Cours Java_III

1998, Cnam Paris
jean-michel Douin, douin@cnam.fr
Version du 22 Octobre 2002

Notes de cours java : le langage : plusieurs classes, héritage, polymorphisme

Sommaire

- Classe dérivée syntaxe
- Héritage
- Affectation
- Liaison dynamique
- Héritage d'interface
- exemples de "pattern"

Bibliographie utilisée

- The Java Handbook, Patrick Naughton. Osborne McGraw-Hill.1996. http://www.osborne.com
- Thinking in Java, Bruce Eckel, http://www.EckelObjects.com
- http://java.sun.com/docs/books/jls/
- http://java.sun.com/docs/books/tutorial.html
- http://hillside.net/patterns/
- http://www.eli.sdsu.edu/courses/spring98/cs635/notes/index.html
- [GHJV95] DESIGN PATTERNS, catalogue de modèles de conception réutilisables, E.Gamma, R.Helm,R.Johnson et J.Vlissides. Thomson publishing.1995
- ttp://www.enteract.com/~bradapp/docs/patterns-intro.html

Concepts de l'orienté objet

- Classe et objet (instance)
- Etat d'un objet et encapsulation
- Comportement d'un objet et méthodes
- Héritage
- polymorphisme
  - ad'hoc
  - d'inclusion
  - paramétrique

Héritage et classification

- définir une nouvelle classe en ajoutant de nouvelles fonctionalités à une classe existante
  - ajout de nouvelles fonctions
  - ajout de nouvelles données
  - redéfinition de certaines propriétés héritées

- Classification en langage naturel

- les carrés sont des polygones réguliers
- les polygones réguliers sont des objets Java
  (en java java.lang.Object est la racine de toutes classe)

Polymorphisme

- **Polymorphisme ad'hoc**

  surcharge

- **Polymorphisme d'inclusion**

  est fondé sur la relation d'ordre partiel entre les types, relation induite par l'héritage. si le type B est inférieur selon cette relation au type A alors on peut passer un objet de type B à une méthode qui attend un paramètre de type A, le choix de la méthode est résolu dynamiquement en fonction du type de l'objet receveur

- **Polymorphisme paramètrique**

  généricité

  extrait de M Baudouin-Lafon. La Programmation Orientée Objet. ed. Armand Colin

Classe : Syntaxe

- class NomDeClasse **extends NomDeLaSuperClasse{**

```
type  variableDeClasse1;
type  variableDeClasse2;
type  variableDeClasseN;

type nomDeMethodeDeClasse1( listeDeParametres) {

}
type nomDeMethodeDeClasse2( listeDeParametres) {

}
type nomDeMethodeDeClasseN( listeDeParametres) {

}
```

- **}**

Héritage exemple

Les polygones réguliers sont des "objets java"

- class PolygoneRegulier extends java.lang.Object{ ....}

Les carrés sont des polygones réguliers

- class Carre **extends** PolygoneRegulier{ ...}

Les carrés en couleur sont des carrés

- class CarreEnCouleur **extends** Carre {.....}

Héritage

- **Les instances des classes dérivées effectuent :**
- **Le cumul des données d'instance,**
- **Le "cumul" du comportement,**

- le comportement des instances issu de la classe dérivée

dépend de :

La surcharge des méthodes (la signature est différente)

et du masquage des méthodes(la signature est identique)

Exemple la classe Carre

- **class Carre extends PolygoneRegulier{**
- ```
  // pas de champ d'instance supplémentaire
  Carre( int longueur){
    nombreDeCotes = 4;
    longueurDuCote = longueur;
  }


  // masquage de la méthode  PolygoneRegulier.surface()
  int surface(){
   return longueurDuCote* longueurDuCote;
  }

  String toString(){
   return "<4,"+ longueurDuCote +">";
  }
  ```
- **}**
  ```

super

- Appel d'une méthode de la super classe
- Appel du constructeur de la super classe

```
class Carre extends PolygoneRegulier{

    Carre( int longueur){
       super(4,longueur);
    }

    int surface(){
      return super.surface();
    }
```

Création d'instances et affectation

- **Création d'instances et création**

```
Carre  c1 = new Carre(100);
Carre  c2 = new Carre(10);
PolygoneRegulier p1 = new PolygoneRegulier(4,100);
```

- **Affectation**

```
c1 = c2;                // synonymie, c2 est un autre nom pour c1
```

- **Affectation polymorphe**

```
p1 = c1;
```

- **Affectation et changement de classe**

```
c1 = (Carre) p1;  // Hum, Hum ...
```

Liaison dynamique

- **Sélection de la méthode en fonction de l'objet receveur**
- **type déclaré / type constaté à l'exécution**

// classe déclarée

- **PolygoneRegulier p1 = new PolygoneRegulier(5,100);**
- **Carre c1 = new Carre(100);**

- **int s = p1.surface();** // la méthode surface() de PolygoneRegulier
- **p1 = c1;** // affectation polymorphe
- **s = p1.surface();** // la méthode surface() de Carre est sélectionnée
- la recherche de la méthode s'effectue uniquement dans l'ensemble des méthodes masquées associé à la classe dérivée

Récréation...http://cedric.cnam.fr/personne/barthe/cours-oo.html

