



JAVA – COLLECTIONS

Wijin
v1.0 2024



PLAN

- Présentation des collections
- Itérer sur une collection
- D'autres types de collection

JAVA – PRÉSENTATION DES COLLECTIONS



Java – Collections ?

- Les tableaux Java vu précédemment permettent de stocker des **collections** de données
- Il existe en Java une façon plus simple de gérer ces collections
- « Java Collections » désigne un ensemble d'interfaces et de classes permettant de stocker, trier et traiter les données (classes utiles avec multiples méthodes)



Java – Collections

- La plupart des collections se trouve dans le package **java.util**
- Une **collection** est un « objet » qui **collecte** des éléments dans une **liste**
- Une collection représente un ensemble de données **de même type**



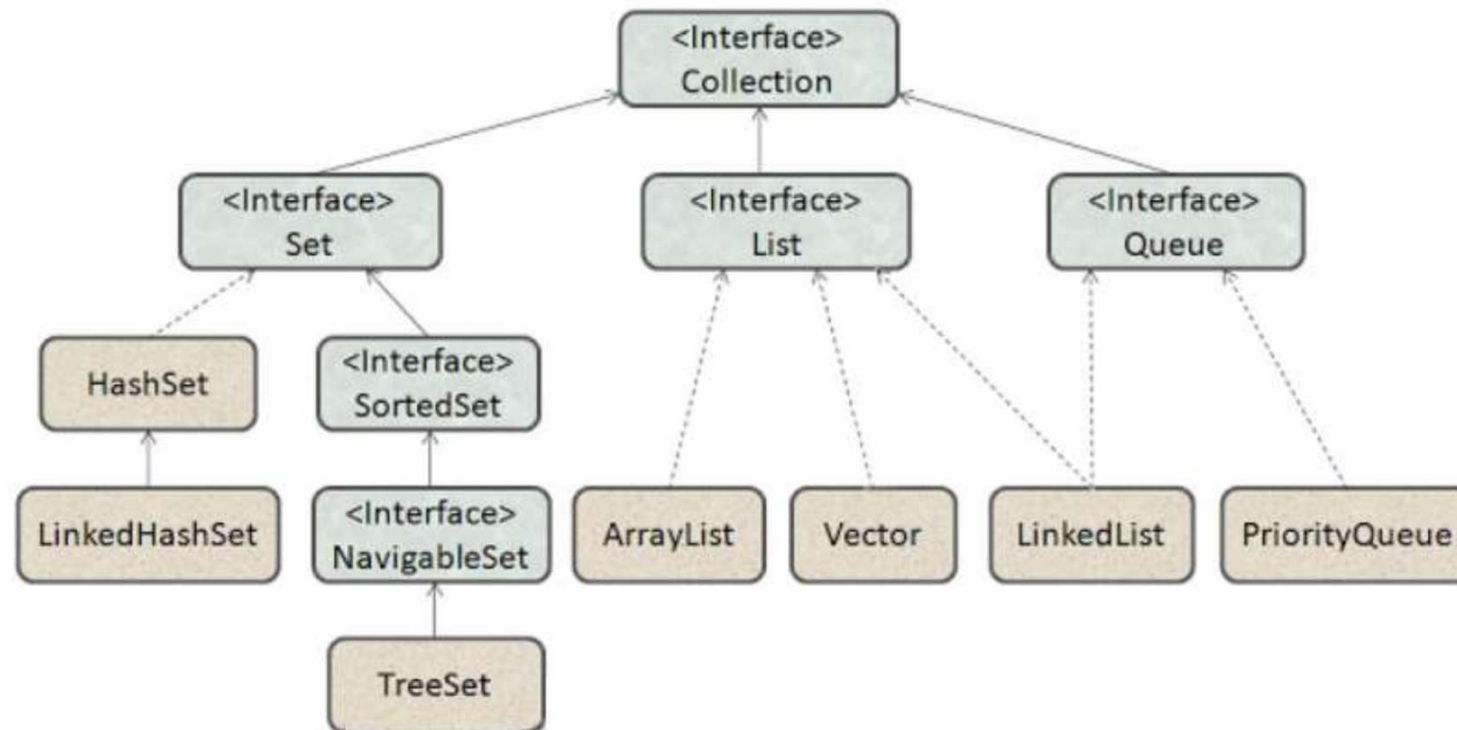
Java – Collections

- Composition des collections de Java
 - Structure **hiérarchique**
 - Deux grands **ensembles**
 - **Collection** (*java.util.Collection*)
 - **Map** (*java.util.Map*)



Java – Collections

- java.util.Collection



Java – Collections

- Le type **List** en Java
 - La classe **ArrayList** implémente l'interface **List**, qui elle-même hérite de l'interface **Collection**
 - La classe **ArrayList** dispose de l'ensemble des méthodes de l'interface **List** et de l'interface **Collection**
 - Une « **ArrayList** » peut être vue et traitée comme une interface **List**



Java – Collections

- Le type **List** en Java

- On peut écrire : *List listedechaines = new ArrayList();*

- Il n'y a pas encore la précision du **type de données** à stocker dans cette liste !

