Java : 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

Collections ?

- Classes/interfaces
- Permettant de gérer dynamiquement les données

Java

Question

Pourquoi ne pas utiliser les tableaux ?

Java

Question

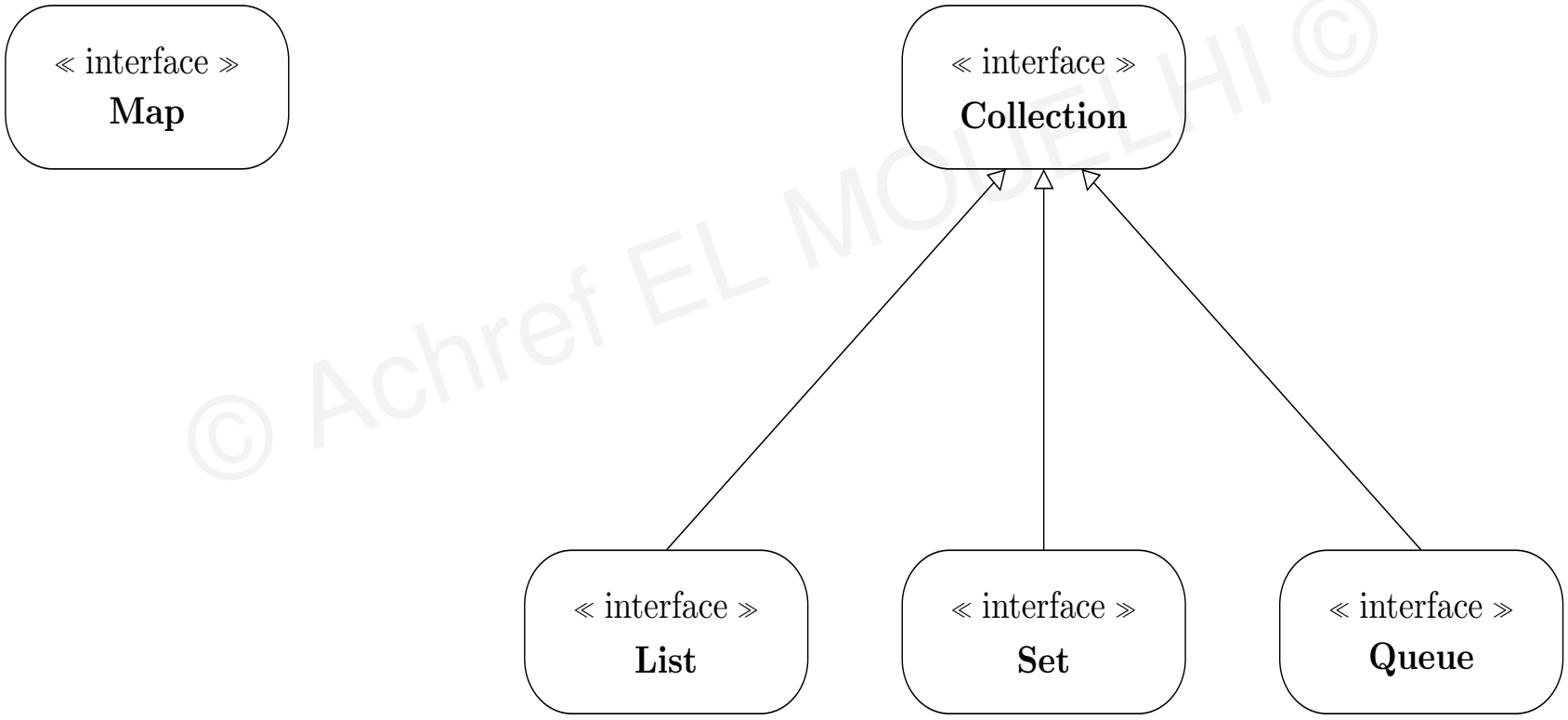
Pourquoi ne pas utiliser les tableaux ?

