Les collections et dictionnaires en Java
(gérer des groupes d'objets)

1 Introduction
Les tableaux sont inadéquats pour gérer une quantité importante d'informations du même type quand leur nombre n'est pas connu à l'avance. Par exemple, le nombre de messages postés dans un sujet du forum n'étant pas limité, il existe des solutions plus simples que les tableaux pour stocker ces messages.
Le package java.util contient plusieurs classes de collection utilisées pour gérer un ensemble d'éléments de type objet et résoudre les limitations inhérentes aux tableaux. Chaque classe est prévue pour gérer un ensemble avec des caractéristiques différentes :

- Les ensembles gérés par les classes java.util.ArrayList et java.util.LinkedList peuvent contenir des éléments égaux. Chaque élément mémorisé ayant une position déterminée, ces classes ressemblent aux tableaux Java mais ne sont pas limitées en taille.
- Les ensembles gérés par les classes java.util.HashSet et java.util.TreeSet ne peuvent pas contenir des éléments égaux. La classe java.util.TreeSet stocke les éléments de son ensemble dans l'ordre ascendant.
- Les ensembles gérés par les classes java.util.HashMap et java.util.TreeMap utilisent une clé pour accéder aux éléments au lieu d'un indice entier. La classe java.util.TreeMap stocke les éléments de son ensemble dans l'ordre ascendant des clés.

Utilisation de clés ou d'indices pour retrouver les objets dans une collection?
Suivant l'ensemble d'objets, chaque élément est associé soit à un indice qui donne son numéro d'ordre, soit à une clé qui l'identifie de manière unique. Par exemple, la clé associée à un utilisateur d'un forum pourrait être son pseudonyme.

2 Comparaison des tableaux et des collections
Les tableaux Java sont simples d'utilisation mais ont certaines limitations gênantes dans certains cas :

- Les tableaux ne sont pas redimensionnables.
- L'insertion d'un élément au milieu d'un tableau oblige à déplacer tous les éléments qui suivent.
- La recherche d'un élément dans un grand tableau est longue s'il n'est pas trié.
- Les indices pour accéder aux éléments d'un tableau doivent être entiers.

Aucune des classes de collection n'est limitée en taille.
Chacune résout de façon optimale certaines des autres limitations des tableaux :

