Introduction à la Programmation Orientée Objet

8 - Les Collections

Valentin Honoré

valentin.honore@ensiie.fr

FISA 1A

Mais où sont les objets ? ? ?

- On parle beaucoup d'objets mais
  - dans le DM on décrit nos propres structures
  - dans les exceptions, on utilise `Exception`

- Une bibliothèque Java fournit tout un tas de structures de données génériques !
  - Laquelle ? (souvenir du cours d'avant)

La bibliothèque `java.util`

- Fournit un ensemble d'interfaces abstrayant les structures de données les plus fréquemment utilisées
- Fournit un ensemble de classes mettant en œuvre ces structures de données
- Toutes ces classes et interfaces sont génériques !

Préambule

- La bibliothèque de structures de données Java fournit des méthodes utilitaires
  - Pour effectuer une comparaison par valeurs entre objets
    (Offert directement par `Object` via la méthode `equals`)
  - Pour connaître le code hachage d'un objet
    (Offert directement par `Object` via la méthode `hashCode`)
  - Pour savoir si un objet est plus petit qu'un autre
    (Offert par l'interface `Comparable` via la méthode `compareTo`)
- N'oubliez pas de lire la doc quand vous avez un doute !

Deux familles de structures de données

- La collection stocke une collection d'éléments
  - Tableaux extensibles (stocke des villes)
  - Listes chaînées (stocke des villes)
- La liste d'association associe des clés et des valeurs
  - La table de hachage (associe des noms à des villes)
  - L'arbre binaire de recherche (associe des noms à des villes)

Deux familles de structures de données

- La collection stocke une collection d'éléments
  - Tableaux extensibles — `java.util.ArrayList<E>`
  - Listes chaînées — `java.util.LinkedList<E>`
- La liste d'association associe des clés et des valeurs
  - La table de hachage — `java.util.HashMap<K, V>`
  - L'arbre binaire de recherche — `java.util.TreeMap<K, V>`

Plan du cours

La collection d'éléments

- L'interface `Collection<E>` représente une collection
  - `boolean add(E e)` : ajoute l'élément `e`
  - `boolean remove(Object o)` : supprime l'élément `o`
  - `boolean contains(Object o)` : vrai si collection contient `o`

- `contains` et `remove` prennent en argument un `Object` et non un `E`, pourquoi ?

La collection d'éléments

- L'interface `Collection<E>` représente une collection
  - `boolean add(E e)` : ajoute l'élément `e`
  - `boolean remove(Object o)` : supprime l'élément `o`
  - `boolean contains(Object o)` : vrai si collection contient `o`

- `contains` et `remove` prennent en argument un `Object` et non un `E`, pourquoi ?

  pour permettre à l'utilisateur de tester la présence ou de supprimer un élément en utilisant une instance d'une autre classe

Exemple d'utilisation des collections [TestCollection.java]

```
import  java.util.Collection;
import  java.util.ArrayList;

class Monster {
    String name;
    int health;
    Monster(String name) {
        this.name = name; this.health = 10;
    }
}

class TestCollection {
     public static void main(String[] args) {
        Collection<Monster> col = new ArrayList<>();
        Monster m = new Monster("Pikachu");
        col.add(m);
        if(col.contains(m)) {
            System.out.println("Facile les collections...");
        }
    }
}
```

La classe Iterator

- Un `Iterator` permet de parcourir une collection
  - `E next()` : renvoie l'élément suivant et avance dans la collection
  - `boolean hasNext()` : renvoie vrai si il existe un suivant
  - `void remove()` : supprime l'élément courant

- Exemple d'utilisation

```
Collection<Monster> col = new LinkedList<>();
... /* ajouter des monstres à la collection ici */

Iterator<Monster> it = col.iterator();

while(it.hasNext()) {
    Monster cur = it.next();
    System.out.println("Monstre: " + cur.name);
}
```

La boucle "pour chaque" (1/2)

