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

Quatrième devoir
Héritage

J.-C. Chappelier & J. Sam

1 Exercice 1 — Philatélie

Un philatéliste vous demande d'écrire un programme lui permettant d'estimer le prix auquel il pourrait vendre ses timbres.

1.1 Description

Télécharger le programme `timbres.cc` fourni et le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez en aucun cas modifier ni le début ni la fin du programme fourni, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc impératif de respecter la procédure suivante :

1. sauvegarder le fichier téléchargé sous le nom `timbres.cc` ou `timbres.cpp` ;
2. écrire le code à fournir (voir ci-dessous) entre ces deux commentaires :

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

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

3. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs utilisées dans l'exemple de déroulement donné plus bas ;

4. soumettre le fichier modifié (toujours `timbres.cc` ou `timbres.cpp`) dans « My submission » puis « Create submission ».

Le code fourni crée 4 timbres (de différentes sortes) et affiche leur prix.

Ce code est largement incomplet et il vous est demandé de le compléter suivant les directives suivantes.

Un timbre (classe `Timbre`) est caractérisé par : son *nom* (`nom`, de type `string`), son *année d'émission* (`annee`, de type `unsigned int`), son *pays d'origine* (`pays`, de type `string`) et sa *valeur faciale* en francs (`valeur_faciale`, de type `double`). Veuillez respecter le nom des attributs donnés ici.

Le philatéliste distingue de plus deux grandes catégories de timbres, se distinguant notamment par les modalités du calcul du prix de vente :

- des timbres *rares* (classe `Rare`) : dotés d'un attribut supplémentaire indiquant le nombre d'exemplaires recensés dans le monde ;
- les timbres *commémoratifs* (`Commemoratif`), sans attribut spécifique particulier.

Il n'y a pas de timbre commémoratif rare (ni rare commémoratif `:-P`).

Prix de vente des timbres Le prix de vente d'un *timbre* quelconque (ni rare, ni commémoratif) est sa valeur faciale, si le timbre a moins que cinq ans ; sinon, il vaut la valeur faciale multipliée par 2.5 fois l'âge du timbre (cf exemple de déroulement fourni plus bas).

Le prix de vente d'un timbre *rare* utilise un *prix de base* de 600 francs si le nombre d'exemplaires recensés est (strictement) inférieur à 100, de 400 francs si le nombre d'exemplaires est entre 100 et 1000 (exclu) et 50 francs sinon. Le prix de vente du timbre est alors donné par la formule `prix_base * (age_timbre / 10.0)`. Vous devrez définir les constantes de *classe* suivantes pour la classe `Rare` :

- `PRIX_BASE_TRES_RARE`, de valeur 600 ;
- `PRIX_BASE_RARE`, de valeur 400 ;
- et `PRIX_BASE_PEU_RARE`, de valeur 50.

De son coté, le prix de vente d'un timbre *commémoratif* est le double du prix de vente d'un timbre quelconque.

Timbres quelconques (ni rares, ni commémoratifs) La classe `Timbre` devra avoir au moins les méthodes publiques suivantes :

- des constructeurs conformes à la méthode `main()` fournie, ayant pour paramètres dans l'ordre : le nom, l'année d'émission, le pays d'origine,

et la valeur faciale ; le pays d'origine et la valeur faciale auront comme valeur par défaut respectivement `"Suisse"` et `1.0` ;

— une méthode `vente()` retournant le prix de vente (de type `double`).

— une méthode `age()` retournant l'âge du timbre sous la forme d'un `unsigned int`, différence entre `ANNEE_COURANTE` (un attribut de classe fourni) et l'année d'émission du timbre ;

Il devra par ailleurs être possible d'afficher un timbre au moyen de l'opérateur usuel <<, en respectant ***strictement*** le format suivant :
`Timbre de nom <nom> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs`
sur une seule ligne, où `<nom>` est à remplacer par le nom du timbre, `<annee>` par son année d'émission, `<pays>`, par son pays d'origine et `<valeur faciale>`, par sa valeur faciale (voir les exemples de déroulement fournis plus bas).

