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

IUP MIAGe 3^{eme} année
Année 1999-2000

Vendredi 21 janvier, 8 heures - 10 heures
Tous documents autorisés.

Programmation par Objets
Alain Griffault

SUJET + CORRIGE

1 Héritage et/ou Composition (10 points)

Les sources suivants permettent le calcul du périmètre d'une figure fermée décrite par une suite de points.

```
public class Point {
    private int abs;
    private int ord;
    public Point(int x, int y) {abs = x;ord = y;}
    public int abscisse(){return abs;}
    public int ordonnee(){return ord;}
    public double distance(Point p) {
        return Math.sqrt((abs-p.abscisse())*(abs-p.abscisse()) +
                         (ord-p.ordonnee())*(ord-p.ordonnee()));
    }
}
```

```
public interface Figure {
    public double perimetre();
}
```

```
public class FigureNPoints implements Figure{
    Point[] liste;
    public FigureNPoints(Point[] l) {liste = l;}
    public FigureNPoints(Point[] l, int nb) {
        liste = new Point[nb];
        for (int i=0; i< nb; i++)
            liste[i]=l[i];
    }
    public double perimetre(){
        double res=0;
        for (int i=0; i < liste.length; i++)
            res += liste[i].distance(liste[(i+1)%liste.length]);
        return res;
    }
}
```

Pour diverses raisons, nous souhaitons disposer de classes pour des figures connues. Par exemple un triangle sera défini par les trois premiers points d'une suite quelconque de points.

```
public class TriangleA implements Figure {
    FigureNPoints fig;
    public TriangleA(Point[] l){fig = new FigureNPoints(l,3);}
    public double perimetre(){return fig.perimetre();}
}
```

```
public class TriangleB extends FigureNPoints {
    public TriangleB(Point[] l){super(l,3);}
}
```

```
public class TriangleC {
    FigureNPoints fig;
    public TriangleC(Point[] l){fig = new FigureNPoints(l,3);}
    public double perimetre(){return fig.perimetre();}
}
```

Le programme principal affiche les périmètres des différentes figures créées.

```
public class Application {
    Figure figure, triA, triB, rect, cerc, para;
    TriangleC triC;
    public Application() {
        Point[] liste = new Point[4];
        liste[0] = new Point(1,2);
        liste[1] = new Point(2,3);
        liste[2] = new Point(3,1);
        liste[3] = new Point(2,-1);
        figure = new FigureNPoints(liste);
        triA = new TriangleA(liste);
        triB = new TriangleB(liste);
        triC = new TriangleC(liste);
        rect = new Rectangle(liste);
        cerc = new Cercle(liste);
        para = new Parallelepipede(liste);
    }
    public void affichage(){
        System.out.println("Perimetre figure : "+figure.perimetre());
        System.out.println("Perimetre triA   : "+triA.perimetre());
        System.out.println("Perimetre triB   : "+triB.perimetre());
        System.out.println("Perimetre triC   : "+triC.perimetre());
        System.out.println("Perimetre rect   : "+rect.perimetre());
        System.out.println("Perimetre cerc   : "+cerc.perimetre());
        System.out.println("Perimetre para   : "+para.perimetre());
    }
    public static void main(String args[]){
        Application app = new Application();
        app.affichage();
    }
}
```

Question 1.1 (1 point) *Expliquez pourquoi dans* `Application.java`*, l'objet* `triC` *n'est pas déclaré comme* `Figure`*. Énoncez la règle associée de la programmation par objets.*

Réponse à la question 1.1: Pour que l'objet `triC` puisse être déclaré comme `Figure`, il faudrait que soit il implémente directement l'interface `Figure`, soit il hérite d'une classe qui implémente cette interface.

Sur cette exemple simple, on voit l'intérêt de respecter le principe suivant : Programmer pour une interface, non pour un développement.

Question 1.2 (1 point) *Expliquez les mécanismes utilisés par* `triA` *et* `triB`*.*

Réponse à la question 1.2: `triA` est un objet de la classe `TriangleA` qui implémente l'interface `Figure`. C'est un objet construit par composition (et donc en faisant appel explicite aux constructeurs), et l'implémentation de l'interface est obtenue par délégation.

`triB` est un objet de la classe `TriangleB` qui implémente l'interface `Figure` par héritage de classe. C'est donc un objet construit par le constructeur de la super classe, et le code des méthodes est hérité.