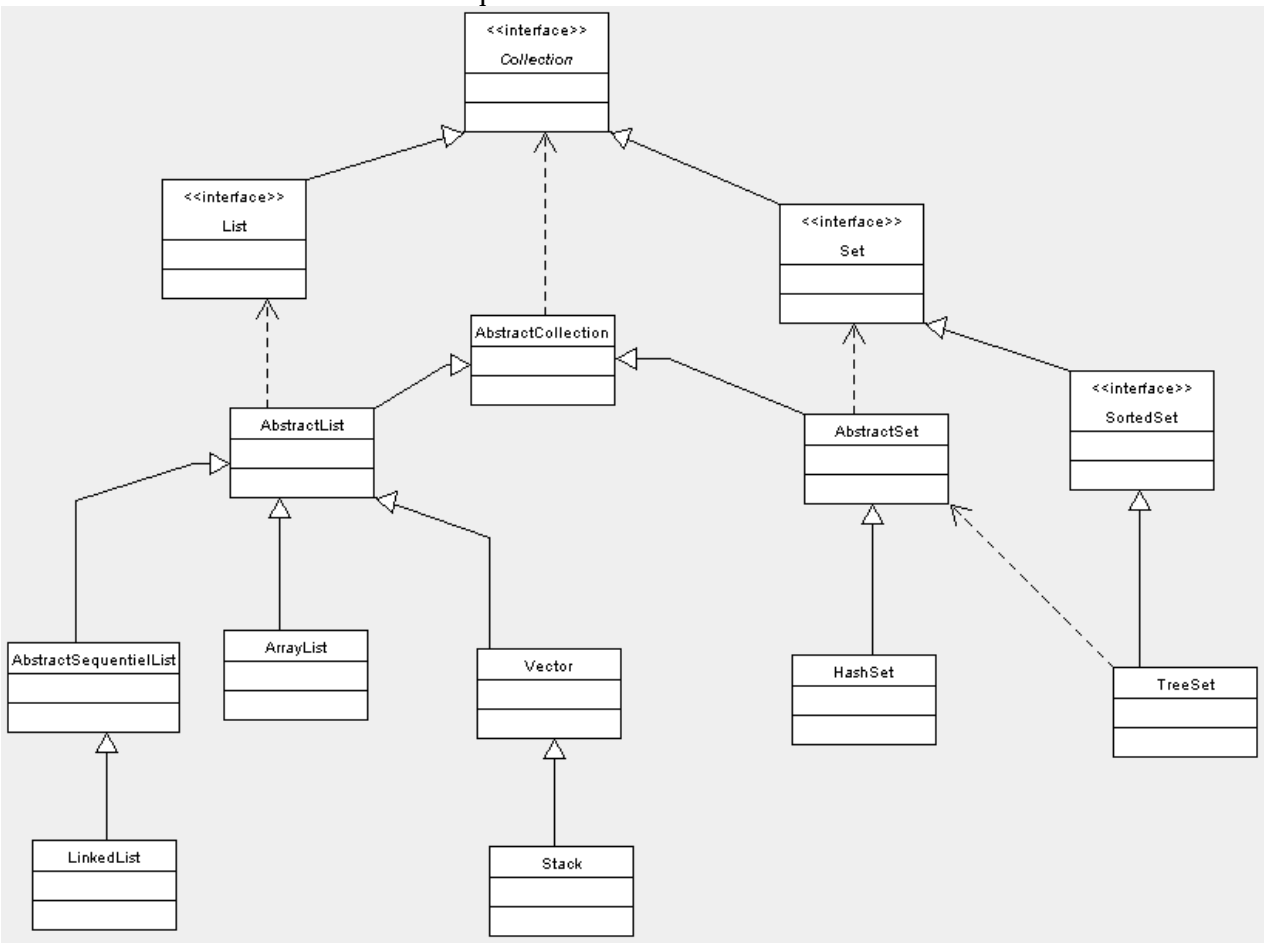
- La classe ArrayList est idéale pour ajouter à la suite les uns des autres des éléments dans un ensemble ordonné.
- La classe java.LinkedList est idéale pour insérer de nombreux éléments au milieu d'un ensemble ordonné.
- La classe java.util .HashSet est idéale pour gérer un ensemble dont chaque élément doit être unique. Elle améliore aussi les performances de recherche d'un élément.
- La classe TreeSet est idéale pour gérer un ensemble trié d'objets uniques.

- La classe java.util.HashMap est idéale pour accéder aux éléments d'un ensemble grâce à une clé.
- La classe java.util.TreeMap est idéale pour gérer un ensemble d'éléments trié dans l'ordre de leur clé d'accès.

