Programmation Orientée Objet Avancée – Java
GIGI2 ENSA-Tétouan

A. EL HIBAOUI

Faculté des Sciences de Tétouan – Université Abdelmalek Essaâdi
Département Informatique
2014–2018

a.hibaoui@gmail.com

Programmation Orientée Objet Avancée – Java
GIGI2 ENSA-Tétouan

A. EL HIBAOUI

Faculté des Sciences de Tétouan – Université Abdelmalek Essaâdi
Département Informatique
2014–2018

a.hibaoui@gmail.com

Collections en Java

Plan

Structures de données

C'est l'organisation efficace d'un ensemble de données, sous la forme de tableaux, de listes, de piles etc. Cette efficacité réside dans la quantité mémoire utilisée pour stocker les données, et le temps nécessaire pour réaliser des opérations sur ces données.
Une collection gère un groupe d'un ensemble d'objets d'un type donné ; ou bien c'est un objet qui sert à stocker d'autres objets.

Objectifs :

- organisation "efficace" d'un ensemble de données
- efficacité = compromis entre la quantité de mémoire utilisée
- rapidité d'accès aux données
- opérations usuelles sur les collections : tri, recherche, ...

Interfaces de Collections

Interfaces de Collections sont organisées en deux catégories : Collection & Map.

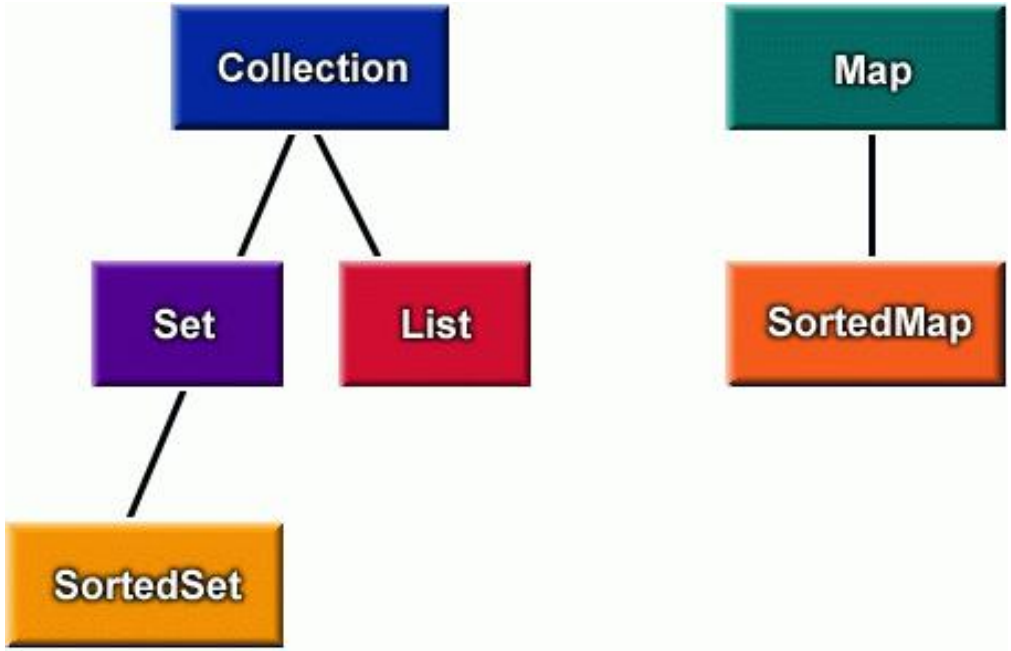

Figure: Collection & Map

Collections à partir du Java 2

- Collection : interface qui est implémentée par la plupart des objets qui gèrent des collections.
- Map : interface qui définit des méthodes pour des objets qui gèrent des collections sous la forme clé/valeur.
- Set : interface pour des objets qui n'autorisent pas la gestion des doublons dans l'ensemble.
- List : interface pour des objets qui autorisent la gestion des doublons et un accès direct à un élément.
- SortedSet : interface qui étend l'interface Set et permet d'ordonner l'ensemble.
- SortedMap : interface qui étend l'interface Map et permet d'ordonner les cartes.

