Université Bordeaux I
U.F.R. Mathématiques et Informatique

IUP MIAGe 3^{eme} année
Année 2001-2002

Vendredi 18 janvier 2002, 14 heures - 16 heures
Tous documents autorisés.

Programmation par Objets
Alain Griffault

SUJET + CORRIGE

Le sujet est court, seules les réponses très précises seront prises en compte.

1 Typage et polymorphisme (10 points)

Soit la déclaration de la classe `Point` suivante :

```
class Point{
    int x, y;
    Point(int px, int py){
        x = px;
        y = py;
    }
    public double distanceOrigine(){
        return Math.sqrt(x*x+y*y);
    }
    public double distance(Point p){
        return Math.sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y));
    }
    public void deplacer(int dx, int dy){
        x += dx;
        y += dy;
    }
}
```

Question 1.1 (1 point) *Dites pour chacune des quatre méthodes de la sous-classe* `PointTopographique` *dans le cas où elle vous semble correcte : si c'est une définition, une redéfinition, une surcharge ; et dans le cas contraire si elle est refusée à la compilation, ou bien provoque une erreur à l'exécution.*

```
class PointTopographique extends Point{
    int alt;
    PointTopographique(int px, int py, int a){
        super(px,py);
        alt = a;
    }
    public int altitude(){
        return alt;
    }
    public double distanceOrigine(){
        return Math.sqrt(x*x+y*y+alt*alt);
    }
    public void deplacer(int da){
        alt += da;
    }
    public void deplacer(int dx, int dy, int da){
        deplacer(dx,dy);
        deplacer(da);
    }
}
```

Réponse à la question 1.1:

nouvelle méthode : définition
code différent : redéfinition
nouvelle signature : surcharge
nouvelle signature : surcharge

Question 1.2 (2 points) *Dites pour chacune des quatre lignes de la méthode* `main` *de la classe* `Test1` *si elle est correcte, refusée à la compilation, ou bien provoque une erreur à l'exécution.*

```
public class Test1{
    public static void main(String[] args){
        PointTopographique pt1 = new PointTopographique(1,2,3);
        PointTopographique pt2 = new Point(1,2);
        Point p1 = new PointTopographique(1,2,3);
        Point p2 = new Point(1,2);
    }
}
```

Réponse à la question 1.2:

correcte
refusée compilation
correcte
correcte

Question 1.3 (3 points) *Dites pour chacune des* **douze** *dernières lignes de la méthode* `main` *de la classe* `Test2` *si elle est correcte, refusée à la compilation, ou bien provoque une erreur à l'exécution.*

```
public class Test2{
    public static void main(String[] args){
        PointTopographique pt = new PointTopographique(1,2,3);
        Point p = new Point(1,2);
        p.distanceOrigine();
        p.deplacer(2,3);
        p.deplacer(2,3,4);
        pt.distanceOrigine();
        pt.deplacer(2,3);
        pt.deplacer(2,3,4);
        int hpt = pt.altitude();
        p = pt;
        p.distanceOrigine();
        p.deplacer(2,3);
        p.deplacer(2,3,4);
        int hp = p.altitude();
    }
}
```

Réponse à la question 1.3:

correcte
correcte
refusée compilation
correcte
correcte
correcte
correcte
correcte
correcte
correcte
refusée compilation
refusée compilation

Question 1.4 (2 points) *Donnez d'une part les ajouts à faire aux classes* `Point` *et* `PointTopologique`*; d'autre part une méthode* `main` *d'une classe* `Test` *qui provoquent le message d'erreur suivant à la compilation :*

```
> javac Point.java PointTopographique.java Test.java
Test.java:5: reference to distance is ambiguous,
both method distance(PointTopographique) in Point
and method distance(Point) in PointTopographique match
pt1.distance(pt1);
          ^
1 error
```

