Licence d'informatique 2004-2005
Examen Programmation orientée objet
Juin 2005

Durée de l'épreuve : 2 heures

TRÈS IMPORTANT : respectez l'ordre des questions. La lisibilité sera un critère important pour la notation.

Les seuls documents autorisés sont les supports de cours distribués pendant le cours par les enseignants. Tout autre document est interdit, en particulier les énoncés et corrigés des TP. Si vos possédez un document interdit, déposez-le hors de votre portée et de votre vue.

Les téléphones portables doivent être éteints, même si vous n'avez pas d'autre moyen de lire l'heure...

1 (4 points) Un programmeur débutant a écrit cette méthode pour renvoyer le plus grand élément d'une liste d'entiers :

```
public static Integer max(ArrayList<Integer> l) {
  int max = Integer.MIN_VALUE;
  for (int i = 0; i < l.size(); i++) {
    if (l.get(i) > max) {
      max = l.get(i);
    }
  }
  return max;
}
```

(a) Il a oublié de prendre en compte le cas où `l` est une liste vide. Modifiez ce code pour que la méthode lance une exception `NoSuchElementException` si la liste est vide. Cette exception est une classe fille de `RuntimeException` ; elle est située dans le paquetage `java.util`.

(b) Que pensez-vous de la signature de la méthode ? Essayez de l'améliorer (**sans changer le corps de la méthode**) pour qu'on puisse utiliser la méthode dans plus de circonstances.

Donnez quelques lignes de code (moins de 5) dans lesquelles votre nouvelle méthode est appelée alors que l'ancienne méthode n'aurait pas pu être appelée.

Est-ce que la solution que vous donnez peut impliquer des problèmes de performances (expliquez votre réponse si besoin est) ?

(c) Le programmeur débutant veut écrire une nouvelle classe d'exceptions pour traiter le cas où la liste est vide. Elle s'appellera `ListeVideException` et sera une exception contrôlée.

Écrivez pour lui une telle classe et récrivez la méthode `max` pour l'utiliser.

Qu'est-ce que ça changera pour les utilisateurs de la méthode `max` (par rapport à la version de la question (a)) ?

2 (4 points) Ce même programmeur débutant souhaite généraliser sa méthode et il écrit pour cela le code suivant :

```
public static <T> T maxG(ArrayList<T> l) {
  T max = l.get(0);
  for (int i = 1; i < l.size(); i++) {
    if (l.get(i) > max) {
      max = l.get(i);
    }
  }
  return max;
}
```

Dans toutes les questions de cet exercice, pour simplifier on ne vous demande pas de prendre en compte le cas où la liste est vide.

(a) Évidemment son code ne compile pas. Indiquez la ligne qui va provoquer une erreur de compilation. Donnez (en français) le message d'erreur qui pourrait être affiché par le compilateur ; on ne vous demande pas le message exact mais l'idée contenue dans ce message.

(b) Modifiez la méthode pour qu'elle fonctionne. La méthode ne prendra qu'un seul paramètre d'un type `ArrayList` générique (à vous de préciser le type).

(c) Améliorez cette méthode pour qu'elle accepte une collection de n'importe quel type (et pas seulement un `ArrayList`), par exemple une collection de type `Set`.

(d) Utilisez la méthode `maxG` pour implémenter la méthode `max` de la première question de l'exercice 1.

3 (2 points) Sur votre feuille, écrivez la (ou les) bonne réponse pour les questions suivantes.

Répondez sur le modèle suivant :

(a) A1 et B1

(b) B2

(c) ne sais pas

(d) B4 et D4

Toute mauvaise réponse enlève 0,5 point. Toute bonne réponse ajoute 0,5 point. Vous pouvez ne pas répondre à une des questions (vous écrivez "ne sais pas").

(a)
```
public class Entier {
  private int i;
  public Entier(int i) {
    this.i = i;
  }
  public void setVal(int i) {
    this.i = i;
  }
  public int getVal() {
    return i;
  }
  public String toString() {
    return i + "";
  }
}

public class Main {
  public static void main(String[] args) {
    Entier i = new Entier(1);
    Entier j = i;
    j.setVal(2);
    System.out.println(i);
  }
}
```

Réponses possibles :

A1. L'exécution de `Main` affiche 1

B1. L'exécution de `Main` affiche 2

C1. L'exécution de `Main` affiche autre chose que 1 ou 2.

