



P.O.O. (Programmation Orientée Objet)

CHOUTI Sidi Mohammed

Cours pour L2 en Informatique
Département d'Informatique
Université de Tlemcen
2017-2018

1. Introduction à la Programmation Orientée Objet
2. Classes et objets
3. Héritage et polymorphisme
4. Interface et implémentation
5. **Classes importantes en java**
6. Interface graphique et Applet
7. ...

- Permet de gérer **les chaînes de caractères**
- Une quarantaine de méthodes, et quelques caractéristiques importantes :
 - Déclarée **final** → **non extensible** et **comportement nominal**
 - Un objet String contient **un tableau de char**
 - **Immuable (immutable)** , ce qui signifie que la donnée qu'elle contient est en "lecture seule". Une fois qu'elle a été définie, sa valeur ne peut être modifiée

Exemple : Construction d'un objet String

```
String s1 = "Bonjour le monde !" ;  
String s11 = "Bonjour le monde !" ;  
String s2 = new String("Bonjour le monde !") ;
```

(s1 == s11) **// true**, un seul objet a été créé (optimisation)

(s1 == s2) **// false**, new a forcé la création d'un nouveau objet

length()

renvoie la longueur de la chaîne

charAt()

renvoie le caractère qui occupe la position spécifiée dans l'argument.

Le premier caractère occupe la position **0** et

Le dernier la position **length() – 1** :

Classe StringBuffer

un **StringBuffer** peut être utilisé partout où un String est utilisé.
Il est simplement plus flexible : on **peut modifier son contenu**.

Exemple

```
StringBuffer sb = new StringBuffer();  
sb.append("Cours "); sb.append("de "); sb.append(" POO .");
```

Définition

Une collection est un objet qui contient d'autres objets

Exemple

Un tableau est une collection

Classes

- AbstractCollection, ArrayList, Arrays, Collections, HashSet, LinkedList, TreeSet, Vector...

Interfaces

- List, Map, Set, SortedMap, SortedSet

Ces classes et interfaces se trouvent dans le package **java.util**

Description

- ArrayList fournit un tableau **dynamique** et
- spécifie AbstractList et implémente List.

Déclaration

```
public class ArrayList<E> extends AbstractList<E>  
    implements List<E>, RandomAccess, Cloneable, Serializable
```


Quelques caractéristiques

La classe **java.util.ArrayList** est la classe la plus utilisée

- Un ArrayList se comporte comme un tableau, il contient plusieurs objets (de la classe Object uniquement)
- Ne peut contenir des types primitifs
- Accède à ses éléments à l'aide d'un index
- Pas de taille prédéfinie
- Existe des méthodes pour ajouter ou enlever un élément

Création d'une instance ArrayList

Il y a des constructeurs :

ArrayList()

ArrayList(int initialCapacity)

Il y a deux manières d'ajouter un élément :

- à la fin d'un ArrayList : **boolean add(Object newElement)**
 - à une position donnée : **void add(int index, Object newElement)**
- le paramètre **index** indique où insérer le nouvel élément

Autres méthodes d'ArrayList

- Pour remplacer un objet à une position donnée Object :
set(int index, Object newElement)
- Pour accéder à un élément **Object get(int index)**
- Pour tester le contenu **boolean isEmpty()**
- pour connaître le nombre d'éléments dans la liste
int size()
- Pour savoir si un objet est présent ou non dans une liste
boolean contains(Object obj)
- Pour supprimer un élément à une position donnée,
remove(int index)

Exemple

```
public class Employe {  
  
    private String leNom, lePrenom;  
    private double leSalaire  
  
    public Employe (String unNom, String unPrenom) {  
        leNom = unNom; lePrenom = unPrenom;  
    }  
  
    public void setNom(String nom) { leNom = nom; }  
    public String getNom() { return leNom; }  
}
```

Exemple (sans déclaration préalable de type)

```
public static void main(String [] args) {  
    ArrayList tableauEmployes = new ArrayList();  
    Employe emp1 = new Employe("Belmir", « Réda");  
    Employe emp2 = new Employe("Mehdi", "Said");  
    tableauEmployes.add(emp1);  
    tableauEmployes.add(emp2);  
  
    if (!tableauEmployes.isEmpty()) {  
        for (int i = 0; i < tableauEmployes.size(), i++)  
            System.out.println((Employe) tableauEmployes.get(i).getNom());  
  
        tableauEmployes.remove(1);  
    }  
}
```

Exemple (avec déclaration préalable du type Employe)

```
public static void main(String [] args) {  
    ArrayList<Employe> tableauEmployes =  
        new ArrayList<Employe> ();  
    Employe emp1 = new Employe("Belmir", « Réda");  
    Employe emp2 = new Employe("Mehdi", "Said");  
    tableauEmployes.add(emp1);  
    tableauEmployes.add(emp2);  
  
    if (!tableauEmployes.isEmpty()) {  
        for (int i = 0; i < tableauEmployes.size(), i++)  
            System.out.println((Employe) tableauEmployes.get(i).getNom());  
        tableauEmployes.remove(1);  
    }  
}
```

