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

Fourth assignment
Inheritance

J.-C. Chappelier & J. Sam

1 Exercise 1 — Philately

A philatelist wishes to estimate at which price he could sell his stamps. The purpose of this exercise is to write a program which allows him to do so.

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 `timbres.cc` or `timbres.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 `timbres.cc` or `timbres.cpp`) in "My submission" then "Create submission".

The provided code creates 4 stamps (of different types) and display their price.

This code is incomplete and you are asked to write the missing material.

A stamp (class `Timbre`) is characterized by : its *code* (`nom`, of type `string`), its *année d'émission* (issue year, `annee`, of type `unsigned int`), its *pays d'origine* (country of origin, `pays`, of type `string`) and its *valeur faciale* (denomination value, `valeur_faciale`, of type `double`) in francs. The denomination value is the equivalent in francs of the value printed on the stamp. You must use the suggested names for the attributes.

Our philatelist distinguishes two broad categories of stamps, which are distinguished by the way the selling price is computed.

- the *rare* stamps (class `Rare`) : possesses an extra attribute indicating the number of copies that are inventoried throughout the world ;
- the *commemorative* stamps (`Commemoratif`) : without any particular specific attribute.

You will assume that there is no rare commemorative stamp.

Selling price of the stamps The selling price of a *timbre* (stamp) of any kind is its denomination value, if the stamp is less than 5 years old. If not, the selling price is the denomination value multiplied by the age of the stamp and by the coefficient 2.5. The selling price of a *rare* stamp starts from a *prix de base* (base price) : 600 francs if the number of the inventoried copies is less than 100, 400 francs if the number of copies is between 100 and 1000 and 50 francs in any other case. The selling price of a rare stamp is then given by the formula `prix_base * (age_timbre / 10.0)`. You must define and use the following *class* constants :

- `PRIX_BASE_TRES_RARE`, with 600 as value ;
- `PRIX_BASE_RARE`, with 400 as value ;
- et `PRIX_BASE_PEU_RARE`, with 50 as value.

The selling price of a *Commemoratif* stamp is twice as much as the price of a generic one.

Basic stamps (neither rare nor commemorative) The class `Timbre` must have at least the following public methods :

- constructors that conform to the provided `main()` method, with the following order of the parameters : the code, the issue year, the country of origin, and the denomination value ; the country of origin and the denomination value will have respectively the following default values `"Suisse"` and `1.0`;
- a method `vente()` returning the selling price in the format of a `double` ;
- the method `age()` returning the age of a stamp in the format of an `unsigned int`, difference between `ANNEE_COURANTE` (current year, a provided attribute) and the issue year of the stamp ;

The stamp must be displayable using the usual << operator respecting

strictly the following format :
`Timbre de nom <code> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs`
in only one line where `<code>` is to be replaced by the code of the stamp, `<annee>` by its issue year, `<pays>`, by its country of origin and `<valeur faciale>`, by its denomination value . An example of display is given below.

Rare stamps and commemorative stamps The classes `Rare` and `Commemoratif` must have the same public methods as the basic stamps. They must also be displayable using the << operator. The differences with the basic stamps will simply be as follows :

1. the `Rare` stamps will have an additional parameter in their constructor representing the number of copies of the stamp, with 100 as default value; they will also have a method named `nb_exemplaires()` returning the number of copies of the rare stamp ;
2. the rare stamps must be displayed using ***strictly*** the following format :
   `Timbre rare (<exemplaire> ex.) de nom <nom> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs`
   in only one line where `<exemplaire>` is to be replaced by the number of copies of the stamp, and where the other informations have the same meaning as for the basic stamps; (see the execution example below) ;

3. the commemorative stamps must be displayed using ***strictly*** the following format :
   `Timbre commémoratif de nom <nom> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs`
   in only one line

Note that all the stamps (basic, rare or commemorative) have the same end of display; the last information displayed is always :

`de nom <nom> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs`
in only one line. You are asked to code this common part in a method called `afficher`.

In this exercise, you are therefore asked to code the hierarchy of classes according to the above description. Avoid code duplication and attribute shadowing, pay attention to the access modifiers of the attributes and name the classes as suggested in the statement.

