Développement initiatique

Sujet 10 : Listes chaînées

Récupérer sur l'ENT les classes `Liste`, `Maillon` et `MainListe` et les compléter avec les algorithmes ci-dessous. La classe `Liste` implante les listes chaînées d'éléments de type entier.

1 Algorithmes de base sur les listes

Ecrire les fonctions suivantes. Pour chacune d'elles, vérifier bien que vous traitez tous les cas de figure : liste vide, liste à un seul élément, etc.

Question 1 Longueur d'une liste

```
/**  Résultat : le nombre d'éléments de la liste 'this'.
 */
public int longueur() {
```

Notez que cette méthode parcourt tous les maillons de la liste. Pour des raisons d'efficacité (et de pédagogie), sauf mention contraire, il sera donc interdit de faire appel à la fonction `longueur` pour répondre aux questions suivantes.

Question 2 Somme des éléments

```
/**  Résultat : la somme des éléments de la liste 'this'.
 */
public int somme() {
```

Question 3 Maximum des éléments

Déterminer la valeur maximum des éléments d'une liste.

Prérequis : la liste est non vide.

Question 4 Nombre d'occurrences

Déterminer le nombre d'occurrences d'un entier n dans une liste.

Question 5 Clones

Déterminer si deux listes sont clones l'une de l'autre, c'est-à-dire ont les mêmes valeurs rangées dans le même ordre.

Question 6 Longueur supérieure à k ?

```
/** Prérequis  : k est un entier positif ou nul
    Résultat   : vrai ssi la longueur de la liste 'this' est
                 supérieure ou égale à k.
    Contrainte : pensez à un parcours partiel (ex : si k=1)
 */
public boolean estSupK(int k) {
```

Question 7 Dernier élément

```
/**  Prérequis : 'this' est non vide.
     Résultat : le dernier maillon de la liste 'this'.
 */
private Maillon dernierMaillon() {...}
```

Ecrire la méthode publique `dernierElt` qui retourne le dernier élément (entier) d'une liste.

Question 8 Ajout en fin de liste

```
/**  Prérequis : Aucun !
     Action    : ajoute un élément de valeur n comme dernier élément de la
                 liste 'this'
 */
public void ajoutFin(int n) {
```

Vous pouvez éventuellement utiliser la méthode `dernierMaillon` pour écrire cette méthode.

Le premier bonus vous permettra d'écrire une version plus efficace de `ajoutFin`.

Question 9 Ajout en fin de liste *si absent*

```
/** Action : ajoute un élément de valeur n comme dernier élément de la liste 'this',
              seulement si la liste ne possède pas déjà un élément valant n.
    Contrainte : la méthode parcourt la liste au maximum une fois.
 */
public void ajoutFinSiAbsent (int n) {
```

Question 10 Extraction des éléments impairs

```
/** Résultat : une nouvelle liste contenant les éléments de valeur impaire
                de 'this', dans chacun des 2 cas suivants (deux versions) :
        (a) l'ordre des éléments de la liste retournée n'a pas d'importance,
        (b) l'ordre doit être le même que dans 'this'.
 */
public Liste extractionImpairs() {
```

Comparer les performances des deux versions.

Question 11 Troncation après le $k^{\text{ème}}$ élément

Lorsque la liste `this` est de longueur supérieure à k, la tronquer après son $k^{\text{ème}}$ élément.

Question 12 Suppression d'un élément

```
/** Pré-requis : aucun
    Action : supprime de la liste 'this' la première occurrence d'un entier n.
    Résultat : retourne vrai si l'élément n a été trouvé, faux sinon.
 */
public boolean supprOcc (int n) {
```

Question 13 Liste à l'envers

```
/** Pré-requis : aucun
    Résultat.  : une nouvelle liste contenant les éléments de 'this' dans l'ordre inverse.
 */
public Liste inverse() {
```

2 Exercices plus difficiles

Question 14 Liste à l'envers, récursive et en place (bonus difficile)

```
/** Pré-requis : aucun
    Action : inverse les éléments de 'this' en faisant appel à une fonction
       récursive "private Maillon inverseRec (Maillon m)".
 */
public void inverseRec() {...}

/** Action/résultat : sans créer de nouveau maillon, inverse la liste 'this' à
         partir du maillon m et renvoie le nouveau maillon de tête.
 */
private Maillon inverseRec (Maillon m) {
```

Pour les exercices suivants, il est fortement recommandé de bien *tester* vos algorithmes. Vous trouverez en commentaires de `MainListe.java` des exemples compliqués pour `suppToutesOcc` et `sousListe`.

Question 15 Suppression de toutes les occurrences d'une valeur

```
/** Action : supprime de la liste 'this' toutes les occurrences d'un entier n.
    Contrainte : une version simple à écrire mais coûteuse fait appel à la fonction supprOcc.
                 La version demandée ne parcourt la liste qu'une seule fois !
 */
public void suppToutesOcc(int n) {
```

Question 16 Sous-liste

```
/** Résultat : vrai ssi 'this' est une sous-liste (éléments consécutifs) de l.
    Exemples : cf. mainListe.java
    Stratégie possible : parcourt l et fait appel à chaque itération à une
                         fonction estPrefixe
 */
public boolean sousListe (Liste l) {...}

/** Résultat : vrai ssi la suite des éléments de 'this' est un préfixe de la liste l
        à partir du maillon m, c'est-à-dire est égale à la suite des premiers
        éléments de l commençant au maillon m.
    Exemple : (3,3,4) est un préfixe de la liste (1,6,3,3,4,2) à partir du 3ème maillon.
 */
private boolean estPrefixe (Maillon m) {
```

3 Bonus : des listes plus riches

Question 17 Créer *le plus vite possible* une classe `Liste2` (testée par `MainListe2`) qui a des spécifications très proches de `Liste`, mais qui contient deux autres attributs pour améliorer les performances en temps de calcul CPU :

- un attribut `longueur` de type entier qui stocke la longueur de la liste `this` ;
- un attribut `dernier` qui est une référence sur le dernier maillon de la liste `this`. Cet attribut permet d'améliorer les performances des méthodes travaillant en fin de liste.

4 Bonus : affichage de polygônes dans une fenêtre

Modifier (en copiant) la classe `Liste` en une classe `ListePoints` (sans reprendre toutes les méthodes) et avec un attribut supplémentaire permettant de référencer le dernier élément d'une liste. Copier aussi dans le même répertoire que `Liste.java` les fichiers du répertoire `graphisme` sous l'ENT.

Question 18 Créer une classe Polygone décrite par une liste de points (le dernier étant relié au premier par un segment) et une couleur (R,G,B).

Question 19 Ecrire un constructeur de `Polygone`, qui en plus dessine la figure géométrique correspondante sur la fenêtre.

Question 20 Ajouter une méthode `ajoutPoint(Point p, int k)` à la classe `ListePoints` qui ajoute un point p à la liste au rang k.

Ecrire une méthode `ajoutPoint(Point p, int k)` de `Polygone`.

Question 21 Ecrire une méthode `concatener` de `Polygone` prenant un polygône p en paramètre. La procédure remplace `this` par le polygône obtenu à partir des deux polygônes `this` et p en reliant, pour chacun d'eux, le dernier point de sa liste de points au premier point de l'autre.