UFR de mathématiques et d'informatique UFR 27
Licence 2ème année MIASHS

Programmation orientée objet - Niveau 2

Khelifi Tesnim

Chapitre 2: Les Collections

LES COLLECTIONS

On appelle **collection**, ou **structure de données** (**abstraite**) ([abstract] data structure), un objet servant de conteneur à d'autres objets.

Exemples :

- les tableaux,
- les listes et leurs variantes : piles, queues,
- les ensembles,
- les tables associatives.

Chaque type de collection a ses caractéristiques propres, ses forces et ses faiblesses. Le choix de la collection à utiliser dans un cas particulier dépend donc de ce qu'on veut en faire.

LES COLLECTIONS

Pour chaque type de collection (liste, ensemble, etc.), l'API Java contient généralement :

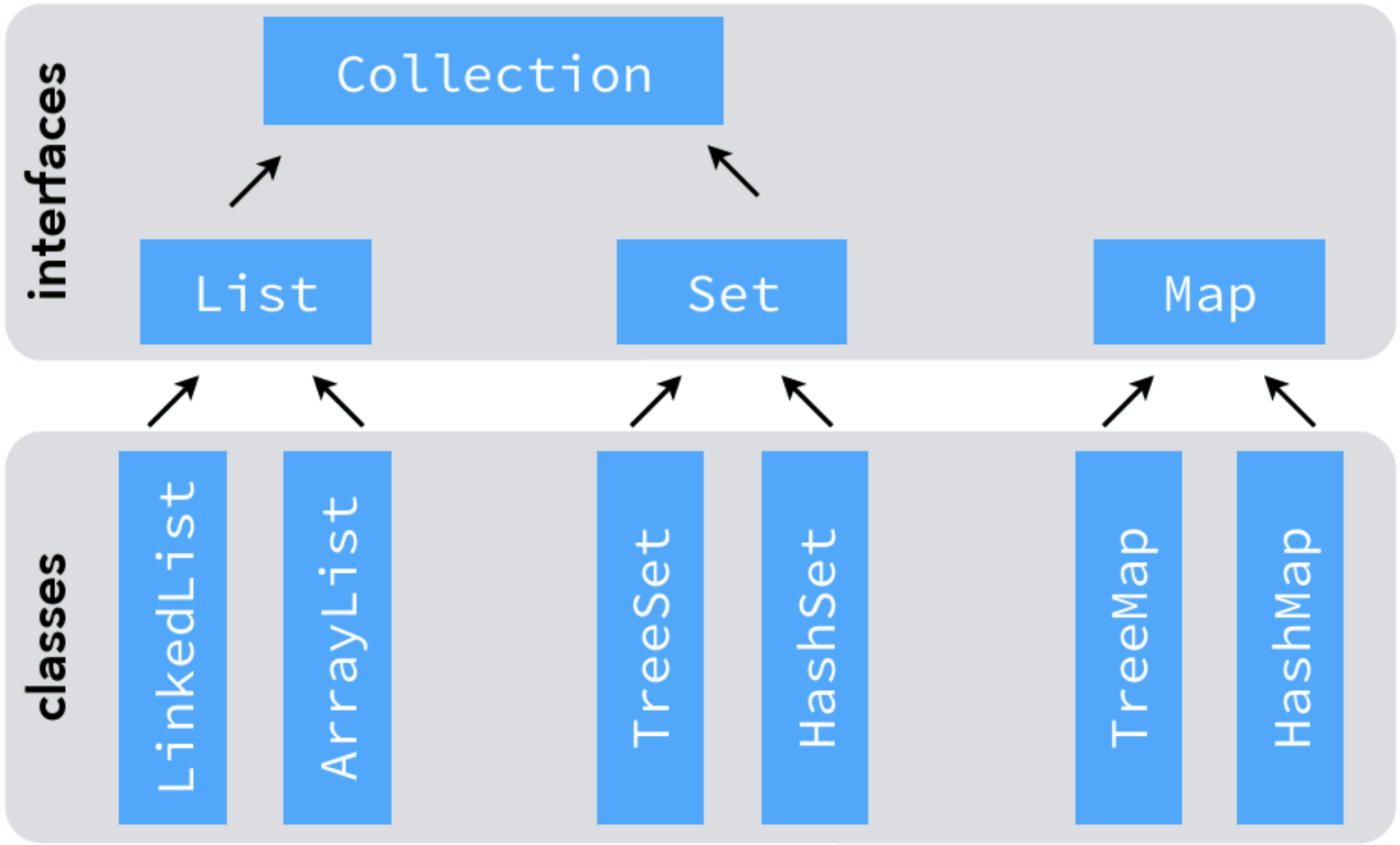
- **Une interface**, qui décrit les opérations offertes par la collection en question.
- **Une classe**, qui implémente l'interface et contient le code commun à toutes les mises en œuvre.
- **Plusieurs sous-classes** concrètes de la classe abstraite, qui sont les mises en œuvre de la collection, ayant chacune leurs propres caractéristiques. Par exemple, pour les listes, l'interface est List, la classe abstraite AbstractList et parmi les mises en œuvre concrètes figurent ArrayList et LinkedList.

