JAVA
FONDAMENTAUX

Présenté par :

Xavier TABUTEAU

Les collections

Les tableaux Java vu précédemment permettent de stocker des « collections » de données. Il existe en Java une façon plus simple de gérer ces collections. « Java Collections » désigne un ensemble d'interface et de classes permettant de stocker, trier et traiter les données (classes utiles avec de multiples méthodes).

La plupart des collections se trouvent dans le package java.util

Une collection est un « objet » qui collecte des éléments dans une liste.
Une collection représente un ensemble de données de même type.

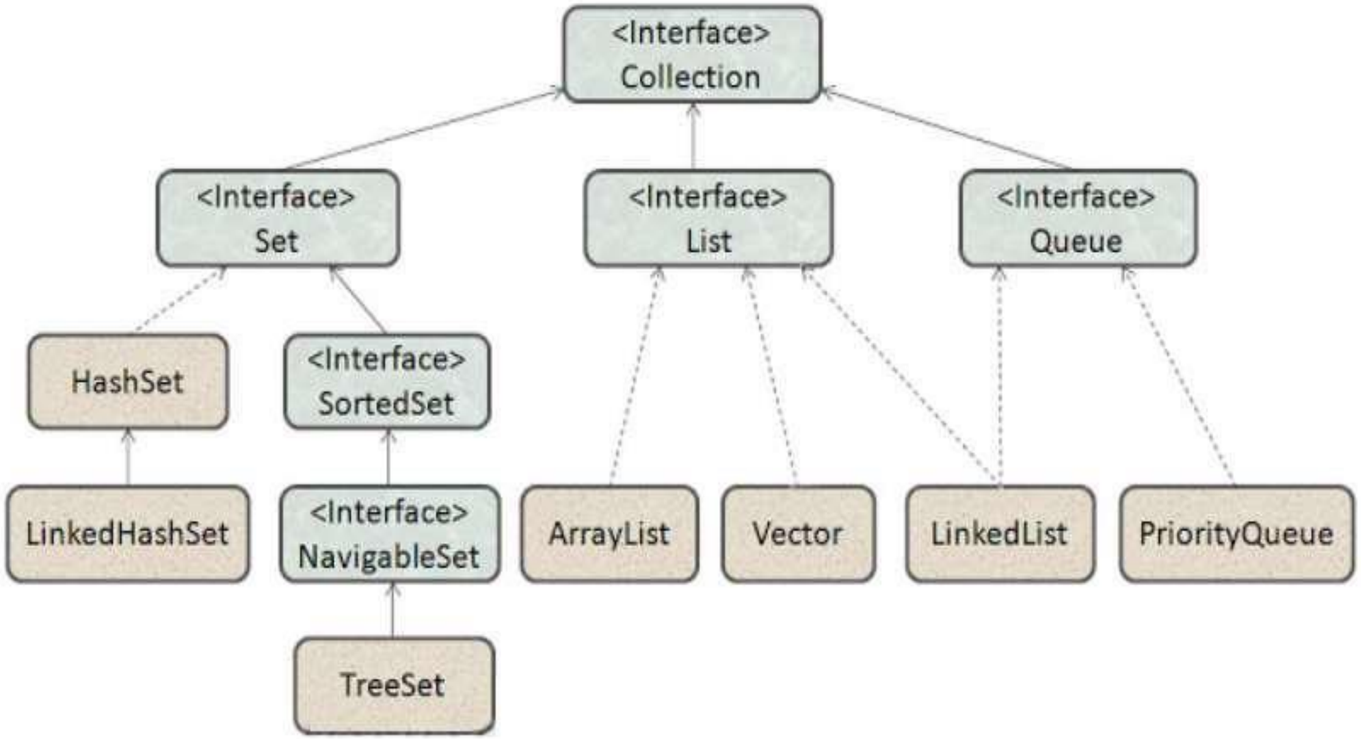
La composition des collections de Java est une structure hiérarchique établie à partir de deux grands ensembles.

- Collection (java.util.Collection)

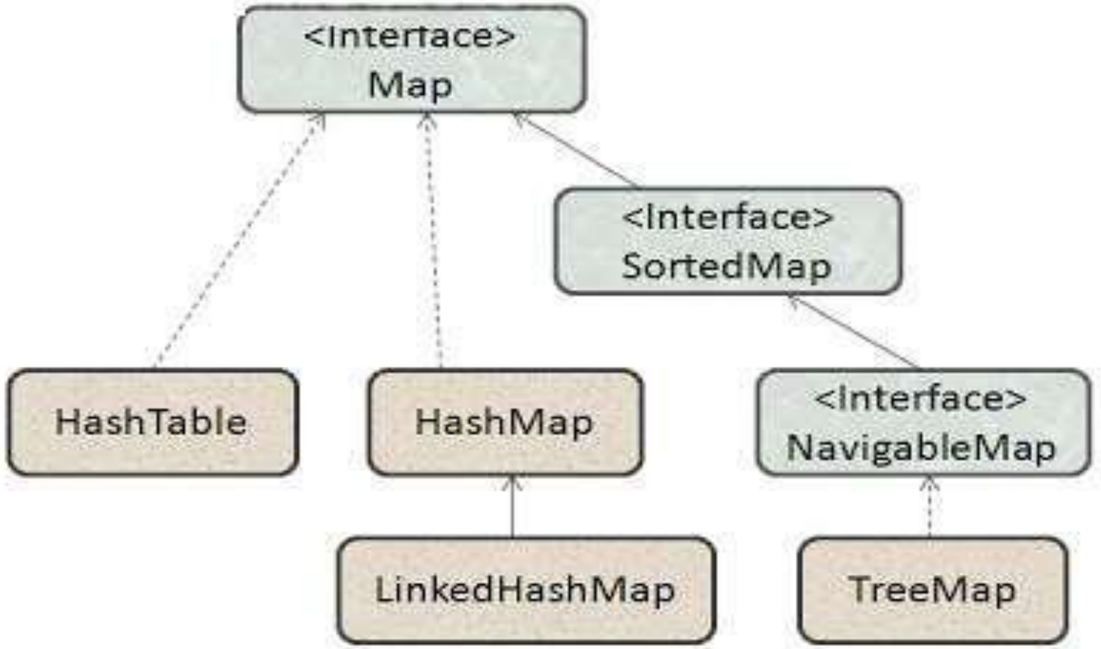
- Map (java.util.Map)

