# CMP-5015A Programming 2

# Week 8: Lab Exercises

**Learning Outcome:**
- solid basis on topics covered in Java in Programming 1
- good understanding of the concepts of classes and objects
- understanding the notion of inheritance

## About the labs...

**Lab sheets.** Every week, the lab sheets will propose exercises from easy to difficult ones. You do not have to do them all but make sure you understand well the concepts that have been covered that week. For the most difficult exercises and the ones written optional, treat them as a challenge on which you can work on over several weeks (even as a group) and don't rush to check the solution.

**My advice.** Try by yourself first, before asking for help or looking at the solution. The best way to become a good programmer is to practise and code, even if sometimes it is a bit of a headache. Teaching assistants and lecturers are here to help (and very happy to), but we will try to make you understand your mistake by yourself, rather than feeding you the solution. Your code is not compiling? Check what the error message says. It is not running as expected? Try to print within the code, to see where it goes wrong. And don't copy-paste solutions... What's the point?

Another advice is to make sure you understand the basics of programming (in whatever language) and have solid foundations before trying to do more advanced fancy things. This is an interesting read:
https://blog.codinghorror.com/why-cant-programmers-program/
So, make sure you can do the "easy stuff" before going for the more difficult ones.

# Types and Control flow

## Exercise 1

A popular interview question: Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

## Exercise 2

The De Morgan's laws states that given two Boolean variables a and b, one has:

- NOT(a AND b) = NOT(a) OR NOT(b)
- NOT(a OR b) = NOT(a) AND NOT(b)

Given two Boolean variables a and b, implement Java expressions for the four members above and write some code to prove that the De Morgan's law are correct for all values of a and b.

## Exercise 3

Write the following if statement as a switch statement, and then a conditional operator (?).

```
if(x==1)
   a=10;
else
   if (x==2)
      a=x*2;
   else
      a=33333;
```

## Optional Exercise 1

1. Write a program that asks the user to enter a `String` and displays the longest palindrome that is a substring of it (or one of the longest if several). For example, if the user enters the word "aaabcbaababb" then the program should display "aabcbaa".

2. In the worst case, how many operations will your program do, with respect to the length of the String input by the user?

# Arrays and Methods

## Exercise 4

In a class Exercise4, write the following static methods and test them in your main:

1. A method passing an integer i and an array of integers x as arguments and returning the Boolean true if i appears in x and false otherwise.
2. A method passing an integer i and an array of integers x as arguments and returning the number of times i appears in x.
3. A method passing an array of integers x as argument and returning the maximal element in x.
4. A method passing an array of integers x as argument and returning the first index of the maximal element in x.
5. A method passing two arrays of integers x and y of the same length (you can assume it) as arguments and copying the content of x into y.
6. A method passing a two-dimensional array as argument and printing its elements starting with the first row, then second row, etc...

# Classes and Objects

Unless stated otherwise, all the fields should be private. You should also test all your answers in a main method.

## Exercise 5

Define a class Date with three integer fields day, month and year.

1. Write a getter and setter for each field (can you do that automatically?)
2. Write a constructor passing three integers as arguments for the day, month and year (can you do that automatically?)
3. Write a printDate method to print out the date.
4. Write a method equalsObject that tests (returns a boolean) whether this Date object is equal to one passed as an argument.
5. Write a method equalsFields that tests (returns a boolean) whether the fields of this Date object are equal to ones of the object passed as an argument.
6. Write a method arrayDate that creates and returns an array of Date of size 5 where all elements are this Date.

## Exercise 6

1. Implement a class `Singer` with three private fields:

- a field `name` of type String,
- a field `age` of type int,
- a field `professional` of type Boolean.

In your class Singer, implement a constructor which creates an instance of the class Singer based on a name and an age and which sets initially professional to true.

2. Provide getters for all the fields.

3. Provide a setter to modify the age of the singer with an age taken as parameter. You should only allow the age to be modified if the new age is greater than or equal to the current age. If this is not the case, nothing should happen.