Implémentation des interfaces

Interface	Implémentation
Set	HashSet
SortedSet	TreeSet
List	ArrayList, LinkedList, Vector
Map	HashMap, Hashtable
SortedMap	TreeMap

Implémentations

La classe collection implémente un Type Abstrait de Données (TAD) comme une classe
Java implémente toutes les méthodes de l'interface
Sélectionne les instances de variables appropriées
Peut être instancié
Java implémente les interfaces :

- List : ArrayList, LinkedList, Vector
- Map : HashMap, TreeMap
- Set : TreeSet, HashSet

L'interface Collection

```
import java.util.Iterator;
interface Collection<E> {
    boolean add(E o);
    boolean addAll(Collection<? extends E> c);
    void clear();
    boolean contains(Object o);
    boolean containsAll(Collection<?> c);
    boolean equals(Object o);
    int hashCode();
    boolean isEmpty();
    Iterator<E>  iterator();
    boolean remove(Object o);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    int size();
    Object[ ] toArray();
    <T> T[ ] toArray(T[ ] a);
}
```

Méthodes

- **boolean add(Object o) :** ajoute l'objet spécifié au sein de la collection courante.
- **boolean addAll(Collection c) :** ajoute tous les éléments de la collection spécifiée au sein de l'objet Collection courant.
- **void clear() :** supprime tous les éléments de la collection.
- **boolean contains(Object o) :** retourne true si la collection contient l'objet spécifié.
- **boolean containsAll(Collection c) :** retourne true si la collection courante contient tous les éléments de la collection spécifiée.
- **boolean equals(Object o) :** teste l'égalité entre la collection et l'objet spécifié.
- **int hashCode() :** retourne le code de hachage de la collection.
- **boolean isEmpty() :** retourne true si la collection ne contient aucun élément.
- **Iterator iterator() :** retourne un itérateur sur les éléments de la collection.
- **boolean remove(Object o) :** supprime une seule instance de l'élément spécifié, s'il est présent au sein de la collection.
- **boolean removeAll(Collection c) :** supprime tous les éléments de la collection spécifiée, s'ils sont présents dans la collection courante.
- **boolean retainAll(Collection c) :** retient seulement les éléments de la collection courante qui sont contenus dans la collection spécifiée.
- **int size() :** retourne le nombre d'éléments de la collection courante.
- **Object[] toArray() :** retourne un tableau contenant tous les éléments de la collection courante.
- **Object[] toArray(Object[] a) :** retourne un tableau contenant tous les éléments de la collection courante. Le type d'exécution du tableau retourné est celui du tableau spécifié

Itérateurs

Les itérateurs sont des objets qui permettent de parcourir un par un les différents éléments d'une collection. Java propose deux sortes d'itérateurs :

- **Monodirectionnels :** parcours début vers la fin
- **Bidirectionnels :** parcours peut se faire dans les deux sens

Itérateur monodirectionnel: interface Iterator

Chaque classe de collection dispose d'un objet nommé : iterator, i.e. un objet d'une classe implémentant l'interface Iterator.

Remarque

- Un itérateur indique/désigne la position courante
- La méthode next() permet d'obtenir l'objet désigné et de faire passer l'itérateur à l'élémenet suivant.
- La méthode hasNext() permet de savoir si l'itérateur est à la fin de la collection.

Canevas de parcours d'une collection

```
Iterator iter = c.iterator();
     while (iter.hasNext()) {
     Objet o = iter.next();
     // traitement
     System.out.println(o);
}
```

Itérateurs bidirectionnels: interface ListIterator

Objet implementant l'interface ListIterator (dérivée de Iterator). En plus des méthodes de Iterator.

Méthodes duales :

- previous et hasPrevious
- Méthodes d'addition d'un élément à la position courante (add)
- Méthode de modification de l'élément courant (set)
- ...

Exemple: parcours inversé

```
ListIterator iter = c.ListIterator(c.size());
// position courante: fin de liste
While (iter.hasPrevious()){
       Object o = iter.previous();
       System.out.println(o);
}
```

Toute collection dispose d'une méthode

- add(element) indépendante d'un quelconque itérateur.
- size() fournit la taille de la collection
- isEmpty() teste si la collection est vide
- clear() supprime tous les éléments de la collection
- ...

