Programmation et Langages de Programmation

Licence Informatique, septembre 2001

Tout document autorisé, durée 4 heures

Justifier clairement et succinctement vos choix, et décrivez brièvement vos algorithmes lorsque vous le jugez nécessaire.

Toute réponse difficile à comprendre sera considérée comme fausse.

Vous pouvez rajouter du code (méthode, donnée ou classe) non demandé à chaque fois que vous le jugerez nécessaire. Ecrivez entièrement ce code, en particulier dans le cas d'exceptions, de classes abstraites ou d'interfaces.

Vous pouvez toujours considérer une classe ou une méthode demandée dans une question précédente comme disponible, même si vous n'avez pas traité cette question. Il vous est également conseillé de lire le sujet entièrement avant de commencer votre travail.

Si vous souhaitez utiliser une classe (ou une méthode particulière) de l'API Java dont vous ne vous souvenez pas du nom, donnez-lui un nom explicite et précisez-le clairement sur votre copie.

Le but de ce problème est de se munir d'un ensemble de classes pour traiter des jeux de manière la plus générique possible.

Pour commencer, on considère les interfaces suivantes :

```
public interface Jeu {
    public Reponse jouer(Coup coup) throws CoupInvalideException;
}

public interface Coup {
}

public interface Reponse {
}
```

On utilisera ces interfaces pour programmer un jeu à un seul joueur, le MasterMind. Il s'agit pour le joueur de découvrir une combinaison de k couleurs en moins de n tentatives. Le nombre de couleurs possibles sera désigné par la variable m.

La combinaison à découvrir et les propositions du joueur seront représentées à l'aide d'un tableau de k valeurs entières comprises entre 0 et $m - 1$.

La réponse à une proposition sera donnée à l'aide de deux valeurs : le nombre de couleurs correctes et bien placées et le nombre de couleurs correctes, mais décalées.

Par exemple, si la combinaison à découvrir est (1, 2, 1, 3), et la proposition (1, 1, 2, 2), la réponse sera 1 couleur bien placée (le 1 en première position) et deux décalées (le 2 et le 1 respectivement en deux et troisième positions sur la combinaison à découvrir).

Question 1 Ecrire les classes `CoupMasterMind` et `ReponseMasterMind` qui implémentent respectivement les interfaces `Coup` et `Reponse` pour le jeu du MasterMind.

Question 2 Ecrire la classe `MasterMind` qui implémente l'interface `Jeu`. La taille de la combinaison, le nombre de tentatives autorisées et le nombre de couleurs possibles seront passés à la construction. Un coup sera considéré invalide dans les cas suivants :

1. le nombre de tentatives autorisées est dépassé ;
2. la proposition ne contient pas k valeurs ;
3. la proposition contient une couleur dont la valeur n'est pas comprise entre 0 et $m - 1$.

Pour générer la combinaison à découvrir, on utilisera la méthode d'instance `public int nextInt(int n)` de la classe `java.util.Random` qui renvoie un entier aléatoire compris entre 0 et $n - 1$.

Question 3 Implémenter la classe `CoupInvalideException`.

On considère maintenant une nouvelle interface `Joueur` qui fournit un coup en fonction du coup précédent et de la réponse donnée. Pour obtenir le premier coup, les valeurs de `precedent` et `reponse` passées en paramètre vaudront `null`.

```
public interface Joueur {
    public Coup suivant(Coup precedent, Reponse reponse);
}
```

Question 4 Ecrire une classe `JoueurMasterMindAleatoire` qui implémente l'interface `Joueur` et qui renvoie une proposition aléatoire sans tenir compte des coup et résultat passés en paramètre.

Ecrire une classe `JoueurMasterMindLecture` qui implémente l'interface `Joueur` et qui demande le coup suivant sur l'entrée standard après avoir affiché la dernière réponse. On supposera pour cela disponible une méthode de classe `public static int [] lireEntiers(int n)` d'une classe `Lecture` qui lit n entiers sur l'entrée standard.

Question 5 On souhaite pouvoir facilement implémenter des joueurs prenant

en compte l'ensemble des coups et des réponses depuis le départ du jeu. Pour cela, concevoir une classe abstraite `JoueurAvecMemoire` qui implémente l'interface `Joueur` et pour laquelle l'appel de la méthode
`public Coup suivant(Coup precedent, Reponse reponse)` tiendra compte de l'ensemble des coups et des réponses. Le caractère abstrait de cette classe permettra de paramétrer la stratégie souhaitée.

Question 6 On souhaite pouvoir calculer le temps moyen utilisé par un joueur pour calculer le coup suivant. On utilisera pour cela l'expression
`new Date().getTime()` qui retourne le nombre de millisecondes écoulées depuis le 1 janvier 1970. La classe `Date` appartient au paquetage `java.util`.

Une première solution envisagée est d'étendre par héritage chaque classe de joueur. Donner, en deux ou trois lignes, au moins deux inconvénients de cette méthode.

La solution retenue est d'utiliser le modèle du décorateur. En suivant ce modèle, écrire une classe `JoueurAvecTemps` qui décore les classes concrètes de joueurs en calculant le temps moyen en seconde pris par le calcul des coups.

Question 7 Ecrire la méthode `public static void main(String [] args)` d'une classe `TestMasterMind` qui permette de jouer au MasterMind en utilisant l'entrée et la sortie standards. Le nombre de coups autorisés, de couleurs disponibles et la taille de la combinaison seront précisés sur la ligne de commande. Le temps moyen utilisé pour proposer les coups sera également affiché à la fin de chaque partie.

Le jeu de Nim est un jeu à deux joueurs. On considère au départ k paquets de $n_1, \ldots, n_k$ jetons. A chaque coup, un joueur retire d'un seul paquet un nombre de jetons non nul. Le perdant est celui qui ramasse le dernier jeton.

Question 8 Reprendre les questions 1, 2, 4 et 7 pour le jeu de Nim. Pour la question 7, le joueur qui sera opposé au joueur de l'entrée standard sera un joueur qui joue au hasard.