UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE
FACULTÉ D'INFORMATIQUE

Cours
Programmation Orientée Objet
Pour
ISIL, L2 A et B

Chap 05:
Gestion des collections

MEKAHLIA Fatma Zohra LAKRID
Maître de Conférences Classe B

Laboratoire de Modélisation, Vérification et Evaluation des Performances des systèmes complexes (MOVEP)
Bureau 123, Département SIQ

Novembre 2022

INTRODUCTION

- Le package **java.util** contient plein de classes pour la gestion de structures de données plus évoluées : *listes* , *ensembles*, *arbres*, *vecteurs*, *files*, *piles*.

- Il s'agit des Collections!

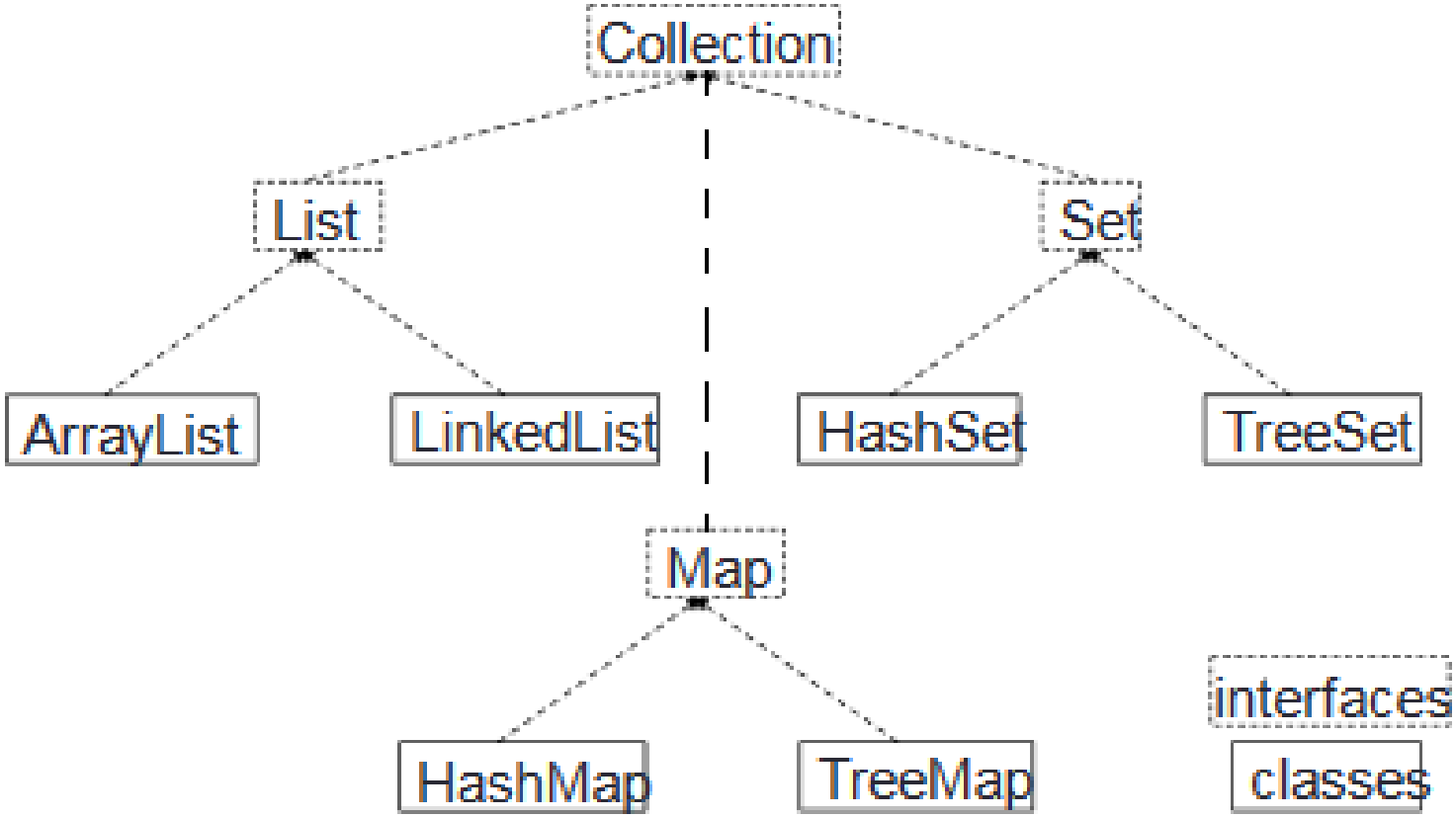
- Une collection de données est un conteneur d'éléments qui possède un protocole particulier pour l'ajout, le retrait et la recherche d'éléments.

