Les collections (2)

DUT M3105 : Conception et programmation objet avancées

Serge Rosmorduc
serge.rosmorduc@lecnam.net
Conservatoire National des Arts et Métiers

2018-2019

Les Maps ; tableaux associatifs

- *Tableaux associatifs* ;
- gèrent des couples *clef/valeur* ;
- exemple : dictionnaire
  - la clef est le mot à définir (une String) ;
  - la valeur est une définition :

```
1  Map<String, Definition> dictionnaire
```

  - notez qu'on a deux types comme paramètres.
- implémentations :
  - `HashMap` (ce cours)
  - `TreeMap` (prochain cours)
- généralisation des tableaux « classiques », avec les différences suivantes :
  - dans un tableau, les clefs sont uniquement des entiers ;
  - dans un tableau, les clefs couvrent intégralement un intervale entier.

Les maps

Opérations fondamentales :

- `put(clef, valeur)` : associe valeur à clef dans la map. S'il y avait déjà une valeur pour cette clef, on *écrase* l'ancienne valeur ;
- `get(clef)` : renvoie la valeur associée à `clef`, ou `null` s'il n'y en a pas ;
- `containsKey(clef)` : permet de savoir si la map contient une entrée pour `clef` ;
- `clear()` efface le contenu de la map ;
- `isEmpty()` dit si la map est vide.

Exemple

```
// Compte le nbre d'occurrence
// des mots dans un texte
public static Map<String,Integer>
            compteMots(String [] mots) {
  HashMap<String,Integer> map= new HashMap<>();
  for (String m: mots) {
    int cpt= 0;
    if (map.containsKey(m))
      cpt= map.get(m);
    cpt++;
    map.put(m,cpt);
  }
  return m;
}
```

Autres méthodes utiles

- `getOrDefault(key, defaultValue)` : renvoie la valeur associée à `clef`, si elle existe, ou `defaultValue` sinon. Très pratique (essayez de l'utiliser pour l'exemple précédent).

Parcours des maps

pas d'itérateur directement sur les maps, mais trois collections disponibles :

- keySet() : *ensemble* des clefs ;
- values() : collection des valeurs (si une valeur apparaît plusieurs fois dans la map, elle apparaît aussi dans values()) ;
- entrySet() : retourne l'ensemble des couples clef/valeur de la map.

Chacun de ces ensembles peut être parcouru.

Parcours des maps

```
public static void
    afficheCompte(Map<String,Integer> compte) {
  Set<String> clefs= compte.keySet();
  for (String c: clefs) {
    System.out.println(c + " : " + compte.get(c));
   }
}
```

entrySet

- ensemble des couples clef/valeur ;
- pour une Map<C,V>, le type de ses éléments est Map.Entry<C,V> ;
- et le type de l'entrySet est : `Set<Map.Entry<String, Integer>`
- avantage pour le parcours : on récupère directement la clef et la valeur (avec les méthodes `getKey()` et `getValue()`.

```
public static void
    afficheCompte(Map<String,Integer> compte) {
  for (Map.Entry<String,Integer> e: compte.entrySet()) {
    System.out.println(e.getKey() + " : " + e.getValue());
  }
}
```

Notion d'index

Dans une base de données

Personne
id
nom
prenom
ville

- *en plus de la table*, la BD crée un index $id \rightarrow \text{personne}$
- on peut créer d'autres index, par exemple sur `ville` ;
- un index sur ville associerait potentiellement **plusieurs** personnes à une ville.
- les index sont **redondants** : ils n'ajoutent pas d'information, mais accélèrent l'accès à celle-ci.

En mémoire

- Les `Map` java permettent d'implémenter des index **en mémoire**.
- typiquement, on prend alors comme clef un des champs de la valeur (par exemple valeur = Personne, clef = id de la personne).

Les map comme index, exemple

```
public class Repertoire {
  private Map<Integer, Personne> personnes=
                                        new HashMap<>();
  public void ajouterPersonne(Personne p) {
    personnes.put(p.getId(), p);
  }
  public Personne getPersonneParId(int id) {
    return personnes.get(id);
  }
}
```

- accélère l'accès à une personne connaissant son id ;
- garantit très simplement qu'il n'y a pas plusieurs personnes avec le même id dans le répertoire ;
- (problème un peu fin : et si on a une méthode `setId()` sur `Personne` ?

Index multivalué

Ici, la valeur est elle-même une collection. Pour une ville, on enregistre l'ensemble des personnes qui habitent cette ville.

```
public class Repertoire {
  private Map<String, Set<Personne>> villeVersPersonnes=
                                          new HashMap<>();
  public void ajouterPersonne(Personne p) {
    personnes.put(p.getId(), p); // map des ids
    if (villeVersPersonnes.containsKey(p.getVille())) {
      Set<Personne> dansVille=
            villeVersPersonnes.get(p.getVille());
      dansVille.add(p);
    } else {
      Set<Personne> dansVille= new HashSet<>();
      dansVille.add(p);
      villeVersPersonnes.put(p.getVille(), dansVille);
    }
  }
}
```

Problèmes liés aux index

- rajouter un index multivalué complexifie la classe ;
- ne le faire que si c'est necessaire (problème de perfomance) ;
- ... et si on a une méthode `setVille()` sur Personne ? ? ? En introduisant des redondances, on rend le logiciel plus difficile à modifier.

Modélisation avec les maps

Une bibliothèque comporte des exemplaires de livres (parfois plusieurs pour le même livre) ; et est ouverte à des emprunteurs. Les exemplaires ont un numéro d'inventaire, qui est unique ; même si un livre est volé ou détruit, son numéro ne sera jamais redonné ; les ouvrages sont identifiés par leur ISBN, qui est une chaîne de caractères, et les emprunteurs par un numéro d'identification. On désire pouvoir enregistrer de nouveaux livres, de nouveaux exemplaires et de nouveaux emprunteurs. On désire aussi pouvoir enregistrer un emprunt, ainsi que le retour d'un livre. On veut pouvoir savoir si un livre est emprunté, et connaître la liste des ouvrages en retard (plus de deux semaines d'emprunt).