Lic.Info 2ème année Université d'Orléans

SOL3IF02 – POO

*Feuille d'exercice n°6 – Les collections*

**Exercice** Nous allons reprendre l'exercice 1 du TD3 et .réécrire plusieurs fois la classe `Magasin` en utilisant différentes interfaces de la bibliothèque des collections Java consultables en Annexe.

Dans les trois cas décrits ci-dessous, la classe `Magasin` fournira un constructeur sans paramètre et les méthodes suivantes :

- `public boolean ajouterArticle(Article a)` qui ajoute l'article passé en paramètre au magasin.
- `public Article rechercherArticle(int reference)` qui retourne l'article correspondant à la référence passée en paramètre si elle existe dans le magasin et `null` sinon.
- `public boolean supprimerArticle (int ref )` qui supprime l'article correspondant à la référence passée en paramètre si elle existe dans le magasin et retourne `true` sinon retourne `false`.
- `public boolean promotion(int ref, int pourcentage)` qui applique la promotion à l'article correspondant à la référence passée en paramètre si elle existe dans le magasin et retourne `true` sinon retourne `false`.
- `public void liquidationTotale(int pourcentage)` qui applique une promotion à tous les articles du magasin selon le pourcentage passé en paramètre.
- `public String toString()` qui retourne les informations des articles présents dans le magasin sous forme d'une chaîne de caractères.

  Vous utiliserez les différentes méthodes permettant d'itérer sur une collection :

  - la méthode `Iterator<T> iterator()` de l'interface `Iterable<T>` et sa version `for (… :…)`
  - la méthode `default void forEach (Consumer< ? super T> action)` de l'interface `Iterable<T>`.

1. Créer une première classe `Magasin` caractérisée par un attribut `lesArticles`. Choisir le type de l' attribut parmi les interfaces de la bibliothèque des collections sachant qu'on souhaite gérer la collection des articles comme un ensemble, sans doublons. Quelles méthodes doit-on redéfinir dans la classe `Article` ?
   Ajouter à cette classe `Magasin` la méthode ci-dessous et la compléter pour que les articles de la liste retournée soient triés par ordre croissant de leur prix de vente.

```
public List<Article> triParOrdreCroissant(){
   List<Article> liste= new ArrayList<Article>(lesArticles);
   ...
   return liste ;
}
```

2. Créer une deuxième classe `Magasin` caractérisée par un attribut `lesArticles`. Choisir le type de l'attribut parmi les interfaces de la bibliothèque des collections sachant qu'on souhaite gérer la collection des articles comme un ensemble, sans doublons dont les éléments sont munis d'une relation d'ordre.
   Que doit-on ajouter à la classe `Article` ? Implémentez-le sachant qu'on souhaite comparer les articles par ordre croissant de leur référence.

   Ajouter à cette classe `Magasin` les méthodes :

   - `public Article lePremierArticle()` qui retourne l'article ayant la plus petite référence.
   - `public Article leDernierArticle()` qui retourne l'article ayant la plus grande référence.

3. Quelle modification doit-on apporter à la classe `Magasin` ci-dessus si on souhaite modifier la relation d'ordre sans modifier l'ordre naturel défini dans la classe `Article`. On souhaite maintenant comparer les articles par ordre croissant de leur libellé.
4. Créer une troisième classe `Magasin` caractérisée par un attribut `lesArticles`. Choisir le type de l' attribut parmi les interfaces de la bibliothèque des collections sachant qu'on souhaite enregistrer des paires d'informations de type (référence, Article). Ajouter un article `a` de référence `r` consistera à ajouter une paire `(r, a)`.
   Pour cette classe, la méthode `toString` affichera les paires sur des lignes différentes au format : Référence : r – informations sur l'article.

