

Les collections dans Java



Les collections

Collection :

- Modèle de structures de données
 - Abstraction d'un regroupement de données élémentaires
 - Réduit à quelques cas
 - Protocole de gestion
 - protocole d'accès
- Un cas classique d'abstraction générique et de mise en œuvre des concepts objets

Utilisation de collections pour

- Stocker, retrouver des données élémentaires dans un regroupement de données de nature similaire.

Exemples :

- un dossier de courrier : collection de mails
- un répertoire téléphonique : collection d'associations noms numéros de téléphone.
- ...



Les collections dans Java

– Types préexistants

- Tableau
 - Intégré dans le langage sous une forme spécifique []
 - Opérateurs de conversion depuis ou vers cette représentation.
 - Généricité prise en compte
- Vector<T>
 - Tableau à taille variable, abstraction du tableau dans le modèle objet.
- Hashtable<T>
 - Premier modèle de table associative, est devenu obsolète depuis l'existence de Map

– Conversions

- Depuis le type []
 - Dans la classe Arrays, la méthode statique asList de profil :

T[] ->List<T>

T... ->List<T>
- Vers le type []
 - Dans la classe Collection, la méthode statique toArray de profil : -><Object>[]
 - Dans la classe Collection, la méthode statique toArray de profil : <T>[] -> <T>[]



Abstraction des structures de données

$$\forall x, y \in \text{Set} \ni x.\text{equals}(y) \Rightarrow x == y$$

Collection

Set

List

[]

Map<K->

Dictionary

SortedSet

SortedMap

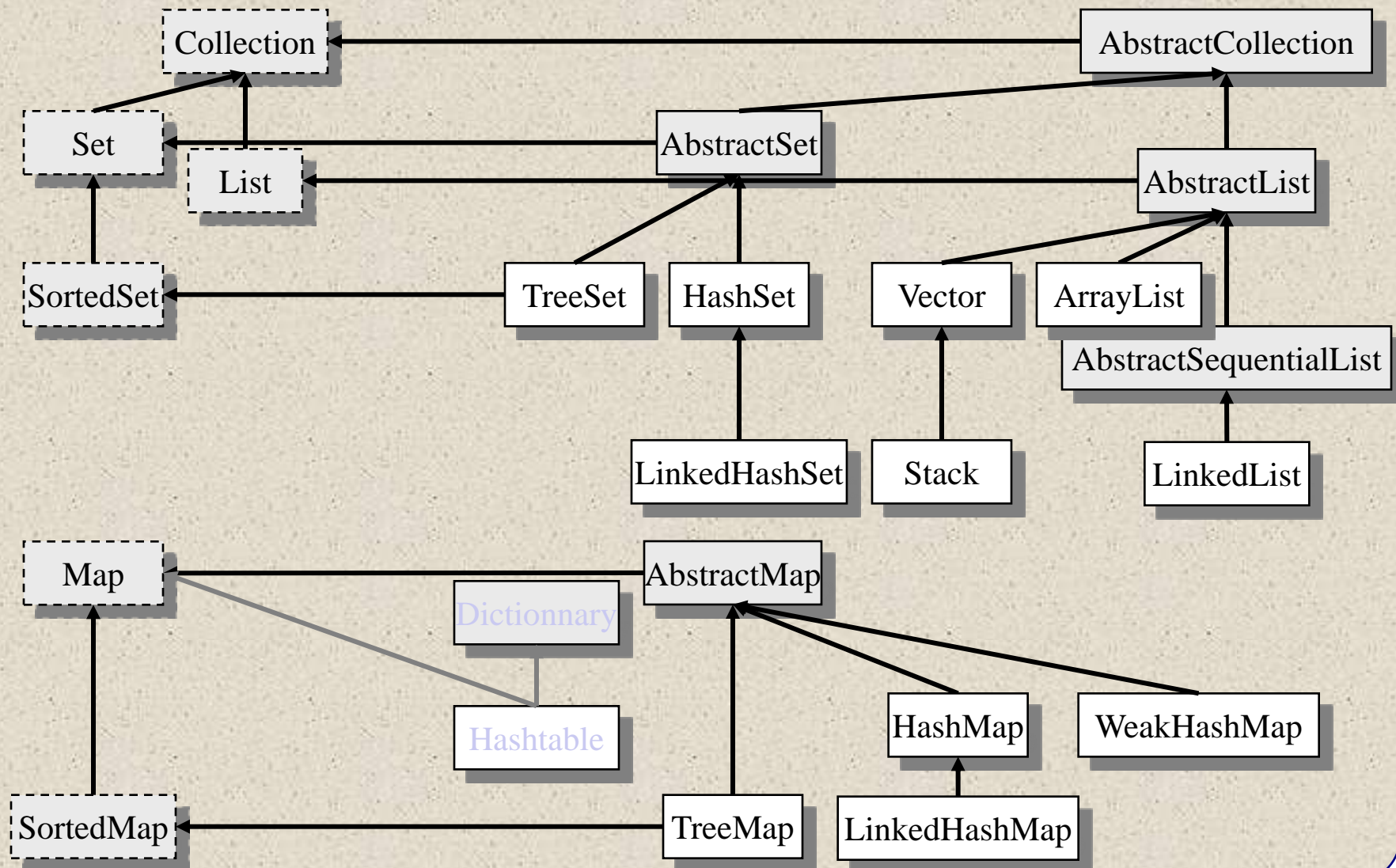
$$\begin{aligned} \exists x, y \in L \ni x.\text{equals}(y) \wedge x \neq y \\ \forall x, y \in L \ni \text{indexOf}(x) == \text{indexOf}(y) \Rightarrow x.\text{equals}(y) \end{aligned}$$

