Programmation Objet. Cours 6

Marie-Pierre Béal

DUT 1

Collections. Ensembles. Dictionnaires.

Vue d'ensemble

Deux paquetages

- `java.util` pour les ensembles, collections, itérateurs.
- `java.util.concurrent` structures supplémentaires pour la programmation concurrente.

Deux interfaces

- `Collection<E>` pour les ensembles d'objets, avec ou sans répétition.
- `Map<K,V>` pour les dictionnaires (ou tables), c'est-à-dire des ensembles de couples (clé, valeur), où la clé et la valeur sont de types paramétrés respectifs `<K,V>`. Chaque clé existe en un seul exemplaire mais plusieurs clés distinctes peuvent être associées à une même valeur.

Utilitaires

De plus des classes implémentant les interfaces `Collection` et `Map`, on a trois classes d'utilitaires

- `Collections` avec des algorithmes de tri etc,
- `Arrays` algorithmes spécialisés pour les tableaux.
- `Objects` algorithmes spécialisés pour les objets.

Principales méthodes des collections

Les opérations principales sur une collection

- `add` pour ajouter un élément.
- `remove` pour enlever un élément,
- `contains` test d'appartenance,
- `size` pour obtenir le nombre d'éléments,
- `isEmpty` pour tester si l'ensemble est vide.

Le type des éléments est un type paramétré `E`. L'utilisateur doit le remplace par un nome de classe.

Interface `List`

- `List<E>` spécifie les *séquences*, avec les méthodes
  - `int indexOf(Object o)` retourne la position de `o`.
  - `E get(int index)` retourne l'objet à la position `index`.
  - `E set(int index, E element)` remplace l'élément en position `index`, et retourne l'élément qui y était précédemment.

Interface Set

On a deux interfaces

- `Set<E>` spécifie les ensembles sans duplication.
- `SortedSet<E>` sous-interface de `Set` pour les ensembles ordonnés, qui a des méthodes spécifiques
  - `E first()` retourne le premier objet.
  - `E last()` retourne le dernier objet.
  - `SortedSet<E> subset(E fromElement, E toElement)` retourne une référence vers le sous-ensemble des objets $\geq$ `fromElement` et $<$ `toElement`.

Operations ensemblistes sur les collections

Ces méthodes sont des méthodes de l'interface `Collection<E>`.

- `boolean containsAll(Collection<?> c)` pour tester l'inclusion.
- `boolean addAll(Collection<? extends E> c)` pour la réunion.
- `boolean removeAll(Collection<?> c)` pour la différence.
- `boolean retainAll(Collection<?> c)` pour l'intersection.

Les trois dernières méthodes retournent `true` si elles ont modifié la collection.

Implémentation d'une collection

Une sous-interface

- Une sous-interface `Deque<E>` pour les piles et les files d'attente.

Des classes

- `ArrayList<E>` (recommandée, par tableaux), et `LinkedList<E>` (par listes doublement chaînées) implémentent `List<E>`.
- `Vector<E>` est une vieille classe (JDK 1.0) "relookée" qui implémente aussi `List<E>`. Elle est `Deprecated`. Il faut utiliser les `Deque`.
- `HashSet<E>` (recommandée) implémente `Set<E>`.
- `TreeSet<E>` implémente `SortedSet<E>`.

Le choix de l'implémentation résulte de l'efficacité recherchée : par exemple, l'accès indicé est en temps constant pour les `ArrayList<E>`, l'insertion entre deux éléments est en temps constant pour les `LinkedList<E>`.

Discipline d'abstraction

- les attributs, paramètres, variables locales sont déclarés avec, comme type, une interface (`List<Integer>`, `Set<Double>`),
- les classes d'implémentation ne sont utilisées que par leurs constructeurs.

```
List<Integer> l = new ArrayList<Integer>();
Set<Integer> s = new HashSet<Integer>();
```

