Laboratoire 9 – Listes
ITI 1521. Introduction à l'informatique II

Semaine 29 Mars 2021

Dû en ligne après une semaine de votre Lab

/10

- ## Objectifs

•**Concevoir** une liste doublement chaînée possédant un noeud factice
• **Modifier** l'implémentation d'une liste chaînée afin d'y ajouter des méthodes.

1. Interface : `InterfaceNode`

Pour ce laboratoire, vous devez créer une implémentation de l'interface ***InterfaceNode*** à l'aide de noeuds doublement chaînés et d'un noeud factice. L'interface ***InterfaceNode*** **vous est fournie ci-joint** et déclare les méthodes suivantes :
• *int* ***size****() ;*
• *boolean* ***add****(Comparable obj) ;*
• *Object* ***get****(int pos) ;*
• *void* ***remove****(int pos).*

Les implémentations de cette interface doivent sauvegarder les éléments en ordre croissant. L'ordre des éléments est défini par l'implémentation de la méthode **compareTo(Object other)** que déclare l'interface **`Comparable`**. Les classes **`Integer`** et **`String`** implémentent toutes les deux cette interface et peuvent donc servir pour vos tests. Voir le lien suivant pour plus de détails.
https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html

Note: Juste pour simplifier, nous n'utiliserons pas le concept de type générique. Ainsi,
1) le compilateur produira quelques avertissements. Mais aussi,
2) les utilisateurs de la méthode *get* devront forcer une conversion de type puisque la valeur de retour est de type Object.

2. Liste : `ListNode`

Créez une implémentation de l'interface ***InterfaceNode***. L'implémentation nommée ***ListNode***, doit utiliser des noeuds doublement chaînés, ainsi qu'un noeud factice (Voir lecture 16). La classe doit posséder une référence ***head*** vers le noeud factice. De plus, l'implémentation doit lancer toutes les exceptions nécessaires. Voir ***InterfaceNode***.
Nous allons développer cette implémentation étape par étape. Nous avons d'abord implémenté les éléments de haut niveau (variables d'instance, classe imbriquée statique, et des coquilles pour chacune des méthodes). Cette implémentation vous est fournie ci-contre. Nous implémenterons le corps des méthodes, une à une.
Nous avons créé une définition initiale. Pour l'instant, nous ignorons l'implémentation des méthodes afin de nous concentrer sur les variables et la nécessité d'une classe « static »

imbriquée afin de représenter les noeuds de la liste. Toutes les méthodes de l'interface ***InterfaceNode*** sont déclarées. La classe contient donc une classe **Node**, les variables d'instance et des déclarations vides pour chacune des méthodes de l'interface.
Le compilateur n'acceptera pas des déclarations vides pour les méthodes retournant une valeur. Pour compiler la classe et travailler sur l'implémentation des méthodes une à une, **Nous avons** créé une définition initiale ne contenant que l'énoncé suivant,
throw new UnsupportedOperationException (" error ! ") ;

Toute tentative d'utiliser une méthode qui n'a pas encore d'implémentation sera signalée par une exception. Vous pouvez demander des explications supplémentaires à votre TA si nécessaire.
Une classe **Test** pour tester, **vous est fournie ci-joint**. Pour l'instant, elle ne contient qu'une méthode principale déclarant une variable de type ***ListNode***, désignant un objet ***ListNode***.
Assurez-vous que l'implémentation partielle comprenne tous les éléments ci-haut et qu'elle compile parfaitement. Lorsque ce sera fait, vous pouvez procéder à la première étape, l'implémentation de la méthode **size**.

```
/*** Classe ListNode ***/
public class ListNode implements InterfaceNode {
   // Implementation of the doubly linked nodes (nested-class)
   private static class Node {
          private Comparable value;

          private Node prev; //précédent
          private Node next; //suivant

          private Node ( Comparable value, Node prev, Node next ) {
          this.value = value;
           this.prev = prev;
          this.next = next;
          }
 }

 // Instance variables
   private Node head;

// Constructor: Empty list.
   public ListNode() {
    // a dummy node is created
      head = new Node(null, null,null);
      head.next = head;
      head.prev = head;
   }

   // Instance methods
   public int size() {
          throw new UnsupportedOperationException( "error!" );
   }
   public Object get( int pos ) {
          throw new UnsupportedOperationException( "error!" );
   }
```

```
    public boolean add( Comparable obj ) {
        throw new UnsupportedOperationException( "error!" );
    }

    public void remove( int pos ) {
        throw new UnsupportedOperationException( "error!" );
    }
}
```

Fichiers

- *InterfaceNode.java*
- *ListNode.java*
- *Test.java*

Question 1: (2 POINTS)

Implémenter la méthode *int* ***size()*** qui doit retourner le nombre d'éléments de la liste.
Utiliser une variable locale ***count***.
Une classe **Test1** pour tester, **vous est fournie ci-joint**.
(Vous pouvez ajouter des tests supplémentaires pour tester. Utiliser JUnit si vous le souhaitez.)