Réponse à la question 1.4:
Comme le suggère le message d'erreur, il faut une méthode `distance(PointTopographique)` dans la classe `Point`.

```
class Point{
    int x, y;
    Point(int px, int py){
        x = px;
        y = py;
    }
    public double distanceOrigine(){
        return Math.sqrt(x*x+y*y);
    }
    public double distance(Point p){
        return Math.sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y));
    }
    public void deplacer(int dx, int dy){
        x += dx;
```

```
        y += dy;
    }
    public double distance(PointTopographique pt){
        return Math.sqrt((x-pt.x)*(x-pt.x)+(y-pt.y)*(y-pt.y)+pt.alt*pt.alt);
    }
}
```

De même, il il faut une méthode `distance(Point)` dans la classe `PointTopographique`.

```
class PointTopographique extends Point{
    int alt;
    PointTopographique(int px, int py, int a){
        super(px,py);
        alt = a;
    }
    public int altitude(){
        return alt;
    }
    public double distanceOrigine(){
        return Math.sqrt(x*x+y*y+alt*alt);
    }
    public void deplacer(int da){
        alt += da;
    }
    public void deplacer(int dx, int dy, int da){
        deplacer(dx,dy);
        deplacer(da);
    }
    public double distance(Point p){
        return Math.sqrt((x-p.x)*(x-p.x)+(y-p.y)*(y-p.y)+alt*alt);
    }
}
```

et un appel ambiguë `pt1.distance(pt1);` dans la classe `Test`.

```
public class Test{
    public static void main(String[] args){
        PointTopographique pt1 = new PointTopographique(1,2,3);
        pt1.distance(pt1);
    }
}
```

Question 1.5 (2 points) *Donnez les trois instructions possibles qui calculent des distances entre* `pt1` *et lui-même.*

Réponse à la question 1.5:
Pour lever l'ambiguïté, il faut préciser dans quelle(s) situation(s) l'objet `pt1` doit être forcé par un *cast* comme un `Point` et non un `PointTopographique`.

```
public class Test1{
    public static void main(String[] args){
        PointTopographique pt1 = new PointTopographique(1,2,3);
        pt1.distance((Point)pt1);
        ((Point)pt1).distance(pt1);
        ((Point)pt1).distance((Point)pt1);
    }
}
```

2 Modèles de conception (10 points)

Soit la classe `RandomPolynomial` qui représente un polynôme aléatoire de degré deux. Elle permet de créer, de modifier, d'afficher un polynôme, et de retourner sa valeur pour un argument.

```
import java.util.Random;
public class RandomPolynomial{
    /* le tableau coeff stocke les coeffs d'un polynome de degree 2 */
    private double coeff[]=new double[3];
    /* random - generateur de nombres aleatoires */
    private static Random random;

    public RandomPolynomial(){
        if(random==null)
            random=new Random();
        initializePolynomial();
    }

    private void initializePolynomial(){
        random.setSeed(System.currentTimeMillis());
        for(int i=0;i<coeff.length;i++)
            coeff[i]=random.nextDouble();
    }

    /* changer le polynome */
    public void changeState(){
        initializePolynomial();
    }

    /* calculer la valeur du polynome pour l'argument value */
    public double value(double value){
        double result=0;
        for(int i=coeff.length-1;i>=0;i--)
            result=result*value+coeff[i];
        return result;
    }

    public String toString(){
      String s="[";
      for(int i=0;i<coeff.length;++i)
        s+= Double.toString(coeff[i]) + ( (i<coeff.length-1) ? ", " : "]");
      return s;

    }
}
```

La classe `Test` est un client typique de la classe `RandomPolynomial`.

```
public class Test{
    public static void writeValue(RandomPolynomial polynom, double x){
        System.out.println("polynom.value(" + x + ") = " + polynom.value(x));
    }

    public static void main(String[] args){
        RandomPolynomial  polynome=new RandomPolynomial();
        System.out.println(polynome);
        writeValue(polynome, 1.0);
        writeValue(polynome, 0.0);
    }
}
```

Un résultat possible est :

```
[0.29554777921195297, 0.6839832081936477, 0.9900763108803116]
polynom.value(1.0) = 1.969607298285912
polynom.value(0.0) = 0.29554777921195297
```

Le Client désire maintenant manipuler plusieurs polynômes, et surtout pouvoir réutiliser un polynôme. Il souhaite donc sauvegarder un polynôme. Le créateur de la classe ne souhaitant pas divulguer le codage, il va proposer au client un mécanisme de sauvegarde préservant l'encapsulation des données.

