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

First assignment
Classes and objects

J.-C. Chappelier & J. Sam

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 the source code available at the course webpage and complete it according to the instructions below.

WARNING: you should modify neither the beginning nor the end of the provided file. It's thus mandatory to proceed as follows:

1. save the downloaded file as `imc.cc` or `imc.cpp`;
2. write your code between these two provided comments:

```
/*******************************************
 * Complétez le programme à partir d'ici.
 *******************************************/

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

3. save and test your program to be sure that it works properly; try for instance the values used in the example given below;

4. upload the modified file (still named `imc.cc` or `imc.cpp`) in “My submission” then “Create submission”.

The provided code does the following:

- creates a patient;
- reads the patient’s weight and height from the standard input;
- displays the patient’s data and BMI.

The last two steps are repeated as long as neither the weight nor the height are zero.

The implementation of the `Patient` class is missing. This is what you have to provide.

A patient is characterized by a weight and an height. You shall name the corresponding attributes respectively `masse` and `hauteur` (this is absolutely required).

Methods specific to a patient are:

- a method `init` taking two `double` as parameters, the first to initialize the patient weight (`masse`) and the second for its height (`hauteur`);

  these data will be assigned to the patient only if they are positive; if not, weight and height shall **both be initialized to zero**; in order to simplify, there won’t be any further check on these data;

- a method `afficher` that displays on the terminal the attributes of the patient, ***strictly*** with the following format:
  `Patient : <weight> kg pour <height> m`
  where `<weight>` is to be replaced by the weight of the patient and `<height>` by its height ; **this output shall end with a line break**.

- 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 its 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

```
Entrez un poids (kg) et une taille (m) : 80.0 1.7
Patient : 80 kg pour 1.7 m
IMC : 27.6817
Entrez un poids (kg) et une taille (m) : 56.5 1.8
Patient : 56.5 kg pour 1.8 m
IMC : 17.4383
Entrez un poids (kg) et une taille (m) : 0 0
Patient : 0 kg pour 0 m
IMC : 0
```

2 Exercise 2 — Piggy bank

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

2.1 Description

Download the source code available at the course webpage and complete it according to the instructions below.

WARNING: you should modify neither the beginning nor the end of the provided file. It's thus mandatory to proceed as follows:

1. save the downloaded file as `tirelire.cc` or `tirelire.cpp`;
2. write your code between these two provided comments:

```
/*******************************************
 * Complétez le programme à partir d'ici.
 *******************************************/

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

3. save and test your program to be sure that it works properly; try for instance the values used in the example given below;

4. upload the modified file (still named `tirelire.cc` or `tirelire.cpp`) in “My submission” then “Create submission”.

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 (s)he wishes to put in his/her holidays.

If the piggy bank contains enough money (the 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 have to provide it.

A piggy bank is characterized by the amount it contains. You shall name this attribute “`montant`” in your code.

A piggy bank 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
    (Note: the diacritic has deliberately been removed on “êtes”).
  - `Vous avez : <amount> euros dans votre tirelire.`
    in the opposite case, where `<amount>` is the money amount in the piggy bank.
- a method `secouer` (means “shake”) 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 not display anything otherwise;
- a method `remplir` (means “fill/add”) adding a given amount, given as `double` parameter, in the piggy bank; only the positive amounts will be accepted (otherwise nothing is done);
- a method `vider` (means “empty”) which (re)initializes the amount of the piggy bank to zero;
- a method `puiser` (means “take/remove”) allowing to withdraw from 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 `montant_suffisant` (means "`enough_amount`") with two parameters returning `true` if the piggy bank contains enough money to spend a budget given as first parameter (of type `double`) and `false` otherwise; if there is enough money, the second parameter `solde` (of type `double` and passed by reference) will contain the amount of money that would remain in the piggy bank if we spend the given budget; otherwise it will contain the amount (a positive number) missing from the piggy bank to reach the given budget; if the budget is negative or null, the method `montant_suffisant` returns `true` and the second parameter `solde` is assigned the amount of money contained in the piggy bank : it is just like if we asked for a null budget.

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 euros dans votre tirelire.
Vous avez : 535 euros dans votre tirelire.

Donnez le budget de vos vacances : 1000.0
Il vous manque 465 euros pour partir en vacances !
```

or

```
Vous etes sans le sou.
Vous etes sans le sou.
Bing bing
Vous avez : 550 euros dans votre tirelire.
Vous avez : 535 euros dans votre tirelire.

Donnez le budget de vos vacances : 400.0
Vous êtes assez riche pour partir en vacances !
Il vous restera 135 euros à la rentrée.
```