
Cours 1 : Introduction aux patrons

jean-michel Douin, douin au cnam point fr
version : 3 Septembre 2009

Notes de cours,

elles ne remplacent la lecture d'ouvrages ou de tutoriels sur ce sujet cf. bibliographie

ESIEE

1

Sommaire

- Conception à l'aide de patrons (*design pattern*).
- BlueJ : www.patterncoder.org

ESIEE

2

Principale bibliographie utilisée pour ces notes

- [Grand00]
 - Patterns in Java le volume 1
<http://www.mindspring.com/~mgrand/>
- [head First]
 - Head first : <http://www.oreilly.com/catalog/hfdesignpat/#top>
- [DP05]
 - L'extension « Design Pattern » de BlueJ : <http://hamilton.bell.ac.uk/designpatterns/>
 - Ou bien en <http://www.patterncoder.org/>
- [Liskov]
 - Program Development in Java, Abstraction, Specification, and Object-Oriented Design, B.Liskov avec J. Guttag Addison Wesley 2000. ISBN 0-201-65768-6
- [divers]
 - Certains diagrammes UML : <http://www.dofactory.com/Patterns/PatternProxy.aspx>
 - informations générales <http://www.edlin.org/cs/patterns.html>

ESIEE

3

Java : les objectifs, rappel

- « Simple »
- « sûr »
- Orienté Objet
- Robuste
- Indépendant d'une architecture
- Environnement riche
- Technologie « Transversale »

- → un langage de programmation

ESIEE

4

Design Pattern

- En quelques mots ...
- Moyen d'accomplir quelque chose
- Une méthode éprouvée, réutilisée
- Un code simple, « propre et peu perfectible »
- Un jargon pour discuter du savoir faire
- intra discipline ...

ESIEE

5

Pattern pourquoi ?

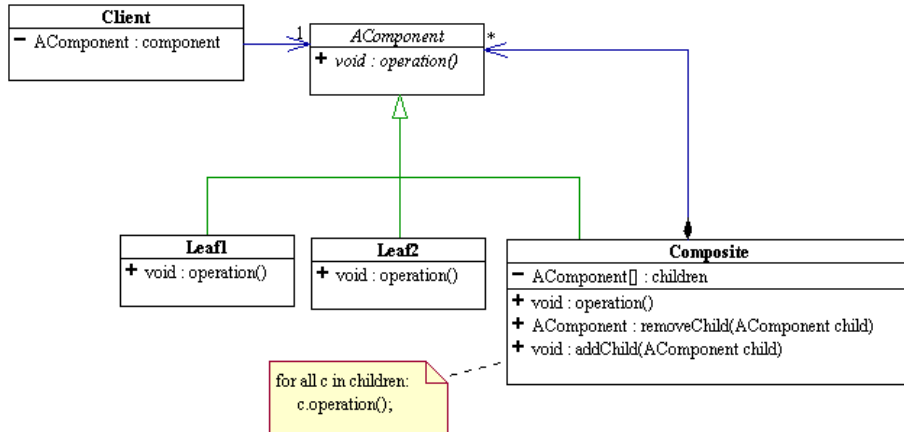
- Patterns ou Modèles de conception réutilisables
- Un modèle == plusieurs classes == Un nom de Pattern
 - > Assemblage de classes pour un discours plus clair
- Les librairies standard utilisent ces Patterns
 - L'API AWT utilise le patron/pattern composite ???
 - Les événements de Java utilisent le patron Observateur ???
 - ...
 - etc. ...
- Une application = un assemblage de plusieurs patterns
 - Un rêve ?

ESIEE

6

La bibliothèque graphique du JDK utilise un composite ?

- Le pattern Composite ?, usage d'un moteur de recherche sur le web ...

