Ministère de l'Enseignement Supérieur et de la
Recherche Scientifique et de la Technologie
Institut Supérieur des
Études Technologiques Bizerte

Les collections génériques en java

Introduction aux Collections

Java propose plusieurs moyens de manipuler des ensembles d'objets :

on a vu les tableaux dont l'inconvénient est de ne pas être dynamique vis à vis de leur taille.

Java fournit des interfaces qui permettent de gérer des ensembles d'objets dans des structures qui peuvent être parcourues.

Ce chapitre donne un aperçu de ces collections. Elles sont toutes génériques.

Toutes les collections d'objets:

- sont dans le paquetage java.util
- implémentent l'interface générique Collection

Introduction aux Collections

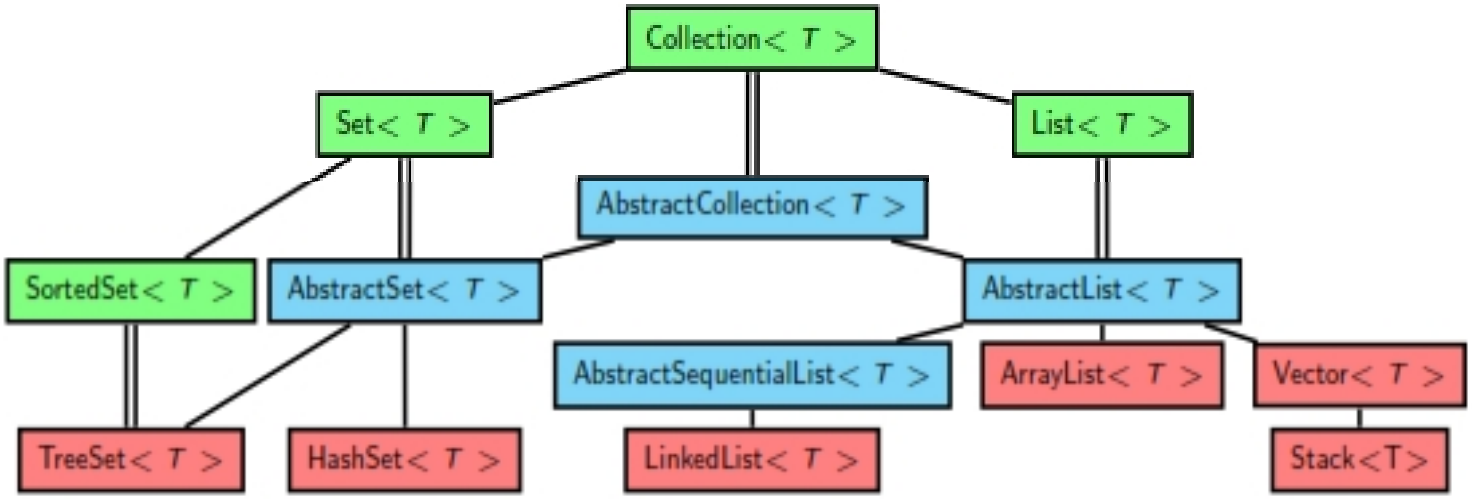
• les collections Java forment un **framework** qui permet de gérer des structures d'objets.

• **L'interface Set< T >** sert à implémenter les collections de type ensemble : les éléments n'y figurent qu'une fois et ne sont pas ordonnés.

• **L'interface List< T >** sert à implémenter les collections dont les éléments sont ordonnées et qui autorisent la répétition.

