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

IUP MIAGe 3^{eme} année
Année 2002-2003

Mardi 7 janvier 2003, 10 heures 30 - 12 heures
Tous documents autorisés.

Programmation par Objets
Alain Griffault

Numéro d'anonymat :

Introduction

Une expression arithmétique est composée d'entiers, des opérateurs $\{+, -, *\}$ et éventuellement des parenthèses $\{(,)\}$. La structure classique pour stocker une telle expression est un arbre binaire dont les nœuds contiennent les opérateurs et les feuilles les entiers. Par exemple la figure 1 représente l'expression $(((3-4)+(5*6))+(1*2))$.

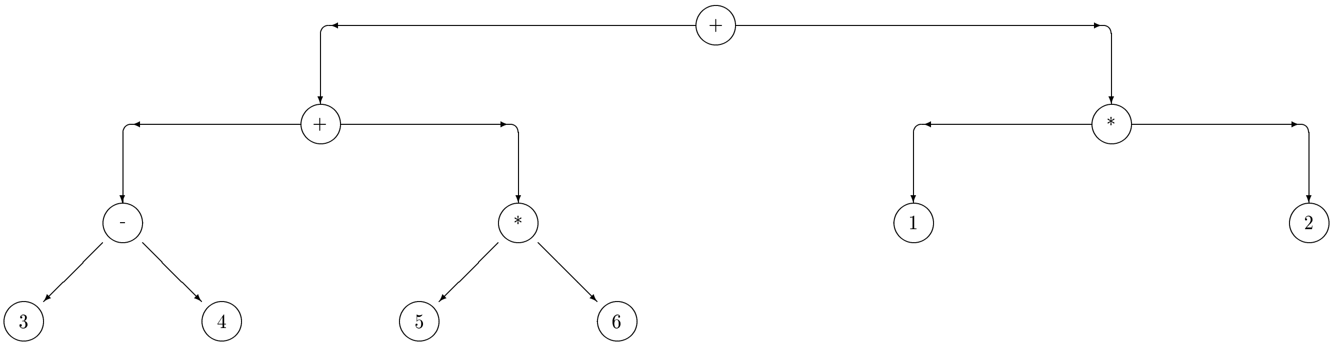

FIG. 1 – arbre binaire de $(((3-4)+(5*6))+(1*2))$

Le sujet porte sur les parcours en profondeur des arbres qui visitent une et une seule fois chaque feuille de l'arbre, et trois fois (en descendant, entre les deux fils et en remontant) chaque nœud. L'étude compare l'utilisation de modèles de conception afin de faciliter l'ajout de fonctionnalités concernant le traitement des expressions arithmétiques.

Avertissement

Les parties 2 et 3 sont indépendantes, et peuvent être traitées dans un ordre quelconque. Cependant, il peut s'avérer utile de lire la partie 3 avant de faire la partie 2 et réciproquement.

1 Parcours d'arbre pour expressions arithmétiques

On souhaite en effectuant des parcours récursifs de la structure arborescente afficher l'expression sous différents formats : préfixe, infixe et postfixe.

Question 1.1 (2 points) *Donnez le code de la méthode* `afficheInfixe` *de la classe* `Arbre` *afin que, pour l'exemple précédent, le programme* `Test` *génère la sortie donnée. N'oubliez pas les parenthèses.*

```
public class Arbre{
    private int   info;
    private char  op;
    private Arbre fg, fd;

    public Arbre(int value){
```

```
            info = value;
        }
        public Arbre(char c, Arbre ag, Arbre ad){
            op = c;
            fg = ag;
            fd = ad;
        }

        public final void affichePrefixe(){
            if ((fg == null) & (fd == null))
                System.out.print(" "+info+" ");
            else {
                System.out.print(" "+op+" ");
                fg.affichePrefixe();
                fd.affichePrefixe();
            }
        }

        public final void affichePostfixe(){
            if ((fg == null) & (fd == null))
                System.out.print(" "+info+" ");
            else {
                fg.affichePostfixe();
                fd.affichePostfixe();
                System.out.print(" "+op+" ");
            }
        }

        public final void afficheInfixe(){









            }
        }
}
```

