Cours MOOC EPFL d'introduction à la programmation orientée objet, illustré en Java

First assignment
Classes and objects

J. Sam & J.-C. Chappelier

1 Exercise 1 — BMI

The goal of this exercise is to create « patients » that have a weight and a height, and to compute their « Body Mass Index » (BMI) (IMC in French).

1.1 Description

Download[1] the source code available at the course webpage and complete it.

WARNING: you should modify neither the beginning nor the end of the program, only add your own lines as indicated. It is therefore very important to respect the following procedure (the points 1 and 3 concern only Eclipse users):

1. remove automated formatting of the code in Eclipse:
   `Window > Preferences > Java > Editor > Save Actions`
   (and untick the formatting option if it's on);
2. save the downloaded file as `Imc.java` (respect the upper case). If you work with Eclipse, save it to
   `[projectFolderUsedForThatExercise]/src/`;
3. refresh the Eclipse project where the file is stored (right click on the project > "refresh") in order to take into account the new file;
4. write your code between these two provided comments:

[1] We actually mean "*download*", i.e. save the file as it is, not copy-paste its contents from the browser.

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. save and test your program to be sure that it works properly, for example using the values given below;
6. upload the modified file (still named `Imc.java`) in "OUTPUT submission" (not in "Additional"!).

The provided code does the following things:

- creates a patient,
- displays patient data and his BMI.

These two step are repeated twice with different values.

The implementation of the `Patient` class is missing and this is what you are asked to write.

A patient is characterized by a weight and a height. You will name the corresponding attributes respectively `masse` and `hauteur` (this is to be strictly followed).

Methods specific to a patient are :

- a method `init` taking two `double` as parameters, the first to initialize the patient weight and the second for his height ;

  these data will be assigned to the patient only if they are positives ; if not, **weight and height will be initialized to zero** ; in order to simplify, there won't be any other check on these data ;

- a method `afficher` that displays on the terminal the attributes of the patient by ***strictly*** respecting the following format:
  `Patient : <weight> kg pour <height> m`
  where `<weight>` is to be replaced by the weight of the patient and `<height>` by his height ; **this output will be ended by a line break**. The output of the decimal numbers will have a one digit precision for the weight and the height. You will use the following example lines of code that displays the two doubles with one digit precision :

```
double x = 1.2456;
double y = 1.4568;
// affiche : "1.2 et 1.5"
System.out.printf("j'affiche deux doubles formattés %.1f et %.1f",
   x, y);
```

- a method `poids` returning the patient's weight ;
- a method `taille` returning the patient's height ;
- a method `imc` returning the BMI of the patient : his weight divided by the square of his height ; **if the height is zero, the method will return zero**.

These methods will be part of the public interface of the class.

An execution example is provided below.

1.2 Execution example

```
Patient : 74.5 kg pour 1.8 m
IMC : 24.3265306122449
Patient : 0.0 kg pour 0.0 m
```

2 Exercise 2 — Piggy bank

The goal of this exercise is to simulate a piggy bank in which we deposit and withdraw money. We wish to use it to pay some specific amounts.

2.1 Description

Download[2] the source code available at the course webpage and complete it.

WARNING: you should modify neither the beginning nor the end of the program, only add your own lines as indicated. It is therefore very important to respect the following procedure (the points 1 and 3 concern only Eclipse users):

1. remove automated formatting of the code in Eclipse:

   `Window > Preferences > Java > Editor > Save Actions` (and untick the formatting option if it's on);

[2]We actually mean "*download*", i.e. save the file as it is, not copy-paste its contents from the browser.

2. save the downloaded file as `TestTirelire.java` (respect the upper case). If you work with Eclipse, save it to `[projectFolderUsedForThatExercise]/src/`;

3. refresh the Eclipse project where the file is stored (right click on the project > "refresh") in order to take into account the new file;

4. write your code between these two provided comments:

```
/*****************************************
 * Completez le programme a partir d'ici.
 *****************************************/

/*****************************************
 * Ne rien modifier apres cette ligne.
 *****************************************/
```

5. save and test your program to be sure that it works properly, for example using the values given below;

6. upload the modified file (still named `TestTirelire.java`) in "OUTPUT submission" (not in "Additional"!).

The provided code creates a piggy bank and manipulates it (empty it, shake it, display its content etc.).

This program also asks the user the budget he wishes to put in his/her holidays.

If the piggy bank contains enough money (this budget or more), it tells how much money would be left after the holidays. In the opposite case, it indicates the amount missing to go on holidays with the desired budget.

The implementation of the class `Tirelire` is missing and you are asked to provide it.

A piggy bank is characterized by the *montant* (amount) it contains. You will name this attribute with the name *montant* in your code.

Its specific procedures are :

- a method `getMontant` returning the amount of the piggy bank;
- a method `afficher` displaying the informations about the piggy bank in the following format :

  - `Vous etes sans le sou.`
    if the piggy bank is empty (the accents have deliberately been removed).
  - `Vous avez : <amount> euros dans votre tirelire.`
    in the opposite case (where `<amount>` is the money amount in the piggy bank, displayed without any specific formatting)

- a method `secouer` displaying in the terminal the message `Bing bing`, followed by a line break, in the case where the piggy bank is not empty, and does display anything otherwise;
- a method `remplir` adding a given amount, given as `double` parameter, in the piggy bank. Only the positive amounts will be accepted (otherwise we don't do anything);
- a method `vider` (re)initializing the amount to zero;
- a method `puiser` allowing to withdraw in the piggy bank an amount given as parameter. If the amount is negative it will be ignored. If the amount passed as parameter is bigger than the amount in the piggy bank, the piggy bank is emptied. The `puiser` method does not return anything.
- a method `calculerSolde` that returns the difference between the amount in the piggy bank and the budget we are willing to spend (a `double`). If the budget is negative (or zero), the method `calculerSolde` will return the piggy bank amount.

These methods will be part of the public interface of the class.

Two execution example are provided below.

2.2 Execution examples

```
Vous etes sans le sou.
Vous etes sans le sou.
Bing bing
Vous avez : 550.0 euros dans votre tirelire.
Vous avez : 535.0 euros dans votre tirelire.

Donnez le budget de vos vacances :
450
Vous etes assez riche pour partir en vacances !
il vous restera 85.0 euros a la rentree
```

or

```
Vous etes sans le sou.
Vous etes sans le sou.
Bing bing
Vous avez : 550.0 euros dans votre tirelire.
Vous avez : 535.0 euros dans votre tirelire.

Donnez le budget de vos vacances :
1250.0
Il vous manque 715.0 euros pour partir en vacances !
```