Identification d'une main de Poker

But du laboratoire

Ce laboratoire vous permettra :

- de comprendre l'utilité des patrons GoF « itérateur » et « chaîne de responsabilités » et de savoir comment les utiliser,
- d'apprendre à utiliser l'interface Comparable de l'API Java,
- de comprendre l'utilité des tests unitaires et de savoir les utiliser avec JUnit.

Description du laboratoire

Dans la conception du logiciel, parfois on cherche à favoriser son extensibilité. C'est à dire que le problème à résoudre pour l'instant peut changer et nous cherchons une conception permettant des changements faciles. Ce laboratoire traite le problème d'un logiciel pour identifier une main de Poker. Ce logiciel serait utilisé éventuellement par plusieurs applications de jeu de Poker, par exemple, un vidéopoker, un jeu de Poker réparti, etc.

Le jeu de Poker a plusieurs variantes, par exemple, Draw (à cinq cartes), Stud (à cinq ou à sept cartes), Hold 'em (Texas), etc. Puis, il y a des variantes permettant les cartes spéciales telles que le Joker ou des frimes. Malgré toutes ces variantes, les rangs de main sont plus ou moins normalisés.

On veut donc concevoir un ensemble de classes logicielles fournissant « un service d'identification de main de Poker » en dépit des variantes concernant le nombre de cartes dans une main, la présence des frimes, etc.

Deux patrons vous seront utiles pour réaliser cette tâche : le patron GoF « itérateur » pour manipuler les mains de Poker (sans se soucier du nombre de cartes dans la main) et le patron GoF « chaîne de responsabilités » pour découpler le code responsable de l'identification d'une main du reste du système.

Vous utiliserez aussi l'outil JUnit pour faire des tests unitaires des classes que vous allez concevoir dans ce laboratoire. Il faut faire ce tutoriel sur JUnit avant de continuer avec ce laboratoire.

Travail à effectuer

Il n'y a pas d'application à réaliser. Votre travail consiste à développer les classes pour reconnaître les mains ainsi que les tests unitaires pour démontrer que ces classes fonctionnent.

- Concevoir des classes logicielles capables d'identifier des rangs de mains de Poker et de les comparer, selon les règles indiquées dans cet énoncé.
- Concevoir des classes jumelles de test unitaire pour les classes importantes dans ce laboratoire. Pour les stratégies de test unitaire, ce chapitre d'un livre sur JUnit peut vous être utile.

Vous devez implémenter toutes les classes pour reconnaître les onze (11) rangs spécifiés dans cet énoncé. Vous devriez en choisir un sous ensemble à développer au début, d'une façon itérative et incrémentale.

Lexique

Couleur - C'est la sorte d'une carte. Les quatre couleurs sont : cœur, pique, carreau, trèfle. Au Poker, le terme "couleur" ne représente jamais le noir ou le rouge.

Dénomination - C'est la valeur d'une carte, parmi 13 valeurs possibles, p. ex.: deux, trois, ..., dix, valet, dame, roi, as.