```
class A{
  void m(A a){  System.out.println(" m de A"); }
  void n(A a){System.out.println(" n de A"); }
}

public class B extends A{
  public static void main(String args[]){
    A a = new B();
    B b = new B();

    a.m(b);
    a.n(b);
  }

  void m(A a){  System.out.println(" m de B"); }
  void n(B b){  System.out.println(" n de B"); }
}
```

- **Quelle est la trace d'exécution ?**

m de B
n de A

Récréation : une explication

- **mécanisme de liaison dynamique en Java :**

  La liaison dynamique effectue la sélection d'une méthode en fonction du type constaté de l'objet receveur, la méthode doit appartenir à l'ensemble des méthodes masquées,

  (la méthode est masquée dans l'une des sous-classes, si elle a exactement la même signature)

  Sur l'exemple récréatif, nous avons uniquement dans la classe B la méthode m(A a) masquée

  en conséquence :

  A a = new B(); *// a est de type déclaré A, mais constaté B*

  a.m --> sélection de ((B)a).m(...) car m est bien masquée

  a.n --> sélection de ((A)a.n(...) car n n'est pas masquée dans B

  Choix d'implémentation : vitesse d'exécution / sémantique ...

Génie logiciel et P.O.O. (Récréation suite)

- class A{
- void m(A a){ System.out.println(" m de A"); }
- void n(A a){ System.out.println(" n(a) de A"); }
- void n(B b){ System.out.println(" n(b) de A"); } // ajout de cette méthode
- }
- **Seule la classe A est recompilée, la classe B est à nouveau exécutée**
- *DOS> javac A.java*
- *DOS>java B*

- **Quelle est la trace d'exécution ?** B əp u• B əp m•

- **Lecture conseillée**
- **http://java.sun.com/docs/books/jls/html chapitre 13, Binary Compatibility**

- **Gestion de projets et internet ...**

retour sur la classe incomplète dite abstraite

- **Une classe partiellement définie, dans laquelle certaines méthodes sont laissées à la responsabilité des sous-classes**
- **pas de création d'instances possible,**
- **Affectation possible d'une référence de classe incomplète par une instance de classe dérivée**
- **la classe dérivée reste abstraite si toutes les implantations ne sont pas effectuées**

- **exemple :**

```
abstract Figure {
   ...
}
class Losange extends FigureGeometrique{
}

Figure f = new Losange();  ..
```

Interface et héritage

- *interface I extends I1,I2,I3 { .... }*

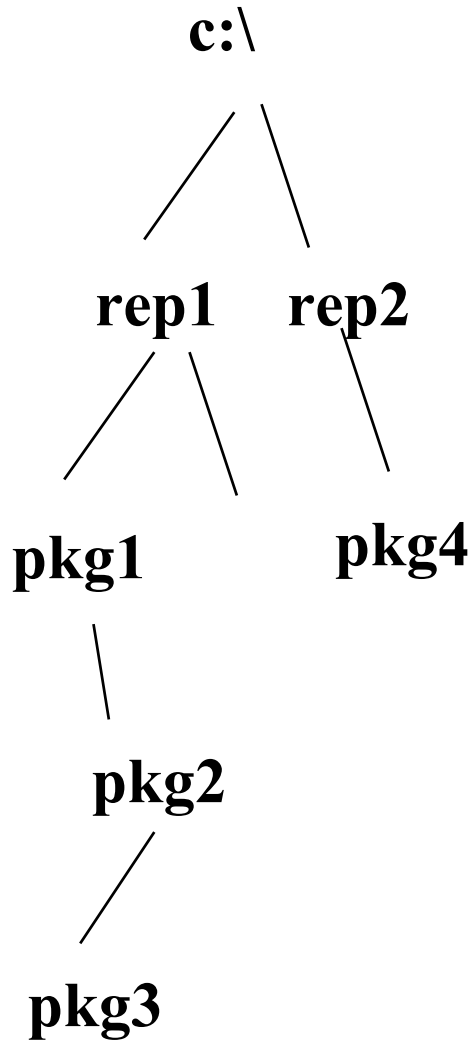
- public interface Transformable extends Scalable, Rotateable, Reflectable{}
- public interface DrawingObject extends Drawable, Transformable{}
- public class Shape implements DrawingObject{....}

- **interface comme marqueur**
- public interface java.lang.Cloneable{ /** vide **/ }
  --> usage de clone possible, si la classe implémente cet interface
- public interface java.io.Serializable{ /** vide **/ }
  --> les instances pourront être sérialisées..

  test de la bonne implémentation par **instanceof**

Package, bis (java_II)

- **Fonction**
  - **Unité logique par famille de classes**
  - **découpage hiérarchique des paquetages**
    - **(ils doivent être importés explicitement sauf java.lang)**
- **Buts**
  - **espace de noms**
  - **restriction visibilité**
- **Instructions**
  - ***package pkg1[.pkg2[.pkg3];***
    - les noms sont en minuscules
    - c 'est la première instruction du source java
  - ***import pkg1[.pkg2[.pkg3].(nomdeclasse|*);***
  - **liés aux options de la commande de compilation**
  - **dos> javac -classpath .;c:\rep1;c:\rep2**



Exemple