Réponse

- Taille à spécifier à la création du tableau
- Taille fixe : impossible de la dépasser (Pas d'allocation dynamique comme en **C**, **C++**)
- Impossible de supprimer ou d'ajouter un élément au milieu du tableau
- Un tableau $\Rightarrow$ un seul type de données

Java

Framework Collection



Java

Plusieurs types de collections proposés

- `List` : collection dynamique en taille et en type, pouvant contenir des éléments accessibles via leurs indice.
- `Set` : collection n'acceptant pas les doublons.
- `Map` : collection de (clé, valeur).
- `Queue` : collection gérée comme une file d'attente (`FIFO` : First In First Out).

Java

Plusieurs types de collections proposés

- `List` : collection dynamique en taille et en type, pouvant contenir des éléments accessibles via leurs indice.
- `Set` : collection n'acceptant pas les doublons.
- `Map` : collection de (clé, valeur).
- `Queue` : collection gérée comme une file d'attente (`FIFO` : First In First Out).

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

Java

Quelques méthodes de l'interface `Collection`

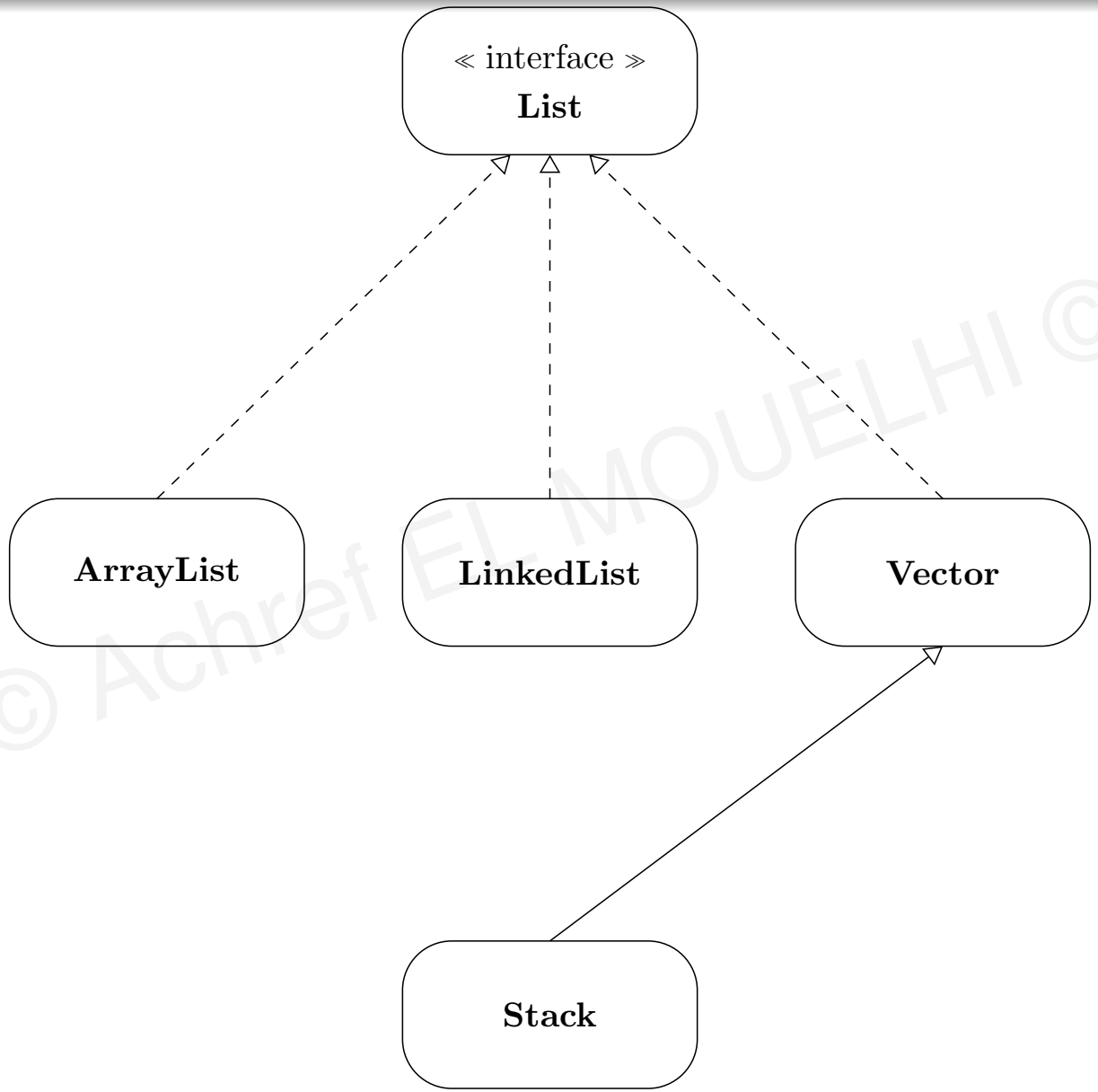
- `add`
- `remove`
- `size`
- `isEmpty`
- `clear`
- ...

