INFO0062 - Object-Oriented Programming

Presentation of the project

Jean-François Grailet

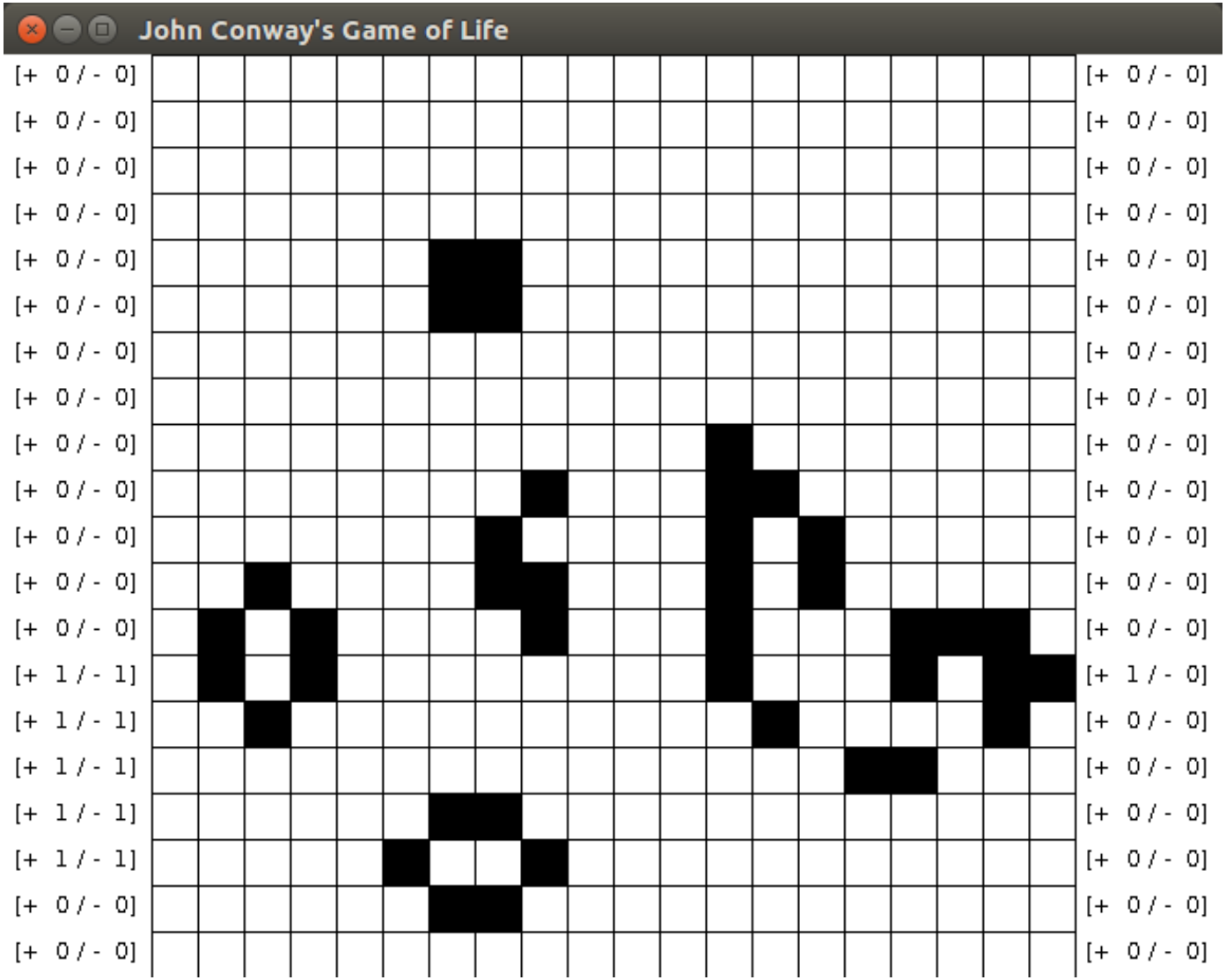
University of Liège

Faculty of Applied Sciences

Academic Year 2017 - 2018

Project

John Conway's Game of Life



Statement

- You are asked to implement John Conway's *Game of Life* in Java.
- This is an **individual** project.
- You are free to implement it in console or in a graphical fashion.
- The `main()` method must be located in a `GameOfLife` class.
- Your solution should receive two parameters `n` and `m`.
  - More on these parameters in a few slides.