Les collections vues comme des ensembles réalisent les 3 opérations mathématiques sur des ensembles:

- **union** : add et addAll
- **intersection** : retainAll
- **différence** : remove et removeAll

add et remove

Méthode add

Interface ListIterator prévoit une méthode add

```
ListIterator it = c.listIterator();
it.next(); // premier élément
it.next(); // deuxieme élément
it.add(elem); // ajoute elem à la position courante, i.e. entre le deuxième et le troisième élément
```

Méthode remove()

Supprime de la collection le dernier objet renvoyé par next.

```
Iterator iter = c.iterator();
While (iter.hasNext()){
    Object oc = iter.next();
    // fournit l'élément désigné par l'itérateur et avance l'itérateur à la position suivante
    If (oc==op) iter.remove();
    // si l'objet désigné par l'itérateur oc est égal à un objet op alors supprimer oc de la collection
}
```

set

set(elem) : remplace par elem l'élément courant, i.e. le dernier renvoyé par next() ou previous()

```
ListIterator it = c.listIterator();
While (it.hasNext()) {
    Object o = it.next();
    If (UneCondition) it.set(null);
} // remplace par null tous les éléments d?une collection vérifiant une condition
```

Listes chaînées: classe LinkedList

Permet de manipuler des listes dites "doublement chaînées"

```
import java.util.*;
public class Liste {
   public static void main(String args[]) {
        LinkedList l = new LinkedList();
        System.out.print("Liste A:"); affiche(l);
        l.add("a"); l.add("b");
        System.out.print("Liste B:"); affiche(l);
        ListIterator it = l.listIterator();
        it.next();
        it.add("c"); it.add("b");
        System.out.print("Liste C:"); affiche(l);
        it = l.listIterator();
        it.next();
```

```
        it.add("b"); it.add("d");
        System.out.print("Liste D:"); affiche(l);
        it = l.listIterator(l.size());
        while (it.hasPrevious()) {
            String ch = (String) it.previous();
            if (ch.equals("b")) {
                it.remove();
                break;
            }
        }

        System.out.print("Liste E:"); affiche(l);

        it = l.listIterator();
        it.next(); it.next();
        it.set("x");
        System.out.print("Liste F:"); affiche(l);
    }

    public static void affiche (LinkedList l) {
          ListIterator iter = l.listIterator();
          while (iter.hasNext())
          System.out.print (iter.next() + " ");
          System.out.println();
    }
}
```

```
        it.add("b"); it.add("d");
        System.out.print("Liste D:"); affiche(l);
        it = l.listIterator(l.size());
        while (it.hasPrevious()) {
            String ch = (String) it.previous();
            if (ch.equals("b")) {
                it.remove();
                break;
            }
        }

        System.out.print("Liste E:"); affiche(l);

        it = l.listIterator();
        it.next(); it.next();
        it.set("x");
        System.out.print("Liste F:"); affiche(l);
    }

    public static void affiche (LinkedList l) {
          ListIterator iter = l.listIterator();
          while (iter.hasNext())
          System.out.print (iter.next() + " ");
          System.out.println();
    }
}
```

Résultats :

```
>> java Liste
Liste A:
Liste B:a b
Liste C:a c b b
Liste D:a b d c b b
Liste E:a b d c b
Liste F:a x d c b
```

Collection ArrayList

Offre des fonctionnalités d'accès rapide comparables à celle d'un tableau d'objets

```
import java.util.*;
public class Array {
   public static void main(String args[]) {
        ArrayList v = new ArrayList();
        for (int i=0;i<10;i++)
            v.add(new Integer(i));
        System.out.println("En 0: contenu de v" + v);
        v.add(2,"AAA");
        v.add(4,"BBB");
        v.add(8,"CCC");
        v.add(5,"DDD");

        System.out.println("En I: contenu de v" + v);
        for (int i=0;i<v.size(); i++)
          if (v.get(i) instanceof String) v.set(i,null);
        System.out.println("En II: contenu de v" + v);

        LinkedList l = new LinkedList();
        l.add(new Integer (5));
        l.add(null);
        v.removeAll(l);
        System.out.println("En III contenu v" + v);
   }
}
```

