UFR ST - Besançon- L2 Info - POA

TP 2 - Héritage et classes abstraites : implantation du jeu d'échec

Ce sujet est prévu pour 3H de TP (2 séances de 1,5h). Le but est de mettre en œuvre le concept de polymorphisme, en créant un jeu d'échec (à deux joueurs) selon la modélisation préparée en TD.

1 Principe d'implantation

1.1 L'échiquier

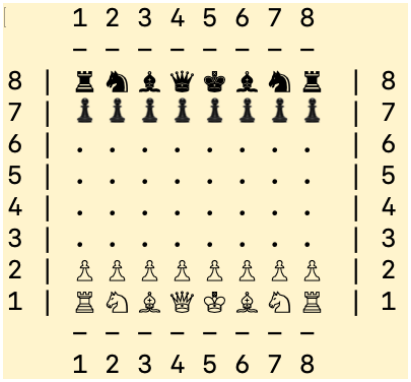
L'échiquier est une matrice de pièces. Selon votre choix de modélisation, une case inoccupée peut soit contenir la référence null, soit une « pièce vide », qui serait une instance concrète de la classe abstraite Piece créée spécialement pour représenter une case vide. L'avantage d'une pièce vide concrète est que ça garantit que les appels de méthodes seront définis sur toutes les cases du tableau. L'avantage d'une case « null » est conceptuel : pas de pièce, donc pas de référence à une pièce. De plus, tester qu'une case est vide revient à tester si elle référence la valeur null. Mais il faudra vérifier si une case est « null » ou pas avant de tenter d'invoquer une méthode sur celle-ci.

1.2 Affichage

L'échiquier sera affiché en mode texte sur la console à partir de sa méthode toString(). Pour cela, l'échiquier interroge (éventuellement pour ses cases non « null » selon votre modélisation) la méthode toString() des pièces contenues. Il vous faut donc redéfinir la méthode toString() de chaque pièce, de façon à ce qu'elle retourne une chaîne d'un caractère représentant la pièce. Si votre console supporte l'affichage des caractères unicode, vous pouvez utiliser l'encodage unicode des pièces d'échec :

- Roi (King) : "♚" (caractère '\u265A') / "♔" (caractère '\u2654')
- Reine (Queen) : "♛" (caractère '\u265B') / "♕" (caractère '\u2655')
- Tour (Rook) : "♜" (caractère '\u265C') / "♖" (caractère '\u2656')
- Fou (Bishop) : "♝" (caractère '\u265D') / "♗" (caractère '\u2657')
- Cavalier (Knight) : "♞" (caractère '\u265E') / "♘" (caractère '\u2658')
- Pion (Pawn) : "♟" (caractère '\u265F') / "♙" (caractère '\u2659')

L'échiquier (dans son état initial) pourra par exemple ressembler à ceci :



1.3 Prises de pièces

En plus de vérifier la validité des mouvements, votre jeu devra gérer les prises de pièces : une pièce A arrivant sur une case déjà occupée par une pièce B la « prend » : la pièce B est retirée du jeu.

1.4 Programme principal

On crée d'abord les pièces, puis l'échiquier en y déposant les pièces.

Le principe est ensuite de réaliser les actions suivantes alternativement pour chaque joueur :
— demander les coordonnées d'une pièce de son jeu,
— demander les coordonnées d'une case cible pour cette pièce,
— réaliser le mouvement s'il est OK, sinon le refuser,
— gérer l'éventuelle prise de pièce.

2 Réalisations

Le développement complet est long et prendra vraisemblablement plus de temps que les trois heures prévues en séance. Il est conseillé de développer d'abord une version minimale, qui permettra tout de même d'utiliser le polymorphisme des pièces. Vous développerez la version complète en fonction du temps restant à votre disposition et du temps de travail personnel.

2.1 Version minimale

Il s'agit d'*afficher* un échiquier en utilisant le polymorphisme de la méthode `toString()`, et de réaliser le déplacement de quelques pièces, sans tenir compte des autres pièces pouvant faire obstacle au déplacement. Le mouvement du pion, qui est le plus compliqué, pourra être simplifié. On peut se contenter par exemple de le faire avancer d'une case verticalement dans la version minimale.

2.2 Version complète

Dans la version complète, vous devrez d'abord vérifier lors du déplacement d'une pièce que d'autres pièces ne font pas obstacle à ce déplacement. En effet, seul le cavalier est autorisé à sauter par dessus les autres pièces. De plus, vous vous attacherez à implanter les mouvements le plus complètement possible : pions, roque éventuel.