Université Paris Diderot **Programmation Orientée Objet**
L2, EIDD et M1 linguistiques **Année 2016-2017**

TP n° 4

Jeu d'échec

Le but de ce tp est la réalisation d'une version simplifiée d'un Jeu d'échecs. Les règles complètes sont disponible ici : https ://fr.wikipedia.org/wiki/Échecs. Vous pouvez proposer votre propre modélisation, potentiellement différente de celle proposée ci-dessous.

Exercice 1 Modélisation du Plateau

1. Écrire une classe Case, qui a comme attribut un booléen indiquant si elle est blanche ou noir, et des coordonnées (destinées à indiquer sa position sur le plateau).
2. Écrire une méthode toString() dans Case, qui (pour l'instant) affiche si c'est une case noire ou blanche.
3. Écrire une classe Plateau, qui a comme attribut :
   — des entiers static final correspondant à la longueur et la largeur du plateau (fixés à 8).
   — un tableau de case
4. Ajouter à la classe Plateau :
   — Un constructeur

   ```
   1 | public Plateau()
   ```

   ,qui génère un plateau de 8×8 cases, de couleur alternées.
   — Une méthode :

   ```
   1 | public void afficher(),
   ```

   qui affiche le plateau.
   — une méthode :

   ```
   1 | public Case giveCase(int x,int y)
   ```

   , qui renvoie la Case de coordonnées x et y.
5. Créer une classe Test, où vous créerez un plateau que vous affichez.

On veut maintenant ajouter la possibilité de mettre des pièces sur les cases, et de déplacer les pièces d'une case sur l'autre. On veut séparer la partie du code qui est générique (c'est à dire, ne dépend pas de quel type de pièce on considère), de celle qui doit vraiment prendre en compte les spécificités de la pièce (par exemple, un pion ne peut avancer que dans une direction...). Pour cela, on écrit d'abord une classe Piece, qui représente une pièce pouvant évoluer sur le plateau sans contraintes, et on fera ensuite, pour chaque type de pièces, une classe spécifique qui hérite de Piece.

Exercice 2 Pièces sans Contraintes

1. Écrire une classe Piece, qui a comme attribut :
   — un booléen qui correspond à la couleur de la pièce
   — Une case qui donne la case sur laquelle est la pièce
2. Ajouter un attribut Piece à la classe Case (qui vaut null si la case est vide), ainsi que les méthodes suivantes :
   — public boolean estVide() ;
   — public Piece getPiece() ;
   — public void removePiece() ;
   — public void fill(Piece p) ;
3. Écrire les méthodes appropriés, et écrire un programme de test qui génère un échiquier, met des pièces dans des cases, et imprime le résultat.

Un déplacement correspond à la donnée de la case de départ et la case d'arrivée : l'idée est qu'on déplace la pièce de la case de départ à la case d'arrivée, à condition que :

— Il y a effectivement une pièce p sur la case de départ
— Cette pièce appartient au joueur en train de jouer
— Il n'y a pas de pièce appartenant à ce même joueur sur la case d'arrivée (et si il y a une pièce appartenant à l'autre joueur, il faut la supprimer).
— Le mouvement est valide pour la pièce p.

Exercice 3 Déplacement

On ajoute à la classe Piece une méthode :

```
1 public boolean isValid(Plateau p, Case arrivee){
2         return true;
3     }
```

Cette méthode est destinée à être redéfinie dans chacune des classes héritées de Piece. Pour l'instant, elle correspond à une pièce dont tous les déplacement sont valides. Elle devra renvoyer true si le déplacement de la pièce considérée (this) jusqu'à la case arrivée est valide, et false sinon.

1. Écrire une classe Mouvement, qui a comme attribut une Case de départ est une Case d'arrivée, et comme méthode :
   — public boolean isValid(Plateau p, boolean joueur)
   — public void faire(Plateau p)
   La méthode isValid renvoie true si toutes les conditions pour effectuer le déplacement sont réunies. Elle utilise la méthode isValide de la classe Piece. La méthode faire réalise le déplacement : il faut vider la case de départ, vider la case d'arrivée, et rattacher la pièce à la case d'arrivée.
2. Ajouter à la classe Plateau une méthode :

   ```
   1 public boolean deplace(Mouvement m, boolean joueur)
   ```

   , qui applique le mouvement m au plateau (si il est valide), en utilisant les méthode de Mouvement.
3. Dans le programme de test, réaliser le déplacement d'une pièce d'une case dans une autre, et tester les conditions limites (si la case de départ est vide...).

Exercice 4 Classes héritées de Piece

1. Écrire des classes : Pion, Tour, Fou, Cavalier, Roi, Dame, qui hérite de Piece
2. Écrire les fonctions d'affichage correspondantes
3. Ajouter à la classe Plateau une méthode :

```
1 | public void initialisation()
```

   , qui initialise le plateau avec la configuration initiale du Jeu d'échecs.
4. Dans le programme de test, créer un plateau, initialisez le, et afficher le résultat.
5. Pour avoir un jeu qui correspond aux règle des échecs, il faut redéfinir la méthode isValide pour chacune de ces classes. On peut néanmoins reporter cette redéfinition (ou le faire seulement dans le cas des pions, par exemple) et commencer par gérer l'intéraction avec l'utilisateur.

On reporte à la fin la définition des classes héritées de la classe Piece. On se concentre d'abord sur l'intéraction avec l'utilisateur.

Exercice 5 Lancement du jeu et interface

1. Creer une classe Communication, qui a pour attribut un Scanner, et comme méthode :

```
1 | public Mouvement demanderMouvement(Plateau pl)
```

   , qui demande à l'utilisateur quelle case de départ et quelle case d'arrivée il choisit.
2. Créer une classe Jeu, qui a comme attribut un objet de la classe Communication, et un plateau.
3. Ajouter un constructeur à Jeu, qui crée le plateau et l'initialise avec la méthode initialisation().
4. Écrire une méthode

```
1 | public void jouer(int n)
```

   , qui :
   — demande au joueur 1 quel mouvement il veut jouer jusqu'à obtenir un mouvement valide
   — l'applique au plateau
   — et itère le processus avec le joueur 2, jusqu'à ce que n tours de jeu aient été effectuées

Exercice 6 Fin du jeu

1. Ajoutez une modélisation de la situation où le roi d'un des deux joueurs est mis en échec (par exemple, une méthode dans la classe Plateau :

```
1 | public boolean isInEchec(boolean joueur))
```

2. Un joueur perd la partie si il est en situation d'échec deux tours de jeu consécutifs. Modélisez cette condition de défaite.
3. La partie est ex aequo si il existe un moment où plus aucun mouvement n'est possible.