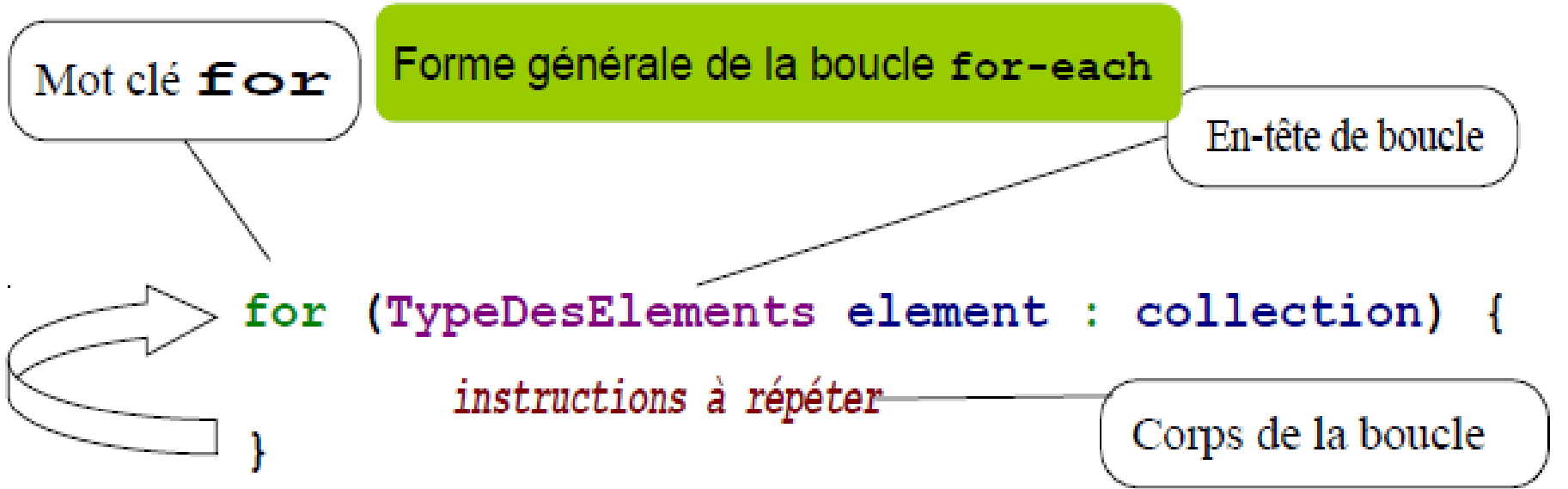
Boucle for-each

Répéter des actions pour chaque objet d'une collection donnée

Mot clé **for**

Forme générale de la boucle **for-each**

En-tête de boucle



```
for (TypeDesElements element : collection) {  
    instructions à répéter  
}
```

The diagram illustrates the components of a Java for-each loop. A callout box labeled 'Mot clé **for**' points to the word 'for'. A green box labeled 'Forme générale de la boucle **for-each**' points to the entire loop structure. A callout box labeled 'En-tête de boucle' points to the header part '(TypeDesElements element : collection)'. A callout box labeled 'Corps de la boucle' points to the body part 'instructions à répéter'. A curved arrow on the left indicates the repetition of the body for each element in the collection.

Exemple

Utilisation d'un index

```
for (int i = 0; i < tableauEmployes.size(); i++)  
    System.out.println( tableauEmployes.get(i).getNom());
```

Boucle for-each

```
for (Employe e : tableauEmployes)  
    System.out.println( e.getNom());
```


Exemple

Utilisation de Iterator

```
for (Iterator<Employe> i=tableauEmployes.iterator();i.hasNext();)  
    System.out.println( i.next().getNom());
```

Exercice

Ecrire une classe **TestClientIterator** qui

1- Remplira les 03 objets suivant dans un objet ArrayList.

Elément	Type	Valeur
1	Integer	42
2	String	"test"
3	Double	-12.34

2- Parcourir cette liste (comme **Iterator** et/ou **non**), afin d'afficher ces trois éléments.

```
import java.util.*;

class TestClientIterator {
    public static void main(String[] args) {
        ArrayList<Object> al = new ArrayList<Object>();
        al.add(new Integer(42));
        al.add(new String("test"));
        al.add(new Double("-12.34"));
        // Comme Iterator
        for(Iterator<Object> iter=al.iterator(); iter.hasNext();)
            System.out.println( iter.next() );
        //Pas comme Iterator
        for(Object o:al) System.out.println( o );
    }
}
```

Comme ArrayList, **LinkedList** implémentent l'interface **List**

- ArrayList utilise un **tableau** extensible
- Utilise **efficacement** les méthodes **get()** et **set()**.
- LinkedList est implémentée sous forme d'une **liste chaînée**,
- Ces performances **d'ajout et de suppression** sont plus meilleures que celles de ArrayList, mais mauvaises pour les méthodes **get()** et **set()**.