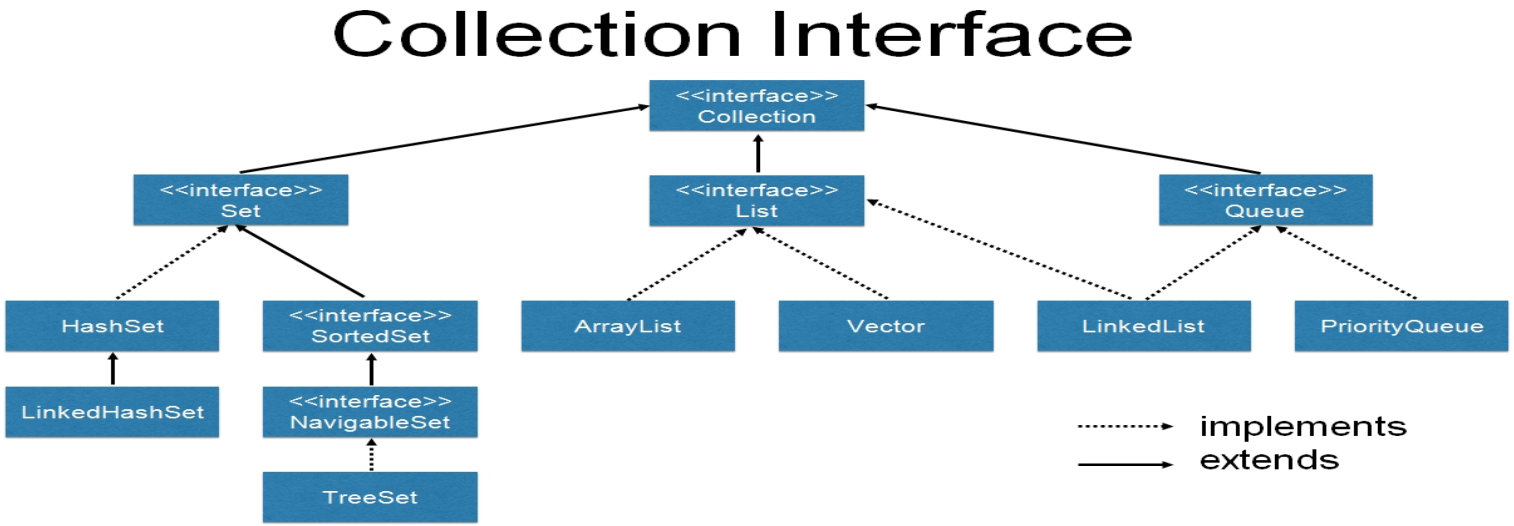
Interface Collection

les interfaces sont en vert, les classes abstraites en bleu et les classes en rouge, et T est le type paramètre des éléments des collections ; les lignes simples indiquent l'héritage et les lignes doubles l'implémentation. (Le schéma est partiel il existe d'autres classes).



Interface Collection<T>

- **Collection :**L'interface **Collection<T>** correspond à un objet qui contient un groupe d'objets de type **T.**
- Elle représente des méthodes communes pour les objets qui gèrent des collections (ajout d'élément ,suppression , vérification de présence….)



Collections :les méthodes(1)

• **boolean add(T e) :**ajoute l'élément fourni en paramètre à la collection. La valeur de retour indique si la collection a été émise à jour.

• **boolean addAll(Collection) :**ajoute à la collection tous les éléments de la collection fournie en paramètre.

• **void clear():** supprime tous les éléments de la collection.

• **boolean contains(T e):** indique si la collection contient au moins un élément identique à celui fourni en paramètre.

• **boolean containsAll(Collection)** indique si tous les éléments de la collection fournie en paramètre sont contenus dans la collection.

• **boolean isEmpty() :**indique si la collection est vide.

•**Iterator iterator() :**renvoie un objet qui permet de parcourir l'ensemble des élments de la collection.

Collections :les méthodes(2)

• **boolean remove(T e) :**supprime l'élément fourni en paramètre de la collection. La valeur de retour indique si la collection a été mise à jour.

• **boolean removeAll(Collection) :**supprime tous les éléments de la collection qui sont contenus dans la collection fournie en paramètre .

• **int size() :**renvoie le nombre d'éléments contenu dans la collection.

• **Object[] toArray() :**renvoie d'un tableau d'objets qui contient tous les éléments de la collection

int hashCode()

Remarque : en Java, chaque instance d'une classe a un hashCode fourni par la méthode hashCode() de la classe Object. Cette méthode associe à l'adresse mémoire de l'instance une valeur entière de type int.

La classe Collections

La classe **java.util.Collections** (notez le pluriel) contient des méthodes statiques qui opèrent sur des objets List ou autre (Set,Map ...) ou bien renvoie des objets.

- **void sort(List list) :**trie le paramètre list.
- **Object max(Collection coll):** renvoie le plus grand objet.
- **Object min(Collection coll):** renvoie le plus petit objet.

...

On peut utiliser ces méthodes statiques sur des objets de toutes les classes du schéma précédent.