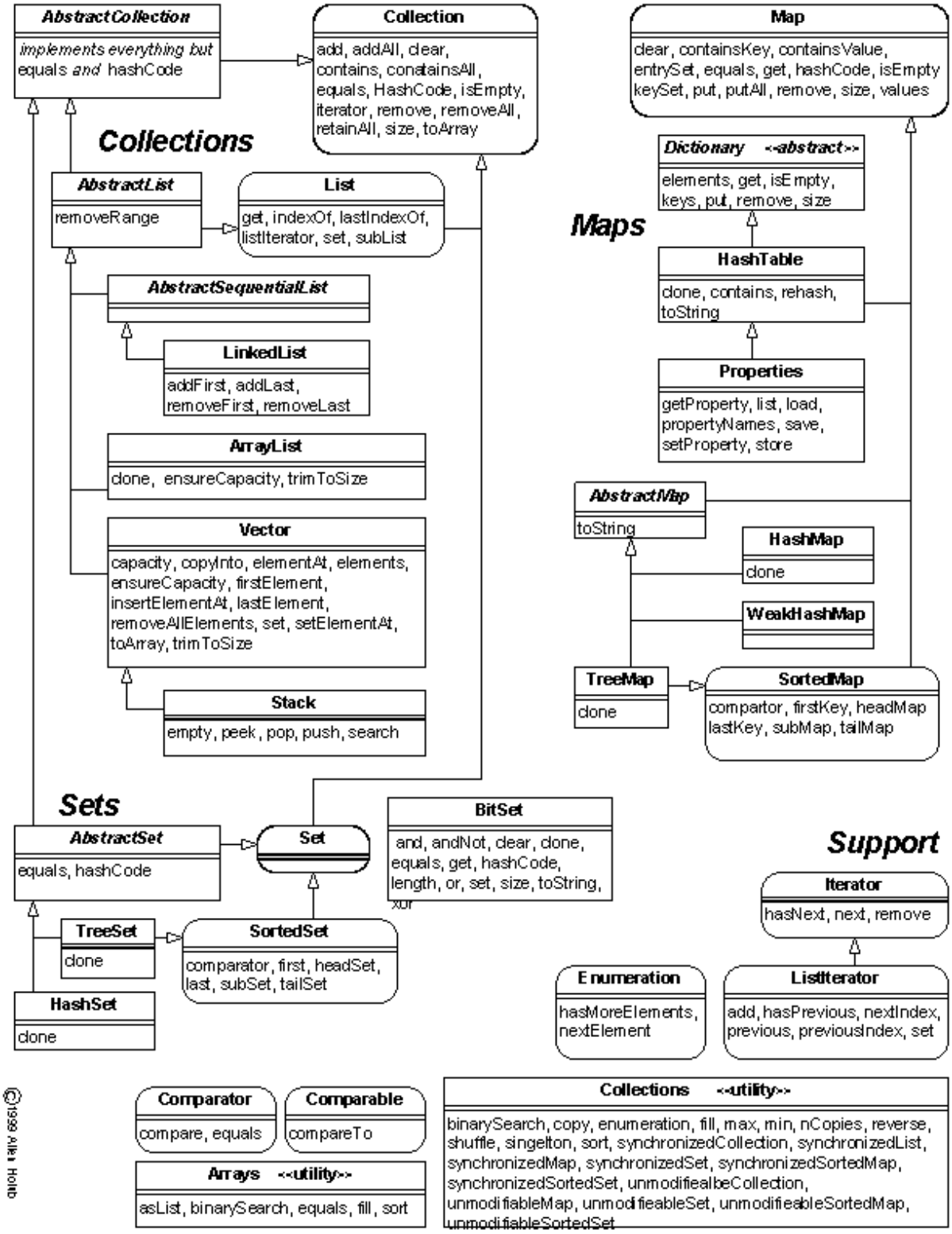
Les tableaux étant typés (grâce au type des éléments donné lors de leur déclaration), plus simples à programmer et moins gourmands en mémoire, ils sont néanmoins très souvent utilisés notamment pour les ensembles dont la taille est connue à l'avance.

3 Anciennes Classes de collection (Java 1.0)

Les classes de collection décrites ici sont présentes depuis la version 1.2 de Java. Les classes java.util.Vector et java.util.Hashtable disponibles depuis la version 1.0 de Java ont des fonctionnalités équivalentes à celles des classes ArrayList et HashMap.
Ces deux classes, comme leur sous-classe Stack (utilisée pour créer une pile d'objets) et Properties (utilisée pour créer une liste de propriétés avec leur valeur comme celles de la classe java. lang. System), sont toujours mises en œuvre dans de nombreux programmes et dans certaines classes de la bibliothèque standard.