Le programme `Test` :

```
public class Test{
    public static void main(String[] args){
        Arbre a6=new Arbre(1);
        Arbre a7=new Arbre(2);
        Arbre a8=new Arbre(3);
        Arbre a9=new Arbre(4);
        Arbre a10=new Arbre(5);
        Arbre a11=new Arbre(6);
        Arbre a4=new Arbre('-',a8,a9);
        Arbre a5=new Arbre('*',a10,a11);
        Arbre a2=new Arbre('+',a4,a5);
        Arbre a3=new Arbre('*',a6,a7);
        Arbre a1=new Arbre('+',a2,a3);
        System.out.print("Affichage prefixe       : ");
```

```
        a1.affichePrefixe();
        System.out.print("\nAffichage infixe        : ");
        a1.afficheInfixe();
        System.out.print("\nAffichage postfixe      : ");
        a1.affichePostfixe();
        System.out.println();
    }
}
```

et la sortie générée.

```
Affichage prefixe       :  +  +  -  3  4  *  5  6  *  1  2
Affichage infixe        : ((( 3  -  4 ) + ( 5  *  6 )) + ( 1  *  2 ))
Affichage postfixe      :  3  4  -  5  6  *  +  1  2  *  +
```

2 Le modèle de conception *stratégie*

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.

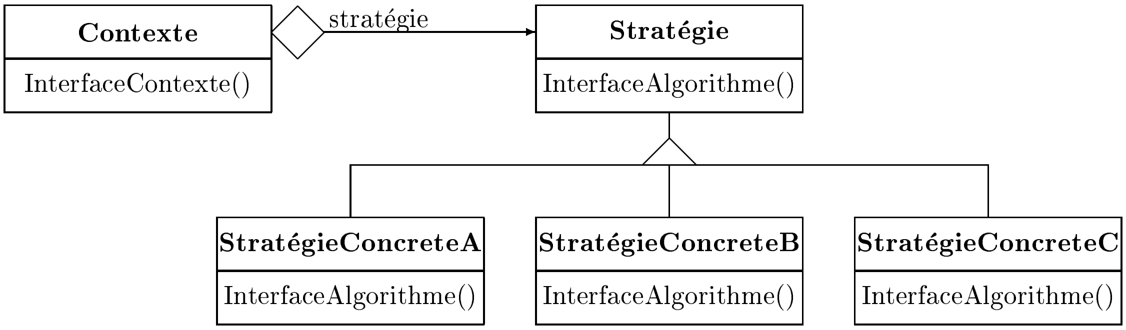

FIG. 2 – La structure du modèle Stratégie

L'objectif de l'exercice est que le nouveau programme `test`

```
public class Test{
    public static void main(String[] args){
        Arbre a6=new Arbre(1);
        Arbre a7=new Arbre(2);
        Arbre a8=new Arbre(3);
        Arbre a9=new Arbre(4);
        Arbre a10=new Arbre(5);
        Arbre a11=new Arbre(6);
        Arbre a4=new Arbre('-',a8,a9);
        Arbre a5=new Arbre('*',a10,a11);
        Arbre a2=new Arbre('+',a4,a5);
        Arbre a3=new Arbre('*',a6,a7);
        Arbre a1=new Arbre('+',a2,a3);
        System.out.print("Parcours sans affichage : ");
        a1.parcours(new StrategieParcours());
        System.out.print("\nAffichage prefixe       : ");
        a1.parcours(new StrategieParcoursPrefixe());
        System.out.print("\nAffichage infixe        : ");
        a1.parcours(new StrategieParcoursInfixe());
        System.out.print("\nAffichage postfixe      : ");
        a1.parcours(new StrategieParcoursPostfixe());
        System.out.println();
    }
}
```

génère la sortie correcte suivante.

```
Parcours sans affichage :
Affichage prefixe       :  +  +  -  3  4  *  5  6  *  1  2
Affichage infixe        : ((( 3  -  4 ) + ( 5  *  6 )) + ( 1  *  2 ))
Affichage postfixe      :  3  4  -  5  6  *  +  1  2  *  +
```

Pour cela, le modèle **Stratégie** est implémenté non pas comme un interface, mais comme la classe `StrategieParcours` donnant les comportements par défauts.

```
public class StrategieParcours{
    public void visiterFeuille      (Arbre arbre){}
    public void visiterNoeudPrefixe (Arbre arbre){}
    public void visiterNoeudInfixe  (Arbre arbre){}
    public void visiterNoeudPostfixe(Arbre arbre){}
}
```