L'interface Iterator

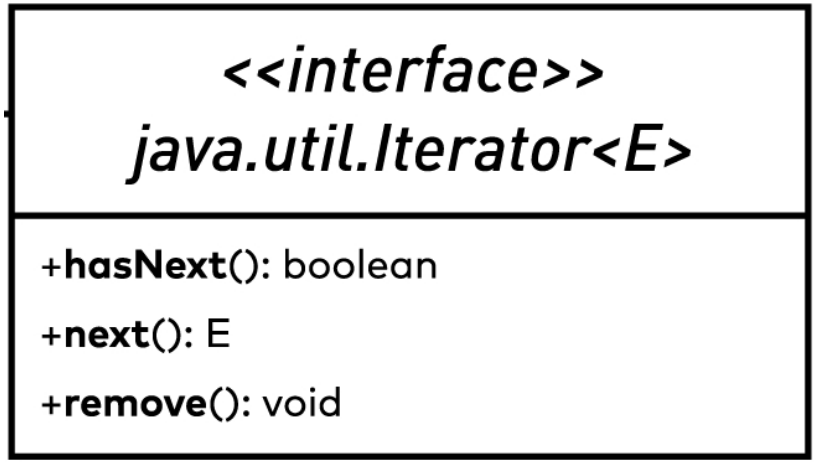
Un itérateur est un objet utilisé avec une collection pour fournir un accès séquentiel aux éléments de cette collection.

L'interface Iterator permet de fixer le comportement d'un itérateur.

- **boolean hasNext() :**indique s'il reste au moins un élément à parcourir dans la collection.
- **T next() :**renvoie le prochain élément dans la collection.
- **void remove() :**supprime le dernier élément parcouru (celui renvoyé par le dernier appel à la méthode next())

```
// parcourir les éléments de la collection avec
 // un itérateur
Iterator<String> it = collection.iterator() ;
 while (it.hasNext()) {

    String element = it.next() ;  // retourne un objet de type String
    System.out.println(element) ;
}
```

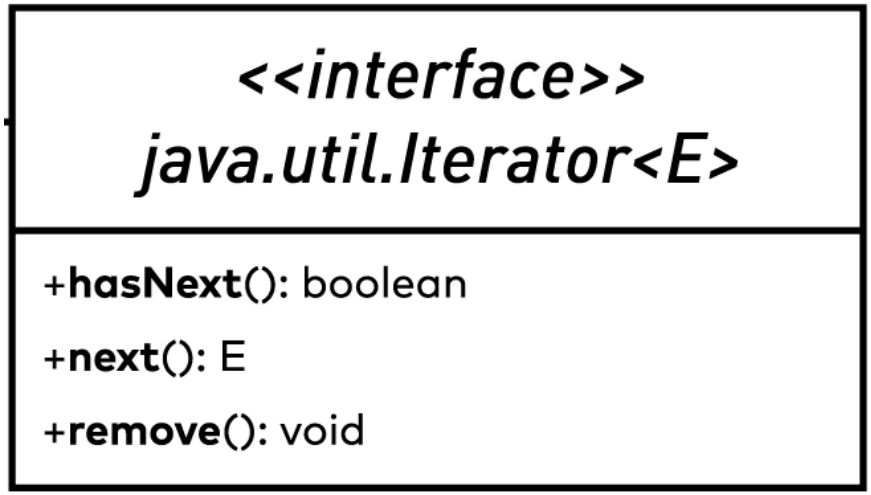


L'interface Iterator

• La méthode next() lève une exception de type NoSuchElementException si elle est appelée alors que la fin du parcours des éléments est atteinte.

• La méthode remove() lève une exception de type **IllegalStateException** si l'appel ne correspond à aucun appel à next(). Cette méthode est optionnelle (exception **UnsupportedOperationException**).

• On ne peut pas faire deux appels consécutifs à remove().



Remarques

- A sa construction un itérateur doit être lié à une collection.
- A sa construction un itérateur se place tout au début de la collection.
- On ne peut pas « réinitialiser » un itérateur pour parcourir de nouveau la collection il faut créer un nouvel itérateur.
- Java utilise un itérateur pour implémenter la boucle **for each** de syntaxe suivante

```
Collection<T> c = new ... ;
for (T element : c) {...} // pour chaque objet element de type T de ma collaction c faire
```

