Lab 1

October 21st, 2019

The tasks on this sheet are intended to be done by students having no or very little experience in (procedural) programming. Other students may skip this task and proceed with lab 2.

Task 1.1 : BMI calculation

Write a program that calculates the body mass index (BMI) of a human. The BMI is a measure for a human's body shape (i.e. underweight, normal weight, overweight) based on his or her weight and height. It is calculated with the formula:

$$\text{BMI} = \frac{w}{h^2}\,, \quad w := \text{weight in kg}, \quad h := \text{height in m} \tag{1}$$

When the program is started, it queries weight and height from the user. Then it calculates the BMI and displays it to the user. Complete the following steps:

1. Create a new Eclipse project and add a class named `BmiCalculation` with a method `main()`, as in the *HelloWorld* example from the Eclipse tutorial.
2. Declare the needed variables given in Eq. (1) and choose appropriate data types.
3. Write statements that query weight and height from the user and assign them to the corresponding variables (mind the comments on the class `SimpleIO` below).
4. Add a statement to calculate the BMI.
5. Add a statement to display the calculated BMI to the user.
6. Test your program for several inputs.

Use the class (= module) `SimpleIO` for the interaction with the user. You need to download it from StudIP and add it to your project. You download it as file *SimpleIO.java*. Create a new package (*File→New→Package*) with the name `de.tuhh.diss.io`. You will see it in the package explorer on the left. Use the mouse to drag the file *SimpleIO.java* into the package in the project explorer.

At the beginning of each of your labs, import `SimpleIO` by typing the declaration:

```
import de.tuhh.diss.io.SimpleIO;
```

Then you can use its methods `SimpleIO.print()` and `SimpleIO.println()` to display data to the user. Methods such as `SimpleIO.readInteger()` and `SimpleIO.readDouble()` can be used to query data from the user, e.g.

```
myNewInteger = SimpleIO.readInteger();
```

queries an integer value from the user and assigns it to the variable `myNewInteger`. Note that you have to declare `myNewInteger` first.

Task 1.2 : Check for correct input values

Since your BMI calculation might produce illogical output data if unexpected values of weight or height are entered, the program should be modified. Use a loop to ensure the user enters neither a negative weight and height nor a height exceeding 2.72 m (height of Robert Wadlow, the tallest men ever recorded). The user should be prompted again and again until a legal value is entered. What kind of loop is useful? Proceed with the calculation and print the correct result to the console.

Why are if-statements *alone* no good idea for this task?

Task 1.3 : Classification of results

Based on the BMI, you can rate if the entered person has underweight, normal weight or overweight:

Condition	Result
$\text{BMI} < 18.5$	underweight
$\text{BMI} \geq 25.0$	overweight
$18.5 \leq \text{BMI} < 25.0$	normal weight

Add two double constants to your program and protect them from modification within the program:

- `BMI_UPPER = 25.0`
- `BMI_LOWER = 18.5`

Use if-statements, the given table and the constants to perform classification. Display the results to the user. How many if-statements do you need?

Task 1.4 : Storing values in a one-dimensional array

For later retrieval, we want to store the BMI for each person separately. For that purpose, replace the double variable `bmi` by a suitable array to store the value for each person.

Create a dialog so that the user is able to input a new user or to print a specific entry of the array. Keep in mind that you cannot store more users than you defined your array for, i.e. prevent out-of-boundary access.

Task 1.5 : Calculate the average

Fo statistical purposes, it is interesting to know the average BMI of different persons. Use a `for-loop` to access all elements in the BMI array and cumulate them in a temporal variable. Only treat non-zero elements. Divide the sum by the number of BMI entries in the array.

Print the average to the console along with the entries itself.

Task 1.6 : Upper-bounding eigenvalues

Let λ be an eigenvalue of $\mathbf{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ and $\|\mathbf{A}\|_{\mathrm{F}}$ the Frobenius norm of the matrix $\mathbf{A}$, then

$$|\lambda|_{\mathrm{F}} \leq \|\mathbf{A}\|_{\mathrm{F}} \tag{2}$$

holds. Thus, for deriving an upper bound of the eigenvalues of $\mathbf{A}$ the calculation of the Frobenius norm of the matrix $\mathbf{A}$ can be used. The Frobenius norm of any matrix $\mathbf{A} \in \mathbb{R}^{\mathrm{n}\times\mathrm{n}}$ is defined as:

$$\|\mathbf{A}\|_{\mathrm{F}} := \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |\mathrm{a}_{ij}|^2}\,, \tag{3}$$

where a_{ij} are the elements in the matrix $\mathbf{A}$. In words, the Frobenius norm calculates the square root of the sum of squares of all elements in the matrix.

