UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence 1 INFO
INFO0201

Cours à distance

Les collections / La classe `Vector`

1 Les collections

Les tableaux ne peuvent pas répondre à tous les besoins de stockage d'un ensemble d'objets et surtout ils manquent de fonctionnalités. Les collections sont des objets qui permettent de gérer des ensembles d'objets. Ces ensembles de données peuvent être définis avec plusieurs caractéristiques : la possibilité de gérer des doublons, de gérer un ordre de tri, de gérer des références *null*, etc.

Une collection est un regroupement d'objets qui sont désignés sous le nom d'éléments. Plusieurs classes qui gèrent une collection implémentent une interface qui hérite de l'interface `Collection`. Cette interface est une des deux racines de l'arborescence des collections. Elle représente un minimum commun pour les objets qui gèrent des collections : ajout d'éléments, suppression d'éléments, vérification de la présence d'un objet dans la collection, parcours de la collection et quelques opérations diverses sur la totalité de la collection.

L'API Collections ne propose pas d'implémentation directe de cette interface : elle propose des implémentations pour des interfaces filles qui définissent les fonctionnalités des grandes familles de collections. Elle propose quatre grandes familles de collections, chacune définie par une interface de base :

- `List` : collection d'éléments ordonnés qui accepte les doublons
- `Set` : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons
- `Map` : collection sous la forme d'une association de paires clé/valeur

Chaque implémentation de ces interfaces devra fournir au minimum les fonctionnalités de l'interface en question et donc permettra la gestion de l'ensemble d'éléments en question.

Classe parametrée

Chaque implémentation est une classe paramétrée, dite "générique", qui peut être utilisée pour des objets de différents types mais ce type doit être spécifié à la création. Le type paramétré est indiqué entre `< >` derrière le nom de la classe. Il est par exemple possible de déclarer une liste de chaînes de caractères `List<String>` ou bien de personne `List<Personne>` (avec la classe `Personne` développée dans les exemples de CM).

Avec l'héritage et le typage dynamique, nous avons vu que nous pouvons affecter à une variable (ou à un paramètre ou un à attribut) une référence d'un objet du même type ou d'un type qui en hérite. Il est donc possible dans notre `List<Personne>` de mettre des `Etudiant` par exemple.

Nous allons nous intéresser plus particulièrement à la classe `Vector` qui implémente l'interface `List` et qui permet donc de gérer une collection d'éléments ordonnés qui accepte les doublons.

2 La classe `Vector`

La classe `Vector` permet de créer une collection d'objets qui fonctionne de la même manière qu'un tableau, à l'exception que sa capacité peut varier en fonction des besoins. Une allocation de mémoire dynamique est utilisée par les objets `Vector`, il est donc possible d'ajouter, de supprimer aisément leurs éléments, puisque leur taille varie dynamiquement.

Un `Vector` peut stocker un nombre quelconque d'objets de type `Object`, ou de n'importe quel type d'objets (String, Integer, URL, Date, etc.) issus de l'API ou créés par vous-même. Par contre, un `Vector` n'accepte pas les valeurs de types primitifs. En faite, un `Vector` ne stocke que les références vers des objets, toutes les références sont obligatoirement du même type au sein du même `Vector` mais peuvent pointer vers des objets du type de la référence ou d'une classe dérivée (comme le permet le typage dynamique).

Un `Vector` est défini par sa capacité (le nombre d'éléments qu'il peut contenir) et sa taille (le nombre d'éléments qu'il contient réellement).

L'ensemble de la classe `Vector` est documentée dans l'API, n'hésitez pas à la parcourir lorsque vous recherchez une fonctionnalité pour cette classe.

2.1 Les constructeurs

Un vecteur comporte quatre constructeurs permettant d'initialiser les instances de cette classe.

- Le premier permet de créer un `Vector<E>` d'élément `E` initialement vide auquel il sera possible d'ajouter des éléments via une méthode spécifique.
  exemple : `Vector<String> v = new Vector<String>();`
- Le second permet d'initialiser un `Vector<E>` à une certaine capacité fournie en argument. Dans ce cas, la valeur contenu par chacun des éléments est *null*. Cependant, cette initialisation peut poser des problèmes de gestion d'espace mémoire, car un `Vector` initialisé à n éléments voit sa capacité doublée lorsqu'un dépassement de sa capacité initiale intervient.
  exemple : `Vector<String> v = new Vector<String>(50);`
- Le troisième permet d'initialiser le `Vector<E>` à une certaine capacité tout en veillant également à définir un pas d'incrémentation de sa capacité suite à un dépassement, palliant ainsi à la contrainte du constructeur précédent.
  exemple : `Vector<String> v = new Vector<String>(100, 10);`
- Enfin, le quatrième permet de construire un `Vector<E>` à partir d'une `Collection` passée en argument.
  exemple : `Vector<String> v = new Vector<String>(collection);`
  // avec `collection` une variable de type `Collection` ou d'une de ses classes dérivées

