Les collections

Branches SRT et ISI – Unité de valeur LO02

Guillaume Doyen

Contact : guillaume.doyen@utt.fr

Plan du cours

- Introduction
- Les interfaces
- Les implantations

Introduction

- **Qu'est-ce qu'une collection ?**
  - Une collection regroupe un ensemble d'objets homogènes dans une structure qui facilite leur manipulation
  - Par exemple
    - Un ensemble de cartes dans une main de joueur
    - Un ensemble de voitures dans un garage
- **Qu'appelle-t-on les collections ?**
  - Les collections Java sont un ensemble de classes qui permettent de manipuler ces ensembles d'objets
    - Différentes classes car différentes structures et différentes manipulations

Introduction

- **Pourquoi utiliser les collections ?**
  - Réduction de l'effort de développement
    - Permet de concentrer le développement sur les points liés à l'application et pas sur les problèmes de bas niveau comme la manipulation d'agrégats
  - Amélioration de la rapidité et de la qualité du programme
    - Les implantations de collections fournies par Java sont optimisées et présentent de meilleures performances que celles qu'on pourrait écrire manuellement
  - Interopérabilité entre API
    - Les collections sont indépendantes de toute application ou librairie mais sont utilisées par celles-ci, étant ainsi une sorte de « glue »

Introduction

- **Pourquoi utiliser les collections ?**
  - Réduction de l'effort de compréhension et d'utilisation de nouvelles API
    - Les API n'utilisent plus de « sous-API » dédiées à des collections propres. Les collections Java sont valables pour l'ensemble des API
  - Réduction de l'effort pour concevoir et développer de nouvelles API
    - Les collections ne demandent qu'à être utilisées sans avoir à être ré-inventées
  - Ré-utilisation du code
    - Les éléments standards comme les collections peuvent être ré-utilisés plus tard facilement

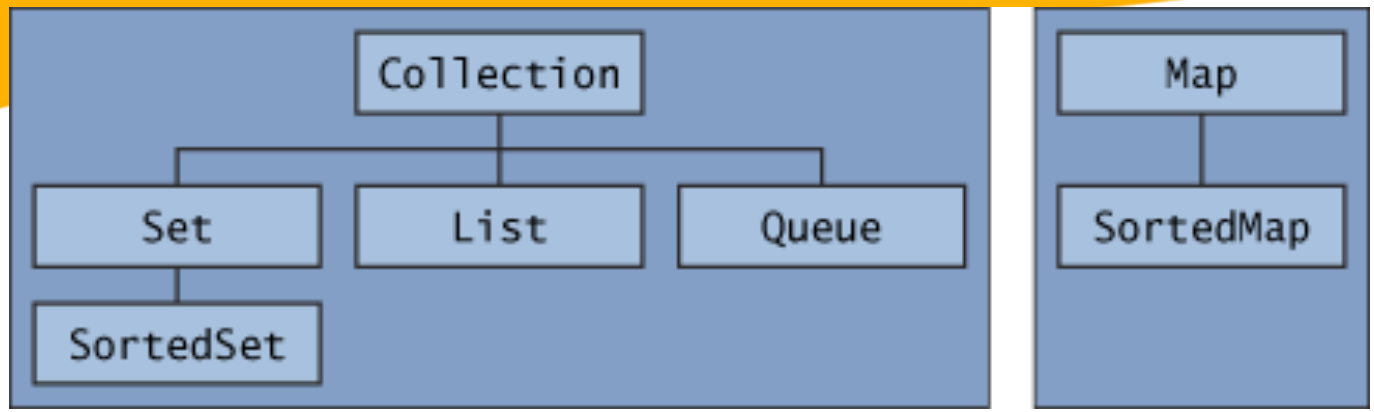
Introduction

- **Que sont les collections ?**
  - Interfaces
    - Un ensemble d'interface qui spécifie les fonctionnalités de collections indépendamment de leur implantation
  - Implantations
    - Un ensemble de classes directement utilisables qui implantent les interfaces de collections
  - Algorithmes
    - Des méthodes de manipulation de collections (recherche, tri) qui sont polymorphes car implantées de différentes manières

Plan du cours

- **Introduction**
- **Les interfaces**
- **Les implantations**

Les interfaces de collections



Source : Java Sun/Oracle tutorial (http://download.oracle.com/javase/tutorial/)

- **Description générale**
  - Collection : une collection au sens le plus large et abstrait qui soit
  - Set : une collection qui ne contient pas d'élément dupliqué
  - List : une collection ordonnée qui peut contenir des éléments dupliqués
  - Queue : collection ordonnées où les éléments sont généralement placés dans une file FIFO
  - Map : collection d'objets associés à une clé unique