INTRODUCTION

- C'est une manière de représenter une structure de donnée en Java consiste à regrouper dans une interface l'ensemble des opérations (noms) applicables sur l'ensemble d'objets de la structure (ajout, suppression, effacement, etc.).

- Toutes les collections en JAVA implémentent l'interface Collection par le biais de sous interfaces comme Set, Map ou List.

INTRODUCTION



INTRODUCTION

- Les classes collection sont définies dans le package java.util à partir de deux Interfaces Java:

- Collection: groupe d'objets, connu par ses éléments.

- Map: groupe d'objets indexées par des clés (eg. Entrées d'un dictionnaire).

Les classes collection

- Les classes collection (qui implémentent l'interface Collection) sont nombreuses dans l'API Java: AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, etc.
- Nous verrons les classes collection suivantes:
  - ArrayList (Vector),
  - LinkedList,
  - HashSet ,
  - TreeSet.

Les classes Map

- Les classes Map (qui implémentent l'interface Map) sont nombreuses dans l'API Java: AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, etc.
- Nous verrons la classe Map suivante:

  - HashMap

L'Interface Collection

- L'interface **Collection<E>** définit la notion de collection d'objets d'une façon assez générale.

- **Les opérations sont :**

- Obtenir le nombre d'éléments de la collection,
- Rechercher un objet donné,
- Ajouter un objet,
- supprimer un objet,
- … etc.

L'Interface List

- Interface pour des objets qui autorisent des doublons et un accès direct à un élément.

- **Deux implémentations possibles:**
  - **ArrayList :** Liste implantée dans un tableau.
  - **LinkedList:** Liste doublement chaînée.

- **Quelques méthodes de ArrayList:**

L'Interface List

- **add(Object element)** permet d'ajouter un élément ;
- **add(Object element, int pos)** permet d'ajouter un élément à la position pos ;
- **get(int index)** retourne l'élément à l'indice demandé.
- **remove(int index)** efface l'entrée à l'indice demandé.
- **isEmpty()** renvoie « vrai » si l'objet est vide ;
- **size()** retourne la taille de l'ArrayList;
- **contains(Object element)** retourne « vrai » si l'élément passé en paramètre est dans l'ArrayList.

L'OBJET ARRAYLIST

- Un **ArrayList** n'a pas de taille limite, et en plus, ils acceptent n'importe quel type de données ! **null** y compris !

- Dans un **ArrayList**, nous pouvons mettre tout ce que nous voulons. Vous devez par contre importer la classe **ArrayList**.

L'OBJET ARRAYLIST

```java
import java.util.ArrayList;

public class Test {

	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));
		}
	}
}
```

```
donnée à l'indice 0 = 12
donnée à l'indice 1 = Une chaîne de caractère !
donnée à l'indice 2 = 12.2
donnée à l'indice 3 = d
```

L'OBJET LINKEDLIST

- Une liste chaînée est une liste dont chaque élément est relié au suivant par une référence à ce dernier, sa taille n'est pas fixe : on peut ajouter et enlever des éléments selon nos besoins.

- Les LinkedList acceptent tout type d'objet.

- Chaque élément contient une référence sur l'élément suivant sauf pour le dernier : son suivant est en fait **null**.

- Vous devez importer la classe LinkedList .

L'OBJET LINKEDLIST

```
 5 public class Test {
 6
 7         public static void main(String[] args) {
 8
 9                 List l = new LinkedList();
10                 l.add(12);
11                 l.add("toto ! !");
12                 l.add(12.20f);
13
14
15                 for(int i = 0; i < l.size(); i++)
16                         System.out.println("Élément à l'index " + i + " = " + l.get(i));
17
18         }
19 }
```

•Vous pouvez implémenter l'interface **Iterator**. Ceci signifie que nous pouvons utiliser cette interface pour lister notre **LinkedList**.

L'OBJET LINKEDLIST

```
 5 public class Test {
 6
 7         public static void main(String[] args) {
 8
 9                 List l = new LinkedList();
10                 l.add(12);
11                 l.add("toto ! !");
12                 l.add(12.20f);
13
14
15                 for(int i = 0; i < l.size(); i++)
16                         System.out.println("Élément à l'index " + i + " = " + l.get(i));
17
18
19                 System.out.println("\n \tParcours avec un itérateur ");
20                 System.out.println("-----------------------------------");
21                 ListIterator li = l.listIterator();
22
23                 while(li.hasNext())
24                         System.out.println(li.next());
25         }
26 }
```

NB: Vu que les éléments ont une référence à leur élément suivant, ce type de listes peut être particulièrement lourd lorsqu'elles deviennent volumineuses !

