Exercices Design Patterns

Adaptateur

Pour pouvoir accéder au contenu d'une collection, Java fournit des itérateurs. Un itérateur est un objet implantant l'interface `java.util.Iterator` et permettant d'accéder de manière uniforme aux objets d'une collection, sans s'occuper de la façon dont ces objets sont stockés. L'interface `Iterator` possède trois méthodes :

- `hasNext()` qui renvoie `true` si la collection possède encore des éléments non parcourus
- `next()` qui renvoie le prochain élément de la collection (sous la forme d'une instance d'`Object`) et qui avance d'un élément dans la collection
- `remove()` qui enlève l'élément courant

1. Écrivez une méthode statique qui prend un itérateur en paramètre et affiche toutes les chaînes de caractères se trouvant dans la collection correspondante.
2. Les anciennes classes représentant les collections en Java (e.g., `Vector`) implémentent une méthode `elements()` qui renvoie un objet de type `java.util.Enumeration`. L'interface `Enumeration` permet également de parcourir ces collections, mais pas d'enlever un élément de la collection. Elle possède deux méthodes :
   - `hasMoreElements()` qui renvoie `true` s'il y a encore des éléments à parcourir dans la collection
   - `nextElement()` qui renvoie le prochain élément (sous la forme d'une instance d'`Object`)

   Est-il possible d'utiliser la méthode statique écrite précédemment avec un objet de type `Enumeration` (sur une instance de `Vector` par exemple) sans modifier la méthode ni les interfaces `Enumeration` et `Iterator` ? Pourquoi ?
3. Proposez un diagramme de classes qui modélise et résout le problème en utilisant le pattern Adaptateur.
4. Écrivez une implémentation de l'adaptateur. Que faire pour la méthode `remove` de l'interface `Iterator` ?
5. Supplément :
   (a) Comment adapter une liste doublement chaînée en une pile ?

```
public interface Stack {
  void push(Object o);

  Object pop();
  Object top();
}
```

   (b) Comment adapter un ensemble d'entiers en une file de priorité ?

```
public interface PriorityQueue {
  void add(Object o);
  int size();

  Object removeSmallest();
}
```

Composite

Starbucks ofre un large choix de boissons (e.g., expresso, chocolat, café, décaféiné, thé...) et d'encas (e.g., pâtisserie, viennoiserie, muffin, cookie, donut, sandwich...). Il est également possible de commander un menu composé d'une boisson et d'un encas au choix, ofrant une remise de 10o/c par rapport au tarif vendus séparément. Les tarifs des boissons et des encas sont donnés à la figure 1. Nous souhaitons réaliser une application pour gérer les commandes des clients chez Starbucks. Dans un premier temps, cette application doit permettre d'afficher la liste des produits commandés avec leur prix, ainsi que le montant total de la commande.

Boisson	Tarif
Expresso	2.75
Chocolat	3.55
Café	1.75

Encas	Tarif
Pâtisserie	2.25
Viennoiserie	1.50
Muffin	2.25

FIGURE 1 - Tarifs des boissons et des encas

1. Proposez un diagramme de classes qui modélise le problème en utilisant le pattern Composite.
2. Écrivez une implémentation des classes du diagramme.
3. Écrivez un programme de test qui affiche la commande suivante : un menu avec une boisson chocolatée et un muffin, et deux cafés. L'affichage obtenu doit correspondre à :

```
- menu (boisson chocolatée + muffin)    5.22
- café    1.75
- café    1.75
Total  8.72
```