Java

L'interface `List`

- Introduite dans **Java 1.2**
- Pouvant contenir des doublons et des valeurs `null`
- Possibilité de la parcourir et la modifier avec un objet `ListIterator`

Java



	ArrayList	LinkedList	Vector
Doublon	oui	oui	oui
Ordre d'insertion préservé	oui	oui	oui
Élément indexé	oui	oui	oui
Synchronized	non	non	oui
Valeur `null`	*	*	*

Explication

Synchronized ou **Thread-safe** : utilisation concurrente dans un environnement multi-thread.

Java

ArrayList

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

Pour créer un `ArrayList`

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

Pour créer un `ArrayList`

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

Pour ajouter la valeur 3 à la liste

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

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 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
```

Java

Exemple avec `ArrayList`

```
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));
}
```

Java

Exemple avec `ArrayList`

```
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

On peut aussi utiliser Double-Brace Initialization

```
ArrayList list = new ArrayList() {{
    add(3);
    add("Bonjour");
    add(3.5);
    add('c');
}};

for (int i = 0; i < list.size(); i++){
    System.out.println("valeur d'indice " + i + " : " + list.get(i));
}
```

Java

On peut aussi utiliser Double-Brace Initialization

```
ArrayList list = new ArrayList() {{
    add(3);
    add("Bonjour");
    add(3.5);
    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 ?

```
ArrayList liste = new ArrayList();
liste.add(0);
liste.add("bonjour");
liste.add(2);
liste.remove(1);
liste.set(1, "bonsoir");

for (var elt: liste) {
    System.out.print(elt + " ");
}
```

Java

Qu'affiche le programme suivant ?

```
ArrayList liste = new ArrayList();
liste.add(0);
liste.add("bonjour");
liste.add(2);
liste.remove(1);
liste.set(1, "bonsoir");

for (var elt: liste) {
    System.out.print(elt + " ");
}
```

Résultat

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 la liste

Considérons le code suivant

```
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);
    }

}
```

Considérons le code suivant

```
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
- Contenant 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()`
  - ...
- Permettant de parcourir une collection

Java

Résolvons le problème précédent avec les itérateurs

```
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` (création, initialisation et accès aux éléments similaires à celle de `ArrayList`)

```
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("élément 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()`)
- ...

Remarques

On peut parcourir une liste chaînée avec un `Iterator`

Java

Parcourir `LinkedList` avec un itérateur

```
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());
}
```

Qu'affiche le programme suivant ?

```
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() + " ");
}
```

Qu'affiche le programme suivant ?

```
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() + " ");
}
```

Résultat

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 [depuis Java 9]

