Travaux Dirigés – Java
Séance 3

M. Safey El Din

Janvier 2004

1 Trier des entiers en utilisant un tas

Utiliser la classe `Tas` de l'exercice précédent pour trier des entiers indiqués sur la ligne de commande, par ordre croissant ou décroissant.

- On étendra d'abord la classe `Tas` en une classe `TasEntiers` gérant un tas d'entiers; il suffit pour cela de définir la méthode `plusGrand` pour des `Integer`.
- On définira une seconde classe `TriTasEntiers` dans laquelle il y aura :
  - une méthode utilisant `TasEntiers` pour trier un tableau d'Integer par ordre décroissant.
  - une méthode utilisant `TasEntiers` pour trier un tableau d'Integer par ordre croissant.
- On définira une classe contenant une méthode `main` pour tester les tris d'entiers par ordre croissant ou décroissant.

2 Définir et utiliser l'interface `Ordonnable`

L'objectif est de de définir une classe capable de gérer un tableau trié d'objets, dès que ces objets appartiennent à une classe implémentant l'interface `Ordonnable`.
On construire :

- Une interface de nom `Ordonnable` contenant uniquement le prototype :
  `boolean plusGrand(Object O);`
  Lorsque cette méthode sera implémentée, elle rendra `true` si l'instance courante doit être considérée comme plus grande que l'instance indiquée dans la parenthèse, et `false` sinon (peu importe pour le cas de l'égalité).
- Une classe `TableauTrie` gérant un tableau d'objets Ordonnable(s) qui reste toujours trié par rapport à la relation définie par la méthode `plusGrand`, par ordre croissant. Cette classe devra contenir au moins :
  - une méthode `inserer` pour insérer une instance de `Ordonnable` dans le tableau
  - une méthode `supprimer` pour supprimer une instance de `Ordonnable` du tableau. Si cette méthode utilise la méthode `equals` de `java.lang.Object`, il faudra utiliser la classe `TableauTrie` avec des objets qui redéfinissent cette méthode.
  - une méthode `toString` qui redéfinit la méthode `toString` de `Object`.
- Une classe `EntiersOrdonnables` qui implémente la classe `Ordonnable` ; cette classe représente essentiellement un `int` pour lequel la méthode `plusGrand` est définie ; elle doit contenir en particulier un attribut privé de type `int`. Cette classe redéfinira aussi la méthode `toString` de la classe `Object` et devra sans doute redéfinir la méthode `equals` de la classe `Object` pour que la méthode `supprimer` de la classe `TableauTrie` puisse fonctionner.
- Une classe, EssaiTableauTrie, qui contienne une méthode main qui prend en arguments une liste d'entiers et qui :
  - met chaque entier dans une instance d'`EntiersOrdonnables`
  - insère les instances d'`EntiersOrdonnables` dans une instance de `TableauTrie`
  - affiche les entiers par ordre croissant.
  - demande un entier à supprimer, et le supprime s'il existe
  - affiche la liste triée des entiers restants.

3 Une classe abstraite pour gérer un tableau trié d'`Object(s)`

L'objectif est de de définir une classe abstraite destinée à gérer un tableau trié d'Object(s) et comportant une méthode abstraite plusGrand. Cette classe devra comparant deux Object. Pour gérer un tableau trié d'objets d'un certain type, il faudra l'étendre la classe abstraite en une classe définissant la méthode plusGrand pour le type d'objets en question.
On construira :

- Une classe abstraite `TableauTrieAbstrait` gérant un tableau d'`Object(s)` qui reste toujours trié par ordre croissant par rapport à la relation définie par une méthode abstraite `plusGrand`. On propose de reprendre le corrigé de l'exercice concernant un tableau triés d'objets implémentant l'interface `Ordonnable`. Cette classe devra contenir au moins
  - une méthode abstraite `plusGrand`;
  - une méthode `inserer` pour insérer une instance de `Object` dans le tableau;
  - une méthode `supprimer` pour supprimer une instance de `Object` du tableau. Si cette méthode utilise la méthode `equals` de `java.lang.Object`, il faudra que la classe des objets qui seront dans le tableau redéfinisse cette méthode.
  - une méthode `toString` qui redéfinit la méthode `toString` de `Object`.
- Une classe `TableauTrieEntiers` qui étend la classe `TableauTrieAbstrait`; cette classe est destinée à gérer un tableau trié d'`Integer`. Il faut essentiellement y définir la méthode `plusGrand` pour des `Integer`.
- Une classe, `EssaiTrieAbstrait` qui contienne une méthode `main` qui prend en arguments une liste d'entiers et qui :
  - met chaque entier dans une instance d'`Integer`;
  - insère les instances d'`Integer` dans une instance de `TableauTrieEntiers`;
  - affiche les entiers par ordre croissant;
  - demande un entier à supprimer, et le supprime s'il existe;
  - affiche la liste triée des entiers restants.

4 Un «vecteur» trié

Le paquetage java.util contient une classe, la classe `Vector`, qui sert à gérer un tableau d'`Object` sans se préoccuper de questions de dépassement.
Voici quelques méthodes de cette classe. Pour plus d'informations, consultez-la en ligne.
Un constructeur : `Vector()`
Quelques unes des méthodes :

- `void addElement(Object)` : ajoute à la fin l'"objet";
- `Object elementAt(int)` : retourne l'élément qui se trouve à l'indice indiqué;
- `Enumeration elements()` : retourne un `java.util.Enumeration` énumérant les objets contenus dans le tableau;
- `void setElementAt(Object, int)` : met à l'indice indiqué l'instance d'`Object` indiqué. L'indice doit être au moins égal à 0 et être strictement inférieur à la taille actuelle du tableau.
- `boolean isEmpty()` : teste si l'instance concerné de `Vector` est vide.
- `void removeAllElements()` : vide le tableau.
- `boolean removeElement(Object)` : retire la première occurrence de l'argument s'il figure et retourne `true`, sinon retourne `false`.
- `void removeElementAt(int)` : retire l'objet qui se trouve à l'indice indiqué.
- `int size()` : retourne le nombre d'objets contenus par le tableau.

Il s'agit dans cet exercice de reprendre l'exercice précédent, en remplaçant le tableau par un `Vector`. Vous verrez que cela simplifie le code.