Informatique 1

9. Structures de données dynamiques

Dr Pierre-André Mudry
pierre-andre.mudry@hevs.ch

Rev 1.25

Objectifs

Découvrir de nouvelles structures de données

- Ensembles et collections
- Vecteurs
- Listes

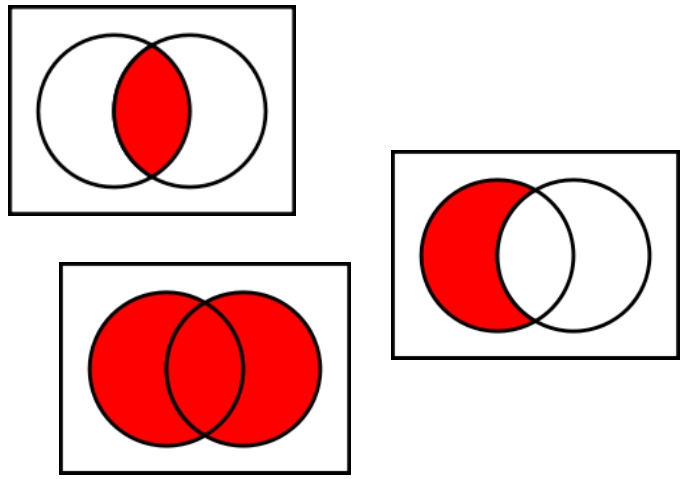
Penser les données

9.1 ENSEMBLES ET COLLECTIONS

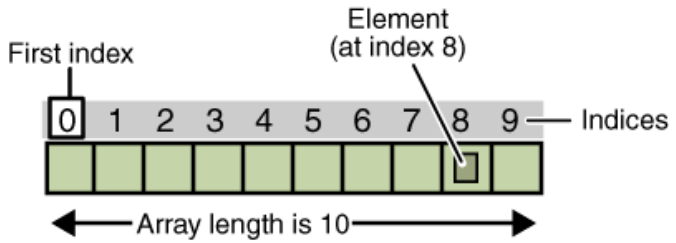
Les ensembles

- Ensemble ≈ groupement d'élément
  1. Homogène / hétérogène
  2. Ordré oui / non
  3. Occurrences multiples oui / non

- Théorie des ensembles
  - Ensemble vide
  - Réunion
  - Intersection
  - Différence



Ensembles (2) – Exemple du tableau



- Ensemble : un tableau
  - Homogène ou hétérogène
  - Ordré ou non
  - Occurrences multiples ou non

Java et les collections

- Les **ensembles** de données importants en programmation:
  - *Java* propose des classes pour les gérer
  - Nommés **collections**

Qu'est-ce qu'une collection ?

Une structure **dynamique** de données **modifiables** que l'on peut **parcourir** facilement

Collection

Les collections en Java (2)

- Plusieurs types complexes basés sur collections: *vecteurs*, *listes*, *stack*, …
- Dynamique VS statique
  - → ensemble *statique* = comme tableau, taille fixe
  - → ensemble *dynamique* = taille variable

Collections

- Parmi l'ensemble des collections, nous allons voir aujourd'hui :
  - Tableau dynamique : **`Vector`**
    - Comme tableau
    - Permet ajouter et enlever éléments
    - Accès direct éléments
  - Liste chaînée : **`LinkedList`**
    - Accès séquentiel aux éléments
    - Permet ajouter et enlever éléments n'importe où

Méthodes standard des collections

Quelques méthodes communes à toutes les collections

- `add(Object arg)`
- `remove(Object arg)`
- `size()`
- `isEmpty()`
- `clear()`
- `contains(Object arg)`

`java.util.Collections`

Comme les tableaux mais en mieux

9.2 VECTEURS

File Edit Source Refactor Navigate Search Project Run Window Help

DemoLists.java DictDemo.java VectorError.java VectorLive.java

```java
package course.collections;

import java.util.Vector;


public class VectorLive {

    public static void main(String[] args) {

    }


}
```

Problems Javadoc Declaration Console History Coverage

No consoles to display at this time.

Vecteurs

- Ressemble aux tableaux (ceux avec [ ])
- Syntaxe déclaration objet :

```
Vector myVector = new Vector();
```

- Accès un élément :

```
myVector.get(int position)
```

- Ajout d'élément

```
myVector.add(Object o)
```

- Stockage élément (position existante)

```
myVector.set(int position, Object content)
```

Vecteurs ≠ tableaux

Différences

1. Nombre d'éléments *dynamique*
2. Un peu plus lent mais plus flexible.
3. Types des éléments à l'intérieur du vecteur *peuvent être différents* ! → Attention !

```
Vector myVector = new Vector();

myVector.add("Hello");
myVector.add(new Auto("Golf", 160));

String r1 = (String) myVector.get(0);
Auto r2 = (Auto) myVector.get(0);
```

→ **ERREUR dynamique**

Vecteurs génériques

- Vecteurs génériques :
  - Si le type des éléments d'un vecteur est fixe, on peut utiliser la *généricité* pour contraindre les vecteurs → plus de *cast* !
  - Vecteur **forcé** à contenir un type particulier
- Syntaxe :

```
Vector<type> identifier = new Vector<type>();
```

Vecteurs génériques (2) : exemples

```
// Auto vector
Vector<Auto> autos = new Vector<Auto>();
autos.add(new Auto("VW", 160));
Auto foo = autos.get(0); // NO cast

// String vector
Vector<String> strings= new Vector<String>();

// Initialize vector
for(int i = 0; i < 10; i++)
    strings.add(new String());
```

Vecteurs génériques (3)

- *Avantages :*
  - Plus sûr à l'usage
    - Compilateur connaît type contenu (*strong type checking*)
  - Pas besoin de *caster* les objets
  - Plus rapide
- *Désavantages :*
  - Contenu homogène

Itération avancée avec les collections

```
for (Auto a : autos) {
    System.out.println(a);
}
```

La reine des structures de données

9.3 LISTES CHAÎNÉES

Introduction aux listes chaînées

- Exemple réel : un itinéraire d'avion
- Dynamique :
- En Java, liste `java.util.LinkedList`
- On va refaire nous-mêmes dans le labo

- Grand nombre d'algorithmes : tri, recherche…
  - La plus connue des structures dynamiques

Listes (2)

- Chaque donnée stockée dans un nœud
- Liens = flèches qui permettent liaison unidirectionnelle d'un nœud à l'autre
- Nœuds chaînés entre eux → *liste chaînée*
- *Terminologie* :
  - Premier nœud : la tête de liste (**`head`**)
  - Dernier nœud : la queue de liste (**`tail`**)

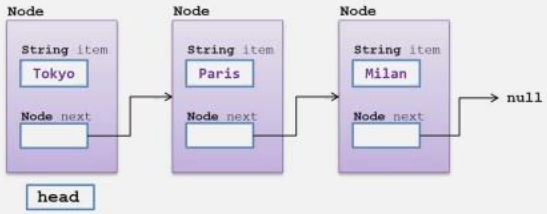
Implémentation des listes

- Nœuds dans une classe *`Node`*
- Donnée stockée dans une variable d'instance
- Liens comme référence à un *`Node`*

→ *Implémentation complète au labo !*

```java
package course.collections;

public class LinkedListDemo {

    public static void main(String[] args) {

    }

}
```

Une liste



Conclusion

- Collection = concept puissant mais un peu plus complexe
- Java propose grand nombre d'outils très complets mais parfois complexes
  - ➔ Nous allons faire les nôtres…
- Aller plus loin ? Page anglaise de Wikipedia sur les listes très bien faite :
- http://en.wikipedia.org/wiki/Linked_list