INFORMATIQUE — Mias 2 — TP 6, 7 et 8

Un jeu d'échecs

Récupération des fichiers

Les fichiers utilisés dans ce TP sont disponibles dans le répertoire `Info Mias-SM\Java\TP6-8` sur `jupiter`. Ce répertoire contient les fichiers suivants :

Nom du fichier	Contenu du fichier
`InterfaceEchiquier.java`	la classe qui définit l'interface du jeu d'échecs
`Echiquier.java`	la classe qui définit le modèle de l'échiquier
`Piece.java`	la classe qui modélise une pièce du jeu
`Tour.java`	la classe qui définit une tour (blanche ou noire)

Pour finir, ce répertoire contient aussi un sous-répertoire `images` dans lequel sont regroupés les images représentant les différentes pièces.

Avant de commencer à travailler, il vous faut copier les fichiers `.java` dans un répertoire qui vous est propre. Pour cela, effectuez les opérations suivantes :

1. créez un nouveau répertoire `TP6-8` dans le répertoire `Java` de votre répertoire personnel (`Z:`);
2. copiez les six fichiers `.java` de `jupiter` dans votre répertoire `TP6-8` (utilisez les raccourcis `Ctrl-C` et `Ctrl-V`, ou cliquez sur le bouton droit et sélectionnez les libellés `copier` et `coller`);

Les fichiers images n'ont pas besoin, quant à eux, d'être copiés. En effet, vous n'allez pas les modifier. De plus, si chacun d'entre vous les copie, vous utiliserez (en totalité) plus de 600 kO sur `jupiter`, qu'il est possible d'économiser. Pour cela, il suffit de créer dans votre répertoire `TP6-8` un *lien* (appelé *raccourci* sous Windows) vers le répertoire `images` de `jupiter`. Ce lien peut être créé par le menu contextuel (obtenu d'un clic sur le bouton droit de la souris) de votre répertoire `TP6-8`.

Cela étant fait, vous pouvez lancer `Emacs`.

Objectifs de l'application

Dans ce TP, nous vous proposons de réaliser un jeu d'échecs sur ordinateur. Ce jeu représentera un échiquier sur lequel l'utilisateur pourra déplacer les pièces affichées, l'ordinateur vérifiant que le déplacement respecte les règles du jeu.

La partie graphique est fournie dans les fichiers indiqués ci-dessus, vous devez réaliser l'implantation des règles vérifiant le déplacement de chacune des pièces.

Exercice 1 : Définition des pièces

Dans notre application, chaque pièce est un objet défini par sa couleur et l'image le représentant. Le type d'une pièce (pion, tour, cavalier, fou, reine ou roi) est quant à lui donné par la classe

.../...

d'instaciation de l'objet associé : en effet, la classe `Piece` est une classe abstraite (aucun objet de ce type ne peut être créé), dont la méthode de profil `boolean deplacementValide(int ld, char cd, int la, char ca)` doit être définie.

Cette méthode indique si la pièce peut se déplacer depuis la position de départ `(ld, cd)` (à l'intersection de la ligne `ld` et de la colonne `cd`) jusqu'à la position `(la, ca)`, en respectant les règles de déplacement associées à ce type de pièce. La validité des positions (leur appartenance au plateau de l'échiquier) ainsi que la la présence d'autres pièces sur le chemin associé au déplacement sont gérés par l'échiquier (dans la classe `Echiquier`).

Définissez les différentes pièces du jeu d'échec, de la même façon que cela a été fait pour la tour : pour chacune, il suffit de définir le constructeur (en utilisant celui de la classe `Piece`) ainsi que la méthode `deplacementValide` de façon à vérifier le respect des règles de déplacement associées à ce type de pièce. On pourra ignorer le cas du Roque, qui est un peu compliqué.

Remarque : le résultat de la méthode `deplacementValide` ne dépend en fait que du type de la pièce, donc de la classe d'instaciation où cette méthode est définie. Cette méthode devrait par conséquent être définie comme statique. Cependant, cela n'est pas possible pour des raisons techniques : une méthode surchargeant une méthode abstraite ne peut pas être statique (incompatibilité de gestion au niveau du compilateur).

On peut par contre définir une méthode statique et appeler celle-ci dans la méthode `deplacementValide`. C'est ce qui a été fait dans la classe `Tour`. Il faut cependant utiliser un autre nom de méthode, pour différencier celle-ci de la méthode `deplacementValide` : les deux ont les mêmes paramètres (même nombre et types) et le même type de retour.

On peut alors utiliser la méthode statique sans créer d'objet de ce type. Ainsi, un déplacement pour une reine pourra validé s'il correspond à un déplacement valide pour une tour ou un fou, ce qui s'écrit :
`Tour.deplacementAuthorise(ligneDepart, colonneDepart, ligneArrivee, colonneArrivee) || Fou.deplacementAuthorise(ligneDepart, colonneDepart, ligneArrivee, colonneArrivee)`.

Exercice 2 : Prise en compte des autres pièces

Lorsque l'échiquier contient plusieurs pièces, un déplacement n'est valable que si :

- aucune pièce ne se trouve sur le chemin associé au déplacement (excepté pour les cavaliers qui peuvent sauter au-dessus d'autres pièces), et
- la case d'arrivée ne contient pas une pièce de la même couleur que la pièce se déplaçant ; si cette case contient une pièce d'une autre couleur, cette pièce est alors prise.

Quelle méthode faut-il modifier dans la classe `Echiquier` pour vérifier le premier point? Cette méthode peut être définie par une boucle, ou en calculant le chemin parcouru sous la forme d'une liste de pièces, qui sera ensuite parcourue.

Exercice 3 : Finir le jeu d'échecs

Pour terminer une première version du jeu d'échecs, il faut placer les pièces au début du jeu, puis vérifier que les déplacements correspondent alternativement au camp blanc ou noir (les blancs jouant les premiers).

Que faut-il rajouter à la classe `Echiquier` pour vérifier ce dernier point?

Exercice 4 : Une assistance au jeu

On souhaite maintenant montrer, lorsqu'une pièce est sélectionnée pour se déplacer, toutes les positions où elle peut aller.

Quelles sont les différentes façons dont on pourrait définir une fonction `positionsValides` donnant la liste de ces positions? Quelle est la façon demandant le moins de calculs? Ajoutez la fonction `positionsValides` pour chacune des pièces.

Comment cette fonction peut-elle être utilisée dans la classe `InterfaceEchiquier` pour obtenir le comportement décrit plus haut? Effectuez les modifications nécessaires, et testez le résultat.