Java : les collections

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en Programmation par contrainte (IA)
Ingénieur en Génie logiciel

elmouelhi.achref@gmail.com

Plan

Java

Les collections, c'est quoi ?

- sont des objets
- permettent de regrouper et gérer plusieurs objets

Java

Pourquoi ne pas utiliser les tableaux ?

Java

Pourquoi ne pas utiliser les tableaux ?

Pour plusieurs raisons

- Il faut connaitre à l'avance la taille du tableau
- Si on veut dépasser la taille déclarée, il faut créer un nouveau tableau puis copier l'ancien (certains langages ont proposés l'allocation dynamique mais ça reste difficile à manipuler)
- Il est difficile de supprimer ou d'ajouter un élément au milieu du tableau
- Il faut parcourir tout le tableau pour localiser un élément (problème d'indexation)
- Les tableaux ne peuvent contenir des éléments de type différent

Java

Plusieurs types de collections proposés

- `List` : tableaux extensibles à volonté, accessibles via leurs indice ou valeur
- `Set` : collection qui n'accepte pas de doublons
- `Map` : collection qui exige une clé unique pour chaque élément
- `Queue` : collection gérée comme une file d'attente (`FIFO` : First In First Out)

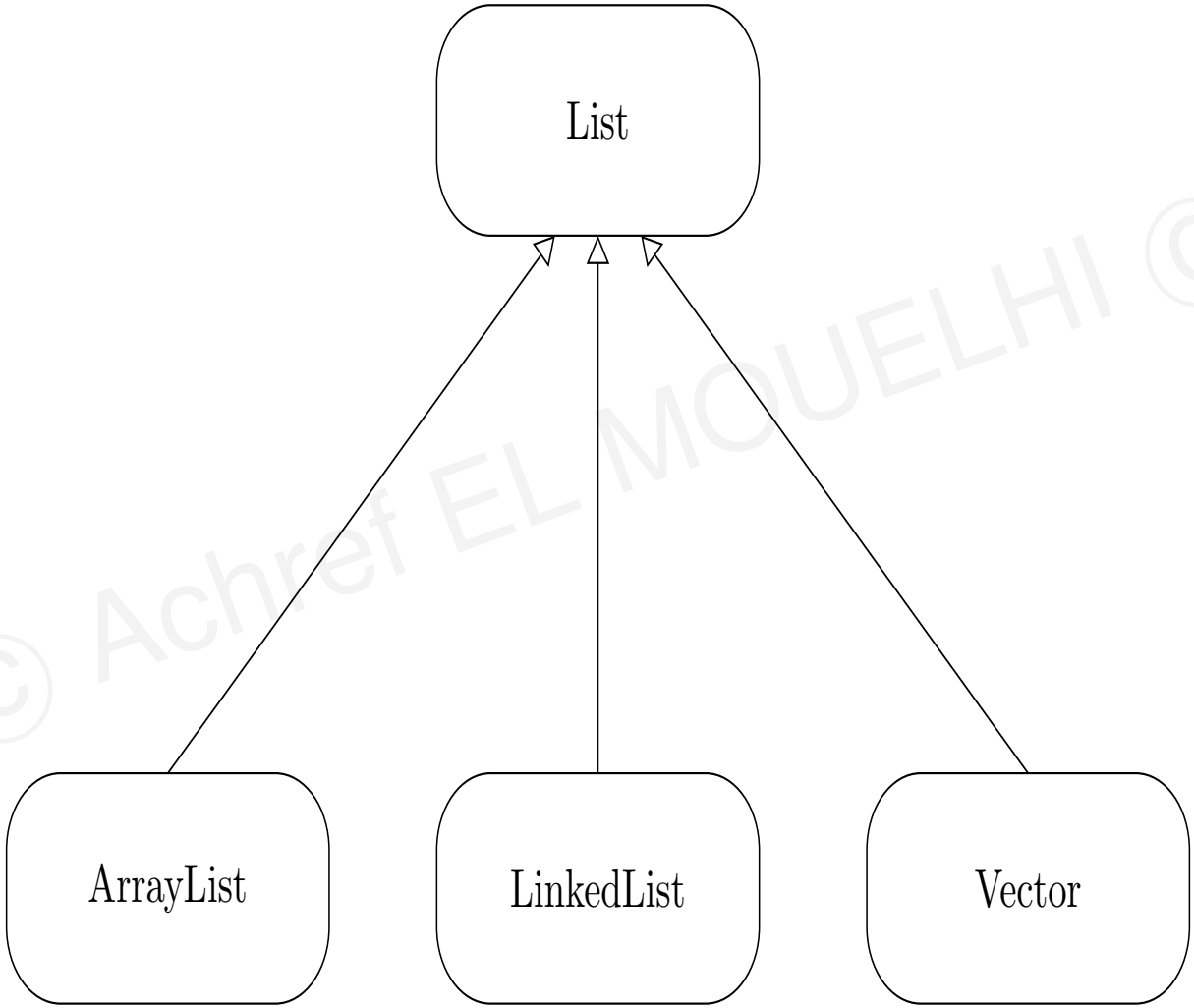
Java

Plusieurs types de collections proposés

- `List` : tableaux extensibles à volonté, accessibles via leurs indice ou valeur
- `Set` : collection qui n'accepte pas de doublons
- `Map` : collection qui exige une clé unique pour chaque élément
- `Queue` : collection gérée comme une file d'attente (`FIFO` : First In First Out)