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

Java

Étant donnée la liste suivante

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

Java

Étant donnée la liste suivante

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

Exercice 1

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

Java

Étant donnée la liste suivante

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


Exercice 1

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

Solution

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

Java

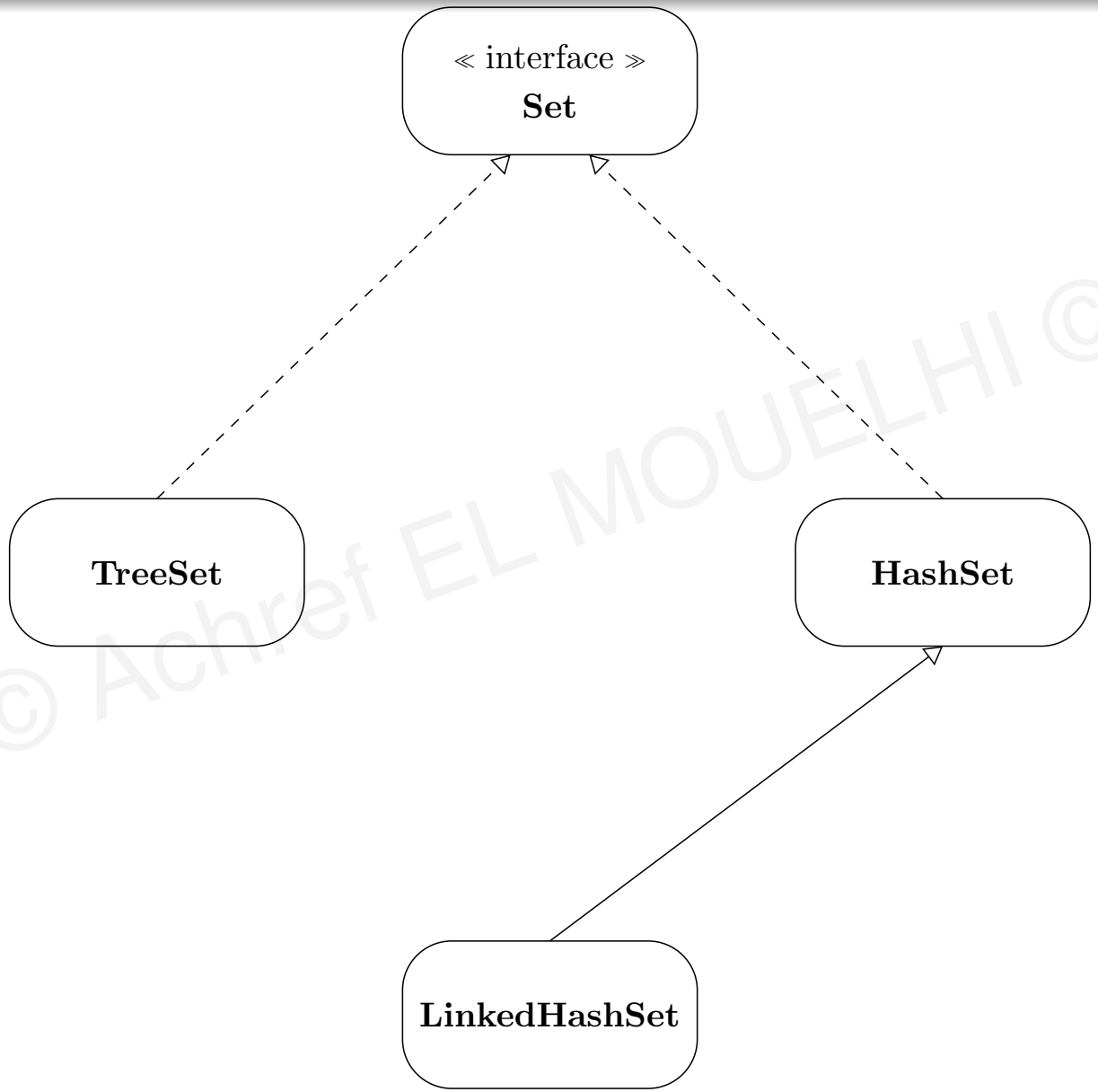
L'interface `Set`

- Introduite dans **Java 1.2**
- Pas de doublons
- Pas d'accès aux éléments via l'indice
- Possibilité de la parcourir avec un objet `Iterator`

Java

	List	Set
Doublon	oui	non
Élément indexé	oui	non
Valeur `null`	*	0..1

Java



Java

	HashSet	LinkedHashSet	TreeSet
Doublon	non	non	non
Ordre d'insertion préservé	non	oui	non
Données triées	non	non	oui
Plusieurs types autorisés	oui	oui	non
Synchronized	non	non	non
Valeur `null`	1	1	0

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`

Java

Exemple avec `HashSet`

```
HashSet hs = new HashSet();
hs.add("bonjour");
hs.add(69);
hs.add('c');
for (Object o : hs) {
        System.out.println(o);
}
```

Java

Exemple avec `HashSet`

```
HashSet hs = new HashSet();
hs.add("bonjour");
hs.add(69);
hs.add('c');
for (Object o : hs) {
        System.out.println(o);
}
```


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 utilise `LinkedHashSet`.

