

Les patrons Composite, Interpréteur, Visiteur

jean-michel Douin, douin au cnam point fr
version : 18 Septembre 2013

Notes de cours

ESIEE

1

Sommaire pour les Patrons

- **Classification habituelle**

- **Créateurs**

- Abstract Factory, Builder, Factory Method, Prototype, Singleton

- **Structurels**

- Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy,

- **Comportementaux**

- Chain of Responsibility. Command, Interpreter, Iterator,
Mediator, Memento, Observer, State,
Strategy, Template Method, Visitor

ESIEE

2

Les patrons déjà vus en quelques lignes ...

- **Adapter**
 - Adapte l'interface d'une classe conforme aux souhaits du client
- **Proxy**
 - Fournit un mandataire au client afin de contrôler/vérifier ses accès
- **Observer**
 - Notification d'un changement d'état d'une instance aux observateurs inscrits
- **Template Method**
 - Laisse aux sous-classes une bonne part des responsabilités
- **Iterator**
 - Parcours d'une structure sans se soucier de la structure interne choisie

ESIEE

3

Sommaire

- **Structures de données récursives**
 - Le patron Composite
 - Le patron Interpréteur
 - API Graphique en java(AWT), paquetage java.awt
 - Parcours : Itérateur et/ou visiteur

ESIEE

4

Principale bibliographie

- **GoF95**

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
- Design Patterns, Elements of Reusable Object-oriented software Addison Wesley 1995

+

- <http://www.eli.sdsu.edu/courses/spring98/cs635/notes/composite/composite.html>
- <http://www.patterndepot.com/put/8/JavaPatterns.htm>

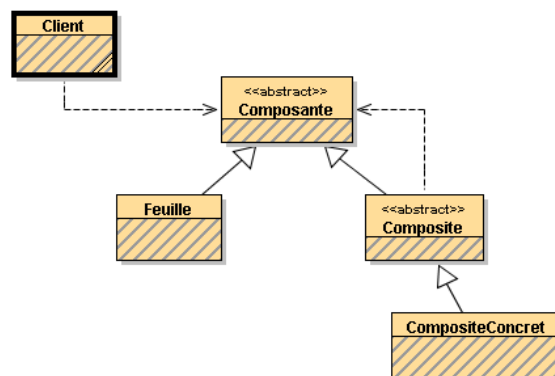
+

- <http://www.javaworld.com/javaworld/jw-12-2001/jw-1214-designpatterns.html>
- www.oreilly.com/catalog/hfdesignpat/chapter/ch03.pdf

ESIEE

5

Structures récursives : le pattern Composite



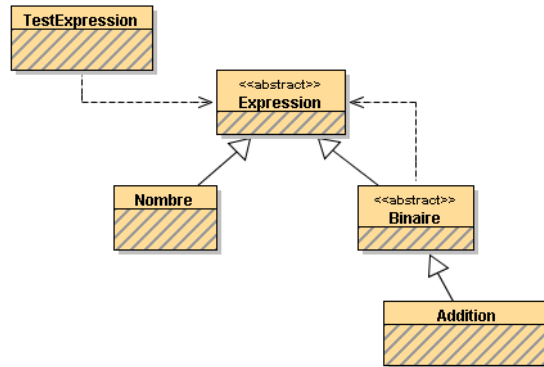
Composante ::= Composite | Feuille
Composite ::= CompositeConcret
CompositeConcret ::= { Composante }
Feuille ::= 'symbole terminal'

Tout est Composante

ESIEE

6

Un exemple : Expression 3 , 3+2, 3+2+5,...



Expression ::= Binaire | Nombre

Binaire ::= Addition

Addition ::= Expression '+' Expression

Nombre ::= 'une valeur de type int'

Tout est Expression

ESIEE

7

Composite et Expression en Java

La Racine du composite : *Tout est Expression*

```
public abstract class Expression{}
```

Au plus simple

ESIEE

8

Binaire est une Expression

```
public abstract class Expression{}

public abstract class Binaire extends Expression{
    // binaire id a deux opérandes
    protected Expression op1;
    protected Expression op2;

    public Binaire(Expression op1, Expression op2){
        this.op1 = op1;
        this.op2 = op2;
    }
}
```

ESIEE

9

Addition est une opération Binaire

// Symbole non terminal

```
public class Addition extends Binaire{

    public Addition(Expression op1, Expression op2){
        super(op1, op2);
    }
}
```

Soustraction, Division ...