Tous les imports de ce chapitre sont de `java.util.*;`

Java



Java

ArrayList

- Pas de limite de taille
- Possibilité de stocker tout type de données (y compris `null`)

Java

Pour créer un `ArrayList`

```
ArrayList list = new ArrayList();
```

Java

Pour créer un `ArrayList`

```
ArrayList list = new ArrayList();
```

Pour ajouter la valeur 3 à la liste

```
list.add(3);
```

Java

Pour créer un `ArrayList`

```
ArrayList list = new ArrayList();
```

Pour ajouter la valeur 3 à la liste

```
list.add(3);
```

Pour récupérer la taille de la liste

```
System.out.println(list.size());
// affiche 1
```

Java

Pour créer un `ArrayList`

```
ArrayList list = new ArrayList();
```

Pour ajouter la valeur 3 à la liste

```
list.add(3);
```

Pour récupérer la taille de la liste

```
System.out.println(list.size());
// affiche 1
```

Pour récupérer un élément de la liste

```
System.out.println(list.get(0))
// affiche 3
```

Exemple avec ArrayList

```
package org.eclipse.classes;

import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList list = new ArrayList();
    list.add(3);
    list.add("Bonjour");
    list.add(3.5);
    list.add('c');
    for(int i = 0; i < list.size(); i++){
      System.out.println("valeur d'indice " + i + " : " + list.get(i));
    }
  }
}
```

Exemple avec `ArrayList`

```
package org.eclipse.classes;

import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList list = new ArrayList();
    list.add(3);
    list.add("Bonjour");
    list.add(3.5);
    list.add('c');
    for(int i = 0; i < list.size(); i++){
      System.out.println("valeur d'indice " + i + " : " + list.get(i));
    }
  }
}
```

Le résultat est

```
valeur d'indice 0 : 3
valeur d'indice 1 : Bonjour
valeur d'indice 2 : 3.5
valeur d'indice 3 : c
```

Java

Autres méthodes de `ArrayList`

- `add(index,value)` : insère `value` à la position d'indice `index`
- `remove(index)` : supprime l'élément d'indice `index` de la liste (l'index doit être de type primitif : `int`)
- `remove(object)` : supprime l'objet `object` de la liste
- `removeAll()` : supprime tous les éléments de la liste
- `set(index, object)` : remplace la valeur de l'élément d'indice `index` de la liste par `object`
- `isEmpty()` : retourne `true` si la liste est vide, `false` sinon.
- `contains(object)` : retourne `true` si object appartient à la liste, `false` sinon.
- ...

Java

Qu'affiche le programme suivant ?

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList liste = new ArrayList();
    liste.add(0);
    liste.add("bonjour");
    liste.add(2);
    liste.remove(1);
    liste.set(1, "bonsoir");
    for(Object elt : liste) {
      System.out.print(elt + " ");
    }
  }
}
```

Java

Qu'affiche le programme suivant ?

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList liste = new ArrayList();
    liste.add(0);
    liste.add("bonjour");
    liste.add(2);
    liste.remove(1);
    liste.set(1, "bonsoir");
    for(Object elt : liste) {
      System.out.print(elt + " ");
    }
  }
}
```

0 bonsoir

Java

Exercice

Écrire un programme Java qui

1. demande à l'utilisateur de remplir une liste avec des nombres positifs de son choix, il s'arrête à la saisie de zéro
2. demande à l'utilisateur de saisir une valeur à supprimer de la liste
3. supprime la première occurrence de cette valeur de la liste
4. affiche la nouvelle liste (après suppression de la valeur demandée)

Java

Exercice

Écrire un programme Java qui

1. demande à l'utilisateur de remplir une liste avec des nombres positifs de son choix, il s'arrête à la saisie de zéro
2. demande à l'utilisateur de saisir une valeur à supprimer de la liste
3. supprime toutes les occurrences de cette valeur de la liste
4. affiche la nouvelle liste (après suppression de la valeur demandée)

Java

Exercice

Écrire un programme Java qui

1. demande à l'utilisateur de remplir une liste avec des nombres positifs de son choix, il s'arrête à la saisie de zéro
2. demande à l'utilisateur de saisir une valeur
3. affiche les positions (de toutes les occurrences) de cette valeur dans cette liste