Le modèle **Memento** permet : sans violation de l'encapsulation, de saisir et de transmettre à l'extérieur d'un objet, l'état interne de celui-ci, dans le but de pouvoir ultérieurement le restaurer dans cet état.

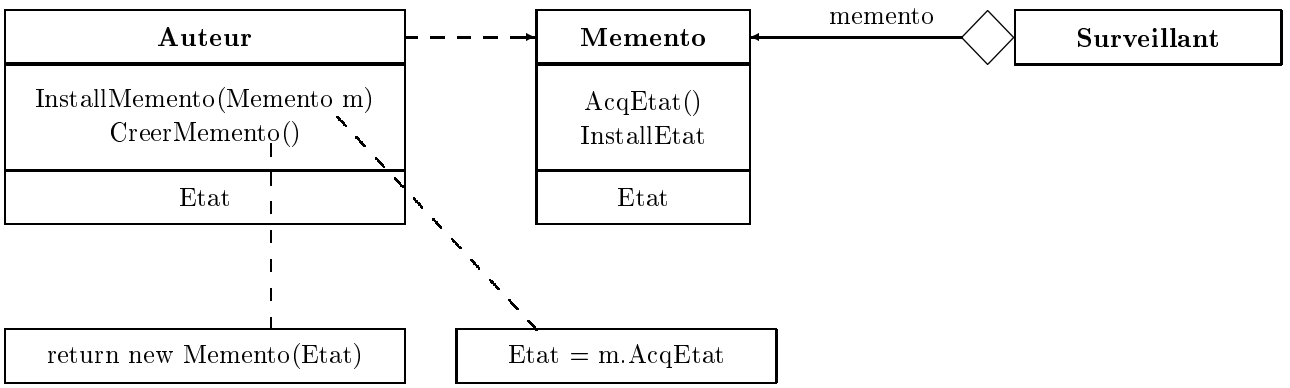

FIG. 1 – La structure du modèle Memento

Suite à cette demande, le créateur de la classe fournit au client l'interface `PolynomialMemento`

```
public interface PolynomialMemento{ //interface vide, ne fait que définir le type
}
```

et un nouveau programme de test.

```
public class Test{
    RandomPolynomial  polynome;
    PolynomialMemento memento;

    public Test(){
        polynome = new RandomPolynomial();
    }

    public void changePoly(){
        polynome.changeState();
    }

    public void save(){
        memento = polynome.createPolynomialMemento();
    }

    public void restore(){
        if (memento == null)
            System.out.println("restauration impossible");
        else
            polynome.setMemento(memento);
    }
```

```
    public void writeTest(){
        System.out.println(polynome);
        writeValue(polynome, 1.0);
        writeValue(polynome, 0.0);
    }

    public static void writeValue(RandomPolynomial polynom, double x){
        System.out.println("polynom.value(" + x + ") = " + polynom.value(x));
    }

    public static void main(String[] args){
        Test test = new Test();
        test.writeTest();
        test.save();
        test.changePoly();
        test.writeTest();
        test.restore();
        test.writeTest();
    }
}
```

Le créateur a donc réussit à cacher *à l'intérieur* de la classe `RandomPolynomial` une classe **Memento**.

Question 2.1 (3 points) *En faisant très attention aux mots clefs (private, public ...), donnez le code complet (attributs, constructeur et méthodes) de la classe* `MyPolynomialMemento`*, classe interne à la classe* `RandomPolynomial`*.*

Réponse à la question 2.1: avec la réponse de la question suivante.