$$\begin{aligned} K \subset \text{Set}, V \subset \text{Collection}, M \subset K \times V \\ \forall r1, r2 \in M, k(r1) == k(r2) \Rightarrow v(r1) == v(r2) \end{aligned}$$

$$\forall y \in S, \exists x \in S \ni x \leq y$$



Hiérarchies des structures de données



Interface Collection

```
public interface Collection<T> {  
    // Basic Operations  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(T element);    // Optional  
    boolean remove(Object element); // Optional  
    Iterator<T> iterator();  
  
    // Bulk Operations  
    boolean containsAll(Collection<? > c);  
    boolean addAll(Collection<? Extends T> c);    // Optional  
    boolean removeAll(Collection<?> c); // Optional  
    boolean retainAll(Collection<?> c); // Optional  
    void clear();    // Optional  
  
    // Array Operations  
    Object[] toArray();  
    T[] toArray(T[] a);  
}
```



Interface Collection<T>

```
/**
 * @ensure this.contains(o)
 * @return !_this.contains(o)
 */
boolean add(T o)
/**
 * @ensure !this.contains(o)
 * @return _this.contains(o)
 */
boolean remove(Object o)
/**
 * @return  $\exists x \in C_{tel} \text{ que } x.equals(o)$ 
 */
boolean contains(Object o)
```

```
/**
 * @require extensibleWith(o)
 * @ensure contains(o)
 */
void add (T o)
           throw ExtensibleException
boolean extensibleWith(object o)
```