Java

Considérons le programme suivant

```
public class Main {
  public static void main(String[] args) {
    ArrayList liste = new ArrayList();
    liste.add(0);
    liste.add(1);
    liste.add(2);
    liste.add(3);
    for(Object elt: liste) {
      if (elt.equals(0))
          liste.remove(elt);
    }
  }
}
```

Java

Considérons le programme suivant

```
public class Main {
  public static void main(String[] args) {
    ArrayList liste = new ArrayList();
    liste.add(0);
    liste.add(1);
    liste.add(2);
    liste.add(3);
    for(Object elt: liste) {
      if (elt.equals(0))
          liste.remove(elt);
    }
  }
}
```

Le résultat est l'exception suivante

```
Exception in thread "main" java.util.ConcurrentModificationException
        at java.base/java.util.ArrayList$Itr.checkForComodification(
           ArrayList.java:1009)
        at java.base/java.util.ArrayList$Itr.next(ArrayList.java:963)
        at org.eclipse.test.Main.main(Main.java:15)
```

Java

Solution : interface `Iterator`

- implémentée par la plupart des collections ayant les méthodes suivantes
- permettant de parcourir une collection
- ayant les méthodes suivantes
  - `hasNext()` : retourne `true` si l'itérateur contient d'autres éléments
  - `next()` : retourne l'élément suivant de l'itérateur
  - `remove()` : supprime le dernier objet obtenu par `next()`
  - ...

Résolvons le problème précédent avec les itérateurs

```
package org.eclipse.classes;

import java.util.ArrayList;
import java.util.ListIterator;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> liste = new ArrayList();
    liste.add(0);
    liste.add(1);
    liste.add(2);
    liste.add(3);
    ListIterator<Integer> li = liste.listIterator();
    while (li.hasNext()) {
    Integer elt = li.next();
    if (elt.equals(0))
      li.remove();
    }
    System.out.println(liste);
    // affiche [1, 2, 3]
  }
}
```

Java

`LinkedList` (liste chaînée)

- C'est une liste dont chaque élément a deux références : une vers l'élément précédent et la deuxième vers l'élément suivant.
- Pour le premier élément, l'élément précédent vaut `null`
- Pour le dernier élément, l'élément suivant vaut `null`

Java

Exemple avec `LinkedList`

```
import java.util.LinkedList;

public class Main {

  public static void main(String[] args) {
    LinkedList liste = new LinkedList();
    liste.add(5);
    liste.add("Bonjour ");
    liste.add(7.5f);
    for(int i = 0; i < liste.size(); i++)
      System.out.println("element d'indice " + i + "
          = " + liste.get(i));
  }
}
```

Java

Autres méthodes de `LinkedList`

- `addFirst(object)` : insère l'élément `object` au début de la liste
- `addLast(object)` : insère l'élément `object` comme dernier élément de la liste (exactement comme `add()`)
- ...

Java

Autres méthodes de `LinkedList`

- `addFirst(object)` : insère l'élément `object` au début de la liste
- `addLast(object)` : insère l'élément `object` comme dernier élément de la liste (exactement comme `add()`)
- ...

Remarque

- On peut parcourir une liste chaînée avec un `Iterator`
- Un itérateur est un objet qui a pour rôle de parcourir une collection

Java

`LinkedList` : parcours avec un itérateur

```
import java.util.LinkedList;
public class Main {

  public static void main(String[] args) {
    LinkedList liste = new LinkedList();
    liste.add(5);
    liste.add("Bonjour ");
    liste.add(7.5f);
    ListIterator li = liste.listIterator();
    while(li.hasNext())
      System.out.println(li.next());
  }
}
```

Java

Qu'affiche le programme suivant ?

```
import java.util.LinkedList;
public class Main {
  public static void main(String[] args) {
    LinkedList l = new LinkedList();
    l.add(0);
    l.add("bonjour");
    l.addFirst("premier");
    l.addLast("dernier");
    String s = "Salut";
    l.add(s);
    int  value = 2;
    l.add(value);
    l.remove("dernier");
    l.remove(s);
    l.remove((Object)value);
    ListIterator li = l.listIterator();
    while(li.hasNext())
      System.out.print(li.next() + " ");
  }
}
```

Java

Qu'affiche le programme suivant ?

```
import java.util.LinkedList;
public class Main {
  public static void main(String[] args) {
    LinkedList l = new LinkedList();
    l.add(0);
    l.add("bonjour");
    l.addFirst("premier");
    l.addLast("dernier");
    String s = "Salut";
    l.add(s);
    int  value = 2;
    l.add(value);
    l.remove("dernier");
    l.remove(s);
    l.remove((Object)value);
    ListIterator li = l.listIterator();
    while(li.hasNext())
      System.out.print(li.next() + " ");
  }
}
```

premier 0 bonjour

Java

On peut utiliser la généricité pour imposer un type à nos listes

```
LinkedList<Integer> liste = new LinkedList<Integer>();
```