LES COLLECTIONS

- Le **Java Collections Framework** fournit un ensemble d'interfaces (telles que **List**, **Set** et **Map**) et un ensemble de classes (**ArrayList**, **HashSet**, **HashMap**, etc.) qui implémentent ces interfaces.

- Toutes font partie du package **java.util**.

- Elles sont utilisées pour stocker, rechercher, trier et organiser plus facilement les données, le tout à l'aide de méthodes et de modèles standardisés.


interfaces
Collection
List
Set
Map
classes
LinkedList
ArrayList
TreeSet
HashSet
TreeMap
HashMap

LES COLLECTIONS

- **Interfaces principales dans le cadre des collections:**

Interface	Classes	Description
List	ArrayList, LinkedList	Collection ordonnée autorisant les doublons
Set	HashSet, TreeSet, LinkedHashSet	Collection d'éléments uniques
Map	HashMap, TreeMap, LinkedHashMap	Stocke des paires clé-valeur avec des clés uniques.

LES COLLECTIONS

- **Exemples de classes:**

Interface	Classe	Description
List	ArrayList	Tableau redimensionnable qui conserve l'ordre et autorise les doublons
	LinkedList	Liste avec opérations d'insertion et de suppression rapides
Set	HashSet	Collection non ordonnée d'éléments uniques
	TreeSet	Ensemble trié d'éléments uniques (ordre naturel)
	LinkedHashSet	Conserve l'ordre dans lequel les éléments ont été insérés.
Map	HashMap	Stocke les paires clé/valeur sans ordre particulier.
	TreeMap	Carte triée selon l'ordre naturel des clés
	LinkedHashMap	Conserve l'ordre dans lequel les clés ont été insérées

LES COLLECTIONS

- En plus des méthodes définies dans les interfaces des différentes collections, on trouve des méthodes statiques relatives aux collections dans les classes **Collections** et **Arrays**.
- Ces deux classes ont pour seul but de regrouper des méthodes statiques, et ne sont donc pas instanciables — leur constructeur est privé.
- Leurs principales méthodes seront présentées en même temps que les collections concernées.
- **Attention à ne pas confondre l'interface Collection et la classe Collections !**

Pourquoi ces classes ne sont pas instanciables?

Les classes Collections et Arrays :

- ne contiennent que des méthodes statiques
- n'ont aucun état interne
- ont un constructeur privé

```
new Collections(); // impossible
new Arrays();      // impossible
```

→ Elles ne servent pas à créer des objets, seulement à fournir des fonctions utilitaires.