Create a new class `EigenvalueUpperBound` that lets the user define the entries of matrix $\mathbf{A} \in \mathbb{R}^{3\times3}$ with `SimpleIO.readDouble()`. The matrix is defined as

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \tag{4}$$

and will be stored in a two-dimensional array `A`. The class also gives an upper bound on the magnitude of eigenvalues of $\mathbf{A}$ by calculating its Frobenius norm. Use two nested for-loops for accessing all rows i and all columns j in your matrix. Use a constant `MAX_MATRIXSIZE = 3` to allocate enough memory and to avoid out-of-bound indexation of your array.

Think about how many for-loops you *really* need and optimize your program accordingly.

Hint: Once the user specified all entries of the matrix, you can use the methods provided by the Java `Math` library:

- `Math.sqrt(b)` to calculate the square root of `b`.
- `Math.pow(b, 2)` to calculate the square of `b`.

Note `b` might be a double variable declared first or any other double value.

Task 1.7 : Upper-bounding eigenvalues in methods

For good software engineering, it is recommended to separate user input and calculation. Thus, create a new method `obtainFrobeniusNorm(...)`, which accepts an array `matrixA` as input and returns a double value containing the Frobenius norm of the matrix. Call this method from `main()` and pass the user-entered matrix $\mathbf{A} \in \mathbb{R}^{3\times3}$ to the new method. Use the return value to generate the output text.

Hint: Use the following method.

```java
public static double obtainFrobeniusNorm(double[][] matrixA) {
  double frobeniusNorm = 0;
  //Put your calculation here
  return frobeniusNorm;
  }
```

Task 1.8 : VO_2 max

In training theory, the maximal oxygen consumption (VO_2 max) is an important factor to assess your aerobic fitness. Research has shown that VO_2 max, maximum heart rate (HR_{max}) and resting heart rate (HR_{rest}) are related by the following equation:

$$VO_2 \text{ max} \approx 15 \times \frac{HR_{max}}{HR_{rest}} \quad [\text{mL}/(\text{kg min})] \tag{5}$$

HR_{max} and HR_{rest} may be estimated by the age of the person, via equation

$$HR_{max} \approx 206.9 - (0.67 \times \text{age}) \quad [1/\text{min}], \tag{6}$$

and by the given (rough) classification table:

age	18-35	36-55	56+
HR_{rest}	71	73	76

Write a class `MaxOxygen` that calculates VO_2 max based on the age of a person. Your program contains three methods for:

- VO_2 max calculation,
- HR_{max} calculation,
- HR_{rest} decision.

To ensure all methods are accessible by each other, declare them as `public static`[1] and use a suitable return type. Write a method `main()`, which queries the age of the user and print its VO_2 max to the console.

Task 1.9 : Newton's method

Write a program that approximates the root x_r ($f(x_r) = 0, x_r \in \mathbb{R}$) of the function:

$$f(x) := x^4 - 2x^3 - x^2 - 2x + 2\,. \tag{7}$$

The root of the function is approximated using the algorithm called "Newton's method", which works as follows:

Algorithm 1 Newton's method

```
  **while** $f(x_{r,n}) \neq 0$ **do**
      $x_{r,n+1} \leftarrow x_{r,n} - f(x_{r,n})/\mathrm{d}f(x_{r,n})$
      $x_{r,n} \leftarrow x_{r,n+1}$
  **end while**
```

Use an initial value $x_{r,0} = 10$. Count the number of iterations your program needs and print it to the console. How many iterations do you need?

Is $f(x_{r,n}) \neq 0$ a useful choice to end the repetition loop of your program? Think of other abort conditions and implement them accordingly. What happens if you use $x_{r,0} = 0$ as starting value?

Hint: Use the following implementations of the methods for $f(x)$ and its derivative.

[1] The purpose of this will be explained in the following lectures

```java
/**
* @returns f(x) = x^4 - 2x^3 - x^2 - 2x + 2
*/
public static double f(double x) {
  return Math.pow(x, 4) - 2*Math.pow(x, 3) - Math.pow(x, 2) - 2*x + 2;
}

/**
* @returns f'(x) = 4x^3 - 6x^2 - 2x - 2
*/
public static double df(double x) {
  return 4*Math.pow(x, 3) - 6*Math.pow(x, 2) - 2*x - 2;
}
```

Task 1.10 : Newton's method II

Use Newton's method to calculate the root for the polynomial $g(x)$ with

$$g(x) := x^3 - 2x + 2\,. \tag{8}$$

Again, use the initial value $x_{r,0} = 10$. Output the current root estimation in each iteration to the console. How can you prevent the unexpected behavior?