جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Chapitre 7

Les Collections en java

S. BOUKHEDOUMA

USTHB – FEI – département d'Informatique
Laboratoire des Systèmes Informatiques -LSI
sboukhedouma@usthb.dz

Collections?

Les objets de type collections sont des objets qui **peuvent stocker des quantités arbitraires (inconnus à priori et non limitées)** d'autres objets.

Les premières classes de Collections:

Vector, Array, Stack

Premières classes de Collections

Vector, Array : vecteur d'objets de taille variable et illimitée

Méthodes usuelles: ***get, elementAt, add, remove, size, …***

Stack: représente une pile d'objets

Méthodes usuelles: ***push, pop, isEmpty, …***

Toutes les classes des collections sont définies dans **java.util**

Le Framework des Collections

A partir de la version 5 du Jdk, le framework des collections est organisé en **deux grandes interfaces**

- L'interface **Collection**: permet de définir *des* ***listes, des tableaux, des ensembles, des files,*** ...
- L'interface **Map** : permet de définir des ***tableaux associatifs*** dont les objets sont de type **(clé, valeur)**

A partir de ces deux interfaces sont dérivées d'autres interfaces, des classes abstraites jusqu'à arriver aux classes concrètes (instanciables)

Le Framework des Collections

Le framework des collections est organisé selon une hiérarchie composée de

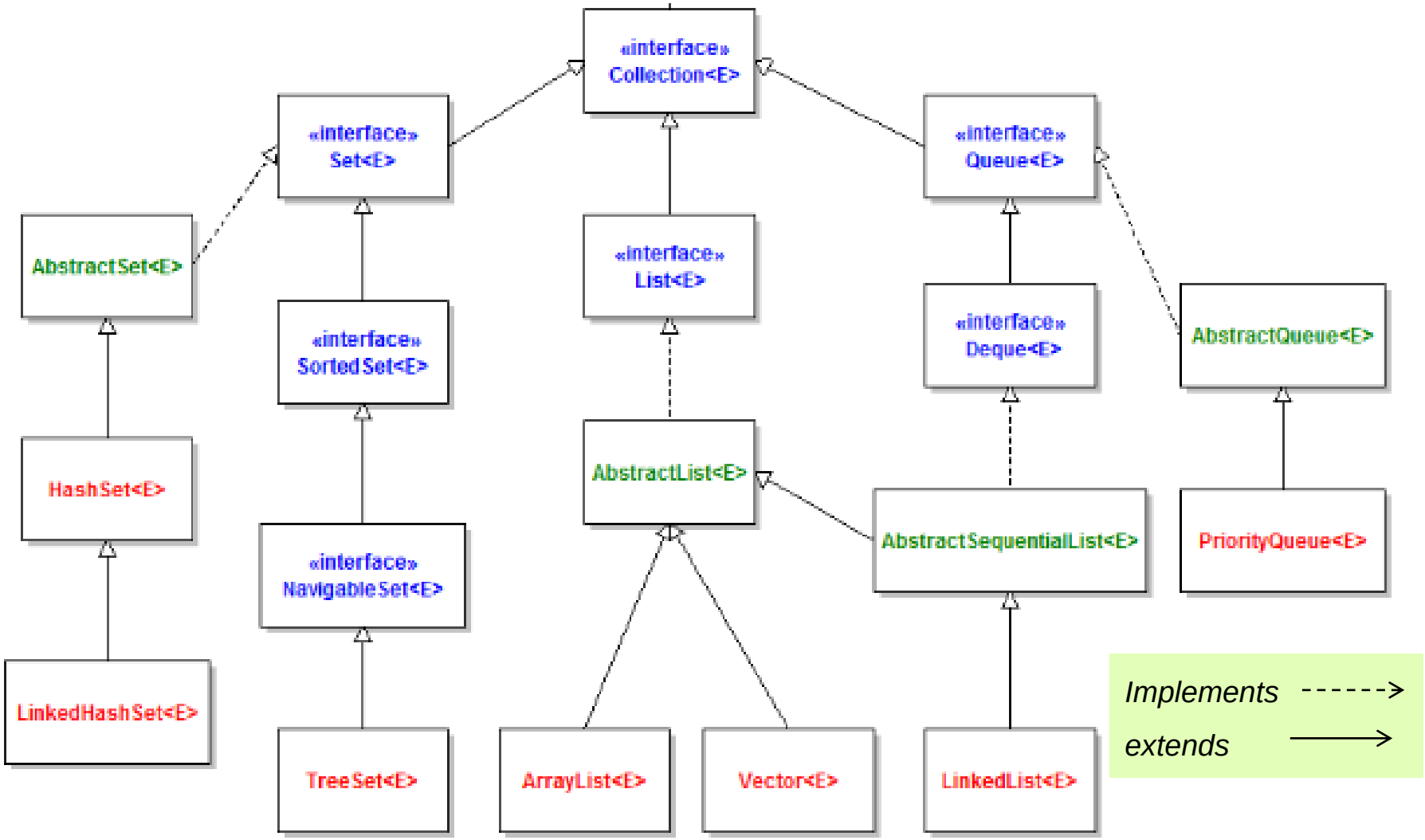
Interfaces : Collection, Map, Set, SortedSet, List, …