LES COLLECTIONS

Quelques exemples concrets :

- `Collections.sort(list)` → trier une liste
- `Collections.reverse(list)` → inverser l'ordre
- `Collections.unmodifiableList(list)` → rendre une liste non modifiable
- `Arrays.sort(array)` → trier un tableau
- `Arrays.asList(array)` → transformer un tableau en liste

→ Ces méthodes **complètent** les fonctionnalités définies dans les interfaces de collections.

L'INTERFACE COLLECTION

- L'interface Collection sert de **super-interface** commune aux **listes** (interface List) et à leurs variantes, ainsi qu'aux **ensembles** (interface Set).
- Comme nous l'avons vu, les listes sont ordonnées mais les ensembles ne le sont pas.
- Dès lors, seules les méthodes qui ne dépendent pas d'une notion d'ordre sont définies dans l'interface Collection.
- Par exemple, Collection ne contient pas de méthode pour insérer un élément à une position donnée, puisque cette opération n'a un sens que si ceux-ci sont ordonnés.
- Une telle méthode existe par contre dans l'interface List.

L'INTERFACE COLLECTION

- L'interface Collection représente une collection dont on ne sait pas si elle est ordonnée ou non.
- Elle est bien entendu générique, son paramètre de type représentant le type des éléments de la collection :

```
public interface Collection<E>
{
// ... méthodes
}
```

- Les méthodes les plus importantes sont présentées ci-après, parfois avec un type simplifié pour des raisons pédagogiques.

L'INTERFACE COLLECTION

Méthodes de consultation:

Les méthodes ci-dessous, comme toutes celles qui ne modifient pas la collection, sont obligatoires :

- **boolean isEmpty()** : retourne vrai ssi la collection est vide.
- **int size()** : retourne le nombre d'éléments contenus dans la collection.
- **boolean contains(Object e)** : retourne vrai ssi la collection contient l'élément donné. Le type de l'argument est malheureusement Object et non pas E pour des raisons historiques.
- **boolean containsAll(Collection<E> c)** : retourne vrai ssi la collection contient tous les éléments de la collection donnée.

L'INTERFACE COLLECTION

Méthodes d'ajout:

Les méthodes d'ajout ci-dessous, comme toutes les méthodes de modification, sont optionnelles (c-à-d qu'elles peuvent lever l'exception UnsupportedOperation...).

- **boolean add(E e)** : ajoute l'élément donné à la collection.
- **boolean addAll(Collection<E> c)** : ajoute à la collection tous les éléments de la collection donnée.

La valeur de retour de ces méthodes indique si le contenu de la collection a changé suite à l'ajout. Si la collection est une liste, cela est toujours le cas, mais si la collection est un ensemble — qui n'admet pas de doublons — ce n'est pas forcément le cas.

Méthodes de suppression:

Les méthodes de suppression ci-dessous sont optionnelles :

- **void clear()** : supprime tous les éléments de la collection,
- **boolean remove(E e)** : supprime l'élément donné, s'il se trouve dans la collection,
- **boolean removeAll(Collection<E> c)** : supprime tous les éléments de la collection donnée,
- **boolean retainAll(Collection<E> c)** : supprime tous les éléments qui ne se trouvent pas dans la collection donnée.

Les trois dernières méthodes retournent vrai ssi le contenu de la collection a changé.

COLLECTION 1: LA LISTE

- Une liste (list) est une séquence ordonnée et dynamique (donc à taille variable) d'objets.
- Les listes sont très similaires aux tableaux, au point que la différence entre les deux est souvent floue.
- Toutefois, les tableaux sont généralement de taille fixe et à accès aléatoire tandis que les listes sont de taille variable et à accès séquentiel.
- L'accès aléatoire signifie que l'accès à un élément dont on connaît l'index se fait en O(1), alors que l'accès séquentiel signifie que la même opération se fait en O(n).

