Algorithmique avancée - TD3

—o000o——o000o—

Dictionnaires et listes d'associations

1 Dictionnaires

On souhaite pouvoir gérer un inventaire qui liste pour un certain nombre de produits sa référence (unique et non vide) et son prix exprimé en euros sous la forme d'un nombre réel positif. Pour gérer un tel inventaire, on va utiliser un Dictionnaire qui étend `SDict` qui permet d'associer une clé, ici la référence du produit au format `String`, à son prix au format `Double`. Nous allons définir pour cela la classe `Inventaire` et ses méthodes dans les questions suivantes.

1. Quel type de dictionnaire doit-on mettre en place ? Quelle précaution doit-on prendre pour que l'on ne puisse pas déclarer un prix négatif ou insérer une clé vide ? Donner un exemple de programme principal qui utilise cette classe pour insérer plusieurs valeurs dont une négative.

2. Écrire une méthode `isEmpty()` qui retourne `true` si et seulement si le dictionnaire est vide.

3. Écrire une méthode `double getTotal()` qui permet de retourner le montant total de l'inventaire en faisant la somme des prix affectés à l'ensemble des produits.

4. Proposer une méthode `(double filterKeysByPrice(priceMax)` qui retourne la liste de toutes les références (clés) de produits dont le prix est inférieur ou égal à `priceMax`.

5. On souhaite maintenant convertir l'ensemble des prix à l'aide d'un taux de conversion multiplicatif. Écrire la méthode `mapConvert(double rate)` qui change les valeurs du dictionnaires en appliquant un taux de conversion passé en argument.

6. On souhaite maintenant simuler une augmentation de prix pour certains articles contenus dans l'inventaire. Pour cela on se propose de coder une méthode `changePrices(Inventaire priceChanges)` qui prend en argument un second inventaire contenant tout ou partie de l'inventaire de base et qui indiques pour ses articles, l'augmentation de prix à répercuter dans l'inventaire de base.

7. Définir une méthode `merge(Inventaire i)` qui permet de fusionner deux inventaires en ajoutant les (clés, valeurs) de l'inventaire `i` à l'inventaire courant. Dans le cas où une clé de `i` est déjà définie dans l'inventaire courant, il suffit de mettre à jour l'inventaire courant avec la valeur issue de l'inventaire `i`.

Solution :

```java
package dict;

import list.LList;
import list.SList;

public class Inventaire extends SDict<String, Double> {

    /**
     * Question 1. Overrides the put method from SDict to ensure
     * non-null keys and positive values
     * @param key
     * @param value
     */
    @Override
    public void put(String key, Double value) {
        if (key != null && value >= 0 && !containsKey(key)){
            keys.add(key);
            values.add(value);
            size++;
        }
    }

    /**
     * Question 2
     * @return
     */

    public boolean isEmpty(){
        return size() == 0;
    }

    /**
     * Question 3
     * @return
     */
    public double getTotal() {
        LList<String> ks = this.keySet();
        double sum = 0;
        for (int i = 0; i < ks.size(); i++){
            sum += this.get(ks.get(i));
```

```
41          }
42          return sum;
43      }
44
45      /**
46       * Question 4
47       * @param priceMax : max value for threshold
48       * @return
49       */
50
51      public LList<String> filterKeysByPrice(double priceMax){
52          LList<String> res = new SList<String>();
53          LList<String> ks = this.keySet();
54          for (int i = 0; i < ks.size(); i++){
55              if (this.get(ks.get(i)) <= priceMax) res.add(ks.
                    get(i));
56          }
57          return res;
58      }
59
60      /**
61       * Question 5
62       * @param rate
63       */
64      public void mapConvert(double rate){
65          LList<String> ks = this.keySet();
66          for (int i = 0; i < ks.size(); i++){
67              this.put(ks.get(i), get(ks.get(i)) * rate);
68          }
69      }
70
71      /**
72       * Question 6
73       * @param priceChanges contains items as keys and
             additional prices as values
74       *                     each price contained in TD3_1 dict
              are modified
75       *                     according to rules contained in
             the argument as a TD3_1 dict
76       */
77      public void changePrices(Inventaire priceChanges){
78          LList<String> ks = priceChanges.keySet(); // list of
                keys for values to be modified
79          for (int i = 0; i < ks.size(); i++){
```

```
80              if (this.containsKey(ks.get(i))) {
81                  double val = this.get(ks.get(i)); // get the
                       actual value for the key
82                  this.put(ks.get(i), priceChanges.get(ks.get(i
                       )) + val);
83              }
84          }
85      }
86
87      public void merge(Inventaire inv){
88          LList<String> ks = inv.keySet();
89          for (int i = 0; i < ks.size(); i++){
90              put(ks.get(i), inv.get(ks.get(i)));
91          }
92      }
93
94  }
```

java/Inventaire.java

2 Liste d'associations

On considère désormais le même inventaire représenté sous la forme d'une liste d'association, c'est-à-dire que chaque élément est maintenant représenté comme un couple Java défini comme un couple de type `Pair` possédant deux éléments. On s'intéresse ici à l'écriture d'une classe `InventaireAssoc` que l'on va compléter.

1. Écrire une méthode `put(Pair p)` dans la classe `InventaireAssoc` de façon à garantir que l'on ne peut ajouter que des couples pour lesquels les valeurs sont positives ou nulles. Vous pouvez cette fois-ci gérer les erreurs de saisies en renvoyant une exception de type `IllegalArgumentException`.

2. Proposer ensuite une méthode `getTotal()` qui calcule le montant total des objets présents dans l'inventaire. Observer les différences et les similarités avec la méthode précédente sur les dictionnaires.

Solution :

```
1  package dict;
2
3  public class InventaireAssoc extends SAssoc<String, Double> {
4
5      @Override
```

```
6       public void put(String key, Double value) throws
           IllegalArgumentException {
7           if (key != null && value >= 0){
8               super.put(key, value);
9           } else throw new IllegalArgumentException("Null keys
               or negative value forbidden");
10      }
11
12      public double getTotal(){
13          double sum = 0;
14          for (int i = 0; i < keySet().size(); i++){
15              sum += get(keySet().get(i));
16          }
17          return sum;
18      }
19  }
```

java/InventaireAssoc.java