Java

On peut utiliser la généricité pour imposer un type à nos listes

```
LinkedList<Integer> liste = new LinkedList<Integer>();
```

Ou

```
List<Integer> liste = new LinkedList<Integer>();
```

Java

On peut utiliser la généricité pour imposer un type à nos listes

```
LinkedList<Integer> liste = new LinkedList<Integer>();
```

Ou

```
List<Integer> liste = new LinkedList<Integer>();
```

La même chose pour `ArrayList`

```
List<Integer> liste = new ArrayList<Integer>();
```

Java

Considérons le tableau suivant

```
Integer [] tab = { 2, 3, 5, 1, 9 };
```

Pour convertir le tableau `tab` en liste

```
List<Integer> liste = new LinkedList(Arrays.asList(tab));
```

Java

Considérons le tableau suivant

```
Integer [] tab = { 2, 3, 5, 1, 9 };
```

Pour convertir le tableau `tab` en liste

```
List<Integer> liste = new LinkedList(Arrays.asList(tab));
```

Ou en plus simple

```
List<Integer> ent = Arrays.asList(tab);
```

Java

On peut le faire aussi sans création de tableau

```
List<Integer> liste = Arrays.asList(2, 7, 1, 3);
```

Java

On peut le faire aussi sans création de tableau

```
List<Integer> liste = Arrays.asList(2, 7, 1, 3);
```

Ou

```
List<Integer> liste = List.of(2, 7, 1, 3);
```

Java

Exercice 1 : Étant donnée la liste suivante

```
ArrayList<Integer> liste = new ArrayList(Arrays.
   asList(2, 7, 2, 1, 3, 9, 2, 4, 2));
```

Écrire un programme Java qui permet de supprimer l'avant dernière occurrence du chiffre 2 de la liste précédente

Java

Exercice 1 : Étant donnée la liste suivante

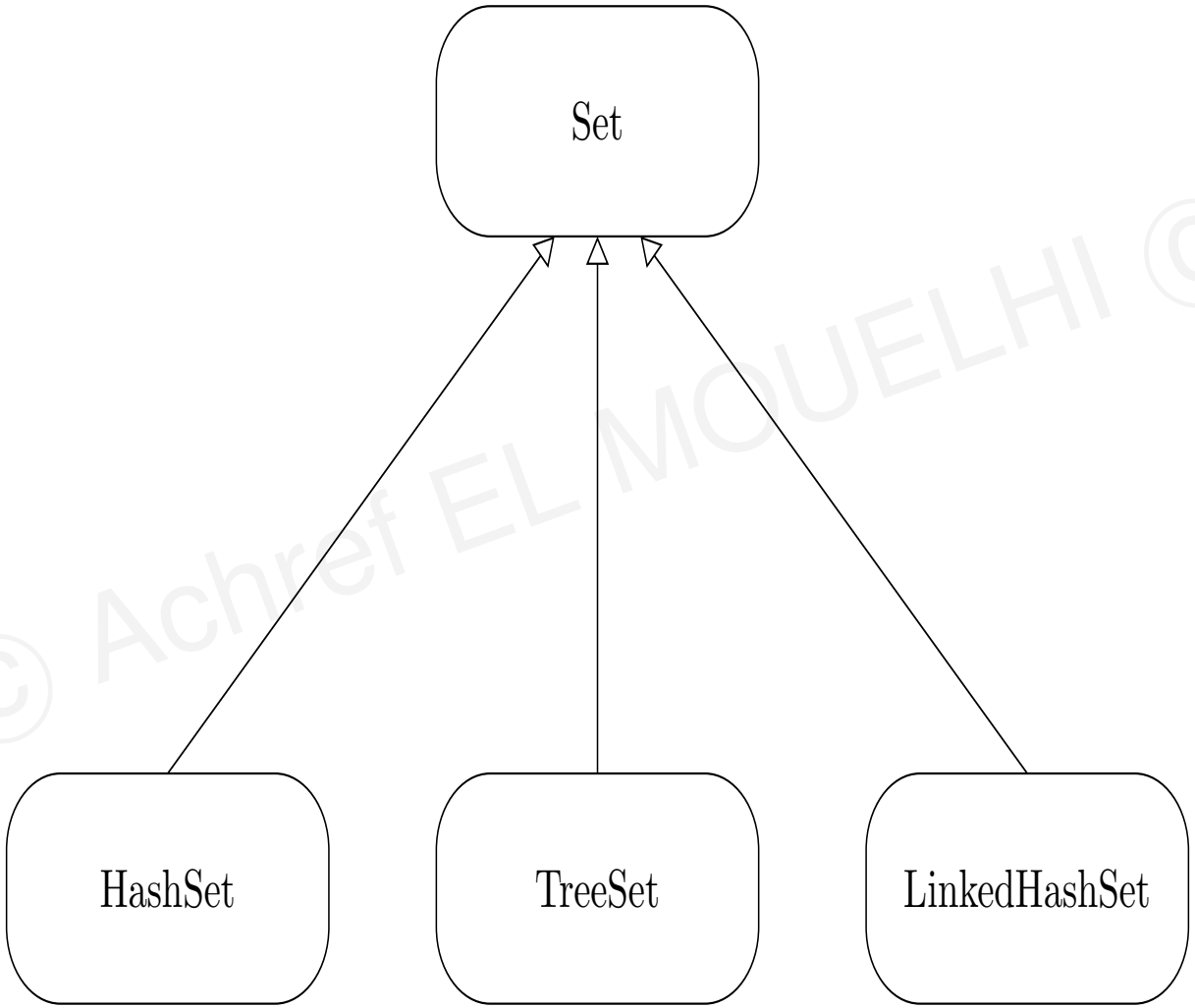
```
ArrayList<Integer> liste = new ArrayList(Arrays.
   asList(2, 7, 2, 1, 3, 9, 2, 4, 2));
```