ESIEE

10

Nombre est une Expression

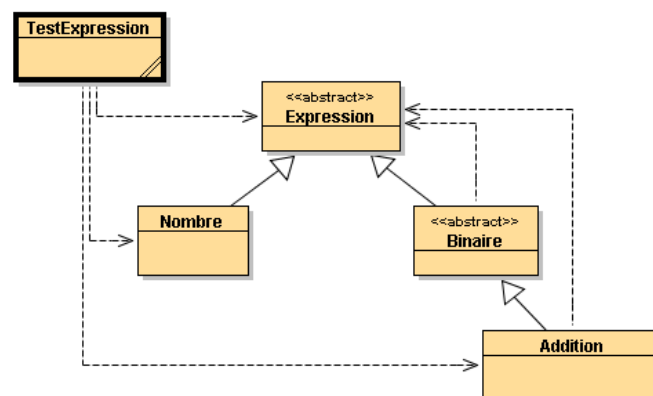
// Symbole terminal

```
public class Nombre extends Expression{  
    private int valeur;  
    public Nombre(int valeur){  
        this.valeur = valeur;  
    }  
}
```

ESIEE

11

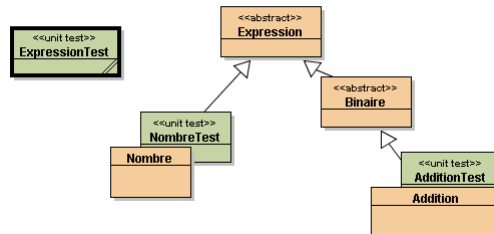
Le diagramme au complet



ESIEE

12

Démonstration avec BlueJ



- Discussion ...

ESIEE

13

Quelques instances d'Expression en Java

```
public class ExpressionTest extends junit.framework.TestCase {

    public static void test1(){

        Expression exp1 = new Nombre(321);

        Expression exp2 = new Addition(
            new Nombre(33),
            new Nombre(33)
        );

        Expression exp3 = new Addition(
            new Nombre(33),
            new Addition(
                new Nombre(33),
                new Nombre(11)
            )
        );

        Expression exp4 = new Addition(exp1,exp3);

    }
}
```

ESIEE

14

L 'AWT utilise-t-il le Pattern Composite ?

- **Component, Container, Label, JLabel**

ESIEE

15

Objets graphiques en Java

- **Comment se repérer dans une API de 180 classes (java.awt et javax.swing) ?**

La documentation : une liste (indigeste) de classes.

exemple

- **java.awt.Component**
 - Direct Known Subclasses:
 - **Button, Canvas, Checkbox, Choice, Container, Label, List, Scrollbar, TextComponent**
- +--java.awt.Component
 - |
 - +--java.awt.Container
 - Direct Known Subclasses:
 - **BasicSplitPaneDivider, CellRendererPane, DefaultTreeCellEditor.EditorContainer, JComponent, Panel, ScrollPane, Window**

ESIEE

16

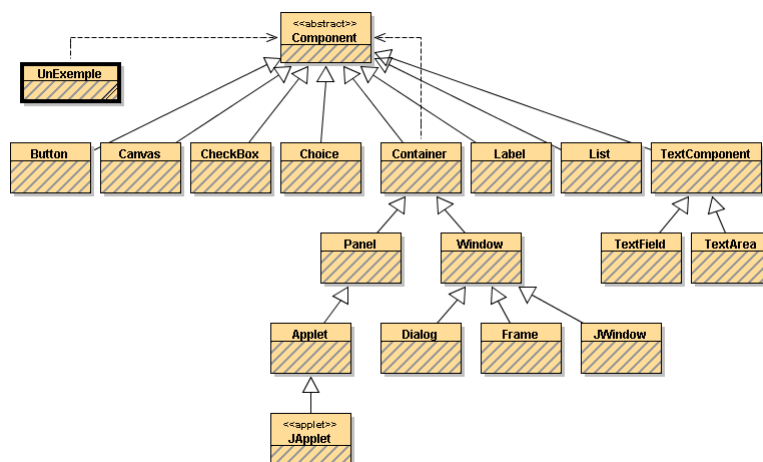
Pattern Composite et API Java

- **API Abstract Window Toolkit**
 - Une interface graphique est constituée d'objets
 - De composants
 - Bouton, menu, texte, ...
 - De composites (composés des composants ou de composites)
 - Fenêtre, applette, ...
- **Le Pattern Composite est utilisé**
 - Une interface graphique est une expression respectant le Composite (la grammaire)
- **En Java au sein des paquetages java.awt et de javax.swing.**

ESIEE

17

l'AWT utilise un Composite



- `class Container extends Component ...{`
 - `Component add (Component comp);`

ESIEE

18

Discussion

Une Expression de type composite (Expression)

