UNIVERSITE DE LORRAINE, NANCY

TELECOM $1^{ère}$ année
Epreuve de Structures de Données (SD)
Date : samedi 25 mai 2013
Horaire : 10h à 12h

Durée du sujet : 1h50

Une feuille A4 autorisée
Calculatrices non autorisées

Consignes : les trois exercices seront rédigés sur des feuilles séparées. La clarté de la rédaction et la justification des réponses sont des éléments essentiels de l'appréciation. Le barème est donné à titre indicatif. Toute tricherie ou tentative de tricherie sera sévèrement sanctionnée.

Exercice 1 : Spécification algébrique et tests (7 points)

Introduction

L'objectif de cet exercice est de spécifier, implanter et tester un type abstrait algébrique. On considère les polynômes à une variable et à coefficient réel, en mathématiques un tel polynôme P s'écrit :

$P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ où n est un entier naturel et les a_i ($0 \leq i \leq n$) sont les coefficient réels. Le polynôme P est constitué des monômes $a_i x^i$ ($0 \leq i \leq n$), chaque monôme $a_i x^i$ étant défini par le réel a_i, son coefficient et par l'entier i, son degré.

On considère les types abstraits `Monome` et `Polynome` suivants :

```
Type Monome
Opérations
creerMonome   : Réel × Nat          → Monome   -- crée un monôme à partir d'un réel et d'un entier
coefficient   : Monome              → Réel     -- le coefficient du monôme
degreMonome   : Monome              → Nat      -- le degré du monôme
plusMonome    : Monome × Monome     → Monome   -- calcule la somme de deux monômes

Préconditions
creerMonome(r,n) est défini ssi r ≠ 0
plusMonome(m1,m2) est défini ssi degreMonome(m1) = degreMonome(m2)
                                 et coefficient(m1) + coefficient(m2) ≠ 0
```

Dans cette spécification `Nat` dénote le type des entiers naturels. La précondition de l'opération `creerMonome` signifie que l'on considère seulement les monômes dont les coefficients sont non nuls. L'opération `plusMonome` est définie si et seulement si les deux monômes ont le même degré et la somme de leur coefficient est non nulle (la somme de $a_i x^i$ et de $b_i x^i$ est alors $(a_i + b_i)x^i$ avec $a_i + b_i \neq 0$).

```
Type Polynome
Opérations
polynomeNul     :  → Polynome                          -- crée le polynôme nul
ajouterMonome   :  Polynome × Monome → Polynome        -- ajoute un monôme à un polynôme
monomePhd       :  Polynome → Monome                   -- fournit le mônome de plus haut degré
restePolynome   :  Polynome → Polynome                 -- supprime le mônome de plus haut degré
estNul          :  Polynome → Booléen                  -- teste si un polynôme est nul
degrePolynome   :  Polynome → Nat                      -- degré d'un polynôme
plusPolynome    :  Polynome × Polynome → Polynome      -- calcule la somme de deux polynômes

Préconditions
ajouterMonome(p,m) est défini ssi estNul(p) ou (degreMonome(m) > degrePolynome(p))
monomePhd(p) est défini ssi non(estNul(p))
restePolynome(p) est défini ssi non(estNul(p))
degrePolynome(p) est défini ssi non(estNul(p))
```

Le polynôme nul ne comporte aucun monôme. Le degré d'un polynôme non nul est le degré de son monôme de plus haut degré. Par exemple, le polynôme $1.5x^3+2x+5$ est construit par le terme suivant :

```
ajouterMonome(
  ajouterMonome(
    ajouterMonome(polynomeNul(),
                  creerMonome(5, 0)),
    creerMonome(2, 1)),
  creerMonome(1.5, 3))
```

Questions

1. (a) Complétez les axiomes suivants de la spécification `Monome` en donnant les termes correspondant à `R1`, `R2`, `R3`, `R4` :

