MOOC Intro POO C++
Exercices semaine 7

Exercice 23 : fichiers et programmes (niveau 1)

> Cet exercice correspond à l'exercice n°71 (pages 183 et 379) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Un fichier informatique contenant un programme (C++ par exemple) est en même temps un document et un programme. On souhaite modéliser informatiquement cette situation.

Dans un fichier `fichierprogramme.cc`, coder :

- une classe `Document` caractérisée par une `taille` (`int`), un `nom` (`string`) et une `extension` (`string`) ;
- une classe `Programme` dotée des attributs : `langage` (`string`), `auteur` (`string`) et `descriptif` (`string`).

Chacune de ces deux classes sera dotée d'une méthode `afficher`, affichant les valeurs de l'ensemble de leurs attributs. Elles devront être également dotées de constructeurs et de destructeurs affichant des messages significatifs.

Coder également une classe `FichierCPP` héritant des deux classes précédentes et munie :

- d'un constructeur permettant d'invoquer les constructeurs des super-classes avec les paramètres nécessaires ; ce constructeur affichera également un message spécifique ;
- d'une méthode `afficher` affichant l'ensemble des caractéristiques d'un objet de ce type ; cette méthode devra faire appel aux méthodes correspondantes des classes parentes.
- d'un destructeur affichant également un message spécifique.

Tester le programme au moyen du `main` de sorte à produire quelque chose similaire à :

```
creation d'un programme
creation d'un document
construction d'un fichier C++
Je suis le fichier numeric.cc
ma taille est de : 1600Kb
Je suis ecrit en: C++
mon auteur est  : C.Hacker
Je fais du : calcul numerique
destruction d'un fichier C++
destruction d'un document
destruction d'un programme
```

Exercice 24 : Échecs (niveau 2)

Cet exercice correspond à l'exercice n°73 (pages 186 et 384) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

On s'intéresse ici au déplacement de quelques-unes des pièces d'un jeu d'échecs. L'idée principale est de représenter le jeu comme une collection de pièces (en général), puis de spécifier chaque type de pièce. Chaque pièce est caractérisée par une position et une couleur.

Commencer par définir ces deux éléments : position et couleur :

- une position est donnée par un couple de coordonnées `(L, C)` comprise chacune entre 1 et 8 ;
- la couleur peut être « blanc » ou « noir ».

En plus des méthodes usuelles (constructeur(s), destructeur, affichage, ...), à définir selon les besoins, une pièce aura une méthode `deplace` permettant de la déplacer vers une position donnée en argument. Cette méthode retournera un booléen « vrai » si le déplacement est valide (et s'est effectué) et « faux » sinon.

Un prototype possible (à adapter si nécessaire) pourrait donc être :

```
bool deplace(const Position&);
```

Porter une attention particulière à la spécification des méthodes de la classe `Piece` (lire peut être la suite !) en essayant de justifier les choix effectués.

Implémenter ensuite différentes pièces d'échecs, par exemple le cheval et la tour.

Le cheval se déplace d'une case dans une direction et de deux cases dans l'autre (en avant ou en arrière). Par exemple si le cheval est en `(4,5)`, il peut aller en `(5,7)`, `(5,3)`, `(3,7)`, `(3,3)`, `(6,6)`, `(2,6)`, `(6,4)` ou `(2,4)`.

La tour peut se déplacer d'autant de cases qu'elle veut, mais toujours dans la même ligne ou dans la même colonne. Par exemple si une tour est en `(4,5)`, elle peut aller en `(4,1)`, `(4,2)`, ..., `(4,8)`, `(1,5)`, `(2,5)`, ..., ou `(8,5)` (mais ne peut pas « se déplacer » en `(4,5)`).

Pour terminer la modélisation, représenter le jeu simplement comme une collection de pièces. Lui ajouter une méthode d'affichage qui donne simplement la liste des pièces qu'il contient (on ne s'intéresse pas ici à l'affichage complet du jeu sous forme de grille).

Terminer le programme en :

1. construisant dans le `main` un jeu contenant :
    - un cheval noir en `(5,6)`,
    - un cheval blanc en `(1,2)`,
    - une tour noire en `(7,6)`,
    - une tour blanche en `(3,2)` ;
2. affichant le jeu ;
3. déplaçant
    - le cavalier blanc en `(6,4)`,
    - le cavalier noir en `(6,4)`,

- la tour blanche en `(7,1)`,
- la tour noire en `(7,1)` ;

4. affichant le jeu.

NOTE : On ne se préoccupe pas dans ce programme de savoir si la case d'arrivée est occupée ou non, ni si la trajectoire est « encombrée ». On ne s'intéresse ici qu'aux déplacements de base.

Exercice 25 : bibliothèque (niveau 2)

> Cet exercice correspond à l'exercice n°74 (pages 188 et 386) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

Le but de cet exercice est d'écrire un programme `biblio.cc` censé aider le personnel d'une bibliothèque.