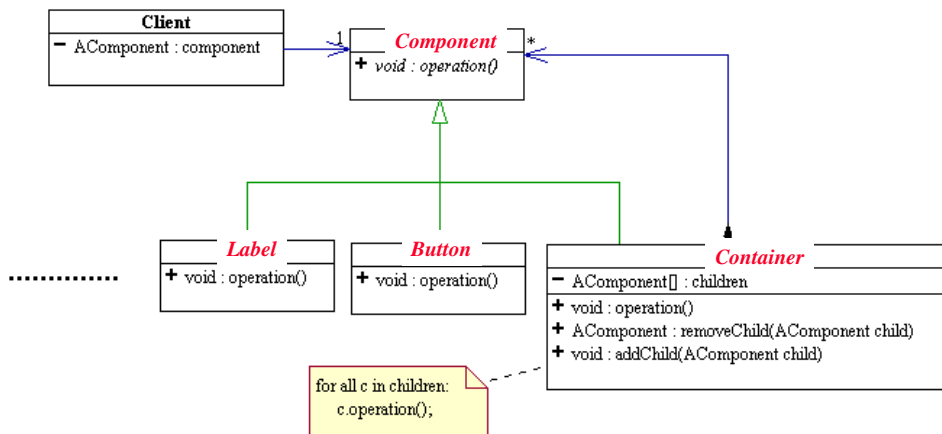


ESIEE

7

la bibliothèque graphique utilise bien un Composite :

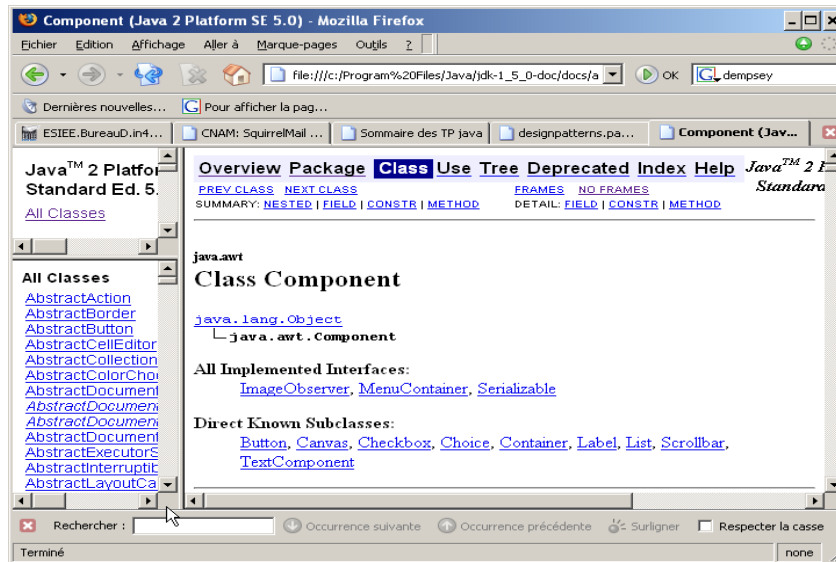
java.awt.Component java.awt.Button java.awt.Container ...



ESIEE

8

À la place de



ESIEE

9

Pattern - Patrons, sommaire

- **Historique**
- **Classification**
- **Les fondamentaux ...**
- **Quelques patrons en avant-première**
 - Adapter, Proxy

ESIEE

10

Patrons/Patterns pour le logiciel

- **Origine C. Alexander un architecte**
 - 1977, un langage de patrons pour l'architecture 250 patrons
- **Abstraction dans la conception du logiciel**
 - [GoF95] la bande des 4 : Gamma, Helm, Johnson et Vlissides
 - 23 patrons/patterns
- **Une communauté**
 - PLoP Pattern Languages of Programs
 - <http://hillside.net>

ESIEE

11

Introduction

- **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

12

Patron défini par J. Coplien

- **Un pattern est une règle en trois parties exprimant une relation entre un contexte, un problème et une solution (Alexander)**

- **Summary by Jim Coplien:**

Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves.