Evaluation and submission

- You project must compile and run as expected.
  - Don't forget to check your project works on Network 8!
- You project must use an object-oriented architecture.
- Submit your project as a ZIP archive named `oop_lastname_firstname.zip`.
  - Prefer writing your first/last names with only the first letter in uppercase[1].
- Your archive should contain **only** `.java` source files.
- Submit this archive to `oop@montefiore.ulg.ac.be`.
- The subject of the e-mail should be: `OOP - 2017 - lastname firstname`

[1] FR: en majuscule

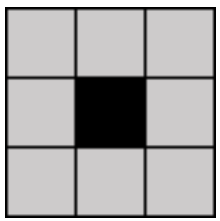
John Conway's Game of Life (I)

- The *Game of life* consists in simulating a cellular automaton.
  - It works as a succession of generations (or steps, states).
  - It was invented in 1970 by British mathematician John Conway[2].
- The setting is a square grid of arbitrary size.
- Cells are initially placed on this grid.
- The game consists in deciding whether these cells are going to live or die.
  - If they die, they disappear from the grid at the next generation.
  - New cells can appear under a certain condition.

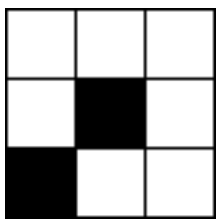
[2] https://en.wikipedia.org/wiki/Conway's_Game_of_Life

John Conway's Game of Life (II)

- The fate of a cell depends on the amount of cells in its 8 adjacent squares.
  - I.e., the cells appearing in the 3x3 square which the cell is the center.
- A cell (in black) with its adjacent squares (in grey; max. 8 cells):

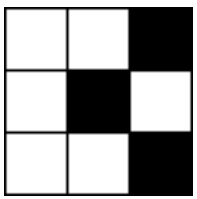


- A cell with only one neighbor:

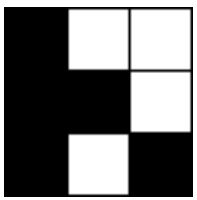


John Conway's Game of Life (III)

- For each living cell
  - 0 or 1 neighbor: the cell dies of loneliness.
  - 2 or 3 neighbors: the cell survives.
  - 4 or more neighbors: the cell dies of asphyxiation.
- Survival scenario:

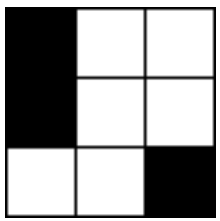


- Death by asphyxiation scenario:



John Conway's Game of Life (IV)

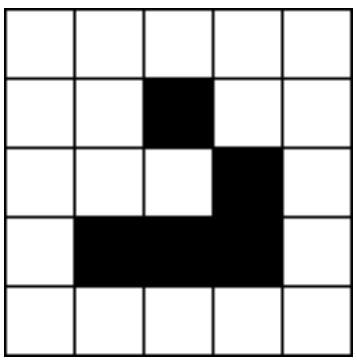
- If an empty square has exactly 3 neighbors, a new cell appears.



- **Nota bene:** the decision for all squares/cells comes before next generation!
  - Suppose a cell should die, therefore disappear at next generation.
  - It is still taken into account to make the decision for its neighboring cells.

