# Les collections en Java

# Objectifs

Revoir le principe et le fonctionnement des collections en Java.

Se préparer à l’Etude de cas : les collections (ou tableaux associatif PHP) sont présents chaque année à l’épreuve d’étude de cas avec des questions très répétitives mais très discriminantes (du genre tous les points ou rien du tout).

# Utilité des collections

Les collections en Java permettent de manipuler de manière cohérente et sûre des ensembles d’objet en fonction des besoins « métier » : performance, accès concurrent, redimensionnement, arborescence, hachage …

Les collections Java (JCF : Java Collection Framework) peuvent être décrites simplement comme les meilleures alternatives aux tableaux (array) …

# Documentation & Syntaxe

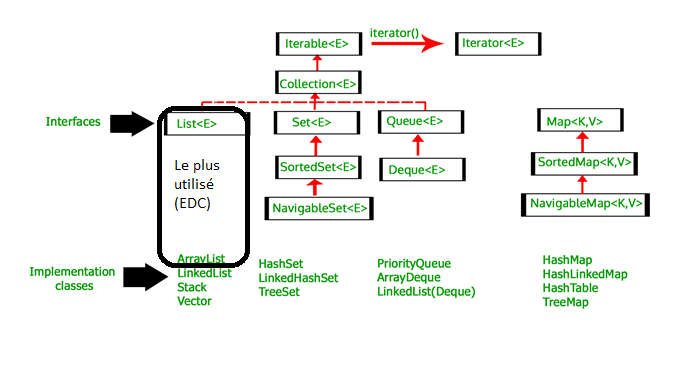
Cf en français : <https://www.jmdoudoux.fr/java/dej/chap-collections.htm> à partir de 12.2 (auparavant obsolète).

Cf en anglais : Big Java, chapitre 15 et 16 (et 18 & 19 pour approfondir).

Dans le Framework JCF, on trouve 2 hiérarchies ***d’interface*** principale, Collection & Maps.

On remarque qu’aucune classe n’implémente directement l’interface Collection, mais plutôt une de ses 3 filles : List, Queue ou Set.

A ce jour, presque 100 % des EDC utilisent l’interface List avec la classe concrète ArrayList (quel ennui !) cf ci-dessous, mais rien ne dit qu’une prochaine EDC n’utiliserait pas un Set / HashSet ou Map / HashMap par exemple, avec la documentation qui va bien.



**Principales interfaces et classes concrètes selon les fonctionnalités**



Le couple List / ArrayList est le plus connu et utilisé (en étude de cas).

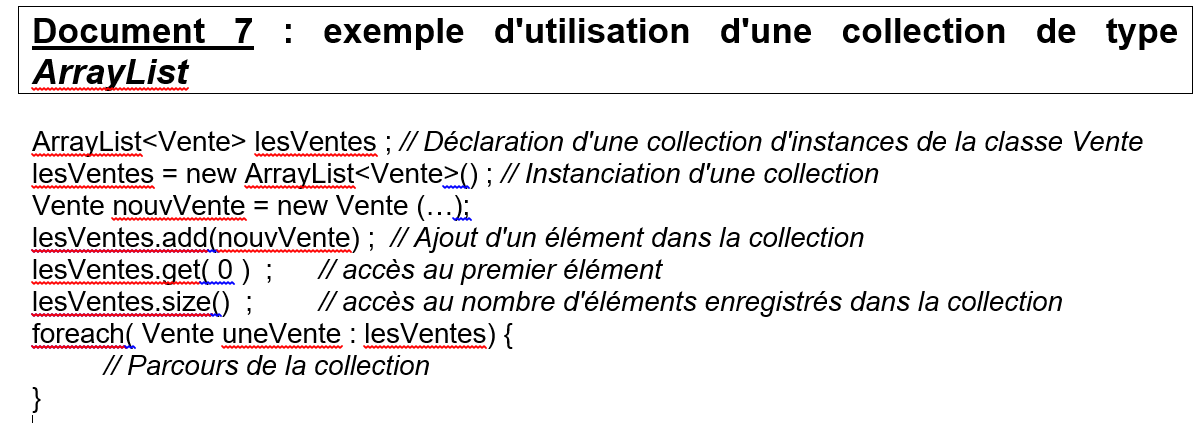
Une Map contient un ensemble de couples clef, valeur (similaire à tableau associatif PHP). Elle n’autorise pas de doublon sur les clefs, mais en accepte sur les valeurs.