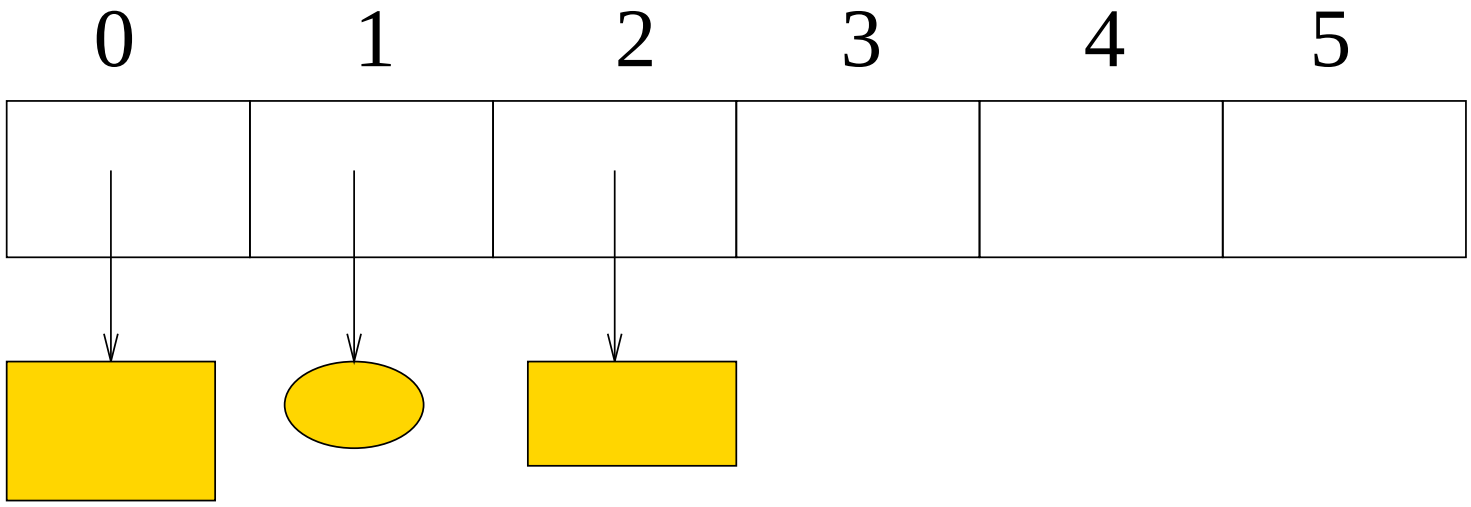
Exemple d'ArrayList

On désire créer un tableau de références sur des objets de type Shape qui peuvent être Rectangle ou Ellipse (déjà vus).

```
class ShapeTest{
  public static void main(String[] args){
    Shape r1 = new Rectangle(6,10);
    Shape r2 = new Rectangle(5,10);
    Shape e = new Ellipse(3,5);
    final List<Shape> liste = new ArrayList<Shape>();
    liste.add(r1);
    liste.add(r2);
    liste.add(1,e); // on a r1, e, r2
    for (Shape s:liste)
      System.out.println(s.toStringArea());
  }
}
```

Exemple d'ArrayList

```
$java -classpath ../shape:. ShapeTest
aire = 60.0
aire = 11.780972450961723
aire = 50.0
```



Exemple

Programme qui détecte une répétition dans les chaînes de caractères entrées sur la ligne de commande.

```
import java.util.Set;
import java.util.HashSet;
class SetTest {
  public static void main(String[] args) {
    final Set<String> s = new HashSet<String>();
    for (String w:args)
      if (!s.add(w))
        System.out.println("Deja vu : " + w);
    System.out.println(s.size() + " distincts : " + s);
  }
}
```

Test

```
$ java SetTest a b c a b d
Deja vu : a
Deja vu : b
4 distincts : [d, a, c, b]
```

Les dictionnaires ou tables ou `Map`

L'interface `Map<K,V>` spécifie les tables, des ensembles de couples (clé, valeur). Les clés ne peuvent être dupliquées, au plus une valeur est associée à une clé.

- `V put(K key, V value)` insère l'association `(key, value)` dans la table et retourne la valeur précédemment associée à la clé ou bien `null`.
- `boolean containsKey(Object key)` retourne vrai s'il y a une valeur associée à cette clé.
- `V get(Object key)` retourne la valeur associée à la clé dans la table, ou `null` si `null` était associé ou si `key` n'est pas une clé de la table.
- `V remove(Object key)` supprime l'association de clé `key`. Retourne la valeur précédemment associée. Retourne `null` si `null` était associé ou si `key` n'est pas une clé de la table.

La sous-interface `SortedMap<K,V>` spécifie les tables dont l'ensemble des clés est ordonné.