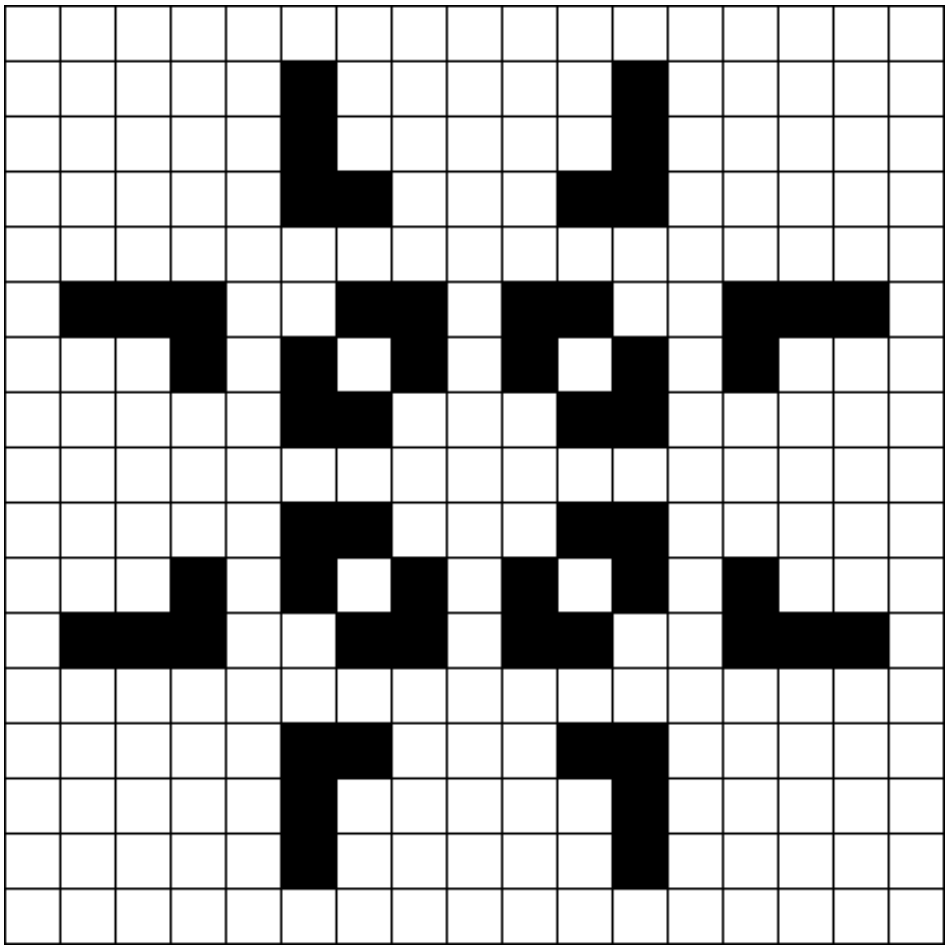
John Conway's Game of Life (V)

- **Property of this game:** specific spread of cells can result in
  - stable or oscillating patterns
  - moving patterns
  - self-replicating patterns
- A simple moving pattern is the *glider*.
  - It moves diagonally across the grid.



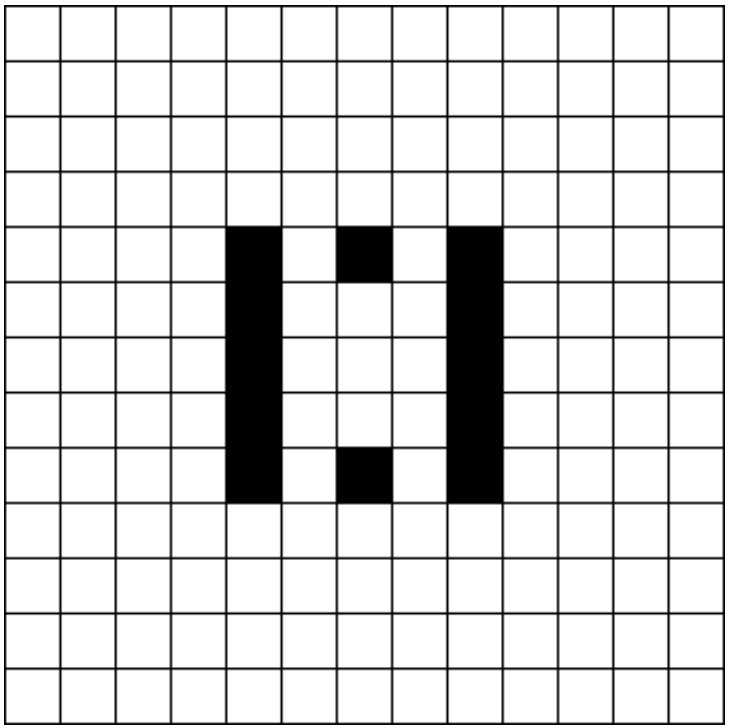
John Conway's Game of Life (VI)

- A more complex pattern is the *pulsar* (oscillating).