Collection ArrayList

Offre des fonctionnalités d'accès rapide comparables à celle d'un tableau d'objets

```
import java.util.*;
public class Array {
   public static void main(String args[]) {
        ArrayList v = new ArrayList();
        for (int i=0;i<10;i++)
            v.add(new Integer(i));
        System.out.println("En 0: contenu de v" + v);
        v.add(2,"AAA");
        v.add(4,"BBB");
        v.add(8,"CCC");
        v.add(5,"DDD");

        System.out.println("En I: contenu de v" + v);
        for (int i=0;i<v.size(); i++)
          if (v.get(i) instanceof String) v.set(i,null);
        System.out.println("En II: contenu de v" + v);

        LinkedList l = new LinkedList();
        l.add(new Integer (5));
        l.add(null);
        v.removeAll(l);
        System.out.println("En III contenu v" + v);
   }
}
```

Résultats :

```
>> java Array
En 0: contenu de v[0, 1, 2, 3, 4, 5
En I: contenu de v[0, 1, AAA, 2, BB
En II: contenu de v[0, 1, null, 2,
En III contenu v[0, 1, 2, 3, 4, 6,
```

L'interface **Set**

C'est une interface identique à celle de Collection. Deux implémentations possibles :

- TreeSet : les éléments sont rangés de manière ascendante.
- HashSet : les éléments sont rangés suivant une méthode de hachage.

```
import java.util.*;
public class Ens {
    public static void main (String args[]) {
        String phrase = "Le prof n'est ni ange ni bête et le malheur veut que qui veut faire l'ange fait
        String voy="aeiouyê";
        HashSet lettres = new HashSet();
        for (int i=0;i<phrase.length();i++)
            lettres.add(phrase.substring(i,i+1));
        System.out.println("lettres presentes:"+lettres);

        HashSet voyelles = new HashSet();
        for (int i=0;i<voy.length();i++)
            voyelles.add(voy.substring(i,i+1));

        lettres.removeAll (voyelles);
        System.out.println("lettres sans les voyelles " + lettres);
     }
}
```

Résultats :

```
>> java Ens
lettres presentes:[f, g,  , e, b, ê, ', a, L, n, o, l, m, h, i, v, u, t, s, r, q, p]
lettres sans les voyelles [f, g,  , b, ', L, n, l, m, h, v, t, s, r, q, p]
```

Exemple

```
import java.util.*;
class Point {
    private int x,y;
    Point (int x, int y) {
       this.x=x;  this.y=y;
    }
    public int hashCode () {
          return x+y;
    }
    public boolean equals (Object pp) {
          Point p = (Point) pp;
          return ((this.x == p.x) & (this.y == p.y));
    }
    public String toString() {
          return "[" + x + " " + y + "]";
   }
}
```

```
import java.util.*;
public class EnsPt1 {
    public static void main (String args[]) {
        Point p1 = new Point (1,3), p2 = new Point (2,2);
        Point p3 = new Point(4,5), p4 = new Point (1,8);

        Point p[] = {p1, p2, p1, p3, p4, p3};
        HashSet ens = new HashSet();
        for (int i=0;i<p.length;i++) {
              System.out.print("le point");
              System.out.println(p[i]);
              boolean ajoute = ens.add(p[i]);
              if (ajoute)
                   System.out.println("a ete ajouté");
              else System.out.println("est déjà présent");
              System.out.print("ensemble=");
              affiche(ens);
         }
    }

    public static void affiche (HashSet ens){
        Iterator iter = ens.iterator();
        while (iter.hasNext()) {
               Point p = (Point) iter.next();
               System.out.print(p);
        }
        System.out.println();
   }
}
```

Résultats :

```
>>java EnsPt1
le point[1 3]
a ete ajouté
ensemble=[1 3]
le point[2 2]
a ete ajouté
ensemble=[2 2][1 3]
le point[1 3]
est déjà présent
ensemble=[2 2][1 3]
le point[4 5]
a ete ajouté
ensemble=[2 2][1 3][4 5]
le point[1 8]
a ete ajouté
ensemble=[2 2][1 3][1 8][4 5]
le point[4 5]
est déjà présent
ensemble=[2 2][1 3][1 8][4 5]
```

L'interface **List**

Liste est une collection ordonnée. Elle permet la duplication des éléments. L'interface est renforcée par des méthodes permettant d'ajouter ou de retirer des éléments se trouvant à une position donnée. Elle permet aussi de travailler sur des sous listes.
On utilise le plus souvent des ArrayList sauf s'il y a insertion d'élément(s) au milieu de la liste. Dans ce cas il est préférable d'utiliser une LinkedList pour éviter ainsi les décalages.

```
import java.util.ListIterator;
interface List<E> extends Collection<E> {
    void add(int index, E element);
    boolean addAll(int index, Collection<? extends E> c);
    E get(int index);
    int indexOf(Object o);
    int lastIndexOf(Object o);
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);
    E remove(int index);
    E set(int index, E element);
    List<E> subList(int fromIndex, int toIndex);
}
```