- On le précise en utilisant les **Generics** de Java :

- List<String> listedechaines = new ArrayList<>();*



Java – Collections

- Le type **List** en Java

- Chargement dans la liste :

```
List<String> listedechaines = new ArrayList<>();
```

```
listedechaines.add("chaine1");
```

Ajout de « chaine1 » dans la liste

```
listedechaines.add(1, "chaine2");
```

Ajout de « chaine2 » dans la liste,
à la position 1



Java – Collections

- Le type **List** en Java
 - Chargement à partir d'une autre liste:

```
List<String> listedechaines2 = new ArrayList<>();
```

Création d'une nouvelle liste

```
listedechaines2.addAll(listedechaines);
```

Chargement à partir de la
liste précédente



Java – Collections

- Le type **List** en Java
 - Accès aux éléments d'une liste

```
listedechaines.get(0);
```

Récupération de l'élément
de la liste situé en première
position



Java – Collections

- Le type **List** en Java
 - Récupération de l'index de la **première occurrence** du contenu d'un élément

```
List<String> maliste = new ArrayList<>();  
String element1 = "contenu 1";  
String element2 = "contenu 2";  
maliste.add(element1);  
maliste.add(element2);  
int index1 = maliste.indexOf(element1);  
int index2 = maliste.indexOf(element2);  
System.out.println("premier indexe = " + index1);  
System.out.println("deuxième indexe = " + index2);
```

Récupération de l'index de la
1^{ère} occurrence pour le
contenu de « element1 »



Java – Collections

- Le type **List** en Java
 - Récupération de l'index de la **dernière occurrence** du contenu d'un élément

```
List<String> maliste = new ArrayList<>();  
String element1 = "contenu 1";  
String element2 = "contenu 2";  
maliste.add(element1);  
maliste.add(element2);  
maliste.add(element1);  
int lastIndex = maliste.lastIndexOf(element1);  
System.out.println(« dernier index = » + lastIndex);
```

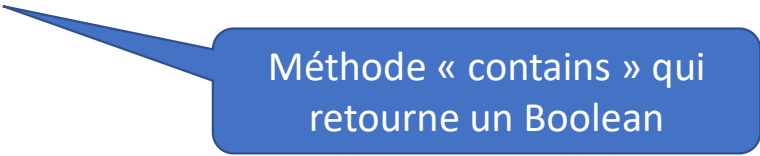
Récupération de l'index de la dernière occurrence pour le contenu de « element1 »



Java – Collections

- Le type **List** en Java
- Savoir si une liste contient ou non un élément

```
maliste.contains("contenu 1");
```



Méthode « contains » qui
retourne un Boolean



Java – Collections

- Le type **List** en Java
 - Supprimer un élément

```
maliste.remove(0);
```

Suppression par rapport à
l'index (position)

```
maliste.remove("contenu 2");
```

Suppression par rapport au
contenu



Java – Collections

- Le type **List** en Java
 - Supprimer tous les éléments

```
maliste.clear();
```



Java – Collections

- Le type **List** en Java
 - Obtenir la taille d'une liste

```
maliste.size();
```



Java – Collections

- Le type **List** en Java
- Conversion d'une liste (**List**) en **Set**
 - **Impact** : suppression des doublons (un **Set** ne contient que des **éléments uniques**)

```
List<String> list = new ArrayList<>();  
list.add("element 1");  
list.add("element 2");  
list.add("element 3");  
list.add("element 3");  
Set<String> set = new HashSet<>();  
set.addAll(list);  
for (String s : set) {  
    System.out.println(s);  
}
```

Ne contiendra qu'un seul
élément « element 3 »



Java – Collections

- Le type **List** en Java
 - Conversion d'une liste (**List**) en **Array** :

```
List<String> list = new ArrayList<>();  
list.add("element 1");  
list.add("element 2");  
list.add("element 3");  
list.add("element 3");  
Object[] objects = list.toArray();  
for (Object s : objects) {  
    System.out.println(s.toString());  
}
```

Méthode « toArray() »



Java – Collections

- Le type **List** en Java
 - Conversion d'un **Array** (tableau) en **List** :

```
String[] valeurs = new String[]{ "Ain", "Aisne", "Allier" };
```

```
List<String> liste = (List<String>) Arrays.asList(valeurs);
```

