

Contrôle de Programmation Objet

Durée : 0h30

Aucun document autorisé

Nom :

Prénom :

1 Question 1 : Récursivité

On souhaite calculer la somme des éléments d'une liste de réels double de manière récursive et itérative.

- 1. Écrivez les fonctions `somme_réursive` et `somme_itérative` qui calculent la somme des n premiers éléments de la liste l . On pourra utiliser la méthode `get(i)` qui permet d'accéder au i^{e} élément d'une liste.

```
public static double somme_recursive(List<Double> l, int n) {
    //Écriture itérative : À COMPLÉTER
}
public static double somme_itérative(List<Double> l, int n) {
    //Écriture itérative : À COMPLÉTER
}
```

2 Question 2 : Pile

Dans le TD3, vous avez écrit une implémentation du type abstrait Pile à l'aide d'une liste de type `LinkedList`. Le type abstrait Pile classique suit le modèle LIFO (Last In, First Out).

Dans cet exercice, nous souhaitons implémenter une variante du type Pile. Cette variante se distingue par le fait qu'elle maintient une suite **ordonnée** d'entiers, le sommet de la pile étant **toujours** le plus petit des entiers de la pile. La méthode `peek` qui renvoie le sommet de pile, renvoie l'entier le plus petit. De même `pop` qui supprime le sommet de pile, supprime l'entier le plus petit contenu dans la pile.

Le code de `MaPile` donné ci-dessous implémente cette variante. Toutefois, il manque le code de la méthode `push` que vous devez écrire. Vous devez utiliser une `LinkedList` intermédiaire, et en plus des méthodes déjà données dans le code, vous devrez utiliser la méthode `add` de `LinkedList`.

```
public class MaPile {
    private LinkedList<Integer> maPile = new LinkedList<Integer>();

    public boolean empty() { return maPile.isEmpty(); }
}
```

```
public void pop() throws EmptyStackException {
    if (this.empty()) throw new EmptyStackException();
    maPile.remove(0);
}
public Integer peek() {
    if (this.empty()) throw new EmptyStackException();
    return maPile.get(0);
}
public void push(Integer e) {
    //À compléter page suivante
}
}
```

Ci-dessous, un exemple d'utilisation :

```
MaPile p = new MaPile();
p.push(671); p.push(9); p.push(82); p.push(671); p.push(6);
//p = [ 6 9 82 671 671 ]
p.pop(); p.pop();
//p = [ 82 671 671 ]
p.pop(); p.pop();
//p = [ 671 ]
```

- 2. Écrivez la méthode `push` :

```
public void push(Integer e) {
    //À COMPLÉTER
}
```

3 Question 3 : Liste

On souhaite écrire en Java les méthodes `concaténer` et `inverser` qui, respectivement, ajoute à la fin de la liste courante les éléments d'une seconde liste, et qui inverse l'ordre des éléments de la liste courante.

- 3. Dans la classe `MaListe`, donnée ci-dessous, complétez les méthodes `concaténer` et `inverser`. Les seules méthodes de `ArrayList` dont vous avez besoin, sont `size()`, `add(r,e)` et `set(r,e)`. Cette dernière, affecte un élément `e` au rang `r`, la taille de la liste reste inchangée.

```
import java.util.*;
public class MaListe<T> extends ArrayList<T> {

    /**
     * Rôle : concatène la liste l à la liste courante
     */
    void concaténer(MaListe<T> l) {
        //À COMPLÉTER
    }
}
```

```

    /**
     * Rôle : inverse ls éléments de la liste courante
     */
    void inverser() {
        //À COMPLÉTER
    }
}

```

La classe Test donne un exemple d'utilisation des deux méthodes.

```

class Test {
    public static void main(String [] args) {
        MaListe<Integer> l1 = new MaListe<Integer>();
        l1.add(1); l1.add(2); l1.add(3); l1.add(4); l1.add(5);

        MaListe<Integer> l2 = new MaListe<Integer>();
        l2.add(6); l2.add(7); l2.add(8);

        l1.concatener(l2);
        System.out.println(l1); //[1, 2, 3, 4, 5, 6, 7, 8]

        l1.inverser();
        System.out.println(l1); //[8, 7, 6, 5, 4, 3, 2, 1]
    }
}

```