Question 2.1 (4 points) *Donnez le code de la méthode* `parcours` *de la classe* `Arbre`*.*

```
public class Arbre{
    private int   info;
    private char  op;
    private Arbre fg, fd;

    public Arbre(int value){
        info = value;
    }
    public Arbre(char c, Arbre ag, Arbre ad){
        op = c;
        fg = ag;
        fd = ad;
    }

    public final int   getInfo(){return info;}
    public final char  getOp()  {return op;}

    public final void parcours(                                   ){










        }
    }
}
```

Question 2.2 (2 points) *Donnez le code de la classe* `StrategieParcoursPrefixe`*.*

```
public class StrategieParcoursPrefixe                              {





}
```

Question 2.3 (2 points) *Donnez le code de la classe* `StrategieParcoursInfixe`*.*

```
public class StrategieParcoursInfixe                                {













}
```

3 Le modèle de conception *patron de méthode*

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.

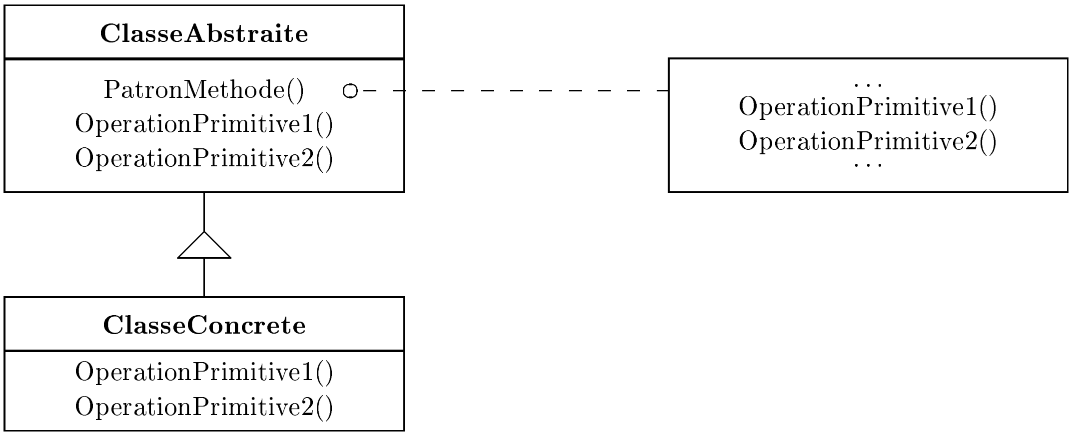

FIG. 3 – La structure du modèle Patron de Méthode

L'objectif de l'exercice est le même que précédemment ; malheureusement, la classe `Arbre` suivante :

```
public class Arbre{
    private int   info;
    private char  op;
    private Arbre fg, fd;

    public Arbre(int value){
        info = value;
    }
    public Arbre(char c, Arbre ag, Arbre ad){
        op = c;
        fg = ag;
        fd = ad;
    }

    public final int  getInfo(){return info;}
```

```
    public final char getOp()  {return op;}

    public final void parcours(){
        if ((fg == null) & (fd == null))
            visiterFeuille(this);
        else {
            visiterNoeudPrefixe(this);
            fg.parcours();
            visiterNoeudInfixe(this);
            fd.parcours();
            visiterNoeudPostfixe(this);
        }
    }

    public void visiterFeuille      (Arbre arbre){}
    public void visiterNoeudPrefixe (Arbre arbre){}
    public void visiterNoeudInfixe  (Arbre arbre){}
    public void visiterNoeudPostfixe(Arbre arbre){}
}
```