Méthode « asList() » de Arrays



Java – Collections

- Le type **List** en Java
 - Tri d'une liste (**Quicksort** par défaut)

```
List<String> departements = new ArrayList<>();  
departements.add("vaucluse");  
departements.add("allier");  
departements.add("loire");  
Collections.sort(departements);  
for (String dep : departements) {  
    System.out.println(dep);  
}
```

Méthode « sort() »



Java – Collections

- Le type **List** en Java
 - **Tri** d'une liste (**Quicksort** par défaut)
 - Attention, si le type d'objet utilisé n'implémente pas l'interface **Comparable**, ou si on souhaite trier selon un ordre différent, alors il faut fournir une implémentation spécifique de l'interface **Comparator** !
 - Exemple avec ajout d'un critère de tri sur la propriété d'un objet utilisé dans la liste



Java – Collections

- Le type **List** en Java

```
1 package collections;
2
3 public class Planete {
4
5     public String nom;
6     public Integer taille;
7     public Integer distance;
8
9     public Planete(String nom, Integer taille, Integer distance) {
10         this.nom = nom;
11         this.taille = taille;
12         this.distance = distance;
13     }
14
15 }
```

Classe **Planete** :
On va souhaiter trier les
objets dans la liste par
rapport à la valeur de la
propriété « **taille** »



Java – Collections

- Le type **List** en Java

```
List<Planete> listeDesPlanetes = new ArrayList<>();
listeDesPlanetes.add(new Planete("Mercure", 6000, 50));
listeDesPlanetes.add(new Planete("Mars", 5000, 200));
listeDesPlanetes.add(new Planete("Jupiter", 11000, 1100));

Comparator<Planete> compareurDeTaille = new Comparator<>() {
    public int compare(Planete planete1, Planete planete2) {
        return planete1.taille.compareTo(planete2.taille);
    }
};

Collections.sort(listeDesPlanetes, compareurDeTaille);

for (Planete p : listeDesPlanetes) {
    System.out.println(p.nom + " " + p.distance + " " + p.taille);
}
```

Création de la liste

Implémentation de **Comparator** : on instancie l'interface en fournissant immédiatement l'implémentation de la méthode « *compare* »

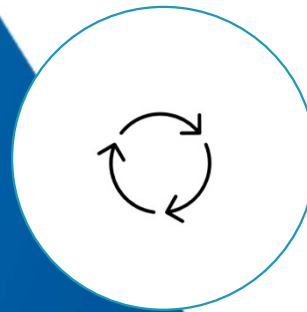
Comparaison par rapport à la propriété « **taille** »

La méthode *sort()* va utiliser notre implémentation de **Comparator**

Les planètes doivent être triées par taille !



JAVA - ITÉRER SUR UNE COLLECTION



Java – Itérer sur une collection

- Il existe plusieurs possibilités **d'itérer** sur une liste en Java
 - Utilisation de l'interface **Iterator**
 - Utiliser le **for** (:)
 - Utiliser les boucles classiques (**while**, **do**, **for**(,,))
 - Utiliser le **forEach**



Java – Itérer sur une collection

- Itérer en utilisant l'interface **Iterator**

```
List<String> list = new ArrayList<>();  
list.add("premier");  
list.add("deuxième");  
list.add("troisième");
```

```
Iterator<String> iterator = list.iterator();
```

```
while (iterator.hasNext()) {  
    String next = iterator.next();  
    System.out.println(next);  
}
```

Récupération d'un **iterator**
sur la liste

Méthode **hasNext()** pour
vérifier s'il y a toujours des
éléments dans la liste

Méthode **next()** pour
récupérer l'élément suivant
dans la liste



Java – Itérer sur une collection

- Itérer en utilisant la boucle **for**

```
for(String element : list) {  
    System.out.println(element);  
}
```



Java – Itérer sur une collection

- Itérer en utilisant la boucle **forEach**

```
for (String name : names) {  
    System.out.println(name);  
}
```

Utilisation classique du for (:)

```
names.forEach(name -> {  
    System.out.println(name);  
});
```

Utilisation du forEach :

- Pas de boucle
- Chaque élément de la liste est donné à la fonction définie dans le **forEach**

Fonction lambda !



Java – Itérer sur une collection

- Itérer en utilisant la boucle **forEach** avec un **Stream**

```
List<String> stringList = new ArrayList<String>();  
stringList.add("valeur 1");  
stringList.add("valeur 2");  
stringList.add("valeur 3");
```

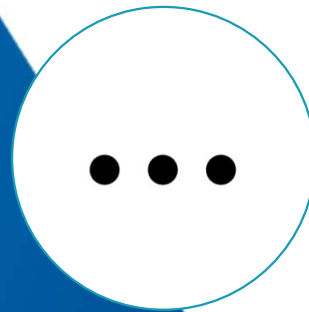
Création d'un Stream à partir
d'une collection

```
Stream<String> stream = stringList.stream();  
  
stream.forEach( element -> { System.out.println(element); } );
```

Un Stream est un flux de données !