Classes abstraites: définies à partir d'interfaces AbstractList, AbstractSequentialList, AbstractSet, …

Classes concrètes: complètement implémentées et peuvent être instanciées – **ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap, …**

L'interface « Collection »

Le Framework des Collections – L'interface « Collection »



Le Framework des Collections – L'interface « Collection »

Les interfaces

Collection, Set, SortedSet, NavigableSet, List, Queue, Decque

Les classes abstraites

AbstractSet, AbstractList, AbstractSequentialList, AbstractQueue

Les classes implémentées

A partir de l'interface « Set »: ***HashSet, TreeSet, LinkedTreeSet,***

A partir de l'interface « List »: ***Vector, ArrayList, LinkedList***

A partir de l'interface « Queue » : ***PriorityQueue***

L'interface **« Collection »:** comporte les signatures de méthodes suivantes

Méthode	Rôle
boolean add(E e)	Ajouter un élément à la collection (optionnelle)
boolean addAll(Collection<? extends E> c)	Ajouter tous les éléments de la collection fournie en paramètre dans la collection (optionnelle)
void clear()	Supprimer tous les éléments de la collection (optionnelle)
boolean contains(Object o)	Retourner un booléen qui précise si l'élément est présent dans la collection
boolean containsAll(Collection<?> c)	Retourner un booléen qui précise si tous les éléments fournis en paramètres sont présents dans la collection
boolean equals(Object o)	Vérifier l'égalité avec la collection fournie en paramètre
int hashCode()	Retourner la valeur de hachage de la collection
boolean isEmpty()	Retourner un booléen qui précise si la collection est vide
Iterator<E> iterator()	Retourner un Iterator qui permet le parcours des éléments de la collection

Le Framework des Collections

L'interface « Collection »: suite des signatures de méthodes

boolean remove(Object o)	Supprimer un élément de la collection s'il est présent (optionnelle)
boolean removeAll(Collection<?> c)	Supprimer tous les éléments fournis en paramètres de la collection s'ils sont présents (optionnelle)
boolean retainAll(Collection<?> c)	Ne laisser dans la collection que les éléments fournis en paramètres : les autres éléments sont supprimés (optionnelle). Elle renvoie un booléen qui précise si le contenu de la collection a été modifié
int size()	Retourner le nombre d'éléments contenus dans la collection
Object[] toArray()	Retourner un tableau contenant tous les éléments de la collection
<T> T[] toArray(T[] a)	Retourner un tableau typé de tous les éléments de la collection

L'interface « Collection » : représente un *minimum commun* pour les objets qui gèrent des collections comme l'ajout d'éléments, suppression d'éléments, recherche d'un élément dans la collection, parcours de la collection et quelques opérations diverses sur la totalité de la collection.

L'interface « List »

Les Collections – l'interface « List »

L'interface « List »

interface List extends Collection {…}

Une collection de type List est une collection où on considère **la notion d'ordre entre les éléments** (1er, 2ème, …) à travers la manipulation **d'index** (indice représentant le rang de l'élément dans la collection)

Dans une collection de type List, on accepte aussi **les doublons (les répétitions d'éléments).**

La **notion d'index** est utilisée dans les méthodes de cette interface:

get (index),

add (index, E),

addAll(index, Collect),

remove (index),

set (index, E)

…

Les Collections – l'interface « List »

L'interface **« List »:** **hérite** de toutes les signatures de l'interface Collection et comporte **en plus** les signatures de méthodes suivantes:

Méthode	Rôle
void add(int index, E e)	Ajouter un élément à la position fournie en paramètre
boolean addAll(int index, Collection<? extends E> c)	Ajouter des éléments à la position fournie en paramètre
E get(int index)	Retourner l'élément à la position fournie en paramètre
int indexOf(Object o)	Retourner la première position dans la liste du premier élément fourni en paramètre. Elle renvoie -1 si l'élément n'est pas trouvé
int lastIndexOf(Object o)	Retourner la dernière position dans la liste du premier élément fourni en paramètre. Elle renvoie -1 si l'élément n'est pas trouvé
ListIterator<E> listIterator()	Renvoyer un Iterator positionné sur le premier élément de la liste
ListIterator<E> listIterator(int indx)	Renvoyer un Iterator positionné sur l'élément dont l'index est fourni en paramètre
E remove(int index)	Supprimer l'élément à la position fournie en paramètre
E set(int index, E e)	Remplacer l'élément à la position fournie en paramètre
List<E> subList(int fromIndex, int toIndex)	Obtenir une liste partielle de la collection contenant les éléments compris entre les index fromIndex inclus et toIndex exclus fournis en paramètres

Les Collections – l'interface « List »

Les implémentations de l'interface **« List »** (*le lien implements*)

Les classes implémentées : Vector, ArrayList, LinkedList, Stack.

Implémentation	Rôle
java.util.Vector<E>	Une implémentation thread-safe fournie depuis Java 1.0
java.util.Stack<E>	Une implémentation d'une pile : elle hérite de la classe Vector et fournit des opérations pour un comportement de type LIFO (Last In First Out)
java.util.ArrayList<E>	Une implémentation qui n'est pas synchronized, donc à n'utiliser que dans un contexte monothread
java.util.LinkedList<E>	Une implémentation qui n'est pas synchronized d'une liste doublement chaînée. Les insertions de nouveaux éléments sont très rapides
java.util.concurrent.CopyOnWriteArrayList<E>	Une variante thread-safe de la classe ArrayList dans laquelle toutes les opérations de modification du contenu de la liste recréent une nouvelle copie du tableau utilisé pour stocker les éléments de la collection

Les Collections – l'interface « List »

Les implémentations de l'interface **« List »** (*le lien implements*)

La classe ArrayList

représente un tableau (indicé) d'une taille variable

public abstract class AbstractList implements List {…
}
public class ArrayList extends AbstractList {…
//implémentation de toutes les méthodes de List}

Constructeur	Rôle
ArrayList()	Créer une instance vide de la collection avec une capacité initiale de 10
ArrayList(Collection<? extends E> c)	Créer une instance contenant les éléments de la collection fournie en paramètre dans l'ordre obtenu en utilisant son iterator
ArrayList(int initialCapacity)	Créer une instance vide de la collection avec la capacité initiale fournie en paramètre

Les Collections – l'interface « List »

Les implémentations de l'interface **« List »** (*le lien implements*)

La classe ArrayList

Les principales méthodes implémentées dans la classe ArrayList sont

Méthode	Rôle
boolean add(Object)	Ajouter un élément à la fin du tableau
boolean addAll(Collection)	Ajouter tous les éléments de la collection fournie en paramètre à la fin du tableau
boolean addAll(int, Collection)	Ajouter tous les éléments de la collection fournie en paramètre dans la collection à partir de la position précisée
void clear()	Supprimer tous les éléments du tableau
void ensureCapacity(int)	Augmenter la capacité du tableau pour s'assurer qu'il puisse contenir le nombre d'éléments passé en paramètre
Object get(index)	Renvoyer l'élément du tableau dont la position est précisée
int indexOf(Object)	Renvoyer la position de la première occurrence de l'élément fourni en paramètre
boolean isEmpty()	Indiquer si le tableau est vide
int lastIndexOf(Object)	Renvoyer la position de la dernière occurrence de l'élément fourni en paramètre
Object remove(int)	Supprimer dans le tableau l'élément fourni en paramètre
void removeRange(int, int)	Supprimer tous les éléments du tableau de la première position fournie incluse jusqu'à la dernière position fournie exclue
Object set(int, Object)	Remplacer l'élément à la position indiquée par celui fourni en paramètre
int size()	Renvoyer le nombre d'éléments du tableau
void trimToSize()	Ajuster la capacité du tableau sur sa taille actuelle