ESIEE

13

Définition d'un patron

- **Contexte**
- **Problème**
- **Solution**

- **Patterns and software :**
 - Essential Concepts and Terminology par Brad Appleton
<http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>

- **Différentes catégories**
 - Conception (Gof)
 - Architecturaux(POSA/GoV, POSA2 [Sch06])
 - Organisationnels (Coplien www.ambyssoft.com/processPatternsPage.html)
 - Pédagogiques(<http://www.pedagogicalpatterns.org/>)
 -

ESIEE

14

Les fondamentaux [GrandOO] avant tout

- **Constructions**
 - Delegation
 - Interface
 - Abstract superclass
 - Immutable
 - Marker interface

ESIEE

15

Delegation

- **Ajout de fonctionnalités à une classe**
- **Par l'usage d'une instance d'une classe**
 - Une instance inconnue du client
- **Gains**
 - Couplage plus faible
 - Sélection plus fine des fonctionnalités souhaitées

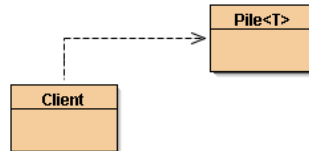
ESIEE

16

Delegation : un exemple classique...

```
import java.util.Stack;
public class Pile<T>{
    private final Stack<T> stk;
    public Pile(){
        stk = new Stack<T>();
    }
    public void empiler(T t){
        stk.push(t);
    }
    ...}

```



```
public class Client{
    public void main(String[] arg){
        Pile<Integer> p = new Pile<Integer>();
        p.empiler(4);
    }
}

```

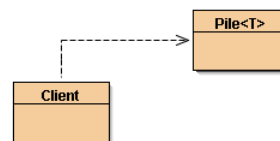
ESIEE

17

Delegation : souplesse ... Client inchangé

```
import java.util.List;
import java.util.LinkedList;
public class Pile<T>{
    private final List<T> stk;
    public Pile(){
        stk = new LinkedList<T>();
    }
    public void empiler(T t){
        stk.addLast(t);
    }
    ...}

```



```
public class Client{
    public void main(String[] arg){
        Pile<Integer> p = new Pile<Integer>();
        p.empiler(4);
    }
}

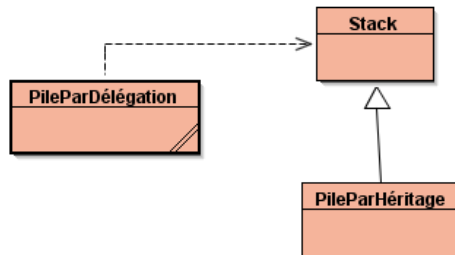
```

ESIEE

18

Délégation / Héritage

- Petite discussion...



- Avantages/inconvénients
- Délégation préférée

mais

ESIEE

19

Délégation : une critique

```
public class Pile<T>{
    private final List<T> stk;

    public Pile(){
        stk = new LinkedList<T>();
    }
    ...}

```

L'utilisateur
n'a pas le choix de
l'implémentation de la Liste

ESIEE

20

Interface

- **La liste des méthodes à respecter**

- Les méthodes qu'une classe devra implémenter
- Plusieurs classes peuvent implémenter une même interface
- Le client choisira une implémentation en fonction de ses besoins

- Exemple

- Collection<T> est une interface
- ArrayList<T>, LinkedList<T> sont des implémentations de Collection<T>
- Iterable<T> est une interface
- L'interface Collection « extends » cette interface et propose la méthode
 - public Iterator<T> iterator();

ESIEE

21

Interface : java.util.Iterator<E>

```
interface Iterator<E>{  
    E next();  
    boolean hasNext();  
    void remove();  
}
```

Exemple :

Afficher le contenu d'une Collection<E> nommée *collection*

```
Iterator<E> it = collection.iterator();  
while( it.hasNext()){  
    System.out.println(it.next());  
}
```

