Programmation Java

Cours 8

Collections

Collections : introduction

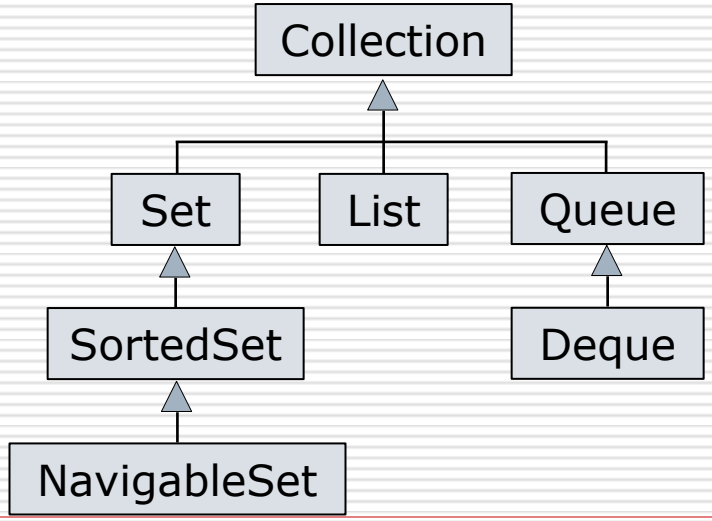
- Une collection Java :
  - conteneur d'objets
  - structure de taille dynamique
  - interfaces génériques
  - implémentations diverses
  - algorithmes efficaces
    - accès, parcours, recherche
    - modifications basiques (ajout, suppresion)
    - tris, fusion, extraction, renversement, …

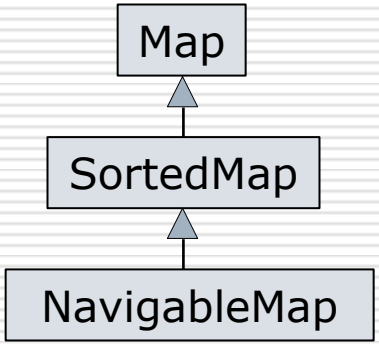
Les collections dans le JDK

- Packages du JDK :
  - java.util
    - collections de bases
      - interfaces + implémentations
    - boîte à outils Collections
    - modifications concurrentes non synchronisées
  - java.util.concurrent
    - collections à usage concurrent
    - mécanismes de synchronisation

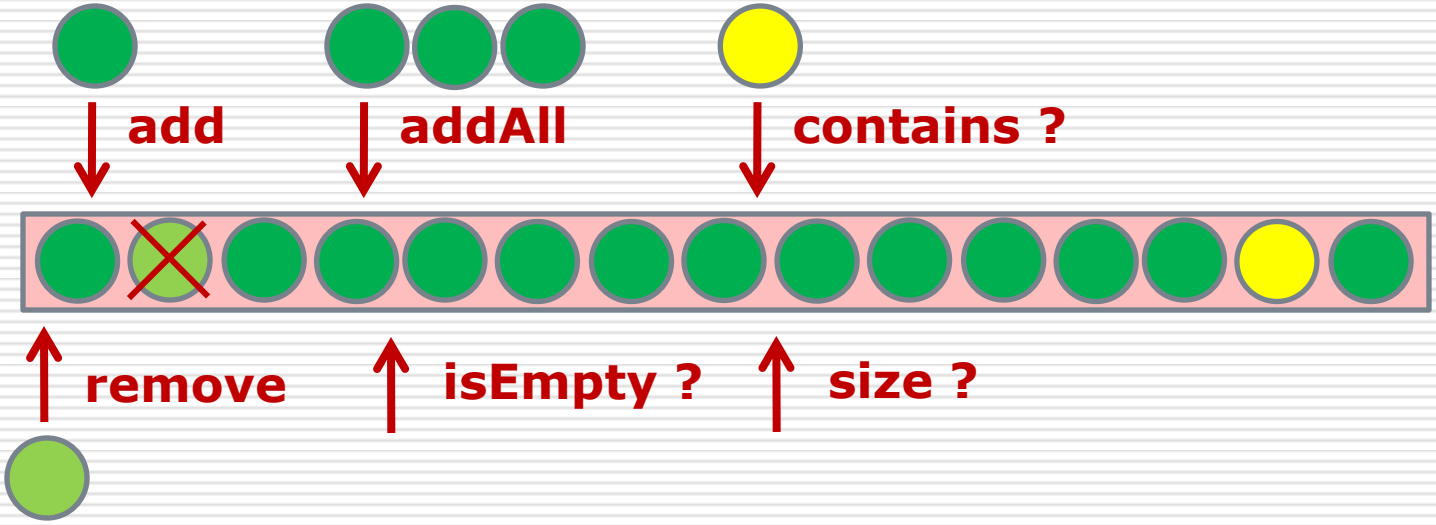
Interfaces de collection

- Donne la liste des opérations indépendamment de l'implémentation réelle.
- Deux grandes familles de structures :
  - les rangements linéaires (ensembles, listes, piles)
  - les structures d'indexations (catalogues, annuaires)



