# DM550: INTRODUCTION TO PROGRAMMING
# Exercises for the laboratories (Autumn 2021)

## Week 36

The goal of this lab is to get started with Java.

1. Go to `https://adoptopenjdk.net/` and install OpenJDK 8. Be sure to select version 8 (the default is 11).
2. Download `HelloWorld.java` from the somewhere something.
3. Open a terminal window and compile that file by typing `javac HelloWorld.java`
4. Run this program by typing `java HelloWorld`

If you have any problems or are unsure about what to do in any of these steps, get help! There may be some configurations that need to be adjusted.

If everything works and you want to try your luck at writing simple Java programs, here are some exercises.

1. Write a class `Variables` whose `main` method declares and initializes five variables of numberic types (`int`, `float` or `double`) and prints messages with their names and values. Compile, execute and if necessary debug your class.
2. Extend class `Variables` with three new variables `i`, `j` and `k` of type `int`. Assign to them the values 1, 2 and 4, and print their average. Check the returned value and, if necessary, fix the code.

## Week 37

The goal of this lab is to write very short programs that introduce the basic principles of writing Java classes.

1. Write a class `Time` to convert a time interval given in hours, minutes and seconds (all of type `int`) into the equivalent time interval in seconds. The input should be stored in variables inside the `main` method, and the result should be printed on the screen.

   Remember to structure your code and include comments where useful. Test your class with different input values.
2. Write a class `Durations` to perform the opposite conversion: from a time interval given in seconds (stored in a variable of type `int`) to one in a readable format (hours, minutes and seconds). The input should again be stored in a variable inside the `main` method, and the result should be printed on the screen.

   Think about the algorithm to use and which extra variables are useful. Remember to structure and comment your code. Test your class with different input values.
3. Develop a class `Stars.java` containing the methods listed below. Its `main` method should include calls to some of these methods, illustrating their functionality.

   (a) A method `void stars(int n)` that prints on the screen a single line containing `n` stars (`*`).

   (b) A method `void linesOfStars(int n)` that prints on the screen lines with 1, 2, …, `n` stars. For example, for `n`=5 the result should be the following.

   ```
   *
   **
   ***
   ****
   *****
   ```

   (c) A method `void triangle(int n)` that prints on the screen an isosceles triangle of stars with `n` lines. For example, for `n`=5 the result should be the following.

   ```
       *
      ***
     *****
    *******
   *********
   ```

*Hint:* include a variant of the method `stars` from the previous exercise that also parameterizes the symbol to be printed.

4. While developing a program, it is often useful to include monitoring statements to check that the variables are storing the expected values.

   Consider the following problem. Given a number `n`, we repeatedly apply the following recipe: if our current value is even, we divide it by 2; if it is odd, we multiply it by 3 and add 1. Write a method `int downToOne(int n)` that finds out how many steps it takes to get to 1. Include auxiliary statements to check that your program is correct.

## Week 38

This week we use the techniques developed so far to solve some simple (but important) numerical problems.

1. *Finding zeros.* Given a function $f$ on the natural numbers, a simple way to find the least $n$ for which $f(n) = 0$ is by testing: we compute $f(0)$, $f(1)$, etc, until we find the right value.

   Write a class `FindZeros` that implements this algorithm. The function $f$ should be defined as a private method `int f(int n)`; for example, for $f(n) = n^2$, the class should include the following method.

```
private static int f(int n){
  return n*n;
}
```

   What happens if you call your class with the function $f(n) = n + 1$?

2. *Solving equations.* Consider the following method (*bisection method*) to solve an equation $f(x) = 0$ for a continuous function $f$. Given $a < b$ with $f(a) \times f(b) < 0$, we first compute the midpoint $c = \frac{a+b}{2}$ and the value of $f(c)$; if $f(a) \times f(c) < 0$, we repeat the procedure with $b = c$; otherwise, we repeat it with $a = c$. This procedure terminates when the difference between $a$ and $b$ is smaller than a given value (the error), and the current value of $a$ is an approximate solution of $f(x) = 0$.

   Implement this method as a class `Bisection` that prints the final solution on the screen. The initial values of `a` and `b`, as well as the error, should be parameters of the `main` method. The function $f$ is defined as a private method as in the previous exercise; for example, for $f(x) = x^2$, your class should include the following method.

```
private static double f(double x){
  return x*x;
}
```

## Week 39

*The exercises for this lab will be indicated in class.*

## Week 40

The number system used throughout the Roman Empire used seven uppercase letters: `I`, `V`, `X`, `L`, `C`, `D` and `M`, with values respectively of 1, 5, 10, 50, 100, 500 and 1000. Numbers were written as ordered sequences of these letters, subject to the following restrictions:

- letters appear in descending order of the value they represent;
- the letters `V`, `L` and `D` can only occur once, and `I`, `X` and `C` can occur at most four times.

To find the number represented by a particular sequence we simply add all values corresponding to the letters in it. Thus, the number 2341 is written `MMCCCXXXXI`, while `DCCII` corresponds to 702.[1]

Develop a class `Roman` to work with Roman numberals. This class should include the following methods.

