SN/UCAD/ESP/DGI/DIC2_M1/APA/TDTP2

Exercice 1 : Listes chaînées

Une liste chaînée est définie récursivement comme suit :

- une liste *vide* est représentée par la valeur `null` ;
- une liste non vide est une paire constituée d'une *valeur* et d'une autre liste «*suite de la liste ».*

1. 1. Écrire une classe **`Liste`** représentant une liste non vide contenant :

   1) un attribut de type `String` représentant les valeurs stockées par la liste ;

   2) un attribut de type `Liste` représentant le reste de la liste (une variable de type `Liste` peut contenir la valeur `null`) ;

   3) un constructeur **`public Liste(String valeur)`** permettant de construire une liste contenant une unique valeur;

   4) un constructeur **`public Liste(String valeur, Liste suite)`** permettant de construire un objet liste à partir d'une valeur à stocker et du reste de la liste;

1. 2. On désire trier ces listes avec l'algorithme de tri rapide *quicksort*.

L'idée de cet algorithme récursif est la suivante :

- choisir une valeur `p` dans la liste ;
- découper la liste en deux sous-listes, l'une constituée des valeurs de la liste inférieures à `p` et l'autre de celles supérieures ;
- appeler récursivement le tri sur les deux sous-listes, si nécessaire (une liste vide ou à un élément est toujours triée) ;
- concaténer la liste triée des valeurs supérieures, la valeur `p` et la liste des valeurs inférieures.

a) 2.1. Ecrire la méthode **`public void append(Liste l)`** qui ajoute la liste passée en argument à la fin de la liste sur laquelle est appelée la méthode ;

b) 2.2. Ecrire la méthode **`public Liste suprimerInferieur()`** qui supprime itérativement de la liste sur laquelle est appelée la méthode les maillons dont la valeur est strictement inférieure à celle du premier maillon et renvoie une liste constituée des maillons supprimés.

c) 2.3. En utilisant la valeur du premier maillon de la liste chaînée comme valeur pour `p`, en déduire une méthode publique **`quicksort()`** qui effectue le tri rapide d'une liste et retourne la liste triée.

Exercice 3 : Grands entiers

La représentation machine des entiers ne permet pas de manipuler des valeurs entières trop grandes. Pour palier à cette limitation une solution consiste à représenter un entier par la liste de ses chiffres. Par convention le chiffre le plus significatif est en fin de liste et il est non nul, sauf pour la valeur zéro.

Ecrire une classe **`GrandInt`** permettant de manipuler des entiers représentés sous cette forme et comprenant les méthodes ci-dessous :

a. 1. Un constructeur **`private GrandInt(int chiffre, GrandInt suite)`** qui construit un grand entier dont la valeur est `chiffre+10*suite`.

b. 2. Un constructeur **`public GrandInt(int nombre)`** qui convertit le nombre passé en argument en un grand entier.

c. 3. Une méthode **`public String toString()`** qui retourne une chaîne de caractères représentant la liste des chiffres du grand entier sur lequel est appelée la méthode.

d. 4. Une méthode **`public int nombreDeChiffres()`** qui retourne le nombre de chiffres du grand entier sur lequel est appelée la méthode.

e. 5. Une méthode **`main`** pour tester ces différentes fonctions.

Lors de la définition de la classe **`GrandInt`** prendre le soin d'intégrer une documentation technique au moyen de l'outil ***JavaDoc***.

Exercice 4 : Représentation et Manipulation de Polynômes

Un polynôme (par exemple $2-3x^2+x^4$) peut être représenté par un tableau comme suit :

0	*1*	*2*	*3*	*4*
2	0	-3	0	1

1) En adoptant cette représentation, définir une classe nommée **Polynome** permettant de réaliser les opérations suivantes :

- · construction d'un polynôme à partir d'une notation vectorielle (par exemple 2, 0, -3, 0, 1) pour le polynôme précédent
- · addition de deux polynômes
- · affichage d'un polynôme
- · dérivée d'un polynôme
- · calcul de la valeur d'un polynôme pur un réel donné.

2) Ecrivez un programme principal :

a. · saisissant un polynôme P

b. · calculant et affichant $P + 2-3x^2+x^4$

c. · calculant et affichant la dérivée de P en 2

3) Construire une classe **Polynôme2** adaptée aux polynômes de degré 2 à partir de la classe **Polynome.** Cette classe doit posséder une méthode nommée **ResoudreEquation** cherchant les solutions de l'équation du second degré associée au polynôme.

Exercice 5 : Intervalles d'entiers

1. 1. Définissez la classe ***IntervOuvert*** (intervalle d'entiers ouvert) suivant les spécifications suivantes :

2. o attributs :

- - bornes inférieure et supérieure, appelées `bInf` et `bSup`. On

considère qu'il s'agit d'intervalles ouverts]`bInf,bSup[` (les bornes n'appartiennent pas à l'intervalle),

- - nombre d'éléments, `card`,

- o méthodes
  - - un constructeur,
  - - `estVide` indique si l'intervalle est vide (`bInf  bSup`),
  - - `estDans(n)` indique si l'entier *n* est dans l'intervalle,
  - - `contient(I)` indique si l'intervalle contient l'intervalle `I` passé en paramètre,

1. 2. Complétez la classe ***IntervOuvert*** avec une méthode qui, étant donnés deux intervalles, détermine leur intersection.
2. 3. Ecrivez la classe ***IntervFerme*** qui hérite de la classe ***IntervOuvert*** et définit la classe des intervalles fermés (c'est-à-dire, tels que les bornes inférieure et supérieure sont dans l'intervalle).

3. 4. Proposez un diagramme de classes UML pour représenter les classes ***IntervOuvert*** et ***IntervFerme.***

Exemple d'utilisation :

```
IntervOuvert I1 = new IntervOuvert(1,4); // définit ]1,4[={2,3}
IntervFerme I2 = new IntervFerme(1,4); // définit [1,4]={1,2,3,4}
```