Le programme doit être orienté objets et *sans duplication de code*. Il est donc demandé d'utiliser au maximum les notions d'encapsulation, abstraction, héritage (éventuellement multiple) et polymorphisme.

On aimerait représenter la bibliothèque comme un tableau pouvant contenir diverses catégories d'ouvrages :

- les journaux périodiques : chaque périodique est décrit par son nom, sa date (mois, année) et le nom de son éditeur ;
- les livres : chaque livre est décrit par son titre, ses auteurs, sa date de parution et son éditeur ;
- les vidéos : chaque vidéo est décrite par son nom, l'année de parution, ses auteurs, sa durée en minute, et son éditeur. \e

NOTE : Les dates peuvent, dans le cadre de cet exercice, être représentées sous forme de `string`.

Un ouvrage (au sens général) devra comprendre les données relatives à son emprunt : un indicateur de la disponibilité de l'ouvrage (booléen), le nom de l'emprunteur et la date (jour, mois, an) de l'emprunt.

De plus, la bibliothèque pourra gérer des « emprunts extérieurs ». Ce sont des ouvrages (livres, périodique ou vidéos) empruntés dans une autre bibliothèque. Un emprunt extérieur, en plus d'être un ouvrage, contient le nom de la bibliothèque d'origine.

Notons que, par définition, un emprunt extérieur est nécessairement emprunté au moment de sa construction.

Indications :

- Bien réfléchir à la hiérarchie de classes à mettre en place. Par exemple, il pourrait être utile de penser à l'héritage multiple pour des livres empruntés dans une bibliothèque extérieure.
- Pour simplifier, il ne sera pas nécessaire de coder tous les cas d'emprunts extérieurs. Le fait de traiter uniquement le cas des *livres* suffira.

Prévoir les opérations suivantes dans le programme (voir l'exemple ci-après) :

- des méthodes permettant d'emprunter, de rendre à nouveau disponible et d'afficher un ouvrage ;
- des constructeurs adéquats ;
- des méthodes permettant d'ajouter un ouvrage, de supprimer un ouvrage (identifié par un numéro que la méthode `ajoute` aura retourné) et d'afficher (ainsi qu'un destructeur pertinent) pour la classe représentant une bibliothèque.

Au niveau du programme principal :

- remplir la bibliothèque avec au moins un ouvrage de chaque catégorie ;
- emprunter un des ouvrages (internes) ;
- supprimer l'(un des) ouvrage(s) externe(s).

On pourrait par exemple écrire le `main` suivant (à *adapter* à sa propre implémentation ; d'autres choix étant bien sûr possibles) :

```
int main()
{
  Bibliotheque mabib;

  mabib.ajoute(new Livre("Programmation orientee objets en C++ "
                         "(2nd edition)", "Micheloud et Rieder",
                         "2003", "PPUR"));
  mabib.ajoute(new Periodique("Computational Linguistics",
                              "12/2003", "MIT Press"));
  unsigned int o1, o2;
  o1 = mabib.ajoute(new Video("Nanometers and Gigabucks",
                              "G. E. Moore", "1996",
                              "University Video Communications", 61));
  o2 = mabib.ajoute(new LivreExterne("Elements of Information Theory",
                                     "Cover and Thomas", "1991",
                                     "Wiley", "25/02/2004",
                                     "ETHZ-BC", "C. Chatnonne"));
  mabib.affiche();
  cout << "================================================" << endl;

  mabib[o1]->emprunt("J.-C. Chappelier", "05/07/2004");
  mabib.supprime(o2);

  mabib.affiche();

  mabib.vider();

  return 0;
}
```

et obtenir la sortie suivante :

```
0- Livre :
        Programmation orientee objets en C++ (2nd edition)
        par Micheloud et Rieder
        publie en 2003 par PPUR
Disponible.
1- Journal :
        Computational Linguistics de 12/2003
        publie par MIT Press
Disponible.
2- Video :
        Nanometers and Gigabucks
        par G. E. Moore
        publiee en 1996 par University Video Communications
        duree : 61 minutes
Disponible.
3- Livre :
        Elements of Information Theory
        par Cover and Thomas
        publie en 1991 par Wiley
Emprunte le 25/02/2004 par C. Chatnonne
provenant de ETHZ-BC
================================================
0- Livre :
```

Programmation orientee objets en C++ (2nd edition)
par Micheloud et Rieder
publie en 2003 par PPUR
Disponible.
1- Journal :
Computational Linguistics de 12/2003
publie par MIT Press
Disponible.
2- Video :
Nanometers and Gigabucks
par G. E. Moore
publiee en 1996 par University Video Communications
duree : 61 minutes
Emprunte le 05/07/2004 par J.-C. Chappelier
