

Programmation Orientée Objets avec Java

Chapitre 5 : Les Collections

Stéphane Malandain / Yassin Rekik

Exercice 1

Réalisez des fonctionnalités sur des listes. Complétez le code ci-dessous:

```
1  public class ListHelper {
2
3      /* Retourne une nouvelle liste où toutes les valeurs sont doublées */
4      public static List<Integer> doubleThat(List<Integer> is) {
5      }
6      /* Retourne une nouvelle liste où toutes les valeurs sont plus grandes ou égale à un seuil */
7      public static List<Integer> filterUpper(List<Integer> is, int value) {
8      }
9
10     public static void main(String[] args) {
11         /* Vos tests éventuels */
12     }
13 }
```

Exercice 2

Réalisez une classe `StringUtil`.

Cette classe doit fournir une première méthode statique qui prend un argument une chaîne de caractères de type `String` et retourne une `Map` dont les clés sont les caractères qui se trouvent dans la chaîne et la valeur le nombre d'occurrences de la lettre dans la chaîne.

Elle fournit une seconde méthode statique prenant une liste de chaînes de caractères en argument et compte la taille moyenne d'une chaîne dans la liste.

Exercice 3

Ecrivez une classe `ListUtil` qui comprend deux méthodes statiques :

- Une méthode qui prend une liste de doubles en argument et retourne la moyenne de tous les éléments.
- Une méthode qui prend une liste de doubles en argument et retourne une nouvelle liste de tous les éléments positifs appartenant à la première.

Exercice 4

Soit la collection ci-dessous :

```
1 Deque<String> stack = new LinkedList<>();
2 stack.addFirst("Bonjour");
3 stack.addFirst("Hello");
4 stack.addFirst("Hi");
```

Complétez les méthodes permettant de parcourir et d'afficher les éléments de cette collection en utilisant les méthodes associées au type de l'argument (Iterable, Iterator ou Collection) :

```
1 interface Loop {
2     public static void loop(Iterable<String> iterable) { ... }
3     public static void loop(Iterator<String> iterator) { ... }
4     public static void loop(Collection<String> coll) { ... }
5 }
```

Laquelle des trois méthodes sera appelée lors de cet appel:

```
Loop.loop( stack );
```

Exercice 5

Réécrivez correctement une classe `Properties` qui supprime totalement la notion d'héritage. Choisissez judicieusement la structure à utiliser pour stocker des propriétés. (indication : du côté des map 😊)

```
1 class Properties {
2     public String getProperty(String key) {
3         /* TODO */
4     }
5     public String getPropertyOrElse(String key, String defaultValue) {
6         /* TODO */
7     }
8     public void addProperty(String key, String value) {
9         /* TODO */
10    }
11    public List<String> keys() {
12        /* TODO */
13    }
14    public List<String> values() {
15        /* TODO */
16    }
17    public ????? allProperties() {
18        /* TODO */
19    }
```

Exercice 6

Vous devez réaliser une implémentation d'une liste dynamique d'entiers. Votre implémentation, appelée `ArrayListInt` doit **impérativement** utiliser un tableau statique d'entiers pour simuler le comportement d'une telle liste.

Voici une utilisation possible :

```
1 public class exo6 {
2     public static void main(String[] args) {
3         ListInt list = new ArrayListInt();
4         list.insert(0);
5         list.insert(3);
6         list.insertAll(2,1); // insertAll prend un nombre arbitraire d'éléments
7         System.out.println( "Size: " + list.size() );
8         for (int i = 0; i < list.size(); i+=1) {
9             int v = list.get(i);
10            System.out.println("Value: " + v);
11        }
12        list.clear();
13        System.out.println( "Size: " + list.size() );
14        /*
15         * Cet exemple afficherait
16         * Size: 4
17         * Value: 0
18         * Value: 3
19         * Value: 2
20         * Value: 1
21         * Size: 0
22         */
23    }
24 }
```

Vous êtes libre d'utiliser un tableau de type `int` primitif (`int[]`) ou un tableau d'objets `Integer` (`Integer[]`).

Contraintes supplémentaires

- Implémentez tous les composants pour que le code ci-dessus compile et s'exécute correctement
 - Votre implémentation doit respecter le contrat de `ListInt`
 - Les méthodes `isEmpty` et `addAll` doivent être **concrètes** dans `ListInt`
 - L'implémentation de `ArrayListInt` réserve initialement une taille de tableau de 10 éléments. A chaque dépassement de capacité, vous devez allouer 10 éléments supplémentaires.
-
- **Utilisez uniquement ces fonctionnalités** suivantes sur les tableaux statiques:
 - l'attribut `length` qui retourne la taille d'un tableau
 - la notation crochet pour modifier ou extraire une valeur du tableau
 - la boucle de parcours

Exercice 7

Reprenez l'exercice précédent sur les listes d'entiers et réalisez votre propre itérateur sur celle-ci.