Java - 1ère année

Paquetages de collections

Cours 8

Introduction

• Une collection représente un groupe d'objets, connu par ses éléments.
• Certaines collections acceptent les doublons, d'autres pas.
• Certaines sont ordonnées, d'autres pas.
• Certaines collections émettent quelques restrictions, comme le type ou l'interdiction de la valeur null.

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

On peut citer

ArrayList,
HashSet,
HashMap
LinkedList

Les collections

- C’est une infrastructure => ensemble de ressources qui forment un système technique cohérent
  - Des interfaces => définissent des « objets typiques » qui sont des collections.
  - Des implémentations => des classes qui mettent en œuvre les interfaces typiques pour un usage concret.
  - Des algorithmes => des objets ou façon de faire qui apportent à des collections des propriétés intéressantes.

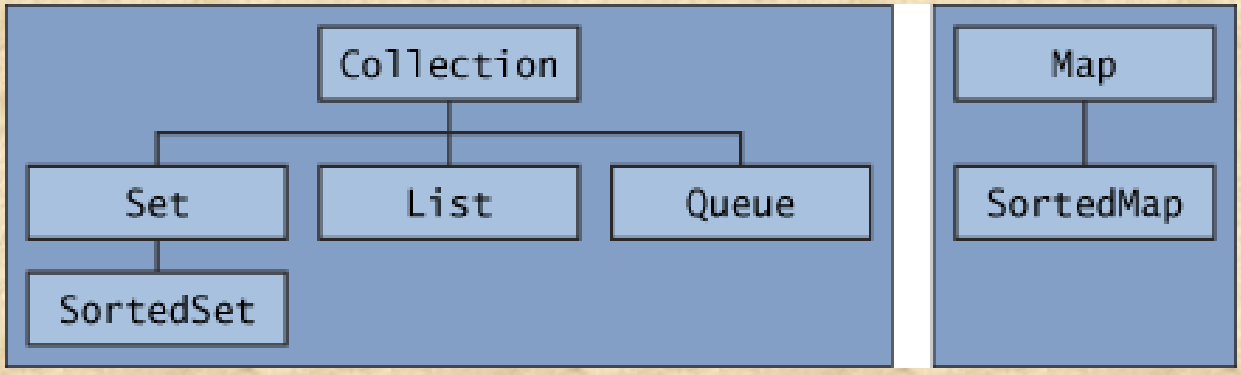
Container = Implémentation + Algorithmes

