Lab session 1

UML diagrams and Java

**Theory**You covered the theory that you must have processed before this lab session in the lecture of 28 September.

**Target**In this lab we explore the integrated development environment (IDE) **BlueJ** and learn how to start **a first Java project**. We also practice using Unified Modeling Language (**UML**).

**Format**  
Each lab session is divided into three different parts: starting exercises, lab exercises and home exercises. **Starting exercises** are exercises that we expect you to solve at home, before the lab session. In this way you become familiar with the material and you check whether you have properly understood and processed the theory of the lectures. **Lab exercises** are exercises that are covered during the lab sessions. **Home exercises** are exercises that delve deeper into the matter. They are optional for those who work faster, or as extra exercise material when studying/practicing at home. Contact your lab assistant for feedback on these exercises.

**Learning Objectives**After this session you should be able to understand and implement the following concepts:

* + Create and interpret a first UML diagram
  + Write classes and methods
  + Create objects and perform simple tasks on them
  + Use different primitive data types

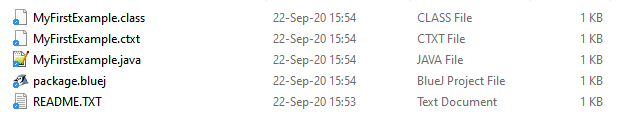
# Starting Exercises

## Exercise 0: Install BlueJ

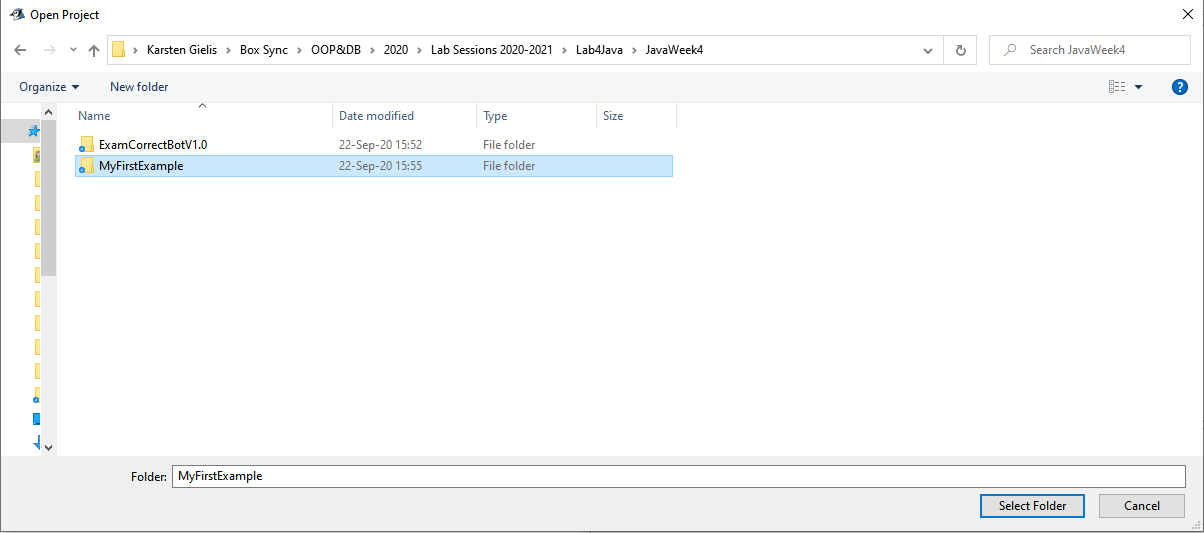
Download and install BlueJ from <https://bluej.org/>.

## Exercise 1: Hello World

Download the zip file “MyFirstExample.zip” from Toledo. Unzip this file (so don't just open it!). You should see the following content:



Next, open BlueJ, click on Project🡪Open Project. Select the MyFirstExample folder. Do not *enter* the folder, *select* the folder and click “Select Folder”. This should look more or less like this:



Double-click the MyFirstExample class and inspect the code. Try to figure out what each line of code does. Close the source code and run the following commands.

1. Create an object of the MyFirstExample class. To do this, right-click on the class and choose MyFirstExample(). Name it myFirstObject. You should see a red object named myFirstObject appear in the "object bench" at the bottom.
2. Run the printText() method by right-clicking on the object.
3. Change the text to be printed using setTextToPrint(). Change the text to “Hello World!”. What is the name of the variable where you stored the new string?
4. Print the modified text.

## Exercise 2: Analyzing UML diagrams

This is the UML diagram of the MyFirstExample class:

|  |
| --- |
| **MyFirstExample** |
| -textToPrint: String |
| +MyFirstExample()  +setTextToPrint(String newText): void  +printText(): void |

Answer the following questions.

1. Which keyword in the Java code corresponds to the minus sign in front of the variable textToPrint?
2. What is the technical term for MyFirstExample()?
3. What is the name of a variable like newText?
4. What is the role of the keyword “void”?

## Exercise 3: Instances

Look again at the UML diagram of MyFirstExample. You can see that there is one instance variable (also called a *field*) named textToPrint. The actual value of this variable may differ between different objects (instances) of the same class. Do the following:

1. Create two objects of the MyFirstExample class.
2. Look at the value of textToPrint on both objects (right click on object 🡪 inspection).
3. Apply the setTextToPrint() method to one of the two objects to modify the text to be printed.
4. Inspect both objects. Has the value of the variable changed for both?