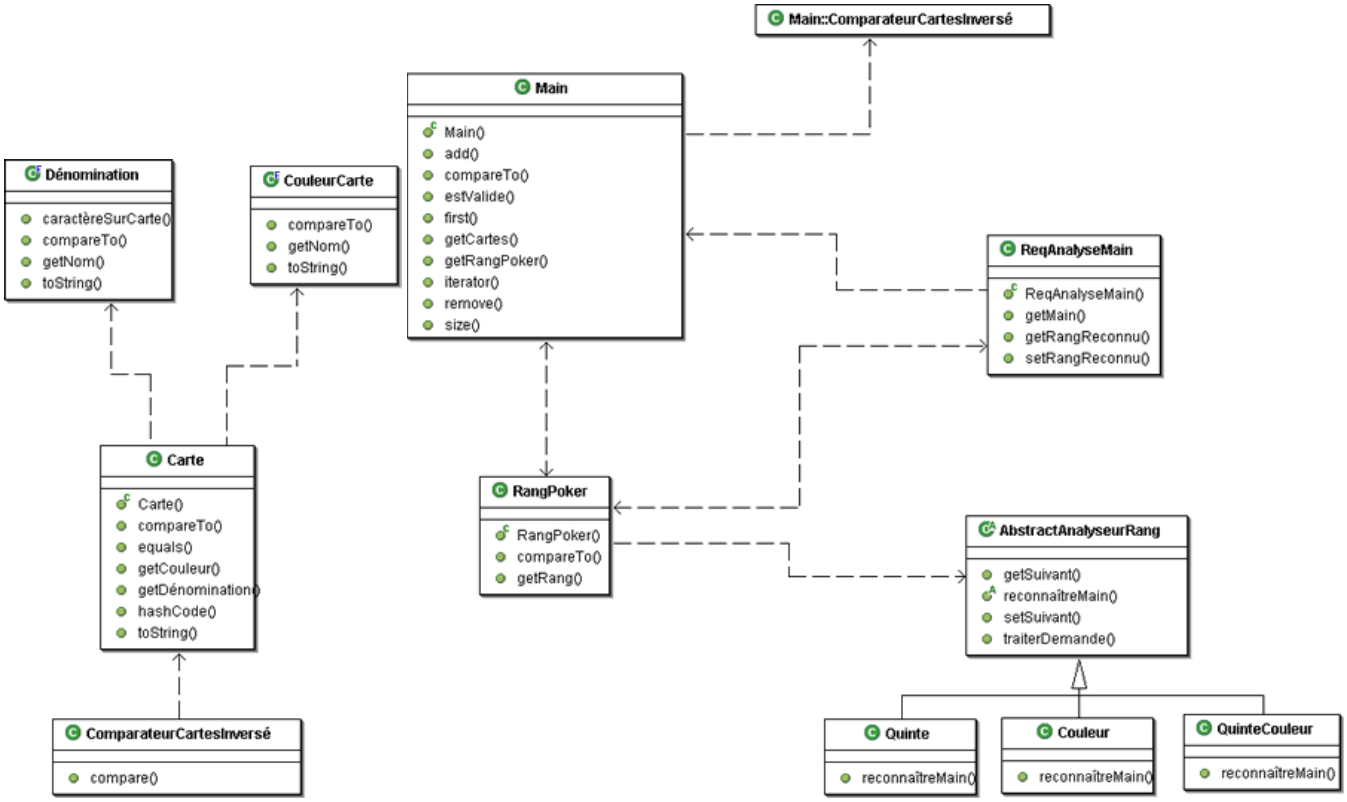
Frime - Une carte qui peut prendre n'importe quelles dénomination ou couleur.

Contraintes de conception

La conception doit être orientée objet avec des classes cohésives. La classe Carte devrait modéliser correctement une carte de Poker. La classe Main devrait représenter correctement une main de Poker, sans être trop basée sur une variante de Poker. Il doit y avoir une classe RangPoker qui modélise la notion de rang comme on l'explique dans cet énoncé. Les classes CouleurCarte, Dénomination, Carte et RangPoker doivent implémenter l'interface Comparable de l'API Java.

Vous devriez avoir deux paquetages : mains et cartes.

Voir les diagrammes ci-dessous pour plus de détails sur la conception :



Utilisation obligatoire du patron GoF « chaîne de responsabilités » : Dans ce laboratoire, la chaîne sera composée d'objets, chacun étant capable d'identifier un rang de main différent, par exemple, Quinte, Couleur, QuinteCouleur, etc. Ce patron nécessite la définition d'une requête (ReqAnalyseMain dans l'UML) qui est passée entre les objets dans la chaîne (instances des sous-classes de AbstractAnalyseurRang). Vous devez définir cette classe de requête ainsi que les classes d'objet pour reconnaître une main. Les classes reconnaissantes ne devraient pas avoir de couplage direct avec la classe Main. Idéalement, c'est la classe RangPoker qui, dans son constructeur, va déléguer le travail pour identifier la main en question.

Utilisation obligatoire d'un « itérateur » de l'API Java (Collection) : Une main de Poker (la classe Main) peut être facilement réalisée, si on utilise une classe concrète de l'interface « Collection » de l'API Java pour stocker les cartes dans la main. Ainsi, vous serez obligé d'utiliser des itérateurs. Il y a d'autres endroits où vous trouverez pratique d'utiliser des « Collection ».