Les collections

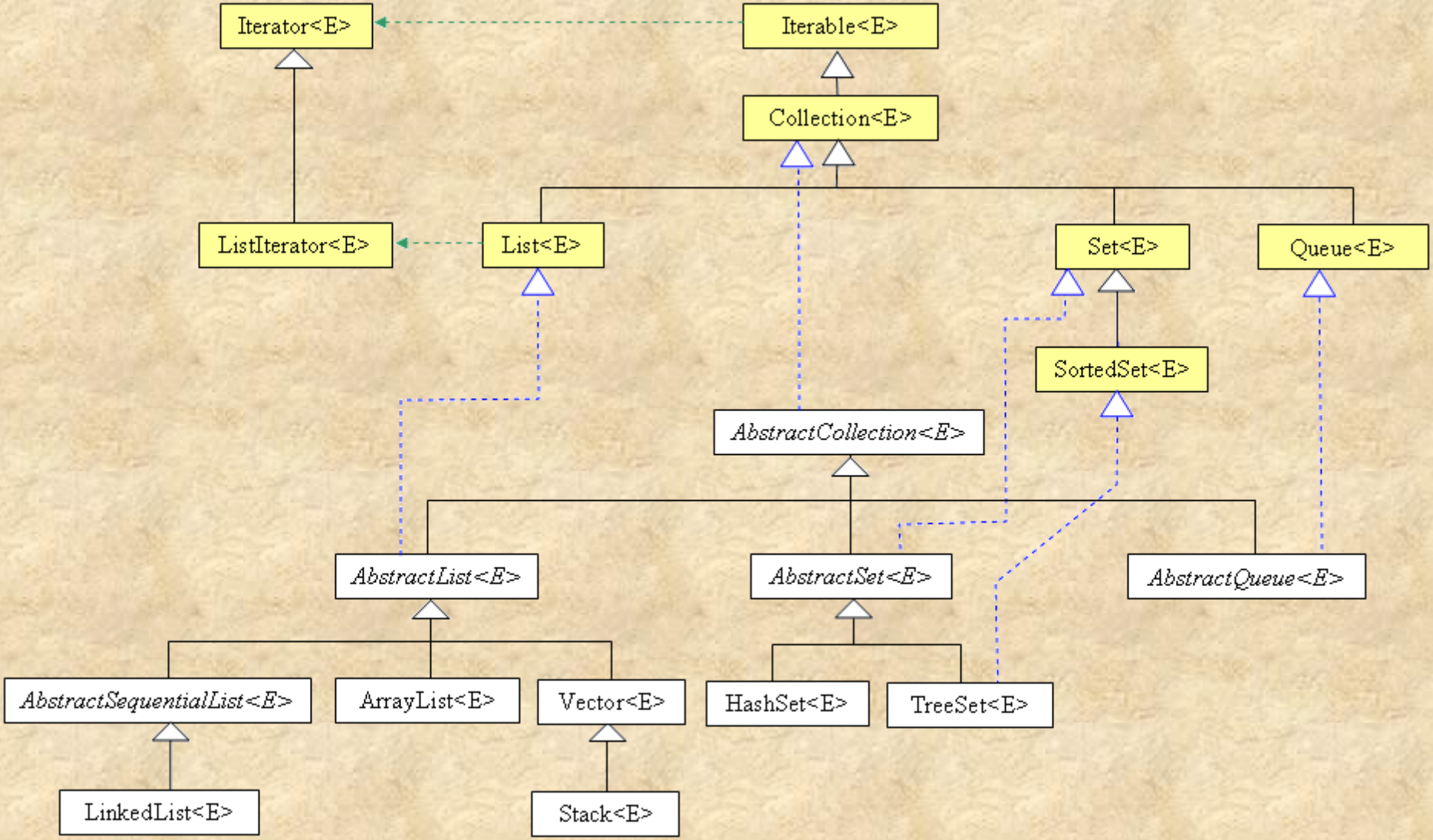
- Les arguments de Sun
  - Effort de programmation réduit. => on vous les propose toutes faîtes
  - Augmente la vitesse et la qualité du programme. => Toutes les recherches et tris sont systématiquement algorithmés
  - Permet l'interopérabilité d'API disjointes. => Des collections manœuvrées en tant que tel peuvent masquer la réalité des éléments qui les composent
  - Réduit le temps et l'effort d'apprentissage de nouvelles API. => Il n'y a pas de collection « custom », on utilise un modèle unifié
  - Réduit l'effort pour créer des nouvelles API. => s 'inscrire dans le modèle standard réduit le temps d'analyse
  - Réutilisation plus massive. => tout ce qui s 'appuie sur ces collections est plus réutilisable