Comparaison de temps d'exécution de méthodes entre objets LinkedList et ArrayList

-----méthode **add** -----

ArrayList : **101**

LinkedList : **469**

-----méthode **get** -----

ArrayList : **1**

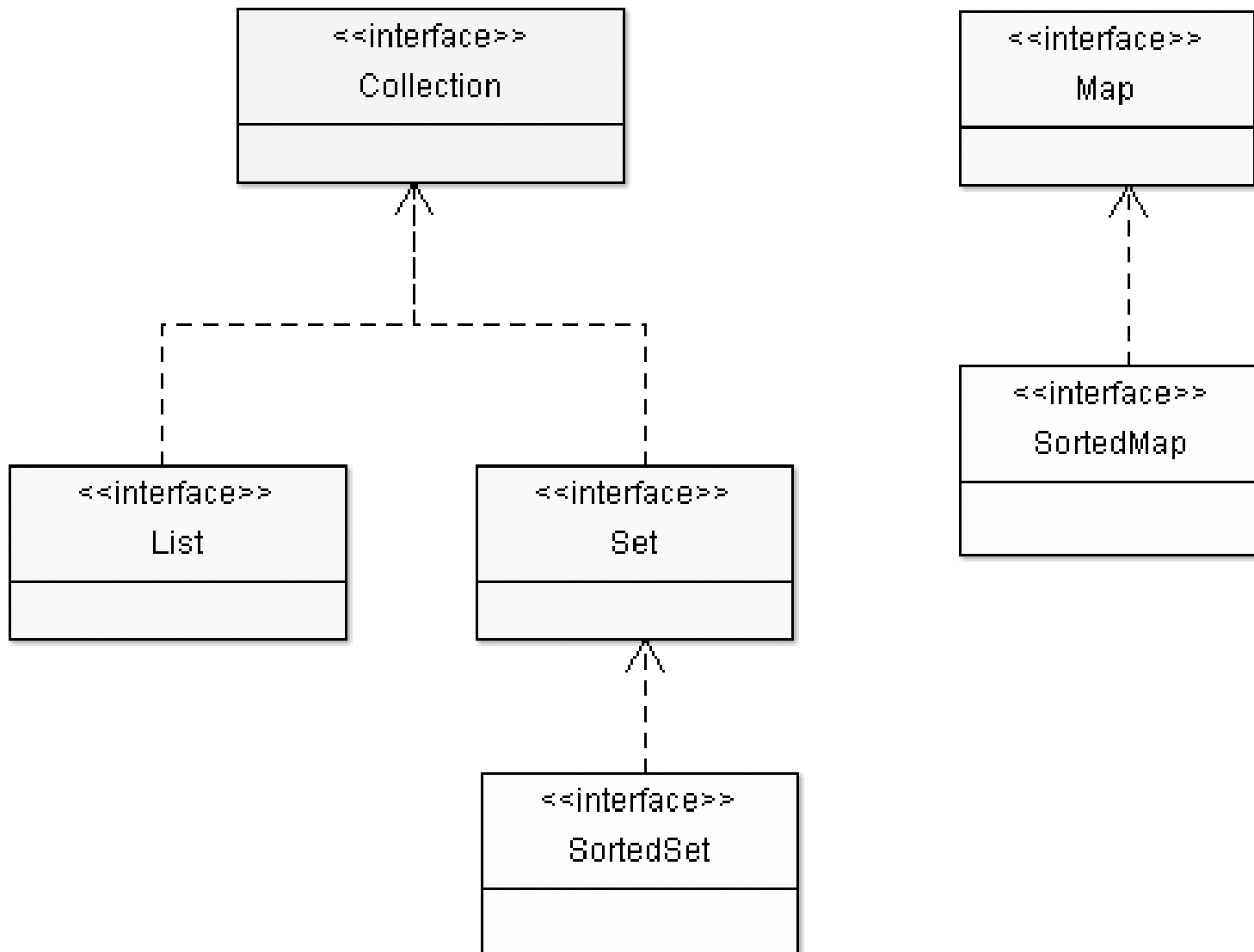
LinkedList : **24592**

-----méthode **remove** -----

ArrayList : **2671**

LinkedList : **94**

Autres Collections



Plusieurs types de collections :

- Interfaces **List** et **Set** qui héritent l'interface **Collection**
 - Les objets **List** acceptent toutes les valeurs, même les valeurs null
 - **Set** n'autorise pas deux fois la même valeur (le même objet), ce qui est pratique pour une liste d'éléments uniques.
- Ainsi que l'interface **Map**
Les **Map** fonctionnent avec un système clé - valeur pour ranger et retrouver les objets qu'elles contiennent.