Interface List<T>

```
public interface List<T> extends Collection<T> {
    // Positional Access
    T get(int index);
    T set(int index, T element); // Optional
    void add(int index, T element); // Optional
    Object remove(int index); // Optional
    boolean addAll(int index, Collection<? Extends T> c); // Optional

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

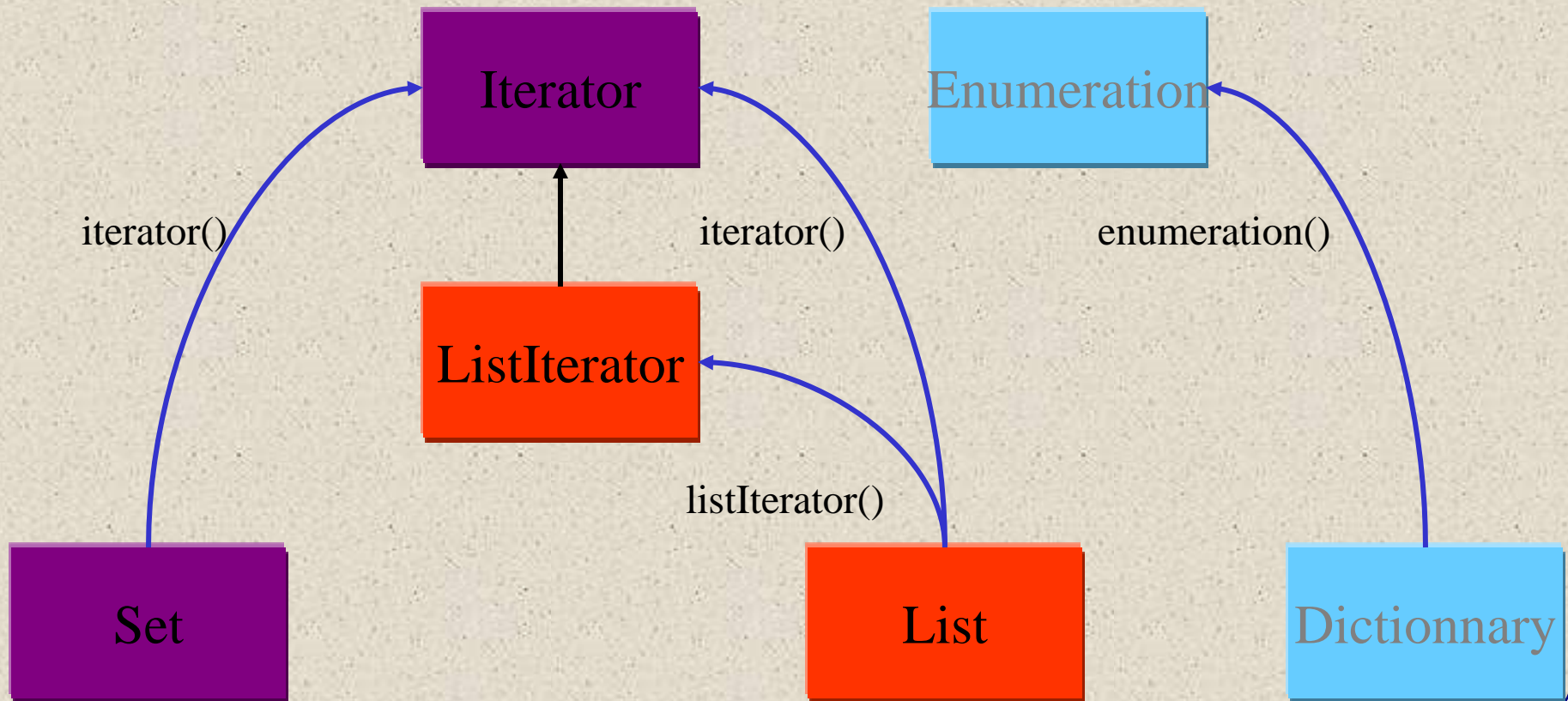
    // Iteration
    ListIterator<T> listIterator();
    ListIterator<T> listIterator(int index);

    // Range-view
    List<T> subList(int from, int to);
}
```



Iterator

Un iterator est un objet permettant d'énumérer les éléments constituant une collection dans ordre et d'une manière qui dépend de la nature de la collection.



