Série No : 1

Orientée-Objet

Abdallah EL Asmar

Considérons la classe suivante CircularListe:

```
public class CircularList {
        private Object[] elements;
        public CircularList(Object... elements) {
                this.elements = elements.clone();
        }
        /** get the element at index i % size(). */
        public Object get(int i) {
                return elements[i % elements.length];
        }
    /**
    * return the first index of the object o in the circular list,
    *-1 if the object is not present.
    */
        public int indexOf(Object o) {
                for (int i = 0; i < elements.length; i++) {
                        if (elements[i].equals(o)) return i;
                }
        return -1;
        }
       /** number of elements contained in the circular list. */
        public int size() {
                return elements.length;
        }
}
```

a) Modifier CircularList de telle façon que l'instruction `System.out.println(C);` où C est une instance de CircularList et n sa taille, affichera sur la sortie standard le résultat suivant :

Circular list of size n

b) Ajouter une méthode Object nextElement() qui renvoie l'élément suivant l' élément précédemment retourné par nextElement. Si nextElement n'a jamais été appelée, elle renverra le premier élément de la liste circulaire, c'est-à-dire celui d'indice 0. D'autre part, l'élément qui suit l'élément d'indice size() - 1 est l'élément d'indice 0.
Par exemple, les instructions suivantes

```
CircularList cl = new CircularList(1, 2, 3);
for (int i = 0; i < 10; i++)
System.out.println(cl.nextElement());
```

auront comme résultat l'affichage sur la sortie standard de

1 2 3 1 2 3 1 2 3 1

c) Modifier CircularList de telle façon qu'invoquer la méthode `indexOf(Object o)` lève une exception `ObjectMissingException` si l'objet o n'est pas présent. Donner le code de la classe `ObjectMissingException`.

d) Modifier CircularList de façon à ce que l'utilisateur ne peut créer qu'une seule instance à la fois.

e) Créer une classe CircularListGeneric comme version générique de CircularList , de façon que le type des éléments de CircularListGeneric soit générique.

f) On souhaite pouvoir utiliser une instance de la classe CircularListGeneric comme instance d'Iterator. Ecrire une classe CircularListIterator qui étend cette classe et implémente Iterator. La méthode hasNext() retournera true tant que il y'aura des éléments dans la liste. La méthode remove() lèvera une exception UnsupportedOperationException.

Code de l'interface Iterator :

```
public interface Iterator<E> {
    /** returns true if the iteration has more elements. */
    public boolean hasNext();
    /* returns the next element of the iteration. Throws
    NoSuchElementException if the iteration has no more element. */
    public E next();
    /* Removes from the underlying collection the last element returned by the iterator.
    Throws UnsupportedOperationException if the remove operation is not supported by
    this Iterator. */
    public void remove();
}
```

g) Créer une classe CircularListDelegate comme une solution alternative de la classe CircularListIterator, cette solution doit être basée sur la délégation.

h) Créer une sous-classe de la classe CircularList et vérifier si CircularListDelegate est compatible avec cette sous classe.
Donner (en quelques lignes) les avantages et les inconvénients de chacune de deux solutions proposées en (f) et (g).

i) Créer une classe CircularListIteratorAnonym cette solution consiste d'utiliser une classe anonyme qui implémente l'interface Iterator dans la classe CircularListGeneric.