1.2 Execution example

```
Timbre rare (98 ex.) de nom Guarana-4574 datant de 1960 (pro\
venance Mexique) ayant pour valeur faciale 0.2 francs
Prix vente : 3360 francs
Timbre rare (3 ex.) de nom Yoddle-201 datant de 1916 (proven\
ance Suisse) ayant pour valeur faciale 0.8 francs
Prix vente : 6000 francs
Timbre commémoratif de nom 700eme-501 datant de 2002 (proven\
ance Suisse) ayant pour valeur faciale 1.5 francs
Prix vente : 105 francs
Timbre de nom Setchuan-302 datant de 2004 (provenance Chine)\
 ayant pour valeur faciale 0.2 francs
Prix vente : 6 francs
```

The character `'\'` is not part of the output and is not followed by a newline.

2 Exercise 2 — Clash of the titans

In this exercise, you are going to have dragons and hydras fight each other.

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 `dragons.cc` or `dragons.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 `dragons.cc` or `dragons.cpp`) in "My submission" then "Create submission".

The provided main program below simulates a combat between a dragon and a hydra. The hierarchy of classes that model the creatures of this game are missing and you are asked to provide them.

2.1.1 The class `Creature`

A creature is characterized by:

- its name (`nom_`, a constant string);
- its level (`niveau_`, an integer);
- its number of health points (`points_de_vie_`, an integer);
- its force (`force_`, an integer);

- its position (`position_`, also an integer; for simplicity, our game takes place in 1D).

You must use strictly these attribute names. The attributes must be accessible to classes deriving from `Creature`.

The methods defined for this class are:

- a constructor allowing the initialization of the name, level, health points, force and position of the creature using the values passed as parameters, in this order; the constructor accepts zero as default value for the position;
- a method `bool vivant()` (English: alive) returning `true` if the creature is alive (number of health points greater than zero) or `false` otherwise;
- a method `points_attaque` (English: attack points) returning the number of attack points that can be inflicted by the creature to others; the value is computed as the level multiplied by the force if the creature is alive, or zero otherwise;
- a method `deplacer(int)` (English: move), which does not return anything and adds the integer passed as parameter to the position of the creature;
- a method `adieux()` (English: goodbye), which does not return anything and displays the message (English: <name> is no more!): `<nom> n'est plus!` using strictly this format. `<nom>` is the name of the creature;
- a method `faiblir` (English: weaken), which does not return anything and substracts the number of points passed as parameter from the number of health points of the creature, if it is alive; if the creature dies, its number of health points is set to zero and the method `adieux` is called;
- a method `afficher()` (English: display), which does not return anything and displays informations about the creature using strictly the following format:

```
<nom>, niveau: <niveau>, points de vie: <points>, force: <force>, \
points d'attaque: <attaque>, position: <position>
```

  The character `'\'` is not part of the output and is not followed by a newline. `<nom>` is the name of the creature, `<niveau>` is its level, `<points>` is its number of health points, `<force>` is its force, `<attaque>` is its number of attack points and `<position>` is its position.

2.1.2 The class `Dragon`

A `Dragon` is a `Creature`. It has as specific characteristic the range of its flame (`portee_flamme_` (English: flame range), an integer). Its specific methods are:

- a constructor which initializes its name, level, number of health points, the force, the range of the flame and the position of the dragon using the values passed as parameters, in this order; the constructor accepts zero as default value for the position;
- a method `voler(int pos)` (English: fly) which does not return anything and allows the dragon to move to the given position `pos`;
- a method `souffle_sur(Creature& bete)` (English: blow on a creature) which does not return anything and simulates what happens when the dragon blows its flame towards another `Creature`:

  1. if the dragon and the creature are both alive and if the creature is in range of its flame, the dragon inflicts its attack points as damage to the creature; the creature weakens by the number of attack points; The dragon also weakens; it loses `d` health points, with `d` being the distance between the dragon and the creature (the further the dragon has to blow, the more it weakens);
  2. if after this epic fight the dragon is still alive and the creature dies, the dragon increases in level by one unit;

  The creature is in the range of the flame of the dragon if the distance between them is smaller or equal to the range of the flame (you should use the function `distance` we provide).

