Université De Montpellier/ Faculté Des Sciences

Structures de données — HMIN 215

Liste doublement chaînée

1 Objectif

Dans ces travaux dirigés et pratiques, nous étudions une implémentation des listes dans laquelle les éléments sont stockés dans des cellules chaînées entre elles. Cette implémentation, qui restera partielle, se rapproche de celle qui est réalisée en Java dans la classe `LinkedList<E>`. Pour la rendre plus réaliste, nous implémenterons (partiellement) l'interface `List<E>` qui définit le type liste de manière assez complète. Nous garderons pour les travaux dirigés seulement une partie de ces opérations, parmi les plus importantes pour leur utilisation.

Question 1

- *Créez dans Eclipse un projet dédié à cette implémentation.*
- *Créez un package puis une classe* `ListeDoublementChainee` *qui implémente* `List<E>`*.*
- *Cela vous oblige à implémenter les méthodes abstraites de* `List<E>`*.*
- *Dans les méthodes que vous n'implémenterez pas, le corps se réduira à l'instruction* `throw new UnsupportedOperationException();`
- *Alternativement, dérivez votre classe de* `AbstractList<E>`

2 Structure doublement chaînée

Dans cette mise en œuvre, les éléments de la liste (valeurs) sont stockés dans des cellules chaînées entre elles. Une cellule contient donc la référence d'un élément (valeur), ainsi que la référence vers la cellule qui la suit, et pour avoir un double chaînage, la référence vers la cellule qui la précède. (cf. Fig.1) Une liste contient une référence vers la première cellule (Début) et une référence vers la dernière cellule (Fin). Le plus souvent, on stocke également le nombre d'éléments (Taille).

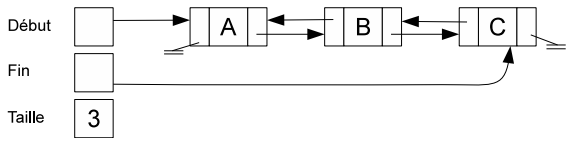

Figure 1 – Exemple de liste doublement chaînée

On donne la classe suivante qui représente les cellules. Notez qu'elles sont paramétrées par le type des valeurs qui y sont stockées (type formel `E`).

```
public class Cellule<E> {
    private Cellule<E> precedent;
    private E valeur;
    private Cellule<E> suivant;

    public Cellule() {}
    public Cellule(Cellule<E> prec, E valeur, Cellule<E> suiv){
        this.precedent = prec;
        this.valeur = valeur;
        this.suivant = suiv;
    }

    public Cellule<E> getPrecedent() {return precedent;}
    public void setPrecedent(Cellule<E> precedent) {this.precedent = precedent;}
    public E getValeur() {return valeur;}
    public void setValeur(E valeur) {this.valeur = valeur;}
    public Cellule<E> getSuivant() {return suivant;}
    public void setSuivant(Cellule<E> suivant) {this.suivant = suivant;}
    public String toString(){return ""+this.valeur;}
}
```

QUESTION 2

Implémentez une liste en testant au fur et à mesure les méthodes ajoutées.

1. *Ecrivez la classe* `ListeDoublementChainee<E>` *représentant les listes doublement chaînées. Cette classe possèdera trois attributs, (1) la référence vers la première cellule de la liste, (2) la référence vers la dernière cellule de la liste, (3) la taille de la liste. Cette classe devra posséder les méthodes :*
   (a) `add(E element)` *permettant l'ajout de* `element` *en fin de liste.*
   (b) `add(int index, E element)` *permettant l'ajout de* `element` *à l'indice* `index` *de la liste. Les éléments déjà présents, s'il y en a, sont décalés vers la droite.*
   (c) `String toString()` *permettant de visualiser les valeurs.*
   (d) `contains(E element)` *retournant vrai si* `element` *est présent dans liste, faux sinon.*
   (e) `get(int index)` *retournant la valeur de la cellule à l'indice* `index`*. La fonction devra retourner* `null` *si l'index est supérieur ou égal à la taille de la liste, ou s'il est strictement inférieur à 0.*
   (f) `isEmpty()` *retourne vrai si la liste est vide, faux sinon.*
   (g) `remove(int index)` *supprime la cellule située à l'indice* `index` *de la liste et retourne la valeur de l'élement supprimé.*
   (h) `remove(E element)` *supprime la cellule contenant la valeur* `element`*. Retourne vrai si un élement a été supprimé, faux sinon.*
   (i) `size()` *retourne un entier représentant la taille de la liste.*
2. *(optionnel) Complétez la définition de la liste par les autres méthodes de l'interface* `List<E>` *que vous comprenez.*

.