Algorithmique avancée - TD1

—o000o——o000o—

Exercices simples sur les listes chaînées

1 Construction de listes chaînées

On travaille tout d'abord sur la notion de constructeur de liste et de manipulation des éléments d'une liste simple. On va pour cela créer une nouvelle classe qui étend `SList`, de façon à bénéficier de ses méthodes et de la classe interne `Node`. Ajouter une méthode `main()` à votre classe et ajouter les instructions correspondant aux questions suivantes

1. Quelle est l'instruction pour créer une liste vide nommée `p` ?
2. Quelle est l'instruction pour créer une liste d'entiers `p` ne contenant qu'un seul élément (par exemple la valeur 1) ?
3. Quelles sont les instructions pour créer une liste d'entiers contenant exactement les valeurs 1, 2 et 3 dans cet ordre sans itération, ni appel récursif ?
4. Comment générer une liste L qui contient les entiers de 1 à 10 ?
5. Écrire trois méthodes qui retournent le 1er, le 3ème et le 7ème élément de la liste.

2 Barrière d'abstraction des listes chaînées

On s'intéresse ici à l'écriture de méthodes pour compléter la définition de l'interface `LList<T>` étudiée en cours.

1. écrire une méthode `isEmpty()` qui retourne `true` si et seulement si la liste est vide.
2. proposer maintenant une méthode `contains(T elem)` de recherche d'un élément dans une liste chaînée qui retourne `true` si `elem` appartient à la liste et `false` sinon.
3. définir une méthode `indexOf(T elem)` qui retourne l'indice auquel la première occurrence de la valeur `elem` est présente dans la liste.
4. proposer une méthode `pop()` qui enlève le premier élément de la liste et retourne sa valeur.
5. écrire une méthode `add(T elem, int index)` qui ajoute la valeur `elem` dans la liste à l'indice `index`. Si la liste est trop petite, l'élément est ajouté en fin de liste.
6. définir la méthode `dequeue()` qui supprime le dernier élément de la liste et retourne sa valeur.
7. écrire une méthode **itérative** `remove(int index)` qui supprime l'élément de la liste à l'indice `index`. Si la liste est vide ou sil 'indice est en dehors de la liste, celle-ci reste inchangée.

Solution :

```java
package td.td1;

import list.SList;
import list.Node;

public class TD1<T> extends SList<T> {

    public static void main(String[] args) {
        // Q1. Construction liste vide
        SList<Integer> p = new SList<>();

        // Q2. Un seul element
        p.add(1);
        System.out.println(p);

        // Q3. Trois valeurs sans boucle / recursivite
        p.add(2);
        p.add(3);
        System.out.println(p);

        // Q4. Liste de 1 a 10

        // solution potentiellement inefficace car ajout en
           fin
        // ici nous avons le pointeur last dans SList qui
           permet
        // d'etre en O(1) pour une insertion en fin
        SList<Integer> M = new SList<>();
        for (int i = 1; i <= 10; i++) {
            M.add(i);
        }
        System.out.println(M);

        // sinon solution efficace
        // dans tous les cas :
        // ajout en tete de liste
        TD1<Integer> N = new TD1<>();
        for (int i = 10; i >= 1; i--) {
            N.push(i);
        }
        System.out.println(N);
```

```
40
41          System.out.println(N.getFirst());
42          System.out.println(N.getThird());
43          System.out.println(N.getSeventh());
44      }
45
46
47      public void push(T elem) {
48          this.head = new Node<T>(elem, this.head);
49          size ++;
50      }
51
52      public T getFirst() {
53          if (head != null) return head.value;
54          else return null;
55      }
56
57      public T getThird() {
58          if (head != null && head.next != null && head.next.
               next != null)
59              return head.next.next.value;
60          else return null;
61      }
62
63      public T getSeventh() {
64          int i = 1;
65          Node<T> p = head;
66          while (p != null && i < 7) {
67              i++;
68              p = p.next;
69          }
70          return (p != null) ? p.value : null;
71      }
72
73      @Override
74      public boolean isEmpty() {
75          return head == null;
76      }
77
78      @Override
79      public boolean contains(T elem) {
80          Node<T> p = head;
81          while (p != null){
82              if (p.value.equals(elem)) return true;
```

```
83              else p = p.next;
84          }
85          return false;
86      }
87
88      @Override
89      public int indexOf(T elem) {
90          int index = 0;
91          Node<T> p = head;
92          while (p != null){
93              if (p.value.equals(elem)) return index;
94              else {
95                  p = p.next;
96                  index ++;
97              }
98          }
99          return -1;
100     }
101
102     public T pop(){
103         if (head == null) return null;
104         else {
105             T tmp = head.value;
106             head = head.next;
107             size --;
108             return tmp;
109         }
110     }
111
112     @Override
113     public void add(T elem, int index) {
114         Node<T> p = this.head;
115         Node<T> tmp = new Node<T>(elem);
116         if (index == 0) {
117             tmp.next = head;
118             head = tmp;
119         } else {
120             int i = 1;
121             while (p.next != null && i < index){
122                 p = p.next;
123                 i++;
124             }
125             tmp.next = p.next;
126             p.next = tmp;
```

```
127         }
128
129     }
130
131     public T dequeue(){
132         if (head == null) return null;
133         else if (head.next == null){
134             T tmp = head.value;
135             head = null;
136             size --;
137             return tmp;
138         } else {
139             Node<T> p = head;
140             while (p.next != last){
141                 p = p.next;
142             }
143             last = p;
144             size --;
145             T tmp = p.next.value; // last.value
146             p.next = null;
147             return tmp;
148         }
149     }
150
151     @Override
152     public void remove(int index) {
153         if (index < size){
154             if (index == 0) this.head = this.head.next;
155             else {
156                 int cpt = 0;
157                 Node<T> p = head;
158                 while (cpt < index - 1) {
159                     p = p.next;
160                     cpt++;
161                 }
162                 p.next = p.next.next;
163             }
164             size --;
165         }
166     }
167
168
169 }
```

java/TD1.java