```
package tp7.q1;

public class Exemple{

  public void m(){
    System.out.println(" methode m");
  }
}
```

```
DOS> javac –classpath . tp7/q1/Exemple.java
```

Exemple

```
package tp7.q1;

public class Exemple{

  public void m(){
    System.out.println(" methode m");
  }
}
```

```
DOS> javac –classpath . tp7/q1/Exemple.java
```

Exemple2

```
package tp7.q2;

import tp7.q1.Exemple;

public class Exemple2{
  public void methode(){
  }
}

DOS> javac –classpath . Tp7/q2/*.java
```

Exemple jar, *compressÚs !!*

DOS> jar cvf Exemple.jar tp7/q1/*.class

ajoutÚ

ajout : tp7/q1/Exemple.class(entrÚe = 397) (sortie = 283)(28% compressÚs)

DOS> javac –classpath Exemple.jar tp7/q2/*.java

// un doute ➔ -verbose

DOS>javac -classpath Exemple.jar -verbose tp7/q2/*.java

```
[parsing started tp7/q2/Exemple2.java]
[parsing completed 160ms]
[loading Exemple.jar(tp7/q1/Exemple.class)]
[checking tp7.q2.Exemple2]
[loading c:\j2sdk1.4.0_01\jre\lib\rt.jar(java/lang/Object.class)]
[wrote tp7\q2\Exemple2.class]
[total 701ms]
```

Compilation javac

DOS>javac -verbose -classpath Exemple.jar -bootclasspath c:\j2sdk1.4.0_01\jre\lib\rt.jar tp7/q2/*.java

[parsing started tp7/q2/Exemple2.java]
[parsing completed 160ms]
[loading Exemple.jar(tp7/q1/Exemple.class)]
[checking tp7.q2.Exemple2]
[loading c:\j2sdk1.4.0_01\jre\lib\rt.jar(java/lang/Object.class)]
[wrote tp7\q2\Exemple2.class]
[total 681ms]

Paquetages prédéfinis

- **le paquetage java.lang.* est importé implicitement**

  ce sont les interfaces : *Cloneable, Comparable, Runnable*
  et les classes : Boolean,Byte,Character, Class, ClassLoader, Compiler,
  Double, Float, InheritableThreadLocal, Long, Math, Number,
  Object, Package, Process, Runtime, RuntimePermission,
  SecurityManager, Short, StrictMath, String, StringBuffer,
  System, Thread, ThreadGroup, ThreadLocal,
  Throwable, Void,
  toutes les classes dérivées de Exception, ArithmeticException,….
  et celles de Error, AbstractMethodError,….

- **java.awt.* java.io.* java.util.* ....**

- **→documentation du j2sdk**

Règle d'accès

	private	défaut	**protected**	**public**
même classe	oui	oui	oui	oui
même paquetage et sous-classe	non	oui	oui	oui
même paquetage et classe indépendante	non	oui	oui	oui
paquetages différents et sous-classe	non	non	oui	oui
paquetages différents et classe indépendante	non	non	non	oui

Classes internes et statiques (niveau 0)

```
package tp7.q3;
public class Exemple3{
  private static Object obj = new Object();

  public static class InterneEtStatique{
    public void methode(){
      Object o = Exemple3.obj;
      new Exemple3().methode();
      o = new Exemple3.InterneEtStatique();
      o = new Exemple3();
    }
  }
  public void methode(){
    InterneEtStatique is = new InterneEtStatique();
    is.methode();
  }
}

Exemple3$InterneEtStatique.class
Exemple3.class
```

Interface internes -> niveau 0

```
package tp7.q3;

public class Exemple4{
  private static Object obj = new Object();

  public interface Exemple4I{
    public void methode();
  }

  public void methode(){
    Exemple4I i4;
  }
}
```

```
Exemple4$Exemple4I.class
Exemple4.class
```

Classes internes et membres

```
package tp7.q3;
public class Exemple5{
  private Object obj = new Object();
  public class InterneEtMembre{
     public void methode(){
        obj = null;
        Exemple5.this.obj = null;
     }
  }

    public void methode(){
      this.new InterneEtMembre();
}
}

Exemple5$InterneEtMembre.class
Exemple5.class
```

Classes internes et membres !

```
package tp7.q3;
public class Exemple5{
  private Object obj = new Object();
  public class InterneEtMembre{
    private Object obj = new Object();
      public class InterneEtMembre2{
        private Object obj = new Object();
        public void methode(){
          Object o = this.obj;
          o = Exemple5.this.obj;
          o = InterneEtMembre.this.obj;
        }
      }
    }
    public void methode(){
      this.new InterneEtMembre();
      this.new InterneEtMembre().new InterneEtMembre2();
      Exemple5 e = new Exemple5();
      Exemple5.InterneEtMembre e1 = e.new InterneEtMembre();
    }}

Exemple5$InterneEtMembre$InterneEtMembre2.class
```

Classes internes et anonymes

```
import java.awt.Button;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class Exemple6{
  public void methode(){
    Button b = new Button("b");
    b.addActionListener( new ActionListener(){
      public void actionPerformed(ActionEvent ae){
      }
    });
  }
}

Exemple6$1.class
Exemple6.class
```

Classes internes et locales

```
import java.util.Iterator;
public class Exemple7{
  public Iterator methode(){

    class Locale implements Iterator{
      public boolean hasNext(){ return true;}
      public Object next(){return null;}
      public void remove(){}
    }
    return new Locale();
  }
}


Exemple7$1$Locale.class
Exemple7.class
```

Une petite dernière

```
import java.util.Iterator;
public class Exemple8{
  public Iterator methode(){

    return new Iterator(){
      public boolean hasNext(){ return true;}
      public Object next(){return null;}
      public void remove(){}
    }
  }
}
```