Écrire un programme Java qui permet de supprimer l'avant dernière occurrence du chiffre 2 de la liste précédente

Solution

```
liste.remove(liste.subList(0,liste.lastIndexOf(2)).
   lastIndexOf(2));
```

Java



Java

`HashSet`

- Collection utilisant une table de hachage
- Possibilité de parcourir ce type de collection avec un objet `Iterator`
- Possibilité d'extraire de cet objet un tableau d'`Object`

Exemple avec `HashSet`

```
package org.eclipse.classes;

import java.util.HashSet;
import java.util.Iterator;

public class Main {
  public static void main(String[] args) {
    HashSet hs = new HashSet();
    hs.add("bonjour");
    hs.add(69);
    hs.add('c');
    Iterator it = hs.iterator();
    while(it.hasNext())
      System.out.println(it.next());
  }
}
```

Exemple avec `HashSet`

```
package org.eclipse.classes;

import java.util.HashSet;
import java.util.Iterator;

public class Main {
  public static void main(String[] args) {
    HashSet hs = new HashSet();
    hs.add("bonjour");
    hs.add(69);
    hs.add('c');
    Iterator it = hs.iterator();
    while(it.hasNext())
      System.out.println(it.next());
  }
}
```

Le résultat

```
c
69
bonjour
```

Java

Les éléments ne sont pas ordonnés

- `HashSet` utilise une valeur de hachage (difficile à prédire) calculée pour chaque élément.
- Cette valeur de hachage détermine l'indice de l'élément dans un tableau conteneur.
- Ainsi, l'ordre des éléments insérés n'est naturellement pas conservé.
- Ceci permet d'accéder aux éléments souhaités avec une complexité $O(1)$ en temps (mais reste coûteux en espace).

Java

Les éléments ne sont pas ordonnés

- `HashSet` utilise une valeur de hachage (difficile à prédire) calculée pour chaque élément.
- Cette valeur de hachage détermine l'indice de l'élément dans un tableau conteneur.
- Ainsi, l'ordre des éléments insérés n'est naturellement pas conservé.
- Ceci permet d'accéder aux éléments souhaités avec une complexité $O(1)$ en temps (mais reste coûteux en espace).

Remarque

Pour avoir un affichage ordonné selon l'ordre d'insertion, on peut utiliser `LinkedHashSet`.

Java

`HashSet` : exemple avec conversion en tableau

```
public class Main {
  public static void main(String[] args) {
    HashSet hs = new HashSet();
    hs.add("bonjour");
    hs.add(69);
    hs.add('c');
    Object[] obj = hs.toArray();
    for(Object o : obj)
      System.out.println(o);
  }
}
```

Java

`HashSet` : exemple avec conversion en tableau

```
public class Main {
  public static void main(String[] args) {
    HashSet hs = new HashSet();
    hs.add("bonjour");
    hs.add(69);
    hs.add('c');
    Object[] obj = hs.toArray();
    for(Object o : obj)
      System.out.println(o);
  }
}
```

Le résultat

```
c
69
bonjour
```

Java

Exemple avec `LinkedHashSet`

```
public class Main {
  public static void main(String[] args) {
    LinkedHashSet hs = new LinkedHashSet();
    hs.add("bonjour");
    hs.add(69);
    hs.add('c');
    Object[] obj = hs.toArray();
    for(Object o : obj)
      System.out.println(o);
  }
}
```