Les interfaces de collection

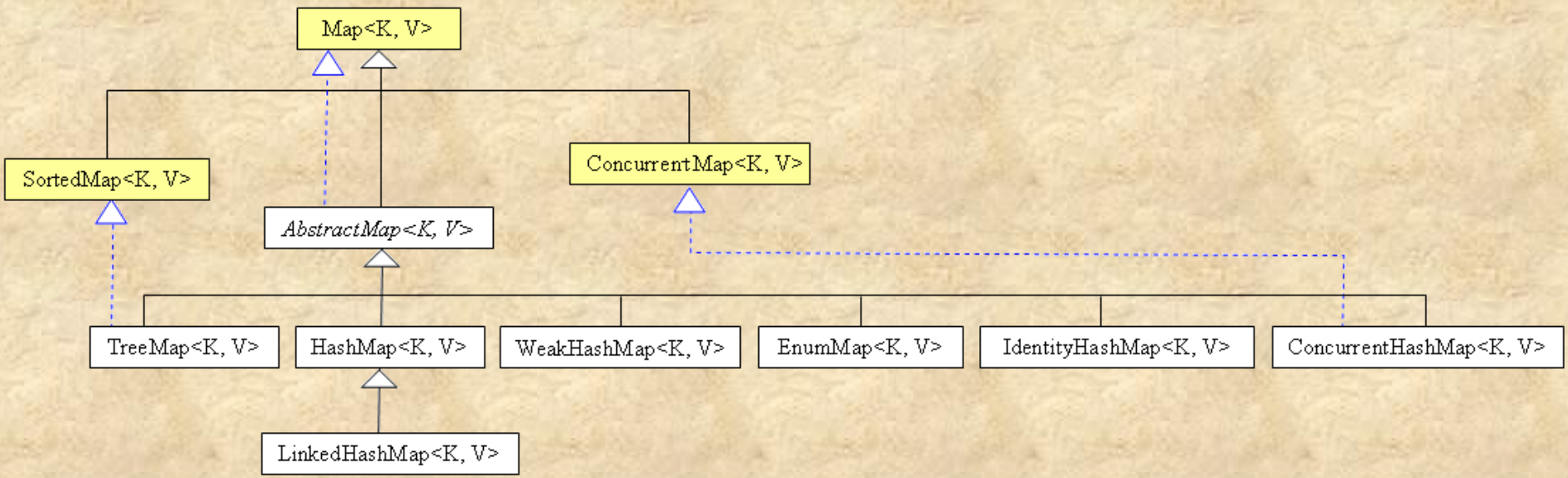
- Définir des interfaces permet de fournir des principes de collection indépendamment de leur implémentation réelle.
- Deux grandes familles centrales :
  - Les structures de Collection proprement dites (ensembles, listes, piles) - rangement linéaire.
  - Les structures de Map (catalogues, annuaires) - structures d'indexation.



Hiérarchie des Collections



Hiérarchie des Maps



Les interfaces

- Iterable
- Iterator
- ListIterator
- Collection
- List
- Set
- SorteSet
- Queue
- Map

L’interface Iterable

- Cette interface contient une seule méthode :

  Iterator<E> iterator() Retourne un littérateur sur le conteneur

- Toute collection qui implémente cette interface (Y compris les tableaux Java) permet l'écriture de *for* simplifié

L’interface Iterator

• L'interface *Iterator* est utilisée par l'interface *Iterable*

• L’interface *Iterator*, permet le parcours des éléments d’un conteneur, sans avoir besoin de savoir de quelle nature est ce conteneur.

• L’interface *Iterator* contient les méthodes suivantes :

- ✓ **boolean** hasNext() : retourne *true* si l’itérateur n’est pas arrivé en fin de l’ensemble et *false* sinon.
- ✓ E next() : permet d’avancer l’itérateur, et qui retourne l’élément qui a été sauté. Lève une exception *NoSuchElementException* si l'itérateur n'a pas d'objet qui le suit.
- ✓ **void** remove() : supprime l’objet qui vient d’être obtenu par *next*. Cette méthode est facultative

L’interface ListIterator

- L'interface *ListIterator* dérive de l'interface *Iterator*
- Elle est utilisée par l'interface *List*.
- Elle introduit notamment des méthodes qui permettent de parcourir une collection en sens inverse.
- On peut obtenir un itérateur calé sur la liste

```
public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();
    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    void remove(); //optional
    void set(E e); //optional
    void add(E e); //optional
}
```

L’interface Collection

- La « collection de … » est le moyen le plus général pour transporter plusieurs objets d’un endroit à un autre.
- Les collections sont "inter-compatibles" . On peut donc facilement convertir des collections d’une implémentation vers une autre.
- Une collection peut être une liste, un ensemble, voire un « tableau de ... »

Les interfaces Collection

- Les grandes fonctions des collections :
  - La construction des collections.
  - Les opérations de masse.
  - Les parcours.
  - Le déclenchement des algorithmes (recherche, accès rapide, tri --- implicites ou explicites)

L’interface Collection

- Les manœuvres « de masse »
  - Une manœuvre de masse, c’est une méthode qui permet la manipulation de tout ou partie de la collection, en bloc.
  - Exemple : fusion de liste, ajout d’un tableau à une liste, extraction d’un sous-tableau d’un tableau.
  - S’oppose à « manœuvre unitaire », « manipulation d’élément isolé »
  - Permet en quelques sortes de former une « algèbre des collections »

L'interface Collection

- La collection est parcourable de deux façons :
  - Par une boucle d'exploration for-each

```
for (Object o : collection)
    System.out.println(o);
```

  - Par obtention d'un Itérateur :