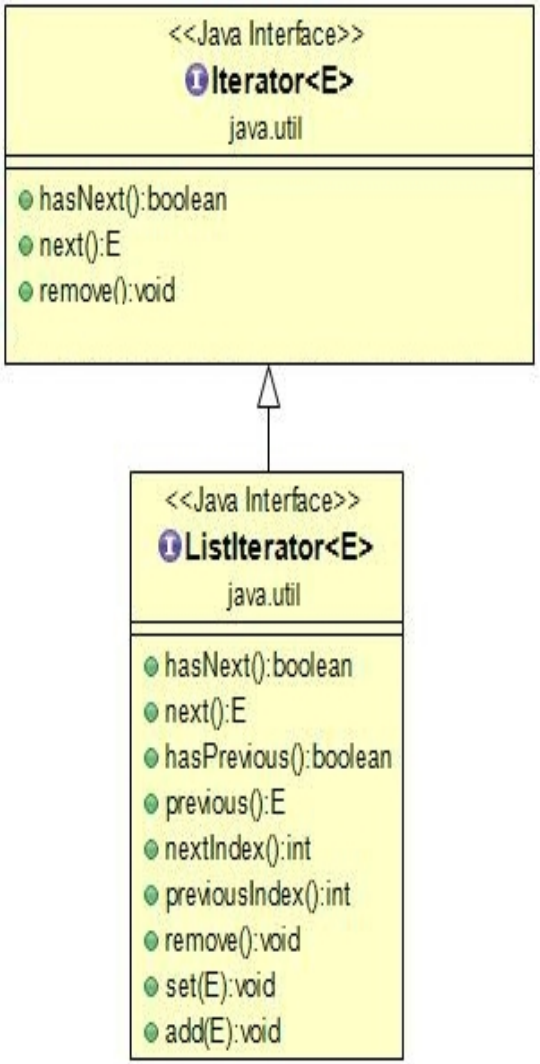
Collections:toString()

Pour tout objet de type Collection, la méthode print (ou println) appelle itérativement la méthode toString() de chacun de ses éléments.

Collections:Interface ListIterator(1)

L'interface ListIterator<T> étend l'interface Iterator<T> et permet de parcourir la collection (list) dans les deux sens.

- **T previous() :**renvoie l'élément précédent dans la collection
- **boolean hasPrevious() :**teste l'existence d'un élément précédent
- **T next() :**renvoie l'élément suivant de la liste
- **int nextIndex() :**renvoie l'indice de l'élément qui sera renvoyé au prochain appel de next()
- **int previousIndex():** renvoie l'indice de l'élément qui sera renvoyé au prochain appel de previous()

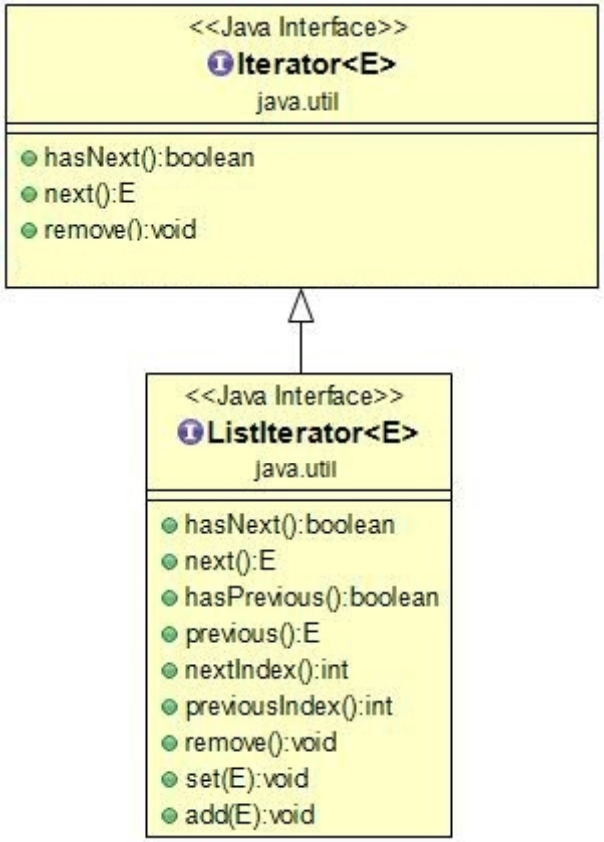


Collections:Interface ListIterator(2)

void add(T e): ajoute l'élément e à la liste à l'endroit du curseur (i.e. juste avant l'élément retourné par l'appel suivant à next())

void remove(): supprime le dernier élément retourné par next() ou previous()

void set(T e) :remplace le dernier élément retourné par next() ou previous() par e



Collections:Interface ListIterator

remarques :

• next() et previous() lèvent une exception de type NoSuchElementException

• si l'itérateur est en fin de liste alors nextIndex() renvoie la taille de la liste

• si l'itérateur est au début de la liste alors nextIndex() renvoie -1

• add(), remove() et set(T e) lèvent une exception de type IllegalStateException si l'appel ne correspond à aucun appel à next() ou previous(). Elles sont toutes les trois optionnelles.

•set(T e) lève une exception de type ClassCastException si le type de e ne convient pas.