Java

Exemple avec `LinkedHashSet`

```
public class Main {
  public static void main(String[] args) {
    LinkedHashSet hs = new LinkedHashSet();
    hs.add("bonjour");
    hs.add(69);
    hs.add('c');
    Object[] obj = hs.toArray();
    for(Object o : obj)
      System.out.println(o);
  }
}
```

Ordre d'affichage = ordre d'insertion

```
bonjour
69
c
```

Java

`TreeSet`

- Possibilité de parcourir ce type de collection avec un objet `Iterator`
- Les éléments enregistrés sont automatiquement triés

Java

Exemple avec `TreeSet`

```
public class Main {
  public static void main(String[] args) {
    TreeSet ts = new TreeSet();
    ts.add(5);
    ts.add(8);
    ts.add(2);
    Iterator it = ts.iterator();
    while(it.hasNext())
      System.out.println(it.next());
  }
}
```

Java

Exemple avec `TreeSet`

```
public class Main {
  public static void main(String[] args) {
    TreeSet ts = new TreeSet();
    ts.add(5);
    ts.add(8);
    ts.add(2);
    Iterator it = ts.iterator();
    while(it.hasNext())
      System.out.println(it.next());
  }
}
```

Le résultat est ordonné

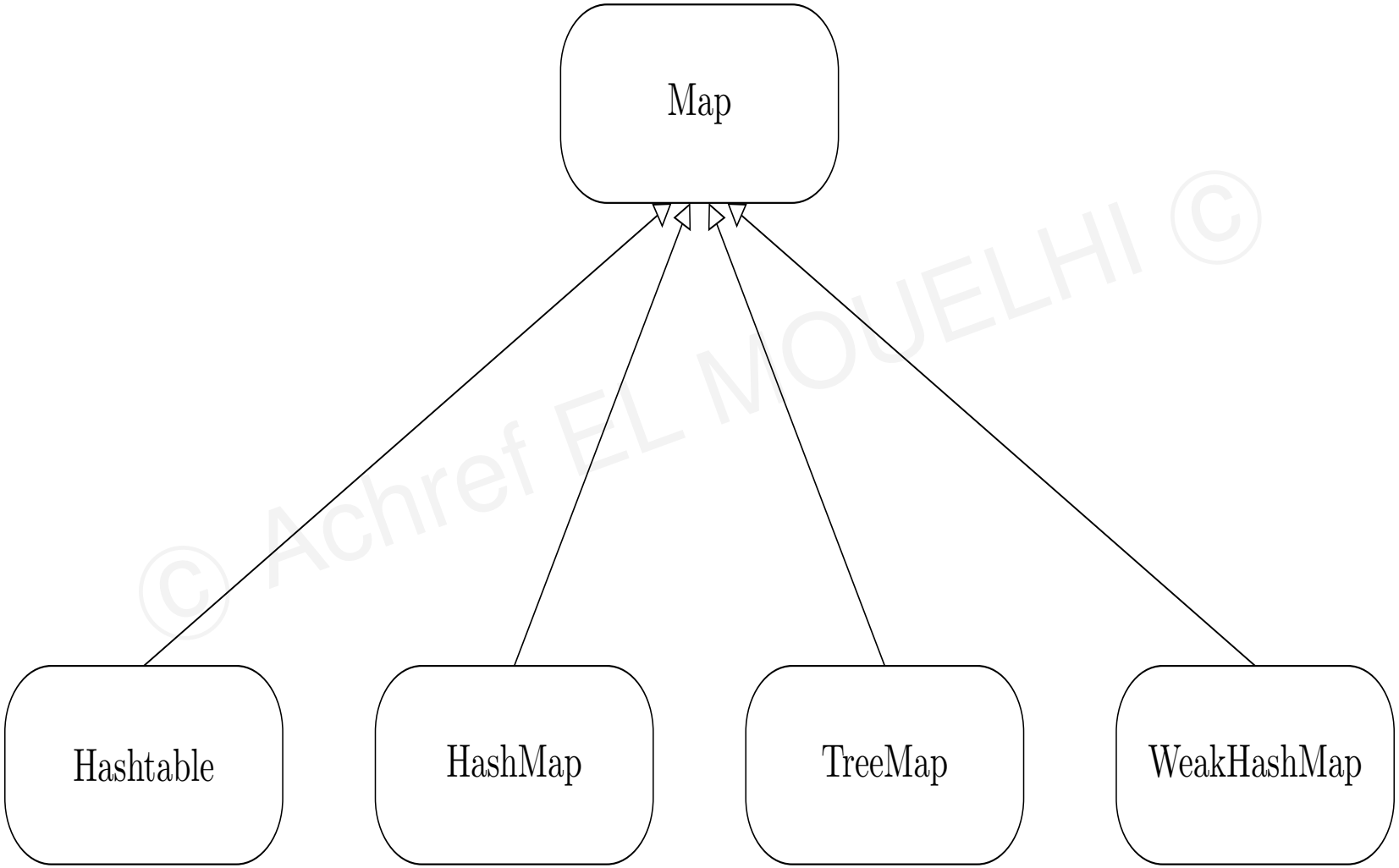
```
2
5
8
```

Java

Les éléments ne sont pas ordonnés

- `TreeSet` ordonne les données insérées.
- Elle n'accepte qu'un seul type.