Une Map ne peut pas être traversées (parcourue) : si cela est nécessaire, on utilise la méthode entrySet() qui renvoie un Set de toutes les clefs (keys).

# Principes utilisation Etude de cas

On lit attentivement la documentation associée à la Collection, dans une annexe séparée, pour ensuite utiliser les bonnes méthodes dans les différentes missions.

Exemple : document 7 du sujet 2018 WebCaisse.



# Principes utilisation en contexte

Après une étude attentive des besoins métier, des classes, attributs etc, le typage des attributs de type collections dépendra des besoins métier et contrainte technique.

Exemple : si j’ai un attribut equipage dans la classe Navire qui est une collection de Marin, quelle collection utiliser ? List / ArrayList, Map / HashMap, Set / HashSet, plusieurs solutions sont possibles.

Ci-dessous les principales méthodes pour la classe Arraylist : <https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html>



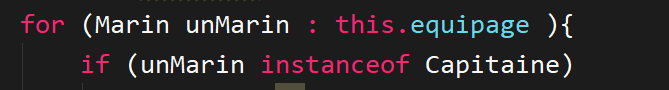
# Parcourir des Collections

Pour les Collections dérivant de l’interface Collection, 2 grandes possibilités :

## Boucle for (équivalent foreach)

C’est la boucle la plus simple quand **on n’a pas besoin de modifier ou supprimer un élément** de la Collection.

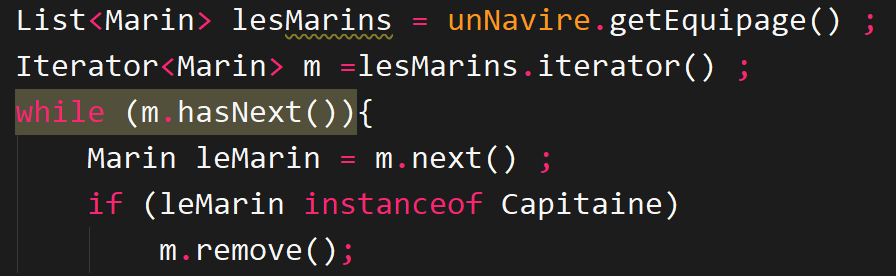
Attention la syntaxe est inversée par rapport au PHP ( for( élément : collection) en Java et foreach( $collection as $element) en PHP).



## Utilisation d’un Iterator

Boucle plus complexe à utiliser mais permet accès en modification et suppression des éléments.

Toujours faire un iterator.next() avant de modifier ou supprimer l’élément.



## Création d’un Iterator ou ListIterator pour une classe implémentant une interface dérivant de Collection

On trouve 2 interfaces **Iterator** : Iterator et ListIterator (qui dérive de Iterator). Iterator est unidirectionnel tandis que ListIterator est bidirectionnel.

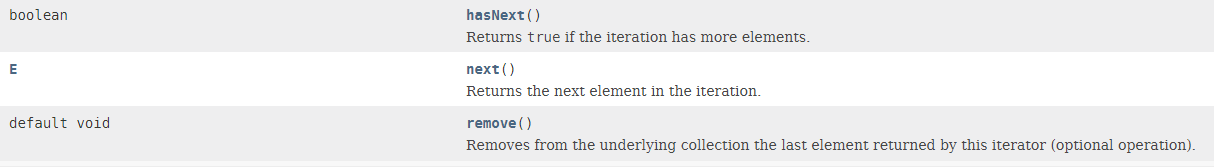
Un Iterator est généré par la **méthode iterator()** .

Chaque ***classe concrète*** , par exemple ArrayList, HashSet, TreeSet … liée à l’interface Collection implémente également l’interface **Iterable** qui contient la méthode ***iterator()*** à implémenter dont la signature est la suivante :



**Très complexe** (bien écarquiller les yeux) : l’Iterator renvoyé par la méthode iterator() est une classe inner (interne c’est-à-dire à l’intérieur) de la classe concrète qui implémente l’interface Iterator et donc ses 3 méthodes hasNext(), next() et remove, cf ci-dessous.