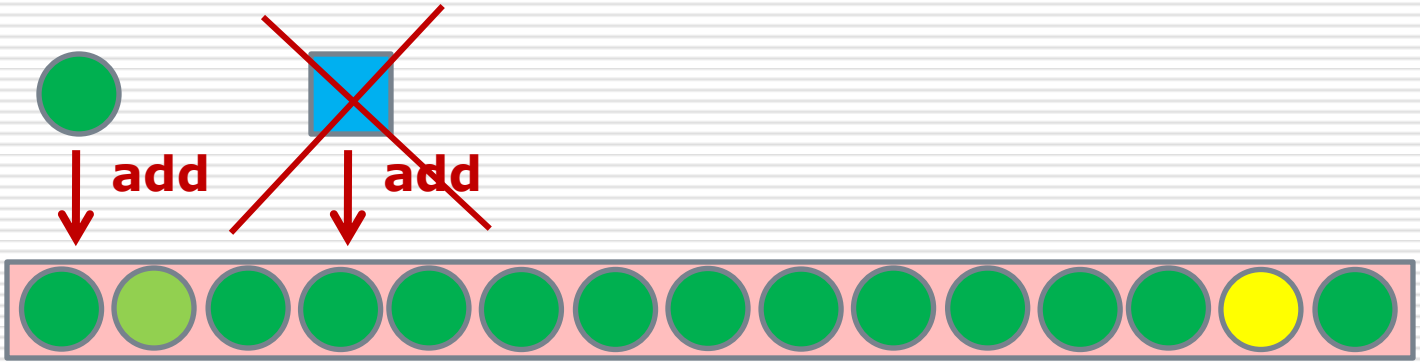


Interface Collection



- contrat des opérations communes
  - unitaires : add/remove/contains
  - par lot : addAll/removeAll/containsAll/retainAll/clear
  - optionnelles (UnsupportedOperationException)

Collection générique

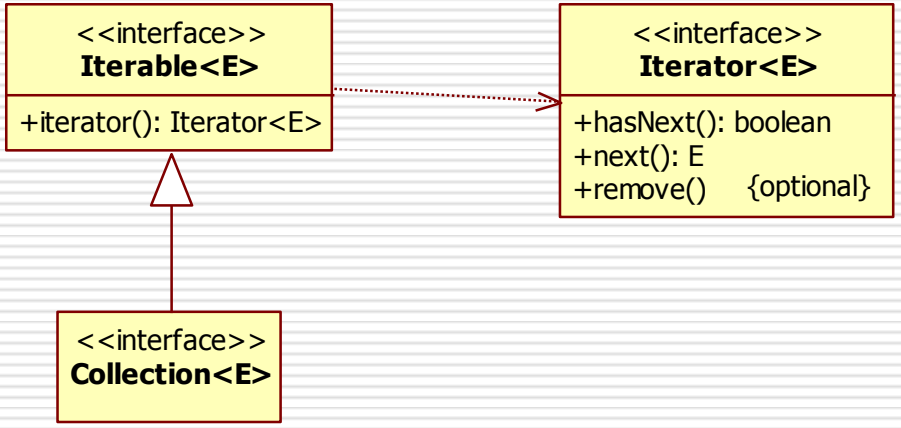


- Interface Collection<E>
- E = type **objet** des éléments de la structure
  - Collection<Voiture>
  - Collection<Integer>
  - ~~Collection<int>~~
- Manipulation contrôlée par le compilateur et la JVM

Collection générique

```
Collection<Voiture> c;
Voiture v;
Personne p;

c.add(v); // OK
c.add(p); // NO
```

Collection itérable



- Iterator
  - assure un parcours
    - exhaustif
    - non redondant
  - objet externe à la structure
  - offert par l'implémentation/structure
- Parcours
  - modification via la structure interdite
  - modification via l'itérateur possible (option)

Itération implicite

```
Collection<Voiture> garage;

for (Voiture v : garage) {
  v.nettoyer();
}
```

Itération explicite

```
Collection<Voiture> garage;
Iterator<Voiture> it;
boolean trouve = false;

it = garage.iterator()
while (it.hasNext() && !trouve) {
   Voiture v = it.next();
   trouve = (v.getCouleur() == Color.YELLOW);
}
System.out.println(trouve);
```