L'OBJET ARRAYLIST VS LINKEDLIST

- On utilise le plus souvent **ArrayList <E>** si l'ajout et l'accès sont direct (indicé).

- Mais, **LinkedList<E>** est utile s'il y a beaucoup d'opérations d'insertions / suppressions afin d'éviter les décalages.

L'INTERFACE SET

- Eléments non dupliqués

- **Deux implémentations possibles:**
  - **HashSet :** table de hashage (très utilisée).
  - **TreeSet :** arbre binaire de recherche.

- **Quelques méthodes de HashSet:**

L'INTERFACE SET

- **add(Object element)** ajoute un élément.
- **contains(Object element)** retourne « vrai » si l'objet contient element.
- **isEmpty()** retourne « vrai » si l'objet est vide.
- **iterator()** renvoie un objet de type Iterator.
- **remove(Object element)** retire l'objet element de la collection ;
- **toArray()** retourne un tableau d'Object.

L'OBJET HASHSET

- Un Set est une collection qui n'accepte pas les doublons. Elle n'accepte qu'une seule fois la valeur **null**, car deux fois cette valeur est considérée comme un doublon.

- On peut dire que cet objet n'a que des éléments différents.

- On peut parcourir ce type de collection avec un objet **Iterator** où, cet objet peut retourner un tableau d'**Object**.

L'OBJET HASHSET

```
 4   public class Test {
 5
 6           public static void main(String[] args) {
 7
 8                   HashSet hs = new HashSet();
 9                   hs.add("toto");
10                   hs.add(12);
11                   hs.add('d');
12
13                   Iterator it = hs.iterator();
14                   while(it.hasNext())
15                           System.out.println(it.next());
16
17                   System.out.println("\nParcours avec un tableau d'objet");
18                   System.out.println("----------------------------------");
19
20                   Object[] obj = hs.toArray();
21                   for(Object o : obj)
22                           System.out.println(o);
23
24           }
25   }
```

L'OBJET HASHSET

```
10    public static void main(String[] args) {
11        HashSet<String> hset = new HashSet<String>();
12        hset.add("h1");
13        hset.add("h2");
14        hset.add("h3");
15
16        System.out.println("Boucle for avancée");
17        for(String s : hset)
18          System.out.println(s);
19
20        System.out.println("Boucle While+Iterator");
21        Iterator it = hset.iterator();
22        while(it.hasNext())
23        System.out.println(it.next());
24
25        System.out.println("Boucle While+Ennumération");
26        // récupérer l'objet Ennumeration
27        Enumeration enumeration = Collections.enumeration(hset);
28        // lire à travers les éléments de HashSet
29        while(enumeration.hasMoreElements())
30           System.out.println(enumeration.nextElement());}}
```

L'OBJET HASHSET

```
Boucle for avancée
h1
h2
h3
Boucle While+Iterator
h1
h2
h3
Boucle While+Ennumération
h1
h2
h3
```

Les collections en Java – Map

- Les collections de type Map, tableau associatif ou dictionnaire en Java, sont définies à partir de la racine Interface Map <K, V> (et non Collection <E>).

- La raison est qu'une telle collection est un ensemble de paires d'objets, chaque paire associant un objet de l'ensemble de départ K à un objet de l'ensemble d'arrivée V ; on parle de paires (clé, valeur)

HashMap: Exercice

- on utilise **HashMap** pour simuler un répertoire dans lequel le numéro de téléphone est la clé et le nom du propriétaire est la valeur.

- Les clés ne sont jamais dupliquées.

HashMap: correction

```
1  package collection;
2  import java.util.HashMap;
3  import java.util.Map;
4  import java.util.Set;
5  public class HashMapExp {
6
7      public static void main(String[] args) {
8          Map<String, String> repPhonne = new HashMap<String, String>();
9          repPhonne.put("12121212", "Mohamed");
10         repPhonne.put("13131313", "Ali");
11         repPhonne.put("14141414", "Amine");
12         repPhonne.put("15151515", "Tamim");
13         Set<String> numPhonnes = repPhonne.keySet();
14  for (String numPhonne : numPhonnes) {
15     String nom = repPhonne.get(numPhonne);
16     System.out.println("numéro de tél: " + numPhonne + " ==> Name: " + nom);
17         }
18     }
19 }
20
```

Merci