Développement orientée objet - Java
TP 06 - Partie 1

Lionnel Conoir, Isabelle Delignières, Wajdi Elleuch
Rémi Synave et Franck Vandewiele

Ce TP va mettre en place la base du jeu d'échec, les classes principales et la première interface graphique. Il est donc relativement long et se fera donc sur 4 créneaux de TP d'1h30. Si vous avez correctement suivi le cours, vous constaterez qu'il n'y a pas trop besoin de réfléchir mais simplement d'appliquer la théorie.

Analyse et mise en place de la base du jeu d'échecs

Le jeu d'échecs se joue à deux joueurs. Un joueur joue avec les pièces blanches, l'autre avec les pièces noires. Chaque joueur possède, en début de partie, les pièces suivantes : 8 pions, 2 tours, 2 cavaliers, 2 fous, une ~~reine~~ dame et un roi. La position initiale des pièces est montrée en figure 1. Le plateau utilisé fait 8 cases de large sur 8 cases de haut. La case en bas à gauche est noire et les couleurs s'alternent sur les cases voisines. L'application que vous allez développer se base sur cette configuration et aucune autre.

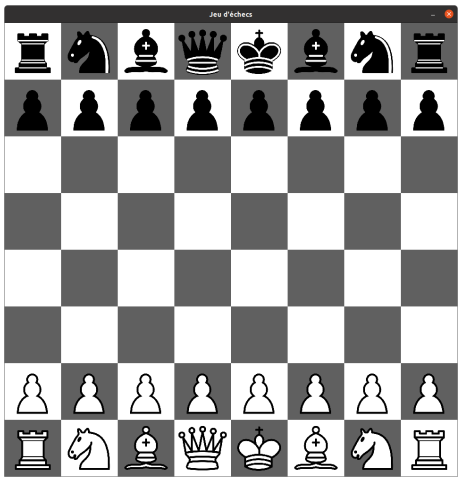

FIGURE 1 – Placement initial des pièces sur le palteau.

Ce jeu se joue en tout à tour et les pièces ont toutes des règles particulières pour bouger.

Ce jeu pourrait être modélisé par une grille à 2 dimensions dont les cases contiendraient ou non des pièces. Toutefois, dans notre cas, nous modéliserons ce jeu par un tableau de taille dynamique de pièces. Chaque pièce sera identifié par son type, sa couleur et sa position. Le plateau initial sera donc modélisé par :

```
palteau =
{
["tour", "blanc", 0, 0], ["cavalier", "blanc", 1, 0], ["fou", "blanc", 2, 0],
["dame", "blanc", 3, 0], ["roi", "blanc", 4, 0], ["fou", "blanc", 5, 0],
["cavalier", "blanc", 6, 0], ["tour", "blanc", 7, 0], ["pion", "blanc", 0, 1],
["pion", "blanc", 1, 1], ["pion", "blanc", 2, 1], ["pion", "blanc", 3, 1],
["pion", "blanc", 4, 1], ["pion", "blanc", 5, 1], ["pion", "blanc", 6, 1],
["pion", "blanc", 7, 1], ["pion", "noir", 0, 6], ["pion", "noir", 1, 6],
["pion", "noir", 2, 6], ["pion", "noir", 3, 6], ["pion", "noir", 4, 6],
["pion", "noir", 5, 6], ["pion", "noir", 6, 6], ["pion", "noir", 7, 6],
["tour", "noir", 0, 7], ["cavalier", "noir", 1, 7], ["fou", "noir", 2, 7],
["dame", "noir", 3, 7], ["roi", "noir", 4, 7], ["fou", "noir", 5, 7],
["cavalier", "noir", 6, 7], ["tour", "noir", 7, 7]
}
```

ou tout autre tableau ayant le même contenu mais dans un ordre différent.

Les positions données dans l'exemple précédent sont basées sur des coordonnées classiques. Toutefois, les cases d'un échiquier sont normalement identifée par une lettre suivie d'un nombre comme montré ci-dessous.

```
 |---|---|---|---|---|---|---|---|
8|ToN|CaN|FoN|DaN|RoN|FoN|CaN|ToN|8
 |---|---|---|---|---|---|---|---|
7|PiN|PiN|PiN|PiN|PiN|PiN|PiN|PiN|7
 |---|---|---|---|---|---|---|---|
6|   |   |   |   |   |   |   |   |6
 |---|---|---|---|---|---|---|---|
5|   |   |   |   |   |   |   |   |5
 |---|---|---|---|---|---|---|---|
4|   |   |   |   |   |   |   |   |4
 |---|---|---|---|---|---|---|---|
3|   |   |   |   |   |   |   |   |3
 |---|---|---|---|---|---|---|---|
2|PiB|PiB|PiB|PiB|PiB|PiB|PiB|PiB|2
 |---|---|---|---|---|---|---|---|
1|ToB|CaB|FoB|DaB|RoB|FoB|CaB|ToB|1
 |---|---|---|---|---|---|---|---|
   A   B   C   D   E   F   G   H
```

Quelques équivalences :

- Case (0,0) → Case A1
- Case (0,3) → Case A4
- Case (4,5) → Case E6
- Case (7,2) → Case H3

Objectifs

L'objectif de cette première partie est de mettre en place les premières classes et méthodes permettant de modéliser un plateau de jeu dans sa position initiale. Dans les parties suivantes, ces classes seront modifiées ou complétées. Il est alors demandé de bien suivre l'ordre des questions.