Collection vs Tableau

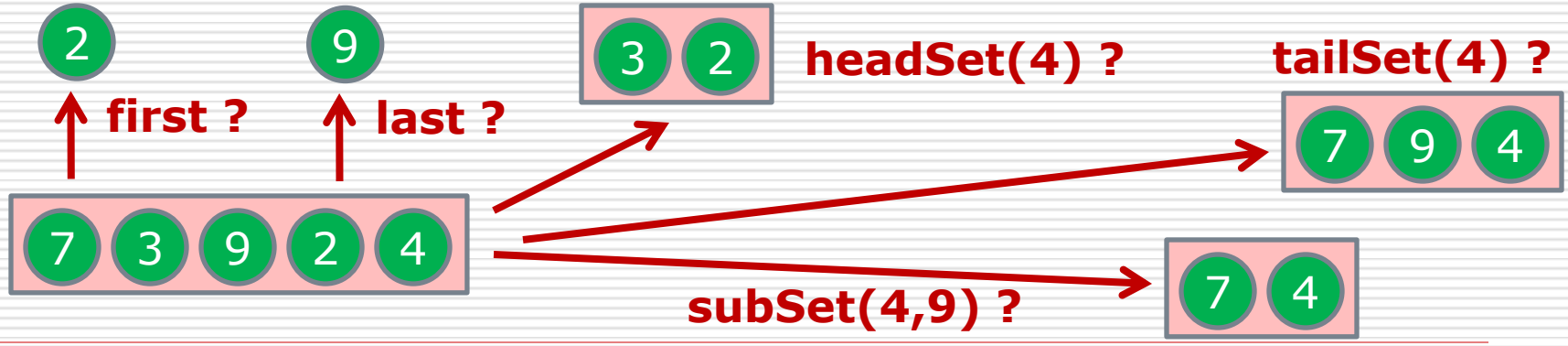
- Tableau
  - accès optimisé
  - taille fixe/manque de souplesse
  - extension périlleuse (cf langage C)
- Collection
  - manipulation éprouvée
  - choix implémentation/algo optimal
- Collection vers tableau : collec.toArray()

Les ensembles : Set<E>

- Pas de méthode complémentaire
- Contrainte supplémentaire
  - pas de doublons (par rapport à equals)
- 3 interfaces :
  - Set<E>
  - SortedSet<E> : ordre sur les éléments
  - NavigableSet<E> : recherche approchée
- Implémentations :
  - HashSet/TreeSet/LinkedHashSet/EnumSet/…