ESIEE

22

Usage des interfaces

un filtre : si la condition est satisfaite alors retirer cet élément

```
public static
<T> void filtrer( Collection<T> collection,
                 Condition<T> condition){

    Iterator<T> it = collection.iterator();
    while (it.hasNext()) {
        T t = it.next();
        if (condition.isTrue(t)) {
            it.remove();
        }
    }
}

public interface Condition<T>{
    public boolean isTrue(T t);
}
```

Collection et Condition
sont des interfaces

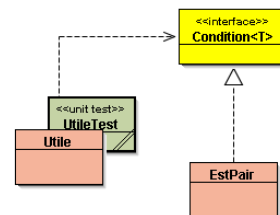
discussion

ESIEE

23

Exemple suite

- Usage de la méthode filtrer
 - retrait de tous les nombres pairs d'une liste d'entiers



```
Collection<Integer> liste = new ArrayList<Integer>();
liste.add(3);liste.add(4);liste.add(8);liste.add(3);
System.out.println("liste : " + liste);
```

```
Utile.filtrer(liste,new EstPair());
System.out.println("liste : " + liste);
```

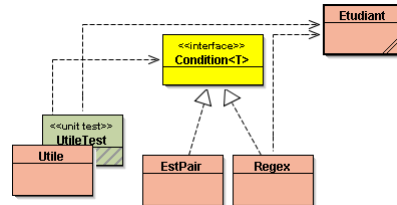
```
BlueJ: BlueJ : Terminal - filtre
Options
liste : [3, 4, 8, 3]
liste : [3, 3]
```

ESIEE

24

Exemple suite bis

- Usage de la méthode filtrer
 - retrait de tous les étudiants à l'aide d'une expression régulière



```

Collection<Etudiant> set = new HashSet<Etudiant>();
set.add(new Etudiant("paul"));
set.add(new Etudiant("pierre"));
set.add(new Etudiant("juan"));
System.out.println("set : " + set);

```

```

Utile.filtrer(set,new Regex("p[a-z]+"));
System.out.println("set : " + set);

```

```

BlueJ: BlueJ: Terminal - filtre
Options
set : [juan, paul, pierre]
set : [juan]

```

discussion

ESIEE

25

Délégation : une critique, bis

```

public class Pile<T>{
    private final List<T> stk;

    public Pile(){
        stk = new LinkedList<T>();
    }
    ...}

```

L'utilisateur
n'a pas le choix de
l'implémentation ...

```

public class Pile<T>{
    private final List<T> stk;

    public Pile(List<T> l){
        stk = l;
    }
    public Pile(){
        stk = new LinkedList<T>();
    }
    ...}

```

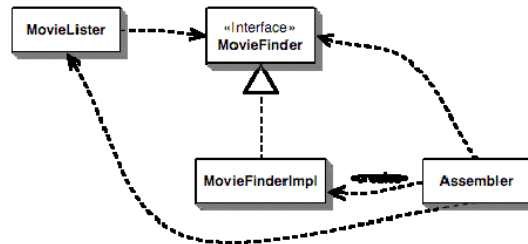
Ici l'utilisateur
a le choix de
l'implémentation de la Liste ...

ESIEE

26

Vocabulaire : Injection de dépendance

- Délégation + interface = injection de dépendance
- Voir Martin Fowler
 - « Inversion of Control Containers and the Dependency Injection pattern »
 - <http://martinfowler.com/articles/injection.html>



- L'injection de dépendance est effectuée à la création de la pile ...
- Voir le paragraphe « Forms of Dependency Injection »

ESIEE

27

Délégation : une critique de critique

```
public class Pile<T>{  
    private final List<T> stk;  
  
    public Pile(List<T> l){  
        stk = l;  
    }  
}
```

← Ici l'utilisateur
a le choix de
l'implémentation de la Liste ...

Mais rien n'empêche ici une utilisation *malheureuse* de l à l'extérieur de Pile

```
List<String> l = new LinkedList<String>(); // ← correct  
  
Pile<String> p = new Pile(l); // ← idem  
p.empiler("ok"); // ← idem  
  
l.add("attention"); // ← attention  
// état de la pile ?
```

Une solution ? Satisfaisante ?