et ses sous-classes `ArbreParcoursPrefixe`, `ArbreParcoursInfixe` et `ArbreParcoursPostfixe` qui redéfinissent les méthodes `visiterFeuille(Arbre arbre)`, `visiterNoeudPrefixe(Arbre arbre)`, `visiterNoeudInfixe(Arbre arbre)` et `visiterNoeudPostfixe(Arbre arbre)` donnent avec le nouveau programme `test`

```
public class Test{
    public static void main(String[] args){
        Arbre a6=new Arbre(1);
        Arbre a7=new Arbre(2);
        Arbre a8=new Arbre(3);
        Arbre a9=new Arbre(4);
        Arbre a10=new Arbre(5);
        Arbre a11=new Arbre(6);
        Arbre a4=new Arbre('-',a8,a9);
        Arbre a5=new Arbre('*',a10,a11);
        Arbre a2=new Arbre('+',a4,a5);
        Arbre a3=new Arbre('*',a6,a7);
        Arbre a1=new Arbre('+',a2,a3);
        System.out.print("Parcours sans affichage : ");
        a1.parcours();
        a1=new ArbreParcoursPrefixe('+',a2,a3);
        System.out.print("\nAffichage prefixe       : ");
        a1.parcours();
        a1=new ArbreParcoursInfixe('+',a2,a3);
        System.out.print("\nAffichage infixe        : ");
        a1.parcours();
        a1=new ArbreParcoursPostfixe('+',a2,a3);
        System.out.print("\nAffichage postfixe      : ");
        a1.parcours();
        System.out.println();
    }
}
```

la sortie incorrecte suivante.

```
Parcours sans affichage :
Affichage prefixe       :  +
Affichage infixe        : ( + )
Affichage postfixe      :  +
```

Question 3.1 (3 points) *Expliquez la sortie précédente. Proposez une correction ou bien expliquez pourquoi il n'y a pas de correction simple.*

La nouvelle classe `Arbre`

```
public class Arbre{
    private int     info;
    private char    op;
    private Arbre   fg, fd, pere;
    private boolean empile;

    public Arbre(int value){
        info = value;
    }
    public Arbre(char c, Arbre ag, Arbre ad){
        op = c;
        fg = ag;
        fd = ad;
        fg.setPere(this);
        fd.setPere(this);
    }

    public final int     getInfo()           {return info;}
    public final char    getOp()             {return op;}
    public final boolean getEmpile()         {return empile;}
    public final void    setPere(Arbre arbre){pere = arbre;}
    public final void    setEmpile(boolean b){empile = b;}

    public final void parcoursIteratif(){
        Arbre courant = this;
        boolean fini = false;
        while (fini == false) {
            while ((courant.fg != null) & (courant.fd != null)){
                visiterNoeudPrefixe(courant);
                courant.setEmpile(true);
                courant = courant.fg;
            }
            visiterFeuille(courant);
            courant = courant.pere;
            while ((courant.getEmpile()==false) &&  (courant.pere != null)) {
                visiterNoeudPostfixe(courant);
                courant = courant.pere;
```

```
            }
            if (courant.getEmpile()==true) {
                courant.setEmpile(false);
                visiterNoeudInfixe(courant);
                courant = courant.fd;
            }
            else {
                visiterNoeudPostfixe(courant);
                fini = true;
            }
        }
    }

    public void visiterFeuille      (Arbre arbre){}
    public void visiterNoeudPrefixe (Arbre arbre){}
    public void visiterNoeudInfixe  (Arbre arbre){}
    public void visiterNoeudPostfixe(Arbre arbre){}
}
```

qui utilise un parcours itératif et non plus récursif permet d'obtenir avec des sous-classes inchangées le résultat correct suivant.

```
Parcours sans affichage :
Affichage prefixe       :  +  +  -  3  4  *  5  6  *  1  2
Affichage infixe        : ((( 3  -  4 ) + ( 5  *  6 )) + ( 1  *  2 ))
Affichage postfixe      :  3  4  -  5  6  *  +  1  2  *  +
```

Question 3.2 (2 points) *Expliquez pourquoi le parcours itératif donne le bon résultat.*

4 Comparaison et ajout de fonctionnalités

Question 4.1 (2 points) *Pour ce problème relatif aux expressions arithmétiques, comparez les deux modèles de conception proposés et justifiez celui que vous retiendriez dans un projet.*

Le client vous demande d'ajouter l'évaluation de l'expression, afin d'avoir le résultat suivant :

```
Parcours sans affichage :
Affichage prefixe       :  +  +  -  3  4  *  5  6  *  1  2
Affichage infixe        : ((( 3  -  4 ) + ( 5  *  6 )) + ( 1  *  2 ))
Affichage postfixe      :  3  4  -  5  6  *  +  1  2  *  +
Valeur de l'expression  : 31
```

Question 4.2 (3 points) *À partir du modèle retenu, donnez toutes les modifications et les ajouts nécessaires.*