Programmation Orientée Objets - JAVA

Erick STATTNER

Maître de Conférences en Informatique

Université des Antilles

erick.stattner@univ-ag.fr

www.erickstattner.com

Chapitre IV. Collections

1. **Introduction**
2. **Les listes**

IV. Collections

1. Introduction

IV. Collections

1) Introduction

Contexte

- Dans de nombreux contexte, nécessité de stocker, manipuler un ensemble d'objets
- Solution vu jusqu'à maintenant: tableaux
- Problème des tableaux:
  - Gestion de la taille
  - Tout doit être implémenté soit même
    Ex. Tableau trié, possibilité d'avoir des éléments similaires, etc;

En JAVA: les collections

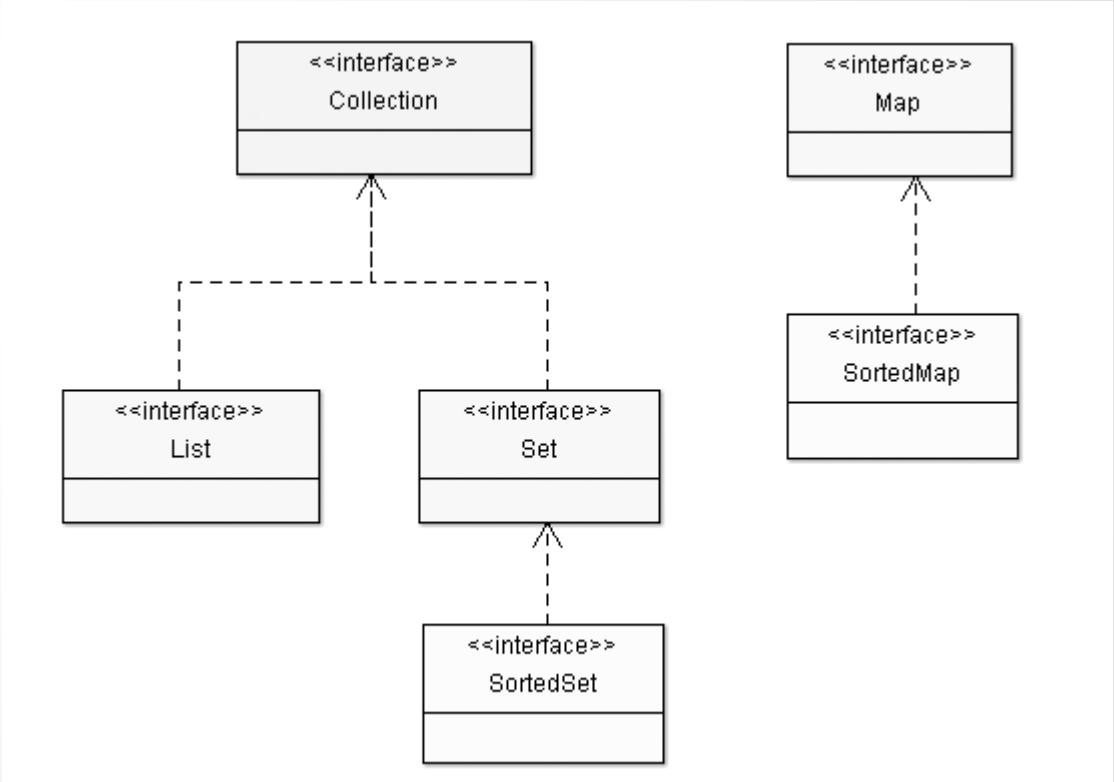
- Interface qui fournit un ensemble de méthode pour la manipulation des listes
- Possibilité de manipuler des ensembles de très haut niveau
- Plusieurs implémentations possédant chacune un comportement et des fonctionnalités particulières.

IV. Collections

1) Introduction

Hiérarchie des classes

- **Collection**
  interface qui est implémentée par la plupart des objets qui gèrent des collections
- **List**
  Collections qui autorisent des doublons et un accès direct à un élément
- **Set**
  collections qui n'autorisent pas de doublons dans l'ensemble
- **Map**
  collections sous la forme clé/valeur



IV. Collections

1) Introduction

Avantages:

- Manipuler des listes de très haut niveau
- Ne pas se soucier des problèmes de stockage, de taille, d'optimisation, etc.
- Plusieurs implémentation selon les besoins

IV. Collections

2. Les listes

IV. Collections

2) Les listes

Les listes

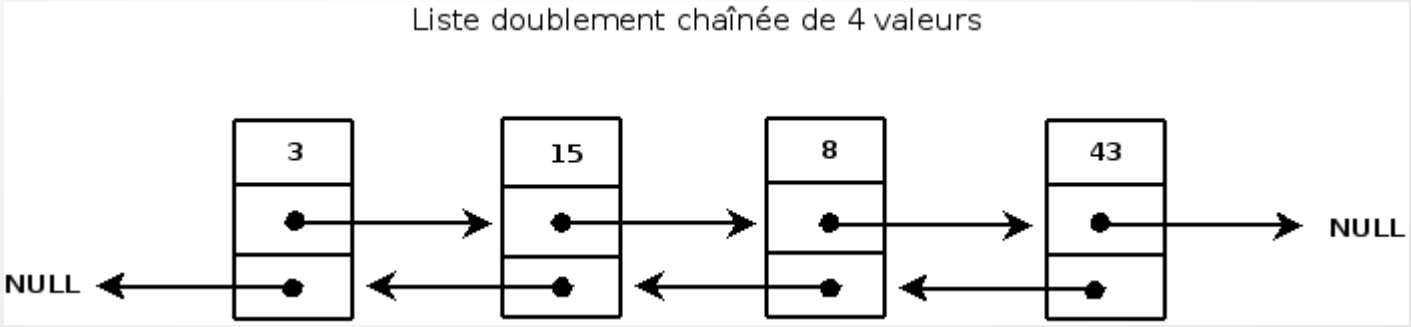
- Liste d'objets extensible à volonté
- Permet de
  - Contenir des doublons
  - Interagir avec un élément en utilisant sa position dans la liste
  - Insérer des éléments **null**
- Plusieurs implémentations disponibles en JAVA **(java.util)**
  - LinkledList
  - ArrayList
  - Vector
  - …