4. In your class Singer, implement a method `reverseProfessional` which changes the Boolean professional: if professional is true then it becomes false, and if it is false then it becomes true.

5. In your class Singer, implement a method `isAdult` which returns the Boolean true if the Singer is at least 18 years old.

6. In another class called `Test`, implement a main method in which you should:

- Create a singer with your own name and age.
- Print "Adult" if the singer you have just created is at least 18.
- Print the name of the singer you have just created.

7. Implement another class `Duet` with two fields `singer1` and `singer2` both referring to an object of the class Singer. Your fields should be private.

8. In your class Duet, provide getters and setters for both fields, and implement a constructor which creates an instance of the class Duet, passing two objects of the class Singer as parameters.

9. In the class Duet, implement a method `bothProfessional` which returns true if the two singers of the duet have true for the field professional, and false otherwise.

10. In the main method of your class Test, create a duet composed with the two following singers:

- Chris aged 34,
- Maria aged 30,

and test your answers from the previous question.

11. Implement another class `Quatuor` with two fields `duet1` and `duet2` both referring to an object of the class Duet. Your fields should be private. Implement a constructor and setter and getter for both fields.

12. In the main method of your class test, create two new duets of your choice and a band. Display the age of the first singer of the second duet of your band.

## Exercise 7

1. Implement a class `Circle` with an instance field `radius` of type `int`, and a final static field `PI` of type `double` initialized at 3.14. What does that mean that a field is `final`?

2. Implement a constructor with one parameter of type int for the radius and getter and setter for the radius. The setter method for the radius should check that the radius is only modified by a positive integer. Implement also a constructor with no parameter, and initialising the radius at 1.

3. Implement a private method which returns the surface of an object Circle. If a circle has radius $r$, its surface is $\pi r^2$. We will use the static field `PI` defined above.

4. Add a private static field `numberOfCircles` of type `int` initialized at 0, and getter and setter methods.

5. Add a private static field `totalSurface` of type `double` initialized at 0, and getter and setter methods.

6. Every time a constructor is used, we would like the `numberOfCircle` to be incremented by 1 and the `totalSurface` static field to be updated, adding the surface of the circle that has just been created. How can you implement this?

7. In another class, in a main method, use the previous questions to compute the total surface of three circles of radius 1, 5 and 8 respectively. If you have not made a mistake you should get 282.6.

## Exercise 8

Implement a class Vehicule with a private integer field `speed`, a public static field `standardSpeed` and a method usedFuel returning a String "high" if the field speed is greater than standardSpeed and "low" otherwise. Implement a subclass of Vehicule called Bike. Override the method usedFuel to always return "no fuel".

Play around with these two classes. What methods and fields are inherited, and which ones are not?

# Optional fun

## Exercise 9

We are going to code the game 2048. If you do not know this game, have a go here: https://play2048.co/

You have a grid (of dimension $4 \times 4$ in the real game, but we are going to generalise the game on a grid of dimension $n \times n$ for any $n$), with squares which are empty or contain numbers that are powers of 2: 2, 4, 8, 16,... The aim is to get $2048 = 2^{11}$ in one of the squares.

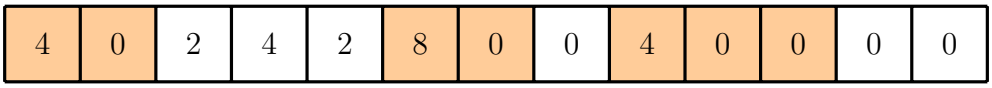
From now on, empty squares will be encoded as squares containing 0.

The player repeatedly can choose to go Up, Down, Left or Right, which will update the grid in the way described below.