2.2 Quelques-unes de ses méthodes

Deux méthodes permettent respectivement de récupérer la capacité et la taille d'un `Vector`. Evidemment, ces deux dernières fournissent des informations différentes. La capacité correspond au nombre d'éléments qu'il est possible de stocker (nombre qui peut évoluer), alors que la taille reflète le nombre d'éléments effectif contenu (nombre qui peut évoluer aussi). Exemples d'utilisation :

- `int capacite = v.capacity();`
- `int taille = v.size();`

L'ajout d'objets dans un `Vector` s'accompli par l'intermédiaire de plusieurs méthodes dont la plupart sont préfixées par `add`. En voici 2 exemples :

— La méthode `void add(int index, Object element)` insère un élément à la position spécifiée.
exemple : `v.add(3, "Une chaîne");`

— La méthode `boolean add(Object element)` ajoute un élément à la fin du `Vector`.
exemple : `boolean reussi = v.add("Une chaîne");`

La suppression est prise en charge par plusieurs méthodes commençant en général par `remove`. La suppression d'un ou plusieurs éléments entraîne une réorganisation du `Vector`, c'est-à-dire que l'emplacement d'un élément effacé et immédiatement remplacé par l'élément suivant et cela en cascade jusqu'au dernier élément du `Vector`. En voici 3 exemples :

— La méthode `Object remove(int index)` supprime l'objet à l'index spécifié et le retourne.
exemple : `Object element = v.remove(5);`

— La méthode `boolean remove(Object obj)` supprime la première occurrence de l'objet trouvée dans le `Vector`. Retourne vrai ou faux selon si l'occurence a été trouvée et supprimée ou non.
exemple : `boolean reussi = v.remove("mardi");`

— La méthode `void removeAllElements()` supprime tous les éléments et remet à zéro la taille du `Vector`.
exemple : `v.removeAllElements();`

La consultation des éléments présents dans un `Vector` s'effectue par l'intermédiaire de diverses méthodes permettant de récupérer un élément précis ou plusieurs. En voici 3 exemples :

— La méthode `Object elementAt(int index)` retourne l'élément trouvé à l'index spécifié.
exemple : `String obj = (String)v.elementAt(5);`

— Les méthodes `Object firstElement()` et `Object lastElement()` retournent respectivement le premier et le dernier élément du `Vector`.
exemple : `Object prem = v.firstElement();` *//Retourne l'élément à l'index 0*
exemple : `Object der = v.lastElement();` *//Retourne l'élément à l'index v.size()-1*

Les moyens de contrôler la présence d'un ou plusieurs éléments sont assurés par plusieurs méthodes prenant comme argument l'objet recherché et parfois un index de démarrage de la recherche. En voici quelques exemples :

— La méthode `boolean contains(Object obj)` vérifie si l'objet spécifié est contenu dans le `Vector`.
exemple : `boolean reussi = v.contains("mercredi");`

— Les méthodes `int indexOf(Object obj)` et `int lastIndexOf(Object obj)` recherchent respectivement la première et la dernière occurrence d'un objet donné en testant l'égalité entre les objets à l'aide de la méthode `equals()`.
exemple : `int position = v.indexOf("Samedi");`
exemple : `int position = v.lastIndexof("samedi");`

— Les méthodes `int indexOf(Object obj, int index)` et `int lastIndexOf(Object obj, int index)` recherchent respectivement la première et la dernière occurrence d'un objet donné en démarrant à une certaine position et en s'appuyant sur la méthode `equals()`.
exemple : `int position = v.indexOf("Samedi",3);`
exemple : `int position = v.lastIndexof("samedi", 6);`

Il existe bien d'autres méthodes, n'hésitez pas à chercher dans la documentation de l'API en ligne.

Exemple d'utilisation de la classe `Vector` :

```
import java.util.Vector;
public class TestVector {
    public static void main(String[] args) {
        Vector vecteur = new Vector<String>();
        vecteur.add("lundi");
        vecteur.add("mardi");
        vecteur.add("mercredi");
        String vendredi = "vendredi";
        vecteur.add(vendredi);
        vecteur.add(3, "jeudi");
        System.out.println("vecteur␣contient-il␣\"mercredi\"␣?␣" +
           vecteur.contains("mercredi"));
        System.out.println("A␣quel␣position␣en␣partant␣du␣debut␣?␣" +
           vecteur.indexOf("mercredi"));
        System.out.println("Quel␣est␣l'element␣en␣deuxieme␣position␣?␣"
           + vecteur.elementAt(2));
        Vector v2 = new Vector<String>(vecteur);
        v2.remove(1);
        v2.removeAllElements();
        System.out.println("La␣taille␣de␣vecteur␣:␣" + vecteur.size() +
           ",␣et␣sa␣capacite␣:␣" + vecteur.capacity());
        System.out.println("La␣taille␣de␣v2␣:␣" + v2.size() + ",␣et␣sa␣
           capacite␣:␣" + v2.capacity());
    }
}
```