- Sucre syntaxique introduit dans Java 5

```
for(Monster cur: col) {
  System.out.println("Monstre : " + cur.name) ;
}
```

$\leftrightarrow$

```
Iterator<Monster> it = col.iterator();
while(it.hasNext()) {
  Monster cur = it.next();
  System.out.println("Monstre: " + cur.name);
}
```

La boucle "pour chaque" (2/2)

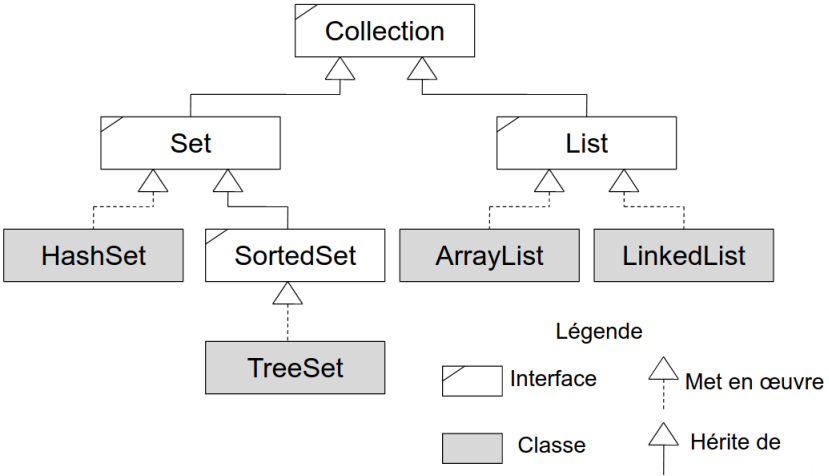
- On peut aussi utiliser une boucle « pour chaque » pour parcourir un tableau

- Bon à savoir, rappelez vous en !

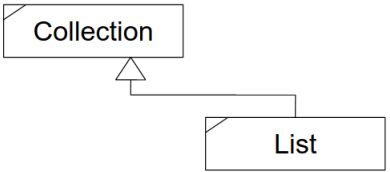
```
String[] tab = { "a", "b" };
for(String s: tab) {
  System.out.println(s);
}
```

Plan du cours

L'arbre d'héritage de `Collection`



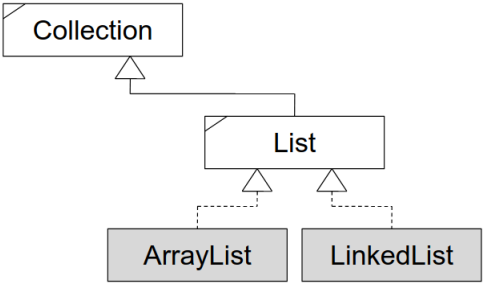
L'interface `List` (1/2)



- Propriétés :
  - Les éléments possèdent un indice → généralisation des tableaux
  - Duplication d'éléments autorisée

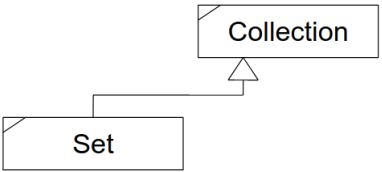
- Ajoute principalement quatre nouvelles méthodes
  - `void add(int i, E e)`
  - `void set(int i, E e)`
  - `E get(int i)`
  - `E remove(int i)`

L'interface `List` (2/2)



- Deux principales mises en œuvre
  - `ArrayList` : tableau extensible
  - `LinkedList` : liste chaînée

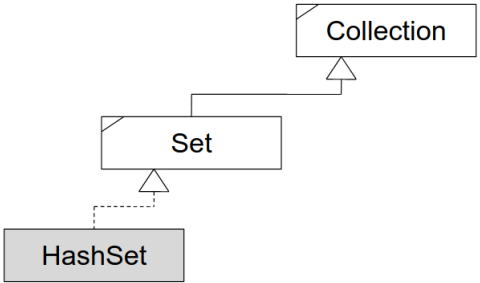
L'interface Set (1/2)



