Examen de programmation par objets

Septembre 2001 - UE2241MBE

Licence informatique (durée 2h)

Responsable J.Ferber

Tous documents manuscrits sur la base des cours, TD, TP autorisés.

(Pas de livres ni photocopies de livres ni impressions de cours issu du Web)

Exercice 1: Questions de cours (5 pts)

Etant donné le programme suivant:

```
class Bidule {
   int a=0;
   Bidule(int x){
      a = x;
   }
}
```

a) Indiquez ce que retournent les deux expressions suivantes (en d'autres termes, quelle est la "valeur" de ces expressions)?

```
new Bidule(15)
new Bidule[15]
```

b) Soient les classes suivantes:

```
class Chose extends Bidule {
   Chose(int x){ super(x);}
}

class Truc extends Chose {
   Truc(int x){ super(x);}
}
```

Indiquez, pour chacune des expressions suivantes, si elles sont correctes ou non, et dans ce cas, quel type d'erreur produisent-elles (et est ce une erreur de compilation, ou d'exécution) et comment peut on faire pour qu'elles ne produisent plus d'erreur?

```
Chose c1 = new Truc(10);
Bidule c2 = c1;
Truc c3 = c1;
```

Exercice 2 : Classes abstraite (15 pts)

L'objectif est de définir une classe abstraite **EnsembleTrie** destinée à gérer un ensemble d'objets triés et qui comporte une méthode abstraite

```
boolean superieur(Object o1, Object o2)
```

qui compare deux objets. Pour gérer un tableau trié d'objets d'un certain type, il faudra étendre la classe abstraite en une classe définissant la méthode **superieur** pour le type d'objets en question.

De plus, on désire qu'il soit impossible de mémoriser deux fois le même objet dans cet ensemble.

Pour représenter cet ensemble d'objets, on décide d'utiliser tout d'abord un **Vector**, c'est à dire un type de collection Java qui permet de manipuler les collections de

Voici le squelette de cette classe :

```
public abstract class EnsembleTrie {
   private Vector contenu = new Vector();
   Public abstract boolean superieur(Object o1, Object o2);
   public void inserer(Object o){ ... }
   public void supprimer(Object o) { ... }
   public String toString(){ ... }
   public Object element(int i) { ... }
   public int taille() { ... }
}
```

1ère partie

Question 1: Définir les méthodes **inserer** et **supprimer** de cette classe (voir en annexe, les méthodes de la classe **Vector** dont vous pourriez avoir besoin).

Question 2 : Définir la méthode **toString** afin que les objets qui se trouvent dans l'ensemble soient affichés de la manière suivante:

{<elt1>, <elt2>, ... , <eltn>}

par ordre croissant.

Question 3: Définir les méthodes **element(int i)** qui retourne le ième objet dans l'ensemble trié, et **taille()** qui retourne la taille de l'ensemble.

Question 4 :

Définir la classe **EnsembleTrieChaine** qui décrit un ensemble trié de chaînes de (**String**) de manière à ce que le code suivant:

```
EnsembleTrieChaine e = new EnsembleTrieChaine() ;
e.inserer("toto");
e.inserer("titi");
e.inserer("tutu");
e.inserer("toto");
System.out.println(e);
```

Produise l'affichage suivant:

{titi, toto, tutu}

Question 5: Définissez la méthode **public String concat()** qui retourne une chaîne obtenue par concaténation de toutes les chaînes de l'ensemble. Ex:

```
String s = e.concat();
System.out.println(s);
```

Va produire:

tititototutu

Attention: il n'est pas possible d'accéder directement à l'attribut **contenu** (marquée **private**) de la classe **EnsembleTrie** à partir de la classe **EnsembleTrieChaine** ! D'autre part, nous n'avez pas le droit de créer un nouvel attribut **contenu**.

2ème partie

On désire maintenant utiliser non plus un **Vector** pour conserver les éléments de l'ensemble, mais un tableau d'objets. Ce tableau est construit avec une capacité

donnée. Lorsque la capacité est dépassée, on construit un nouveau tableau dont la nouvelle capacité est égale à celle de la précédente ajoutée d'un incrément. Voici la définition de la nouvelle classe:

```
public abstract class EnsembleTrieTableau {
   private int capacite=20;
   private int increment=10;
   private Object[] contenu;
   private int taille=0; // nombre d'objets dans 'contenu'
   public EnsembleTrieTableau (){... }
   public abstract boolean superieur(Object o1, Object o2);
   public void inserer(Object o){ ... }
   public void supprimer(Object o) { ... }
   public String toString(){ ... }
   public Object element(int i) { ... }
   public int taille() { ... }
}
```

Question 6: donnez le constructeur de cette classe.

Question 7: donnez le code des méthodes **`toString()`**, **`element(int i)`** et **`taille()`**

Question 8: donnez le code des méthodes **`insérer`** et **`supprimer`**

Annexe

On rappelle que la méthode

```
public int compareTo(String s)
```

permet de comparer deux chaînes de caractères et retourne un entier négatif si la chaîne courante précède (de manière lexicographique) s; 0 si les deux chaînes sont égales et un entier positif si la chaîne courante suit s.

Description des principales méthodes de Vector:

- **`void add(Object o)`** : ajoute l'objet **`o`** à la fin du **`Vector`**.
- **`public void add(int i, Object o)`** : insère l'objet **`o`** à l'indice **`i`** du vecteur et déplace l'élément qui s'y trouvait ainsi que tous les éléments suivants vers la droite (il incrémente leur indice d'une unité).
- **`Object elementAt(int i)`** : retourne l'élément qui se trouve à l'indice i.
- **`Enumeration elements()`** : retourne un **`java.util.Enumeration`** énumérant les objets contenus dans le **`Vector`**.
- **`void setElementAt(Object o, int i)`** : met à l'indice **`i`** l'objet **`o`**, éventuellement en écrasant l'élément précédent qui s'y trouvait. L'indice doit être au moins égal à 0 et être strictement inférieur à la taille actuelle du tableau. Ne modifie pas les indices des autres éléments.
- **`boolean isEmpty()`** : teste si l'instance concernée de **`Vector`** est vide.
- **`void removeAllElements()`** : vide l'instance concernée de **`Vector`**.
- **`boolean remove(Object o)`** : retire la première occurrence de l'objet **`o`** s'il figure dans le **`Vector`** et retourne **`true`**, sinon retourne **`false`**. Déplace vers la gauche tous les objets situés après o (décrémente leur indice d'une unité).
- **`void remove(int i)`** : retire l'objet qui se trouve à l'indice indiqué. Déplace vers la gauche tous les objets situés après cet indice en décrémentant leur indice d'une unité.
- **`int size()`** : retourne le nombre d'objets contenus par le **`Vector`**.