Les Collections – l'interface « List »

Méthode	Rôle
boolean add(Object)	Ajouter un élément à la fin du tableau
boolean addAll(Collection)	Ajouter tous les éléments de la collection fournie en paramètre à la fin du tableau
boolean addAll(int, Collection)	Ajouter tous les éléments de la collection fournie en paramètre dans la collection à partir de la position précisée
void clear()	Supprimer tous les éléments du tableau
void ensureCapacity(int)	Augmenter la capacité du tableau pour s'assurer qu'il puisse contenir le nombre d'éléments passé en paramètre
Object get(index)	Renvoyer l'élément du tableau dont la position est précisée
int indexOf(Object)	Renvoyer la position de la première occurrence de l'élément fourni en paramètre
boolean isEmpty()	Indiquer si le tableau est vide
int lastIndexOf(Object)	Renvoyer la position de la dernière occurrence de l'élément fourni en paramètre
Object remove(int)	Supprimer dans le tableau l'élément fourni en paramètre
void removeRange(int, int)	Supprimer tous les éléments du tableau de la première position fournie incluse jusqu'à la dernière position fournie exclue
Object set(int, Object)	Remplacer l'élément à la position indiquée par celui fourni en paramètre
int size()	Renvoyer le nombre d'éléments du tableau
void trimToSize()	Ajuster la capacité du tableau sur sa taille actuelle

Les Collections – l'interface « List »

La classe ArrayList

Pour l'utiliser : import java.util.ArrayList;

Exemple

```
ArrayList <String>  A = new ArrayList <String> ();
A.add ("Hello" ); A.add ("good" );
A.add ("morning" ); A.add ("every body" );

ArrayList <String>  B = new ArrayList <String> (A);
                                    // on met dans B tous les éléments de A.
```

Equivalent à écrire:

```
ArrayList <String>  B = new ArrayList <String> (); // créer une collection vide
B.addAll(A);        // ajouter tous les éléments de A à B
```

Les Collections – l'interface « List »

Les implémentations de l'interface **« List »** (*le lien implements*)

La classe ArrayList

Exemple (suite)

modification

```
String s = A.set (2, "afternoon" );    //remplacer l'élément d'indice 2 par "afternoon"
```

suppression

```
String s = A.remove (0);          // supprimer l'élément d'indice 0
```

Affichage (parcours)

```
for (int i = 0; i< A.size(); i++) System.out.print (A.get(i) + "\t");
```

ou bien for **(String s: A)** System.out.print (**s** + "\t"); *//s est un objet qui parcourt les éléments de A*

ou bien System.out.print (**A**);

Les Collections – l'interface « List »

Les implémentations de l'interface **« List »** (*le lien implements*)

La classe ArrayList

Exemples (suite)

Extraction d'une sous-liste

```
List <String> SL = A.subList (0,2)
        // crée une sous-liste de A des indices 0 inclu jusqu'à 2 non inclu
```

Suppression d'un ensemble d'éléments

```
A.removeRange (0, 2);   // supprime les éléments des indices 0 inclus jusqu'à 2 non inclu
```

Changer capacité du tableau

```
A. ensureCapacity (50);
```

Les Collections – l'interface « List »

Les implémentations de l'interface **« List »** (*le lien implements*)

La classe ArrayList

Autre exemple

```
ArrayList <Point>  C = new ArrayList <Point> ();

C.add (new Point(3, 6));        C.add ( new Point (-2, 0));
C.add ( new Point (1, 1));

System.out.println (C);
for (int i = 0; i< C.size(); i++) System.out.print (C.get(i) + "\t");
```

ou bien for **(Point p: C)** System.out.print (**p** + "\t"); *//p est un objet qui parcourt les éléments de C*

On aura:

(3,6) (-2,0) (1, 1) *// en utilisant la méthode toString de la classe Point*

Les Collections – l'interface « List »

Parcours d'une ArrayList avec un itérateur (iterator)

Un **itérateur (iterator)** est un objet défini pour les collections itérables, qui permet de parcourir les éléments d'une collection (itérable) comme une liste, en séquentiel en utilisant des méthodes spécifiques next, hasNext

Syntaxe

```
Iterator <Type> nom-itérateur = nom-Collection.iterator();
```

Exemple:

```
Iterator <String> it = A.iterator();
```

L'itétateur *it* va parcourir les éléments de la Arraylist *A*

Les Collections – l'interface « List »