Un itérateur est un « machin » qui itère

```
public interface Iterator<E>
{
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

Les implémentations de collection

- Une implémentation est une classe qui réalise l'interface
- Une implémentation favorise certains aspects : performance, vitesse de recherche, empreinte mémoire, mutabilité
- Une implémentation peut ne pas réaliser TOUTES les méthodes préconisées par une interface, mais elle ne peut déroger à la règle de l'implémentation => Exceptions

Les algorithmes de collection

- Un algorithme est un calcul applicable sur la collection.
- Les algorithmes principaux concernent la recherche et les tris.
- Plusieurs algorithmes de même fonction peuvent être nécessaires en fonction des contraintes environnementales.

L’interface Set

- Le Set (ensemble) est une collection qui ne peut contenir de doublons. => Théorie des ensembles (Bourbaki)
  - il n'y a pas dans un *Set* deux éléments *e1* et *e2* tels que *e1.equals(e2)*.
- Usages principaux :
  - le marquage unique d’une qualité (l’élément a la qualité si il est indexé dans l’ensemble).
  - Le dédoublonnage : enlever les doubles.

L’interface Set

- Les commandes de masse reproduisent les opérateurs de la théorie générale des ensembles :
  - containsAll -> Test de sous ensemble
  - addAll -> Union de deux ensembles
  - retainAll -> Intersection de deux ensembles
  - removeAll -> Différence de deux ensembles

- La liste des méthodes est identique à la liste des méthodes de l'interface Collection.

L’interface SortedSet

• Un *SortedSet* est un *Set* qui garantit que le parcours des éléments, du premier au dernier, se fera dans l'«ordre» des éléments.

• Les éléments ajoutés dans un *SortedSet* doivent implémenter l'interface *Comparable*.

L’interface List

- L’interface List propose un comportement dit de « collection ordonnée » ou « séquence »
  - Une liste accepte des tuples

- La List propose quatre « jeux » de manœuvres :
  - Accès par position
  - Recherche d’élément
  - Exploration/Parcours/Itération
  - Plages (accès et utilisation de sous-listes)

L’interface List

- Deux implémentations de List :
  - ArrayList, à partir d’un tableau
  - LinkedList, à partir de cellules chaînées
- La première est plus performante dans le cas général.
- L’ancienne classe java.util.Vector implémente désormais la List

L’interface List

- Opérations de masse :
  - addAll, issue des fonctions de masse de la collection générique
- Opérations d’accès aux éléments :
  - get(int), set(int, Object), add(Object) et remove(int ou Object)

L’interface List

- L’interface de l’itérateur de liste permet la modification de la liste sous-jacente lors de l’exploration.

- Sous-listes : subList(fromIndex, toIndex)
  - La sous-liste est « appuyée » sur la liste d ’origine => modifier la sous-liste modifie la liste d ’origine.

L’interface Queue

- On parle en français d’une « File ».
- Une file admet 3 méthodes principales :
  - (1) insertion en tête, (2) retrait en queue, (3) examen de la tête sans retrait
- La queue propose deux jeux de méthodes :
  - (1) add(Object), (2) remove(), (3) element() lèvent des exceptions
  - (1) offer(Object), (2) poll(), (3) peek() renvoient des valeurs d’erreur sans exceptions.

L’interface Queue

- Fonctionnalités annexes apportées aux Queues :
  - Gestion de l’ordre des éléments de la file :
    - L’ordre trivial est de type FIFO
    - On peut imaginer un ordre de type « priorité à … »
  - Gestion de la taille maximale des files

L’interface Map

- Une Map gère une association clef -> valeur.
  - La clef est nécessairement unique
  - Le paradigme est celui de y = f(x)

- Une Map gère donc :
  - Des clefs
  - Des valeurs
  - Des couples (clef,valeur)

L’interface Map

- Méthodes élémentaires : inscription, récupération et désinscription
  - get(Clef)
  - set(Clef, Valeur)
  - remove(Clef)
- Méthodes d’examen :
  - containsKey(Clef) ?, containsValue(Valeur) ? size() ? isEmpty() ?

L’interface Map

- Trois implémentations de base de cette interface :
  - La HashMap
  - La TreeMap
  - La LinkedHashMap

=> Trois implémentations pour trois niveaux de performances distincts.

L’interface Map

- La Map est une évolution d’une ancienne classe Hashtable

Les différences sont :

- la Hashtable admet des répétitions de clef
- la Map permet l’exploration des clefs, valeurs et couples d’association.
- La Hashtable ne permet pas l’exploration couples d’association.
- La Map fournit un itérateur sur la map d’appui..
- La Map dispose des méthodes génériques de collection, pas la Hashtable.

L’interface Map

- Vues de collection : comment on peut extraire certaines collections de la Map :
  - keySet() -> Fournit une collection des clefs
  - values() -> Fournit la collection des valeurs
  - entrySet() -> Fournit la collection des couples
- Algèbre des Map : On peut définir des macro-opérateurs en combinant les vues de collection et les opérations de collection

ex : les clefs communes à deux Maps :