- Expression e = new Addition(new Nombre(1),new Nombre(3));

Une Expression de type composite (API AWT)

- Container p = new Panel();
- p.add(new Button("b1 ")) ;
- p.add(new Button("b2 ")) ;

ESIEE

19

Démonstration ici UnExemple

```
import java.awt.*;
public class UnExemple{

    public static void main(String[] args){
        Frame f = new Frame("UnExemple");

        Container p = new Panel();
        p.add(new Button("b1"));
        p.add(new Button("b2"));
        p.add(new Button("b3"));

        Container p2 = new Panel();
        p2.add(p);p.add(new TextField(" du texte"));

        f.add(p2);
        f.pack();f.setVisible(true);
    }
}
```



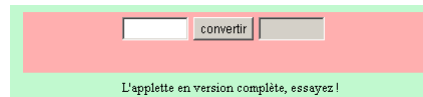
ESIEE

20



Une Applette

```
public class UneApplette extends Applet {  
    private TextField entree = new TextField(6);  
    private Button bouton = new Button("convertir");  
    private TextField sortie = new TextField(6);  
  
    public void init() {  
        add(entree); // ajout de feuilles au composite  
        add(boutonDeConversion); // Applet  
        add(sortie);  
        ...  
    }  
}
```



ESIEE

21

Le Pattern Composite : un premier bilan

- **Composite :**
 - Représentation de structures de données récursives
 - Techniques de classes abstraites
 - Liaison dynamique
 - Manipulation uniforme (tout est Composant)
 - C'est bien !, mais pour faire quoi ?

ESIEE

22

Pour quoi faire ?

- **Un Composite**

- Une instance d'un composite est une représentation interne, parmi d'autres ...
- Apparenté arbre de syntaxe abstraite
- Transformation, évaluation, compilation,

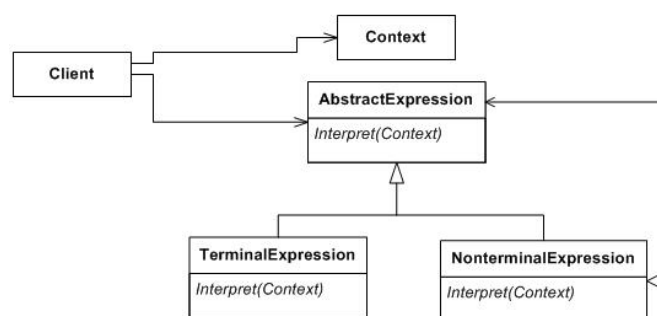
- **Une évaluation de cette structure : le Pattern Interpréteur**

- // 3+2
- Expression e = new Addition(new Nombre(3),new Nombre(2));
- // appel de la méthode interpreter, installée dans la classe Expression
- int resultat = e.interpreter();
- assert(resultat == 5); // enfin

ESIEE

23

Pattern Interpréteur : l'original



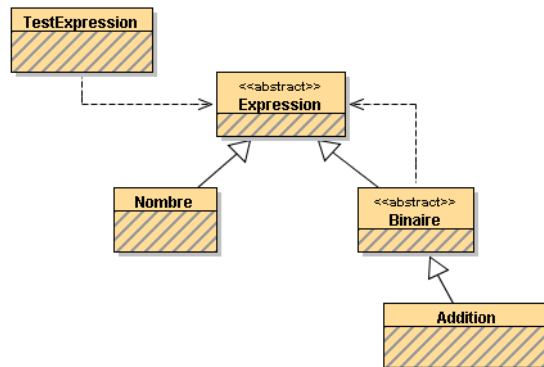
- **De GoF95 ...**

- une interprétation, évaluation dans un certain contexte

ESIEE

24

Le Pattern Interpréteur : Composite + évaluation



- Première version : chaque classe possède la méthode
 - **public int interpreter();**
 - abstraite pour les classes abstraites, concrètes pour les autres
 - ici pas de contexte (pour le moment)

ESIEE

25

Le pattern interpréteur

```
public abstract class Expression{
    abstract public int interpreter();
}

public abstract class Binaire extends Expression{
    protected Expression op1;
    protected Expression op2;

    public Binaire(Expression op1, Expression op2){
        this.op1 = op1;
        this.op2 = op2;
    }

    public Expression op1(){ return op1; }
    public Expression op2(){ return op2;}

    abstract public int interpreter();
}
```

ESIEE

26

Le pattern interpréteur

```
public class Addition extends Binaire{
    public Addition(Expression op1,Expression op2){
        super(op1,op2);
    }

    public int interpreter(){
        return op1.interpreter() + op2.interpreter();
    }
}
```