Utilisation obligatoire de l'outil JUnit : Vous devez écrire des classes jumelles de test unitaire pour au moins les classes suivantes : Main, RangPoker ainsi que les classes qui reconnaissent les rangs. On vous donne déjà une bonne classe de test unitaire pour la classe Carte : CarteTest.java. Nous vous proposons également CouleurCarteTest.java, DénominationTest.java, CouleurTest.java, QuinteTest.java et QuiteCouleurTest.java à titre d'exemple pour des tests unitaires. **Note : vous devez faire des tests pour la classe RangPoker afin de montrer qu'il respecte l'ordre des rangs avec la méthode compareTo().**

Normalisation des noms de sous classe de AbstractAnalyseurRang. Afin de faciliter le contrôle de vos classes analyseurs, on vous demande de respecter les noms de classes suivants (tous sans accent):

- CarteSuperieure
- Paire
- DeuxPaires
- Brelan
- Quinte
- Couleur
- MainPleine
- Carre
- QuinteCouleur
- Quintuplet
- QuinteRoyale

De plus, assurez-vous d'implémenter les méthodes suivantes dans la classe Main, puisqu'elles seront contrôlées par le chargé de laboratoire avec un test unitaire "standardisé" :

getCartes() : Collection<Carte>
getRangPoker() : RangPoker

[Selon l'API du type interface List]
add(c : Carte) : boolean
remove(c : Carte) : boolean
size() : int

[Selon l'API du type interface Comparable<Main>]
compareTo(m : Main) : int

Il s'agit des mêmes noms présentés dans les diagrammes UML ci-dessus.

La classe ComparateurCartesInversé est simplement l'équivalent du comparator retourné par `Collections.reverseOrder()`, qui permet d'inverser l'ordre d'un tri (utile pour les TreeSet). L'utilisation de cette classe est facultative. Elle existe pour montrer comment un tel comparator fonctionne.

Rang des mains

Voici la liste des rangs de main, dans l'ordre ascendant :

1. Carte supérieure : des cartes qui ne forment aucune combinaison. Voici la valeur des dénominations de cartes dans l'ordre décroissant : l'as, le roi, la dame, le valet, le 10, le 9, le 8, le 7, le 6, le 5, le 4, le 3 et le 2. Voir la classe Dénomination.java pour plus d'informations.
2. Paire : deux cartes de même dénomination, par exemple, 2 deux, 2 valets, 2 as.
3. Deux paires : deux paires distinctes, par exemple, deux cinq et deux valets.
4. Brelan : trois cartes de même dénomination, par exemple, 3 dix.
5. Quinte (suite, séquence, straight) : une séquence de cinq cartes, pas toutes de la même couleur. Bien que l'as ait la dénomination supérieure, il peut être utilisé pour former une séquence as-2-3-4-5. Pourtant, il doit être toujours au début ou à la fin d'une séquence.
6. Couleur (flush) : Cinq cartes de la même couleur qui ne forment pas de séquence.
7. Main pleine (full house) : Un brelan et une paire. Le rang de la main pleine est déterminé par le triple et, dans un cas d'égalité des triples, par la paire.
8. Carré : Quatre cartes de même dénomination.
9. Quinte couleur (suite couleur, straight flush) : Cinq cartes de la même couleur formant une séquence.
10. Quintuplet : Cinq cartes de même dénomination, seulement possible avec une frime normalement.
11. Quinte royale : Une quinte couleur dont la carte la plus haute est l'as.

Pour comparer les rangs de mains, il y a des cas spéciaux :

- En général, c'est la carte « kicker » qui départage les mains. Cette règle s'applique lorsque deux mains ont la même *paire*, ou les mêmes *deux paires*. La carte « kicker » est la carte supérieure succédant à la paire. **[réalisation de ce cas est facultative, pour un demi point bonus, seulement si les autres fonctionnalités sont réalisées correctement]**
- Dans le cas de deux mains avec *deux paires*, c'est la paire supérieure qui l'emporte, ou bien la paire inférieure en cas d'égalité pour la paire supérieure.
- Dans le cas de deux *mains pleines* (full houses) ayant le même triple, c'est le rang de la paire qui détermine la main la plus forte.
- Dans le cas de deux *couleurs* (flushes), la main la plus forte est déterminée par le rang des cartes individuelles, en partant avec la plus haute. Dans ce cas, il n'y a pas de couleur plus forte que les autres.
- Pour tous les autres cas, par exemple, une *carte supérieure* qui est égale pour deux mains, deux *séquences* avec la même dénomination supérieure, deux *couleurs* (flushes) avec les mêmes valeurs, etc., c'est égalité.