L’interface Iterator définit 3 méthodes (implémentées dans les classes inner des classes concrètes) : <https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html>



Cf exemple précédent où les 3 sont utilisées : hasNext() pour la boucle while, next() pour récupérer l’élément suivant et remove() pour supprimer l’élément.

# Parcourir des Maps

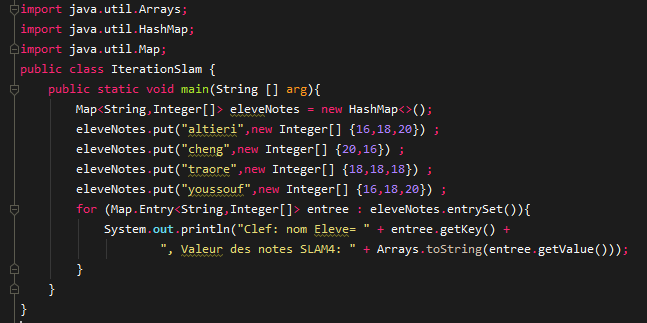
Il y a 5 façons d’itérer sur des éléments d’une Map, même si aucune n’est directe, comme indiqué plus haut (une Map n’étant pas une Collection et ne pouvant être traversée directement).

Cet excellent article <https://www.geeksforgeeks.org/iterate-map-java/> les passe en revue en détail.

Une méthode est obsolète (itération sur les clefs via keySet() puis récupération valeur correspondante à chaque clef), une est très moderne mais nécessite la compréhension des lambdas expression.

La plus commune est avec la fonction entrySet et a sa correspondance avec un Iterator (si besoin d’enlever des valeurs).

Cf exemple fourni : **IterationSlam.java**



Etape 1 : déclaration de la Map.



Comme d’hab on déclare de la forme Interface = Classe Concrète 🡪 polymorphe on pourra substituer par la suite une autre classe concrète qui implémente la même interface.

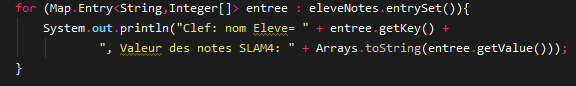
Chaque entrée de notre Map est constituée d’un couple clef, valeur. Dans notre exemple la clef est de type String et la valeur un tableau d’Integer.

Etape 2 : on remplit notre HashMap



Attention : si on utilise plusieurs fois la même clef, la valeur sera écrasée car la clef doit être unique dans une Map (mais pas les valeurs).

Etape 3 : on affiche le contenu en parcourant la Map



Extrait doc java de la classe HashMap :



La méthode entrySet() renvoie un ensemble (Set) de Map.Entry qui est une interface permettant d’avoir une vision Collection de la Map … ! 🡪 on peut donc itérer dessus et chaque instance d’itération, nommée entree ici, est une Map.Entry

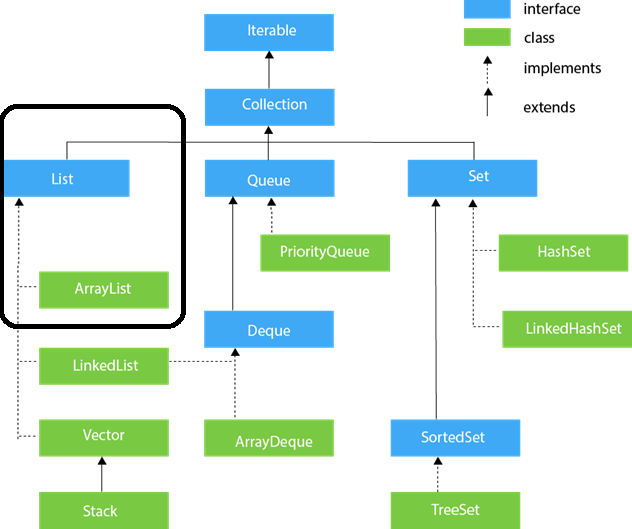
La méthode getKey() de l’interface Map.Entry <https://docs.oracle.com/javase/8/docs/api/java/util/Map.Entry.html> renvoie la clef et la méthode getValue() la valeur.

Comme la valeur est un tableau on utilise la méthode statique toString() de la classe Arrays pour l’afficher.

# Annexes

Schémas plus détaillés de JCF

## Collection



## Map

Java Map Hierarchy