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

IUP MIAGe 3^{eme} année
Année 2000-2001

Vendredi 19 janvier 2001, 14 heures - 16 heures
Tous documents autorisés.

Programmation par Objets
Alain Griffault

SUJET + CORRIGE

---

1 Implémentation de relations (10 points)

Les sources suivants permettent la construction d'un dessin contenant des formes diverses coloriées, disposées sur un support.

```
/** Fichier Forme.java */
public interface Forme {
    public void afficher();
}
```

```
/** Fichier Cercle.java */
import java.awt.*;
import java.awt.geom.*;
public class Cercle implements Forme {
    Point2D centre;
    double rayon;
    Color couleur;
    public Cercle(Point2D centre, double rayon, Color c){
        this.centre = centre;
        this.rayon = rayon;
        couleur = c;
    }
    public void afficher(){
        System.out.println("affichage d'un cercle de couleur "+
                           couleur.toString());
    }
}
```

```
/** Fichier Rectangle.java */
import java.awt.*;
import java.awt.geom.*;
public class Rectangle implements Forme {
    Point2D origine;
    double largeur, hauteur;
    Color couleur;
    public Rectangle(Point2D p, double l, double h, Color c){
        origine = p;
        largeur = l;
        hauteur = h;
        couleur = c;
    }
    public void afficher(){
        System.out.println("affichage d'un rectangle de couleur "+
                           couleur.toString());
    }
}
```

```
/** Fichier Triangle.java */
import java.awt.*;
import java.awt.geom.*;
public class Triangle implements Forme {
    Point2D[] liste = new Point2D[3];
    Color couleur;
    public Triangle(Point2D p0, Point2D p1, Point2D p2, Color c){
        liste[0] = p0;
        liste[1] = p1;
        liste[2] = p2;
        couleur = c;
    }
    public void afficher(){
        System.out.println("affichage d'un triangle de couleur "+
                           couleur.toString());
    }
}
```

```
/** Fichier Support.java */
import java.awt.*;
import java.awt.geom.*;
public class Support {
    Point2D origine;
    double largeur, hauteur;
    public Support(Point2D p, double l, double h){
        origine = p;
        largeur = l;
        hauteur = h;
    }
}
```

```
/** Fichier Dessin.java */
import java.awt.*;
import java.awt.geom.*;
import java.util.*;
public class Dessin {
    Support support;
    Color fond;
    Vector formes;
    public Dessin(Color c) {
        support = new Support(new Point(0,0), 10.0, 10.0);
        fond = c;
        formes = new Vector();
        formes.add(new Cercle(new Point(2,3), 2.2, Color.red));
        formes.add(new Rectangle(new Point(3,2), 1.8, 3.0, Color.blue));
        formes.add(new Triangle(new Point(4,2), new Point(4,5), new Point(7,8), Color.green));
    }
    public void afficher(){
        System.out.println("affichage d'un support de couleur "+
                           fond.toString());
        for (Enumeration e = formes.elements() ; e.hasMoreElements() ;) {
            ((Forme)e.nextElement()).afficher();
        }
    }
    public static void main(String args[]){
        Dessin dessin = new Dessin(Color.yellow);
        dessin.afficher();
    }
}
```

Une exécution de ce programme donne :

```
affichage d'un support de couleur java.awt.Color[r=255,g=255,b=0]
affichage d'un cercle de couleur java.awt.Color[r=255,g=0,b=0]
affichage d'un rectangle de couleur java.awt.Color[r=0,g=0,b=255]
affichage d'un triangle de couleur java.awt.Color[r=0,g=255,b=0]
```

Question 1.1 (2 points) *Dessinez le diagramme des classes de ce programme.*

Réponse à la question 1.1:
Le support du dessin est un rectangle, en fait il pourrait être d'une géométrie quelconque. Le support et la couleur de fond peuvent donc être vus comme une forme.

Question 1.2 (1 point) *Dessinez le nouveau diagramme des classes.*

Réponse à la question 1.2:

Question 1.3 (1 point) *Donnez le nouveau code de la classe* `Dessin`.