Java



Java

`Hashtable`

- `Hashtable` fonctionne avec un couple (clé,valeur)
- Elle utilise une table de hachage
- Elle n'accepte pas la valeur `null`
- La clé doit être unique
- Pour la parcourir, on utilise l'objet `Enumeration`

Java

Exemple avec `Hashtable`

```
public class Main {
  public static void main(String[] args) {
    Hashtable ht = new Hashtable();
    ht.put(1, "Java");
    ht.put(2, "PHP");
    ht.put(10, "C++");
    ht.put(17, "Pascal");
    Enumeration e = ht.elements();
    while(e.hasMoreElements())
      System.out.println(e.nextElement());
  }
}
```

Java

Exemple avec `Hashtable`

```
public class Main {
  public static void main(String[] args) {
    Hashtable ht = new Hashtable();
    ht.put(1, "Java");
    ht.put(2, "PHP");
    ht.put(10, "C++");
    ht.put(17, "Pascal");
    Enumeration e = ht.elements();
    while(e.hasMoreElements())
      System.out.println(e.nextElement());
  }
}
```

Le résultat

```
C++
Pascal
PHP
Java
```

Java

`put` **ajoute si la clé n'existe pas, modifie sinon.**

```
public class Main {
  public static void main(String[] args) {
    Hashtable ht = new Hashtable();
    ht.put(1, "Java");
    ht.put(2, "PHP");
    ht.put(10, "C++");
    ht.put(17, "Pascal");
    ht.put(17, "Cobol");
    Enumeration e = ht.elements();
    while(e.hasMoreElements())
      System.out.println(e.nextElement());
  }
}
```

Java

`put` **ajoute si la clé n'existe pas, modifie sinon.**

```
public class Main {
  public static void main(String[] args) {
    Hashtable ht = new Hashtable();
    ht.put(1, "Java");
    ht.put(2, "PHP");
    ht.put(10, "C++");
    ht.put(17, "Pascal");
    ht.put(17, "Cobol");
    Enumeration e = ht.elements();
    while(e.hasMoreElements())
      System.out.println(e.nextElement());
  }
}
```

Le résultat

```
C++
Cobol
PHP
Java
```

Java

Autres méthodes de la classe `Hashtable`

- `isEmpty()` retourne `true` si l'objet est vide, `false` sinon.
- `contains(value)` retourne `true` si la valeur existe dans la `Hashtable`, `false` sinon.
- `containsKey(key)` retourne `true` si la clé existe dans la `Hashtable`, `false` sinon.
- `elements()` retourne une énumération des éléments de l'objet
- `keys()` retourne la liste des clés sous forme d'énumération

Java

`HashMap`

- `HashMap` fonctionne aussi avec un couple (clé,valeur)
- Elle utilise aussi une table de hachage
- `HashMap` accepte la valeur `null`
- La clé doit être unique
- Pour la parcourir, on utilise un objet `Set`

Java

Exemple avec `HashMap`

```
package org.eclipse.classes;

import java.util.HashMap;
import java.util.Set;

public class Main {
  public static void main(String[] args) {
    HashMap<Integer, String> hm = new HashMap();
    hm.put(1, "Java");
    hm.put(2, "PHP");
    hm.put(10, "C++");
    hm.put(17, null);
    Set s = hm.entrySet();
    Iterator it = s.iterator();
    while(it.hasNext())
      System.out.println(it.next());
  }
}
```

Java

Pour afficher la clé et la valeur, on peut utiliser l'`Entry`

```
import java.util.HashMap;
public class Main {
  public static void main(String[] args) {
    HashMap<Integer, String> hm = new HashMap();
    hm.put(1, "Java");
    hm.put(2, "PHP");
    hm.put(10, "C++");
    hm.put(17, null);
    for (Entry<Integer, String> entry : hm.entrySet()) {
      System.out.println(entry.getKey() +  " " + entry.getValue
          ());
    }
  }
}
```

Java

Étant donné ce dictionnaire

```
HashMap<String, Integer> repetition = new HashMap();
repetition.put("Java",2);
repetition.put("PHP",5);
repetition.put("C++",1);
repetition.put("HTML",4);
```

Exercice

Écrire un programme Java qui permet de répéter l'affichage de chaque clé de ce dictionnaire selon la valeur associée

Résultat attendu (l'ordre n'a pas d'importance) :

```
JavaJava PHPPHPPHPPHPPHP C++ HTMLHTMLHTMLHTML
```

Java

Exercice 2 : Étant donnée la liste suivante :

```
List list = Arrays.asList(2,5,"Bonjour",true,'c',"3"
   ,"b",false,10);
```

Écrire un programme Java qui permet de stocker dans un dictionnaire (`Map`) les types contenus dans la liste `list` ainsi que le nombre d'éléments de cette liste appartenant à chaque type.