ESIEE

28

Abstract superclass

- **Construction fréquemment associée à l'Interface**

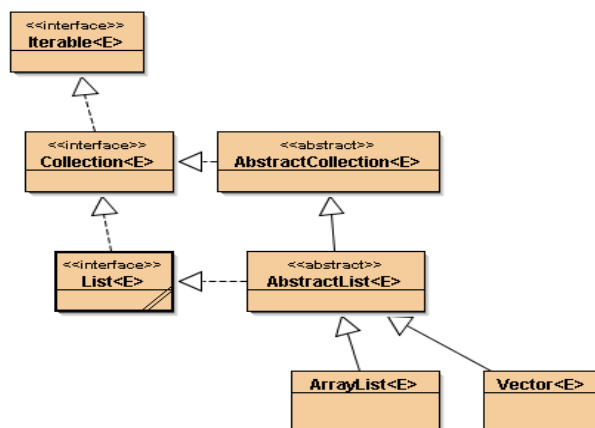
- Une classe propose une implémentation incomplète
 - **abstract class en Java**
- Apporte une garantie du « bon fonctionnement » pour ses sous-classes
- Une sous-classe doit être proposée
- Souvent liée à l'implémentation d'une interface
- Exemple extrait de java.util :
 - **abstractCollection<T>** propose 13 méthodes sur 15
 - **et implémente Collection<T>** ...

ESIEE

29

Abstract superclass exemple

- java.util.Collection un extrait



ESIEE

30

Immutable

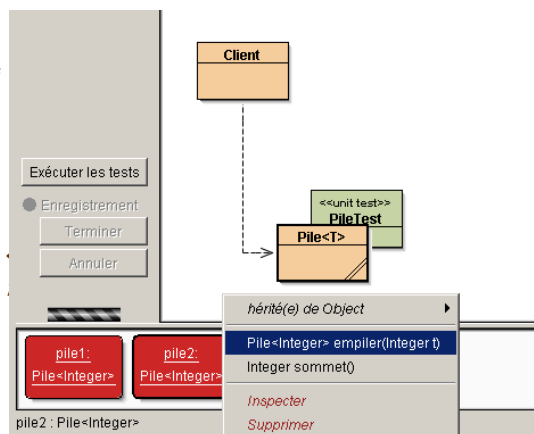
- **La classe, ses instances ne peuvent changer d'état**
 - Une modification engendre une nouvelle instance de la classe
- **Robustesse attendue**
- **Partage de ressource facilitée**
 - Exclusion mutuelle n'est pas nécessaire
- **java.lang.String est « Immutable »**
 - Contrairement à java.lang.StringBuffer

ESIEE

31

Immutable : exemple

```
public class Pile<T>{  
  
    private final Stack<T> stk;  
  
    public Pile(){  
        stk = new Stack<T>();  
    }  
  
    public Pile<T> empiler(T t){  
        Pile<T> p = new Pile<T>();  
        p.stk.addAll(this.stk);  
        p.stk.push(t);  
        return p;  
    }  
  
    public T sommet(){  
        return stk.peek();  
    }  
  
    ...  
}
```



ESIEE

32

Marker Interface

- **Une interface vide !**

- Classification fine des objets
- implements installée sciemment par le programmeur

- Exemples célèbres

- java.io.Serializable, java.io.Cloneable

- Lors de l'usage d'une méthode particulière une exception sera levée si cette instance n'est pas du bon « type »

