IUT de Paris Descartes – Base de la Programmation Objet

Travaux Pratiques – Sujet n°5

Objectifs

- Définir des constructeurs
- Permettre la copie d'objets
- Définir des opérations
- Réaliser des entrées/sorties

Manipulation de polynômes

Nous voulons mettre en place une classe permettant la représentation de polynômes à coefficients réels ayant un degré maximal égal à 8. Une des principales opérations à réaliser est la somme de deux polynômes.

1. Sachant que nous disposerons de l'addition de polynômes, donnez le prototype du ou des constructeurs qui permettront un usage simple de la classe.

Solution: Voici trois constructeurs possibles :

```
public class Polynôme {
  // construit le polynôme nul
  public Polynôme() {
    ...
  }

  // construit le monôme v * x^c
  public Polynôme(double v, int c) {
    ...
  }

  // construit le polynôme constant v
  public Polynôme(double v) {
    ...
  }
}
```

2. Donnez le prototype des méthodes permettant d'évaluer un polynôme pour une valeur donnée (exemple : le polynôme 2x + 3 est évalué à 7 pour la valeur 2), de calculer la somme de deux polynômes et de calculer la dérivée d'un polynôme.

Solution:

```
public class Polynôme {
  ...
```

```
  public double valeur(double x) {
    ...
  }

  public Polynôme plus(Polynôme p) {
    ...
  }

  public Polynôme dérivée() {
    ...
  }
  ...
}
```

Notez bien que la méthode réalisant la somme ne prend qu'un paramètre et qu'elle réalise la somme entre **l'instance courante** et celui-ci. Voici l'extrait d'un programme utilisant la somme :

```
Polynôme a, b, c;
a = new Polynôme(3., 2); b = new Polynôme(1.);
c = a.plus(b) ;
```

3. Choisissez comment stocker les données d'un polynôme et programmez les méthodes définies ci-dessus.

Solution: J'ai choisi un tableau de réels. Chaque réel du tableau représente un coefficient du polynôme et son indice indique le degré de ce coefficient. La taille du tableau est définie par une constante. La définition de la classe est la suivante :

```
public class Polynôme {
  private static final int MAX_COEF = 8;
  private double[] coefficients;

  // construit le polynôme nul
  public Polynôme() {
    coefficients = new double[MAX_COEF + 1];
    Arrays.fill(coefficients, 0);
  }

  // construit le monôme v * x^c
  public Polynôme(double v, int c) {
    this();
    coefficients[c] = v;
  }

  // construit le polynôme constant v
  public Polynôme(double v) {
    this(v, 0);
  }

  public double valeur(double x) {
    double r = 0, p = 1;
    for (int i = 0; i <= MAX_COEF; ++i) {
      r += coefficients[i] * p;
      p *= x;
```

```
    }
    return r;
  }

  public Polynôme plus(Polynôme p) {
    Polynôme r = new Polynôme();
    for (int i = 0; i <= MAX_COEF; ++i)
      r.coefficients[i] = coefficients[i] + p.coefficients[i];
    return r;
  }

  public Polynôme dérivée() {
    Polynôme r = new Polynôme();
    for (int i = 1; i <= MAX_COEF; ++i)
      r.coefficients[i - 1] = i * coefficients[i];
    return r;
  }
}
```

4. Complétez la classe de manière à autoriser la copie d'un polynôme.

Solution:

```
public class Polynôme {
  ...
  // construit une copie d'un polynôme p
  public Polynôme(Polynôme p) {
    coefficients = p.coefficients.clone();
  }

  // retourne une copie du Polynôme
  public Polynôme clone() {
    return new Polynôme(this);
  }
}
```

5. Complétez votre classe d'une méthode `toString` et réalisez un court programme illustrant l'usage de la classe.

Solution:

```
public class Polynôme {
  ...
  // Une version plus simple est aussi donnée dans les sources
  public String toString() {
    StringBuilder sb = new StringBuilder();
    boolean first = true;
    for (int i = MAX_COEF; i >= 0; --i) {
      if (coefficients[i] != 0) {
        String prefixe = "";
        if (first)
          prefixe = Double.toString(coefficients[i]);
        else if (coefficients[i] < 0)
```

```
          prefixe = "␣-␣" + (-coefficients[i]);
        else
          prefixe = "␣+␣" + coefficients[i];
        first = false;
        if (i > 1)
          sb.append(prefixe + "*x^" + i);
        else if (i == 1)
          sb.append(prefixe + "*x");
        else
          sb.append(prefixe);
      }
    }
    if (first)
      sb.append("0");
    return sb.toString();
  }
}

public class Appli {
  public static void main(String[] args) {
    Polynôme p1, p2;
    p1 = new Polynôme(8., 2);
    p2 = new Polynôme(5., 3);
    System.out.println(p1);
    System.out.println(p2);
    System.out.println(p1.plus(p2));
    System.out.println(p1.plus(p2).dérivée());
    System.out.println(p1.plus(p2).dérivée().valeur(5));
    System.out.println(p1.valeur(p1.plus(p2).valeur(5)));
    System.out.println(p1.dérivée().dérivée().valeur(5));
  }
}
```

6. Proposez une structuration des données permettant la manipulation de polynômes de degré quelconque (i.e. non limité à 8).

Solution: On se limite à des polynômes tels que l'ensemble des degrés ayant un coefficient associé qui soit non nul est un ensemble fini. Le principe consiste alors à ne mémoriser que les couples (degré, coefficient) tels que le coefficient soit non nul. Une solution consiste à employer un tableau associatif qui fait correspondre à chaque degré (la clé – un entier naturel) son coefficient non nul (la valeur associée – un réel). L'absence d'un degré dans le tableau associatif indique que le coefficient qui lui correspond est nul.

Les classes `java.util.HashMap` et `java.util.TreeMap` sont les tableaux associatifs de la bibliothèque standard. Nous verrons bientôt comment employer ses structures de donnée.