ESIEE

27

Classe Nombre

```
public class Nombre extends Expression{
    private int valeur;
    public Nombre(int valeur){
        this.valeur = valeur;
    }

    public int valeur(){ return valeur;}

    public int interpreter();
    return valeur;
}
}
```

ESIEE

28

Interprétations simples

```
// quelques assertions
Expression exp1 = new Nombre(321);
int resultat = exp1.interpreter();
assert resultat == 321;

Expression exp2 = new Addition(
    new Nombre(33),
    new Nombre(33)
);
resultat = exp2.interpreter();
assert resultat==66;
```

ESIEE

29

Quelques « interprétations » en Java

```
Expression exp3 = new Addition(
    new Nombre(33),
    new Addition(
        new Nombre(33),
        new Nombre(11)
    )
);

resultat = exp3.interpreter();
assert resultat == 77;

Expression exp4 = new Addition(exp1,exp3);
resultat = exp4.interpreter();
assert resultat == 398;

}}
```

ESIEE

30

Démonstration/Discussion

- Simple
- Mais

ESIEE

31

Evolution ... une galerie

- Une expression peut se calculer à l'aide d'une pile :

– Exemple : $3 + 2$ engendre
cette séquence `empiler(3)`
 `empiler(2)`
 `empiler(depiler() + depiler())`

Le résultat se trouve (ainsi) au sommet de la pile

→ **Nouvel entête de la méthode interpreter**

ESIEE

32

Evolution ...

```
public abstract class Expression{  
    abstract public void interpreter(PileI p);  
}
```

– Mais ... encore !

Cette nouvelle méthode engendre une

modification de toutes les classes !!

ESIEE

33

Evolution ... bis

- **L'interprétation d'une expression utilise une mémoire**

- Le résultat de l'interprétation est en mémoire

→ **Nouvel entête de la méthode interpreter**

```
public abstract class Expression{  
    abstract public void interpreter(Memoire p);  
}
```

→ mêmes critiques : modification de toutes les classes

.....

→ **Encore une modification de toutes les classes !**, la réutilisation prônée par l'usage des patrons est plutôt faible ...

ESIEE

34

Evolution... ter ... non merci !

- mêmes critiques : modification de toutes les classes
- Encore une modification de toutes les classes !, la réutilisation prônée par l'usage des patrons est de plus en plus faible ...
- Design pattern : pourquoi faire, utile/inutile
- Existe-t-il un patron afin de prévenir toute évolution future ?
du logiciel ...

ESIEE

35

Le pattern Visiteur vient à notre secours

- Objectifs :
 - Il faudrait pouvoir effectuer de multiples interprétations de la même structure sans aucune modification du Composite

- → Patron Visiteur

Un visiteur par interprétation

Dans la famille des itérateurs avez-vous le visiteur ...

ESIEE

36

Patron visiteur

- L'utilisateur de la classe Expression devra proposer ses Visiteurs
 - VisiteurDeCalcul, VisiteurDeCalculAvecUnePile
 - Le problème change de main...

- → Ajout de cette méthode, ce sera la seule modification Du composite

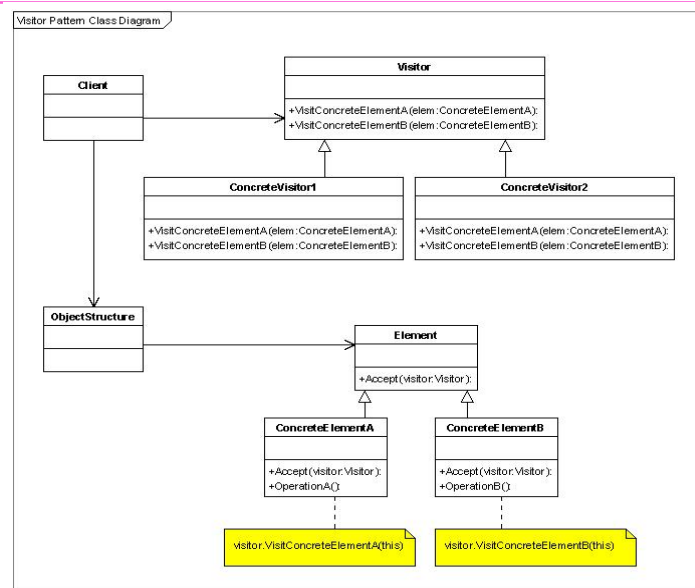
```
public abstract class Expression{  
    abstract public <T> T accepter(Visiteur<T> v);  
}
```

- → emploi de la généricité, pour le type retourné

ESIEE

37

Le diagramme UML à l'origine



- Lisible ?