John Conway's Game of Life (VII)

- Alternatively, a *pulsar* can be initiated by an *exploder*.
- The *pulsar* will appear after a few periods.



John Conway's Game of Life (VIII)

- Other patterns you can look up on the Internet
  - Blinker
  - Pentadecathlon
  - Glider gun
  - Etc.
- Cf. `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`
- You can also find online implementations of this game to toy with it.
  - Example: `https://bitstorm.org/gameoflife/`

What is asked for this project

- Your final program should accept two parameters `n` and `m`.
  - `n` is the size of the square grid (`n` x `n` grid).
  - `m` is the size of a (centered) square region (`m` x `m`) within the grid.
  - You will randomly generate cells in the `m` x `m` region.
- There are a few requirements for these parameters.
  - $0 < \texttt{m} \leq \texttt{n}$
  - `n` and `m` must have the same parity (i.e. both odd or both even)[3]

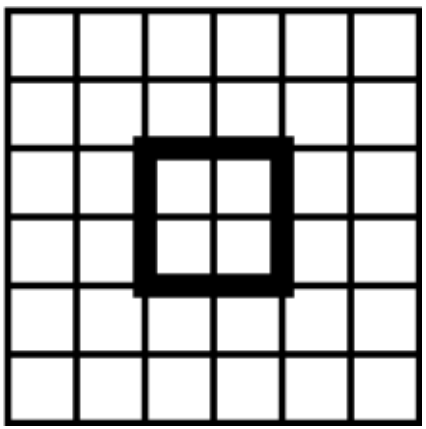
[3] FR: tous les deux pairs ou tous les deux impairs

What is asked for this project (II)

- For example, you could run your project with this command (`n` = 6 and `m` = 2):

```
java GameOfLife 6 2
```

- This would produce the following grid (inner square is the `m` x `m` region):