```
Set commonKeys = new HashSet (m1.keySet());
commonKeys.retainAll(m2.keySet());
```

Algorithmes sur les collections

- Sur des collections, on peut « plugger » des algorithmes.
- Les algorithmes courants sont les algorithmes de tri et de recherche.
- Cette combinaison donne lieu à la création de classes hybrides
  - SortedSet est un ensemble qui maintien un ordre sur les objets qui y sont rangés
  - SortedMap est une Map qui maintien l'ordre de ses clefs

Mapping UML vers Collections

Ensemble d'éléments


```
public class A {
        private Set<B> b;

        public A() {
                b = new HashSet<B>();
                ...
        }

        ...
}
```

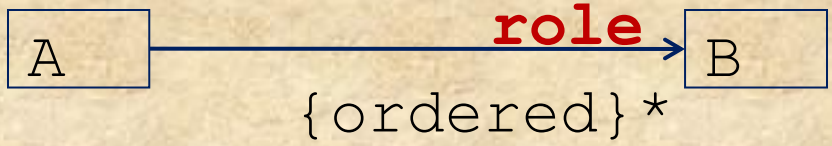
Ensemble d'éléments nommé


```
public class A {
        private Set<B> role;

        public A() {
                role = new HashSet<B>();
                ...
        }

        ...
}
```

Eléments ordonnés

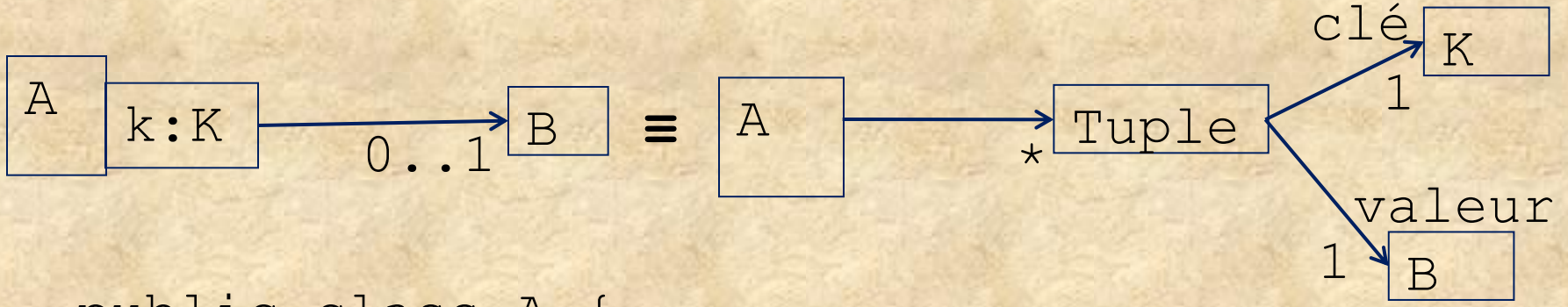

```
public class A {
     private List<B> b;

     public A() {
          b = new LinkedList<B>();
          ...
     }
     ...
}
```

Eléments ordonnés nommés




```
public class A {
     private List<B> role;

     public A() {
          role = new ArrayList<B>();
          ...
     }

     ...
}
```

Liste d'associations




```
public class A {
      private Map<K,B> b;

      public A() {
            b = new HashMap<K,B>();
            ...
      }
      ...
}
```

Rq: si multiplicité *, type de b : Map<K, Set<B>>

FIN DU COURS