•dans toutes les classes prédéfinies implémentant Iterator ou ListIterator, les méthodes next() et previous() renvoient les références des objets de la collection.

Les listes

L'interface List

List:

• Collection ordonnée selon l'ordre d'insertion

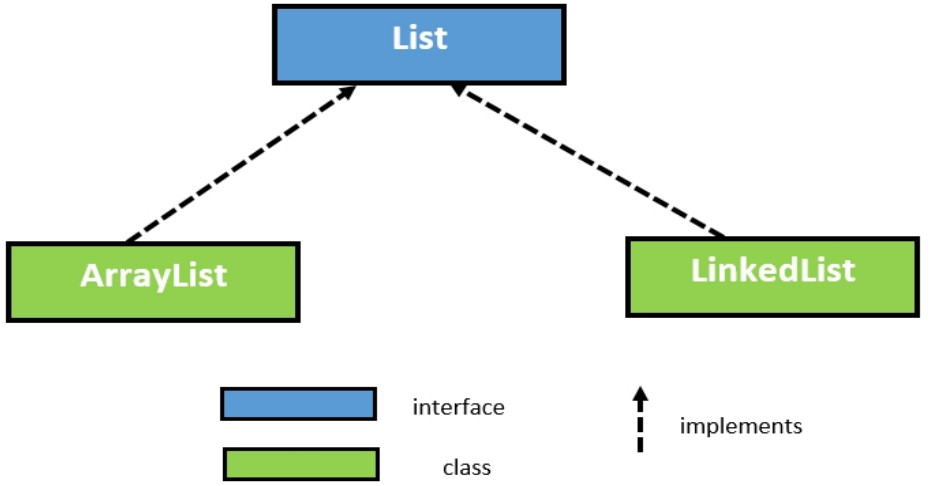
• Tableau dynamique extensible à volonté(pas de crainte de débordement).

• Accepte les doublons

• Accepte l'élément null.

• Pour les listes l'interface **ListIterator** est définie pour permettre le parcours dans les deux sens.

.



Classe ArrayList<T>

ArrayList:

•Les éléments d'un ArrayList sont accessibles par leur indice. L'index commence à partir de 0.

• Un ArrayList est un tableau d'objets dont la taille est dynamique.

• La classe ArrayList<T> implémente en particulier les interfaces Iterator, ListIterator et List.

Classe ArrayList<T>:Constructeurs

- **public ArrayList(int initialCapacite) :**crée un arrayList vide avec la capacité initialCapacite (positif)
- **public ArrayList():** crée un arrayList vide avec la capacité 10
- **public ArrayList(Collection<? extends T> c)** crée un arrayList contenant tous les éléments de la collection c dans le même ordre obtenu en utilisant son iterator.

Classe ArrayList<T>:Méthodes

- **add et addAll** ajoute à la fin du tableau
- **void add(int index, T element)** ajoute au tableau le paramètre élément à l'indice index en décalant d'un rang vers la droite les éléments du tableau d'indice supérieur
- **void ensureCapacity(int k)** permet d'augmenter la capacité du tableau pour s'assurer qu'il puisse contenir le nombre d'éléments passé en paramètre
- **T get(int index):** renvoie l'élément du tableau dont la position est précisée
- **T set(int index, T element)** renvoie l'élément à la position index et remplace sa valeur par celle du paramètre élément

Classe ArrayList<T>:Méthodes

- **int indexOf(Object o) :**renvoie la position de la première occurrence de l'élément fourni en paramètre
- **int lastIndexOf(Object o):** renvoie la position de la dernière occurrence de l'élément fourni en paramètre
- **T remove(int index) :**renvoie l'élément du tableau à l'indice index et le supprime décalant d'un rang vers la gauche les éléments d'indice supérieur
- **void removeRange(int j,int k)** supprime tous les éléments du tableau de la position j incluse jusqu'à la position k exclue
- **void trimToSize()** ajuste la capacité du tableau sur sa taille actuelle

Classe ArrayList<T>:Exemples

Exemple1 voir Exemple ArrayListExemple dans le document TutoCollections.

Exemple2:

On veut gérer un ensemble de films connaissant leurs rating, name et year par ordre croissant de date de sortie.

On définit:

- la classe **Movie** (rating,name,year) qui implémente l'interface Comparable
- la classe Main qui utilise ArrayList.

Interface Comparable<T>

• Cette interface correspond à l'implantation d'un ordre naturel dans les instances d'une classe .