[1]The variant of the Roman number system most commonly known, where 2341 is written `MMCCCXLI`, is a later version. The original system used `IIII`, `XXXX` and `CCCC` rather than `IV`, `XL` and `CD`.

1. Write a method `int romanToNum(String num)` that converts a string representing a valid Roman numeral into the corresponding integer number.

2. Write a method `String numToRoman(int n)` that converts an integer between 1 and 4000 into the corresponding Roman numeral.

3. Write a method `String add(String num1, String num2)` that adds two Roman numerals. In order to keep it challenging, you are only allowed to use methods over `String`s.

   *Hint.* Split the problem in two steps. In the first step, merge the two argument strings keeping the ordering of the letters, without worrying about getting too many copies of each letter. In the second step, replace all invalid sequences by equivalent, valid, ones (e.g., `IIIII` is replaced by `V`). Thus, to add `MMCCCXXXXI` with `DCCII`, we first build the intermediate string `MMDCCCCCXXXXIII`.

4. Write a method `String diff(String num1, String num2)` that takes as arguments two strings representing Roman numbers, where the first is stricly larger than the second, and returns their difference. Once again, only methods over `String`s are allowed.

   *Hint.* Recall that $m - n$ is the only integer $k$ satisfying $n + k = m$.

5. Write a method `boolean isRomanNum(String s)` to determine whether a string represents a valid Roman numeral.

When invoked, your class should inform the user of the available methods, ask for an operation to perform, and query for any eventually needed input. Use a top-down programming methodology: start by writing the skeleton of each auxiliary method, then focus on the `main` method. Afterwards, you can implement and test the auxiliary methods one by one.

## Week 41

The goal of this exercise is to develop a class to scramble texts. The techniques we want to implement are the following.

- Remove all vowels from words in the text, except for those words that only consist of vowels.
- Remove all spaces from the original text and add a new space after every n-th character.
- Reverse every individual word.
- Divide the text into blocks of length `m` and reverse each block individually (if there are less than `m` characters at the end of the string, reverse them also).
- Shift each individual word in the text by a given number of characters.

Your class `ScrambleIt` should implement an interactive loop that, at each iteration, asks the user to introduce the text to be scrambled, lists the implemented scrambling techniques, lets the user choose which one to apply, displays the result, and asks whether to continue.

Use a top-down programming methodology: start by writing only the skeleton of the scrambling methods, and develop the main loop. This is a good point to use a `do-while` loop. Use a `switch` command to choose the auxiliary method to invoke for scrambling.

After the main loop is correctly implemented, start implementing the scrambling methods one by one. You can test them from the main loop as soon as they are written.

## Week 43

Recall the contract for the class `Image`, used in the exercise class for this week:

- a constructor with two arguments that creates a black image with the specified dimensions;
- a constructor with one argument that takes the name of a file containing an image and creates an object representing that image;
- methods `int width()` and `int height()` that return the width and height of this image;
- methods `int red(int x, int y)`, `int green(int x, int y)` and `int blue(int x, int y)` that return the red, green and blue components of pixel `(x,y)`, assuming that these coordinates represent a valid pixel;

- a method `void setPixel(int x, int y, int red, int green, int blue)` that sets the red, green and blue values of the pixel with coordinates `(x,y)` to the given values (assuming that the coordinates represent a valid pixel and that the color values are in the range $[0, 255]$);
- a method `void display()` that displays this image on the screen.

Recall that pixels are counted from the top left corner of the picture.

The goal of this lab write a new utility class `ImageCoder.java` that provides additional methods to draw upon an existing image, together with methods to encrypt and decrypt the image using an external key.

The methods that you should implement are the following:

- a method `void addRectangle(Image image, int x, int y, int width, int height, int red, int green, int blue)` that draws a rectangle of the given color with size `width`×`height` with upper left corner in position `(x,y)`, truncated to fit the picture;
- a method `void addCircle(Image image, int x, int y, int radius, int red, int green, int blue)` that draws a circle of radius `radius` with center in position `(x,y)`, truncated to fit the picture;
- a method `void encrypt(Image image, String key)` that encrypts the image as follows: change each color component of each pixel by adding it to the same color component of the previously visited pixel and to the character code of the next character in `key` (modulo 256);
- a method `void decrypt(Image image, String key)` that decrypts an image, assuming it was encoded by the previous method using `key`.

## Week 44

Finish the implementation of the tic-tac-toe game developed during the lectures in this week. You can also extend your client following the suggestions in section 2.5.

## Week 45

The calculator developed in the previous week relies on a class `Fraction` whose objects represent fractions. Implement this class. Decide which attributes it should have, and which getters and setters should be available for these attributes. Recall that the contract for this class specifies the following methods:

- constructors with zero, one or two arguments, building the fraction 1, a whole number or an arbitrary fraction, respectively;
- methods `Fraction add(Fraction f)`, `Fraction subtract(Fraction f)`, `Fraction multiply(Fraction f)` and `Fraction divide(Fraction f)` returning the result of adding, subtracting, multiplying or dividing this fraction with/by fraction `f`, respectively;
- a method `void simplify()` that transforms this fraction into an equivalent irreducible fraction (i.e., one where the numerator and denominator do not have common divisors);
- a method `double value()` that returns a floating point approximation of the value represented by this fraction;
- methods `int integerPart()` and `Fraction properPart()` returning the integer and proper part of the fraction (for example, $\frac{8}{3} = 2 + \frac{2}{3}$, where 2 is its integer part and $\frac{2}{3}$ is its proper part);
- a method `boolean equals(Object other)` that checks whether this fraction is equal to `other`;
- a method `Fraction copy()` that returns a fraction equal to this one;
- a method `String toString()` that returns a textual representation of this fraction.