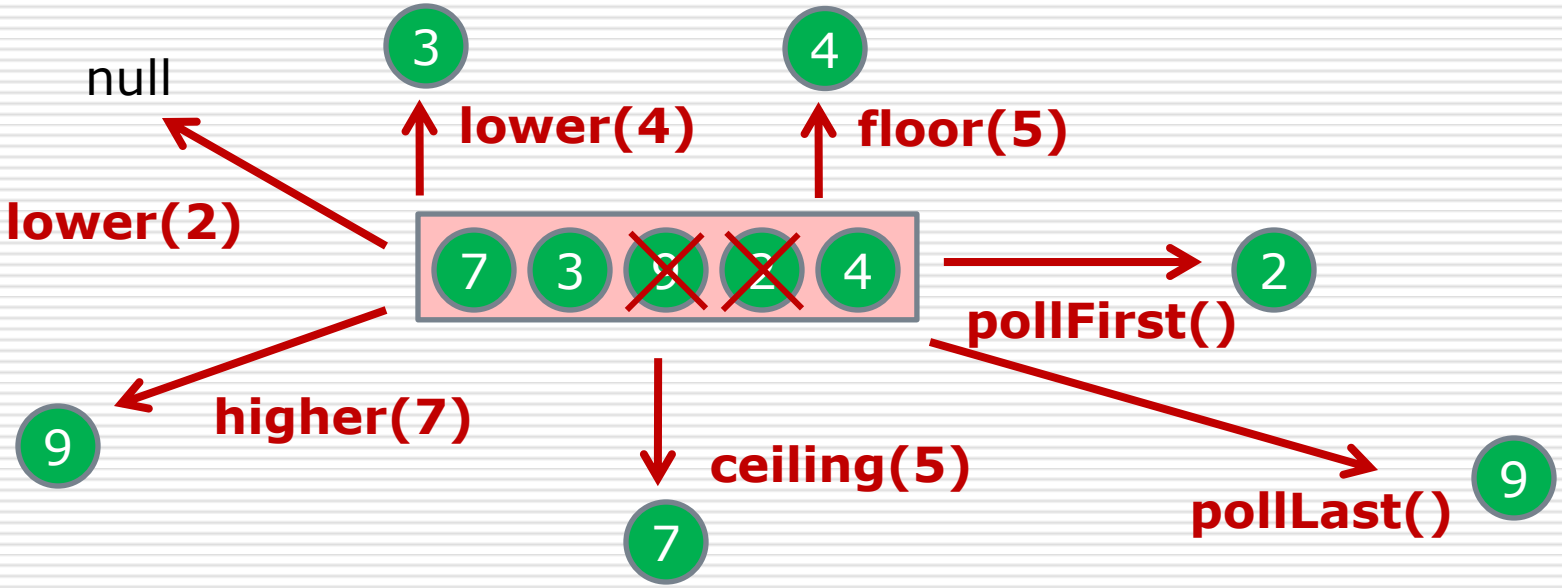
Ensemble ordonné SortedSet<E>

- Ordre total sur les éléments :
  - naturel : Integer, Double, …
  - implémentant Comparable
  - avec un Comparator externe (à la création)
- Méthodes supplémentaires (accès) :



Ensemble navigable NavigableSet<E>

- Etends l'interface SortedSet<E>
- Autorise des recherches approchées



Ensemble navigable NavigableSet<E>

- Parcours :
  - iterator() : ascendant/ordre
  - descendingIterator() : ordre inverse
- Ensemble ordre inverse
  - descendingSet()
- Réglage bornes incluses/excluses
  - headSet/tailSet/subSet
- Implémentation : TreeSet

Les listes : List<E>

- Doublons possibles
- Ajout d'un index (position)
- Méthodes supplémentaires :
  - liste.add(index,element)
  - liste.set(index,element)
  - liste.remove(index)
  - element = liste.get(index)
  - index = liste.indexOf(element)
  - index = liste.lastIndexOf(element)

Les listes

- Sous liste : subList(index1,index2)
- Parcours :
  - iterator() : ordre des positions
  - listIterator()
    - parcours double sens
    - ajout possible pendant le parcours
    - suppression possible pendant le parcours
- Implémentations :
  - LinkedList, ArrayList, Vector, …

Les piles/files : Queue/Deque

- Queues FIFO, LIFO et autres
- 2 modes pour les operations invalides
  - exception
  - special value
- Queue<E> : 3 opérations supplémentaires
  - ajout : q.add(e) / q.offer(e)
  - suppr tête : e=q.remove() / e=q.poll()
  - examine tête : e=q.element() / e=q.peek()

Les piles/files : Deque

- Deque<E> : Double Ended Queue
- Accès par la tête (head) :
  - addFirst/offerFirst i.e. push
  - removeFirst/pollFirst i.e. remove/poll
  - getFirst/peekFirst i.e. element/peek
- Accès par la queue (tail) :
  - addLast/offerLast i.e. add/offer
  - removeLast/pollLast i.e. pop
  - getLast/peekLast

Les piles/files : Deque

- Parcours :
  - iterator() : tête vers queue (head to tail)
  - descendingIterator() : inverse
- Implémentations :
  - ArrayDeque
  - LinkedList
  - ~~Stack~~

Indexation : Map<K,V>

- Couples (clé,valeur) : Map.Entry<K,V>
- Opérations élémentaires
  - map.put(clé,valeur)
  - valeur = map.get(clé)
  - map.remove(clé)
  - map.clear()
- Autres questions/recherches
  - containsKey/containsValue
  - isEmpty/size

Indexation : Map<K,V>

- Extractions
  - Clés : map.keySet() -> Set<K>
  - Valeurs : map.values() -> Collection<V>
- Parcours possibles :
  - par les clés
  - par les valeurs
- Implémentations :
  - HashMap, Hashtable, LinkedHashMap, EnumMap, TreeMap, WeakHashMap

Indexation ordonnée

- SortedMap<K,V>
  - ordre total sur les clés
  - firstKey(), lastKey()
  - headMap(toKey)
  - tailMap(fromKey)
  - subMap(fromKey,toKey)
- Implémentation :
  - TreeMap

Indexation ordonnée

- NavigableMap<K,V>
  - ordre total sur les clés
  - ceilingKey(k), floorKey(k)
  - firstEntry(), lastEntry()
  - ceilingEntry(),floorEntry(k)
  - higherEntry(k), lowerEntry(k)
  - pollLastEntry(), pollFirstEntry()
- Implémentation : TreeMap