4 Le Framework de collection Java 1.2



The "operations" compartment shows only those methods that are introduced in a given class; inherited and overriden methods are not shown

Le Framework Java 1.2 est destiné à apporter des fondations solides à tous les types d'ensembles d'objets, par le biais d'interfaces spécifiant les fonctionnalités de chacun de ces types.
Les particularités de ces interfaces sont les suivantes :

- Collection représente un groupe d'objets, et autorise les doublons.
- List est une Collection dans laquelle entre la notion de position de chaque élément.
- Set est une Collection mais qui interdit les doublons.
- SortedSet est un Set d'objets pouvant être triés.
- Map représente un groupe de paires d'objets.
- SortedMap est une Map de paires triables.
- Iterator est une interface semblable à Enumeration.
- Comparable est une interface permettant de rendre des objets comparables.

On trouvera dans le Framework un certain nombre de classes implémentant ces interfaces :

- Vector
- HashSet et TreeSet (objets Comparable) qui implémentent Set
- ArrayList (équivalent d'un Vector 1.0) et LinkedList (équivalent d'une Stack 1.0) qui implémentent List
- HashTable et TreeMap (équivalent d'un Properties 1.0) qui implémentent Map.

4.1 L'interface java.util.Collection

Elle représente donc un groupe d'objets qui autorise les doublons.
Elle permet de définir les fonctionnalités de base (ne pas la confondre avec la classe Collections (avec un s à la fin) qui offre des services.

Voir les méthodes qu'elle définit.

4.2 L'interface java.util.Iterator

Elle est très semblable à la classe Enumeration, mais permet en plus la suppression d'éléments. Les méthodes sont les suivantes :

- boolean hasNext ( ) ; Renvoie true s'il existe encore au moins un élément à récupérer à l'aide de la méthode next.
- Object next () ; Retourne l'élément suivant.
- void remove() ; Supprime le dernier élément retourné avec next.

4.3 L'interface java.util.List

Les tableaux dynamiques. Un certain nombre de ses méthodes font double emploi avec l'interface Collection dont elle hérite. Elles ont été créées car leurs noms et leurs signatures sont plus proches des conventions que l'on a cherché à promouvoir à partir de la version 1.1 de Java.

Cette interface sera utile notamment pour créer des objets Swing représentant graphiquement des objets List.

Voir les méthodes qu'elle définit.

4.4 L'interface java.util.Set

Les ensembles. Elle hérite aussi de Collection, elle représente un ensemble d'objets, donc il n'y a pas de doublon.
Notons les classes qui implémentent cette interface :

- HashSet qui est une implémentation de Set,
- TreeSet

4.5 La classe java.util.TreeSet

Elle implémente l'interface Collection, et a les particularités suivantes :
N'accepte que des objets comparables, c'est-à-dire implémentant l'interface Comparable
N'autorise pas les doublons d'objets, ayant la même référence mais aussi étant identiques par la méthode compareTo de l'interface Comparable

4.6 L'interface java.util.Map

Les dictionnaires. Elle représente un ensemble de paires d'objets. On trouvera notamment comme implémentation les classes HashMap, HashTable.

