Programmation Java TP #3

A. Pigeau

1 Exercice : [Interface]

Soit le cahier des charges suivants :
- un étudiant est définie par un nom, un prénom et sa moyenne générale
- le constructeur doit prendre en paramètre tous ses attributs

Questions :

1. implémenter la classe `Etudiant` (ne pas oublier la méthode `toString()`
2. implémenter un `Test` dans lequel plusieurs étudiants sont créés
3. dans votre test, ajouter tous les étudiants créés dans une liste (un `Vector` par exemple)
4. consulter la documentation sur la classe `java.util.Collections`, en particulier la fonction `sort()` de cette classe
5. consulter la documentation sur l'interface `Comparable<T>`
6. modifier la classe `Etudiant` pour qu'elle implémente l'interface `Comparable<T>`
7. implémenter une fonction `static` affichant les étudiants par moyenne générale ascendante

2 Exercice : [Interface & implémentation de pile]

1. définir une interface `Stack` déclarant trois méthodes usuelles d'une pile :
   - `void push (Object o)`
   - `Object pop ()`
   - `Object top ()`
   - `boolean isEmpty ()`
   - `String toString()`
2. définir une classe `ArrayStack` qui implémente l'interface `Stack` en utilisant un tableau. Tester la classe.
3. définir une classe `VectorStack` qui implémente l'interface `Stack` en utilisant un `Vector`. Tester la classe.

On désire maintenant disposer d'une méthode permettant d'empiler une pile sur une autre. Par exemple, si p_1 est du type `StackableStack` et s'est vu empiler successivement a, b et c, et que p_2, du type `Stack`, s'est vu empiler successivement d, e et f, on souhaite que l'appel à `p1.pushAll (p2)` aboutisse à ce qu'on puisse empiler successivement f, e et d dans p_1 (les éléments de p_2 sont dépilés).

Proposer une architecture de classes, classes abstraites et/ou interfaces permettant d'offrir cette fonctionnalité, selon les deux modes suivants :

1. définir une interface `StackableStack` spécifiant le comportement `pushAll` ;
2. définir les classes `ArrayStackableStack` et `VectorStackableStack` héritant respectivement de `ArrayStack` et `VectorStack`, et implémentant l'interface `StackableStack` ;
3. quel est l'inconvénient de cette approche ?
4. afin d'éviter la redondance de code, définir maintenant la classe `StackableStackImpl` qui factorise le code des méthodes `pushAll`.

5. nous voudrions que cette classe fonctionne avec un `ArrayStack` ou un `VectorStack`, mais elle ne peut pas hériter des deux classes. Une solution est d'utiliser une association entre un `Stack` et un `StackableStackImpl`.

   Implémenter la classe `StackableStackImpl`. Cette classe contient un attribut de type `Stack`, qui pourra ainsi pointer soit sur une instance de `ArrayStack`, soit sur une instance de `VectorStack`
6. redéfinir toutes les méthodes de `Stack` dans la classe `StackableStackImpl` (elle implémente donc l'interface `Stack`), en faisant en sorte qu'elle rappelle tout simplement les mêmes méthodes sur son attribut
7. tester le code suivant : (et faire en sorte que ce code fonctionne)

```
Stack a = new ArrayStack();
Stack v = new VectorStack();

v.push("ab");
v.push("cd");

// attention abs et a partage la meme pile...
StackableStackImpl abs = new StackableStackImpl(a);
abs.push("ef");
abs.push("gh");

abs.pushAll(v);
System.out.println(abs); // [ef, gh, cd, ab]
```

3 Exercice : [Les exceptions]

L'objectif de l'exercice est d'implémenter un programme qui divise deux nombres positifs ou nuls passés en paramètres sur la ligne de commande. Pour cela, il est nécessaire de convertir les chaînes de caractères en nombres réels, puis effectuer la division.

Les quatre exceptions suivantes doivent être traitées :
- Il y a moins de deux paramètres sur la ligne de commande
- Les chaînes de caractères ne correspondent pas à des nombres
- Le dénominateur est nul
- Les nombres sont négatifs

Les conversions de type sont opérées à l'aide des classes d'enveloppe de types primitifs (cf. documentation de l'API).

Les paramètres de la ligne de commande sont stockés en ordre d'apparition dans le tableau de chaînes de caractères passé en argument de la méthode `main`.

1. Créer les exceptions nécessaires pour votre programme (donc les classes non disponibles dans l'API standard). Pour chaque implémentation, vous devez redéfinir la méthode `getMessage()` (en retournant un message d'erreur lié à l'exception).
2. Implémenter un test devant prendre en argument deux entiers et lançant les exceptions liées aux événements *moins de deux arguments* et *les arguments ne correspondent pas à des nombres*. Tester votre code (en sachant que les exceptions doivent être traitées par votre `main`).
3. Définir une méthode de classe `diviser(double a, double b)` qui réalise la division et spécifie les exceptions de division par zéro et de nombre négatif. Le `main` doit invoquer cette méthode et traiter ce type d'exception.
4. Proposer la *récupération* de l'exception *nombre négatif* en réalisant la division de la valeur absolue des nombres négatifs à partir du traitement de l'exception.

4 Exercice : [Table dynamique générique]

Le but de cet exercice est d'écrire une classe permettant d'ajouter des éléments dans une table et de parcourir celle-ci.

1. Écrire la classe `Table`, implémentée à l'aide d'un tableau. Pour cela, définir un constructeur prenant en argument un entier correspondant à la taille maximale de la table, une méthode `add(Object)` permettant d'ajouter un élément dans la table et une méthode `size()` retournant le nombre d'éléments contenus dans la table.
2. Écrire la classe `TableIterator` qui implémente l'interface `java:util:Iterator` et qui permet d'énumérer séquentiellement tous les éléments d'une table.

   Rappel : Cette interface est composée de trois méthodes :

   (a) `hasNext()` qui renvoie `true` s'il reste encore des éléments à parcourir ;

   (b) `next()` qui renvoie l'élément parcouru et avance à l'élément suivant. S'il n'y a plus d'élément à parcourir, la classe devra déclencher l'exception `NoSuchElementException` ;

   (c) `remove()` qui supprime de la table l'élément que `next()` vient de renvoyer. Ici, nous n'implémenterons pas cette méthode, mais déclencherons plutôt l'exception `UnsupportedOperationException`.
3. Écrire une méthode `iterator()` dans la classe `Table`, renvoyant un objet de type `Iterator`. Tester la solution en affichant tous les éléments d'une table.
4. Transformer la classe `TableIterator` en classe interne simple de la classe `Table`. Tester.
5. Transformer cette classe en classe interne locale de la méthode `iterator()` de `Table`. Tester.
6. Enfin, transformer cette classe en classe anonyme. Tester.