JAVA - AUTRES TYPES DE COLLECTION



Java – Autres types de Collection

- Collections de type **Set**
 - Liste **sans duplication**
 - **Pas de garantie de l'ordre d'insertion !**
 - Principales classes : *HashSet* et *LinkedHashSet*, *SortedSet*



Java – Autres types de Collection

- Collections de type **Set**

Liste avec doublons

```
List<Integer> listNumbers = Arrays.asList(3, 9, 1, 4, 7, 2, 5, 3, 8, 9, 1, 3, 8, 6);  
System.out.println(listNumbers);
```

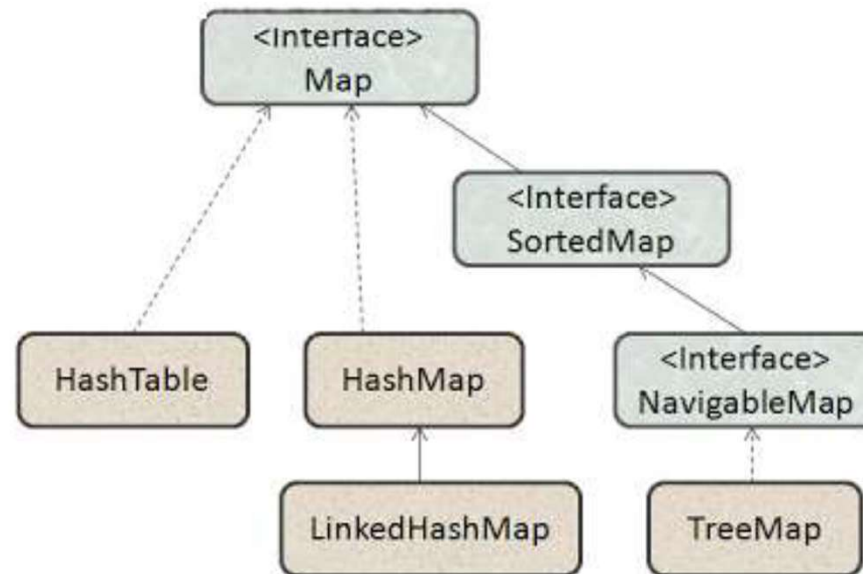
```
Set<Integer> uniqueNumbers = new HashSet<>(listNumbers);  
System.out.println(uniqueNumbers);
```

Le Set ne contient pas les doublons !



Java – Autres types de Collection

- Collections de type **Map**



Java – Autres types de Collection

- Collections de type **Map**
 - L'interface **Map** permet de stocker un ensemble de paires **clé/valeur** dans une table de hachage
 - Une Map ne peut pas contenir des éléments dupliqués, **chaque clé est unique**
 - **Map** est implémentée notamment avec :
 - *HashMap* : pas de garantie de l'ordre d'insertion
 - *LinkedHashMap* : garantie l'ordre d'insertion
 - *TreeMap* : stockage des éléments triés selon leur valeur



Java – Autres types de Collection

- Collections de type **Map**
 - L'interface **SortedMap** hérite de **Map** et implémente les méthodes pour ordonner les éléments dans l'ordre croissant et décroissant
 - **TreeMap** est une implémentation de **SortedMap**



Java – Autres types de Collection

- Collections de type **Map**

```
Map<Integer, String> map = new HashMap<>();  
map.put(1, "Mercure");  
map.put(2, "Vénus");  
map.put(3, "La Terre");  
map.put(4, "Mars");
```

Déclaration d'une Map avec l'implémentation **HashMap**

Ajout d'éléments avec une clé numérique et une valeur de type String



Java – Autres types de Collection

- Collections de type **Map**

Parcours de la Map (avec for auto)

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

Récupération de la clé

Récupération de la valeur



Java – Autres types de Collection

- Collections de type **Map**

Parcours de la Map (avec forEach)

```
map.forEach((k,v) -> System.out.println("entrée: " + k + ", valeur: " + v));
```

Expression lambda :
k = clé
v = valeur



JAVA - QUIZ



Java – Autres types de Collection

- Les tableaux et collections ont les mêmes possibilités dans Java

- Vrai

- Faux



Java – Autres types de Collection

- Cette instruction est valable :

List<String> listedechaines = new ArrayList<>();

• Vrai

• Faux



Java – Autres types de Collection

- Quel est le package qui contient les collections ?

• java.util

• java.array

• java.lang

• java.collection



MERCI POUR VOTRE
ATTENTION

Faites-moi part de vos remarques
concernant le cours afin qu'il soit
amélioré pour les prochaines
sessions : nicolas.sanou@wijin.tech