Ing39 – Bibliothèque Collections (3)

Virginia Aponte Pierre Courtieu

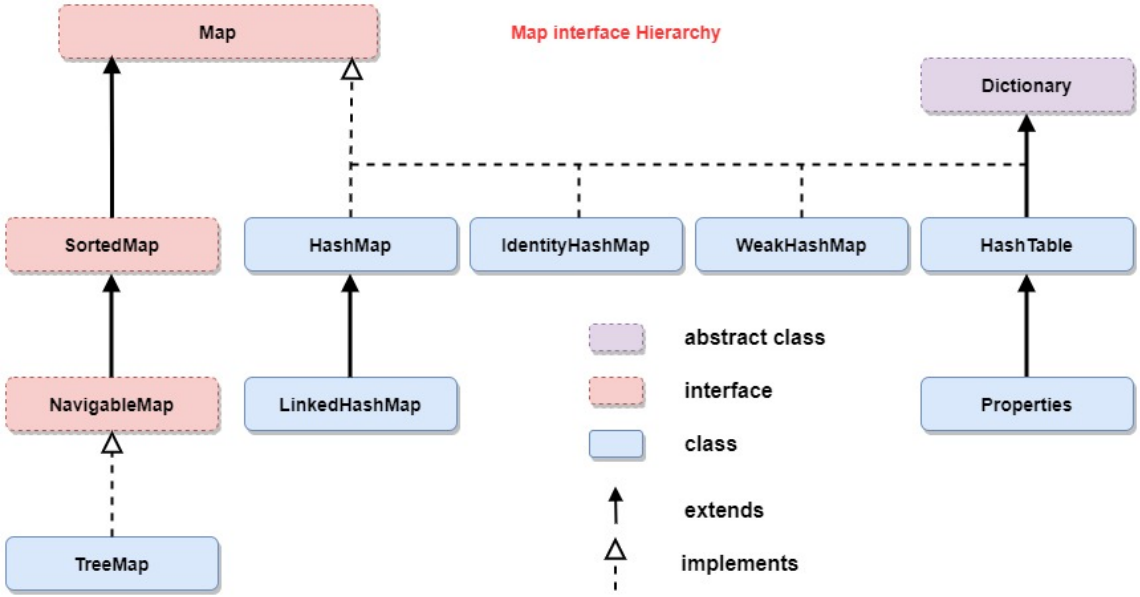
Dans ce cours

- ⋆ Hiérarchie `Map<K,V>` et ses vues collections

Attention :

- ⋆ il faut avoir bien compris le parcours de collections (for-each, itérateurs)
- ⋆ ainsi que la spécification de l'ordre pour le tri

Hiérarchie `Map<K,V>` (`https://vidvaan.com/java-collection-framework-overview/`)



Structure des donées « dictionnaire associatif »

Table permettant :

- ⋆ stocker des valeurs v (de type V),
- ⋆ organisées/récupérables via leur clé de recherche k (de type K).

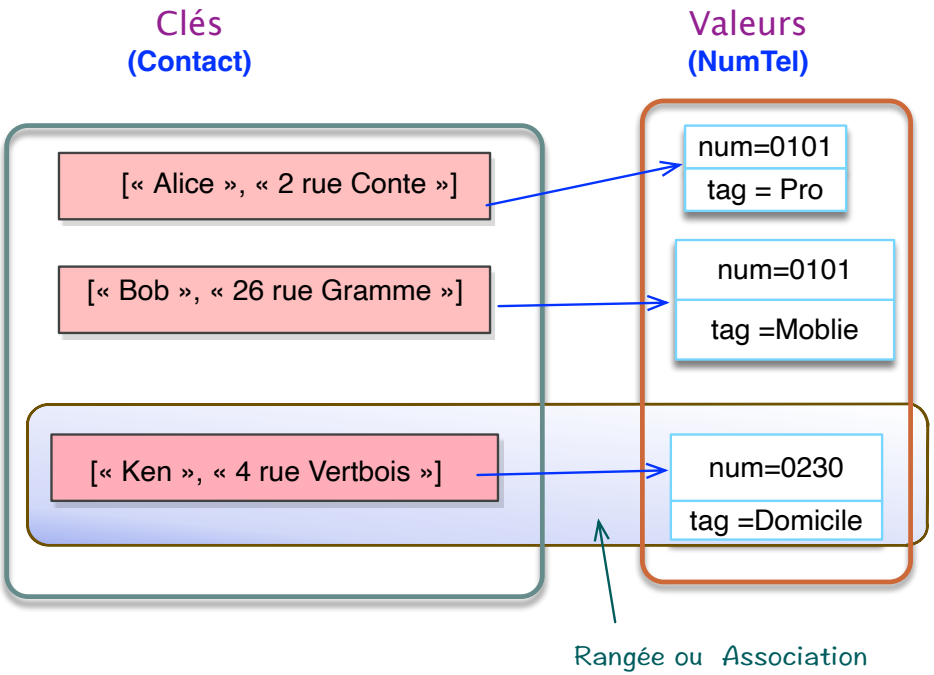
Fonctionne comme :