Question 1.3 (6 points) *Deux points peuvent définir un cercle : le premier est son centre, le second un point du cercle.*
Deux points peuvent définir un rectangle : le premier est un sommet, le second est son opposé.
Trois points peuvent définir un parallélépipède : les trois points permettent de "calculer" le quatrième sommet.
Sur ces considérations, et afin de pouvoir calculer les périmètres (cf le source de la classe Application), complétez les sources suivantes :

Réponse à la question 1.3:

```
public class Cercle extends FigureNPoints {
    static final double pi=3.14159;
    public Cercle(Point[] l){
        super(l,2);
    }
    public double perimetre(){
        return 2*pi*liste[0].distance(liste[1]);
    }
}
```

Plusieurs choix sont possibles pour le quatrième point, une solution :

```
public class Parallelepipede implements Figure {
    FigureNPoints fig;
    public Parallelepipede(Point[] l){
        Point[] aux = new Point[4];
        for (int i=0; i< 3; i++)
            aux[i]=l[i];
        aux[3] = new Point(l[0].abscisse()+(l[2].abscisse()-l[1].abscisse()),
                           l[0].ordonnee()+(l[2].ordonnee()-l[1].ordonnee()));
        fig = new FigureNPoints(aux,4);
    }
    public double perimetre(){
        return fig.perimetre();
    }
}
```

Plusieurs choix sont possibles pour les troisième et quatrième points, une solution :

```
public class Rectangle extends FigureNPoints {
    public Rectangle(Point[] l){
        super(l,2);
    }
    public double perimetre(){
        return 2*(Math.abs(liste[0].abscisse() - liste[1].abscisse())+
                  Math.abs(liste[0].ordonnee() - liste[1].ordonnee()));
    }
}
```

Question 1.4 (2 points) *Justifiez rapidement dans le tableau suivant vos choix.*

Réponse à la question 1.4:

Classe	Mécanisme	Justification
Cercle	héritage de classe	il est obligatoire de surcharger la méthode perimetre()
Parallelepipede	composition et délégation	- l'héritage est impossible car le constructeur FigureNPoints ne peut pas etre utilisé avec super(). Il faut donc créer un objet ⇒ composition.
Rectangle	héritage de classe ou composition	pour rendre perimetre() plus efficace si l'on avait eu besoin des 4 sommets C'est pourquoi je préfère ici l'héritage.

2 Modèles de conception (6 points)

Soit la classe `Arbre` qui permet de manipuler un arbre binaire dont chaque noeud et chaque feuille contiennent une donnée entière.

```
public class Arbre{
    public Arbre gauche, droite;
    public int data;
    public Arbre(int s, Arbre fg, Arbre fd){
        data = s;
        gauche = fg;
        droite = fd;
    }
    public void afficher(){
        if (gauche != null) gauche.afficher();
        System.out.print(data+" ");
        if (droite != null) droite.afficher();
    }
    public int maximum(){
        int d=data;
        int g=data;
        if (gauche != null) g = gauche.maximum();
        if (droite != null) d = droite.maximum();
        return Math.max(Math.max(d,g),data);
    }
}
```

Soit la classe `Algo2` qui permet de créer aléatoirement un arbre, puis d'afficher son contenu, puis son élément maximal.

```
import java.util.*;
public class Algo2 {
    Arbre info;
    public Algo2(){ // Un arbre binaire est créé aléatoirement.
        Random alea = new Random();
        int nb = (2<<(1+Math.abs(alea.nextInt()%4))) - 1;
        Arbre[] aux = new Arbre[nb];
        for (int i=aux.length -1; i >= 0; i--)
            if ((2*(i+1)-1)<aux.length)
                aux[i] = new Arbre(alea.nextInt()%100,aux[2*(i+1)-1],aux[2*(i+1)]);
            else
                aux[i] = new Arbre(alea.nextInt()%100,null,null);
        info = aux[0]; // l'objet info est associé à la tete de l'arbre.
    }
    public void algorithme(){
        System.out.println("Les données");
        info.afficher();
        System.out.println("\nLa plus grande donnée est : " + info.maximum());
    }
    public static void main(String args[]){
        Algo2 algo = new Algo2();
        algo.algorithme();
    }
}
```

Soit la classe `Algo1` qui permet de créer aléatoirement un tableau d'entiers, puis d'afficher son contenu, puis son élément maximal.

```
import java.util.*;
public class Algo1 {
    int[] info;
    public Algo1(){ // Un tableau est créé aléatoirement.
        Random alea = new Random();
        info = new int[Math.abs(alea.nextInt()%20)+1];
        for (int i=0; i < info.length; i++)
            info[i] = alea.nextInt()%100;
    }
    public void algorithme(){
        System.out.println("Les données");
        for (int i=0; i < info.length; i++)
            System.out.print(info[i]+" ");
        int max = info[0];
        for (int i=1; i < info.length; i++)
            max = Math.max(max,info[i]);
        System.out.println("\nLa plus grande donnée est : "+ max);
    }
    public static void main(String args[]){
        Algo1 algo = new Algo1();
        algo.algorithme();
    }
}
```

Le modèle **Patron de méthode** définit, dans une opération, le squelette d'un algorithme, en en déléguant certaines étapes à des sous-classes. Le patron de méthode permet de redéfinir par des sous-classes, certaines parties d'un algorithme, sans avoir à modifier la structure de ce dernier.
Sur notre exemple, nous souhaitons unifier les deux algorithmes de la façon suivante :

```
public class Algo {
    Structure struc;
    public Algo(Structure s){
        struc = s;
        struc.patronMethode();
    }
    public static void main(String args[]){
        Algo algo1 = new Algo(new Tableau());
        Algo algo2 = new Algo(new Arbre());
    }
}
```

en utilisant la classe abstraite `Structure` qui définit entre autre la méthode `patronMethode()` :

```
public abstract class Structure{
    abstract void afficher();
    abstract int maximum();
    public void patronMethode(){
        System.out.println("Les données");
        this.afficher();
        System.out.println("\nLa plus grande donnée est : "+this.maximum());
    }
}
```

