Informatique 1

9. Structures de données dynamiques

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

Rev 1.20

Objectifs

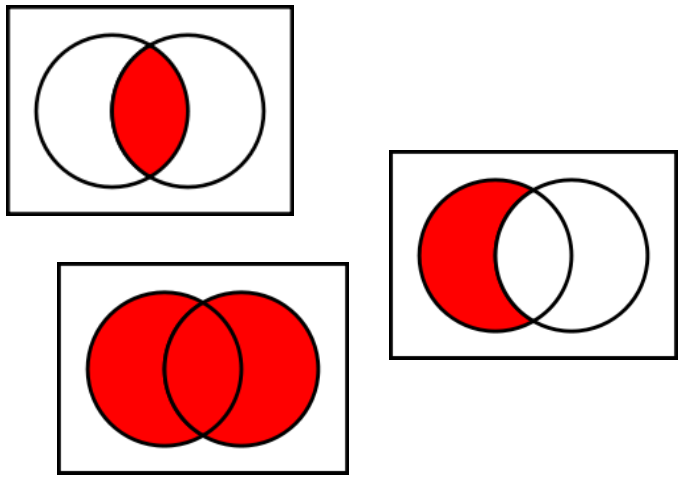
Découvrir de nouvelles structures de données

- Ensembles et collections
- Vecteurs
- Listes

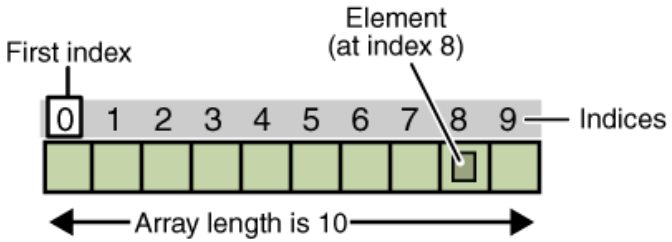
Les ensembles

- 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** sont importants en programmation:
  - *Java* propose des classes pour les gérer
  - Nommés **collections**

Qu'est-ce qu'une collection ?

Une structure **dynamique** de données **modifiables** (ajout et retrait) 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
- Parcourir ?

Méthodes standard des collections

Quelques méthodes sur les collections

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

`java.util.Collections`

Vector live coding

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

Objectifs

Comprendre les structures de données complexes

- Ensembles et collections
- Vecteurs
- Listes

Vecteurs

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

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

- Accès un élément :

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

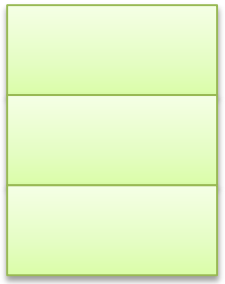
- Ajout d'élément

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

- Stockage élément (position existante)

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

Vecteurs ≠ tableaux



Différences

1. Nombre d'éléments *dynamique*
2. 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

Des objets comme contenu

Contenu standard des vecteurs : des **objets**

```
Vector myVector = new Vector();
myVector.add(new Auto("VW", 120));
myVector.add("Hello");
myVector.add(3);
```

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

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

Objectifs

Comprendre les structures de données complexes

- Ensembles et collections
- Vecteurs
- *Listes chaînées*

Introduction aux listes chaînées

- *Exemple réel* : un itinéraire d'avion

- Dynamique :

- En *Java*, liste `java.util.LinkedList`
- 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`**)

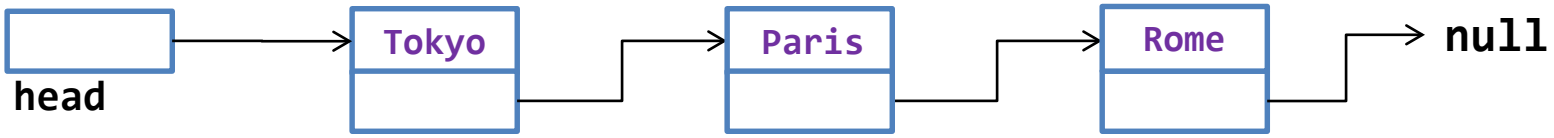
Implémentation des listes

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

→ *Implémentation complète au labo !*

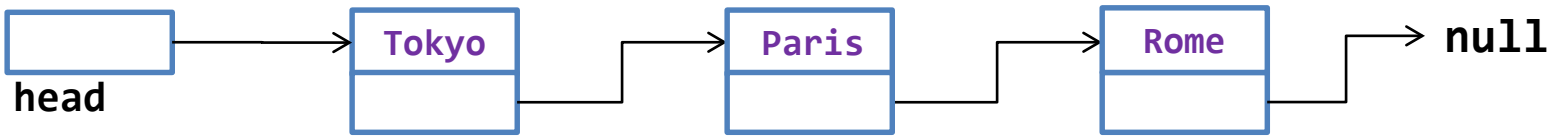
Exemples d'opérations sur les listes

- Enlever un élément



Exemples d'opérations sur les listes (2)

- Ajouter un élément



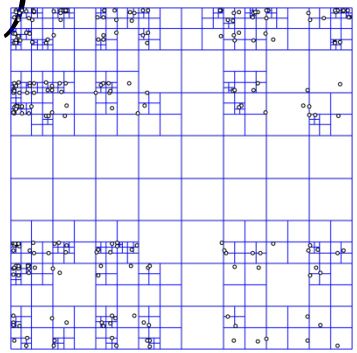
Remarque

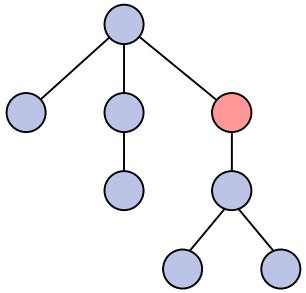
- Tableau ou `Vector`, l'accès est aléatoire (≠ séquentiel), càd on peut accéder chaque élément dans n'importe quel ordre.

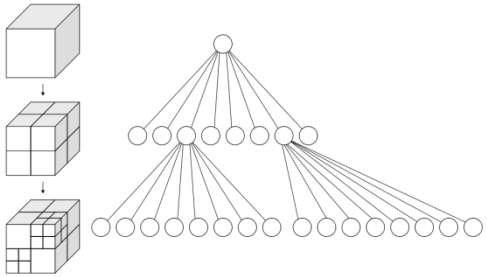
- Dans une liste, on accède aux éléments en parcourant celle-ci, les uns après les autres.

Les autres collections

- Il existe un grand nombre d'autres structures de données complexes :
  - Stack (FIFO, LIFO…)
  - Queues
  - Arbres
  - *Hash map*
  - …







Conclusion

- **Collection** = concept puissant mais un peu plus lent et plus complexe
- *Java* propose grand nombre d'outils très complets mais 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