



P.O.O. (Programmation Orientée Objet)

CHOUTI Sidi Mohammed

Cours pour L2 en Informatique

Département d'Informatique

Université de Tlemcen

2019-2020

1. Introduction à la Programmation Orientée Objet
2. Classes et Objets
3. Héritage, polymorphisme
4. Abstraction et déclaration finale
5. Interface, implémentation et Paquetage
- 6. Classes Courantes en Java**
7. Gestion des Exceptions
8. Interfaces graphiques

Lecture des entrées - Paramètre args de main

```
public static void main (String [] args)
```

- Le tableau **args** est constitué d'arguments passés au programme via la ligne de commande :

Exemple : Soit **ArgsTest** une classe principale et son exécution

```
>java ArgsTest un deux trois
```

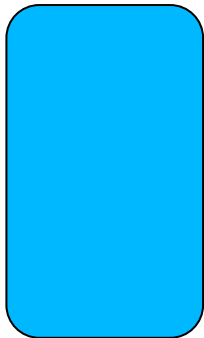
- `args[0] = "un"`, `args[1] = "deux"`, `args[2] = "trois"`.

Les éléments du tableau `args` sont des chaînes de caractères

Paramètre args de main

```
class ArgsTest{
    public static void main(String[] args) {
        for (int i=0; i<args.length; i++) {
            System.out.println(args[i] );
        }
    }
}
```

Après exécution, on aura :



Paramètre args de main

```
class Somme {  
    public static void main(String[] args) {  
        int somme = 0;  
        for (int i=0; i < args.length ; i++) {  
            somme = somme + Integer.parseInt(args[i]); }  
        System.out.println(somme);  
    }  
}
```

Après exécution :

>java Somme 12 23 34, on aura :



Il est souvent nécessaire de convertir les args[i], qui sont toujours de type String

Lecture des entrées – classe Scanner

La classe **Scanner** (java.util)

Permet la **lecture** de données de **l'utilisateur** à partir du **clavier** dans une application **console**

Lecture des entrées – classe Scanner

Exemple d'utilisation

Demande de valeurs de différents types à l'utilisateur :

```
import java.util.Scanner;
class Test {
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);

        System.out.println("Entrez votre nom : ");
        String nom = clavier.next();

        System.out.println("Entrez votre age : ");
        int age = clavier.nextInt();

        System.out.println("Entrez votre moyenne : ");
        float moyenne = clavier.nextFloat();
    }
}
```

Lecture des entrées – classe Scanner

```
System.out.println("Vos caractéristiques sont : \n");
System.out.println("Nom : " + nom);
System.out.println("Age : " + age);
System.out.println("Moyenne : " + moyenne);
    }
}
```


- 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
```

s.equals(Object o)

a été redéfini et renvoie true si l'objet donné "**o**" représente une chaîne équivalente à celle de "**s**", false sinon.

s.length()

Renvoie la longueur de la chaîne **s**

s.charAt(int index)

renvoie le caractère qui occupe la position **index**.

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

Le dernier caractère occupe 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.");
```

```
System.out.print (sb) ; // affichera Cours de 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 paquetage **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 Etudiant{  
  
    private String leNom;  
  
    public Etudiant(String unNom){  
        leNom = unNom;  
    }  
  
    public void setNom(String nom) { leNom = nom; }  
    public String getNom() { return leNom; }  
}
```

Exemple (utilisant la classe Object)

```
public static void main(String [] args) {  
    ArrayList <Object> tableauEtudiants= new ArrayList<Object>();  
    Etudiant e1 = new Etudiant("Bachir")  
    Etudiant e2 = new Etudiant("Nadir");  
    tableauEtudiants.add(e1);  
    tableauEtudiants.add(e2);  
  
    if (! tableauEtudiants.isEmpty()) {  
        for (int i = 0; i < tableauEtudiants.size(); i++)  
            System.out.println(((Etudiant) tableauEtudiants.get(i)).getNom());  
  
        tableauEtudiants.remove(1);  
    }  
}
```