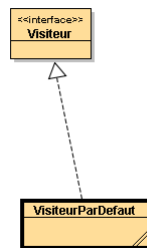
Created with Poseidon for UML Community Edition. Not for Commercial Use.

ESIEE

38

Le pattern Visiteur une méthode par feuille*

```
public abstract interface Visiteur<T>{  
    public abstract T visiteNombre(Nombre n);  
    public abstract T visiteAddition(Addition a);  
}  
  
public class VisiteurParDefaut<T> implements Visiteur<T>{  
    public T visiteNombre(Nombre n) {return n;}  
    public T visiteAddition(Addition a) {return a;}  
}
```



*feuille concrète
du composite

ESIEE

39

La classe Expression et Binaire une fois pour toutes !

```
public abstract class Expression{  
    abstract public <T> T accepter(Visiteur<T> v);  
}  
  
public abstract class Binaire extends Expression{  
    protected Expression op1;  
    protected Expression op2;  
  
    public Binaire(Expression op1, Expression op2){  
        this.op1 = op1;  
        this.op2 = op2;  
    }  
    public Expression op1(){return op1;}  
    public Expression op2(){return op2;}  
    abstract public <T> T accepter(Visiteur<T> v);  
}
```

ESIEE

40

La classe Nombre et Addition une fois pour toutes

```
public class Nombre extends Expression{
    private int valeur;
    public Nombre(int valeur){
        this.valeur = valeur;
    }
    public int valeur(){ return valeur;}

    public <T> T accepter(Visiteur<T> v){
        return v.visiteNombre(this);
    }
}

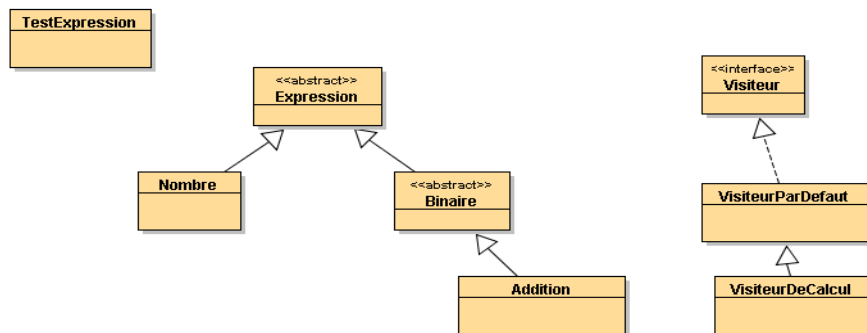
public class Addition extends Binaire{
    public Addition(Expression op1, Expression op2){
        super(op1, op2);
    }

    public <T> T accepter(Visiteur<T> v){
        return v.visiteAddition(this);
    }
}
```

ESIEE

41

Le Composite a de la visite



ESIEE

42

Le VisiteurDeCalcul

```
public class VisiteurDeCalcul
    extends VisiteurParDefaut<Integer>{

    public Integer visiteNombre(Nombre n) {
        return n.valeur;
    }

    public Integer visiteAddition(Addition a) {
        Integer i1 = a.op1().accepter(this);
        Integer i2 = a.op2().accepter(this);
        return i1 + i2;
    }
}
```

ESIEE

43

La classe TestExpression

```
public class TestExpression{

    public static void main(String[] args){
        Visiteur vc = new VisiteurDeCalcul();
        Expression exp1 = new Nombre(321);
        System.out.println(" resultat exp1 : " + exp1.accepter(vc));

        Expression exp2 = new Addition(
            new Nombre(33),
            new Nombre(33)
        );
        System.out.println(" resultat exp2 : " + exp2.accepter(vc));

        Expression exp3 = new Addition(
            new Nombre(33),
            new Addition(
                new Nombre(33),
                new Nombre(11)
            )
        );
        System.out.println(" resultat exp3 : " + exp3.accepter(vc));

        Expression exp4 = new Addition(exp1,exp3);
        System.out.println(" resultat exp4 : " + exp4.accepter(vc));
    }
}
```

BlueJ: Terminal Window

Options

```
resultat exp1 : 321
resultat exp2 : 66
resultat exp3 : 77
resultat exp4 : 398
```

ESIEE

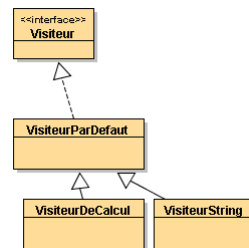
44

Le Composite a une autre visite

```
public class VisiteurString
    extends VisiteurParDefaut<String>{

    public String visiteNombre(Nombre n) {
        return Integer.toString(n.valeur());
    }

    public String visiteAddition(Addition a) {
        String i1 = a.op1().accepter(this);
        String i2 = a.op2().accepter(this);
        return "(" + i1 + " + " + i2 + ")";
    }
}
```