2.1.3 The class `Hydre`

A `Hydre` is a `Creature`. It has as specific characteristics the length of its neck (`longueur_cou_`, an integer) and the dose of poison it can inject in an attack (`dose_poison_`, an integer). Its specific methods are:

- a constructor which initializes its name, level, number of health points, force, the length of its neck, the poison dose and the position using the values passed as parameters, in this order; the constructor accepts zero as default value for the position;

- a method `empoisonne(Creature& bete)` (English: poison creature) which does not return anything and simulates what happens when the hydra poisons another `Creature`:

  1. if the hydra and the creature are alive and the creature is in range of the head of the hydra, then the hydra inflicts damage to the creature; the creature weakens by the number of attack points of the hydra plus its dose of poison;
  2. if at the end of the fight the creature is no longer alive, the hydra increases in level by one unit;

  The creature is "in range of the head of the hydra" if the distance the creature and the hydra is smaller or equal to the length of the neck of the hydra (you should use the function `distance` we provide).

The function `combat`(fight) takes as parameters a dragon and a hydra. It allows:

- the hydra to poison the dragon;
- and the dragon to blow on the hydra.

2.2 Execution examples

The example of output below corresponds to the provided program.

```
Dragon rouge, niveau: 2, points de vie: 10, force: 3, points\
 d'attaque: 6, position: 0
se prépare au combat avec :
Hydre maléfique, niveau: 2, points de vie: 10, force: 1, poi\
nts d'attaque: 2, position: 42

1er combat :
    les créatures ne sont pas à portée, donc ne peuvent pas \
s'attaquer.
Après le combat :
Dragon rouge, niveau: 2, points de vie: 10, force: 3, points\
 d'attaque: 6, position: 0
Hydre maléfique, niveau: 2, points de vie: 10, force: 1, poi\
nts d'attaque: 2, position: 42

Le dragon vole à proximité de l'hydre :
Dragon rouge, niveau: 2, points de vie: 10, force: 3, points\
```

```
 d'attaque: 6, position: 41

L'hydre recule d'un pas :
Hydre maléfique, niveau: 2, points de vie: 10, force: 1, poi\
nts d'attaque: 2, position: 43

2e combat :
  + l'hydre inflige au dragon une attaque de 3 points
      [ niveau (2) * force (1) + poison (1) = 3 ] ;
  + le dragon inflige à l'hydre une attaque de 6 points
      [ niveau (2) * force (3) = 6 ] ;
  + pendant son attaque, le dragon perd 2 points de vie supp\
lémentaires
       [ correspondant à la distance entre le dragon et l'hy\
dre : 43 - 41 = 2 ].
Après le combat :
Dragon rouge, niveau: 2, points de vie: 5, force: 3, points \
d'attaque: 6, position: 41
Hydre maléfique, niveau: 2, points de vie: 4, force: 1, poin\
ts d'attaque: 2, position: 43

Le dragon avance d'un pas :
Dragon rouge, niveau: 2, points de vie: 5, force: 3, points \
d'attaque: 6, position: 42

3e combat :
  + l'hydre inflige au dragon une attaque de 3 points
      [ niveau (2) * force (1) + poison (1) = 3 ] ;
  + le dragon inflige à l'hydre une attaque de 6 points
      [ niveau (2) * force (3) = 6 ] ;
  + pendant son attaque, le dragon perd 1 point de vie suppl\
émentaire
       [ correspondant à la distance entre le dragon et l'hy\
dre : 43 - 42 = 1 ] ;
  + l'hydre est vaincue et le dragon monte au niveau 3.
Hydre maléfique n'est plus !
Après le combat :
Dragon rouge, niveau: 3, points de vie: 1, force: 3, points \
d'attaque: 9, position: 42
Hydre maléfique, niveau: 2, points de vie: 0, force: 1, poin\
ts d'attaque: 0, position: 43

4e Combat :
```

```
    quand une créature est vaincue, rien ne se passe.
Après le combat :
Dragon rouge, niveau: 3, points de vie: 1, force: 3, points \
d'attaque: 9, position: 42
Hydre maléfique, niveau: 2, points de vie: 0, force: 1, poin\
ts d'attaque: 0, position: 43
```

The character `'\'` is not part of the output and is not followed by a newline.