Parcours et Opérations sur les listes

Pour parcourir une liste, il a été défini un itérateur spécialement pour la liste.

```
public interface ListIterator extends Iterator {
       boolean hasNext();
       Object next();
       boolean hasPrevious();
       Object previous();
       int nextIndex();
       int previousIndex();
       void remove(); // Optional
       void set(Object o); // Optional
       void add(Object o); // Optional
}
```

List implementée par 3 classes :

```
// Interface name: List // Three classes that implement the List interface:
//  creation de nouvelles instances :
List<String> bigList = new ArrayList<String>();
List<String> littleList = new LinkedList<String>();
List<String> sharedList = new Vector<String>();

// All three have an add method
bigList.add("in array list");
littleList.add("in linked list");
sharedList.add("in vector");

// All three have a get method
assertEquals("in array list", bigList.get(0));
assertEquals("in linked list", littleList.get(0));
assertEquals("in vector", sharedList.get(0));
```

Exemple

```
import java.util.*;
public class ListExample {
    public static void main(String args[]) {
        List list = new ArrayList();
        list.add("Fatima");
        list.add("Aicha");
        list.add("Hafsa");
        list.add("Aicha");
        list.add("Mariam");
        System.out.println(list);
        System.out.println("2: " + list.get(2));
        System.out.println("0: " + list.get(0));
        LinkedList queue = new LinkedList();
        queue.addFirst("Fatima");
        queue.addFirst("Aicha");
        queue.addFirst("Hafsa");
        queue.addFirst("Aicha");
        queue.addFirst("Mariam");
        System.out.println(queue);
        queue.removeLast();
        queue.removeLast();
        System.out.println(queue);
    }
}
```

Résultats :

```
>> java ListExample
[Fatima, Aicha, Hafsa, Aicha, Mariam]
2: Hafsa
0: Fatima
[Mariam, Aicha, Hafsa, Aicha, Fatima]
[Mariam, Aicha, Hafsa]
```

Sont utilisés pour traiter les éléments d'un ensemble de données. Ils définissent une procédure informatique,

- **Sorting** (e.g. sort)
- **Shuffling** (e.g. shuffle)
- **Routine Data Manipulation** (e.g. reverse, addAll)
- **Searching** (e.g. binarySearch)
- **Composition** (e.g. frequency)
- **Finding Extreme Values** (e.g. max)

Exemple

```
import java.util.* ;
public class ExempleTreeMap{
    public ExempleTreeMap (){
        TreeMap tree = new TreeMap () ;
        tree.put ("Omar",new Integer (26));
        tree.put ("Mohamed", new Integer (1)) ;
        tree.put ("Ali", new Integer (2)) ;
        Iterator itercle = tree.keySet().iterator() ;
        Iterator itervaleurs = tree.values().iterator() ;
        while (itercle.hasNext()) {
            System.out.println (itercle.next() + " --> " + itervaleurs.next()) ;
        }
    }
    public static void main(String[] args){
        new ExempleTreeMap () ;
    }
}
```

Résultats :

```
>>java ExempleTreeMap
Ali --> 2
Mohamed --> 1
Omar --> 26
```

Collections et threads

Si plusieurs threads peuvent accéder à un objet collection, il y a nécessité de synchroniser avec une des méthodes statiques de la classe java.util.Collections :

```
    static Collection synchronizedCollection (Collection c)
    static List synchronizedList (List list)
    static Map synchronizedMap (Map m)
    static Set synchronizedSet (Set s)
    static SortedMap synchronizedSortedMap (SortedMap m)
    static SortedSet synchronizedSortedSet (SortedSet s)
```

Remarque

Les méthodes précédentes ne synchronisent pas les itérateurs. Il faut donc le faire manuellement:

```
synchronized (objet){
    Iterator iter = objet.iterator () ;
       {
           // travailler avec l'iterateur
       }
}
```