(b)
```
public class A {
  public void m() {
    System.out.println("A");
  }
}

public class B extends A {
  public void m() {
    System.out.println("B");
  }
}

public class Main {
  public static void main(String[] args) {
    A b = new B();
    ((A)b).m();
  }
}
```

Réponses possibles :

A2. L'exécution de `Main` affiche A.

B2. L'exécution de `Main` affiche B.

C2. Ce code ne compile pas (indiquez la ligne en cause).

D2. Ce code compile mais provoque une erreur à l'exécution (indiquez la ligne en cause).

(c)
```
public abstract class A {
  public abstract int m();
  public int n() {
    return 2 * m();
  }
}
```

Réponses possibles :

A3. La méthode `n` doit être déclarée abstraite.

B3. La méthode `n` ne doit pas être déclarée abstraite.

C3. On peut choisir de déclarer la méthode `n` abstraite ou pas abstraite. Les 2 solutions sont bonnes.

(d) **Réponses possibles :**

A4. Pour utiliser la classe `java.util.ArrayList` dans le code d'une classe, le code doit nécessairement comporter au début "`import java.util ;`" .

B4 Pour utiliser la classe `java.util.ArrayList` dans le code d'une classe, le code doit nécessairement comporter au début "`import java.util.* ;`".

C4 Pour utiliser la classe `java.util.ArrayList` dans le code d'une classe, le code doit nécessairement comporter au début "`import java.util.ArrayList ;`".

D4 On peut utiliser la classe `java.util.ArrayList` dans le code d'une classe sans importer quoi que ce soit au début.

4 (10 points)

Le but de cet exercice est de proposer une modélisation orientée objets (Java) d'un jeu de dames. **Ne paniquez pas ! Il n'y a aucun piège.** La solution que vous allez développer sera forcément **incomplète et imparfaite** parce qu'il ne s'agit que de la première phase de réflexion. Mais vous devez être capable de proposer rapidement une modélisation **raisonnable et cohérente**. Il y a de nombreuses solutions différentes possibles...

Les éléments que vous devez obligatoirement modéliser sont :

- Le damier : le plateau avec ses cases.
- Les acteurs : joueurs et arbitres. Les arbitres vérifient que les déplacements demandés par les joueurs respectent les règles et signalent les fins de partie. Chaque acteur peut être humain ou automatique. Les joueurs automatiques n'ont pas forcément tous la même stratégie.
- Les pièces : pions et dames.
- Le jeu lui-même : gestion des parties. Dans chaque partie, il y a deux joueurs et un arbitre.

(a) **Les classes (sur 3 points)**

Proposer une liste de classes pertinentes pour ce jeu. Ne donnez pas la liste des membres. Précisez pour chaque classe si elle est concrète, abstraite ou interface. Précisez également toutes les relations d'héritage.

(b) **Le code (sur 6 points)**

Donnez le code complet de toutes les classes qui représentent les pièces. Si vous n'avez pas plusieurs classes pour gérer les pièces, revenez un peu en arrière pour corriger cela...

En donnant ce code, vous allez forcément être amené à appeler des membres de classes de l'application dont vous n'avez pas fait le détail. Dans ces cas, ajoutez un commentaire Java qui indique rapidement ce que représentent et réalisent ces membres.

N'oubliez pas la JavaDoc des attributs et méthodes que vous écrivez.

(c) **Les classifieurs de Java (sur 1 point)**

En technologie orientée objets, on donne le nom de *classifieurs* aux différentes sortes de classes. Par exemple, en Java, il y a de nombreuses sortes de classifieurs dont les classes concrètes, les classes abstraites et les interfaces.

Citez d'autres sortes de classifieurs de Java.

Dans votre modélisation du jeu de dames, vous avez sans doute utilisé plusieurs sortes de classifieurs. Expliquez quelles sont les différences entre les classes concrètes, les classes abstraites et les interfaces et donnez les limitations techniques de chacune (exemple : les classes concrètes ne peuvent pas contenir de méthode abstraite). Utilisez, à chaque fois que possible, les classifieurs de votre modélisation pour illustrer vos arguments et exemples.