• Elle ne contient qu'une seule méthode :**int compareTo(T o)**

```
public interface Comparable<T> {
  public int compareTo(T o);
}
```

•Cette méthode renvoie:

- ✓ un entier positif si l'objet qui reçoit le message (this)est plus grand que l'objet o.
- ✓ 0 si les deux objets ont la même valeur.
- ✓ un entier négatif si l'objet qui reçoit le message est plus petit que l'objet o.

Classe ArrayList<T>:Exemple


```
import java.io.*;
import java.util.*;

//A class 'Movie' that implements Comparable
class Movie implements Comparable<Movie>
{
 private double rating;
 private String name;
 private int year;

 // Used to sort movies by year
 public int compareTo(Movie m)
 {
     return this.year - m.year;
 }

 // Constructor
 public Movie(String nm, double rt, int yr)
 {
     this.name = nm;
     this.rating = rt;
     this.year = yr;
 }

 // Getter methods for accessing private data
 public double getRating() { return rating; }
 public String getName()   {  return name; }
 public int getYear()      {  return year;  }
}
```

```
class Main
{
 public static void main(String[] args)
 {
     ArrayList<Movie> list = new ArrayList<Movie>();
     list.add(new Movie("Force Awakens", 8.3, 2015));
     list.add(new Movie("Star Wars", 8.7, 1977));
     list.add(new Movie("Empire Strikes Back", 8.8, 1980));
     list.add(new Movie("Return of the Jedi", 8.4, 1983));

     Collections.sort(list);

     System.out.println("Movies after sorting : ");
     for (Movie movie: list)
     {
         System.out.println(movie.getName() + " " +
                            movie.getRating() + " " +
                            movie.getYear());
     }
 }
```

```
Movies after sorting :
Star Wars 8.7 1977
Empire Strikes Back 8.8 1980
Return of the Jedi 8.4 1983
Force Awakens 8.3 2015
```

Interface Comparator

Supposons maintenant que nous voulons également trier les films par leur classement et leur nom. Lorsque nous rendons un élément de collection comparable (en l'implémentant Comparable), nous n'avons qu'une seule chance d'implémenter la méthode compareTo (). La solution utilise **Comparator.**

•Pour comparer les films par rating, nous devons faire 3 choses:
Créez une classe qui implémente Comparator (et donc la méthode compare () qui effectue le travail précédemment effectué par compareTo ()).

•Créez une instance de la classe Comparator.

•Appelez la méthode sort () surchargée, en lui donnant à la fois la liste et l'instance de la classe qui implémente Comparator.

Interface Comparator:Exemple


```java
import java.io.*;
import java.util.*;

// A class 'Movie' that implements Comparable
class Movie implements Comparable<Movie>
{
    private double rating;
    private String name;
    private int year;

    // Used to sort movies by year
    public int compareTo(Movie m)
    {
        return this.year - m.year;
    }

    // Constructor
    public Movie(String nm, double rt, int yr)
    {
        this.name = nm;
        this.rating = rt;
        this.year = yr;
    }

    // Getter methods for accessing private data
    public double getRating() { return rating; }
    public String getName() { return name; }
    public int getYear()     { return year; }
}
```

```java
// Class to compare Movies by ratings
class RatingCompare implements Comparator<Movie>
{
    public int compare(Movie m1, Movie m2)
    {
        if (m1.getRating() < m2.getRating()) return -1;
        if (m1.getRating() > m2.getRating()) return 1;
        else return 0;
    }
}

// Class to compare Movies by name
class NameCompare implements Comparator<Movie>
{
    public int compare(Movie m1, Movie m2)
    {
        return m1.getName().compareTo(m2.getName());
    }
}
```

```java
class Main
{
    public static void main(String[] args)
    {
        ArrayList<Movie> list = new ArrayList<Movie>();
        list.add(new Movie("Force Awakens", 8.3, 2015));
        list.add(new Movie("Star Wars", 8.7, 1977));
        list.add(new Movie("Empire Strikes Back", 8.8, 1980));
        list.add(new Movie("Return of the Jedi", 8.4, 1983));

        // Sort by rating : (1) Create an object of ratingCompare
        //                  (2) Call Colle
        //                  (3) Print Sort
        System.out.println("Sorted by r
        RatingCompare ratingCompare = r
        Collections.sort(list, ratingC
        for (Movie movie: list)
            System.out.println(movie.ge
                               movie.getNa
                               movie.getYe
```

```
Sorted by rating
8.3 Force Awakens 2015
8.4 Return of the Jedi 1983
8.7 Star Wars 1977
8.8 Empire Strikes Back 1980
```