Rapport de laboratoire

Remettez un rapport composé de 6 à 10 pages contenant :

- Au moins un (plusieurs si nécessaire) diagramme de classes avec explication présentant les classes logicielles et les classes de tests (1-3 page)
- Un diagramme de séquence expliqué présentant la dynamique du patron GoF « chaîne de responsabilités » dans le contexte de votre solution (1 page)
- Quatre nouvelles décisions de conception expliquées (2 à 4 pages)
    - Expliquez le problème rencontré
    - Décrivez la solution retenue
    - Discutez des alternatives considérées
    - Discutez des améliorations possibles
- Trois décisions d'implémentation expliquées (1-3 page)

- Une discussion pour expliquer comment l'application des patrons GoF facilite l'extensibilité de votre logiciel.(0.5 à 1 page)
- Une discussion pour expliquer ce qu'il faudrait changer pour supporter les frimes et les mains à sept cartes. (0.5 à 1 page)

Attention à la qualité du français (jusqu'à 10%).

Attention au plagiat!!!

Nombre de séances

Trois (3) séances de trois heures.

Trucs et astuces

Stratégies de test unitaire

Considérez les conseils suivants pour vos cas de test unitaire :

- tester toutes les méthodes, y compris le constructeur, les getX()/setX(), le compareTo(), etc.
- pour chaque méthode, faire varier les paramètres de l'appel afin de
  - tester les fonctionnalités de base (appel normal et typique de la méthode)
  - tester les cas limites (valeurs en dessous, au dessus des limites)
  - tester avec des références « null »
  - tester des cas qui **devraient générer les exceptions** « checked » et les **intercepter** (voir CarteTest.java pour des exemples)
  - ne pas tester les exceptions « unchecked » des méthodes
- ne pas tester la performance

Rang et comparaison de rang

En principe, il faut le nombre minimum de cartes pour l'identification d'un rang. Par exemple, une paire n'est que deux cartes, donc une main à deux cartes ayant la même dénomination devrait être reconnue comme une paire. Par contre, une main pleine doit avoir cinq cartes au minimum.

Alors, pour la fonction RangPoker.compareTo(), si deux mains n'ont pas le même nombre de cartes, il ne devrait pas y avoir de problème pour comparer leurs rangs. Par exemple, une main à cinq cartes avec le rang de "carte supérieur" est inférieure à une main à deux cartes avec le rang de "paire" (par exemple). Les tests unitaires devraient tenir compte de ces possibilités.

Voici quelques exemples de mains, leur rang et la comparaison entre rangs. Un but de ce laboratoire est aussi de concevoir de bons tests unitaires, alors on ne vous donne pas beaucoup d'exemples ici pour cette raison.

main1	rang1	main2	rang2	rang1.compareTo(rang2)
10 cœur, 9 pique, 3 trèfle, valet trèfle, 9 carreau	Paire	as cœur	Carte supérieure	entier positif
10 cœur, 9 pique, 3 trèfle, valet trèfle, 9 carreau	Paire	as cœur, 3 pique, as trèfle, 3 carreau	Deux paires	entier négatif
as trèfle, 3 carreau, 4 trèfle, 5 pique, 2	Quinte	3 pique, dame pique, 10 pique,	Couleur	entier négatif

carreau, valet cœur		2 pique, roi pique		

Analyseurs

Il peut être utile d'implémenter des méthodes suivantes pour faciliter l'analyse des mains:

```
/**
 * @param main
 * @return true si toutes les cartes dans la main sont de la même couleur
 */
public static boolean estMêmeCouleur(Main main);

/**
 * @param main
 * @return true si toutes les cartes dans la main sont dans une série
 */
public static boolean estEnSérie(Main main);

/**
 * Identifie les groupes ayant N cartes avec la même dénomination
 * Inspiré de http://6170.lcs.mit.edu/www-archive/Old-2002-Fall/psets/ps2/ps2.html
 * p.ex. si cherche des paires dans la liste {4 de trèfle, 4 de carreau, 4 de pique, 4 de coeur, 8 de pique},
 * cette méthode trouvera deux groupes : {4 de trèfle, 4 de carreau} et {4 de pique, 4 de coeur}.
 * La méthode retournera finalement le SortedSet {4 de trèfle, 4 de pique}.
 *
 * @param cartes la liste de cartes pour la recherche
 * @param n
 * @return un SortedSet<Carte> contenant la carte inférieure de chaque groupe de N cartes étant de la même dénomination
 *
 */
protected static SortedSet<Carte> trouverDénominationN(Iterator cartes, int n);
```

Pour changer l'ordre de tri d'un SortedSet ou d'autre élément ordonné des Collection de Java, il suffit de l'instancier en lui passant un Comparator personnalisé.

Références

Collection interfaces (tutoriel Java) [à jour pour Java 1.5]

Tutoriel sur JUnit.

Tactiques de test