Java

Exemple avec `LinkedHashSet`

```
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`

```
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.
- Avec ou sans généricité $\Rightarrow$ un seul type accepté.

Exemple avec `TreeSet`

```
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());
}
```

Exemple avec `TreeSet`

```
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

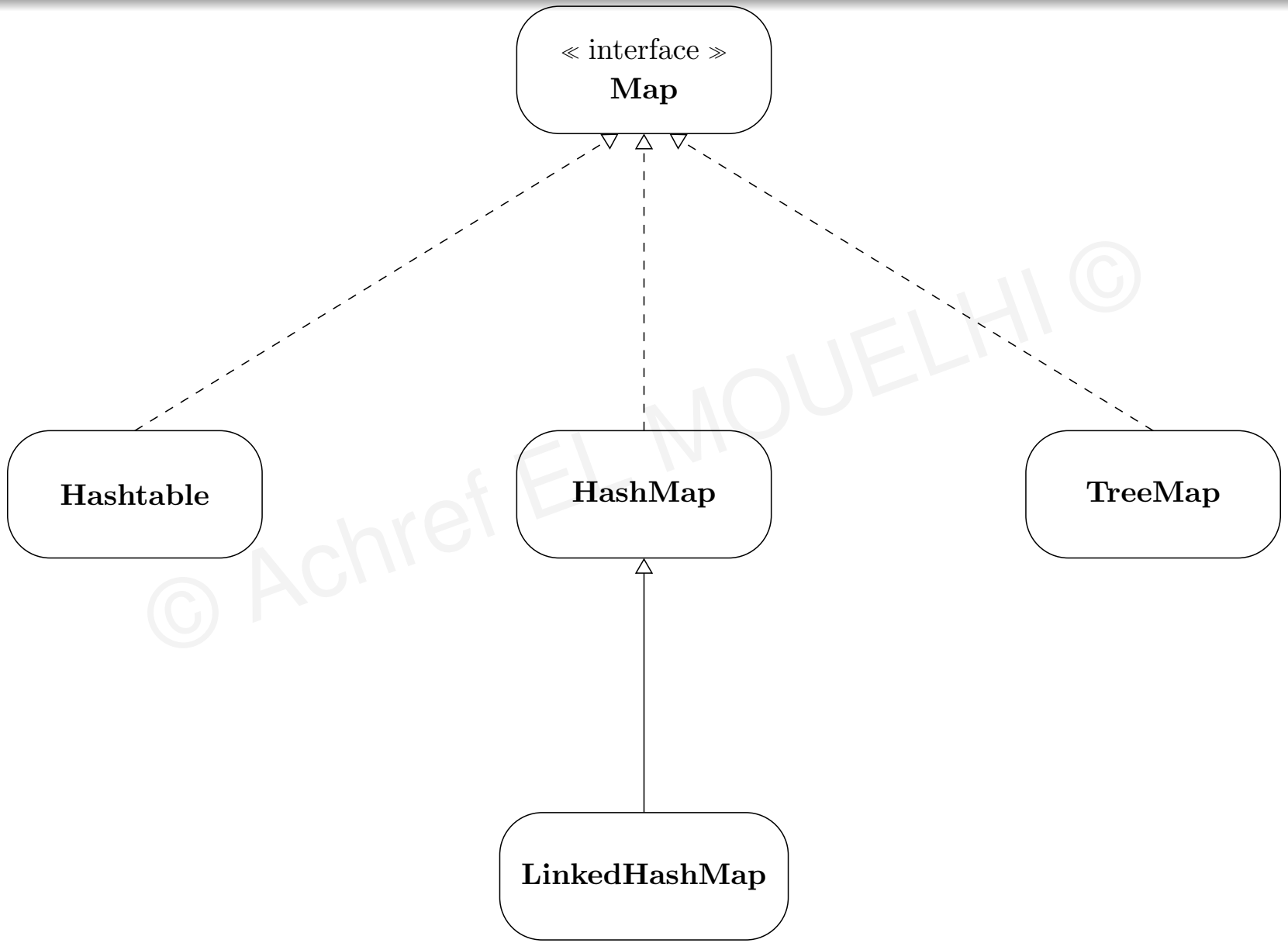
```
2
5
8
```

Java

L'interface `Map`

- Introduite dans **Java 1.2**
- Remplaçante de la classe abstraite `Dictionary` introduite dans **Java 1.1**
- Conçu pour contenir des éléments de type (clé, valeur) appelés : `Entry`
- Clé unique

Java



Java

	Hashtable	HashMap	LinkedHashMap	TreeMap
Utilisation de table de hachage	oui	oui	oui	non
Ordre d'insertion préservé	non	non	oui	non
Données triées selon la valeur	non	non	non	oui
Synchronized	oui	non	non	non
Clé `null`	0	1	1	0
Valeur `null`	0	*	*	*

Java

Hashtable

- `Hashtable` fonctionne avec un couple (clé, valeur).
- Elle utilise une table de hachage.
- Elle n'accepte pas la valeur `null` ni pour les clés, ni pour les valeurs.

Java

Pour créer un `Hashtable`