```
Exemple8$1.class
Exemple8.class
```

Exercice : La pile (2)

- Reprendre l'exercice de la pile vue en java_II en héritant maintenant de java.util.Vector,
- Prendre comme classe de test une calculette en notation "postfixée",

  3+2 -> empiler(3); empiler(2); empiler(depiler() + depiler())

- 2) substituer votre implémentation de la classe pile par la classe java.util.Stack

  http://java.sun.com/docs/books/jls/ chapitre 21.12

- 3) et éventuellement, développer (ou adapter) un IHM d'une calculette (à pile !)

  http://cedric.cnam.fr/~farinone/IAGL/IAGL2.html

  http://www.cs.bu.edu/staff/TA/gclin/class/applet_calc.html

  http://dbserv.st-and.ac.uk:8080/javawww/examples/

  .....

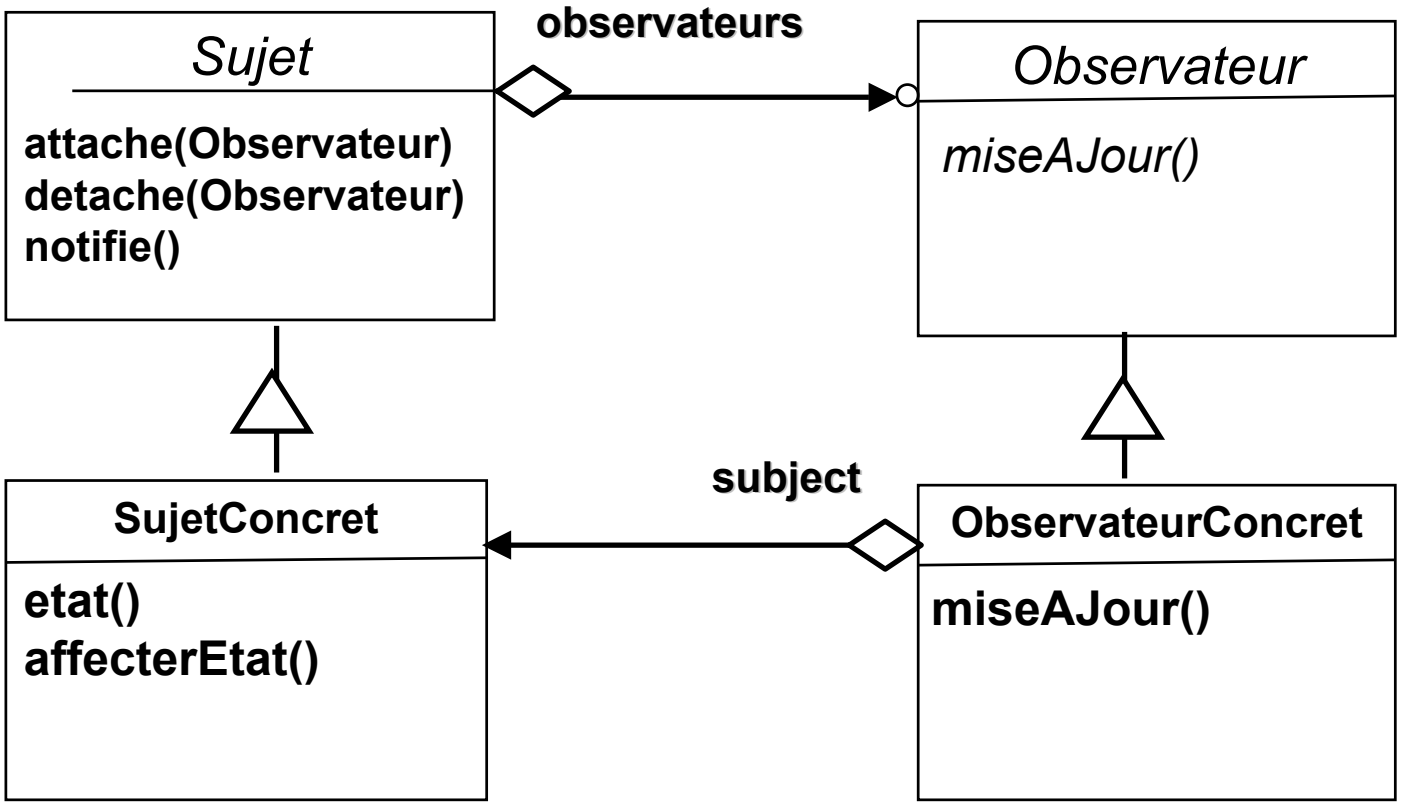
exemples de patterns

// extrait de [GHJV 95] DESIGN PATTERNS, catalogue de modèles de conception réutilisables, E.Gamma, R.Helm,R.Johnson et J.Vlissides. Thomson publishing.1995

// Pattern Observateur page 343, utilisation :

// Quand un concept a deux representations, l'une dependant de l'autre.
// Encapsuler ces
// deux representations dans des objets distincts permet de les reutiliser et de les
// modifier independamment.
// Quand la modification d'un objet necessite de modifier les autres, et que l'on ne
// sait pas combien sont ces autres
// Quand un objet doit etre capable de faire une notification a d'autres objets sans
// faire d'hypotheses sur la nature de ces objets. En d'autres termes, quand ces
// objets ne doivent pas etre fortement couples

Le "pattern" observateur en UML



Le "pattern" observateur en java(1),SujetObservé