- ⋆ un ensemble « d'associations » $[k] \mapsto [v]$ entre clés k et valeurs v,
- ⋆ v est la « valeur » associée à la clé k dans la table.

1 *association* (k, v) entre clé k et valeur v $\Rightarrow$ *rangée* de la table.

Exemple : répertoire téléphonique

Ici, les objets `Contact` (nom, adresse) sont associés à leur numéro de téléphone `NumTel`.




```
public class Contact{…}
public classe NumTel{…}
```

Cette table contient 3 rangées ou associations

- ⋆ à chaque clé *k* correspond *une unique* valeur *v*.
- ⋆ clés peuvent être de n'importe quel type K,
- ⋆ les clés *doivent* former un ensemble (pas de clé en double).
- ⋆ pour récupérer la valeur associée à la clé *k* dans t :

  `t.get(k)` ⇒ *retourne la valeur v associé à k dans t*

- ⋆ `t.put(k,v)` ⇒ *modifie avec v la valeur associée à k dans t*

Opérations dictionnaire associatif `Map<K,V>`

`Map<K,V>` : dictionnaire associatif formé de couples (clé (`K`) $\mapsto$ valeur (`V`)).
1 couple (k,v) est dit *rangée* ou *entry*

Opérations :

- `V get(K k)` : valeur de la clé `k` (null si non trouvée)
- `void put(K k,V v)` : ajout/modification d'une rangée
- `boolean containsKey(K k)` : tester si clé k présente
- `boolean isEmpty()`, ...
- pas d'itération directe $\Rightarrow$ itérer sur 1 vue collection (voir plus loin)
- Implantations :
  - `HashMap<K,V>` `LinkedHashMap<K,V>` (tables de Hash)
  - `TreeMap<K,V>` (clés triés, arbres de recherche)

Vues « collection » d'un Map

Un Map n'est pas une collection ⇒ pas d'itération directe, mais des méthodes pour obtenir toutes ses clés, valeurs, ou associations.

⇒ Ces méthodes renvoient des collections que nous savons parcourir.

⇒ elles sont les « vues collection » d'un Map (permettent de modifier la Map).

Map : pas de for-each, pas d'itérateur

Si m est un Map, on peut obtenir (et parcourir) ses collections *internes* :

- l'ensemble des clés ⇒ `m.keySet()`
- la collection de toutes les valeurs ⇒ `m.values()`
- l'ensemble de toutes les paires clé-valeur ⇒ `m.entrySet()`
- et du coup, réaliser toute sorte de traitement sur ces collections.

