UNIVERSITÉ DU LUXEMBOURG

Programming 1

Patrick Keller (patrick.keller@uni.lu)

Lab 3 – Iterative Structures

2021–10–08

Submission (Preliminary: 17/10, Final: 24/10)
The mandatory exercises for **Code Submission** are **1, 3, 4, 7** (marked with an ❗). All other exercises on this sheet are optional but still highly recommended! The **Explainer Video** 🎥 for this sheet must be realized on **Exercise 4 ("Division by three"-ception!)**. A Flipgrid invitation link will be posted on Moodle.

Template classes
From now on you will no longer find empty classes for the exercises in the GitHub Classroom repository. Each (sub-)exercise will define a class name which you are supposed to create a class with and develop your code inside.
Besides of adhering to the proposed class names also put the exercises into the package structure as seen in previous labs:
"lu.uni.programming1.lab<K>.exercise<N>", where K is the lab number and N is the exercise number.
How to create packages and classes with a main method inside was covered in the very first lecture so please refer to that recording if you face any issues.

If you do not stick to these naming conventions or package structure we will deduce points from your grade - or not grade the exercise at all if we cannot find it due to naming.

Exercise 1 – ❗ Let's count! #ForLoop

In this exercise we write some programs that can do some simple but cumbersome counting for us.
Use 20 lines of code at most. (excluding imports, class definition and blank lines for structuring your code)

1° Write a program **OneToFive.java** that outputs the **integer** numbers 1, 2, 3, 4, 5.

2° Write a program **OneTo100.java** that outputs the **integer** numbers 1 to 100.

3° Write a program **NtoM.java** that outputs the **integer** numbers from **n to m**, after reading **n** and **m** from standard input.

4° Write a program **XtoY.java** that outputs the **decimal** numbers from **x to y** in **steps of d**, after reading **x, y and d** from standard input.

Exercise 2 – Sum #WhileLoop #DoWhileLoop

Write a program **Sum.java** that reads from the console a positive integer n and that calculates the sum of all integers from 1 to n. As long as the user has not entered a *positive* integer, keep on asking. Calculate it in three different ways, using a `for` loop, a `while` loop and a `do-while` loop. Finally, compare the different results with the result of the formula

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$

Exercise 3 – ❗ Statistics

Write a program **Statistics.java** that calculates the average of a sequence of integer numbers read from the console. If the user enters 0, the sequence will stop.

Once the sequence has stopped, the program should write to console:

- the smallest number
- the largest number
- the average of all the numbers

Exercise 4 – ❗ 🎥"Division by three"-ception! #WhileLoop #DoWhileLoop #Switch

In Lab2, we did some division by three by summing up all the digits and checking whether this sum is divisible by 3. We will improve this approach and remove the limitation that we only allow up to 4 digit numbers, so this time they can be much bigger!

This time it is **not allowed** to read each digit one by one - you have to read it as a whole single number!

In a second step, we will also **remove the need** to check if the sum is divisible by three (using modulo 3). The sum of the digits again is divisible by three - you guessed it - if the sum of it's digits is divisible by three! To do that we will re-apply our technique again and again until the sum becomes smaller than 10 (respectively a single digit). Use a switch statement to check if this digit is divisible by 3!

Example: 23251353 → sum of digits: 24 → sum of digits : 6 → Divisible by 3!

Example2: 251353 → sum of digits: 19 → sum of digits : 10 → sum of digits : 1 → Not divisible by 3!

1° Write a program **DivThreeMoreDigits.java** that reads a *long* value from standard input and checks if it is divisible by three. Do so by summing up **once** its digits using a `while` loop and checking the sum using **modulo 3**.

2° Write a new program **DivThreeCeption.java**, which is based on the previous one but removes the **modulo 3** check by reapplying the digit summation (multiple time if necessary) and using a switch statement as described above.

- ℹ Use integer division by 10 and modulo 10 (%) to go over the different digits of the number.
- ℹ Try different numbers to test if your program works correctly, e.g. 0, 3, 5, 15, 17, 3251353, 23251353, ...
- ℹ To implement the repeated summation, a `do-while` loop can be helpful but is not mandatory!

Exercise 5 – Decomposition of a 1 € coin

Write a program **Coins.java** that calculates all possibilities of decomposing a 1 € coin into 20, 10 and 5 cent coins. Print those possibilities and their total number.

Exercise 6 – Guess a number #Random

Write a program **GuessIt.java** that randomly chooses a number between 1 and 100 and lets the user guess which number was chosen. With each guess the program shall tell the user whether the guessed number was bigger, equal to or smaller than the searched number until the right number is guessed. Finally the program shall print out how many tries the user needed to guess the right number.

Exercise 7 – ! Kellacci sequence

Your TA Patrick has invented a new impressive number sequence, derived from the slightly more famous Fibonacci numbers. ;)
Write a program **Kellacci.java** that reads an integer number **n** from standard input and lists the Kellacci sequence $\mathcal{K}$ until the n^{th} term $\mathcal{K}_n$. Your solution must be based on loops, not on recursion. Note that the Kellacci sequence is defined as:

$$\begin{cases} \mathcal{K}_1 & = 1 \\ \mathcal{K}_2 & = 1 \\ \mathcal{K}_n & = (\mathcal{K}_{n-1} - 1) + (\mathcal{K}_{n-2} \times 2) \quad n > 2 \end{cases}$$

Exercise 8 – Refactoring: Extracting variables

As seen in the lecture, there are several reasons why applying a clean coding style is helpful. One way to do that is extracting variables, which is a useful refactoring technique wrt. code maintainability. In VS Code, you can extract a variable from an expression by selecting it, right-clicking and selecting Refactor... and then Extract to local variable. Finally, provide a name for the new variable. Try this by extracting variables for each of the 3 expressions (between the operators) in the `if`-statement of the code snippet below.
Write a new program **VariablesExtracted.java** that is based on the existing class **VariableExtraction.java** from the repository, that contains the code snippet. Apply the refactoring technique described above to the code in your new program.

```
1     String platform = "macOS Sierra";
2     String browser = "Safari";
3     double zoomLevel = 1.5;
4     if (platform.toUpperCase().indexOf("MAC") > -1 && browser.equals("Safari") && zoomLevel >=
          1.5) {
5       // do something ...
6     }
```