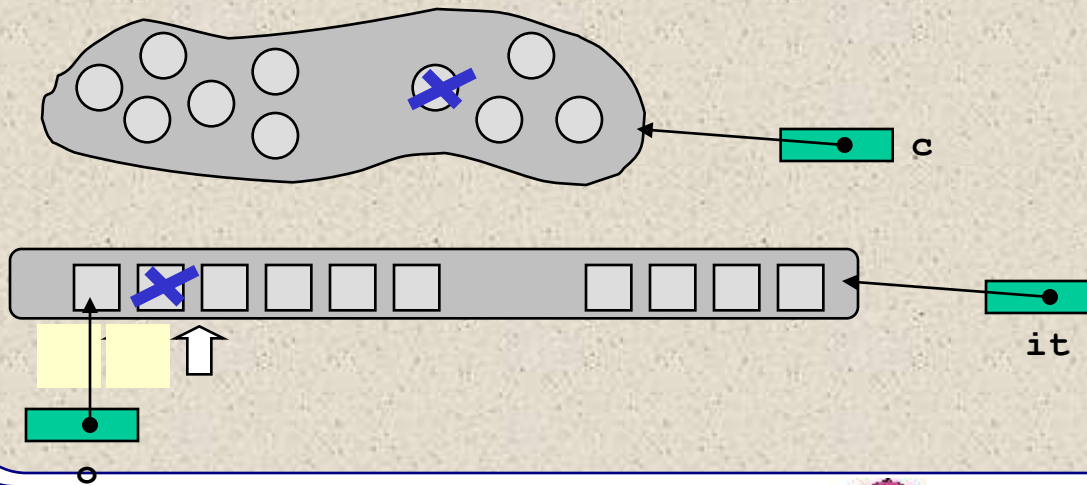
Iterator

`Iterator<T> iterator()`

- cette méthode renvoie un objet « iterator » qui permet le parcours séquentiel des éléments d'une collection.

Iterator **est une interface définie dans le package java.util**

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



`Collection<T> c = new ...`

`Iterator<T> it = c.iterator();`

```
T o = it.next();  
it.next();  
it.remove();  
it.remove();
```



Iterator

Modèle général d'utilisation d'un iterator qui ne constitue pas un modèle d'ordre supérieur car l'opération à effectuer n'est pas paramétrée.

```
Collection<T> c; // c est une collection de <T>
<T> e;
for(Iterator<T> i = c.iterator(); i.hasNext();){
    e= i.next();
    //Traiter e
}
```

Cette forme n'est valide qu'à partir de la version 1.5 de Java. Le type de la variable c doit être une classe qui dérive de Iterable<T>. Toute collection est «itérable».

```
Collection<T> c; // c est une collection de <T>
for(T e : c){
    //Traiter e
}
```

**Un Iterator ne permet le parcours d'une collection qu'une fois et une seule.
Un ListIterator autorise des retours en arrière.**



Cas particulier pour les types énumérés

```
public class test{  
    public enum Jour {LUNDI,MARDI,MERCREDI,JEUDI,VENDREDI,SAMEDI,DIMANCHE;}  
    /**  
     * @param args  
     */  
    public static void main(String[] args){  
        for(Jour j : Jour.values()) System.out.println(j);  
    }  
}
```

Les types énumérés ne sont pas directement «itérables», mais en obtenant le tableau des valeurs du type on peut utiliser la nouvelle notation d'itération.



Patrons d'utilisation d'itérateurs

- Parcours de tous les éléments d'une collection

```
Collection<T> c; // c est une collection de <T>
for(T e : c){
    //Traiter e
}
```

- Recherche d'un élément dans la collection

```
Collection<T> c; // c est une collection de <T>
T item=...;
for(T e : c){if(e.equals(item) break;}
```

```
Collection<T> c; // c est une collection de <T>
T item=...;
for(T e : c){if(e.equals(item){item=e; break;}}
```

- Parcours d'une sous-collection d'une collection

```
Collection<T> c; // c est une collection de <T>
for(T e : c){
    //Traiter e
    if(e.equals(item) break;
}
```



Les collections particulières

– Les collections non modifiables

- L'objectif est d'assurer l'invariance de l'objet que l'on peut caractériser par la formule : `_this.equals(this)`
- Assurée par l'émission systématique de l'exception `UnsupportedOperationException` pour chaque opération sensée violée cette propriété.
- Exemple de la méthode `add` :

```
public boolean add (T o) {  
    throws new UnsupportedOperationException();  
}
```

– Les collections synchronisées

- L'objectif est d'assurer l'accès concurrent à la collection par divers threads
- Exemple de la méthode `add` :

```
Collection<T> c;  
Object mutex;  
public boolean add (T o) {  
    synchronized(mutex) {return c.add(o)};  
}
```



La classe Collections

- Elle offre des fonctionnalités générales sur les collections
 - Des fonctionnalités permettant d'obtenir des collections (comme indiqué précédemment)
 - Des fonctionnalités de recherche, de tri, de constructions, ...
 - Portant soit sur des collections générales soit sur des types plus spécifiques comme les List.
 - Des fonctionnalités assurant la vérification dynamique de la conformité de type
 - Pour palier à l'unchecked conversion qui reste possible dans l'utilisation de la généricité de Java.
- Beaucoup de ses fonctionnalités sont semblables à celles que l'on trouve dans la classe Arrays qui est un cas particulier de collections.