Les vues Collection d'un `Map<k,V>`

Les `Map<K,V>` ne sont pas directement itérables, mais

- ⋆ on peut obtenir des *vues* `Collection` de leurs composantes,
  ⇒ qui sont elles, itérables

Trois sortes de vues :

- ⋆ vue ensemble des clés : `Set<K> keySet()`
- ⋆ vue collection des valeurs : `Collection<V> values()`
- ⋆ vue ensemble des rangées : `Set<Map.Entry<K,V>> entrySet()`
  - ⇒ 1 *rangée* ou *entry* a le type `Map.Entry<K,V>`
    - ⋆ c'est un « objet couple » avec 2 méthodes
      - ⋆ `K getKey()` : retourne composante clé du couple
      - ⋆ `V getValue()` : retourne composante valeur du couple

Les associations d'un tableau associatif

Rappel : un tableau associatif est formé d'*associations* entre clés et valeurs

Tableau Associatif = Ensemble d'associations

- ⋆ 1 association (k, v) = 1 paire (clé,valeur) de la table
- ⋆ Analogie : 1 association $(k, v) \approx$ 1 rangée de la table.
- ⋆ En Java, le type d'une rangée ou association est `Map.Entry<K,V>`

Tableau associatif ⇒ formé par l'ensemble
$(k_1, v_1), (k_2, v_2) \ldots$ de toutes les associations (rangées) de la table.

Le type `Map.Entry<K,V>`

Rappel : m de type `Map<K,V>` correspond à un ensemble d'associations ou paires (k_i, v_i) (ou rangées du tableau associatif).

- ⋆ en Java chaque association est appelée « *entry* »
- ⋆ une entry est objet qui contient la clé+valeur de cette rangée,
- ⋆ le type d'une entry est : `Map.Entry<K,V>`
- ⋆ 2 méthodes sur les *objets entry* :
  - ⋆ `getKey()` (clé dans cette entrée) ;
  - ⋆ `getValue()` (valeur dans cette entrée).
- ⋆ obtenir l' ensemble d'associations ⇒ `m.entrySet()`
- ⋆ le type de cet ensemble est : `Set<Map.Entry<K,V> >`

Exemple dictionnaire associatif


```
// Dictionnaire d'années de naissance
Map<String,Integer> nait = new HashMap<String,Integer>();

// ajout rangées
nait.put("Martin",1980) ;
nait.put("Dupont",2000) ;
nait.put("Laffite",1980) ;

// nombre de rangées
System.out.println(nait.size()); // 3

// test appartenance  "Martin"
System.out.println(nait.containsKey("Martin")) ;  //true

System.out.println(nait.get("Martin")) ; //  1980

Set<String> cles = nait.keySet(); // vue ensemble de clés

Collection<Integer> annees = nait.values(); // vue collection de valeurs

// afficher les années supérieures à 1980
for (Integer an : annees) {
     if (an > 1980) System.out.print(an+" ");
}
```

Suite de l'exemple


```
// Ensemble rangées
Set<Map.Entry<String,Integer>> rangees= nait.entrySet();

// affichons-les
for (Map.Entry<String,Integer> e : rangees) {
      System.out.println(e.getKey()+" nait en " +e.getValue());
}

// itérateur sur les rangées
Iterator<Map.Entry<String,Integer>> itNait =  rangees.iterator();

// enlever années plus grandes que 1980
while(itNait.hasNext()) {
      if (itNait.next().getValue()>1980) itNait.remove();
}

// taille dictionnaire après remove
System.out.println("Taille dictionnaire après remove: "+nait.size());

// affichages (attention: avant remove!)
Dupont nait en 2000
Laffite nait en 1980
Martin nait en 1980
Taille dictionnaire après remove: 2
```

Contraintes d'utilisation de « Map »

Type de clés : `equals`, `hashCode` et relation d'ordre

Pour bien fonctionner, le type des clés devront être équipés d'une méthode `equals` correctement définie.

De plus, si on utilise :

- `HashMap<K,V>`, on veillera à la bonne définition de `hashCode`
- `TreeMap<K,V>`, on veillera à définir « un ordre naturel » via Comparable ou à fournir un objet Comparator (via le constructeur de TreeMap)
- si les clés sont d'un type prédéfini $\Rightarrow$ rien à faire, car toutes ses méthodes et interfaces sont déjà implantes/redéfinies !

Rappel

Pour rappel, on retrouve ces mêmes contraintes sur les éléments d'un `Set`