IV. Collections

2) Les listes

LinkedList

- Liste chainée
- Chaque élément est lié au suivant et au précédent par une référence, A l'exception
  - Du premier élément dont la référence vers l'élément suivant vaut **null**
  - Le dernier élément dont la référence vers l'élément précédent vaut **null**



IV. Collections

2) Les listes

Exemple LinkedList

```
LinkedList liste = new LinkedList();
liste.add(12);
liste.add(" toto!! " );
liste.add(new Voiture());
```

Exemple de LinkedList avec parametre générique

```
LinkedList<Vehicule> liste2 = new LinkedList<Voiture>();
liste2.add(new Voiture());
liste2.add(new Avion());
liste2.add(new Bateau());
```

IV. Collections

2) Les listes

Parcours d'une liste

- **Avec l'indice**

```
for(int i = 0; i < liste2.size(); i++){
        System.out.println( liste2.get(i) );
}
```

- **Avec for simplifié**

```
for(Voiture v : liste2){
        System.out.println( v );
}
```

IV. Collections

2) Les listes

Quelques méthodes spécifiques

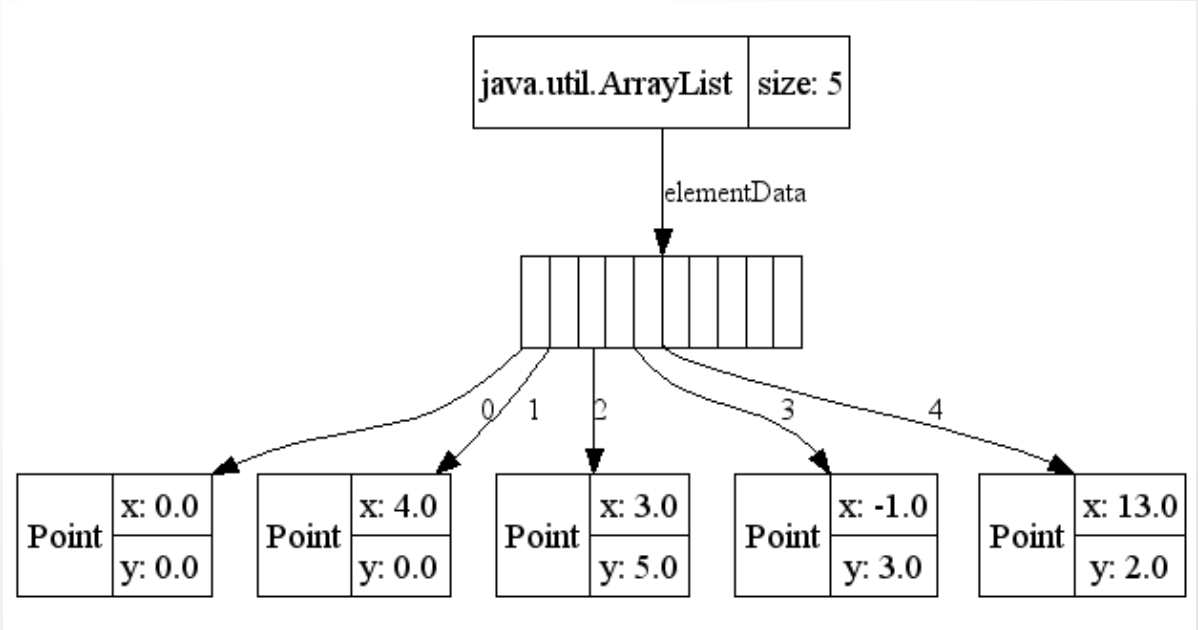
Méthode	Rôle
boolean add(Object)	Ajouter un élément à la fin du tableau
void clear()	Supprimer tous les éléments du tableau
Object get(index)	Renvoyer l'élément du tableau dont la position est précisée
int indexOf(Object)	Renvoyer la position de la première occurrence de l'élément fourni en paramètre
boolean isEmpty()	Indiquer si le tableau est vide
int lastIndexOf(Object)	Renvoyer la position de la dernière occurrence de l'élément fourni en paramètre
Object remove(int)	Supprimer dans le tableau l'élément fourni en paramètre
Object set(int, Object)	Remplacer l'élément à la position indiquée par celui fourni en paramètre
int size()	Renvoyer le nombre d'éléments du tableau

IV. Collections

2) Les listes

ArrayList

- Liste gérée dans un tableau
- Chaque case représente un objet
- Même méthode que pour les LinkedList



IV. Collections

2) Les listes

Exemple ArrayList

```
ArrayList liste = new ArrayList();
liste.add(12);
liste.add(" toto!! " );
liste.add(new Voiture());
```

Exemple de ArrayList avec paramètre générique

```
ArrayList < Vehicule > liste2 = new ArrayList <Voiture>();
liste2.add(new Voiture());
liste2.add(new Avion());
liste2.add(new Bateau());
```

Parcours identiques aux LinkedList !

IV. Collections

2) Les listes

LinkedList VS ArrayList

- **ArrayList**:
  - Efficace en lecture
  - Peu efficace lors de l'ajout/suppression en milieu de liste

- **LinkedList**
  - Efficace pour l'ajout/suppression en milieu de liste
  - Lecture d'un élément particulier lent