Boîte à outils Collections

- Algorithmes divers (cf API) :
  - Min, max
  - Echanges
  - Tris
  - Rotations, permutations aléatoires
  - Conversions
  - Collections constantes (vide, 1 élément)
  - Synchronisation d'une collection

Programmation Java

Mapping UML vers Collections

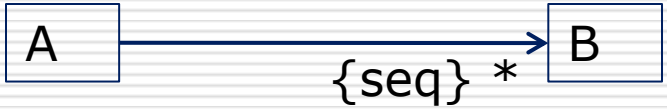
Ensemble d'élements


```
public class A {
        private Set<B> b;

        public A() {
                b = new HashSet<B>();
                ...
        }

        ...
}
```

Ensemble d'élements nommé

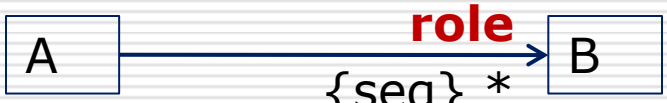

```
public class A {
        private Set<B> role;

        public A() {
                role = new HashSet<B>();
                ...
        }

        ...
}
```

Eléments en séquence



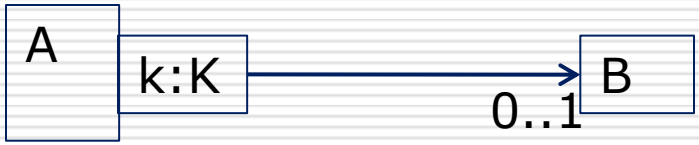

```
public class A {
        private List<B> b;

        public A() {
                b = new LinkedList<B>();
                ...
        }

        ...
}
```

Eléments nommés en séquence

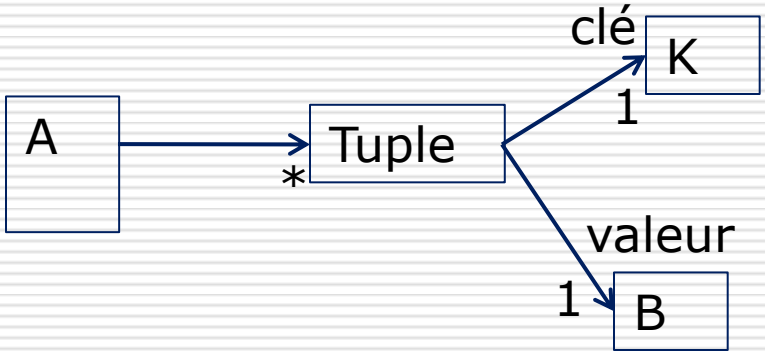



```
public class A {
        private List<B> role;

        public A() {
                role = new ArrayList<B>();
                ...
        }

        ...
}
```

Liste d'associations



≡




```
public class A {
        private Map<K,B> b;

        public A() {
                b = new HashMap<K,B>();
                ...
        }

        ...
}
```

Rq: si multiplicité *, type de b : Map<K, Set<B>>

Exemple : Company/Employee


```
public class Company {
        private Set<Person> employee;

        public Company() {
                employee = new HashSet<Person>();
                ...
        }

        ...
}
```

Gestion des employés

```
public class Company {
        private Set<Person> employee;

        public Company() {
                employee = new HashSet<Person>();
        }

        public void addEmployee(Person e) {
                employee.add(e);
        }

         public void removeEmployee(Person e) {
                employee.remove(e);
        }
}
```

Gestion des employés

```
public class Company {
        private Set<Person> employee;

        public Company() {
                employee = new HashSet<Person>();
        }

        public Company(Collection<Person> firstEmployees) {
                this();
                employee.addAll(firstEmployees);
        }

        …

}
```

Gestion des employés

```
public class Company {
        private Set<Person> employee;
        …

        public void addEmployee(Collection<Person> e) {
                employee.addAll(e);
        }

        public void addEmployee(Person... e) {
                employee.addAll(Arrays.asList(e));
        }

         public void removeEmployee(Collection<Person> e) {
                employee.removeAll(e);
        }
        …
}
```

Gestion des employés

```
public Set<Person> getEmployee() {
        return Collections.unmodifiableSet(employee);
}

public Person getEmployee(String criteria) {
        Person p = null;
        Iterator<Person> it = employee.iterator();
        while ((p == null) && it.hasNext()) {
                Person next = it.next();
                if (…) { // next matches criteria
                        p = next;
                }
        }
        return p;
}
```

Gestion des employés (alt.)

```
public class Company {
        private Set<Person> employee;

        public Company() {
                employee = new HashSet<Person>();
        }

        public void employ(Person e) {
                employee.add(e);
        }

         public void fire(Person e) {
                employee.remove(e);
        }
}
```