4.7 L'interface java.util.SortedMap

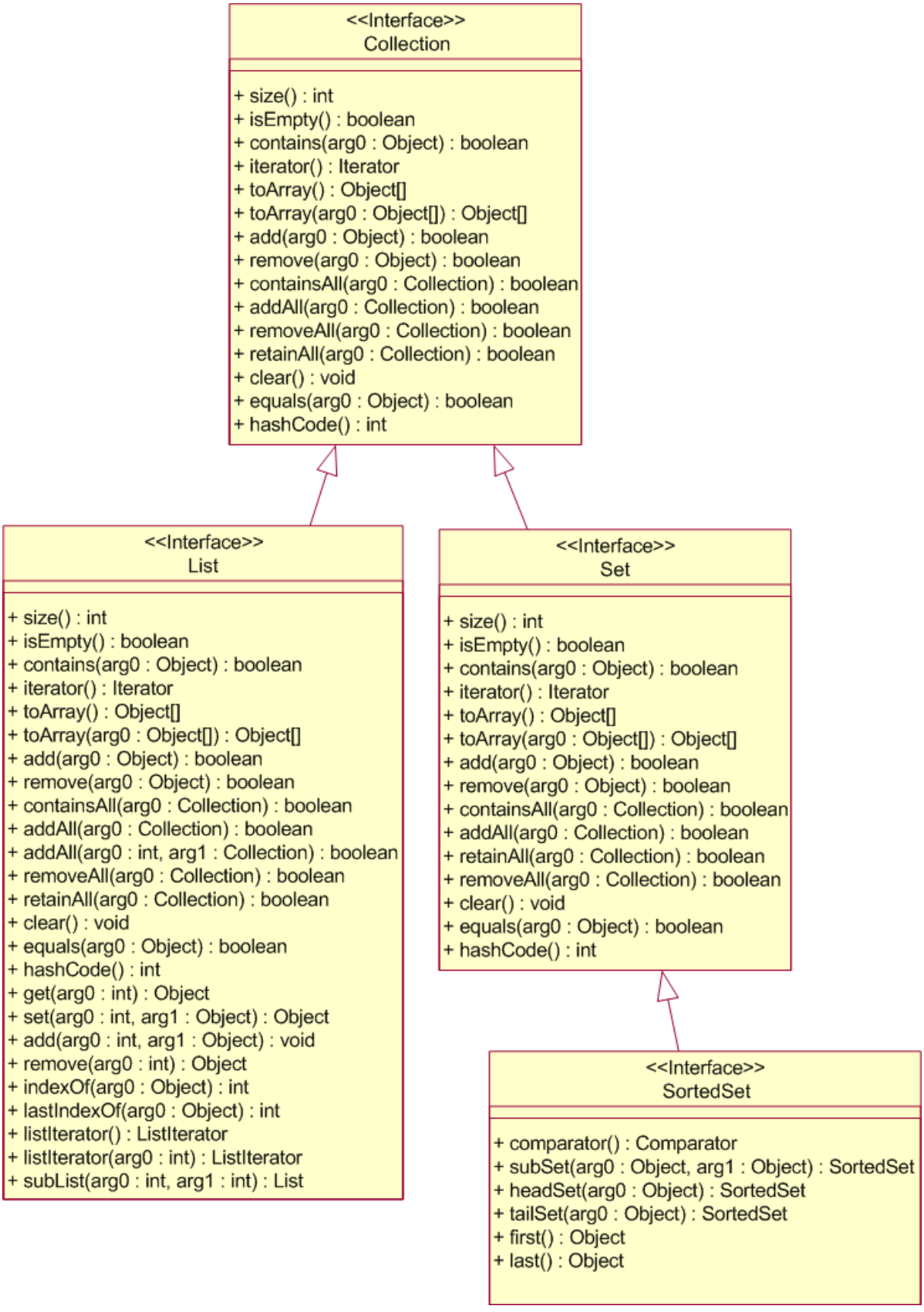
Des tables de hachage triées avec l'interface SortedMap.
Cette interface hérite de Map, et ajoute la notion de tri. Le tri s'effectue sur les clés.
Ces clés doivent donc implémenter l'interface Comparable, ou bien être acceptées par un objet Comparator passé dans le constructeur (ce constructeur n'est pas précisé dans l'interface, car on ne peut pas spécifier de constructeur dans une interface).

4.8 La classe Collections

Des outils de manipulation avec la classe Collections
Elle est exclusivement composée de méthodes statiques, permettant toutes sortes de manipulations sur des objets Collection, List, Set et Map :

- Copies
- Recherches
- Tris
- ,…,etc.