Exemple (utilisant la classe Etudiant)

```
public static void main(String [] args) {  
    ArrayList<Etudiant> tableauEtudiants =  
                                new ArrayList<Etudiant> ();  
    Etudiant emp1 = new Etudiant(« Bachir »);  
    Etudiant emp2 = new Etudiant("Nadir");  
    tableauEtudiants.add(emp1);  
    tableauEtudiants.add(emp2);  
  
    if (!tableauEtudiants.isEmpty()) {  
        for (int i = 0; i < tableauEtudiants.size(), i++)  
            System.out.println((Etudiant) tableauEtudiants.get(i).getNom());  
        tableauEtudiants.remove(1);}  
}
```

Boucles

Exemple

Utilisation d'un index

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

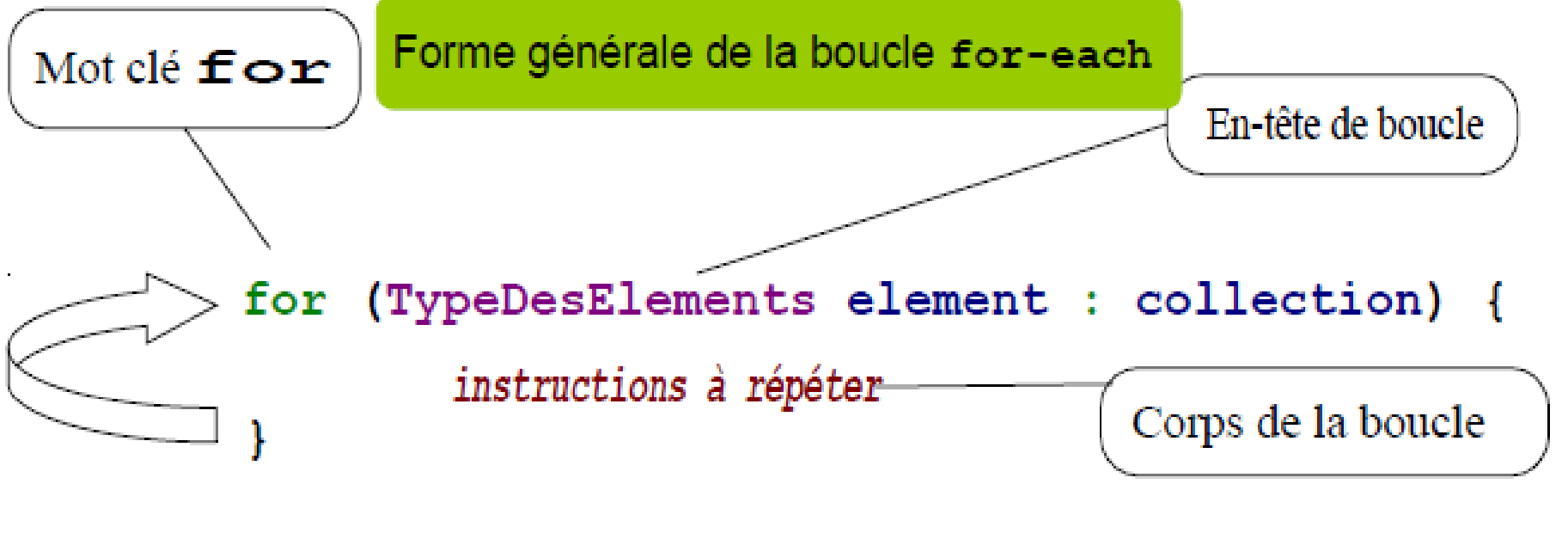
Boucle for-each

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  
}
```

Corps de la boucle

Exemple

Boucle for-each