Implémentation des `Map`

Pour les tables avec clés non ordonnées

- `HashMap<K,V>` (recommandée), implémente `Map<K,V>`.
- `Hashtable<K,V>` est une vieille classe (JDK 1.0) `Deprecated`.

Pour les tables avec clés non ordonnées

- `TreeMap<K,V>` implémente `SortedMap<K,V>`.

La classe `TreeMap<K,V>` implémente les opérations avec des arbres rouge-noir.

La classe `TreeMap`

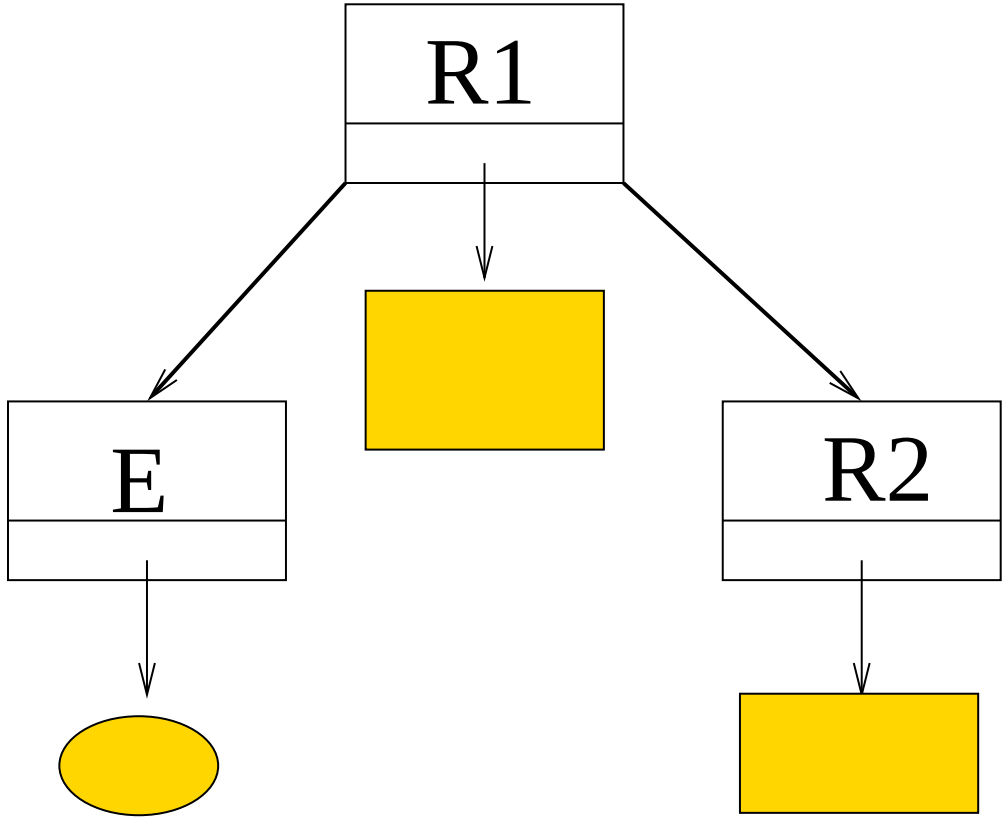
Une `TreeMap<K,V>` stocke ses clés (ordonnées) de telle sorte que les opérations suivantes s'exécutent en temps $O(\log(n))$:

- `boolean containsKey(Object key)`
- `V get(Object key)`
- `V put(K key, V value)`
- `V remove(Object key)`

Exemple : des Shape avec un nom

On associe un nom à chaque objet Shape. Le nom est la clé, la forme est la valeur associée .
L'interface Shape, et les classes Rectangle et Ellipse sont celles déjà vues.

```
public class ShapeMapTest{
  public static void main(String[] args){
    Shape r1 = new Rectangle(6,10);
    Shape r2 = new Rectangle(5,10);
    Shape e = new Ellipse(3,5);
    final Map<String,Shape> tree =
          new TreeMap<String,Shape>();
    tree.put("R1",r2);
    tree.put("R2",r1);
    tree.put("E",e);
    System.out.println(tree.get("R1").toStringArea());
  }
```

On obtient

```
$ java ShapeMapTest
aire = 60.0
$
```