   `coefficient(creerMonome(c,n)) = R1`

   `degreMonome(creerMonome(c,n)) = R2`

   `coefficient(plusMonome(m1,m2)) = R3`

   `degreMonome(plusMonome(m1,m2)) = R4`

   (b) Complétez les axiomes suivants de la spécification `Polynome` en donnant les termes correspondant à `S1`, `S2`, `S3`, `S4` :

   i. `monomePhd(plusPolynome(ajouterMonome(p1,m1) ajouterMonome(p2,m2))) = S1` si `degreMonome(m1) > degreMonome(m2)`

   ii. `monomePhd(plusPolynome(ajouterMonome(p1,m1),ajouterMonome(p2,m2))) = S2` si `degreMonome(m1) < degreMonome(m2)`

   iii. `monomePhd(plusPolynome(ajouterMonome(p1,m1),ajouterMonome(p2,m2))) = S3` si `degreMonome(m1) = degreMonome(m2)` et `coefficient(m1) + coefficient(m2)` $\neq 0$

   iv. `monomePhd(plusPolynome(ajouterMonome(p1,m1),ajouterMonome(p2,m2))) = S4` si `degreMonome(m1) = degreMonome(m2)` et `coefficient(m1) + coefficient(m2)` $= 0$ et `non(estNul(plusPolynome(p1,p2)))`

2. En Java, on implante le type `Monome` par la classe suivante :

```
package polynome;
public class Monome {
    private double coefficient;
    private int degre;

    public Monome(double c, int d){
        assert (c!=0) : "Monome : coefficient nul";
```

```
            coefficient = c;
            degre = d;}

    public double coefficient(){
        return coefficient;}

    public int degreMonome(){
        return degre;}

    public void plusMonome(Monome m){
        assert ((this.coefficient()+m.coefficient()!= 0) &&
                (this.degreMonome() == m.degreMonome())) :
               "plusMonome :  addition impossible'';
            this.coefficient += m.coefficient();}

}
```

Ecrivez le code Java permettant de tester l'axiome `degreMonome(plusMonome(m1,m2)) = R4` de la première question.

Exercice 2 : Graphes fortement connexes (6 points)

Définition

Un graphe orienté $G =< S, A >$ est fortement connexe si et seulement si pour tout couple $(u, v) \in S \times S$, il existe un chemin (c'est-à-dire une succession d'arcs) de u à v.

Pour déterminer si G est fortement connexe, on peut choisir un sommet u quelconque, dresser la liste des sommets accessibles à partir de u, la liste des sommets à partir desquels u est accessible, et vérifier que chacune de ces listes contient tous les sommets de G.

Questions

1. Écrivez en pseudo-code une fonction prenant en argument un graphe G et un sommet u, et renvoyant la liste des sommets accessibles dans G à partir de u. Le nom de votre fonction indiquera si le parcours se fait en profondeur ou en largeur.
2. Que faut-il modifier dans la fonction de la question précédente pour obtenir à la place la liste des sommets à partir desquels le sommet u est accessible ?
3. En utilisant les fonctions des questions précédentes, écrivez en pseudo-code une fonction `estFortementConnexe` prenant en argument un graphe orienté G et décidant s'il est fortement connexe. On supposera qu'on dispose d'une fonction `listeSommets` prenant en argument un graphe, et renvoyant la liste de ses sommets.

Rappel de la signature de MyGraph[T] :

```
Type MyGraph[T]
Opérations
empty       :    → MyGraph[T]                          -- crée un graphe vide
add         :  MyGraph[T] × T → MyGraph[T]             -- ajoute un sommet
add         :  MyGraph[T] × T × T → MyGraph[T]         -- ajoute un arc
remove      :  MyGraph[T] × T → MyGraph[T]             -- retire un sommet
remove      :  MyGraph[T] × T × T → MyGraph[T]         -- retire un arc
isEmpty     :  MyGraph[T] → Boolean                    -- graphe vide ?
isVertex    :  MyGraph[T] × T → Boolean                -- est un sommet ?
isArrow     :  MyGraph[T] × T × T → Boolean            -- est un arc ?
outDegree   :  MyGraph[T] × T → Integer                -- nombre d'arcs partant du sommet
inDegree    :  MyGraph[T] × T → Integer                -- nombre d'arcs arrivant au sommet
successor   :  MyGraph[T] × T × Integer → T            -- i-ème successeur d'un sommet
predecessor :  MyGraph[T] × T × Integer → T            -- i-ème prédécesseur d'un sommet
```

Exercice 3 : Shopping list (7 points)

Introduction

Une grande surface souhaite développer une application permettant à ses clients de gérer leur liste de courses. Elle fait appel à vous pour lui en proposer une implantation efficace.

