PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA

الجمهورية الجزائرية الديمقراطية الشعبية

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

SCHOOL OF APPLIED SCIENCES --TLEMCEN--

وزارة التعليم العالي والبحث العلمي

المدرسة العليا في العلوم التطبيقية -تلمسان-

Course : Object oriented programming
Practicle Work N°1 : getting started

The object-oriented programming lab sessions assume that your working environment is configured for Java 7 or higher (Java 8 is strongly recommended).
To check the versions of your JDK and JRE, type the following commands in a terminal: javac -version and java -version.
To install the JDK on a personal computer, you will need to download it from Oracle's website for Windows, or install the OpenJDK package for a Linux system.
You are free to use your preferred tool for editing your source code. However, it is more practical to use an Integrated Development Environment (IDE). The most well-known IDEs for Java development are **NetBeans**, **Eclipse**, and **IntelliJ**.

I. the « Hello » program

Enter, compile, and run a Java program that displays `Hello everyone!` on the console screen.
Write a public, executable class named **Bonjour**, placed in a file named **Bonjour.java**.
There are three important "details" not to forget:

1. A class is executable **if and only if** it contains a method with **exactly** the following signature:
   `public static void main(String[] args)`
2. You compile a Java file by giving its **file name** (e.g., `javac Bonjour.java`)
3. You run a Java program by giving its **class name** (e.g., `java Bonjour`)

Observe what happens when you fail to follow any of the three rules above.

II. The mean class

Create a file named **Mean.java** that contains the **Mean** class as seen in the course:

```
class Mean {
   int notes = 0;
   int numberOfNotes = 0;

   void addNote(int note) {
      notes += note;
      numberOfNotes += 1;
   }

   double calculateAverage() {
      return ((double) notes) / numberOfNotes;
   }
}
```

1- How can you test this class?
2- What happens if you calculate the average without having entered any notes?
3- What happens if you remove the (double) on line 11? What happens in that case if you calculate the average without entering any notes?

4- Can you put the main method directly inside the Mean class, or is it absolutely necessary to create a separate class called TestMean?

We have seen that the method **System.out.println** displays a string on the screen and then moves to a new line.
Similarly, the method **System.out.print** displays a string **without** a line break.
Finally, the method **System.out.printf** allows you to write a **formatted string** to the standard output; its behavior is similar to the **printf** function in the C programming language.

The **Scanner** class is a utility class that greatly simplifies reading from the standard input. It provides reading methods for all primitive types and strings. This class is easy to use, as shown in the following example.

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

To make the class accessible, you need to add the following line at the very beginning of the file.

```
import java.util.Scanner;
```

Using the display methods and the `Scanner` class, define a class named `TestMean`. This class contains a `main` method that asks the user to enter a certain number of grades, then displays the average of the entered grades.

III. Calculating the factorial

1- Write an executable class with a method: long factorial1(int n) that calculates the factorial of **n**, where n!=n×(n−1)×(n−2)×…×3×2×1
2- Find the value of **n** from which integer overflow makes this method unusable.

Note: One way to verify that the value of n!n!n! is correct, assuming that (n−1)!(n - 1)!(n−1)! is correct, is to display the ratio n!\(n - 1)!

1- To overcome this overflow problem, write a new function using **java.math.BigInteger** objects: BigInteger factorial2(int n) and observe that now you can go practically as far as you want.
2- Could we have given the same name to the functions **factorial1** and **factorial2**?

IV. Seconds Converter

We want to write a program that converts seconds into minutes and hours.

1. Propose a class **Times** with a method String convert(int seconds) which takes a number of seconds as a parameter and returns a string showing the number of hours, minutes, and seconds represented by that number of seconds. For example, calling t.convert(4567) should return the string "1h 16mn 7s".
2. How can you test this method?

V. The arguments of the main function

In its general form, the command that starts the execution of a Java application is written as:

java ClassName string0 string1 ... string(k-1)

Where string0, string1, ..., string(k-1) are strings without spaces, separated by spaces or tabs. These strings are accessible inside the application through an array, usually called args, which is declared in the header of the main function. For example, if an application is launched by the command...

java ClassName Info "Second cycle" 2006 2+3=5

Then the `main` function receives an argument which is an array consisting of the four strings `"Info"`, `"Second cycle"`, `"2006"`, and `"2+3=5"`. Note that:

- When an argument contains spaces (e.g., *Second cycle*), it must be enclosed in quotation marks.
- Numbers are **not** recognized as such, but as strings.
- The only separators recognized at this level are spaces and tabs; for example, 2+3=5 forms a single argument.

Write a class **Calcul** that is executed with three arguments: two numbers and an operator between them, and that displays the result of the operation indicated on the given numbers.

Example of execution:

java Calcul 1200 + 400

Result : 1200 + 400 = 1600