- Note : Les annotations de Java peuvent remplacer « élégamment » cette notion

ESIEE

33

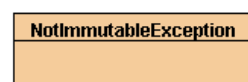
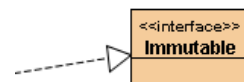
Marker Interface : exemple

```
public interface Immutable{}

public class NotImmutableException
    extends RuntimeException{
    public NotImmutableException(){super();}
    public NotImmutableException(String msg){super(msg);}
}

public class Pile<T> implements Immutable{
    ...
}

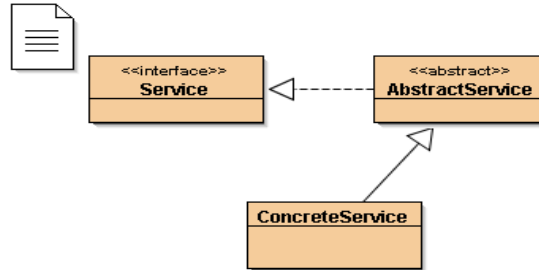
Pile<Integer> p = new Pile<Integer>();
if(!(p instanceof Immutable))
    throw new NotImmutableException();
...
```



ESIEE

34

Interface & abstract

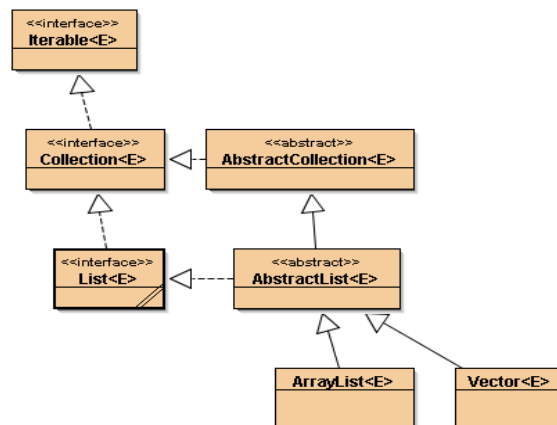


- **Avantages cumulés !**
 - `Collection<T>` interface
 - `AbstractCollection<T>`
 - `ArrayList<T>`

ESIEE

35

Interface & abstract



- **Déjà vu ...**

ESIEE

36

Les 23 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

37

Deux patrons pour l'exemple...

- **Dans la famille "Patrons Structurels"**

- **Adapter**

- **Adapte** l'interface d'une classe afin d'être conforme aux souhaits du client

- **Proxy**

- **Fournit** un mandataire au client afin de contrôler/vérifier ses accès

ESIEE

38

Adaptateurs : exemples



- **Adaptateurs**

- prise US/ adaptateur / prise EU
- PÉritel / adaptateur / RCA

ESIEE

39

Adaptateur exemple : PÉritel ↔ RCA

Ce que nous avons : RCA

```
public interface Plug {  
    public void RCA();  
}
```

Ce que le client souhaite : une prise PÉritel

```
public interface Prise {  
    public void péritel();  
}
```



- Il faut s'adapter aux souhaits du client

ESIEE

40

Adaptateur (implements Prise)

```
public class Adaptateur implements Prise {
    public Plug adapté;

    public Adaptateur(Plug adapté){
        this.adapté = adapté;
    }

    public void péritel(){
        adapté.RCA();
    }
}
```

ESIEE

41

Adaptateur : Le client est satisfait

```
public class UnClient {

    Prise prise = new Adaptateur(new PlugRCA());

    prise.péritel(); // satisfait

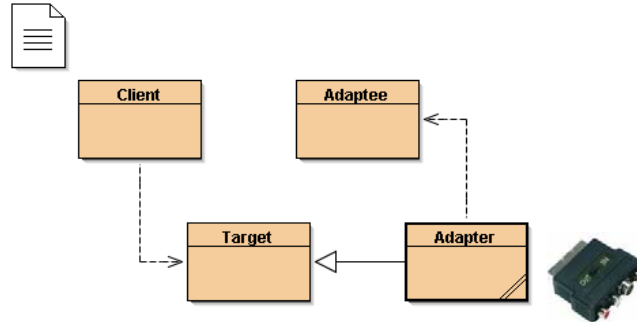
}

public class PlugRCA implements Plug {
    public void RCA(){ ....}
}
```

ESIEE

42

Pattern Adapter [DP05]