Parcours d'une ArrayList avec un itérateur (iterator)

Exemple:

```
Iterator <String> it = A.iterator();
        // L'itétateur it va parcourir les éléments de la Arraylist A

        While (it.hasNext())
                System.out.print (it.next() + "\t ");

// Tant qu'il ya un élément dans A, il faut afficher en faisant un parcours séquentiel
```

Les Collections – l'interface « List »

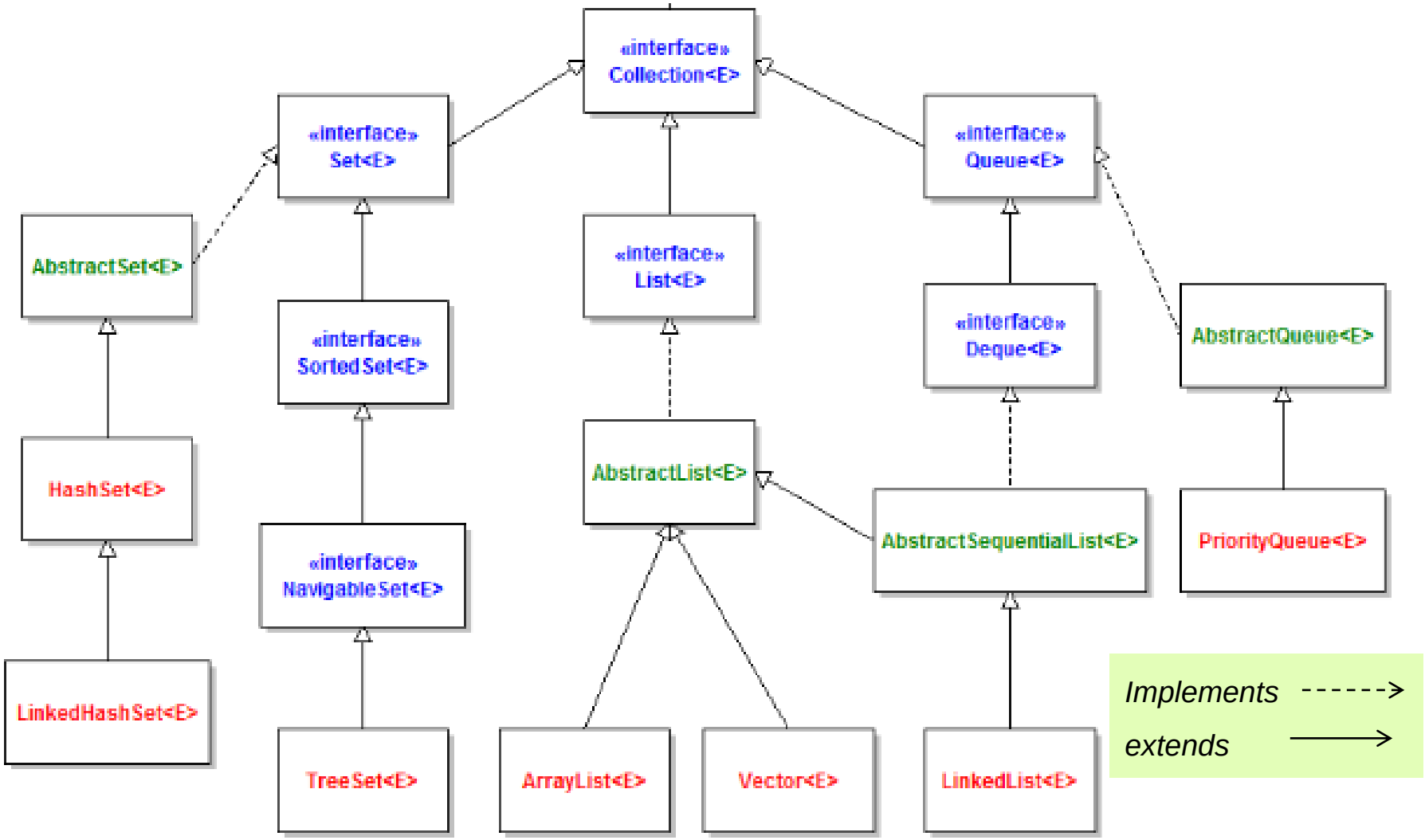
Mettre le contenu d'un tableau [] dans une ArrayList

```
import java.util.ArrayList; import java.util.Collections;
import java.util.List;

public class ArrayToArrayList
{ public static void main(final String[] args) {
String[] tab = { "AA", "BB", "CC", "DD" };
List<String> AL = new ArrayList<String>();
Collections.addAll (AL, tab);        // addAll:  méthode static de la classe
Collections
System.out.println("Contenu du tableau");
for (String s : tab) { System.out.print(" " + s); }
System.out.println("\nContenu de la liste");
for (String str : AL) { System.out.print(" " + str); }
```