Réponse à la question 1.3:

```
import java.awt.*;
import java.awt.geom.*;
import java.util.*;

public class Dessin {
    Forme support;
    Vector formes;
    public Dessin(Color c) {
        support = new Rectangle(new Point(0,0),10.0, 10.0, c);
        formes = new Vector();
        formes.add(new Cercle(new Point(2,3), 2.2, Color.red));
        formes.add(new Rectangle(new Point(3,2), 1.8, 3.0, Color.blue));
        formes.add(new Triangle(new Point(4,2), new Point(4,5), new Point(7,8), Color.green));
    }
    public void afficher(){
        support.afficher();
        for (Enumeration e = formes.elements() ; e.hasMoreElements() ;) {
            ((Forme)e.nextElement()).afficher();
        }
    }
    public static void main(String args[]){
        Dessin dessin = new Dessin(Color.yellow);
        dessin.afficher();
    }
}
```

On souhaite pouvoir n'afficher, en plus du support, que les formes d'une couleur donnée. Une solution consiste à modifier ainsi l'interface `Forme`.

```
import java.awt.*;

public interface Forme {
    public void afficher();
    public void afficher(Color c);
}
```

Question 1.4 (2 points) *Donnez les modifications à apporter aux autres classes.*

Réponse à la question 1.4:

– Il faut implémenter dans les trois classes `Cercle`, `Rectangle` et `Triangle` la methode :

```
public void afficher(Color c){
    if (couleur == c)
        System.out.println("affichage d'un rectangle");
}
```

– Il faut implémenter dans la classe `Dessin` la methode :

```
public void afficher(Color c){
    for (Enumeration e = formes.elements() ; e.hasMoreElements() ;) {
        ((Forme)e.nextElement()).afficher(c);
    }
}
```

Question 1.5 (1 point) *Quel est le principal défaut de cette première solution ?*

Réponse à la question 1.5: Pour afficher les formes d'une couleur donnée, il faut parcourir entièrement la structure `vector`. Cela peut être coûteux en temps (une seule forme rouge sur un ensemble de 1000 pièces par exemple).

Une autre solution consiste à gérer des listes de références par couleur aux différentes formes du dessin.

Question 1.6 (3 points) *Proposez un nouveau diagramme des classes. Justifiez en quelques phrases.*

Réponse à la question 1.6:

2 Modèles de conception (7 points)

Soit la classe `Client` qui permet de tester la présence dans un tableau d'entiers d'une valeur passée en paramètre.

```
import java.util.*;
public class Client {
    int[] info;
    int choixAlgorithme;
    public Client(){ // 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()%6;
        choixAlgorithme = alea.nextInt()%2;
    }
    public void afficher(){
        for (int i=0; i < info.length; i++)
            System.out.print(info[i]+" ");
    }
    public boolean presentSequentiel(int valeur){
        for (int i=0; i < info.length; i++)
            if (info[i]==valeur)
                return true;
        return false;
    }
    public boolean presentDichotomique(int valeur){
        Arrays.sort(info); // il faut que le tableau soit trie
        int gauche = 0;
        int droite = info.length -1;
        while (gauche <= droite) {
            int aux = (gauche + droite) / 2;
            if (info[aux]==valeur)
                return true;
            else if (info[aux] < valeur)
                gauche = aux + 1;
            else
                droite = aux - 1;
        }
        return false;
    }
    public boolean present(int valeur){
        switch(choixAlgorithme){
        case 0 : return presentDichotomique(valeur);
        case 1 : return presentSequentiel(valeur);
        default : return false;
        }
    }
    public static void main(String args[]){
        Client client = new Client();
        client.afficher();
        if (client.present(Integer.parseInt(args[0])))
            System.out.println("Présent");
        else
            System.out.println("Absent");
    }
}
```

Par exemple la commande `java Client 4` peut donner le résultat :

```
1 5 3 -4 3 3 -1 -4 -5 2 0 0 -5 Absent
```

Le modèle **Stratégie** définit une famille d'algorithmes, encapsule chacun d'entre eux, et les rend interchangeables. Le modèle **Stratégie** permet aux algorithmes d'évoluer indépendamment des clients qui les utilisent.