Exo

Dans la classe Proc, la méthose receive doit synchroniser l'objet StackOfMessage :

```
protected  Message receive (int pid){
    synchronized(StackOfMessages){
        Iterator<Message> iterator = StackOfMessages.iterator();
        while (iterator.hasNext()) {
            Message m = iterator.next();
            if(m.getDest()==pid){
                Message m2=new Message(m);
                iterator.remove();
               return m2;
            }
        }
     }
     return null;
}
```

Propriétés

La classe java.util.Properties est une table de hachage pour définir des variables d'environnement sous la forme (nom_variable, valeur).
Exemple:

```
Properties props = new Properties () ;
props.put ("monApp.xSize","50") ;
```

La méthode statique System.getProperties () retourne les variables d'environnement définies telles que:

- java.vendor
- java.home
- file.separator
- path.separator
- user.name
- user.home
- user.dir
- ...

Collection et Java > 1.5

Jusqu'à la version 1.4, on stockait et récupérait des "Object" d'une collection.
Exemple :

```
ArrayList liste = new ArrayList ()
liste.add (new MaClasse ()) ;
MaClasse obj = (MaClasse) liste.get (0) ;
```

Depuis la version 1.5, il est recommandé de spécifier la nature des objets stockés.
Exemple :

```
ArrayList<MaClasse> liste = new ArrayList<MaClasse> () ;
liste.add (new MaClasse ())
MaClasse obj = liste.get (0) ;
```

Interface Comparable

Dans certains cas nécessité de classer les éléments à partir de leur valeur (recherche d'un max, min, tri ...)

- Méthodes concernées considèrent par défaut que ses éléments implémentent l'interface Comparable et recourent à la méthode compareTo
- Possibilité de définir une méthode de comparaison appropriée par le biais d'un objet comparateur

Méthode compareTo de l'interface Comparable

Certaines classes comme String, classes enveloppes (Integer, Float,...) implémente l'interface comparable et dispose donc d'une méthode compareTo qui conduit à un ordre naturel

- Lexicographique pour les chaînes de caractères
- Numériques pour les classes enveloppes numériques
- Pour des objets propres à l'utilisateur : nécessaire de redéfinir la méthode compareTo (object o)

Utilisation d'un objet comparateur

Dans quels cas

- Les éléments sont des objets d'une classe existante n'implémentant pas l'interface comparable
- Besoin de définir plusieurs ordres différents sur une même collection

Exemple de méthodes de l'interface collection

- void copy(List, List) : copie tous les éléments
- Object max(Collection) : renvoie le plus grand élément de la collection
- Object max(Collection, Comparator) : renvoie le plus grand élément de la collection selon l'ordre précisé par l'objet Comparator
- void sort(List) : trie la liste dans un ordre ascendant
- void sort(List, Comparator) : trie la liste dans un ordre ascendant selon l'ordre précisé par l'objet Comparator ...

Note :

Si la méthode sort(List) est utilisée, il faut obligatoirement que les éléments inclus dans la liste implémentent tous l'interface Comparable. Même chose pour max(Collection)

Exemple de Tri

```
import java.util.*;
public class Tri {
    public static void main(String[] args) {
    int nb [] = {4,5,2,1,6,8,3};
    ArrayList t = new ArrayList();
        for (int i=0;i<nb.length;i++) t.add(new Integer(nb[i]));
        System.out.println("t initial="+t);

        Collections.sort(t);
        System.out.println("t trié="+t);

        Collections.shuffle(t);
        System.out.println("t mélangé="+t);

        Collections.sort(t, Collections.reverseOrder()); // un comparateur prédéfini
        System.out.println("t trié="+t);
    }
}
```

Résultat

```
t initial=[4, 5, 2, 1, 6, 8, 3]
t trié=[1, 2, 3, 4, 5, 6, 8]
t mélangé=[2, 4, 3, 8, 1, 6, 5]
t trié=[8, 6, 5, 4, 3, 2, 1]
```

Exemple : Méthode **compareTo**

```
import java.util.*;

public class EmployeC implements Comparable{
         int id;
         EmployeC (int i){
             id = i;
        }
        public int compareTo(Object o) {
               if ((this.id) < (((EmployeC)(o)).id))
                     return -1;
               else if ((this.id) > (((EmployeC)(o)).id))
                        return 1;
                    else
                       return 0;
       }
       void imprimer(){
                System.out.println("EmployeC "+id);
       }
       public String toString(){
                  return "EmployeC :"+id;
       }
}
```