- Propriétés :
  - Les éléments ne possèdent pas d'indice
  - Duplication d'éléments interdite

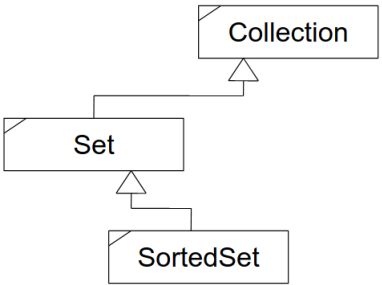
- Pas de nouvelle méthode ajoutée

L'interface `Set` (2/2)



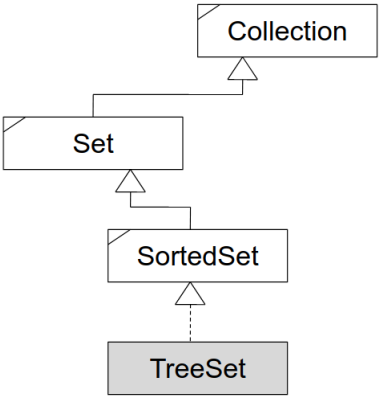
- `HashSet` = table de hachage dans laquelle `clé == valeur`
  - Les éléments **doivent** mettre en œuvre `equals` et `hashCode`
  - Les éléments sont *non triés*

L'interface `SortedSet` (1/2)



- Propriétés :
  - Les éléments sont triés (mettent en œuvre `Comparable`)
- Pas de nouvelle méthode ajoutée

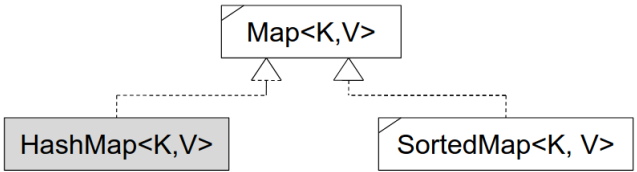
L'interface `SortedSet` (2/2)



- `TreeSet` = arbre binaire rouge-noir dans lequel clé == valeur

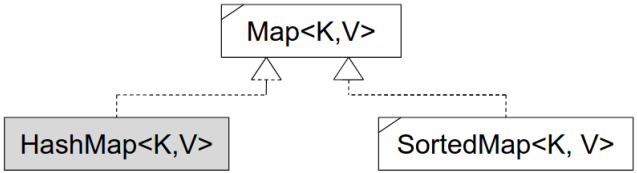
Plan du cours

Les tables d'association en Java (1/2)



- `Map` définit principalement quatre méthodes
  - `V put(K k, V v)` : associe k à v (renvoie ancienne valeur ou null)
  - `V get(K k)` : renvoie la valeur associée à k ou null
  - `boolean remove(K k)` : supprime l'association
  - `Collection<V> values()` : renvoie la collection des valeurs

Les tables d'association en Java (2/2)



- Deux principales mises en œuvre
  - La table de hachage (`HashMap`)
    - `K` doit mettre en œuvre `equals` et `hashCode`
    - Table d'association + éléments non triés
  - L'arbre binaire rouge-noir (`TreeMap` de type `SortedMap`)
    - `K` doit mettre en œuvre Comparable
    - Table d'association + éléments triés dans l'ordre des clés

Notions clés

- Collection pour stocker des collections d'éléments
  - Deux principales mises en œuvre
    - `ArrayList` : tableau extensible, ajout lent (extension du tableau), accès aléatoire rapide
    - `LinkedList` : liste chaînée, ajout rapide, accès aléatoire lent
  - Parcours avec `Iterator` ou boucle "pour chaque"
- Map pour associer clé et valeur
  - Deux principales mises en œuvre
    - `HashMap` : table de hachage, non triée
    - `TreeMap` : arbre binaire rouge-noir, trié suivant les clés