```
Hashtable ht = new Hashtable();
```

Java

Pour créer un `Hashtable`

```
Hashtable ht = new Hashtable();
```

Ou en utilisant la généricité

```
Hashtable<Integer, String> ht = new Hashtable<Integer, String>();
```

Java

Pour créer un `Hashtable`

```
Hashtable ht = new Hashtable();
```

Ou en utilisant la généricité

```
Hashtable<Integer, String> ht = new Hashtable<Integer, String>();
```

Depuis Java 7, les types génériques peuvent être uniquement spécifiés à la déclaration

```
Hashtable<Integer, String> ht = new Hashtable();
```

Java

Pour ajouter des éléments

```
ht.put(1, "Java");
ht.put(2, "PHP");
ht.put(10, "C++");
ht.put(17, "Pascal");
```

Java

Pour ajouter des éléments

```
ht.put(1, "Java");
ht.put(2, "PHP");
ht.put(10, "C++");
ht.put(17, "Pascal");
```


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

```
ht.put(17, "Cobol");
```

Java

Quelques méthodes de la classe `Hashtable`

- `size()` retourne le nombre d'`Entry`.
- `isEmpty()` retourne `true` si l'objet est vide, `false` sinon.
- `containsValue(value)` (ou son raccourci `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
- `remove(key)` supprime un `Entry` selon la clé.
- `remove(key, value)` supprime un `Entry` selon le couple clé/valeur.
- ...

Java

Pour parcourir un `Map`, on utilise `Entry`

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

Java

Pour parcourir un `Map`, on utilise `Entry`

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

Résultat

```
10 C++
17 Pascal
2 PHP
1 Java
```

Java

Quelques méthodes de la classe `Entry`

- `getKey()` : permet de récupérer la clé
- `getValue()` : permet de récupérer la valeur
- `setValue()` : permet de modifier la valeur

Java

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(ht.get(1));
// affiche Java
```

Java

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(ht.get(1));
// affiche Java
```

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(ht.get(3));
// affiche null
```

Java

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(ht.get(1));
// affiche Java
```

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(ht.get(3));
// affiche null
```

Pour spécifier une valeur par défaut si la clé n'existe pas, on utilise `getOrDefault()`

```
System.out.println(ht.getOrDefault(3, "C"));
// affiche C
```

Java

Exerice

Écrire un code **Java** qui permet de remplacer la valeur `PHP`, si elle existe, par `C#`.

Java

HashMap

- `HashMap` fonctionne aussi avec un couple (clé, valeur).
- Elle utilise aussi une table de hachage.
- `HashMap` autorise une seule clé avec la valeur `null` et autant de valeurs `null` que possible.
- La clé doit être unique.