Résultat attendu :

```
Integer=3
Character=1
String=3
Boolean=2
```

Java

Une solution possible

```
HashMap<String, Integer> compteur = new HashMap<>();
for(Object elt : list) {
  String type = elt.getClass().getSimpleName();
  if (compteur.containsKey(type)){
    compteur.put(type, compteur.get(type)+1);
  }
  else {
    compteur.put(type, 1);
  }
}
for (Entry<String, Integer> entry : compteur.entrySet()) {
  System.out.println(entry.getKey() +  " " + entry.getValue());
}
```

Java

Pour créer une entrée, on utilise la méthode `entry()`

```
var x = Map.entry(3, "JavaScript");
```

Java

Pour créer une entrée, on utilise la méthode `entry()`

```
var x = Map.entry(3, "JavaScript");
```

Pour créer une `Map` en utilisant plusieurs entrées prédéfinies

```
var x = Map.entry(3, "JavaScript");
var y = Map.entry(2, "HTML");
var z = Map.entry(1, "CSS");

Map<Integer, String> map = Map.ofEntries(x,y,z);
```

Java

Pour créer une entrée, on utilise la méthode `entry()`

```
var x = Map.entry(3, "JavaScript");
```

Pour créer une `Map` en utilisant plusieurs entrées prédéfinies

```
var x = Map.entry(3, "JavaScript");
var y = Map.entry(2, "HTML");
var z = Map.entry(1, "CSS");

Map<Integer, String> map = Map.ofEntries(x,y,z);
```

Pour afficher

```
for (Entry<Integer, String> entry : map.entrySet()) {
  System.out.println(entry.getKey() +  " " + entry.getValue());
}
```

Pour créer une `Map` et l'initialiser, on peut utiliser la méthode `of`

```
Map<Integer, String> map = Map.of(3, "JavaScript",
                                  2, "HTML",
                                  1, "CSS");

for (Entry<Integer, String> entry : map.entrySet()) {
  System.out.println(entry.getKey() + " " + entry.getValue());
}
```

Pour créer une `Map` et l'initialiser, on peut utiliser la méthode `of`

```
Map<Integer, String> map = Map.of(3, "JavaScript",
                                  2, "HTML",
                                  1, "CSS");

for (Entry<Integer, String> entry : map.entrySet()) {
  System.out.println(entry.getKey() + " " + entry.getValue());
}
```

Pour créer une `HashMap` et l'initialiser en utilisant la méthode `of`

```
HashMap<Integer, String> map = new HashMap(Map.of(
                                  3, "JavaScript",
                                  2, "HTML",
                                  1, "CSS")
);

for (Entry<Integer, String> entry : map.entrySet()) {
  System.out.println(entry.getKey() + " " + entry.getValue());
}
```

Java

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

public class Main {
   public static void main(String[] args) {
      List<String> lettres = new ArrayList<String>();
      lettres.add("d");
      lettres.add("b");
      lettres.add("a");
      lettres.add("c");
      Collections.sort(lettres); // pour trier la liste
      System.out.println(lettres);
      Collections.shuffle(lettres); // pour desordonner la liste
      System.out.println(lettres);
      List<String> sub = lettres.subList(1, 2); // extraire une sous-
         liste
      System.out.println(sub);
      Collections.reverse(sub); // pour trier la liste dans le sens
         decroissant
      System.out.println(sub);
   }
}
```

Java

`ArrayList` VS `LinkedList`

- `ArrayList` est plus rapide pour l'opération de recherche (`get`)
- `LinkedList` est plus rapide pour des opérations d'insertion et de suppression
- `LinkedList` utilise un chaînage double (deux pointeurs) d'où une consommation de mémoire plus élevée.

Java

`ArrayList` vs `LinkedList`

- `ArrayList` est plus rapide pour l'opération de recherche (`get`)
- `LinkedList` est plus rapide pour des opérations d'insertion et de suppression
- `LinkedList` utilise un chaînage double (deux pointeurs) d'où une consommation de mémoire plus élevée.

Remarques

- `Map` à utiliser lorsqu'on veut rechercher ou accéder à une valeur via une clé de recherche
- `Set` à utiliser si on n'accepte pas de doublons dans la collection
- `List` accepte les doublons permet l'accès à un élément via son indice et les éléments sont insérés dans l'ordre (pas forcément triés)