INF1010

Programmation Orientée-Objet

Travail pratique #4
Polymorphisme

Objectifs :	Permettre à l'étudiant de se familiariser le concept de polymorphisme.
Remise du travail :	Jeudi 2 Mars 2017, 8h
Références :	Notes de cours sur Moodle & Chapitre 8 et 19du livre Big C++ 2e éd.
Documents à remettre :	La solution ainsi que les fichiers .cpp et .h complétés réunis sous la forme d'une archive au format .zip.
Directives :	Directives de remise des Travaux pratiques sur Moodle Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires. Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. Veuillez suivre le guide de codage

Travail à réaliser

Le travail présent a pour but d'ajouter de nouvelles fonctionnalités au jeu d'échec pour rendre celui-ci plus complet.

L'ajout de ces fonctionnalités permet d'introduire une notion fondamentale de la programmation orientée objet : *le polymorphisme*. Une classe représente habituellement un concept ou une généralisation, cependant il se peut que deux classes soient deux déclinaisons d'un même concept. Il est alors pertinent de faire intervenir le polymorphisme plutôt que d'implémenter les mêmes méthodes. Cette notion va de pair avec la notion de d'héritage que vous avez vu au cours des prochaines semaines et au cours du TP3.

ATTENTION : Il est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP3.

ATTENTION : Sauf mention explicite du contraire, c'est à vous de déterminer la visibilité de vos attributs (protected, private, public).

ATTENTION : L'un des principes du polymorphisme étant de limiter la duplication du code, pensez à utiliser au maximum les méthodes des classes mères.

Note : Certaines fonctionnalités du TP requièrent de l'aléatoire. Pour cela, veuillez utiliser les fonctions rand et srand. En particulier, soyez très prudent avec l'utilisation de srand, une mauvaise utilisation provoquera un résultat déterministe !

C'est à vous de choisir si vous déclarez ou non les méthodes virtuelles pour chaque classe.

Classe *Piece*

Cette classe est une **classe abstraite**.

Les méthodes suivantes doivent être implémentées/modifiées :

- Une méthode *estMouvementValide(int toX, int toY)* qui identifie si le mouvement vers la position (toX, toY) est valide ou non.
- Une méthode virtuelle pure *deplacer(int toX, int toY)*
- Une méthode *obtenirType()* qui retourne le type de la pièce. **Pensez à utiliser typeid.**
- L'opérateur << qui affiche les informations de la Piece, c'est-à-dire les informations contenu dans ses attributs

Classe *Fou*

Cette classe hérite de Piece et représente une pièce Fou.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut
- Un constructeur par copie prenant en paramètre un Pion et initialisant les attributs correspondants
- Un destructeur
- Une méthode *estMouvementValide(int toX, int toY)* qui identifie si le mouvement vers la position (toX, toY) est valide ou non. On rappelle ici qu'un fou se déplace suivant la diagonale.
- Une méthode *deplacer()* qui modifie la position du Fou si cela est possible
- La méthode afficher qui affiche les informations du Fou en plus du nom de la pièce.

Classe *Reine*

Cette classe hérite de Piece et représente une Reine (La reine peut être considérée dans ses mouvements comme un Fou et une Tour)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut
- Un constructeur par copie prenant en paramètre un Pion et initialisant les attributs correspondants
- Un destructeur
- Une méthode *estMouvementValide(int toX, int toY)* qui identifie si le mouvement vers la position (toX, toY) est valide ou non. On rappelle ici qu'une reine à l'avantage par rapport à un fou de se déplacer aussi suivant les abscisses ou les ordonnées.

- Une méthode *déplacer() qui modifie la position de la reine si cela est possible*
- La méthode afficher qui affiche les informations sur la Reine ainsi que le type de la classe.

Classe *Echiquier*

Cette classe correspond au terrain de jeu. On modifie la conception de cette classe de la manière suivante

Note : Considérez pour ce TP que la relation entre l'échiquier et les différentes pièces est de types compositions, faites donc attention à la gestion de la mémoire.