Les interfaces de collections

- **Les méthodes optionnelles**
  - Spécifiées comme telles dans la documentation de l'API
  - N'indiquent pas que l'implémentation de la méthode est optionnelle
    - Sinon la classe qui implémente l'interface de spécification de collection serait abstraite
  - Indiquent qu'en cas de mauvaise usage de ces méthodes, les opérations qu'elle contiennent ne seront pas obligatoirement effectuées
    - Selon l'implémentation, une valeur de retour spécifique ou la levée d'une exception l'indiquera
  - Pour le developpeur, il ne faut donc pas considérer qu'un appel à une méthode optionnelle sera toujours suivi d'un effet

L'interface Collection

```
public interface Collection<E> extends Iterable<E> {
    // Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);          //optional
    boolean remove(Object element); //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);        //optional
    boolean retainAll(Collection<?> c);        //optional
    void clear();                              //optional

    // Array operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

Parcours d'une collection 1/3

- **1ère méthode : Utilisation d'un iterateur**
  - Un itérateur est un objet de la classe `Iterator` qui permet de récupérer une à une les références d'objets stockés dans une collection
  - L'ordre dans lequel les objets sont récupérés (ordre d'itération) n'est pas connu par le programmeur car il est lié à l'implémentation de la collection (ordonnée, non ordonnée, …)
  - La méthode `E next()` retourne l'objet suivant dans l'itération et fait avancer l'itérateur jusqu'à l'objet suivant (qui sera retourné par un appel à cette même méthode)
  - La méthode `boolean hasNext()` permet de savoir s'il existe un prochain objet dans l'ordre d'itération ou si l'itérateur est à la fin de son parcours.
  - L'itérateur est un patron de conception objet indépendant de Java qu'on retrouve dans de nombreux langages de programmation

Parcours d'une collection 2/3

- **L'interface Iterator**
  - Parcours mono-directionnel d'une collection
  - Méthode à utiliser de manière privilégiée

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

  - Exemple d'utilisation

```
static void filter(Collection<?> c) {
    Iterator<E> it = c.iterator();
    while(it.hasNext())
        if (!cond(it.next())) // avec boolean cond(<E> e)
            it.remove();      // une méthode de test sur un
}                             // élément de la collection
```

Parcours d'une collection 3/3

- **2nde méthode : la boucle for**
  - Forme étendue de la boucle for qui permet d'itérer sur chacun des éléments d'un ensemble
    - L'ensemble n'est pas forcément une collection (par ex. tableau)

```
for (Object o : collection)
    System.out.println(o);
```

- **Quelle méthode choisir ?**
  - L'utilisation d'un itérateur est recommandée et permet d'effectuer des opérations sur la collection (par ex. remove)
    - L'itérateur est un patron de conception reconnu et utilisé dans de nombreux langages de programmation objet
  - La boucle for permet un simple parcours sans manipulation
    - Raccourci syntaxique seulement valable pour Java

L'interface Set

- **Définition**
  - L'interface Set est une collection qui ne peut pas contenir d'élément dupliqué. Elle représente un « ensemble », au sens mathématique du terme.
  - Ses méthodes sont toutes héritées de l'interface Collection mais présentent des garanties supplémentaires (vérification de l'unicité d'un élément)
- **Utilisation**
  - Les méthodes « bulk » (xxxAll) sont particulièrement utiles pour manipuler des ensembles d'éléments uniques
    - Union, intersection, sous-ensemble, …
- **Implantations**
  - HashSet, TreeSet, LinkedHashSet

L'interface Set

```
public interface Set<E> extends Collection<E> {
    // Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);          //optional
    boolean remove(Object element); //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c); //optional
    boolean removeAll(Collection<?> c);        //optional
    boolean retainAll(Collection<?> c);        //optional
    void clear();                              //optional

    // Array Operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

L'interface List

- **Définition**
  - L'interface List est une collection ordonnée d'éléments potentiellement dupliqués. Elle permet ainsi de manipuler les éléments de manière ordonnée.
- **Utilisation**
  - Les méthodes de manipulation liée à la position d'un élément sont l'intérêt principal de cette collection
- **Implantations**
  - ArrayList, LinkedList
- **Comparaison avec la classe Vector**
  - Utiliser de préférence une liste car
    - Les implantations permettent une utilisation plus souple
    - Quelques bugs mineurs sont fixés