Les collections

Java.util.Collection



Java.util.Map



Le type List en Java

La classe ArrayList implémente l'interface List, qui elle-même hérite de l'interface Collection. Elle dispose donc, de l'ensemble des méthodes de l'interface List et de l'interface Collection. Une ArrayList peut-être vue et traitée comme une interface List.

On peut écrire :

```
List listeDeChaines = new ArrayList(); // Polymorphisme !
```

Il n'y a pas encore le type de donnée à stocker de précisé dans cette liste.
On le précise en utilisant les « Generics » de Java :

```
List<String> listeDeChaines = new ArrayList<>();
```

L'appel d'une méthode d'une liste se fait comme suit :

```
maListe.nomDeLaMethode(parametres);
```

Le type List en Java

Liste de méthodes utiles pour gérer les ArrayList :

add(x, val)	→ charge la valeur val dans la liste à la position x. (si x n'est pas préciser, charge la valeur au premier index disponible).
addAll(liste2)	→ charge la liste2 dans une liste.
get(x)	→ accède à l'élément de la liste ayant l'index x.
set(x, val)	→ modifie l'élément à la position x par ma valeur val.
indexOf(val)	→ obtient l'index du premier élément ayant pour valeur val.
lastIndexOf(val)	→ obtient l'index du dernier élément ayant pour valeur val.
contains(val)	→ vérifie si la valeur val est contenu dans la liste (retourne vrai ou faux).
remove(x)	→ supprime de la liste l'élément situé à l'index x.
remove(val)	→ supprime de la liste le premier élément égale à val.
clear()	→ supprime tous les éléments de la liste.
size()	→ retourne la taille de la liste.

Le type Set en Java

Le type Set en Java est une liste ayant pour particularité de contenir que des éléments uniques). Si on converti une liste de type List en Set les doublons de la List seront supprimés dans le Set. Les collections sont convertible d'un type à l'autre.
Attention : pas de garantie de l'ordre d'insertion !

Le type Map en Java

L'interface Map permet de stocker un ensemble de paires « clé / valeur » dans une table de hachage. Une Map ne peut pas contenir des éléments dupliqués, chaque clé à une unique valeur. Map est implémenté avec les classes suivantes :

HashMap → pas de garantie de l'ordre d'insertion.
LinkedHashMap → garantie l'ordre d'insertion.
TreeMap → stockage des éléments triés selon leur valeur.

L'interface SortedMap hérite de Map et implémente les méthodes pour ordonner les éléments dans l'ordre croissant et décroissant.

Remarque : TreeMap est une implémentation de SortedMap.

Liste de méthodes utiles pour gérer les Map :

entrySet() → retourne un objet Set contenant les paires clé / valeurs du Map.
put(cle, val) → ajout d'un ensemble clé / valeur.
getKey() → obtient la clé de l'entrée en cours.
getValue() → obtient la valeur de l'entrée en cours.
replace(cle, val) → remplace la valeur de l'entrée en cours avec la valeur val.

Le tri d’une collection

Le tri d’une collection se fait par la méthode sort().

Exemple :

Tri simple : Collections.sort(liste);
Tri personnalisé : Collections.sort(liste, methode_implementant_Comparator);

Attention : si le type d’objet utilisé n’implémente pas l’interface Comparator, ou si on souhaite trier selon un ordre différent, alors il faut fournir une implémentation spécifique de l’interface Comparator !

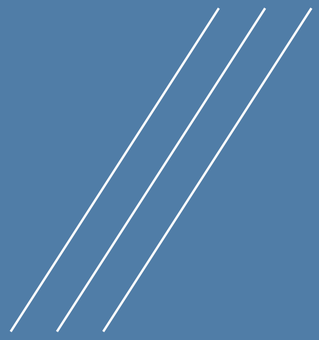
L’itération d’une collection

Il existe plusieurs possibilités d’itérer sur une liste en Java.

- Utilisation de l’interface Iterator
- Utiliser la boucle « intelligente » for (:)
- Utiliser les boucles classiques (for (;;), while, do...while)
- Utiliser la méthode forEach() avec une expression lambda

IterationsCollections.java

JAVA FONDAMENTAUX



Présenté par

Xavier TABUTEAU