```
interface Observateur{
  public void miseAJour(Source src);
}

abstract class SujetObserve{
  private java.util.Hashtable table = new java.util.Hashtable();
  public void attache(Observateur o){
    if (!table.contains(o)) table.put(o,o);
  }
  public void detache(Observateur o){
    table.remove(o);
  }
  public void notifie(){
    for (java.util.Enumeration e = table.elements(); e.hasMoreElements();){
      ((Observateur)e.nextElement()).miseAJour(new Source(this));
    }
  }}

class Source{ private Object obj;
  public Source(Object obj){this.obj = obj;}   public Object getSource(){return obj;}}
```

Observateur, SujetConcret

```
class Nombre extends SujetObserve{
  private int valeur;
  public Nombre(int valeur){ this.valeur = valeur;}
  public void inc(){  valeur++; notifie();}
  public int valeur(){return valeur;}
}
```

```
class Flottant extends SujetObserve{
  private float valeur;
  public Flottant(float valeur){ this.valeur = valeur;}
  public void inc(){  valeur++; notifie();}
  public float valeur(){return valeur;}
}
```

Observateur, ObservateurConcret

```
class ObservateurDeNombre implements Observateur{

  public void miseAJour(Source src){
    if (src.getSource() instanceof Nombre){
      Nombre n = (Nombre)src.getSource();
      System.out.println("instance de ObservateurDeNombre : " + n.valeur());
    }
  }
}
```

Observateur : Test (1)

```
public class Observateur0 implements Observateur{

 public static void main(String args[]) {
   Nombre n = new Nombre(1);
   ObservateurDeNombre o = new ObservateurDeNombre();
   ObservateurDeNombre o1 = new ObservateurDeNombre();
   Observateur0 obs = new Observateur0();

   // n a 3 observateurs, o, o1, obs
   n.attache(o); n.attache(o1); n.attache(obs);
   n.inc(); n.inc();

   n.detache(o); n.detache(o1);

   // obs a maintenant en charge un autre sujet observe f
   Flottant f = new Flottant(10F);
   f.attache(obs);
   f.inc();n.inc();
 }
```

Observateur : Test(2)

```
public void miseAJour(Source src){
  if (src.getSource() instanceof Flottant){
    Flottant f = (Flottant)src.getSource();
    System.out.println("un flottant est mis a jour : " + f.valeur());
  }else if (src.getSource() instanceof Nombre){
    Nombre n = (Nombre)src.getSource();
    System.out.println("un Nombre est mis a jour : " + n.valeur());
  }
 }

}
```

Observateur : les évènements en 1.1

```
import java.applet.Applet; import java.awt.Button; import java.awt.event.*;

public class ObservateurEtEvenement1_1 extends Applet {
  private Button boutonA, boutonB;
  private ObservateurDeBouton o = new ObservateurDeBouton();
  private ObservateurDeBouton o1 = new ObservateurDeBouton();
  private ObservateurDeBouton obs = new ObservateurDeBouton();

   public void init() {
     boutonA = new Button("A"); boutonB = new Button("B");

     // le bouton A a 3 observateurs (Listener) o,  o1 et obs
     boutonA.addActionListener(o);boutonA.addActionListener(o1);
     boutonA.addActionListener(obs);

     // obs a maintenant en charge un autre bouton
     boutonB.addActionListener(obs);

     add(boutonA);add(boutonB);
   }}
```

Observateur : les évènements en 1.1 suite

```
class ObservateurDeBouton implements ActionListener{

  public void actionPerformed(ActionEvent e){
    System.out.println(" evenement bouton " + e.getActionCommand() + " : " +
    this);
  }
}
```

Observateur : usage d'un "Adapter"

```
      // inscription de os observateur de souris sur l'applet
       addMouseListener(os);
       // inscription de os observateur de souris sur le bouton B
       boutonB.addMouseListener(os);