5 L'interface Collection et sa hiérarchie



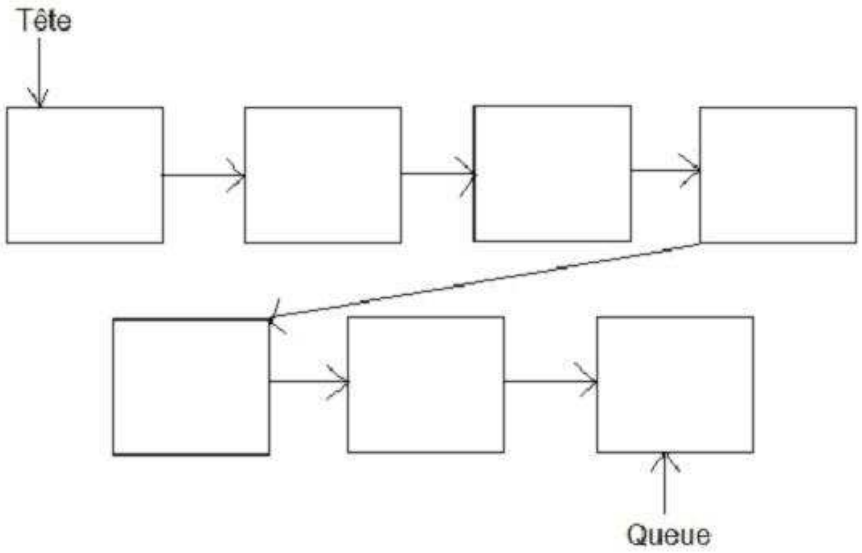
6 ArrayList et LinkedList

Tableaux dynamiques = Listes ordonnées d'objets
java.util.ArrayList et java.util.LinkedList
Ces deux classes gèrent des **ensembles ordonnés** d'éléments accessibles par leur indice. La classe ArrayList mémorise ses éléments dans un tableau Java. Si ce tableau interne est trop petit lors de l'ajout d'un nouvel élément à la collection, il est automatiquement remplacé par un nouveau tableau, plus grand, initialisé avec les références de l'ancien tableau. La classe LinkedList mémorise ses éléments avec une liste doublement chaînée, ce qui permet d'insérer plus rapidement un élément dans la collection, mais ralentit l'accès à un élément par son indice.

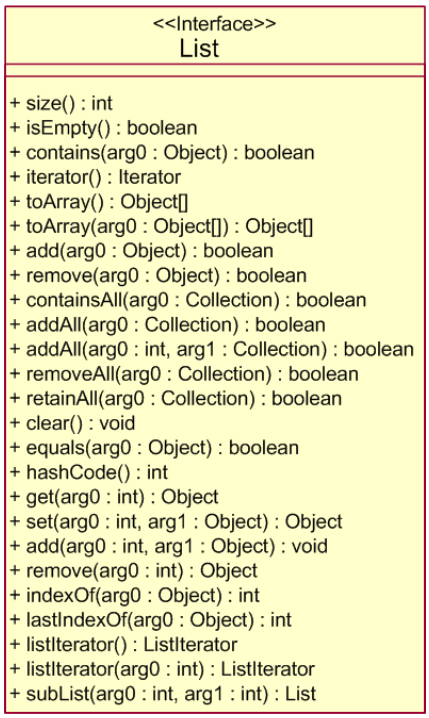
Voir organisation en mémoire d'instance

6.1 Liste chaînée

Comme le montre la figure suivante, chaque élément d'une liste chaînée est mémorisé par un objet intermédiaire appelé chaînon et lié à ses chaînons précédent et suivant (d'où le nom de liste doublement chaînée). Comme une instance de LinkedList ne mémorise que la référence du chaînon de tête, l'accès aux autres éléments de la liste s'effectue séquentiellement grâce aux liens entre les chaînons. L'insertion d'un élément dans une liste chaînée s'effectue en créant un nouveau chaînon puis en affectant les valeurs de quatre liens précédents et suivants. Comme l'insertion d'un élément dans un tableau nécessite de décaler tous les éléments qui suivent, cette opération est plus rapide avec une liste chaînée si l'ensemble comporte beaucoup d'éléments.