Les attributs considérés sont :

- Un vecteur de pointeur de Pièce qui va contenir les pièces blanches
- Un vecteur de pointeur de Pièce qui va contenir les pièces noires

Les méthodes suivantes doivent être implémentées/modifiées :

- Une méthode *deplacerPiece(string id, int toX, int toY)* qui prend comme paramètre un id d'une pièce et les coordonnées à atteindre.
- La surcharge de l'operateur **+=** qui prend comme paramètre un pointeur de pièce et l'ajoute suivant sa couleur au bon vecteur. **Aide : Pensez à faire un cast dynamique pour récupérer le bon pointeur**
- La surcharge de l'operateur **-=** qui prend comme paramètre un id et enlève la pièce correspondante à cet id de l'échiquier.
- Une méthode *promouvoir()* qui prend en paramètre une pièce et qui le transforme aléatoirement en une autre pièce si son type est Pion. Le plan de transformation est le suivant :
    - Une Reine si la valeur de la variable aléatoire est 0.
    - Une Tour si la valeur de la variable aléatoire est 1.
    - Un fou si la valeur de la variable aléatoire est 2.

  Note : Pensez à récupérer le type
- Complétez la méthode << qui vous est fourni et qui affiche les différentes informations de l'échiquier.

Main.cpp

Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées.

```
----------AJOUT DES PIECES----------
----------TEST DES DEPLACEMENTS------------
Deplacement de la piece reussie
Deplacement de la piece reussie
Mouvement non valide
Deplacement de la piece reussie
Mouvement non valide
----------TEST DES PROMOTIONS------------
Promotion impossible
Promotion impossible
Promotion reussie
------------INFORMATION PIECES BLANCHES------------
Il y a 13 Pieces blanches
Piece blanc d'id D1 :
Position (3, 0)
Fou, Piece blanc d'id B2 :
Position (1, 1)
Piece blanc d'id C2 :
Position (2, 1)
Piece blanc d'id D2 :
Position (3, 1)
Piece blanc d'id E2 :
Position (4, 1)
Piece blanc d'id F2 :
Position (5, 1)
Piece blanc d'id G2 :
Position (6, 1)
Piece blanc d'id H2 :
Position (7, 1)
Fou, Piece blanc d'id C1 :
Position (2, 0)
Fou, Piece blanc d'id F1 :
Position (5, 0)
Piece blanc d'id A1 :
Position (0, 0)
Piece blanc d'id H1 :
Position (7, 0)
Reine, Piece blanc d'id E1 :
Position (4, 0)
```

```
------------INFORMATION PIECES NOIRES--------------
Il y a 14 Pieces noires
Piece noir d'id A7 :
Position (0, 6)
Piece noir d'id B7 :
Position (1, 6)
Piece noir d'id C7 :
Position (2, 6)
Piece noir d'id D7 :
Position (3, 6)
Piece noir d'id E7 :
Position (4, 6)
Piece noir d'id F7 :
Position (5, 6)
Piece noir d'id G7 :
Position (6, 6)
Piece noir d'id H7 :
Position (4, 6)
Fou, Piece noir d'id C8 :
Position (2, 7)
Fou, Piece noir d'id F8 :
Position (5, 7)
Piece noir d'id A8 :
Position (0, 7)
Piece noir d'id H8 :
Position (6, 7)
Reine, Piece noir d'id E8 :
Position (4, 7)
Piece noir d'id D8 :
Position (3, 7)
```

Questions

1- Pourquoi la classe *Piece* est-elle une classe abstraite ?
2- Quel est l'intérêt d'avoir un destructeur virtuel ?
3- Quelle autre solution est possible pour construire la classe Reine (aide : pensez a l'héritage) ?

Correction

La correction du TP4 se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme;
- (3 points) Exécution du programme;
- (4 points) Comportement exact des méthodes du programme;
- (4 points) Utilisation adéquate du polymorphisme;
- (1 point) Utilisation de l'héritage.
- (1,5 point) Gestion correcte de la mémoire;
- (1 point) Documentation du code;
- (1 point) Utilisation correcte du mot-clé *this* et const;
- (1,5 points) Réponses aux questions.