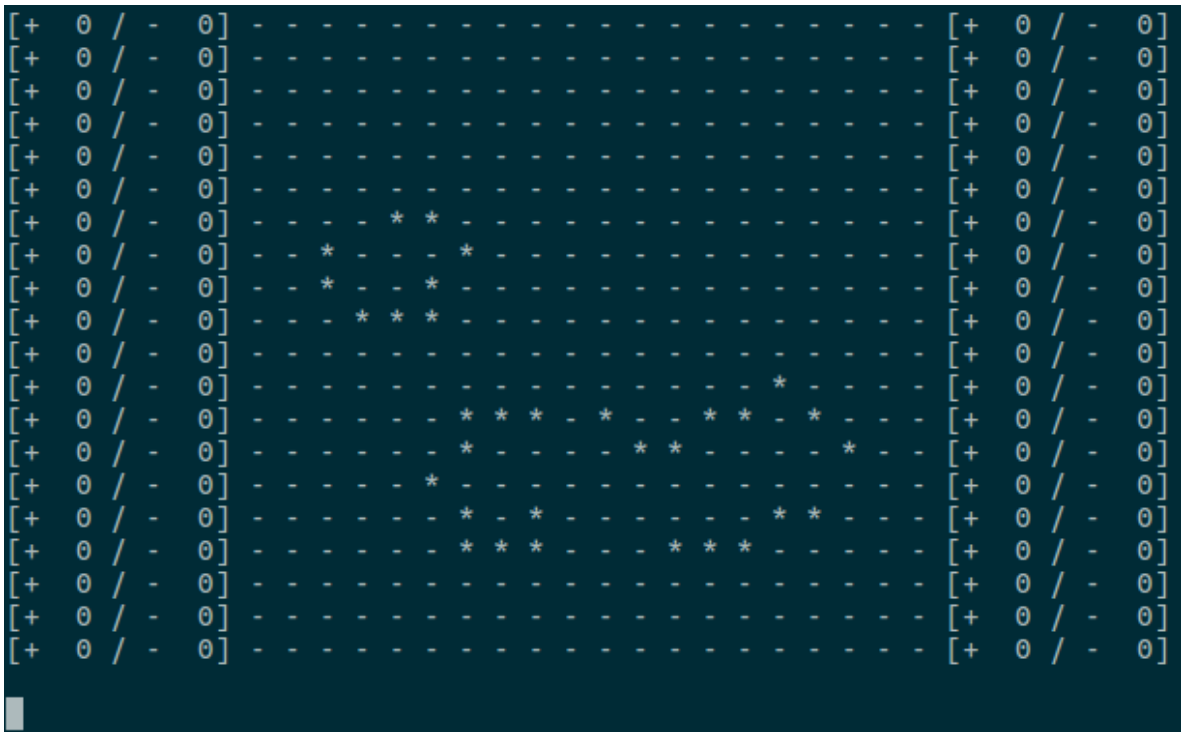


Display

- You have two options to display your grid:
  - either print each generation in console (text output),
  - either create a GUI with an image refreshed at each generation.
- Your choice will not influence your final grade.
- In both cases, squares from the left/right borders must be annotated.
  - `[+ x / - y]`
  - `x` = amount of cells which were born in this square
  - `y` = amount of cells that died in this square

Display (II)

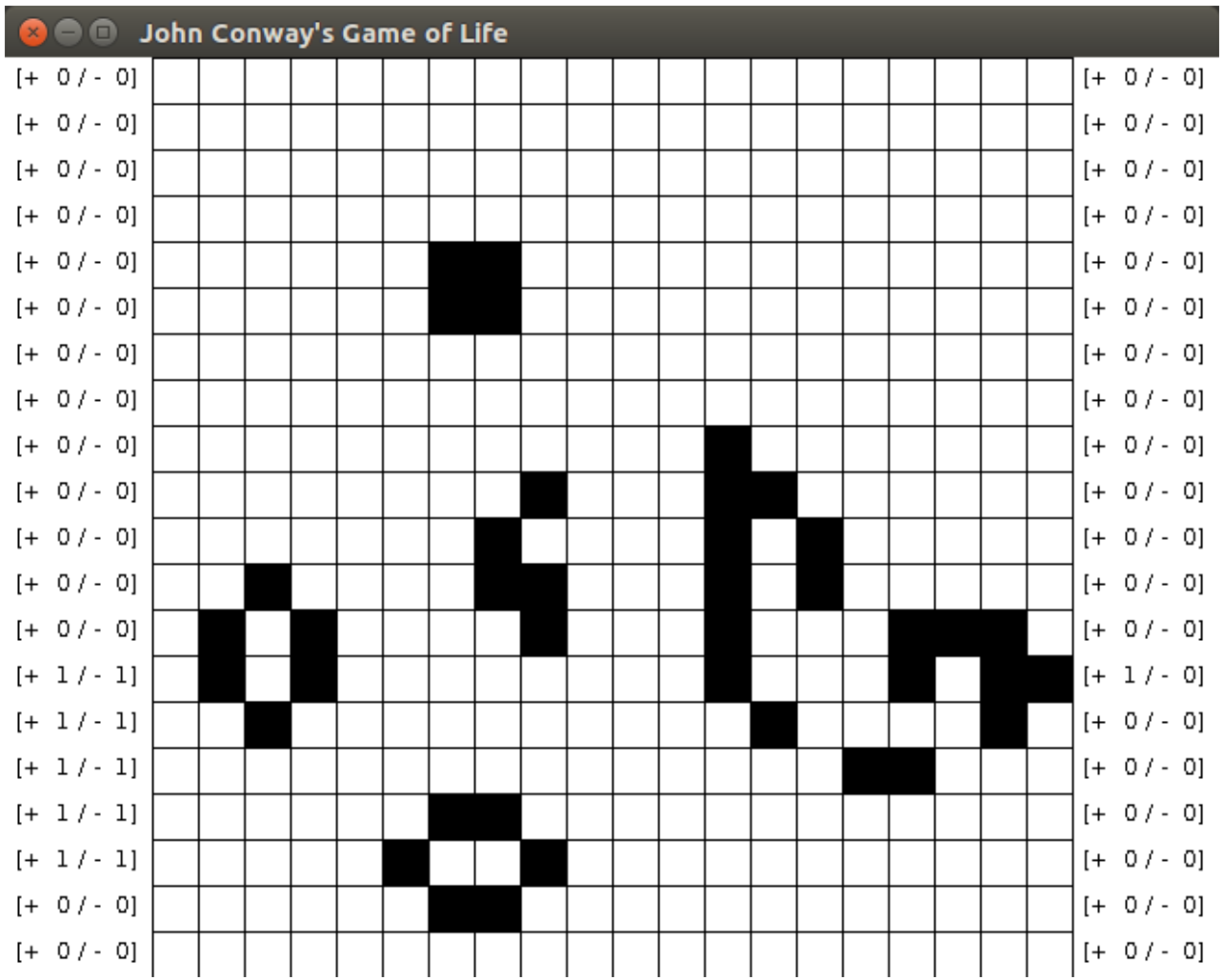
- Example of text display in console:



- **N.B.:** cells should be displayed as `*` and empty squares as `-`.

Display (III)

- Example of graphical display:



Tips & tricks

General advice

- Always keep in mind that this is an **object-oriented** programming project.
- You are therefore expected to
  - model entities of the problem as objects (e.g., each square is an object),
  - give them relevant responsibilities,
  - put to practice OOP concepts (e.g. encapsulation, inheritance).
- Functionality of the project count **at best for 8/20** of the final grade.
- Typically, trying to implement everything with one or two classes is a bad idea.

Tips & tricks for the project

- To help you, we will see today how you can
  - handle the successive generations
  - delay between two generations
  - handle random generation of cells
- And also
  - some tips to display in console
  - what you can use from the Java library
- Other useful tricks will be reviewed during the next exercise sessions.

Handling generations

- You can use an infinite loop.
  - We don't ask you to check when a population has stabilized in order to stop.

```
// Initialization instructions
while(true)
{
    ... // Generate and display a generation
}
```

- To stop your program from running, press CTRL + C.
  - You can also close the terminal, but this is not practical.

Delaying between generations

- With a delay between generations, you will clearly see how cells evolve.
- In Java, you can use `Thread.sleep()` to implement a delay.
  - It receives an integer amount of milliseconds.
- Note that you have to catch the `InterruptedException`.
  - You will see exceptions in an upcoming chapter of the course.

```
try
{
    Thread.sleep(500); // 0,5s of waiting
}
catch(InterruptedException ex)
{
    System.err.println(ex.getMessage());
}
```

Random generation of cells

- You can use the `Random` class from the Java library.
- Note that you have to *import* it.
  - I.e., first line of you code should be `import java.util.Random;`
- `nextInt()` gives you a random integer between 0 and a limit (excluded).
- Use this to draw a number to decide if a square contains a cell or not.

```
import java.util.Random; // First line of the .java file

...
Random rand = new Random();
int draw = rand.nextInt(2); // 0 or 1 (50% of chance)
if(draw == 0)
    ... // This square will have a cell
```

Displaying in console

- For readability, consider letting a blank line between each grid display.
- Use `String.format()` to align the numbers properly.
  - Two parameters: a format and the `String` to format.
  - In particular, format code `"%3d"` always takes the space for 3 digits.
  - I.e., numbers between 1 and 999 will be aligned correctly.

```
String resStr += "[+";
resStr += String.format("%3d", nbBirth) + " / ";
... // Rest of the display
```

Java native library

- You can use classes from the Java native library to ease your task.
- Note that most of these classes need to be *imported* (just like `Random`).
- It's up to you to look for these classes and read their documentation.
  - If you follow a computer science cursus, you will later do that on a regular basis.
- In particular, students wishing to implement a graphical solution should look at
  - `java.awt.image.BufferedImage`
  - `java.awt.Graphics2D`
  - `javax.swing.JFrame`
  - `javax.swing.JLabel`
  - `javax.swing.ImageIcon`