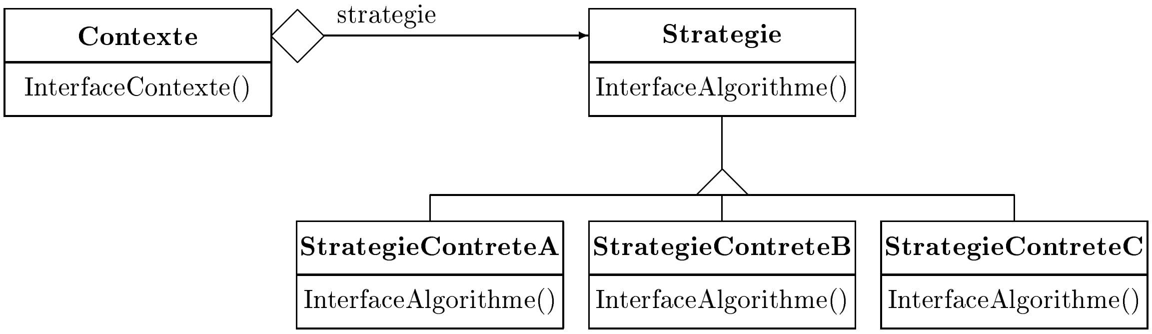

FIG. 1 – La structure du modèle Stratégie

Sur notre exemple, nous souhaitons encapsuler les deux algorithmes de recherche de la façon suivante :

```
public interface Strategie {
    public boolean present(int[]info, int valeur);
}
```

Question 2.1 (1 point) *Complétez la sous-classe* `StrategieSequentielle`.

Réponse à la question 2.1:

```
public class StrategieSequentielle implements Strategie {
    public boolean present(int[]info, int valeur){
        for (int i=0; i < info.length; i++) {
            if (info[i]==valeur)
                return true;
        }
        return false;
    }
}
```

Question 2.2 (1 point) *Complétez la sous-classe* `StrategieDichotomique`.

Réponse à la question 2.2:

```
import java.util.*;
public class StrategieDichotomique implements Strategie {
    public boolean present(int[]info, int valeur){
        Arrays.sort(info); // il faut que le tableau soit trie
        int gauche = 0;
        int droite = info.length -1;
        while (gauche <= droite) {
            int aux = (gauche + droite) / 2;
            if (info[aux]==valeur)
                return true;
            else if (info[aux] < valeur)
                gauche = aux + 1;
            else
                droite = aux - 1;
        }
        return false;
    }
}
```

Question 2.3 (2 points) *Complétez la sous-classe* `Client`*.*

Réponse à la question 2.3:

```
import java.util.*;
public class Client {
    int[] info;
    int choixAlgorithme;
    Strategie strat;
    public Client(){ // 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()%6;
        choixAlgorithme = alea.nextInt()%2;
    }
    public void afficher(){
        for (int i=0; i < info.length; i++)
            System.out.print(info[i]+" ");
    }
    public void setStrategie(){
        switch(choixAlgorithme){
        case 0 : strat = new StrategieDichotomique(); break;
        case 1 : strat = new StrategieSequentielle(); break;
        default : strat = new StrategieSequentielle();
        }
    }
    public boolean present(int valeur){
        return strat.present(info,valeur);
    }
    public static void main(String args[]){
        Client client = new Client();
        client.setStrategie();
        client.afficher();
        if (client.present(Integer.parseInt(args[0])))
            System.out.println("Présent");
        else
            System.out.println("Absent");
    }
}
```

Question 2.4 (3 points) *Dans la classe* `Client`*, un booléen supplémentaire* `contexte` *indique si le tableau d'entiers est trié ou non. Quelles sont les modifications à apporter afin que l'on ne trie pas inutilement un tableau déjà trié ?*