Question 2.2 (1 point) *Donnez le code de la méthode* `createPolynomilaMemento()`*; et dites à quelle classe elle appartient ?*

Réponse à la question 2.2: avec la réponse de la question suivante.

Question 2.3 (1 point) *Donnez le code de la méthode* `setMemento(PolynomialMemento memento)`*; et dites à quelle classe elle appartient ?*

Réponse à la question 2.3:

```
import java.util.Random;

public class RandomPolynomial{

    /* le tableau coeff stocke les coeffs d'un polynome de degree 2 */
    private double coeff[]=new double[3];

    /* random - generateur de nombres aleatoires */
    private static Random random;

    public RandomPolynomial(){
        if(random==null)
            random=new Random();
        initializePolynomial();
    }

    private void initializePolynomial(){
        random.setSeed(System.currentTimeMillis());
        for(int i=0;i<coeff.length;i++)
    coeff[i]=random.nextDouble();
    }

    /* changer le polynome */
    public void changeState(){
```

```
        initializePolynomial();
    }

    /* calculer la valeur du polynome pour l'argument value */
    public double value(double value){
        double result=0;
        for(int i=coeff.length-1;i>=0;i--)
            result=result*value+coeff[i];
        return result;
    }

    private class MyPolynomialMemento implements PolynomialMemento{
        private double oldCoeff[];
        public MyPolynomialMemento(){
            oldCoeff=new double[coeff.length];
            //recopier le tableau de coeff
            System.arraycopy(coeff,0,oldCoeff,0,3);
        }
        public void undo(RandomPolynomial p){
            System.arraycopy(oldCoeff,0,p.coeff,0,3);
        }
    }

    public PolynomialMemento createPolynomialMemento(){
        return new MyPolynomialMemento();
    }

    public void setMemento(PolynomialMemento memento) {
        ((MyPolynomialMemento)memento).undo(RandomPolynomial.this);
    }

    public String toString(){
        String s="[";
        for(int i=0;i<coeff.length;++i)
            s+= Double.toString(coeff[i]) + ( (i<coeff.length-1) ? ", " : "]");
        return s;
    }
}
```

Fort du programme `Test` livré par l'auteur, le client décide d'utiliser le memento pour implanter un *undo*. Pour cela, il décide d'utiliser la classe `Stack` qui représente une pile (FIFO) dont voici un bref (mais suffisant) descriptif.

Stack() construit une pile vide.

boolean empty() teste si une pile est vide.

Object pop() enlève l'objet situé en haut de pile, pour le retourner comme résultat de la méthode.

Object push(Object item) place l'objet item en haut de la pile.

Question 2.4 (5 points) *Donnez le code du nouveau programme* `Test`.

Réponse à la question 2.4:

```
import java.util.*;
public class Test{
    Stack memento;
    RandomPolynomial polynome;

    public Test(){
```

```
        memento = new Stack();
        polynome=new RandomPolynomial();
    }

    public void changePoly(){
        memento.push(polynome.createPolynomialMemento());
        polynome.changeState();
    }

    public void undo(){
        if (memento.empty())
            System.out.println("undo impossible");
        else
            polynome.setMemento((PolynomialMemento)memento.pop());
    }

    public void writeTest(){
        System.out.println(polynome);
        writeValue(polynome, 1.0);
        writeValue(polynome, 0.0);
    }

    public static void writeValue(RandomPolynomial polynom, double x){
        System.out.println("polynom.value(" + x + ") = " + polynom.value(x));
    }

    public static void main(String[] args){
        Test test = new Test();
        test.writeTest();
        test.changePoly();
        test.writeTest();
        test.changePoly();
        test.writeTest();
        test.undo();
        test.writeTest();
        test.changePoly();
        test.writeTest();
        test.undo();
        test.writeTest();
        test.undo();
        test.writeTest();
        test.undo();
        test.writeTest();
    }
}
```