## Exercise 4: Data Types

What is the data type of these values?

|  |  |
| --- | --- |
| Value | Data type |
| 56 |  | |
| 'm' |  | |
| 230 |  | |
| “410” |  | |
| “Hello World” |  | |
| true |  | |
| 0.045 |  | |

# Lab Exercises

## Exercise 1: Writing your own methods

Open the MyFirstExample class. We will add two methods in it. The new methods are indicated in **bold**.

|  |
| --- |
| **MyFirstExample** |
| -textToPrint: String |
| +MyFirstExample()  +setTextToPrint(String newText): void  +printText(): void  **+printSum(int a, int b): void**  **+prettyPrintSubtraction(int a, int b): void** |

1. printSum calculates the sum of two parameters a and b and prints the result. Write this method and test your code by creating an instance and calling the method. Don't forget to click "compile" to compile your code after you've written it.
2. prettyPrintSubtraction subtracts a from b and prints the result in a sentence, for example: “5 minus 2 equals 3”. Again, create an object and test your code.

## Exercise 2: Creating your own class

Time to create your own class. The UML diagram of the target class can be found below:

|  |
| --- |
| **GroupTStudent** |
| -name: String  -yearsAtGroupT: int |
| +GroupTStudent(String studentName)  + addYear(): void  + introduceMyself(): void |

Perform the following tasks:

1. Create a class by right clicking in the blank canvas and choosing “New Class”. Give the class the correct name (capital letter!) and don't worry about the other settings. Open the class.
2. BlueJ automatically sets a number of fields and methods. Analyse what these methods and variables do. Delete them afterwards.
3. Add the fields: name and yearsAtGroupT.
4. Add a studentName parameter in the constructor.
5. In the constructor do the following:
   1. Store the studentName parameter in the name field;
   2. instantiate yearsAtGroupT as 0.
6. Compile your code and create an object of class GroupTStudent. Inspect the object to make sure everything is as it should be. It's a good idea to compile and test every now and then when you're writing new code, that way you'll detect errors earlier.
7. Write the addYear() method. This method increments yearsAtGroupT by 1. Compile, create object, test.
8. Write the introduceMyself() method. This method prints a sentence of the form “My name is … and I've been studying at GroepT for … years”. Compile, create object, test.

## Exercise 4: Design your own UML diagram

Draw a detailed class diagram based on the following problem description. Define the required class, its attributes (name + type) and signature of the methods.

**Problem description:**

Admin wants to design a system to store Group T courses in a software system, in order to be able to manage the Toledo page of these courses more easily. A course has a name and a number of credits. Courses also have a code that you want to save: b, m or p (these stand for bachelor, master or postgraduate course respectively). You must also be able to indicate whether or not the course is visible to students in Toledo. You should be able to print out all this information from the course. Name, credits and code are fixed, but whether the course is visible or not can be altered.

## Exercise 5: Temperature conversion

To help the chemistry lab to convert temperatures, we write a converter. Temperatures can be converted from Celsius to Fahrenheit, or vice versa. The formula for that conversion is given below.

*°C = 5\*(°F-32)/9*

*°F = 9\*(°C)/5+32*

1. Create a new project and a TemperatureConverter class.
2. Define a convertsTo field of type char.
3. Write a constructor for the class that accepts a character as a parameter. Initiate convertsTo with this value. The intended use is that the user creates a converter and indicates to which unit this converter will convert: either 'C' or 'F'.
   1. Bonus: make the constructor “fool proof”: if the user enters a value different from these two, you choose a default value (eg 'C').
4. Write a convert() method that accepts a float as a parameter. This is the value to be converted. Depending on the value of the convertsTo field, you use the correct formula. The result is printed in a sentence, for example: “1832 degrees Fahrenheit is 1000 degrees Celsius”.
5. Create a second class: GeneralTemperatureConverter. In it you write a convert() method that accepts two parameters: a float representing the value to be converted, and a character, the unit of this value. The method does the correct conversion and prints the result.
   1. Why doesn't it make much sense to store that unit in a field?
   2. Bonus: fool-proof the method again: if the user enters a wrong value, an error message will be printed.

You can use the following values ​​as a test case for this exercise:

|  |  |
| --- | --- |
| Celsius | Fahrenheit |
| -273.15 | -459.67 |
| -30 | -22 |
| 0 | 32 |
| 30 | 86 |
| 100 | 212 |
| 1000 | 1832 |

# Home Exercises

## Exercise 1: Temperature conversion bis

Rewrite your code from above to ensure that values ​​that are too cold are discarded. Make sure that temperatures below 0K (absolute zero point) lead to an error message. Converting to Kelvin is done like this:

T (in °C) + 273.15 = T (in K)  
T (in K) − 273.15 = T (in °C)

## Exercise 2: Movie access

Based on:<https://edabit.com/challenge/MvBYeGybFo4iEpWyp>

Create a MovieAdmissions class and implement a method that checks whether a person is allowed to watch a 15+ movie. In that case, one of the following conditions must be met:

* The person is at least 15 years old;
* Or the person has his or her parents with them.

The signature of the method is as follows:

+ acceptIntoMovie (int age, Boolean supervised): void

**Test solutions with printed output**  
acceptIntoMovie(14, true) 🡪 do come in  
acceptIntoMovie(14, false) 🡪 nice try!  
acceptIntoMovie(16, false) 🡪 do come in  
acceptIntoMovie(16, true) 🡪 do come in  
acceptIntoMovie(15, true) 🡪 do come in  
acceptIntoMovie(15, false) 🡪 do come in

## Exercise 3: UML diagram implementation

Implement the UML diagram you created in exercise 4 of the lab exercises.