Timbres rares et timbre commémoratifs Les classes `Rare` et `Commemoratif` devront avoir les mêmes méthodes publiques que les timbres quelconques et devront aussi pouvoir être affichés au moyen de l'opérateur usuel <<. Les différences sont simplement :

1. que les timbres rares ont un argument supplémentaire dans leur constructeur qui est le nombre d'exemplaires, dont la valeur par défaut est 100 ; ils auront également un accesseur `nb_exemplaires()` pour ce nombre d'exemplaires ;
2. que les timbres rares s'affichent au format suivant :
   `Timbre rare (<exemplaire> ex.) de nom <nom> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs`
   sur une seule ligne, où `<exemplaires>` est à remplacer par le nombre d'exemplaires recensés et les autres informations comme pour les timbres quelconques (voir les exemples de déroulement fournis plus bas) ;
3. que les timbres commémoratifs s'affichent au format suivant :
   `Timbre commémoratif de nom <nom> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs`
   sur une seule ligne.

Notez bien que tous les timbres (quelconques, rares et commémoratifs) ont la même fin d'affichage : tous terminent par :
`de nom <nom> datant de <annee> (provenance <pays>) ayant pour valeur faciale <valeur faciale> francs`

(sur une seule ligne). Nous vous demandons de mettre cette partie commune dans une méthode `afficher`.

Ce qu'il vous est demandé, c'est de coder la hiérarchie de classes découlant de cette description. Vous éviterez toute duplication de code et le masquage d'attributs.

1.2 Exemple de déroulement

```
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
```

Le caractère `'\'` ne fait pas partie de l'affichage et n'est pas suivi d'un saut de ligne.

2 Exercice 2 — Choc des titans

Dans cet exercice, vous allez faire s'affronter des dragons et des hydres.

2.1 Description

Télécharger le programme `dragons.cc` fourni et le compléter suivant les instructions données ci-dessous.

ATTENTION : vous ne devez en aucun cas modifier ni le début ni la fin du programme fourni, juste ajouter vos propres lignes à l'endroit indiqué. Il est donc impératif de respecter la procédure suivante :

1. sauvegarder le fichier téléchargé sous le nom `dragons.cc` ou `dragons.cpp` ;
2. écrire le code à fournir (voir ci-dessous) entre ces deux commentaires :

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

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

3. sauvegarder et tester son programme pour être sûr(e) qu'il fonctionne correctement, par exemple avec les valeurs utilisées dans l'exemple de déroulement donné plus bas ;
4. soumettre le fichier modifié (toujours `dragons.cc` ou `dragons.cpp`) dans « My submission » puis « Create submission ».

Le programme principal fourni met en scène un combat entre un dragon et une hydre. La hiérarchie de classes qui permet de modéliser les créatures de ce jeu manque et il vous est demandé de la compléter.

2.1.1 La classe `Creature`

Une créature est caractérisée par :
- son nom (`nom_`, une chaîne de caractère constante) ;
- son niveau (`niveau_`, un entier) ;
- des points de vie (`points_de_vie_`, un entier) ;
- sa force (`force_`, un entier) ;
- et sa position (`position_`, encore un entier ; nous supposons que notre jeu est en 1D pour simplifier).

Vous respecterez strictement le nommage des attributs. Ces attributs seront accessibles dans les sous-classes de `Creature`.

Les méthodes à prévoir pour cette classe sont :
- un constructeur permettant d'initialiser le nom, le niveau, les points de vie, la force et la position de le créature au moyen de valeurs passées en paramètre, dans cet ordre ; le constructeur acceptera zéro comme valeur par défaut pour la position ;
- une méthode `bool vivant()` retournant `true` si la créature est vivante (nombre de points de vie supérieur à zéro) et `false` sinon ;
- une méthode `points_attaque` retournant les points d'attaque que la créature peut infliger ; il s'agit du niveau multiplié par la force si l'animal est vivant et zéro sinon ;
- une méthode `deplacer(int)`, ne retournant rien et ajoutant en entier passé en paramètre à la position de la créature ;