6.2 Principales méthodes de ArrayList et LinkedList



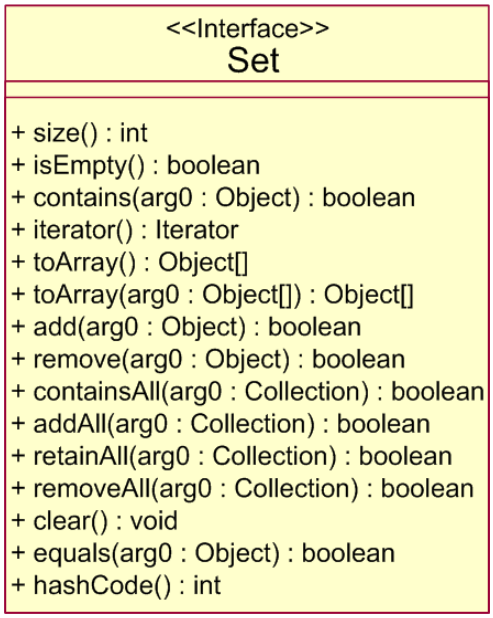
Les méthodes de ces classes effectuent des opérations similaires à celles qui peuvent être réalisées avec un tableau. Voici un tableau avec certaines méthodes de l'interface List et qui sont donc celles présentes dans toutes les collections de type List.
Ce sont généralement celles de l'interface List :

Signature de la méthode	Catégorie
public void add(int index, Object obj) public boolean add(Object obj)	Ajout d'une référence à la collection soit à un indice donné, soit en fin de liste.
public void clear() public Object remove(int index) public boolean remove(Object obj)	Suppression de tout ou partie des éléments de la collection.
public Object get(int index) public Object set(int index, Object obj)	Interrogation et modification d'un élément de la collection à un indice donné
boolean contains(0bject obj) public int index0f(Object obj) public int lastIndex0f(Object obj)	Recherche d'un élément dans la collection, en utilisant la méthode equals pour comparer les objets.
public int size()	Taille de la collection
public Iterator iterator()	Itérateur pour énumérer les éléments de la collection
public Object [] toArray (Object [] a)	Récupération des éléments de la collection dans un tableau

Voir exemple casier à bouteilles ou cave à vin

7 HashSet et TreeSet

Ensembles d'objets uniques
java.util.HashSet et java.util.TreeSet:.



Ces deux classes gèrent des ensembles d'éléments différents les uns des autres. La méthode equals est utilisée pour comparer les éléments. La classe HashSet mémorise ses éléments dans un ordre quelconque avec une table de hash (la référence null peut être utilisée). La classe TreeSet mémorise ses éléments dans l'ordre ascendant avec un arbre, pour maintenir plus rapidement le tri des éléments stockés.

7.1 Importance de bien écrire hashCode pour stocker des objets dans HashSet

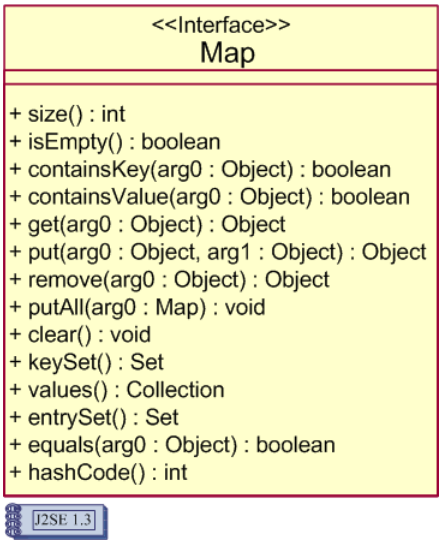
Si la classe des objets stockés par une instance de HashSet redéfinit la méthode equals, il faut redéfinir aussi la méthode hashCode dans cette classe pour que ce type de collection fonctionne correctement.

