Formation Orienté Objet et Java de base

Atelier 9 : Les collections, Generics

Encadré par M.BOULCHAHOUB

Objectif de l'atelier

L'objectif de cet atelier est de manipuler les différentes collections offertes par Java.

.

Vous allez apprendre à manipuler :

- *Les différents types de collections (List, Map, Set…)*
- *Les collections typées (Generics)*

List et LinkedList

Pour comprendre l'implémentation LinkedList de l'interface List, nous allons créer une classe LinkedListTest.java contenant la méthode main ().

```
package ma.test;

import java.util.LinkedList;
import java.util.List;

public class CollectionTest {

public static void main(String[] args) {
List l = new LinkedList();
l.add(12);
l.add("toto ! !");
l.add(12.20f);

for (int i = 0; i < l.size(); i++)
System.out.println("Élément à l'index " + i + " = " + l.get(i));
}
}
```

1. *Constater que les objets de l sont de différents types*
2. *Essayer d'afficher le contenu de la liste via l'interface Iterator*

```
ListIterator li = l.listIterator();
while(li.hasNext())
```

```
System.out.println(li.next());
```

List et ArrayList

Pour comprendre l'implémentation ArrayListt de l'interface List, nous allons créer une classe ArrayListTest contenant la méthode main()

```
public static void main(String[] args) {

 ArrayList al = new ArrayList();
 al.add(12);
 al.add("Une chaîne de caractères !");
 al.add(12.20f);
 al.add('d');

 for(int i = 0; i < al.size(); i++)
 {
  System.out.println("donnée à l'indice " + i + " = " + al.get(i));
} }
```

3. *Essayer d'utiliser la méthode add() de l'objet a1*
4. *Essayer d'utiliser la méthode get(int index) de l'objet a1*
5. *Essayer d'utiliser la méthode remove(int index) de l'objet a1*
6. *Essayer d'utiliser la méthode isEmpty() de l'objet a1*
7. *Essayer d'utiliser la méthode removeAll() de l'objet a1*
8. *Essayer d'utiliser la méthode contains(Object element) de l'objet a1*

HashTable

Nous allons maintenant créer une classe HashTableTest contenant la méthode main().

```
public static void main(String[] args) {

  Hashtable ht = new Hashtable();
  ht.put(1, "printemps");
  ht.put(10, "été");
  ht.put(12, "automne");
ht.put(45, "hiver");
  Enumeration e = ht.elements();
       while(e.hasMoreElements())                    System.out.println(e.nextElement());

}
```

9. *Essayer les méthodes suivantes :*
   a. *contains(Object value),*
   b. *isEmpty(),*
   c. *containsKey(int key),*
   d. *put(int key, Object value),*

e. *elements(),*
f. *keys()*

Set et HashSet

Nous allons maintenant créer une classe HashSetTest contenant la méthode main()

```
    public static void main(String[] args) {

        HashSet hs = new HashSet();
        hs.add("toto");
        hs.add(12);
        hs.add('d');

        Iterator it = hs.iterator();
        while(it.hasNext())
                System.out.println(it.next());

        System.out.println("\nParcours avec un tableau d'objet");
        System.out.println("-----------------------------------");

        Object[] obj = hs.toArray();
        for(Object o : obj)
                System.out.println(o);
}
```

10. *Créer une classe ArraySetTest contenant la méthode main()*
11. *Utiliser la méthode Add() : ajoute un élément.*
12. *Utiliser la méthode contains(Object value) : retourne "vrai" si l'objet contient value.*
13. *Utiliser la méthode isEmpty() : retourne "true" si l'objet est vide.*
14. *Utiliser la méthode iterator(): renvoie un objet de type Iterator.*
15. *Utiliser la méthode remove(Object o) : retire l'objet o de la collection.*
16. *Utiliser la méthode toArray() : retourne un tableau d'Object.*

Map et HashMap

Pour comprendre la l'interface Map et l'implémentation HashMap, nous allons créer ensemble une classe de test MapTest contenant la méthode main(), Le but est d'insérer des données dans une HashMap et de les parcourir.

Exercice

Récupérer les arguments d'un programme (le tableau (String[] args) de la méthode main) dans tous les types de collection précédents et les afficher sur la console.

Introduction aux Generics

Le plus simple pour comprendre les generics est d'analyser un exemple type :

```
public static void main(String[] args) {

	List integerList = new ArrayList();
	integerList.add(new Integer(1));
	Integer i = (Integer)integerList.get(0);

  }
```

Dans cet exemple, nous remarquons que nous sommes obligés de transtyper (Cast) explicitement l'objet que l'on récupère de la collection, car la seule chose dont nous sommes sûrs, c'est que l'itérateur retourne un Object.

```
public static void main(String[] args) {

	List integerList = new ArrayList();
	integerList.add(new Integer(1));
	Integer i = (Integer)integerList.get(0);

	integerList.add("exemple"); // (1)

  }
```

Une telle construction, bien que fausse, ne déclenchera aucune erreur lors de la compilation. C'est seulement à l'utilisation que le programme lèvera une exception de type java.lang.ClassCastException.

Il est possible, via une syntaxe particulière, de spécifier explicitement et avant construction, quel sera le type des objets contenus dans la collection :

```
public static void main(String[] args) {
	List<Integer> integerList = new ArrayList<Integer>();
	integerList.add(new Integer(1));
	Integer i = integerList.get(0);
  }
```

La syntaxe <Integer> spécifie que le type des objets utilisés avec cette collection est Integer.

Ainsi, l'exemple précédent (1) fera échouer la compilation car les generics introduisent une vérification de type statique, c'est-à-dire durant la compilation.

Avantages des generics

Le premier avantage des generics réside dans la suppression du contrôle de type à l'exécution. En effet, étant donné que les éventuelles erreurs de transtypage sont levées à la compilation, il n'est plus nécessaire de contrôler le type à l'exécution. Ainsi, le code gagne en sécurité et le temps de maintenance éventuel en est très fortement réduit.

Un avantage visible directement est une meilleure lisibilité et une plus grande robustesse du code.

Syntaxe des generics en Java

Nous allons fixer dans ce chapitre un certain nombre d'appellations et de notations.

Comme nous l'avons déjà vu dans le premier exemple, les premiers symboles nouveaux avec les types génériques sont les chevrons < et >.
Ils permettent de définir les paramètres de types formels qui peuvent être utilisés dans toutes les déclarations génériques, et en particulier en lieu et place des types ordinaires (bien qu'il y ait d'importantes restrictions).
A l'invocation de l'objet, toutes les occurrences du paramètre de type formel sont remplacées par le type défini.

Ainsi, l'interface List est définie de la sorte :

```
public interface List<T>
{
  void add(T t);
  Iterator<T> iterator();
}
```

T est ici le paramètre de type formel de l'interface List.

D'autre part, une convention de nommage semble être acceptée par la communauté pour la notation des paramètres de type formel. Elle recommande l'utilisation d'un caractère unique en majuscule comme E (Element) ou T (Type).

L'autre symbole important pour les generics est le point d'interrogation (?), que l'on appelle aussi wildcard ou inconnue.

Ainsi nous aurions pu écrire :

```
List<?> unKnownList = new ArrayList<String>();
```

Il permet de définir la variance d'un type générique.
Il permet de spécifier que n'importe quel type peut convenir comme élément de la liste.