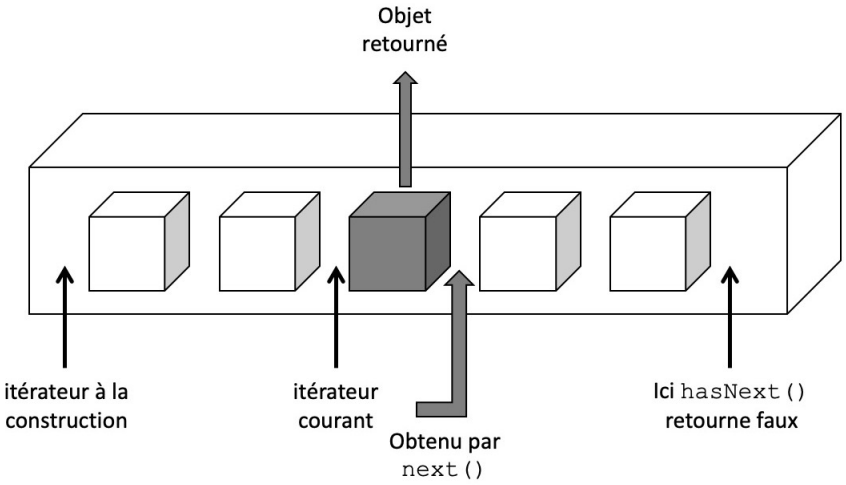
N'hésitez pas à tester cet exemple en particulier pour regarder la capacité appliquée par défaut et la façon dont elle est augmentée lorsque le vecteur atteint sa capacité maximale.

2.3 Parcours d'un objet `Vector`

Le parcours des éléments d'un objet `Vector` peuvent se faire en utilisant une boucle, ou en appliquant un itérateur sur les éléments de la liste. En effet toutes les classes conteneurs, qui implémentent les méthodes de l'interface `Collection`, doivent implémenter la méthode `Iterator iterator()` qui retourne un itérateur sur l'ensembles des éléments. Un itérateur est un objet qui implémente l'interface `Iterator`, et qui a pour but de permettre le parcours des éléments d'un conteneur, sans avoir besoin de savoir de quelle nature est ce conteneur.

L'interface `Iterator` contient les méthodes suivantes :

- La méthode `boolean hasNext()` retourne vrai si l'itérateur n'est pas arrivé à la fin de l'ensemble et faux sinon.
- La méthode `E next()` permet d'avancer l'itérateur, et retourne l'élément passé (lève une exception `NoSuchElementException` si l'itérateur n'a pas d'objet suivant.
- Les méthodes `void remove()` supprime l'objet qui vient d'être obtenu par `next()` (cette méthode est facultative).



De plus il existe une boucle `for` dédiée au collection dont voici la syntaxe : `for(Type t : collection)`. Cette boucle itère les objets de la collection dans la référence `t`, le type de collection et type de `t` doivent être compatibles. Attention elle n'est pas directement utilisable pour la classe `Vector`, il faudra référencer le `Vector` dans une `List` (cf exemple ci-dessous).

Exemple pour parcourir tous les éléments du `Vector vecteur` de l'exemple précédent :

```
// penser a importer les packages java.util.Iterator et java.util.List
    System.out.println("Contenu␣de␣vecteur␣avec␣une␣boucle␣for␣classique
       ␣:");
    for(int i=0;i<vecteur.size();i++)
        System.out.println("␣-␣" + vecteur.elementAt(i));

    System.out.println("Contenu␣de␣vecteur␣avec␣une␣boucle␣for␣et␣un␣
       iterateur␣:");
    Iterator<String> it1 = vecteur.iterator();
    for(it1 = vecteur.iterator() ; it1.hasNext(); )
        System.out.println("␣-␣" + it1.next());

    System.out.println("Contenu␣de␣vecteur␣avec␣une␣boucle␣while␣et␣un␣
       iterateur␣:");
    Iterator<String> it2 = vecteur.iterator();
    while(it2.hasNext())
        System.out.println("␣-␣" + it2.next());

    System.out.println("Contenu␣de␣vecteur␣avec␣une␣boucle␣for␣dediee␣au
       ␣collection␣:");
    List<String> vec = new Vector<String>(vecteur);
    for(String s : vec)
        System.out.println("␣-␣" + s);
```