Classe LinkedList<T>

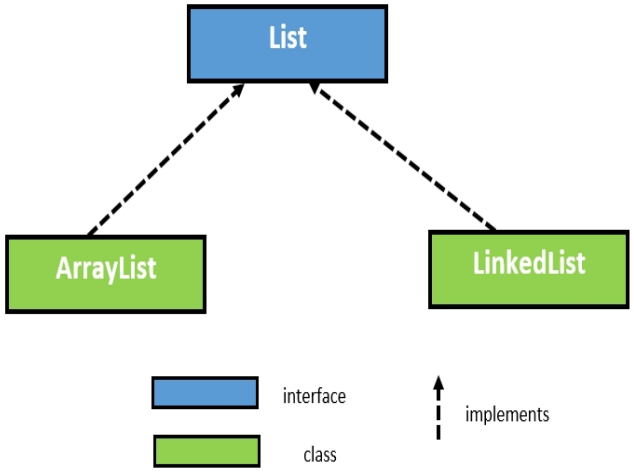
LinkedList

-Est une implémentation d'une liste doublement chainée dans laquelle les éléments sont reliés par des **pointeurs.**

- liste ordonnée

-Peut être parcouru par un ListIterator qui permet de parcourir la liste dans les deux sens.

- Exemple d'utilisation :pile,queue(file d'attente)…



Classe LinkedList<T>(Constructeurs)

La classe LinkedList supporte deux constructeurs:

-**LinkedList()** Ce constructeur crée une LinkedList vide.

-**LinkedList(Collection c)** Ce constructeur crée une LinkedList initialisée avec une collection de données.

-**Exemple:** voir Exemple LinkedListExemple dans le document TutoCollections.

Les ensembles(Set)

Set : HashSet , LinkedHashSet et TreeSet

• Les Set sont des collections qui n'acceptent pas les doublons

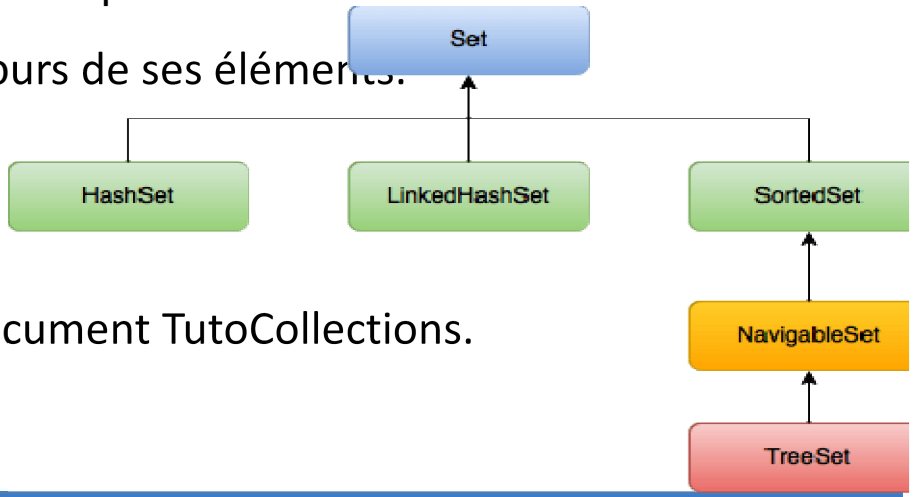
• **HashSet** ne propose aucune garantie sur l'ordre de parcours lors de l'itération sur les éléments qu'elle contient (non ordonnée),elle autorise les doublons et l'ajout de l'élément null

• **LinkedHashSet** est un Set qui maintient les clés dans l'ordre d'insertion

• **TreeSet** garantit que les éléments sont rangés dans leur ordre naturel **(interface Comparable)** ou l'ordre d'un **Comparator**,n'autorise pas les doublons

• **SortedSet:** garantit l'ordre ascendant du parcours de ses éléments.

•



•**Exemple:** voir Exemple SetExemple dans le document TutoCollections.