ESIEE

45

La classe TestExpression re-visitée

```
public class TestExpression{

    public static void main(String[] args){
        Visiteur<Integer> vc = new VisiteurDeCalcul();
        Visiteur<String> vs = new VisiteurString();
        Expression exp1 = new Nombre(321);
        System.out.println(" resultat exp1 : " + exp1.accepter(vs) + " = " +
            exp1.accepter(vc));

        Expression exp2 = new Addition(new Nombre(33),new Nombre(33));
        System.out.println(" resultat exp2 : " + exp2.accepter(vs) + " = " +
            exp2.accepter(vc));

        Expression exp3 = new Addition(
            new Nombre(33),
            new Addition(new Nombre(33),new Nombre(11))
        );
        System.out.println(" resultat exp3 : " + exp3.accepter(vs) + " = " +
            exp3.accepter(vc));

        Expression exp4 = new Addition(exp1,exp3);
        System.out.println(" resultat exp4 : " + exp4.accepter(vs) + " = " +
            exp4.accepter(vc));
    }
}
```

BlueJ: Terminal Window

Options

```

resultat exp1 : 321 = 321
resultat exp2 : (33 + 33) = 66
resultat exp3 : (33 + (33 + 11)) = 77
resultat exp4 : (321 + (33 + (33 + 11))) = 398
    
```

ESIEE

46

Le Composite pourrait avoir d'autres visites

```
public class AutreVisiteur extends VisiteurParDefaut<T>{

    public T visiteNombre(Nombre n) {
        // une implémentation
    }

    public T visiteAddition(Addition a) {
        // une implémentation
    }
}
```

ESIEE

47

Le pattern Visiteur

- **Contrat rempli :**
 - Aucune modification du composite :-> couplage faible entre la structure et son analyse
 - Tout type de visite est à la charge du client :
 - tout devient possible ...

Mais

- Convient aux structures qui n'évoluent pas ou peu
 - Une nouvelle feuille du pattern Composite engendre une nouvelle redéfinition de tous les visiteurs

- **Alors à suivre...**

ESIEE

48

Discussion

- Composite
- Interpréteur
- Visiteur
- Itérateur, est-ce un oubli ?
 - Voir en annexe
 - (prévoir un aspirine ... code extrait de *Tête la première*, migraine en vue)

ESIEE

49

Conclusion

ESIEE

50

Annexes

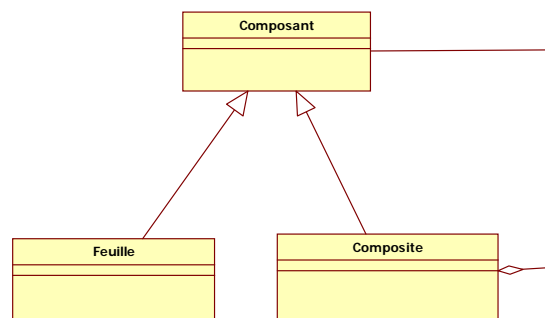
- **Patron Composite et parcours**

- Souvent récursif, « intra-classe »
- Avec un itérateur
 - Même recette ...
 - Une pile mémorise l'itérateur de chaque classe « composite »
- Avec un visiteur

ESIEE

51

Composite et Iterator



- **Structure récursive « habituelle »**

ESIEE

52

- **Classe Composite : un schéma**

```
public class Composite
    extends Composant implements Iterable<Composant>{
    private List<Composite> liste;

    public Composite(...){
        this.liste = ...
    }
    public void ajouter(Composant c){
        liste.add(c);
    }

    public Iterator<Composant> iterator(){
        return new CompositeIterator(liste.iterator());
    }
}
```

ESIEE

53

CompositeIterator : comme sous-classe

```
private
class CompositeIterator
    implements Iterator<Composant>{

    // une pile d'itérateurs,
    // un itérateur par composite
    private Stack<Iterator<Composant>> stk;

    public CompositeIterator (Iterator<Composant> iterator){
        this.stk = new Stack<Iterator<Composant>>();
        this.stk.push(iterator);
    }
}
```

ESIEE

54

next

```
public Composant next(){
    if(hasNext()){
        Iterator<Composant> iterator = stk.peek();
        Composant cpt = iterator.next();
        if(cpt instanceof Composite){
            Composite gr = (Composite)cpt;
            stk.push(gr.liste.iterator());
        }
        return cpt;
    }else{
        throw new NoSuchElementException();
    }
}

public void remove(){
    throw new UnsupportedOperationException();
}
```