Coding style and documentation

About coding style

- Use meaningful variable, method and class names.
- For instance, compare the readability of the two following methods:

```
public static int a(int b) {
  if (b <= 0)
    return 1;

  return b * a(b - 1);
}
```

```
public static int factorial(int input) {
  if (input <= 0)
    return 1;

  return input * factorial(input - 1);
}
```

About coding style (II)

- Convention for variable/method names is to use lowercase[4] words.
- Starting from the second word, the first letter is uppercase[5].
  - E.g. `priceWithTaxes`.
- For constants, the convention is to use uppercase words separated by "_".
  - E.g. `TVA_IN_BELGIUM`.
- For classes and interfaces, lowercase words that begin with an uppercase letter.
  - E.g. `TaxesCalculator`.

[4] FR: en lettre minuscule
[5] FR: en lettre majuscule

About coding style (III)

- Two conventions for curly braces related to blocks (choose one):

```
while (true) {

}
```

```
while (true)
{

}
```

- Indentation must be coherent and strongly respected:

```
public class MyClass {
      public static void m1() {
      instruction1;
    instruction2;
    }

public static void m2() {
 instruction1;
     instruction2;
}
    }
```

```
public class MyClass {
  public static void m1() {
    instruction1;
    instruction2;
  }

  public static void m2() {
    instruction1;
    instruction2;
  }
}
```

About coding style (IV)

- You can insert spaces or empty lines in your code to improve readability.

```
public class Probability{
  public static double arrange(int n,int k){
    return (double)factorial(n)/factorial(n-k);
  }
  public static int factorial(int input){
    if(input<=0)return 1; return input*factorial(input-1);
  }
}
```

```
public class Probability {
  public static double arrange(int n, int k) {
    return (double) factorial(n) / factorial(n - k);
  }

  public static int factorial(int input) {
    if (input <= 0)
      return 1;

    return input * factorial(input - 1);
  }
}
```

About coding style (V)

- Choose a maximal number of characters per line of code.
- Common convention: 80 columns rule.
- But you can also use 100 columns if you prefer.
- **The most important is to make consistent choices and to respect them.**

Documentation

- You can document your code using comments.
- It is useful to remember what you did, but also to inform other programmers.
- Typically, you should at least describe the role of a class.

```
/*
 * This class offers a set of static methods to perform various
 * calculations relative to the probability theory.
 */
public class Probability {
   ...
}
```

Documentation (II)

- You should describe the role of a method by detailing
  - its parameter(s) (if any),
  - its returned value (if any),
  - the instantiation context of its exception(s) (if any).

```
/*
 * This method tests whether the input parameter is odd and
 * returns a boolean to confirm it. In the case where the input
 * parameter is negative, a MyException exception is thrown.
 */
public static boolean isOdd(int input) throws MyException {
  if (input < 0)
    throw new MyException();

  return (input % 2) == 1;
}
```

About language(s)

- You can choose English or French for your documentation.
- Prefer English for the names of variables, methods and classes.
- However, once you chose a language, **stick with it**.

```
/**
 * Cette méthode teste si un entier positif est impair.
 *
 * @param  input        L'entier à tester.
 * @return boolean      Vrai si l'entier est impair, faux sinon.
 * @throws MyException Lancée quand un entier négatif est donné.
 */
public static boolean isOdd(int input) throws MyException {
  if (input < 0)
    throw new MyException();

  return (input % 2) == 1;
}
```

Graphical SSH

Graphical SSH

- If you do a graphical solution, testing on Network 8 requires “graphical SSH”.
- Please note that testing remotely a graphical solution may be slow.
  - Therefore, consider testing on Network 8 before submitting, as a final verification.
- GNU/Linux: just add the `-YC` options to `ssh`.
- macOS: idem, but you must install XQuartz  first.
- Windows:
  - You must install Xming  first.
  - Follow this guide to integrate Xming into Putty  .
  - Check `Enable compression` in the `Connection` → `SSH` panel (see next slide).

Graphical SSH (II)