Licence d'informatique 2004-2005
Programmation orientée objet
Correction des exercices de l'examen de juin 2005

1

(a) Il a oublié de prendre en compte le cas où `l` est une liste vide. Modifiez ce code pour que la liste lance une exception `NoSuchElementException` si la liste est vide. Cette exception est une classe fille de `RuntimeException`.

```
public static Integer max(ArrayList<Integer> l) {
  if (l.isEmpty()) {
    throw new NoSuchElementException("Liste vide");
  }
  int max = Integer.MIN_VALUE;
  for (int i = 0; i < l.size(); i++) {
    if (l.get(i) > max) {
      max = l.get(i);
    }
  }
  return max;
}
```

(b) Que pensez-vous de la signature de la méthode ? Essayez de l'améliorer (**sans changer le corps de la méthode**) pour qu'on puisse utiliser la méthode dans plus de circonstances.

Il vaut mieux mettre `List<Integer>` comme type du paramètre. Ca n'enlève rien et ça rend la méthode plus utilisable.

Donnez quelques lignes de code dans lesquelles votre nouvelle méthode est appelée alors que l'ancienne méthode n'aurait pas pu être appelée.

```
LinkedList<Integer> l = new LinkedList<Integer>();
l.add(12);
int m = max(l);
```

Est-ce que la solution que vous donnez peut impliquer des problèmes de performances (expliquez votre réponse si besoin est) ?

On peut avoir des problèmes de performance car la méthode utilise la méthode `get` qui n'est pas performante si on passe une `LinkedList`. Pour éviter ces problèmes il faudrait en fait tester le type de liste et écrire 2 codes différents : si la liste n'a pas d'accès direct à ses éléments et si elle est d'une taille minimale, il vaut mieux parcourir la liste avec un itérateur. En fait le parcours de la liste par un itérateur convient aussi sans doute pour une liste de type `ArrayList` (à tester sur votre JVM pour les performances).

(c) Le programmeur débutant veut écrire une nouvelle classe d'exceptions pour traiter le cas où la liste est vide. Elle s'appellera `ListeVideException` et sera une exception contrôlée.

Écrivez pour lui une telle classe et récrivez la méthode `max` pour l'utiliser.

```
public class ListeVideException extends Exception {
  public ListeVideException() {
```

```
  }

  public ListeVideException(String message) {
    super(message);
  }

public static Integer max(List<Integer> l)
    throws ListeVideException {
  if (l.isEmpty()) {
    throw new ListeVideException("Liste vide");
  }
  int max = Integer.MIN_VALUE;
  for (int i = 0; i < l.size(); i++) {
    if (l.get(i) > max) {
      max = get(i);
    }
  }
}
```

2

(a) Évidemment son code ne compile pas. Indiquez la ligne qui va provoquer une erreur de compilation. Donnez (en français) le message d'erreur qui pourrait être affiché par le compilateur ; on ne vous demande pas le message exact mais l'idée contenue dans ce message.

`if (l.get(i) > max)`

L'opérateur de comparaison ne peut être appliqué qu'à des nombres (`int`, `double`,...) ou aux classes englobantes depuis le JDK 1.5. Remarque : ne peut s'appliquer qu'aux instances de `Integer`, `Double`,... mais pas aux instances de `Number`.

(b) Modifiez la méthode pour qu'elle fonctionne. La méthode ne prendra qu'un seul paramètre d'un type `ArrayList` générique (à vous de préciser le type).

```
public static <T extends Comparable<? super T>> T maxG(List <T> l) {
  T max = l.get(0);
  for (int i = 1; i < l.size(); i++) {
    if (l.get(i).compareTo(max) > 0) {
      max = l.get(i);
    }
  }
  return max;
}
```

(c) Améliorez cette méthode pour qu'elle accepte une collection de n'importe quel sorte (et pas seulement un `ArrayList`).

```
public static <T extends Comparable<? super T>> T maxG(Collection<T> c) {
  Iterator<T> it = c.iterator();
  T max = it.next();
  while (it.hasNext()) {
    T elt = it.next();
    if (elt.compareTo(max) {
```

```
      max = elt;
    }
  }
  return max;
}
```

Remarque : on pourrait généraliser encore plus en prenant la signature suivante :

```
public static <T extends Comparable<? super T>> T maxG(Iterable<T> c)
```

(d) Utilisez la méthode `maxG` pour implémenter la méthode `max` de la première question de l'exercice 1.

```
public static Integer max(ArrayList<Integer> l) {
  return maxG(l);
}
```

3

(a) B1

(b) B2

(c) B3

(d) D4