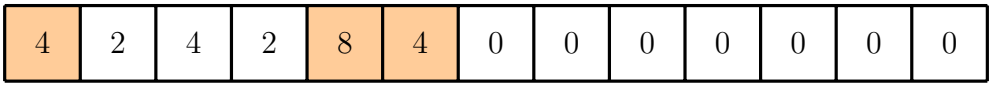
1. First, consider a line of numbers, for example:


   From left to right, every time two identical numbers are found, they are put together (0 are ignored), like in the picture:


   Then the chosen elements are added and the sum is put in the first square and 0 is put in the last square, like in the picture (the other non coloured elements are not changed):

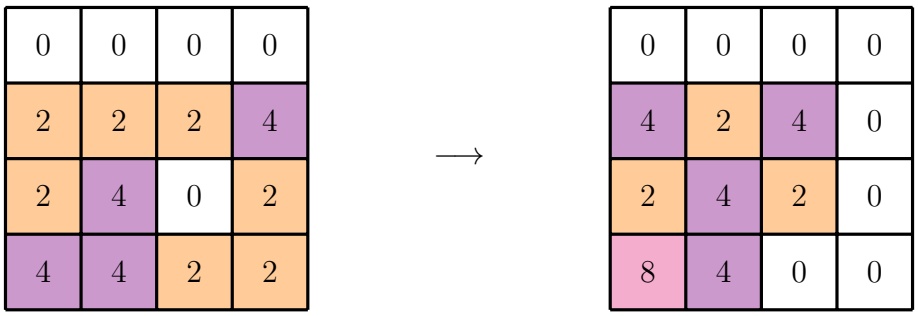



   Finally, every non zero element is pushed to the left, putting all the 0 at the end of the line, like in the picture:

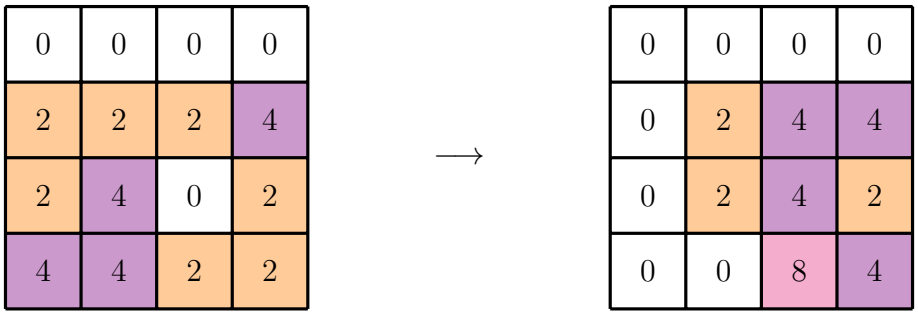



   This shows how to update a line.

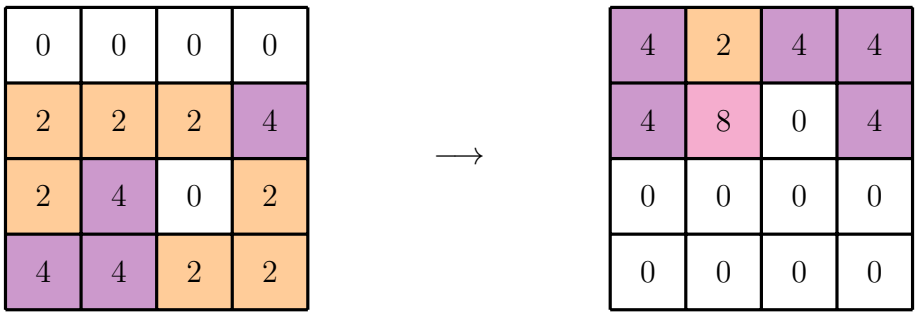
2. If the player decides to go Left, all the rows in the grid are updated as described above, like in the picture:



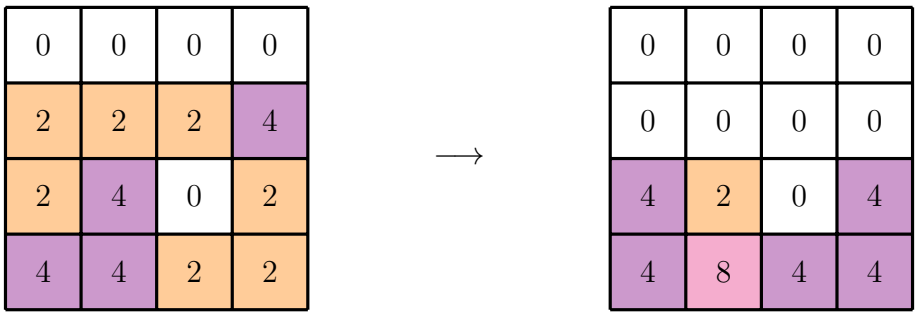

3. If the player decides to go Right, all the rows are reversed, updated as explained above and then reversed again, like in the picture:




4. If the player decides to go Up, all the columns are updated as explained above for the lines (from top to bottom) like in the picture:




5. Finally, if the player decides to go Down, all the columns are reversed, updated as explained above for the lines (from top to bottom), and then reversed again, like in the picture:




6. Once the update is done, a number 2 is added randomly on one of the empty square (containing 0).

The game ends when either 2048 appears in one of the square - and in this case, the player wins, or when going in any of the direction has no effect on the grid - and in this case the player loses.

We are going to code this game. At every step, you should test all your methods.

1. Implement a class `Line` containing one field of type `int[]` and a constructor taking as parameter an integer $n$ and constructing a Line of dimension $n$ containing only 0's.

2. In your class `Line`, implement a method which updates the line as explained above.

3. Implement a class `Grid` containing one field of type `int[][]` and a constructor taking as input an integer $n$ and creating a grid of dimension $n \times n$, with 0's and a 2 in position $(0, 0)$.

4. In your class `Grid`, implement a method taking as parameter an integer $i$ and returning an object of type Line which corresponds to the row $i$ of the grid.

5. In your class `Grid`, implement a method taking as parameter an integer $i$ and returning an object of type Line which corresponds to the row $i$, in reverse order.

6. In your class `Grid`, implement a method taking as parameter an integer $i$ and returning an object of type Line which corresponds to the column $i$ of the grid.

7. In your class `Grid`, implement a method taking as parameter an integer $i$ and returning an object of type Line which corresponds to the column $i$, in reverse order.

8. Using the methods written so far, in your class `Grid`, implement four methods which update the grid depending on whether the player goes Up, Down, Left or Right. To add randomly a 2 in an empty square, you can use the following code, which gives a random integer between 1 and $m$.

```
int r = (int)(Math.random() * m + 1);
```

You can now simulate a complete game: display the initial grid, ask the player whether they want to go Up, Down, Left or Right, update the grid accordingly, and start again... until 2048 is reached or the grid cannot change anymore.

## Exercise 10

In this exercise, we will implement three classes: `Employee`, `Job` and `Company`.

1. Implement the three classes knowing that:
- `Employee` has a field `name` of type String, a field `job` which refers to an object of the class `Job` and a field `company` which refers to an object of the class `Company`.
- `Job` has a field `name` of type String and a field `wage` of type int.
- `Company` has a field `name` of type String, a field `numberOfEmployees` of type `int` initialised at 0 and a field `listOfEmployees` which is an array of objects referring to the class `Employee`.

Provide constructors, getter and setter methods.

2. In a class `Test`, in a main method, create a job "manager" with wage 3000, a company "mycompany" with 0 employees and array of length 0 (empty array) for `listOfEmployees`, and an employee having your own name, the job and company that have just been created.

3. Everytime an employee is created in a given company, we would like the field `numberOfEmployees` of the company to be incremented. Modify your code to achieve this.

4. Similarly, the employee has to be added to the array `listOfEmployees`. Modify your code to achieve this. You can write a method in the class Employee which takes as parameters an array of Employee t and an Employee e and return the array where e has been added at the end of t.

5. In the main method you have implemented, test that the name of the Employee stored in the array `listOfEmployees` of `myCompany` is your own name.

6. In the class containing the main method, define a method taking as parameter a company and printing the wages of all its employees. Test it.