ESIEE

55

hasNext

```
public boolean hasNext(){
    if(stk.empty()){
        return false;
    }else{
        Iterator<Composant> iterator = stk.peek();
        if( !iterator.hasNext()){
            stk.pop();
            return hasNext();
        }else{
            return true;
        }
    }
}
```

ESIEE

56

Un test unitaire possible

```
public void testIterator (){
    try{
        Composite g = new Composite();
        g.ajouter(new Composant());
        g.ajouter(new Composant());
        g.ajouter(new Composant());
        Composite g1 = new Composite();
        g1.ajouter(new Composant());
        g1.ajouter(new Composant());
        g.ajouter(g1);

        for(Composite cpt : g){ System.out.println(cpt);}

        Iterator<Composite> it = g.iterator();
        assertTrue(it.next() instanceof Composant);
        assertTrue(it.next() instanceof Composant);
        assertTrue(it.next() instanceof Composant);
        assertTrue(it.next() instanceof Groupe);
        // etc.
    }
}
```

ESIEE

57

Discussions annexes

- Itérateur compliqué ?
- Le visiteur tu préféreras

ESIEE

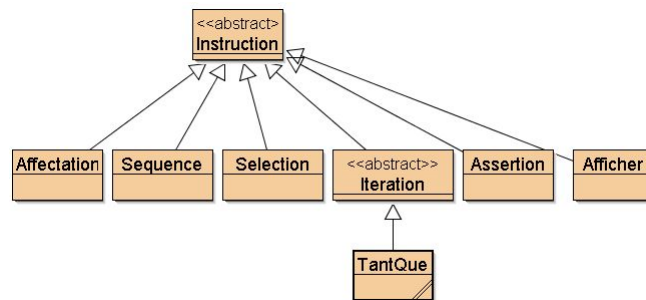
58

Annexe : Un petit langage

- Composite pour un tout petit langage impératif

- Instructions

- Affectation, Séquence, Sélection, Itération
 - Diverses : Assertion, Afficher

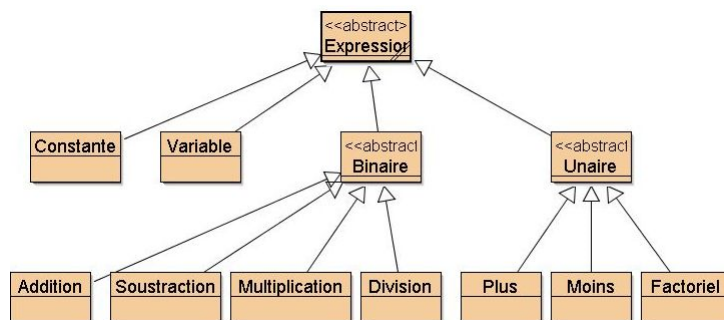


ESIEE

59

Expressions

- Composite Expressions arithmétiques (cf. ce support)

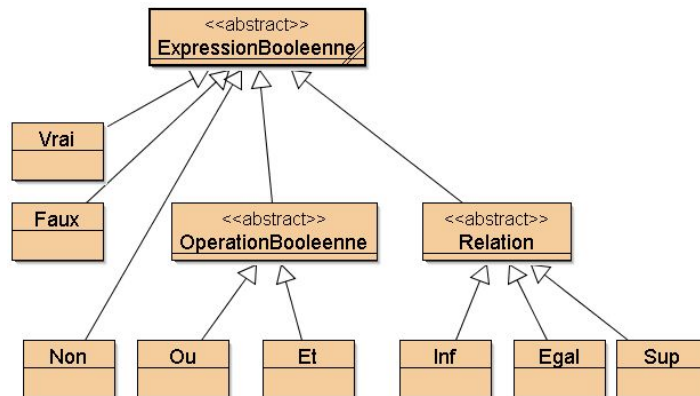


ESIEE

60

Expressions

- Composite Expressions booléennes



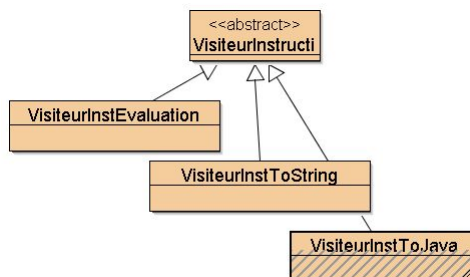
ESIEE

61

Visiteurs

- Visite d'une instance d'un composite

- Pour une conversion en String
- Pour une évaluation
- Pour une conversion en source Java
- ...