- DP05 ou www.patterncoder.org, un plug-in de bluej

ESIEE

43

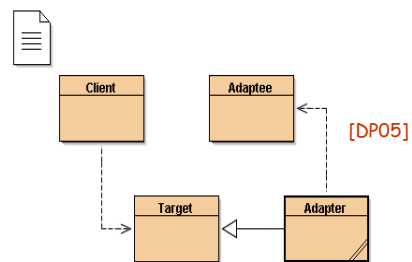
Pattern Adapter [DP05]

```
public interface Target {
    public void serviceA();
}

public class Adaptee {
    public void serviceB(){...}
}

public class Adapter implements Target {
    public Adaptee adaptee;
    public Adapter(Adaptee adaptee){
        this.adaptee = adaptee;
    }

    public void serviceA(){
        adaptee.serviceB();
    }
}
```



ESIEE

44

Adapter et classe interne java

- **Souvent employé ...**

```
public Target newAdapter(final Adaptee adaptee){
    return
        new Target(){
            public void serviceA(){
                adaptee.serviceB();
            }
        };
}
```

- **Un classique ...**

```
w.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
```

ESIEE

45

Une question d'un examen de l'esiee ...

3) Soit l'interface PileI ci-dessous, sans aucune implémentation.

```
1 public interface PileI<E>{
2
3     public void empiler(E e);
4     public E dépiler();
5     public boolean estVide();
6 }
```

par contre nous disposons de plusieurs implémentations de l'interface StackI,

```
3 public interface StackI<E>{
4
5     public void push(E e);
6     public E pop();
7     public boolean isEmpty();
8 }
```

L'utilisateur est francophone et souhaite vivement continuer d'appeler les méthodes définies dans l'interface PileI.

Choisissez un patron permettant à cet utilisateur de respecter ses souhaits et implémentez complètement la solution. Bien entendu empiler à la même sémantique que push, idem pour dépiler/pop et estVide/isEmpty



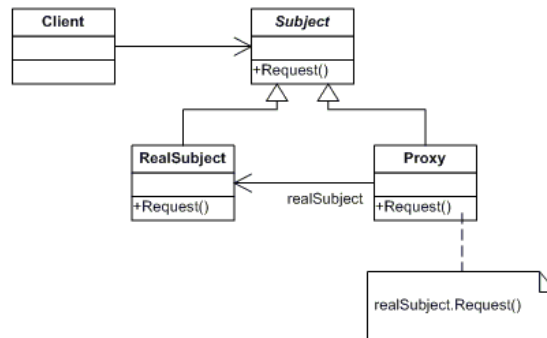
Proposez un scénario d'utilisation pour notre ClientFrancophone

ESIEE

46

Pattern Proxy

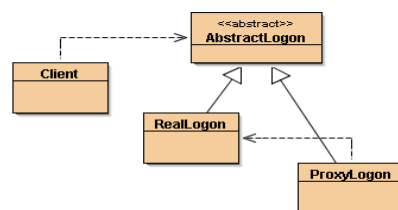
- Fournit un mandataire au client afin de
 - Contrôler/vérifier les accès



ESIEE

47

Proxy : un exemple



```
public abstract class AbstractLogon{
    abstract public boolean authenticate( String user, String password);
}

public class Client{
    public static void main(String[] args){
        AbstractLogon logon = new ProxyLogon();
        ...
    }
}
```

ESIEE

48

Proxy : exemple suite

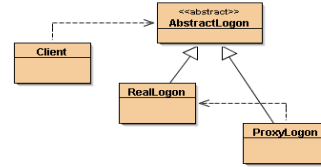
```

public class ProxyLogon extends AbstractLogon{
    private AbstractLogon real = new RealLogon();

    public boolean authenticate(String user, String password){
        if(user.equals("root") && password.equals("java"))
            return real.authenticate(user, password);
        else
            return false;
    }
}

public class RealLogon extends AbstractLogon{
    public boolean authenticate(String user, String password){
        return true;
    }
}

