Hello, World!

Hello, Java!
```

Summary

What have you learned in this chapter?

What is the history of Java?

The history of Java began in 1991. Engineers from the Unix circle created a "light" programming language independent of the hardware platform. Initially, the language was named `Oak`, but then it was changed to `Java`. Since 2010, Java has been developed by Oracle.

What are the basic assumptions of the language?

The basic assumptions are: *simplicity, object-orientation, reliability, networking, independence of architecture, portability, high efficiency and multithreading*.

What is a source code?

A source code is the result of the programmer's work. These are the operations to be executed by the computer. In Java, you place a source code in text files with `.java` extension.

What is a compiler?

A compiler is a program that converts (and at the same time, optimizes and detects errors) the *source code* into a code written in other language. In Java, a compiler processes files with `.java` extension and places the result in `.class` files.

What is a bytecode?

It is the result of compiling a program written in Java.

What is an IDE and which ones are now popular?

IDE means an **I**ntegrated **D**evelopment **E**nvironment. It is used to create, test,

build and run programs. It offers many capabilities that can be expanded with a plugin mechanism. Three environments are now popular: **IntelliJ IDEA**, Eclipse and NetBeans.

Do I need an IDE to write a program in Java?

No, you can use your favorite text editor. You can build an application using the `javac` command and run it using `java`.

How can I check which Java version has been installed on my computer?

You can do this, for example, by using the console and entering the `java --version` command.

What is the name of the method that runs the program in Java?

This method is `public static void main(String[] args)`.

What is the operation of `println()` and `print()` methods?

Both methods display the text passed to them on the screen. However, the `println()` method goes to a new line after the text is displayed.

How can I display an empty line?

There are several ways to do this. You have learned one of them: you can invoke the `println()` method without passing a text to it.

What are the basic rules for naming in Java?

It is assumed to use English to name classes, fields and variables in Java. A class name starts with a capital letter, while other elements start with a lowercase letter. If the name consists of several words, we use the so-called *CamelCase* notation: each subsequent word of the name starts with a capital letter.

Exercises

- Write a program (create a new class using the `main` method) to display `Hello, <Your Name>!` instead of `Hello, World!` on the screen.
- Write a program that displays your business card like the one below:

```
########################
# #
# John Smith #
# Sesame Street 345/7b #
# New York 10019 #
# #
########################
```

- use different characters, for instance:
    - – for horizontal lines
    - | for vertical lines
    - / and \ (or +) for corners
- Test the compiling and running the class from the console
    - create a new directory
    - create a `PrintHello` class (in the file named `PrintHello.java`) in the new directory:

```
public class PrintHello {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

- compile the class using `javac` command:

```
javac PrintHello.java
```

- run the `PrintHello` class by using the `java` command:

```
java PrintHello
```

- Check whether the text like below is displayed:

```
Hello, WOrld!
```

- Remove one of the quotes around the Hello, World! text:

```
public class PrintHello {
  public static void main(String[] args) {
    System.out.println("Hello, World!);
  }
}
```

- try to compile the program...
- check whether the following error appeared:

```
PrintHello.java:4: error: unclosed string literal
System.out.println("Hello, World!);

1 error
```

- correct the quote and try to "damage" the program in some other way (each time try to compile and pay attention to the errors returned):
    - remove or add brace { or }
    - remove ;
    - remove or add a letter to one of the methods
    - change the class name (without changing the file name)

Complete Lesson