L'interface « List »
La classe « LinkedList »

Le Framework des Collections



Les Collections – La classe « LinkedList »

Une autre implémentation de l'interface **« List »** (*le lien implements*)

La classe LinkedList

représente une liste doublement chainée dynamique

public abstract class AbstractList implements List {…
}
public abstract class AbstractSequentialList extends AbstractList implements Deque {…
}
public class LinkedList extends AbstractSequentialList {…
//implémentation de toutes les méthodes de List}

Constructeur	Rôle
LinkedList()	Créer une nouvelle instance vide
LinkedList(Collection<? extends E> c)	Créer une nouvelle instance contenant les éléments de la collection fournie en paramètre triés dans l'ordre obtenu par son Iterator

Les Collections – La classe « LinkedList »

Méthodes spécifiques à une LinkedList pour:

l'accès/ ajout/ suppression ***au début ou à la fin*** *de la liste*

Méthode	Rôle
void addFirst(Object)	Insèrer l'objet au début de la liste
void addLast(Object)	Insèrer l'objet à la fin de la liste
Object getFirst()	Renvoyer le premier élément de la liste
Object getLast()	Renvoyer le dernier élément de la liste
Object removeFirst()	Supprimer le premier élément de la liste et renvoie l'élément qui est devenu le premier
Object removeLast()	Supprimer le dernier élément de la liste et renvoie l'élément qui est devenu le dernier

Les autres méthodes de l'interface Liste sont aussi implémentées **get(i), subList(), remove(i), size(), removeRange (…), …**

La classe LinkedList

Pour l'utiliser : import java.util.LinkedList;

Exemple

```
ArrayList <String>  A = new ArrayList <String> ();
A.add ("Hello" ); A.add ("good" );
A.add ("morning" ); A.add ("every body" );

LinkedList <String>  L = new LinkedList <String> ();
L.add ("premier" ); L.add ("second" );
L.add ("troisième" ); L.add ("quatrième" );
L.addAll (A);    // ajouter tous les éléments de la ArrayList A dans L
L.add ("STOP" );
```

La classe LinkedList

Pour l'utiliser : import java.util.LinkedList;

Exemple (suite)

```
// affichage
for (int i = 0; i< L.size(); i++) System.out.print (L.get(i) + "\t");
ou bien for (String s: L) System.out.print (s +"\t");
                    //s est un objet qui parcourt les éléments de L
ou bien System.out.print (L);
```

On aura:

Premier second troisième quatrième Hello good morning every body STOP

Les Collections – l'interface « List »

La classe LinkedList

Pour l'utiliser : import java.util.LinkedList;

Exemple (suite)

```
// suppression
L.removeRange(2, 4);
L.addFirst ( "DEBUT");  L.removeLast();  L.addLast ("FIN");
// affichage
for (String s: L) System.out.print (s +"\t");
```

On aura:

DEBUT premier second Hello good morning every body FIN

Les Collections – l'interface « List »

Parcours d'une LinkedList avec un itérateur (iterator)

Exemple:

```
Iterator <String> it = L.iterator();
        // L'itétateur it va parcourir les éléments de la LinkedList L

        While (it.hasNext())
                System.out.print (it.next() + "\t ");

// Tant qu'il ya un élément dans L, il faut afficher en faisant un parcours séquentiel
```

Les Collections – La classe « LinkedList »

ListIterator: hérite de l'interface Iterator, c'est un itérateur spécifique à la LinkedList, définit des fonctionnalités d'un Iterator permettant aussi le parcours en sens inverse (de la fin vers le début) de la collection, l'ajout d'un élément ou la modification de l'élément courant.