Les associations de type clé/valeur: Map

L'interface Map<k,V>

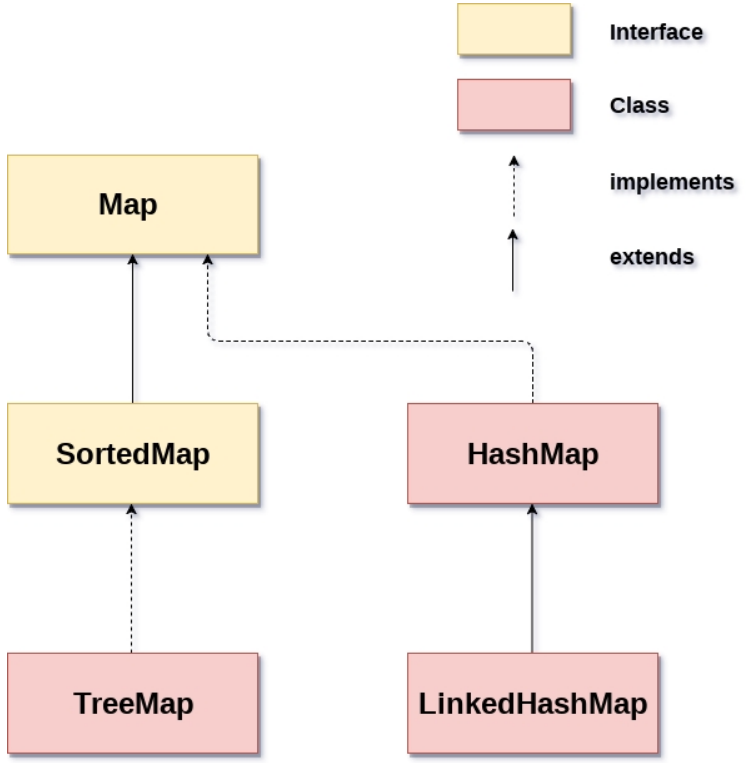
Un objet de type Map stocke des couples clef-valeur. On parle aussi de dictionnaire.

La clé doit être unique mais une valeur peut avoir plusieurs clés.

L'interface Map<K,V> fixe les méthodes pour manipuler de tels couples. Les clés sont de type K et sont associées aux valeurs de type V.

Toutes les collections de couples clef-valeur

- sont dans le paquetage java.util
- implémentent l'interface générique Map<K,V>

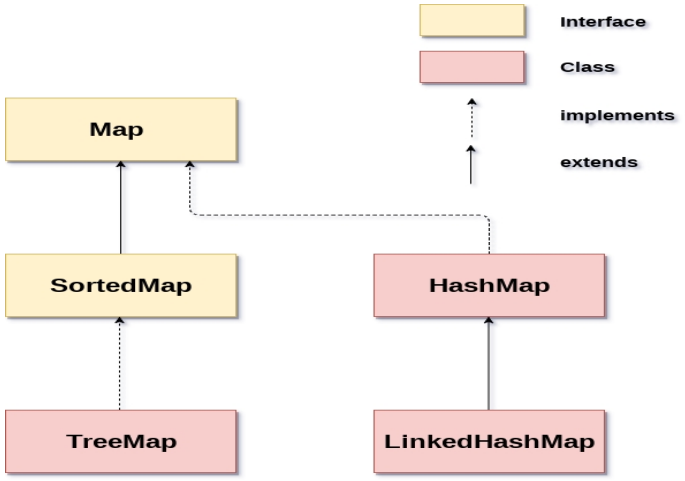


Les implémentations

La classe HashMap<K,V> implémente l'interface Map<K,V> avec une table de hachage .

La classe TreeMap<K,V> implémente l'interface Map<K,V> qui stocke les éléments avec un ordre naturel sur les clés ou bien avec un Comparator sur les clés,cette dernière classe implémente l'interface SortedMap

La classe LinkedHashMap<K,V>implémentation qui combine table de hachage et les liens chainées pour stocker ses éléments.



Interface Map<K,V>:Méthodes

- **V put(K clef, V valeur)** associe valeur à clé. Si clé est déjà associée à un objet V alors il est remplacé par le paramètre valeur puis il est renvoyé (null est renvoyé si clé était associée à rien)
- **V get(Object clef)** renvoie la valeur associée à clé si elle existe ou null sinon.
- **boolean containsKey(Object clef)** indique si un élément est associée au paramètre clé
- **boolean containsValue(Object valeur)** indique si le paramètre valeur est associée à au moins une clé
- **boolean isEmpty()** indique si la collection est vide
- **Set<K> keySet()** renvoie un ensemble constitué des clés de l'objet
- **Collection<V> values()** renvoie la collection constituée des valeurs de l'objet
- **Map.Entry<k,v>:**interface interne pour le parcours d'un ensemble de paire clé/valeur
- **Set<Map.Entry<K,V> > entrySet()** renvoie un ensemble constitué des couples (clefs,valeurs)

Se référer à l'API Java pour voir toutes les méthodes.