       add(boutonA);add(boutonB);
     }
  }

  class ObservateurDeSouris extends MouseAdapter{
    public void mouseExited(MouseEvent e){
      System.out.println(" mouseExited en " + e.getX() + "," + e.getX());
    }
    public void mouseClicked(MouseEvent e){
      System.out.println(" mouseClicked en " + e.getX() + "," + e.getX());
    }
  }
```

Observateur, exercice

- **Reprendre le même test ou celui de votre choix, en utilisant**
  - **la classe : java.util.Observable**
  - **et l'interface java.util.Observer**

- **lecture conseillée**
- **http://java.sun.com/docs/books/jls/ chapitres 21.7 et 21.8**

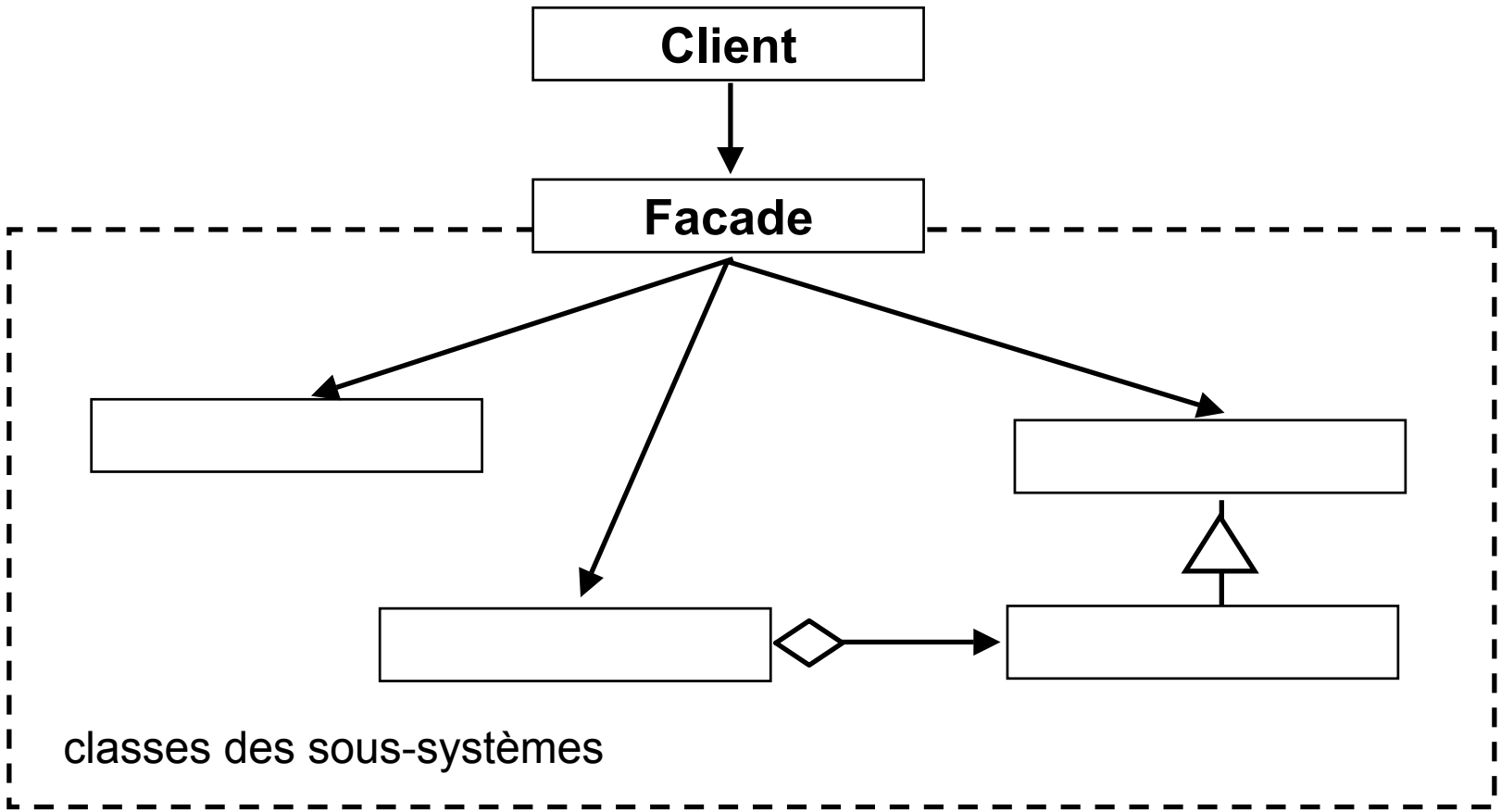
Le "pattern" Facade page 215

- **// Pattern facade, utilisation :**

- **// On souhaite disposer d'une interface simple pour un sous-systeme complexe. les sous-systemes deviennent de plus en plus complexes au fur et a mesure de leur evolution. La plupart des modeles, lorsqu'ils sont employés, engendrent des classes plus nombreuses et plus petites. Ceci rend le sous-systeme plus reutilisable et plus facile a personnaliser, mais il devient plus difficile a employer pour les classes qui n'ont pas besoin de personnalisation. Une facade propose une simple vue par defaut du sous-systeme, qui est suffisante pour la plupart des clients. Seuls les clients demandant plus de specificite devront regarder derriere la facade.**

- **lectures**
- **http://www.eli.sdsu.edu/courses/spring98/cs635/notes/facade/facade.html**
- **http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/patterns/**

Le "pattern" facade en UML



Pattern facade, exercice

- **Développer cette "pattern", prendre en exemple la calculatrice .... (en notation polonaise inversée...)**

Pattern Singleton, page 149

```
// S'il doit n'y a exactement qu'une instance d'une classe, qui, de plus,
//   doit etre accessible aux clients en un point bien determine
// Si l'instance unique doit-etre extensible par derivation en sous-classe, et
//   si l'utilisation d'une instance etendue doit etre permise aux clients, sans
//   qu'ils aient a modifier leur code


final class SingletonPattern{                // schema de la classe Singleton
  public static SingletonPattern instance(){
    if(uniqueInstance == null){
      uniqueInstance = new SingletonPattern();
        }
      return uniqueInstance;
  }