Java

Exemple avec `HashMap`

```
HashMap hm = new HashMap();
```

Java

Exemple avec `HashMap`

```
HashMap hm = new HashMap();
```

Avec la généricité, on peut indiquer un type pour la clé et la valeur

```
HashMap<Integer, String> hm = new HashMap<Integer, String>();
```

Java

Exemple avec `HashMap`

```
HashMap hm = new HashMap();
```

Avec la généricité, on peut indiquer un type pour la clé et la valeur

```
HashMap<Integer, String> hm = new HashMap<Integer, String>();
```

Depuis Java 7, les types génériques peuvent être spécifiés uniquement à la déclaration

```
HashMap<Integer, String> hm = new HashMap<>();
```

Java

Pour ajouter des éléments à un `HashMap`

```
hm.put(1, "Java");
hm.put(2, "PHP");
hm.put(10, "C++");
hm.put(17, null);
```

Java

Pour ajouter des éléments à un `HashMap`

```
hm.put(1, "Java");
hm.put(2, "PHP");
hm.put(10, "C++");
hm.put(17, null);
```

Pour parcourir un `HashMap` on peut utiliser un `Entry`

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

Java

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(hm.get(1));
// affiche Java
```

Java

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(hm.get(1));
// affiche Java
```

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(hm.get(3));
// affiche null
```

Java

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(hm.get(1));
// affiche Java
```

Pour récupérer une valeur selon la clé sans parcourir la `HashMap`

```
System.out.println(hm.get(3));
// affiche null
```

Pour spécifier une valeur par défaut si la clé n'existe pas, on utilise `getOrDefault()`

```
System.out.println(hm.getOrDefault(3, "C"));
// affiche C
```

Java

Quelques méthodes de la classe `HashMap`

- `size()` retourne le nombre d'`Entry`.
- `isEmpty()` retourne `true` si l'objet est vide, `false` sinon.
- `containsValue(value)` (Pas de raccourci `contains(value)` pour `HashMap`) retourne `true` si la valeur existe dans la `Hashtable`, `false` sinon.
- `containsKey(key)` retourne `true` si la clé existe dans la `Hashtable`, `false` sinon.
- `remove(key)` supprime un `Entry` selon la clé.
- `remove(key, value)` supprime un `Entry` selon le couple clé/valeur.
- `keySet()` retourne un `Set` des clés.
- ...

Java

Étant donné le `HashMap` suivant

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

Exercice 1

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

Considérons le `HashMap` suivant

```
Map<Integer, String> fcb = new HashMap<>();
fcb.put(10, "Messi");
fcb.put(9, "Suarez");
fcb.put(23, "Umtiti");
fcb.put(4, "Rakitic");
```


Exercice 2

Écrire un programme **Java** qui permet de demander à l'utilisateur de saisir un nom (valeur) et affiche le numéro associé à ce nom (clé) s'il existe, -1 sinon.

Java

Étant donnée la liste suivante :

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

Exercice 3

Écrire un programme Java qui permet de stocker dans un dictionnaire (`Map`) les types présents dans `list` ainsi que le nombre d'occurrences de chacun (type).

Résultat attendu :

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

Question

Quelle différence entre `Collections` et `Collection` en **Java** ?

Java

Question

Quelle différence entre `Collections` et `Collection` en **Java** ?

`Collection`

- Interface générique introduite dans **Java 1.2**
- Racine de la hiérarchie des collections en **Java**
- Implémentant l'interface `Iterable`

`Collections`

- Classe utilitaire introduite dans **Java 1.2**
- Contenant uniquement des méthodes statiques pour les collections **Java**
- Héritant de la classe `Object`

Remarques

Quelques méthodes statiques de la classe `Collections`

- `Collections.sort()`
- `Collections.min()`
- `Collections.max()`
- `Collections.binarySearch()`
- ...

Java

Exemple

```
List<String> lettres = new ArrayList<String>();
lettres.add("d");
lettres.add("b");
lettres.add("a");
lettres.add("c");

// pour trier la liste
Collections.sort(lettres);
System.out.println(lettres);
// [a, b, c, d]

// pour désordonner la liste
Collections.shuffle(lettres);
System.out.println(lettres);
// [d, a, b, c]

// pour trier la liste dans le sens décroissant
Collections.reverse(sub);
System.out.println(sub);
// [d, a, c, b]
```

Java

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)

Java

	ArrayList	**LinkedList**
`get()`	$O(1)$	$O(n)$
`add()`	$O(1)$	$O(1)$
`contains()`	$O(n)$	$O(n)$
`remove()`	$O(n)$	$O(1)$

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.