Question 2.1 (6 points) *Complétez les sous-classes suivantes de* `Structure` *afin que cela soit possible.*

Réponse à la question 2.1:

```
import java.util.*;
public class Arbre extends Structure{
    public Arbre gauche, droite;
    public int data;
    public Arbre(){
        Random alea = new Random();
        int nb = (2<<(1+Math.abs(alea.nextInt()%4))) - 1;
        Arbre[] aux = new Arbre[nb];
        for (int i=aux.length -1; i >= 0; i--)
            if ((2*(i+1)-1)<aux.length)
                aux[i] = new Arbre(alea.nextInt()%100,aux[2*(i+1)-1],aux[2*(i+1)]);
            else
                aux[i] = new Arbre(alea.nextInt()%100,null,null);
        data = aux[0].data;
        gauche = aux[0].gauche;
        droite = aux[0].droite;
    }
    public Arbre(int s, Arbre fg, Arbre fd){
        data = s;
        gauche = fg;
        droite = fd;
    }
    public void afficher(){
        if (gauche != null) gauche.afficher();
        System.out.print(data+" ");
        if (droite != null) droite.afficher();
    }
    public int maximum(){
        int d=data;
        int g=data;
        if (gauche != null) g = gauche.maximum();
        if (droite != null) d = droite.maximum();
        return Math.max(Math.max(d,g),data);
    }
}
```

```
import java.util.*;
public class Tableau extends Structure{
    int[] info;
    public Tableau(){
        Random alea = new Random();
        info = new int[Math.abs(alea.nextInt()%20)+1];
        for (int i=0; i < info.length; i++)
            info[i] = alea.nextInt()%100;
    }
    public void afficher(){
        for (int i=0; i < info.length; i++)
            System.out.print(info[i]+" ");
    }
    public int maximum(){
        int max = info[0];
        for (int i=1; i < info.length; i++)
            max = Math.max(max,info[i]);
        return max;
    }
}
```

3 JAVA et JDBC (3 points)

Dans une classe qui utilise le package JDBC, vous disposez de la méthode suivante qui permet la modification des tables de votre base de données.

```
public int modifierTable(Connection c, String requete) {
    try {
        Statement s = c.createStatement();
        int res = s.executeUpdate(requete);
        return res;
    }
    catch(SQLException e){}
}
```

L'environnement JDBC vous offre dans l'`interface Connexion` les méthodes suivantes :

- `public void setAutoCommit(boolean autoCommit) throws SQLException`
  Par défaut, le mode est dit `autoCommit`, ainsi chaque requête SQL est suivi d'un ordre `commit`.
  Cette méthode permet de changer ce mode.
- `public void rollback() throws SQLException` qui exécute un ordre `rollback` sur votre base.
- `public void commit() throws SQLException` qui exécute un ordre `commit` sur votre base.

Question 3.1 (3 points) *Écrivez une méthode* `transaction` *qui permet d'effectuer une suite cohérente de modifications sur votre base, et qui satisfait la fonctionnalité suivante : Si la base est cohérente avant l'appel de cette méthode, et quelque soit l'état de la base entre l'exécution de deux requêtes de la transaction, la base est cohérente à la fin de l'exécution de la méthode.*

Réponse à la question 3.1: Une solution possible :

```
public int transaction(Connection c, String[] requetes) {
    int res = 0;
    try {
        c.setAutoCommit(false);                           // Début transaction
        Statement s = c.createStatement();
        for (int i=0; i < requetes.length; i++) res += s.executeUpdate(requetes[i]);
        c.commit();                                       // Transaction réussie
        c.setAutoCommit(true);}
    catch(SQLException e){retourArriere(c);}
    finally {return res;}
}
private void retourArriere(Connection c){         // recursivite pour recommencer si besoin.
    try {c.rollback();}                                   // Transaction annulée
    catch(SQLException e){retourArriere(c);}
}
```

4 L'environnement JDK (2 points)

Un programmeur JAVA aperçoit sur son écran le message suivant :

`Essai.java:3: Can't make a static reference to nonstatic variable abs in class Essai.`

Question 4.1 (1 point) *Que signifie le message précédent, et est-ce un message d'erreur du compilateur, ou bien un message d'erreur lors de l'exécution ?*

Réponse à la question 4.1: Une variable statique est une variable de classe et non pas une variable d'instance.
Ce message du compilateur indique que l'on essaie d'accèder de manière statique (c-à-d en préfixant par le nom de la Classe et non pas par le nom de l'objet), à une variable non statique (c-à-d une variable d'instance).

Question 4.2 (1 point) *Donnez le code d'une classe java provoquant un tel message.*

Réponse à la question 4.2:

```
public class Essai {
    private int abs;
    public Essai() {Essai.abs=0;}
}
```