COLLECTION 1: LA LISTE

```
int[] notes = new int[3];
notes[0] = 12;
notes[1] = 15;
notes[2] = 9;//
notes[3] = 18 → erreur (hors limites)
```

```
List<Integer> notes = new ArrayList<>();
notes.add(12);
notes.add(15);
notes.add(9);
notes.add(18); // OK, la liste s'agrandit
```

COLLECTION 1: LA LISTE

Critère	Tableau	Liste
Taille	Fixe	Dynamique
Types primitifs	Oui	Non
Ajout / suppression	Difficile	Facile
Méthodes intégrées	Aucune	Nombreuses
Performance brute	Excellente	Bonne
Usage moderne	Rare	Très fréquent

COLLECTION 1: LA LISTE

- Quelques cas particuliers des listes se rencontrent assez fréquemment pour avoir un nom propre, p.ex. : – les piles, – les queues, – les « deques ».
- Souvent, ces cas particuliers peuvent être mis en œuvre de manière plus efficace que les listes dans toute leur généralité.
- Il n'est donc pas rare de trouver des classes modélisant ces cas particuliers des listes dans les bibliothèques.

COLLECTION 1: LA LISTE

Pile	Queue	Deque
Une **pile** fonctionne selon la règle **LIFO** (Last In, First Out) : → **le dernier élément ajouté est le premier retiré**. **Opérations typiques:** push : ajouter un élément au sommet pop : retirer l'élément du sommet peek : consulter le sommet sans le retirer **Exemple concret:** Pile d'assiettes	Une **queue** fonctionne selon la règle **FIFO** (First In, First Out) : → **le premier arrivé est le premier servi**. **Opérations typiques** enqueue : ajouter à la fin dequeue : retirer au début peek : consulter le premier élément **Exemple concret** File d'attente au guichet	Une **deque** est une **queue à double extrémité** : → **on peut ajouter et retirer des éléments au début et à la fin**. **Opérations typiques** addFirst, addLast removeFirst, removeLast **Exemple concret** Historique avant / arrière (navigation)

COLLECTION 1: LA LISTE

- Le concept de liste est représenté dans l'API Java par l'interface List du paquetage java.util.
- Tout comme Collection , dont elle hérite, cette interface est générique et son paramètre de type représentant le type des éléments de la liste :

```
public interface List<E>   extends Collection<E>
{
// ... méthodes
 }
```

- Les principales méthodes que cette interface ajoute à celles de Collection sont présentées ci-après, parfois avec un type simplifié pour des raisons pédagogiques.

Méthodes de consultation:

L'interface List ajoute les trois méthodes suivantes aux méthodes de consultation de Collection :

- **E get(int i)** : retourne l'élément qui se trouve à la position donnée ou lève une exception si celle-ci est invalide,
- **int indexOf(E e)** : retourne la position de la première occurrence de l'élément donné, ou –1 s'il ne se trouve pas dans la liste,
- **int lastIndexOf(E e)** : retourne la position de la dernière occurrence de l'élément donné, ou –1 s'il ne se trouve pas dans la liste.

Notez que, comme dans les tableaux, le premier élément de la liste est à la position 0.

COLLECTION 1: LA LISTE

Méthodes d’ajout:

L'interface List ajoute deux variantes des méthodes d'ajout qui permettent d'ajouter un élément à une position donnée :

- **void add(int i, E e)** : insère l'élément donné à la position donnée de la liste,
- **boolean addAll(int i, Collection<E> c)** : insère tous les éléments de la collection donnée à la position donnée de la liste.

Ces méthodes lèvent une exception si la position donnée est invalide.

Méthodes de modification:

L'interface List ajoute les deux méthodes suivantes aux méthodes de modification / suppression de Collection :

- **E remove(int i) :** supprime et retourne l'élément à la position donnée,
- **E set(int i, E e) :** remplace l'élément à la position donnée par celui donné, et retourne l'ancien élément.

Ces méthodes lèvent une exception si la position donnée est invalide.