Question 2 : (2 POINTS)

Implémentez la méthode *boolean* ***add(*** *Comparable obj)*; elle ajoute un élément en ordre croissant selon l'ordre naturel imposé par la classe de l'objet (i.e. sa méthode **compareTo**).
Retourne **true** si l'élément a été ajouté à cette liste ordonnée, et **false** sinon.
Lever une exception de type *IllegalArgumentException* si le paramètre (obj) est null.
Considérer le cas spécial de liste vide.
Une classe **Test2** pour tester, **vous est fournie ci-joint**. **Test2** ajoute quelques tests afin de valider la méthode **add**. La liste s'accroît à mesure que des éléments sont ajoutés à la liste.

Question 3 : (2 POINTS)

Implémentez la méthode *Object* ***get(*** *int pos)*. Cette dernière retourne l'élément se trouvant à la position spécifiée (pos) de cette liste ordonnée ; le premier élément se trouve à la position 0. Cette opération ne doit pas transformer la liste ;
Lever une exception de type *IndexOutOfBoundsException(Integer.toString(pos))* chaque fois que c'est nécessaire (paramètre négatif ...).
Une classe **Test3** pour tester, **vous est fournie ci-joint**. **Test3** ajoute quelques tests afin de valider la méthode **get**.

Question 4 : (2 POINTS)

Implémentez la méthode *void* ***remove(*** *int pos)***;** Retire l'élément à la position spécifiée (pos) de cette liste ordonnée ; le premier élément se trouve à l'index 0.
Lever une exception de type *IndexOutOfBoundsException(Integer.toString(pos))* chaque fois que c'est nécessaire (paramètre négatif ...).

Une classe **Test4** pour tester, **vous est fournie ci-joint**. **Test4** joute quelques tests afin de valider la méthode ***remove***.

Question 5 : (2 POINTS)

Ajoutez la méthode d'instance *void* ***merge(ListNode other)*** qui ajoute tous les éléments de la liste **other** à cette liste tout en préservant l'ordre des éléments. Par exemple, si **list1** et **list2** sont deux listes ordonnées telles que **list1** contient "C", "E" et "H", et **list2** contient "A", "B", "C" et "F", alors l'appel **list1.merge(list2)** transforme **list1** de sorte qu'elle contienne maintenant les éléments suivants "A", "B", "C", "C", "E", "F" et "H" ; **list2** demeure inchangée par cet appel.

Cet exercice a pour but d'apprendre à traverser et transformer des listes doublement chaînées. Ainsi, vous ne devez pas utiliser des méthodes auxiliaires, en particulier, n'utilisez pas la méthode **add** !

Votre implémentation doit traverser les deux listes et insérer les éléments (valeurs) de l'autre liste dans celle-ci ; en ordre croissant.

Vous devez considérer les cas spéciaux et compléter les parties manquantes dans le code suivant.

```
//Both lists store their elements in increasing order, so both lists can be traversed simultaneously.
 public void merge( ListNode other ) {
   Node node = head.next;
   Node nodeNext= other.head.next;
   while(nodeNext!=other.head){
    if ( node == head ) {
        À COMPLÉTER
     }
     else if (nodeNext.value.compareTo(node.value) < 0){
       //insert before
       À COMPLÉTER
     }
     else if (node.next==head){
      //insert after
       À COMPLÉTER
     }
     else {
     À COMPLÉTER
     }
   }
 }
```

Une classe **Test5** pour tester **vous est fournie ci-joint**. **Test5** joute quelques tests afin de valider la méthode **merge**.

(Vous pouvez ajouter des tests supplémentaires pour tester chacune des méthodes de ***`ListNode`***. Utiliser JUnit si vous le souhaitez. Vous pouvez demander de l'aide à votre TA si nécessaire.)

Créer et soumettre un fichier zip comme d'habitude (Q1, ….Q5)

Fichier à soumettre pour chaque Question:

- ***ListNode.java***