```

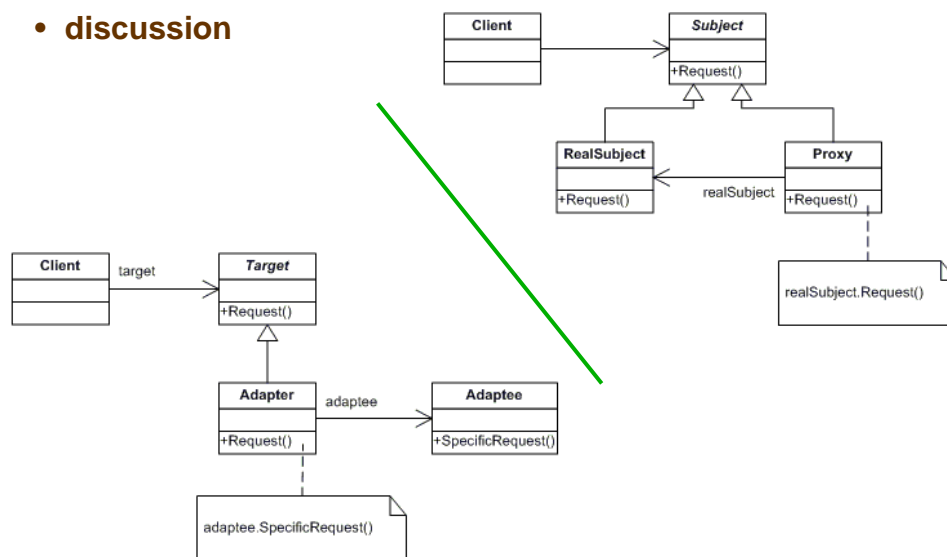


ESIEE

49

Adapter\Proxy

- discussion



ESIEE

50

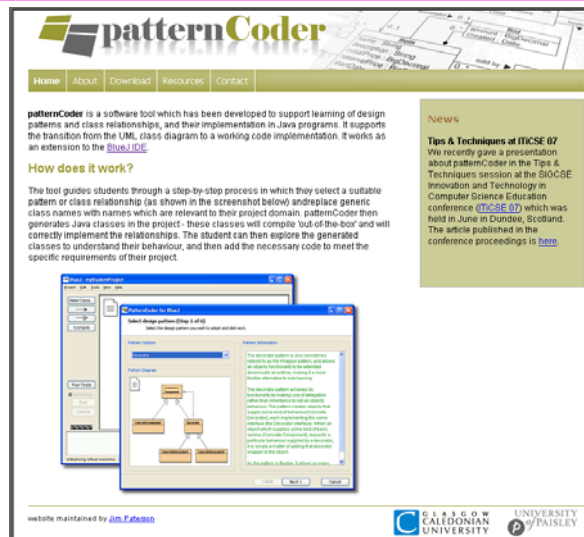
Conclusion

- Est-ce bien utile ?
- Architecture décrite par les patterns ?
- Langage de patterns ?
- Comment choisir ?
- Trop de Patterns ?
- Méthodologie d'un AGL ?

ESIEE

51

BlueJ : www.patterncoder.org



The screenshot shows the website for patternCoder, which is a software tool for learning design patterns. The website features a navigation menu with links for Home, About, Download, Resources, and Contact. The main content area includes a description of the tool, a 'How does it work?' section, and a 'News' section. The 'How does it work?' section explains that the tool guides students through a step-by-step process of selecting a pattern or class relationship, replacing generic class names with project-specific names, and generating Java classes. The 'News' section mentions a presentation at the ITICSE 07 conference. Below the text, there is a screenshot of the patternCoder software interface, which shows a UML class diagram and a dialog box for selecting a pattern. The website is maintained by Jim Fritton and is associated with Glasgow Caledonian University and Paisley University.

- Démonstration : le patron Adapter

ESIEE

52