```
for (Etudiant e : tableauEtudiants)
    System.out.println( e.getNom());
```

Exemple

Utilisation de Iterator

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

ArrayList implémente la méthode *Iterator iterator()* qui retourne un itérateur sur l'ensembles des éléments.

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**

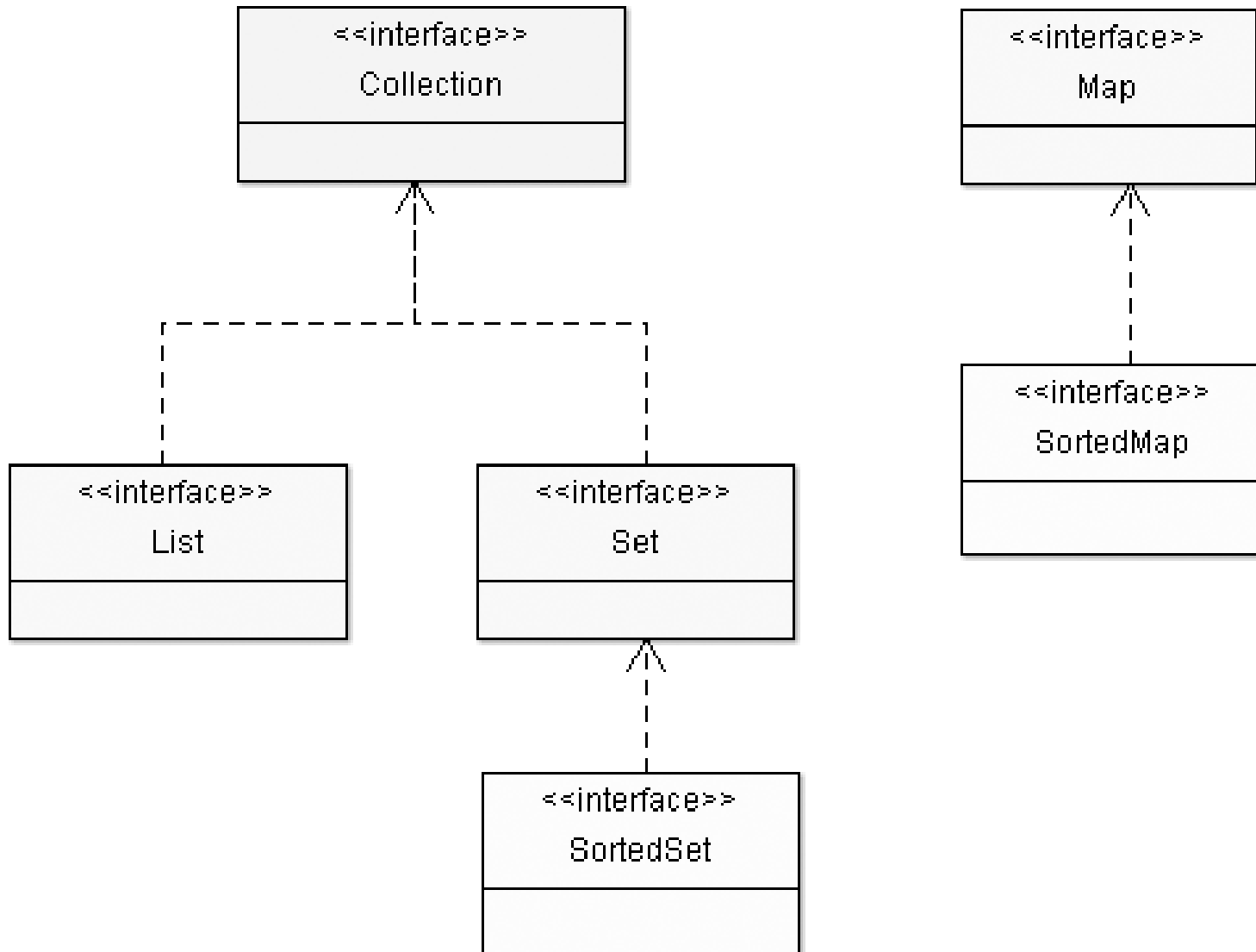
LinkedList : **24592**

-----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.