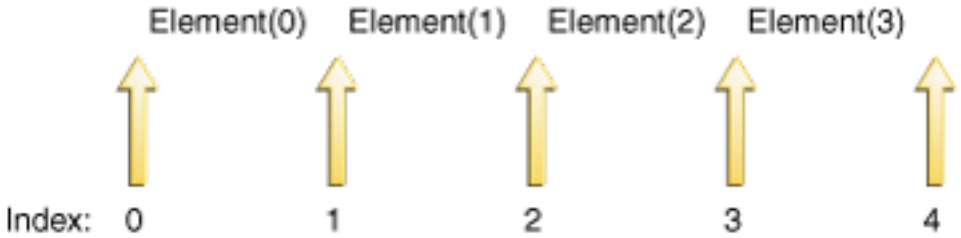
L'interface List

```
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element);    //optional
    boolean add(E element);         //optional
    void add(int index, E element); //optional
    E remove(int index);            //optional
    boolean addAll(int index,
        Collection<? extends E> c); //optional

    // Search
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Range-view
    List<E> subList(int from, int to);
}
```

Parcours d'une liste

- **Un itérateur étendu pour la structure de liste**

```
public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();
    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    void remove(); //optional
    void set(E e); //optional
    void add(E e); //optional
}
```



Source : Java Sun/Oracle tutorial (http://download.oracle.com/javase/tutorial/)

L'interface Queue

- **Définition**
  - Une file (Queue) est une collection ordonnée d'éléments qui peuvent être dupliqués. Généralement les files sont du type FIFO.
- **Utilisation**
  - Les files sont généralement utilisées pour mettre en attente des éléments avant leur traitement. La programmation concurrente utilise notamment ce type de collection.
- **Implantations**
  - LinkedList, PriorityQueue, …

L'interface Queue

```
public interface Queue<E> extends Collection<E> {
    E element();
    void add(E e);
    boolean offer(E e);
    E peek();
    E poll();
    E remove();
}
```

	Lève des exceptions	Retourne une valeur spéciale
Insertion	`add()`	`offer()`
Retrait	`remove()`	`poll()`
Accès	`element()`	`peek()`

L'interface Map

- **Définition**
  - L'interface Map représente une collection de clés associées à un objet. Chaque clé doit être unique et ne peut être associée qu'à un seul objet.
- **Utilisation**
  - L'interface Map représente symboliquement les functions mathématiques.
    - A une valeur de x unique (la clé), on associe une valeur de y (l'objet associé)
- **Implantation**
  - HashMap, TreeMap, LinkedHashMap
- **Comparaison avec la classe Hashtable**
  - Fonctions étendues, suppression de confusions dans les noms de méthodes

```
public interface Map<K,V> {
    // Basic operations
    V put(K key, V value);
    V get(Object key);
    V remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    // Bulk operations
    void putAll(Map<? extends K, ? extends V> m);
    void clear();

    // Collection Views
    public Set<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K,V>> entrySet();

    // Interface for entrySet elements
    public interface Entry {
        K getKey();
        V getValue();
        V setValue(V value);
}   }
```

Plan du cours

- **Introduction**
- **Les interfaces**
- **Les implantations**

Synthèse des implantations

- **Les implantations les plus utilisées**
  - Couvrent tous les usages standard des collections
  - Résumé dans le tableau suivant

Interface	Implementation				
	Hash table	Resizable array	Tree	Linked list	Hash table + linked list
Set	`HashSet`		`TreeSet`		`Linked-HashSet`
List		`ArrayList`		`LinkedList`	
Queue				`LinkedList`	
Map	`HashMap`		`TreeMap`		`Linked-HashMap`

Source : Java Sun/Oracle tutorial (http://download.oracle.com/javase/tutorial/)

Choix d'une collection

- **Quelques règles à savoir**
  - Raisonner en terme d'interface et pas d'implantation
  - Les différences entre implantations
    - Les TreeXXX de par l'utilisation d'un arbre permettent d'ordonner si besoin les éléments
    - Les autres différences sont liées à la performance des algorithmes de traitement
      - Compléxité constante, logarithmique, linéaire, polynomiale
    - Pour plus d'information : http://docs.oracle.com/javase/tutorial/collections/implementations/
  - Les collections ne sont pas synchronisées
    - Les anciennes (par ex. Hashtable, Vector) l'étaient mais le choix de conception actuel des collections est de privilégier un usage courant plus simple
    - L'accès concurrent doit donc être géré manuellement ou wrappé par une collection dédiée (cf. classe Collections)