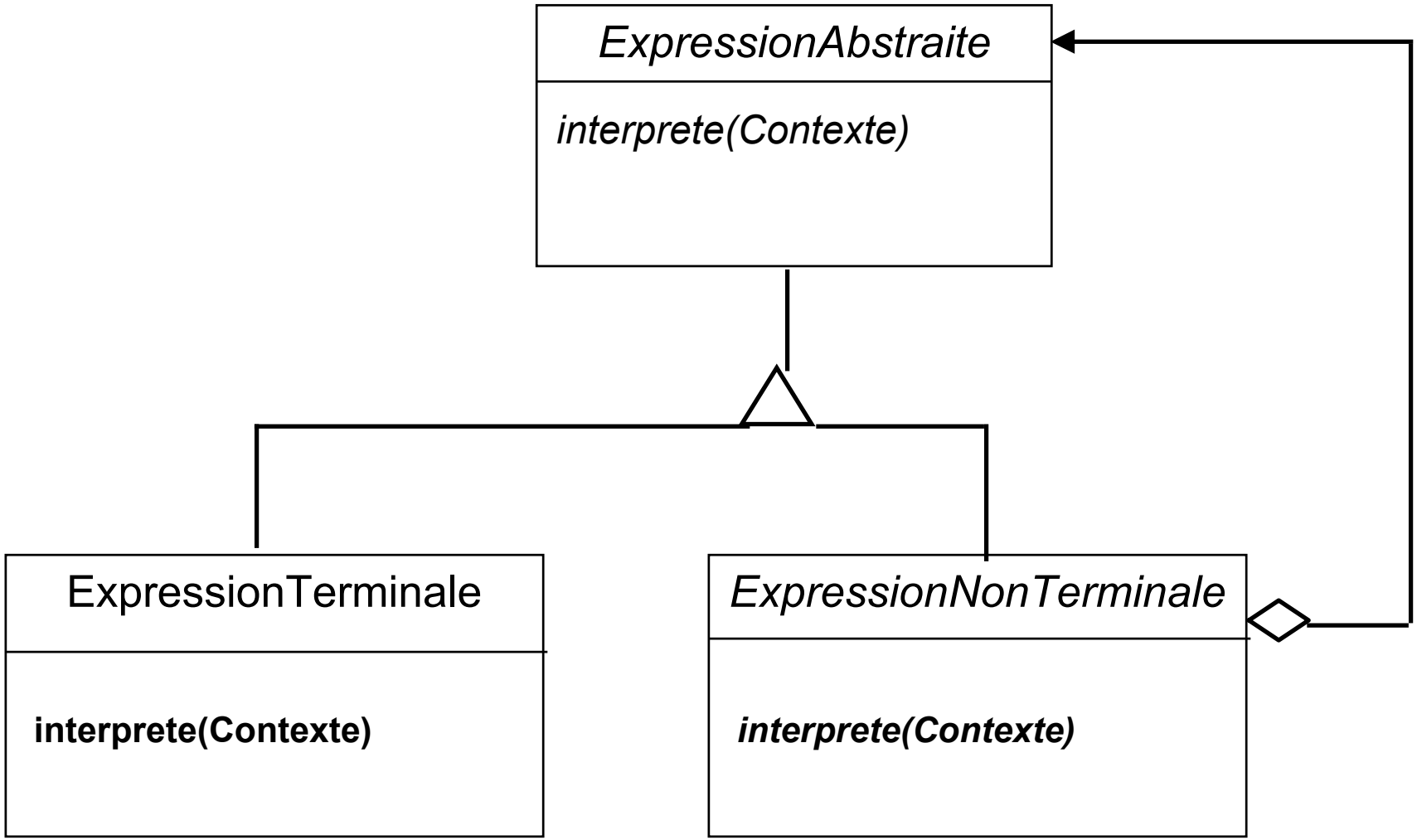

  private static SingletonPattern uniqueInstance = null;
  private SingletonPattern(){}
}
```

Pattern Singleton exemple

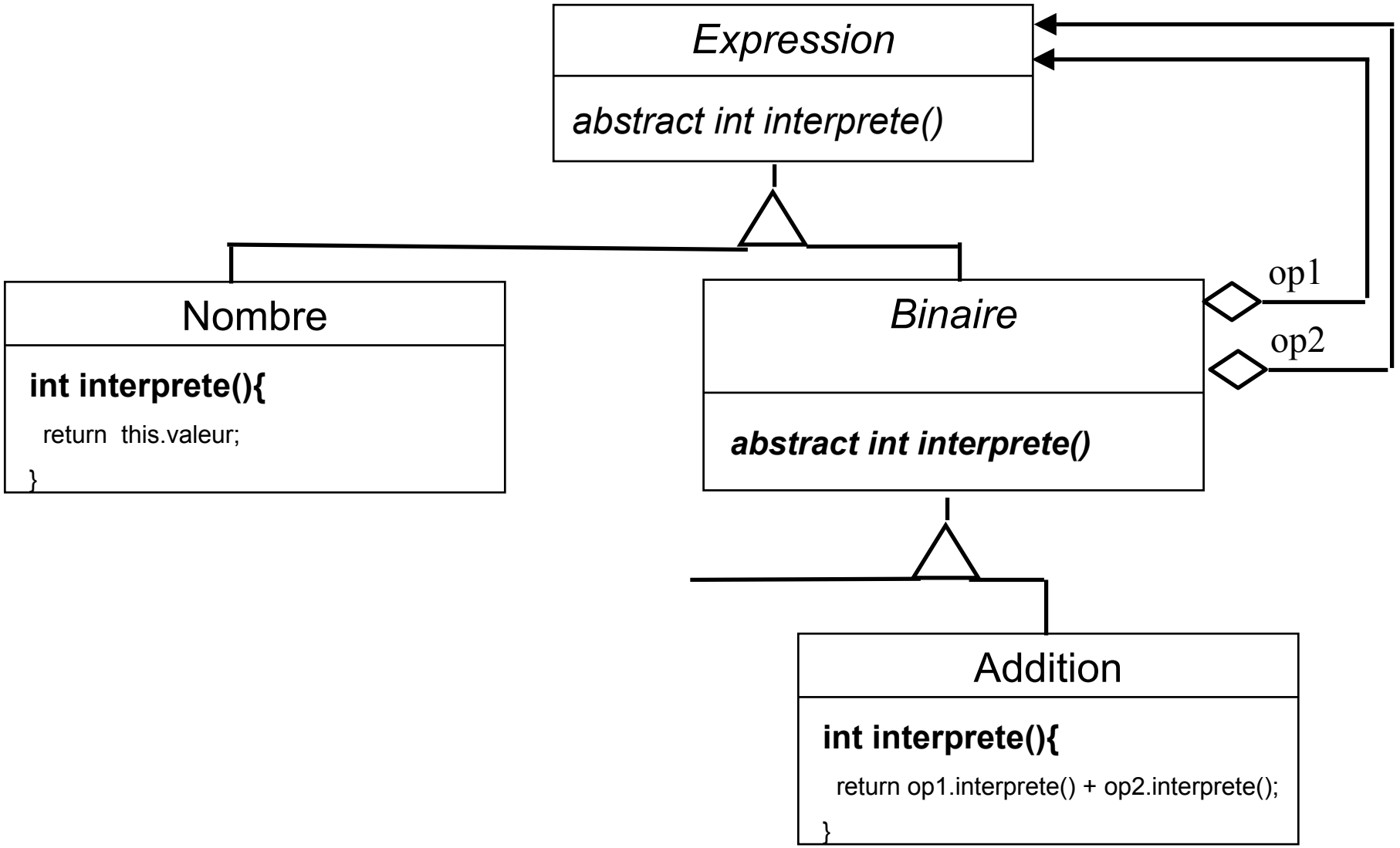
```
final class UnFlottant{
    public static UnFlottant instance(){
        if(uniqueInstance == null){
            uniqueInstance = new UnFlottant();
        }
        return uniqueInstance;
    }
    public double valeur(){
        return laValeur;
    }
    public void affecter(double d){
        laValeur = d;
    }

    private double laValeur;
    private static UnFlottant uniqueInstance=null;
    private UnFlottant(){}
}
```

Pattern Interpreteur, page 285



Pattern Interpreteur, Expression



Pattern Interpreteur, Expression arithmétique

```
/*public*/ abstract class Expression{
  public abstract int interprete();
}

/*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 abstract int interprete();
}
```

Pattern Interpreteur (suite)

```
/*public*/ class Addition extends Binaire{
  public Addition(Expression op1, Expression op2){
    super(op1,op2);
  }
  public int interprete(){
    return op1.interprete() + op2.interprete();
  }
}

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

Pattern Interpreteur, le test

```
public class InterpreteurExpression0{

  public static void main(String args[]){

    Expression exp = new Addition(                  // 3 + 2 + 4
                              new Nombre(3),
                              new Addition(
                                   new Nombre(2),
                                   new Nombre(4)
                              )
                             );

    System.out.println("exp.interprete() : " + exp.interprete());
  }
}
```

Exercice : Expression et machine à pile

- **Evaluation d' expression à l'aide d'une machine à pile**

  Prendre la pile développée en java_II ou la classe Contexte ci dessous;