ESIEE

62

Exemple : évaluation de Factoriel !!!

```
public void testFactoriel(){
    Contexte m = new Memoire();
    Variable x = new Variable(m,"x",5);
    Variable fact = new Variable(m,"fact",1);

    VisiteurExpression ve = new VisiteurEvaluation(m);
    VisiteurExpressionBooleenne vb = new VisiteurBoolEvaluation(ve);
    VisiteurInstruction vi = new VisiteurInstEvaluation(ve,vb);

    Instruction i =
        new TantQue(
            new Sup(x,new Constante(1)),
            new Sequence(
                new Affectation(fact,new Multiplication(fact,x)),
                new Affectation(x,new Soustraction(x,new Constante(1))))
        );

    i.accepter(vi);
    assertTrue(" valeur erronée", m.lire("fact")==fact(5)); // ← vérification
}
```

ESIEE

63

Vérification en Java factoriel...

```
private static int fact(int n){
    if(n==0) return 1;
    else return n*fact(n-1);
}

private static int fact2(int x){
    int fact = 1;
    while(x > 1){
        fact = fact * x;
        x = x -1;
    }
    return fact;
}
```

ESIEE

64

Exemple suite, un autre visiteur qui génère du source Java

```
public void test_CompilationDeFactoriel(){
    Contexte m = new Memoire();
    Variable x = new Variable(m,"x",5);
    Variable fact = new Variable(m,"fact",1);

    Instruction inst = // idem transparent précédent
        new TantQue(
            new Sup(x,new Constante(1)),
            new Sequence(
                new Affectation(fact,new Multiplication(fact,x)),
                new Affectation(x,new Soustraction(x,new Constante(1)))
            )
        );

    VisiteurExpression<String> ves = new VisiteurInfixe(m);
    VisiteurExpressionBooleenne<String> vbs = new VisiteurBoolToJava(ves);
    VisiteurInstruction<String> vs = new VisiteurInstToJava(ves,vbs,4);

    // vérification par une compilation du source généré
}
```

ESIEE

65

Le source généré

```
package question3;

public class Fact{

    public static void main(String[] args)throws Exception{
        int fact=1;
        int x=5;

        while(x > 1){
            fact = (fact * x) ;
            x = (x - 1);
        }
    }

}
```

ESIEE

66

Exemple suite, un autre visiteur qui génère du bytecode Java

```
public void test_CompilationDeFactoriel(){
    Contexte m = new Memoire();
    Variable x = new Variable(m,"x",5);
    Variable fact = new Variable(m,"fact",1);

    Instruction inst = // idem transparent précédent
        new TantQue(
            new Sup(x,new Constante(1)),
            new Sequence(
                new Affectation(fact,new Multiplication(fact,x)),
                new Affectation(x,new Soustraction(x,new Constante(1)))
            )
        );

    Code code = new Code("TestsFactoriel", m);
    VisiteurExprJasmin vej = new VisiteurExprJasmin(m,code);
    VisiteurBoolJasmin vbj = new VisiteurBoolJasmin(vej);
    VisiteurInstJasmin vij = new VisiteurInstJasmin(vej,vbj);

    // vérification par l'exécution de l'assembleur Jasmin
}
```

ESIEE

67

Le bytecode généré compatible jasmin jasmin.sourceforge.net/

```
.class public TestsFactoriel
.super java/lang/Object
.method public static main([Ljava/lang/String;)V
.limit stack 3                                #_25:
.limit locals 3                                ifeq #_43
    ldc 1                                       iload 1
    istore 1                                   iload 2
    ldc 5                                       imul
    istore 2                                   istore 1
#_14:                                           iload 2
    iload 2                                   iconst_1
    iconst_1                                   isub
    if_icmple #_23                             istore 2
    iconst_1                                   goto #_14
    goto #_25                                #_43:
#_23:                                           return
    iconst_0
.end method
```

ESIEE

68

Un détail : la visite pour TantQue

```
public Integer visite(TantQue tq){
    int start = code.currentPosition();
    code.addLabel(start);
    int hc = tq.cond().accepter(this.vbj);
    code.add("ifeq");
    int jumpIfAddr = code.currentPosition();
    code.add("labelxxxxx");
    int h = tq.il().accepter(this);
    code.add("goto");
    int jumpAddr = code.currentPosition();
    code.add("labelxxxxx");
    code.setLabel(jumpAddr, start);
    code.setLabel(jumpIfAddr, code.currentPosition());
    code.addLabel(code.currentPosition());
    return hc + h;
}
```

ESIEE

69

Démonstration

Questions ...

ESIEE

70