# ANNEXE : Quelques interfaces de la bibliothèque des collections

## Interface `List<E>`

| Modifier and Type | Method and Description |
|---|---|
| boolean | **add**(E e)<br>Appends the specified element to the end of this list (optional operation). |
| void | **add**(int index, E element)<br>Inserts the specified element at the specified position in this list (optional operation). |
| boolean | **contains**(Object o)<br>Returns true if this list contains the specified element. |
| E | **get**(int index)<br>Returns the element at the specified position in this list. |
| int | **indexOf**(Object o)<br>Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element. |
| boolean | **isEmpty**()<br>Returns true if this list contains no elements. |
| Iterator<E> | **iterator**()<br>Returns an iterator over the elements in this list in proper sequence. |
| int | **lastIndexOf**(Object o)<br>Returns the index of the last occurrence of the specified element in |

| | |
|---|---|
| | this list, or -1 if this list does not contain the element. |
| `ListIterator<E>` | `listIterator()`<br>Returns a list iterator over the elements in this list (in proper sequence). |
| `E` | `remove(int index)`<br>Removes the element at the specified position in this list (optional operation). |
| `boolean` | `remove(Object o)`<br>Removes the first occurrence of the specified element from this list, if it is present (optional operation). |
| `E` | `set(int index, E element)`<br>Replaces the element at the specified position in this list with the specified element (optional operation). |
| `int` | `size()`<br>Returns the number of elements in this list. |
| `default void` | `sort(Comparator<? super E> c)`<br>Sorts this list according to the order induced by the specified `Comparator`. |

## Interface Set<E>

| Modifier and Type | Method and Description |
|---|---|
| `boolean` | `add(E e)`<br>Adds the specified element to this set if it is not already present (optional operation). |
| `boolean` | `contains(Object o)`<br>Returns `true` if this set contains the specified element. |
| `boolean` | `isEmpty()`<br>Returns `true` if this set contains no elements. |
| `Iterator<E>` | `iterator()`<br>Returns an iterator over the elements in this set. |
| `boolean` | `remove(Object o)`<br>Removes the specified element from this set if it is present (optional operation). |
| `boolean` | `removeAll(Collection<?> c)`<br>Removes from this set all of its elements that are contained in the specified collection (optional operation). |
| `boolean` | `retainAll(Collection<?> c)`<br>Retains only the elements in this set that are contained in the specified collection (optional operation). |
| `int` | `size()`<br>Returns the number of elements in this set (its cardinality). |

## Interface SortedSet<E>

| Modifier and Type | Method and Description |
|---|---|
| `Comparator<? super E>` | `comparator()`<br>Returns the comparator used to order the elements in this set, |

| | |
|---|---|
| | or null if this set uses the **natural ordering** of its elements. |
| E | first()<br>Returns the first (lowest) element currently in this set. |
| SortedSet<E> | headSet(E toElement)<br>Returns a view of the portion of this set whose elements are strictly less than toElement. |
| E | last()<br>Returns the last (highest) element currently in this set. |
| SortedSet<E> | subSet(E fromElement, E toElement)<br>Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. |
| SortedSet<E> | tailSet(E fromElement)<br>Returns a view of the portion of this set whose elements are greater than or equal to fromElement. |

# Rappel : Constructeurs de la classe TreeSet<E>

| Constructor and Description |
|---|
| TreeSet()<br>Constructs a new, empty tree set, sorted according to the natural ordering of its elements. |
| TreeSet(Comparator<? super E> comparator)<br>Constructs a new, empty tree set, sorted according to the specified comparator. |

# Interface Map<K,V>

| Modifier and Type | Method and Description |
|---|---|
| boolean | containsKey(Object key)<br>Returns true if this map contains a mapping for the specified key. |
| boolean | containsValue(Object value)<br>Returns true if this map maps one or more keys to the specified value. |
| Set<Map.Entry<K,V>> | entrySet()<br>Returns a Set view of the mappings contained in this map. |
| V | get(Object key)<br>Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| boolean | isEmpty()<br>Returns true if this map contains no key-value mappings. |
| Set<K> | keySet()<br>Returns a Set view of the keys contained in this map. |
| V | put(K key, V value)<br>Associates the specified value with the specified key in this map (optional operation). |
| default V | putIfAbsent(K key, V value)<br>If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current value. |
| V | remove(Object key)<br>Removes the mapping for a key from this map if it is present (optional opera- |

| | |
|---|---|
| | tion). |
| default boolean | **remove**(**Object** key, **Object** value)<br>Removes the entry for the specified key only if it is currently mapped to the specified value. |
| **Collection**<**V**> | **values**()<br>Returns a **Collection** view of the values contained in this map. |
| int | **size**()<br>Returns the number of key-value mappings in this map. |