le cnam

ING 39

Année universitaire 2015 – 2016
Examen 1re session : 13/11/2015
Responsable : Maria-Virginia APONTE

Durée : 2 heures

Tout document *papier* autorisé. Équipements électroniques, calculettes, interdits.

Les téléphones mobiles et autres équipements communicants doivent être éteints et rangés dans les sacs pendant toute la durée de l'épreuve.

Le barême est donné à titre indicatif ; il est susceptible de modifications.

Sujet de N pages, celle-ci comprise.

Exercice 1 *collections (4 points)*

La classe `Etudiant` fournie plus bas modélise les étudiants identifiés par un numéro (que l'on suppose unique) et possédant une liste de notes.

```
public class Etudiant {
    private int identifiant;          // identifiant de l'étudiant
    private String nom;               // nom de l'étudiant
    private ArrayList<Double> notes; // notes de l'étudiant

    public Etudiant(int id, String nomE){
      identifiant = id;
      nom = nomE;
      notes = new ArrayList<>();
    }
    public void add(double n){
        notes.add(n);
    }
    public void affiche(){
      // Supposée écrite, NE PAS écrire!
    }
    /**
     * Teste si toutes les notes de l'étudiant sont supérieures ou égales à n.
     * @param n note minimale.
     */
    public boolean notesSuperieuresA (double n){
      // A compléter
    }
}
```

Question 1.1

Complétez le code de la méthode `notesSuperieuresA(int n)` dans la classe `Etudiant`.

Question 1.2

Écrire la fonction suivante :

```
public static Set<Etudiant> filtreNoteSuperieureA(
        Map<Integer,Etudiant> m,
        double noteMin) {

}
```

Qui retourne l'ensemble des étudiants dont toutes les notes sont supérieures ou égales à `noteMin`.

Exercice 2 *collections et invariants (5 points)*

Dans cet exercice on veut modéliser les polynômes de manière efficace. On aura besoin de quelques définitions. Entre parenthèses on indique quelles méthodes cela concerne.

— un **terme** est composé d'un coefficient c et d'un exposant e. Exemple : $-4X^3$ est un terme de coefficient -4 et d'exposant 3.
— un **polynôme** est une somme de termes. Généralement on écrit cette somme dans l'ordre des exposants des termes.
— le **degré** (méthode `getDegre()`) d'un polynôme est la valeur du plus grand exposant parmi les termes de coefficients non nuls.

Par exemple, $1 + 2X + 6X^{100}$:
— est un polynôme avec 3 termes : $1X^0$, $2X^1$ et $6X^{100}$.
— Notez que l'on écrit 1 au lieu de $1X^0$, et $2X$ au lieu de $2X^1$ (car ils sont équivalents).
— son degré est 100.

L'**addition** (2 méthodes `add` demandées) de deux polynômes p1 et p2, se fait par addition de tous les coefficients des termes de même exposant. Ex : l'addition de $1+7X+3X^2+6X^{100}$ et de $4+5X^2-6X^{100}$ est égale à :

$$(1+4)+7X+(3+5)X^2+(6-6)X^{100} = 5+7X+8X^2+0X^{100} = 5+7X+8X^2$$

Notez que le polynôme a changé de dégré après addition (le nouveau degré est 2), car le terme d'exposant 100 a un coefficient nul. L'addition d'un polynôme et d'un terme se réalise de manière similaire. Par exemple : l'addition de $1 + 2X$ et du terme $-3X^5$ donne $1 + 2X - 3X^5$. L'addition de $1 + 2X$ avec le terme $4X$ donne $1 + 6X$.

Représentation efficace

Le polynôme $1 + 6X^{100}$ est équivalent au polynôme $1 + 0X^1 + 0X^2 + 0X^3 + \ldots + 0X^{99} + 6X^{100} + 0X^{101} + \ldots$. Par souci d'efficacité, on ne souhaite pas stocker les termes de coefficient nul. La classe `PolynomeCreux` fournie plus bas, modélise les polynômes dits « creux », où **l'on veut garantir qu'aucun terme de coefficient nul n'est stocké.** Les exposants sont toujours non négatifs.

La classe `PolynomeCreux`

Un polynôme est modélisé par une table d'associations où chaque entrée de la table est un terme du polynôme. Elle associe un exposant avec le coefficient d'un terme. Un constructeur fabrique un polynôme sans aucun terme stocké. Chaque méthode fournie ou demandée plus bas est décrite par son contrat. Le code des méthodes que vous écrirez devra respecter le contrat des méthodes et le contrat d'invariant spécifié.

Code fourni :

```
public class PolynomeCreux {
    // polynome composé d'associations <exposant, coefficient>
    private TreeMap<Integer, Integer> p;

     /**
     * Invariant:
     * il n'y a jamais dans p:
     * - un terme d'exposant négatif.
     * - un terme de coefficient nul.
     *  Donc, si (exp, coeff) est une entrée de p,
     * alors exp>=0 et coeff!=0.
     */
```

```
public PolynomeCreux(){
    p = new TreeMap<Integer,Integer> ();
}
 /**
  * Retourne le coefficient du terme d'exposant exp.
  * @throws ArrayIndexOutOfBoundsException si exp négatif
  */
public int getCoeff(int exp) throws ArrayIndexOutOfBoundsException {
    if (exp<0)
        throw new ArrayIndexOutOfBoundsException();
    Integer v = p.get(exp);
    if (v==null)
        return 0;
    else
        return v;
}
/**
 * Retourne le degré du polynôme: l'exposant le plus grand
 * de tous les termes de coefficients non nuls du polynôme.
 * Si tous les termes du polynôme sont nuls, retourne -1
 * @return
 */
public int getDegre() {
    if (p.isEmpty())
        return -1;
    else
        return p.lastKey(); // retourne la plus grande clef.
}


/**
 * Ajoute le terme (c,e) de coefficient c et d'exposant e
 * au polynôme courant. L'ajout se fait par addition.
 * @param c coefficient du terme à ajouter
 * @param e exposant du terme à ajouter
 * @throws ArrayIndexOutOfBoundsExceptio si e est négatif
 */
public void add(int c, int e){
  // A completer
}

/**
 * Addition du polynôme p1 avec le polynôme courant.
 * @param p1 polynôme à ajouter.
 * précondition p1 est différent du pointeur null.
 */
public void add(PolynomeCreux p1){
// A completer
}
```