— une méthode `adieux()` ne retournant rien et affichant le message suivant : `<nom> n'est plus!`
en respectant strictement ce format. `<nom>` est le nom de la créature ;
— une méthode `faiblir`, ne retournant rien et retranchant au nombre de points de vie de la créature, si elle est vivante, un nombre de points passés en paramètre ; si la créature meurt, son nombre de points de vie sera mis à zéro et la méthode `adieux` invoquée ;
— une méthode `afficher()` ne retournant rien, affichant la créature en respectant scrupuleusement le format suivant :

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

Le caractère `'\'` ne fait pas partie de l'affichage et n'est pas suivi d'un saut de ligne. `<nom>` est le nom de la créature, `<niveau>` son niveau, `<points>` son nombre de points de vie, `<force>` sa force, `<attaque>` son nombre de points d'attaque et `<position>` sa position.

2.1.2 La classe `Dragon`

Un `Dragon` est une `Creature`. Il a pour caractéristique spécifique la portée de sa flamme (`portee_flamme_`, un entier). Ses méthodes spécifiques sont :

— un constructeur permettant d'initialiser le nom, le niveau, les points de vie, la force, la portée de la flamme et la position du dragon au moyen de valeurs passées en paramètre, dans cet ordre ; le constructeur acceptera zéro comme valeur par défaut pour la position ;
— une méthode `voler(int pos)` ne retournant rien et permettant au dragon de se déplacer à la position `pos` ;
— une méthode `souffle_sur(Creature& bete)` ne retournant rien et simulant ce qui se passe lorsque le dragon souffle sur une `Creature` :

1. si le dragon est vivant, que la créature l'est aussi et qu'elle est à portée de sa flamme, alors le dragon inflige ses points d'attaque à la créature ; cette dernière faiblit du nombre de points d'attaque ; Le dragon faiblit aussi ; il perd `d` points de vie où `d` est la distance le séparant de la créature (plus il lance sa flamme loin et plus il s'affaiblit) ;
2. si au terme de ce combat épique, le dragon est toujours en vie, il augmente son niveau d'une unité s'il a vaincu la créature (qu'elle n'est plus en vie) ;

La créature est à portée de flamme du dragon si la distance qui les sépare est inférieure ou égale à la portée de la flamme (utilisez la fonction `distance` fournie).

2.1.3 La classe `Hydre`

Une `Hydre` est une `Creature`. Elle a pour caractéristiques spécifiques la longueur de son cou (`longueur_cou_`, un entier) ainsi que la dose de poison qu'elle peut injecter par attaque (`dose_poison_`, un entier). Ses méthodes spécifiques sont :

- un constructeur permettant d'initialiser le nom, le niveau, les points de vie, la force, la longueur du cou, la dose de poison et la position de l'hydre au moyen de valeurs passées en paramètre, dans cet ordre ; le constructeur acceptera zéro comme valeur par défaut pour la position ;
- une méthode `empoisonne(Creature& bete)` ne retournant rien et simulant ce qui se passe lorsque l'hydre empoisonne une `Creature` :
  1. si l'hydre est vivante, que la créature l'est aussi et qu'elle est à portée du cou, alors l'hydre inflige des dommages à la créature ; cette dernière faiblit du nombre de points d'attaque de l'hydre augmentés de la dose de poison ;
  2. si au terme de ce combat, la créature n'est plus en vie, l'hydre augmente son niveau d'une unité ;

  La créature est à « portée de cou » de l'hydre si la distance qui les sépare est inférieure ou égale à la longueur du cou (utilisez la fonction `distance` fournie).

La fonction `combat` prend en paramètre un dragon et une hydre. Elle fait en sorte que :

- l'hydre empoisonne le dragon,
- puis le dragon souffle sur l'hydre.

2.2 Exemples de déroulement

L'exemple de déroulement ci-dessous correspond au programme principal fourni.

```
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
```

Le caractère `'\'` ne fait pas partie de l'affichage et n'est pas suivi d'un saut de ligne.