Méthode	Rôle
void add(E e)	Ajouter un élément dans la collection
boolean hasPrevious()	Retourner true si l'élément courant possède un élément précédent
int nextIndex()	Retourner l'index de l'élément qui serait retourné en invoquant la méthode next()
E previous()	Retourner l'élément précédent dans la liste
int previousIndex()	Retourner l'index de l'élément qui serait retourné en invoquant la méthode previous()
void set(E e)	Remplacer l'élément courant par celui fourni en paramètre

L'interface ListIterator hérite aussi des méthodes hasNext et next de Iterator

Les Collections – l'interface « List »

Parcours d'une LinkedList avec un **ListItérator**

Exemple:

```
ListIterator <String> it2 = L.listIterator();
        // L'itétateur it2 va parcourir les éléments de la LinkedList L

        While (it2.hasNext())
                System.out.print (it2.next() + "\t ");

        // parcours dans le sens inverse
        While (it2.hasPrevious())
                System.ou t.print (it2.previous() + "\t ");
```

On aura:

DEBUT premier second Hello good morning every body FIN
FIN every body morning good Hello second premier DEBUT

Les Collections – l'interface « List »

Parcours d'une LinkedList avec un **ListItérator**

Exemple:

```
ListIterator <String> it2 = L.listIterator();
                                        // modification
        While (it2.hasNext())
                If  (it2.next().equals ("morning"))
                    it2.set("afternoon");
                                        // modifier l'élément courant
        While (it2.hasPrevious())
                System.ou t.print (it2.previous() + "\t ");
        While (it2.hasNext())
                System.ou t.print (it2.next() + "\t ");
```

On aura:

afternoon good Hello second premier DEBUT
DEBUT premier second Hello good afternoon every body FIN

Les Collections – l'interface « List »

Parcours d'une LinkedList avec un **ListItérator**

Remarque: Il est possible de positionner l'itérateur à un index donné de la liste

Exemple:

```
ListIterator <String> it = L.listIterator(3);
                                    // à partir de l'indice 3
        While (it.hasNext())
                System.ou t.print (it.next() + "\t ");
```

On aura:

Hello good afternoon every body FIN

```
ListIterator <String> it = L.listIterator(L.size()-1);
                                    // place l'itérateur à la fin de la liste
```

Tri d'une collection de type List (« ArrayList » ou « LinkedList »)

Il est possible de trier une liste d'objets en utilisant la méthode static sort de la classe Collections définie dans java.util

Exemple

```
import java.util.Collections;
import java.util.LinkedList;
LinkedList <String>  L = new LinkedList <String> ();
L.add (”premier” ); L.add (”second" );
L.add (”troisième” ); L.add (”quatrième” ); L.add (”dernier”); L.add (”stop”);

Collections.sort(L);    System.out.printn (L);
Collections.reverse (L);  // inverser l'ordre     System.out.printn (L);
```

On aura:

dernier premier quatrième second stop troisième
troisième stop second quatrième premier dernier

Les différences entre « ArrayList » et « LinkedList »

- une ArrayList stocke ses éléments en interne dans un tableau de taille fixe alors qu'une LinkedList stocke ses éléments dans une liste doublement chaînée.

- une ArrayList permet un accès direct à un élément alors qu'une LinkedList doit parcourir ses éléments (accès séquentiel) pour obtenir celui désiré, ce qui est particulièrement moins performant

- le coût de variation de la capacité (augmentation de taille) d'une collection de type ArrayList est important car il implique une copie du tableau de stockage interne de ses éléments

- l'ajout/suppression d'un élément en début ou en fin d'une collection de type LinkedList est particulièrement performant et son temps d'exécution, il est constant dans le temps

Le choix d'une implémentation de type « List »

-Si l'ajout ou la suppression d'éléments se font essentiellement à la fin de la collection, alors il faut utiliser la classe ArrayList

-Si les ajouts ou la suppression d'éléments se font à une position aléatoire dans la collection (ou même au début) , alors il faut utiliser la classe LinkedList

Dans le cas d'accès concurrents à la liste (multi-threads)

-Utiliser Vector ou copyOnWriteArrayList : ce sont des classes thread-safe

Performances de certaines opérations

	get	add	contains	next	remove(0)	iterator.remove
ArrayList	O(1)	O(1)	O(n)	O(1)	O(n)	O(n)
LinkedList	O(n)	O(1)	O(n)	O(1)	O(1)	O(1)
CopyOnWriteArrayList	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)