Question 2.1

En supposant que toutes les méthodes sont correctement écrites, donnez les affichages attendus pour ce code :

```
PolynomeCreux p1 = new PolynomeCreux();
System.out.println(p1.getDegre());
System.out.println(p1.getCoeff(7));
p1.add(5,1);
p1.add(10,2);
p1.add(6,0);
System.out.println(p1.getDegre());
p1.add(2,7);
p1.add(3,7);
System.out.println(p1.getCoeff(7));
System.out.println(p1.getDegre());
p1.add(-5,7);
System.out.println(p1.getDegre());
```

Question 2.2 *(méthode `add(int c, int e)`)*

La méthode `add(int c, int e)` doit ajouter un terme de coefficient c et exposant e au polynôme courant. Son contrat est incomplet : certains comportéments souhaités ne sont pas décrits. On vous propose les tests JUNIT4 suivants avec exemples de comportements attendus. Complétez le code de cette méthode pour satisfaire son contrat et se comporter comme décrit dans ces tests.

```
@Test
public void testAdd_int_int1() {
    PolynomeCreux p = new PolynomeCreux();
    p.add(0, 0); // p = 0X^0
    assertEquals(-1, p.getDegre()); // si, si, c'est bien ça, lisez l'énoncé.
}
@Test
public void testAdd_int_int2() {
    PolynomeCreux p = new PolynomeCreux();
    p.add(1, 0); // p = 1X^0
    assertEquals(1, p.getCoeff(0));
    assertEquals(0, p.getDegre());
}
@Test
public void testAdd_int_int3() {
    PolynomeCreux p = new PolynomeCreux();
    p.add(1, 3); // terme c=1, e=3
    p.add(4, 3); // terme c=4, e=3
    // p = 5X^3
    assertEquals(5, p.getCoeff(3));
}
@Test
public void testAdd_int_int4() {
    PolynomeCreux p = new PolynomeCreux();
    p.add(4, 3);  // p = 4X^3
```

```
        p.add(-4, 3); // p = 0X^0
        // (p n'a plus aucun terme)
        assertEquals(-1, p.getDegre());
    }
```

Question 2.3 *(méthode `add(PolynomeCreux p1)`)*

Complétez le code de la méthode `add(PolynomeCreux p1)` qui ajoute le polynôme p1 au polynôme courant.

Exercice 3 *3 points*

Écrire la fonction suivante :

```
/**
 * Compte le nombre de chiffres (entre 0 et 9) compris dans le fichier
 * texte passé en paramètre.
*/
public static int compterChiffre(File f) throws IOException {
}
```

Exercice 4 *5 points*

Un fichier texte contient un carnet de contacts, donné au format suivant :

- une ligne avec le nom et le prénom de la personne ;
- une ligne avec son adresse ;
- ensuite, les numéros de la personne, à raison d'un par ligne.
- on termine les informations sur la personne par une ligne vide.

Par exemple, on aura :

```
Alan Turing
Cambridge
003434341
003443455

Ada Lovelace
Londres
02343
34404
```

Écrire le code de la fonction suivante, qui prend comme argument un numéro de téléphone (donné sous forme de chaîne de caractères), et retourne le *nom* de la personne correspondante.

Si jamais plusieurs personnes ont le même numéro, on retournera le nom de la première qu'on trouve.

```
public static String nomPourTelephone(File carnet, String numero) {
// ...
}
```

Exercice 5 *3 points*

J'ai écrit le code suivant pour pouvoir disposer d'un tableau éditable 2x2 de double utilisable avec une JTable.

```
class DoubleTable extends AbstractTableModel {
    private Double content[][]= new Double[2][2];

    public DoubleTable() {
        for (int i=0; i < 2; i++) {
            for (int j= 0; j < 2; j++)
                content[i][j]= new Double(0.0);
        }
    }

    @Override
    public Class getColumnClass(int columnIndex) {
        return Double.class;
    }

    @Override
    public Object getValueAt(int ligne, int colonne) {
        return content[ligne][colonne];
    }

    @Override
    public void setValueAt(Object v, int ligne, int colonne) {
        content[ligne][colonne]= (Double) v;
    }

    @Override
    public int getRowCount() { return 2;}

    @Override
    public int getColumnCount() { return 2;}

    @Override
    public boolean isCellEditable(int i, int j) {return true;}
}
```

Je le teste avec un petit programme simple, et tout a l'air de fonctionner. Je poursuis en ajoutant un bouton qui exécute le code suivant pour "remettre à zéro" le modèle que j'ai créé :

```
    public void effacer() {
            for (int i= 0; i < 2; i++)
                for (int j= 0; j < 2; j++)
                    model.setValueAt(new Double(0.0), i, j);
    }
```

ça ne fonctionne pas bien.

— pourquoi ? quel est le comportement que j'ai ?

— qu'est-ce qui manque et où ? (proposer une correction du code)