Un première classe à développer sera une classe permettant de stocker la position des pièces et d'identifier les cases comme sur un véritable échiquier. Il faudra que l'on puisse décrire une position par ses coordonnées mais aussi par la notation des échecs

Ensuite, vous développerez la classe permettant de décrire une pièce. Toutes les méthodes qui pourront vous être utile par la suite seront à développer (plusieurs constructeurs, accès au type, au nom court, à la position, etc.).

Puis vous développerez la classe modélisant un échiquier.

Finalement, vous écrirez un porgramme principal permettant l'affichage en mode console puis en mode graphique du plateau initial.

dans l'énoncé ci-dessous, lorsque rien n'est précisé, c'est que la méthode ou l'attribut doit être déclaré `public`.

La classe `Position`

Cette classe se compose simplement de deux attributs `x` et `y` permettant de stocker les valeurs de composantes. Cette classe est spécifique à ce jeu des échecs. Ainsi les valeurs des composantes ne peuvent être comprises qu'entre 0 et 7. Si l'utilisateur essaie de mettre une valeur incohérente, un message d'erreur devra s'afficher et le programme devra s'arrêter.

Développez la classe `Position`. Elle doit comporter :

— Deux attributs privés entiers `x` et `y` permettant de repérer une position particulière. Attention, il ne doit pas être possible d'y mettre autre chose que des chiffres de 0 à 7.
— Un constructeur par défaut qui intialisera les attributs à 0.
— Un constructeur par copie.
— Un constructeur prenant deux entiers en paramètres.
— Un constructeur prenant une chaine de caractère en paramètre. La chaine de caractère en paramètre sera de la forme : lettre en majuscule suivie d'un chiffre (identification d'une case comme sur un échiquier).

— Les getter et setter.
— La méthode `equals` prenant un `Object` en paramètre.
— La méthode `toString` retournant la position sous la forme de du repérage échiquier (type "E4").

N'hésitez pas à tester votre classe en écrivant un programme principal dans cette classe.

La classe `Piece`

Cette classe permet de définir une pièce. Sur le jeu d'échec, les pièces ne peuvent être que : tour, cavalier, foi, dame, roi ou pion. Chaque pièce a également une couleur et une position.

Développez la classe `Piece`. Elle doit comporter :

— Trois attributs privés :
  — `type`, une chaine de caractère prenant comme valeur l'un des types définis juste avant. Attention ! Il n'y a pas de majuscule dans les noms.
  — `couleur`, un caractère qui ne doit pouvoir prendre que deux valeurs : 'B' (pour blanc) ou 'N' (pour noir).
  — `position`, un objet de type `Position`.
— Un constructeur par défaut qui créera un pion blanc en A2.
— Un constructeur par copie.
— Un constructeur prenant en paramètre le type de la pièce, une couleur et une position sous la forme de deux entiers.
— Un constructeur prenant en paramètre le type de la pièce, une couleur et une position sous la forme d'un objet de type `Position`.
— Un constructeur prenant en paramètre le type de la pièce, une couleur et une position sous la forme d'une chaine de caractère identifiant une case à la manière d'un échiquier (type "A4" ou "E3").
— Les getter et setter.
— Une méthode `getNomCourt` qui retourne une chaine de trois caractères repreant les deux premières lettres du type (avec une majuscule en première lettre) suivi du caractère indiquant la couleur de la pièce. (Ex : "RoN" pour "roi Noir" ou "PiB" pour "pion blanc").
— Une méthode `getNomLong` qui retourne une chaine de caractère contenant le type suivi d'un "_" et du caractère donnant la couleur. (Ex : "roi_N" pour "roi noir" ou "pion_B" pour "pion blanc").
— La méthode `equals` prenant un `Object` en paramètre.
— La méthode `toString` retournant une description de la pièce du type : "roi noir en E4".

La classe `Plateau`

Le plateau sera modélisé par un tableau dynamique contenant les pièces présentes sur l'échiquier.

Cette classe `Plateau` doit comporter :

— Un attribut privé de type tableau dynamique stockant des objets de type `Piece`.

— Un constructeur par défaut initialisant le plateau tel que montré en figure 1.

— Trois méthodes `getCase`. L'une prend en paramètre deux entiers, la seconde un objet de type `Position` et une dernière prenant une chaine de caractère indiquant une positoin à la manière de l'échiquier. Cette méthode retournera un objet de type `Piece`. Si une pièce se trouve à la position donnée, cette pièce sera retournée. Si aucune pièce ne se trouve sur la case, la méthode retournera la valeur `null` indiquant ainsi que la case est vide.

— La méthode `toString` retournant le palteau tel que présenté en bas de la page 2.

— Une méthode `getPiecesBlanches` retournant un tableau dynamique rempli des pièces blanches encore présentes sur le plateau.

— Une méthode `getPiecesNoires` retournant un tableau dynamique rempli des pièces noires encore présentes sur le plateau.

La classe `Main`

Créez un programme principal dans une classe `Main`. Le programme créera un objet de type `Plateau` qu'il affichera dans le terminal. Le rendu doit être proche de celui montré en bas de page 2.

Le programme affichera ensuite la liste des pièces blanches puis la liste des pièces noires.

La classe `MainGraphique`

En utilisant la bibliothèque MG2D, créez une interface graphique permettant d'afficher un plateau tel que montré en figure 1.

Question subsidiaire

1. Combien de fois le terme "palteau" a t-il été écrit dans ce sujet à la place de "plateau" ?
2. Avez vous mis votre palto aujourd'hui ?