Université de Lille

TD 4 : Ensembles et Tables de Correspondance

Structures de données
DUT info 2ème année
2019–2020

1 L'interface Set<E>

Quelques méthodes de cette interface :

boolean	add(E o) ajoute l'élément spécifié au sein de l'objet `Set`, s'il n'est pas déjà présent
boolean	addAll(Collection<E> c) ajoute tous les éléments de la collection spécifiée au sein de l'objet `Set`, s'ils ne sont pas déjà présents
boolean	isEmpty() retourne `true` si l'ensemble ne contient aucun élément
boolean	contains(Object o) retourne `true` si l'élément spécifié est contenu dans l'objet `Set`
boolean	containsAll(Collection< ?> c) retourne `true` si l'objet `Set` contient tous les éléments de la collection spécifiée
Iterator<E>	iterator() retourne un itérateur sur les éléments de l'ensemble
boolean	remove(Object o) supprime l'élément spécifié de la collection, s'il est présent
boolean	removeAll(Collection< ?> c) supprime tous les éléments contenus dans la collection spécifiée, de l'objet `Set`, s'ils y sont présents
boolean	retainAll(Collection< ?> c) retient seulement les éléments de l'objet `Set`, qui sont contenus dans la collection spécifiée
int	size() retourne le nombre d'éléments de l'ensemble

Q 1. On veut créer un `Set` *ens* contenant les jours de la semaine.

Q 2. Laissez dans *ens* uniquement les jours ouvrés.

Q 3. Écrivez une méthode `public void afficher(Set s)` permettant d'afficher le nombre d'éléments d'un ensemble, puis chaque élément de cet ensemble avec un élément par ligne.
Voici un petit rappel de la javadoc de Iterator<E> qui peut servir :

boolean	hasNext() Retourne `true` si l'itération a encore des éléments.
E	next() Retourne le prochain élément de l'itération.
void	remove() Retire de la collection sous-jacente, le dernier élément retourné par cet itérateur (opération optionnelle).

2 Implémentation de l'interface Set<E>

Q 1. Donnez au moins deux implémentations possibles de l'interface `Set`.

Q 2. Nous choisissons comme conteneur du `Set`, un tableau non trié. Quels sont les attributs nécessaires à votre classe `ArraySet` ?

Q 3. Programmez les méthodes `isEmpty` et `size`.

Q 4. Que faut-il comme information pour itérer sur le contenu de notre ensemble ? Programmez un `Iterator` sur le `ArraySet`, et programmez la méthode `Iterator<E> iterator()`.

Q 5. Programmez une méthode privée `indexOf` qui retourne l'index de l'élément passé en paramètre (-1 s'il n'est pas présent). Expliquez pourquoi cette méthode doit absolument être privée.

Q 6. Programmez une méthode privée `expand` qui double la taille du conteneur, puis une seconde méthode privée `reduce` qui la réduit de moitié.

Q 7. Programmez alors toutes les autres méthodes de l'interface `Set`. N'hésitez pas à utiliser à bon escient les méthodes privées !

3 L'interface Map<K,V>

Pour la très célèbre course "Le tour de la Terre Adélie en Pédalo", l'organisateur vous a demandé une aide pour la mise au point du logiciel gardant toutes les données de la course.
Q 1. Créez une classe `Temps` qui permette de représenter les différents temps des coureurs. Cette classe contient, entre autres,

— une redéfinition de la méthode `toString` avec un affichage du type "HH-MM-SS" (pas de temps supérieur à 100h),
— une méthode `add` qui retourne l'addition de deux `Temps`.