Test your class with the calculator developed previously, and also with your own client.

## Week 46

Extend the parser in the calculator developed in section 1 with exceptions that detect if incorrect input is entered. This task includes:

- defining a subclass of `Exception` that implements your own exception type;
- throwing new exceptions of this type whenever relevant;
- deciding whether exceptions thrown by one method should be caught by the caller or passed to the level above;
- making sure that no exceptions escape to the user level.

## Week 47

*The exercises for this lab will be indicated in class.*

## Week 48

1. Write a `Mirror` program that reads a text file, line by line, and reverses each line before writing it to a new file. The name of the input file should be provided by the user; the name of the output file should (of course!) be the name of the input file, reversed. Make sure to avoid trying to write on the same file you are reading from.
2. Write a class `PalindromeChecker` that, when executed, queries the user for a string, and then checks whether this string is a palindrome (i.e., whether it reads the same when it is reversed). The palindrome test should be done by an auxiliary method that is programmed recursively.
3. Write a class `PermutationGenerator` that, when executed, queries the user for a string, and then prints on the screen all the permutations of the characters in that string, one per line. Generating all permutations should be done by an auxiliary method that is programmed recursively.

## Week 49

*The exercises for this lab will be indicated in class.*

## Week 50

Recall that the interface `MyCollection<E>` defines the following operations.

- `void add(E e)`: ensures that this collection contains `e`.
- `void clear()`: removes all elements in this collection.
- `boolean contains(E e)`: checks whether this collection contains `e`.
- `boolean isEmpty()`: checks whether this collection is empty.
- `int size()`: returns the number of elements in this collection.

Also recall that a *binary tree* is a datatype for collections where elements are stored in nodes, and each node has a pointer to two other nodes (its *left* and *right children*). The topmost node is called the *root*, and a node whose children are both `null` is called a *leaf.* The class `BinaryTree<E>` implements binary trees of elements of type `E`, and provides the following methods in addition to those defined in the `MyCollection<E>` and `Iterable<E>` interfaces.

- `E root()`: returns the element in the root note of the tree.
- `BinaryTree<E> left()` and `BinaryTree<E> right()`: return the subtrees with the left (respectively, right) child of the current root note as root.
- `int height()`: returns the length of the longest path from the root to a leaf node.

1. Write a recursive method `double sum(BinaryTree<Double> t)` that computes the sum of all values in `t`.
2. Write a recursive method `int zeros(BinaryTree<Integer> s)` that returns the number of zeros in `t`.
3. Write a recursive method `int count(BinaryTree<String> t, String s)` that counts the number of occurrences of `s` in `t`.
4. Write a recursive method `ArrayList<Integer> toArrayList(BinaryTree<Integer> t)` that returns a list containing all the elements of `t`.
5. Write a recursive method `ArrayList<Integer> selectLarger(BinaryTree<Integer> t, int n)` that returns a list containing the elements of `t` that are larger than `n`.
6. Write a recursive method `int mostLeftTurns(BinaryTree<E> t)` that returns the highest number of choices of the left child in a path from the root of `t` to any leaf.

## Week 51

A *cache* is similar to a map: it is a datatype for collections where elements are stored together with a key, but where there is a maximum amount of values that can be stored. When this capacity is exceeded, further additions require (silently) removing one element from the cache according to some predetermined strategy.

1. Develop a class `SimpleCache<K,E>` implementing a cache of elements of type `E` with keys of type `K`. The capacity should be specified in the constructor, and the removal strategy is simply "delete the oldest element".

   This class should implement the same functionality as the class `Map<K,E>` (exercise 3.6), that is: it should implement interface `Iterable<E>` and provide the following additional methods.

   - `void add(K key,E e)`: adds to this collection an element `e` assigned to key `key`, removing any other element eventually assigned to `key`.
   - `void clear()`: removes all elements in this collection.
   - `boolean find(K key)`: return the element in this collection assigned to key `key`, if one exists.
   - `boolean isEmpty()`: checks whether this collection is empty.
   - `void remove(K key)`: removes the element assigned to key `key`, if one exists.
   - `int size()`: returns the number of elements in this collection.
   - `K[] keys()`: returns an array with all the keys in this collection.
   - `E[] values()`: returns an array with all the elements in this collection.
   - `Iterator<K> keyIterator()`: returns an iterator over the keys in this map.

2. A better solution to managing the cache's capacity is to remove the element that has been used the least. This requires that the cache also keep information about how many times each element has been accessed by method `find`.

   Develop a class `Cache<K,E>` that uses this strategy. Reuse as much as possible from the previous exercise.