  Cette pile est le contexte transmis lors de l'appel de la méthode interprete

  La classe abstraite Expression de vient :

```
public abstract class Expression{
  public abstract void interprete(Contexte ctxt);
}
```

```
public class Contexte{
  java.util.Stack pile;
  Contexte(){pile = new java.util.Stack(); }
  public void empiler(int i){pile.push(new Integer(i)); }
  public int depiler(){return ((Integer)pile.pop()).intValue(); }
  public int sommet(){return ((Integer)pile.peek()).intValue(); }
}
```

Exercice : le test

```
public class InterpreteurExpression{
  public static void main(String args[]){
       /** exp = 3 + 2 * 4  */
      Expression exp = new Addition( new Nombre(3),
                                     new Multiplication(new Nombre(2),
                                                        new Nombre(4)));
      Contexte pile = new Contexte();
      exp.interprete(pile);
      System.out.println("pile.sommet() : " + pile.sommet());
  }
}
```

divers: un exemple de sérialisation

```
// la liste de java_II
public class Queue implements UnboundedBuffer, Serializable {
        class Node implements Serializable{ }

       // ajout de ces 2 méthodes
        public void save(String fileName){  .... }
        public Queue restore(String fileName){ ....}
}
```

divers: un exemple de sérialisation

```
public void save(String fileName){
    try{
      ObjectOutputStream out;
      out = new ObjectOutputStream(new FileOutputStream(fileName));
      out.writeObject(this);
      out.flush();
      out.close();
    }catch (IOException e){ e.printStackTrace();} }

 public Queue restore(String fileName){
    Queue q=null;
    try{
      ObjectInputStream in;
      in = new ObjectInputStream(new FileInputStream(fileName));
      q = (Queue) in.readObject();
      in.close();
    }catch (IOException e){
    }catch (ClassNotFoundException e){}
    return q; }
```