D'autre part voici un extrait de la JavaDoc de Map :

void	clear() Removes all of the mappings from this map (optional operation).
boolean	containsKey(Object key) Returns true if this map contains a mapping for the specified key.
boolean	containsValue(Object value) Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K,V> >	entrySet() Returns a Set view of the mappings contained in this map.
boolean	equals(Object o) Compares the specified object with this map for equality.
V	get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
int	hashCode() Returns the hash code value for this map.
boolean	isEmpty() Returns true if this map contains no key-value mappings.
Set<K>	keySet() Returns a Set view of the keys contained in this map.
V	put(K key, V value) Associates the specified value with the specified key in this map (optional operation).
void	putAll(Map< ? extends K, ? extends V> m) Copies all of the mappings from the specified map to this map (optional operation).
V	remove(Object key) Removes the mapping for a key from this map if it is present (optional operation).
int	size() Returns the number of key-value mappings in this map.
Collection<V>	values() Returns a Collection view of the values contained in this map.

Puis un extrait de la JavaDoc de Map.Entry :

boolean	equals(Object o) Compares the specified object with this entry for equality.
K	getKey() Returns the key corresponding to this entry.
V	getValue() Returns the value corresponding to this entry.
int	hashCode() Returns the hash code value for this map entry.
V	setValue(V value) Replaces the value corresponding to this entry with the specified value (optional operation).

Q 2. Les temps effectués par les coureurs à chaque étape sont conservés dans une liste `resultatsEtapes` de `Map` (la liste et les `Map` sont déjà créées). Écrivez les quelques lignes de code qui permettent de stocker, pour le coureur `nomCoureur`, son temps `tEtape` pour l'étape d'index `iEtape`. Si vous aviez dû créer la liste, quelle implémentation de `List` vous semblerait la plus appropriée ?

Q 3. On veut maintenant, pour un coureur `nomCoureur`, récupérer la liste des temps qu'il a effectués pour toutes les étapes successives. Écrivez les quelques lignes de code qui remplissent cette liste.

Q 4. Pour le classement final, il faut connaître le temps cumulé des étapes pour tous les coureurs. Écrivez les quelques lignes de code qui créent et remplisse la `Map` associant chaque coureur à son temps cumulé.

Q 5. Est-il possible d'avoir maintenant le classement des coureurs ? Décrivez (sans les lignes de codes) les étapes qui permettraient de générer un tableau des noms correspondant au classement final à partir des temps cumulés. Quel est le problème ?

Vous avez implémenté dans le premier TD l'interface `Comparable` pour spécifier les relations d'ordre entre deux objets de même type à travers la méthode `compareTo(Object o1)`.

Q 6. Dans la classe `Temps`, implémentez l'interface `Comparable`.

Pour comparer aisément et élégamment les couples de la table (chaque couple est du type `Entry<K,V>`), vous allez maintenant définir un `Comparator`. Cette technique est aussi utilisée lorsque vous voulez définir plusieurs ordres différents pour un même type d'objet, ce qui n'est pas possible avec `Comparable`. Ci-dessous, vous trouverez un exemple de cette utilisation où l'on définit un `Comparator` d'employés basé sur l'ancienneté. On précise que le type `Date` de la bibliothèque Java implémente l'interface `Comparable`.

```
public class Employe implements Comparable<Employe> {
    public String nom()      { ... }
    public int numero()     { ... }
    public Date dateEmbauche() { ... }
    ...
}
public class Quelconque {
    Comparator<Employe> anciennete = new Comparator<Employe>() {
            public int compare(Employe e1, Employe e2) {
                return e2.dateEmbauche().compareTo(e1.dateEmbauche());
            }
    };

}
```

Une fois défini, le `Comparator` pourra être passé en paramètre de la méthode `sort` de `Collection` comme décrit ci-dessous.

static <T> void	sort(List<T> list, Comparator<? super T> c) Sorts the specified list according to the order induced by the specified comparator.

Q 7.Écrivez un `Comparator` d'`Entry` (qui correspond au couple (nom, temps cumulés)). Copiez ces couples dans une `ArrayList` et triez là à l'aide de la méthode `sort` de `Collections` et du comparator. Écrivez le code qui permet maintenant d'afficher le classement sous la forme :

```
[rang] nom : temps cumulés
```