7.2 Principales méthodes

Signature de la méthode	Catégorie
public boolean add(Object obj)	Ajout d'une référence à la collection si l'objet n'est pas déjà dans la collection
public void clear() public boolean remove(Object obj)	Suppression de tout ou partie des éléments de la collection
public boolean contains(Object obj)	Recherche d'un élément dans la collection, en utilisant la méthode equals pour comparer les objets
public int size()	Taille de la collection
public Iterator iterator	Itérateur pour énumérer les éléments de la collection

La classe TreeSet contient aussi les méthodes first et last, renvoyant le plus petit et le plus grand élémént d'une collection.
L'accès aux éléments de ces classes s'effectue séquentiellement grâce à un itérateur de type java.util.Iterator renvoyé par leur méthode iterator.
La mise en oeuvre de l'interface Iterator et de ces classes est abordée au chapitre "Abstraction et interface".

8 HashMap et TreeMap



java.util.HashMap et java.util.TreeMap: Dictionnaire d'objets ou tableau associatif
Ces deux classes gèrent des ensembles d'éléments accessibles par une clé correspondant (map en anglais) à un élément. Ces classes mémorisent en fait un ensemble d'entrées (entries) associant une clé (key) et son élément (value). L'accès par clé dans une collection est comparable à l'accès par indice dans un tableau.

Rappel sur l'acccès à un ensemble par indice (List)

- Chaque indice du tableau est unique
- Un élément peut être mémorisé plusieurs fois à des indices différents

Accès à un ensemble par clé

- Chaque clé de la collection est unique
- Un élément peut être mémorisé plusieurs fois avec des clés différentes

8.1 Comparaison de l'accès par clé et de l'accès par indice

Tableau ensemble de type Object[]	Collection ensemble de classe ArrayList ou LinkedList	Collection ensemble HashMap ou TreeMap
ensemble[i] renvoie l'élément d'indice i dans ensemble	ensemble.get(i) renvoie l'élément d'indice i dans ensemble	ensemble.get(key) renvoie l'élément de clé key de ensemble.
ensemble[i] = val; affecte val à l'élément d'indice i.	ensemble.set(i, val); affecte val à l'élément d'indice i.	ensemble.put(key, val); affecte val à l'élément de clé key

Comme les clés peuvent être des chaînes de caractères, des instances des classes d'emballage ou d'autres classes, les classes HashMap et TreeMap permettent d'accéder à ces ensembles d'éléments de manière plus élaborée qu'avec un indice entier.
La classe HashMap mémorise ses entrées dans un ordre quelconque tandis que la classe TreeMap mémorise ses entrées dans l'ordre ascendant des clés avec un arbre, pour maintenir plus rapidement le tri des éléments stockés.

8.2 Principales méthodes

Catégorie	Signature de la méthode
Interrogation et modification d'un élément donné de la collection avec une clé donnée	public Object get(Object key) public Object put(Object key, Object value)
Suppression de tout ou partie des éléments de la collection	public void clear() public Object remove(Object key)

Recherche d'un élément dans la collection par sa clé ou sa valeur. Ces méthodes utilisent la méthode equals pour comparer key aux clés ou value aux valeurs de la collection.	public boolean containsKey(Object key) public boolean containsValue(Object value)
Taille de la collection	public int size()
Ensemble des clés et des éléments de la collection	public Set keySet() public Collection values()

9 La Généricité avec JAVA 5.0

La généricité équivalant au template C++ est probablement la fonctionnalité la plus demandée dans Java depuis son origine. Intégrée à Java 5.0, la généricité est utilisée par les classes de collection pour laisser le choix au programmeur de spécifier une classe différente de « Object » comme classe des éléments stockés. Autrement dit, il s'agit de paramétrer les collections avec un ou plusieurs types (Classe).
Le type (la classe) des éléments est spécifié entre les symboles < et > qui suivent la classe de collection.
Exemple :

```
ArrayList<Integer>
```

représente une collection de classe ArrayList dans laquelle seuls des objets de classe java.lang.Integer pourront être ajoutés.

La généricité simplifie alors la consultation des éléments d'une collection en évitant de faire appel à l'opérateur de cast (down-casting !).

9.1 En résumé...

Il est essentiel de connaître les collections car c'est à partir de ces classes Java que vous programmerez les classes qui formeront le cœur de vos applications dans le monde de l'entreprise.