Enchanté par ce défi, vous acceptez et proposez une première architecture en 4 classes pour stocker les produits de la grande surface :

- La classe *Produit* avec un nom et un prix
- La classe *CelluleProduit* avec un produit et la cellule suivante
- La classe *Rayon* avec un nom et la première cellule du rayon
- La classe *Magasin* composée d'un ensemble de rayons

Questions

Le stockage

1. A quelle structure de données correspondent les classes *Rayon*, *CelluleProduit*, et *Produit* ?
2. Implantez les fonctions */* A compléter */* de ces 3 classes de l'annexe A.

L'accès aux produits

La classe *Magasin* sera l'unique classe accessible aux clients. Vous décidez d'une implantation avec l'architecture de l'annexe B.

3. Proposez une structure de données pour stocker les différents rayons d'un magasin en détaillant ses avantages et inconvénients.
4. Implantez les fonctions */* A compléter */* de la classe *Magasin* de l'annexe B avec la structure de données choisie à la question précédente.
5. Quelles sont les complexités au pire cas des fonctions *prixTotalDuRayon* et *prixTotalDuMagasin* de la classe *Magasin*, avec la structure de données choisie à la question 3 ?

Annexe A

```
/**
 * Un rayon est une collection de produits.
 */
class Rayon
{
```

```
    private String nom;
    private CelluleProduit first;

    public Rayon( String nom ) { /* À compléter */ }

    public CelluleProduit premiereCellule()     { /* À compléter */ }
    public CelluleProduit derniereCellule()     { /* À compléter */ }

    public void ajouterProduit( Produit prod )  { /* À compléter */ }
    public void enleverProduit( String nom )    { /* À compléter */ }
    public Rayon triProduitsParPixDecroissant() { /* À compléter */ }
}

/**
 * La classe CelluleProduit permet de lier un produit au suivant
 */
class CelluleProduit
{
    private Produit value;
    private CelluleProduit suivant;

    public CelluleProduit( Produit produit, Produit suivant ) { /* À compléter */ }
    public CelluleProduit( Produit produit )                  { /* À compléter */ }

    public Produit value()                           { /* À compléter */ }
    public CelluleProduit suivant()                  { /* À compléter */ }
    public void setSuivant( CelluleProduit cellule ) { /* À compléter */ }
}

/**
 * Un produit possède un nom et un prix.
 */
class Produit
{
    private String nom;
    private double prix;

    public Produit( String nom, double prix )
    {
        this.nom = nom;
        this.prix = prix;
    }

    public String getNom()  { return this.nom; }
    public double getPrix() { return this.prix; }
    public void setNom( String nom )  { this.nom = nom; }
    public void setPrix( double prix ) { this.prix = prix }
}
```

Annexe B

```
/**
 * Un magasin est une collection de rayons.
 */
class Magasin
{
    /* Attribut(s) de la structure de données de rayons */
    /* À compléter */

    public Magasin() { /* À compléter */ }

    public Rayon rayon( String nomRayon )         { /* À compléter */ }
    public void ajouterRayon( String nomRayon )   { /* À compléter */ }
    public void supprimerRayon( String nomRayon ) { /* À compléter */ }

    public double prixTotalDuRayon( String nomRayon )   { /* À compléter */ }
    public double prixTotalDuMagasin()                  { /* À compléter */ }
    public double prixMinimumDuRayon( String nomRayon ) { /* À compléter */ }
    public double prixMinimumDuMagasin()                { /* À compléter */ }
}
```