Exemple : Méthode **compareTo**

```
class Main_EmployeC1 {
    public static void main(String[] args) {
            ArrayList c= new ArrayList();
            Random r = new Random();
            for (int i=0;i<10;i++) c.add(new EmployeC (r.nextInt(100)));
            for(int i=0; i<c.size();i++) System.out.println(i + " " +c.get(i));
            System.out.println("max " + Collections.max(c));
           Collections.sort(c);
            for(int i=0; i<c.size();i++) System.out.println(i + " " + c.get(i));
    }
}
```

```
import java.util.*;

public class Employe {
     int id;
     int salaire ;
     Employe (int i, int s){
        id = i;
        salaire =s;
     }
     void imprimer(){
         System.out.println("Employe "+id +" salaire "+salaire);
     }
     public String toString(){
         return "Employer :"+id+" salaire "+salaire;
     }
}
```

```
import java.util.Comparator;
class EmployeComparator implements Comparator{
    public int compare (Object o1, Object o2){
      if ((((Employe)(o1)).salaire) < (((Employe)(o2)).salaire)) return -1;
      else if ((((Employe)(o1)).salaire) > (((Employe)(o2)).salaire)) return 1;
        else return 0;
    }
}
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.Random;

class Main_Employe {
    public static void main(String[] args) {
        ArrayList c= new ArrayList();
        Random r= new Random();
        for (int i=0;i<10;i++) c.add(new Employe(i,r.nextInt(5000)));
        for(int i=0; i<c.size();i++) System.out.println(i + " " +c.get(i));

        Comparator comp = new EmployeComparator();
        System.out.println("max " + Collections.max(c,comp));
        Collections.sort(c,comp);
        for(int i=0; i<c.size();i++)
        System.out.println(i + " " + c.get(i));

   }
}
```

Résultats :

```
0 Employer :0 salaire 210
1 Employer :1 salaire 4785
2 Employer :2 salaire 3696
3 Employer :3 salaire 843
4 Employer :4 salaire 1633
5 Employer :5 salaire 2723
6 Employer :6 salaire 582
7 Employer :7 salaire 1398
8 Employer :8 salaire 629
9 Employer :9 salaire 2581
max Employer :1 salaire 4785
0 Employer :0 salaire 210
1 Employer :6 salaire 582
2 Employer :8 salaire 629
3 Employer :3 salaire 843
4 Employer :7 salaire 1398
5 Employer :4 salaire 1633
6 Employer :9 salaire 2581
7 Employer :5 salaire 2723
8 Employer :2 salaire 3696
9 Employer :1 salaire 4785
```

Combinaison des deux : Tri sur différents critères

```
import java.util.*;
class Employe implements Comparable{
        int id;
        String nom;
        float  salaire;
        Employe (int i, String N, float S){
             id=i;
             nom = N;
             Salaire = S;
        }
        public int compareTo(Object o) {
             if this.salaire<((Employe)(o).salaire) return -1;
            else  if this.salaire>((Employe)(o).salaire) return 1;
            else return 0;
        }
        void imprimer(){
             System.out.println("Employe : "+nom );
        }
        public String toString(){
             return "Employé -- id : "+id+ " nom : "+ nom+ " salaire : ''+salaire;
        }
}
```

```
class EmployeComparator implements Comparator{
      public int compare (Object o1, Object o2){
         return (Employe)(o1).nom.equals((Employe)(o2).nom);
       }
}
```

```
public class Main_Employe_2 {
public static void main(String[] args) {
          ArrayList c= new ArrayList();
          Random r = new Random();
          for (int i=0;i<10;i++)
          c.add(new Employe(r.nextInt(100),"empoloye"+r.next(100), r.nextFtoat()));
          for(int i=0; i<c.size();i++)
         System.out.println(i + " " +((Employe) c.get(i)).nom+":"+((Employe)                    c

          Comparator comp = new EmployeComparator();
         System.out.println("max comparateur" + Collections.max(c,comp));
         Collections.sort(c,comp);
         for(int i=0; i<c.size();i++)
         System.out.println(i + " " +((Employe) c.get(i)).nom+":"+((Employe)                    c

       System.out.println("max comparable" + Collections.max(c));
        Collections.sort(c);
        for(int i=0; i<c.size();i++)
                 System.out.println(i + " " +((Employe) c.get(i)).nom+":"+((Employe)

   }
}
```