Réponse à la question 2.4: Il existe généralement deux solutions lorsque les méthodes d'une classe sont dépendantes d'un attribut d'une autre classe. La première consiste à fournir l'attribut comme paramètre des méthodes (il faut souvent surcharger les méthodes) ; la seconde consiste d'une part que la classe qui contient l'attribut fournisse une méthode pour connaître l'attribut, d'autre part que l'autre classe possède une référence de la première classe (il faut modifier le constructeur).
On ne donne que les modifications des sources

Première solution

```
public class Client {
    ...
    boolean contexte;
    public Client(){ // Un tableau est créé aléatoirement.
        ...
        contexte = false;
    }
    public boolean present(int valeur){
```

```
        return strat.present(info,valeur,contexte);
    }
}
```

```
public interface Strategie {
    public boolean present(int[]info, int valeur);
    public boolean present(int[]info, int valeur, boolean trie);
}
```

```
import java.util.*;
public class StrategieDichotomique implements Strategie {
    public boolean present(int[]info, int valeur){
        return present(info,valeur,false);
    }
    public boolean present(int[]info, int valeur, boolean trie){
        if (!trie)
            Arrays.sort(info); // il faut que le tableau soit trie
        int gauche = 0;
        int droite = info.length -1;
        while (gauche <= droite) {
            int aux = (gauche + droite) / 2;
            if (info[aux]==valeur)
                return true;
            else if (info[aux] < valeur)
                gauche = aux + 1;
            else
                droite = aux - 1;
        }
        return false;
    }
}
```

```
public class StrategieSequentielle implements Strategie {
    public boolean present(int[]info, int valeur){
        for (int i=0; i < info.length; i++) {
            if (info[i]==valeur)
                return true;
        }
        return false;
    }
    public boolean present(int[]info, int valeur, boolean trie){
        return present(info,valeur);
    }
}
```

Deuxième solution

```
public class Client {
    ...
    boolean contexte;
    public Client(){ // Un tableau est créé aléatoirement.
        ...
        contexte = false;
    }
    public boolean isSorted(){
        return contexte;
    }
    public void setStrategie(){
        switch(choixAlgorithme){
        case 0 : strat = new StrategieDichotomique(this); break;
```

```
        ...
        }
    }
}
```

```
import java.util.*;
public class StrategieDichotomique implements Strategie {
    Client client;
    public StrategieDichotomique(Client cl){
client = cl;
    }
    public boolean present(int[]info, int valeur){
        if (!client.isSorted())
    Arrays.sort(info); // il faut que le tableau soit trie
        int gauche = 0;
        ...
    }
}
```

Sur cet exemple, la seconde solution semble mieux adaptée.

3 L'environnement JDK (3 points)

Suite à la commande `javac *java`, un programmeur JAVA aperçoit sur son écran le message suivant :

```
Modele.java:1: class Modele must be declared abstract.
It does not define boolean satisfaite() from interface Specification.
public class Modele implements Specification {
             ^
1 error
```

Question 3.1 (1 point) *Que signifie le message précédent ?*

Réponse à la question 3.1: La classe `Modele` déclare implémenter l'interface `Specification`, elle doit donc implémenter toutes les méthodes de cette interface. Le compilateur indique par le message d'erreur que la classe `Modele` n'implémente pas la méthode `boolean satisfaite()`. En pratique, deux solutions : soit déclarer la classe `Modele` abstraite (ce qui autorise à ne pas implémenter toutes les méthodes des interfaces), soit implémenter effectivement toutes les méthodes.

Question 3.2 (1 point) *Le message précédent est-il un message d'erreur du compilateur, ou bien un message d'erreur lors de l'exécution ?*

Réponse à la question 3.2: C'est un message d'erreur du compilateur.

Question 3.3 (1 point) *Donnez le contenu des fichiers* `Specification.java` *et* `Modele.java` *provoquant un tel message.*

Réponse à la question 3.3:

```
public interface Specification {
    public boolean satisfaite();
}
```

```
public class Modele implements Specification {
}
```