
Memento, Command, template Method et Composite, Iterator, Visitor et les Transactions

jean-michel Douin, douin au cnam point fr
version : 15 Octobre 2010

Notes de cours

ESIEE

1

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 ...

- **Adapter**
 - Adapter l'interface d'une classe conforme aux souhaits du client.
- **Proxy**
 - Fournir un mandataire au client afin de contrôler/vérifier ses accès.
- **Observer**
 - Notifier un changement d'état d'une instance aux observateurs inscrits.
- **Template Method**
 - Laisser aux sous-classes une bonne part des responsabilités.
- **Iterator**
 - Parcourir une structure sans se soucier de la structure interne.
- **Singleton**
 - Garantir une et une seule instance.
- **Composite, Interpreter, Visitor, Decorator, ...**

ESIEE

3

Bibliographie utilisée

- Design Patterns, catalogue de modèles de conception réutilisables de Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides [Gof95]
International thomson publishing France

<http://www.fluffycat.com/Java-Design-Patterns/>

http://www.cs.wustl.edu/~levine/courses/cs342/patterns/compounding-command_4.pdf

<http://www.javapractices.com/Topic189.cjp>

Pour l'annexe :

<http://www.oreilly.com/catalog/hfdesignpat/>

ESIEE

4

Pré-requis

- Notions de
 - Les indispensables constructions
 - Interface, Abstract superclass, delegation...
 - Composite en première approche
 - Notion de Transaction
 - Commit-rollback

ESIEE

5

Sommaire

- **Les patrons**
 - **Memento**
 - Sauvegarde et restitution de l'état d'un objet
 - **Command**
 - Ajout et suppression de « Commande »
 - **Transaction ?**
 - **Memento** : persistance et mise en œuvre de la sauvegarde
 - **Template Method** : begin, end, ou rollback
 - **Composite** : structure arborescente de requêtes ...
 - **Visiteur** : parcours du composite

ESIEE

6

Memento

Sauvegarde et restitution de l'état interne d'un objet
sans violer le principe d'encapsulation.

- Pas d'accès aux attributs en général
 - Structures internes supposées inconnues

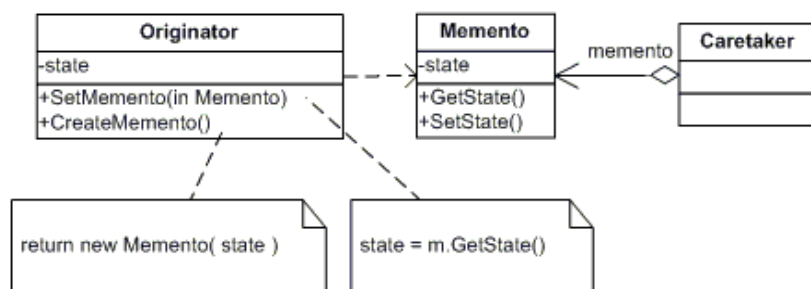
- Afin de stocker cet état, et le restituer

- Sauvegarde, annulation, journal, ...

ESIEE

7

Memento



- Sauvegarde et restitution de l'état interne d'un objet sans violer le principe d'encapsulation.

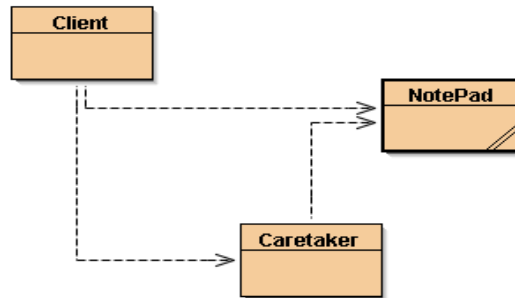
- Caretaker le conservateur
- Memento l'état de l'objet
- Originator contient les services de sauvegarde et de restitution d'un memento
 - Un exemple (vite...)

ESIEE

8

Memento exemple : NotePad/Agenda

Un agenda que l'on souhaite sauvegarder ...



- **NotePad** // Originator
- **NotePad.Memento** // Memento
- **CareTaker** // Gestion d'une instance de Memento

ESIEE

9

NotePad / ou Agenda

- **Une classe (très simple)**
 - permettant d'ajouter et de retirer des notes/rdv

```
public class NotePad implements ...{
    private List<String> notes;
```

```
    public void addNote(String note) throws NotePadFullException{
```

```
    public void remove(String note){...}
```

Note chaque agenda a un nombre limité de notes/rdv i.e. NotePadFullException (NFE)

Patron Memento :

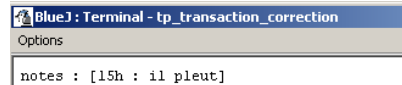
- **NotePad.Memento est une classe interne**
 - pour la sauvegarde et la restitution du notepad/agenda
 - *Note : L'état interne par définition est rarement accessible à l'extérieur ... « private ... » ...*

ESIEE

10

Usage : Un Client un memento, 1)

```
public class Client {  
    public static void main(String[] args) throws NotePadFullException {  
        NotePad notes = new NotePad();  
        notes.addNote("15h : il pleut");  
        System.out.println("notes : " + notes);
```



```
BlueJ: Terminal - tp_transaction_correction  
Options  
notes : [15h : il pleut]
```

```
        Caretaker gardien = new Caretaker();  
        gardien.setMemento(notes.createMemento()); // sauvegarde
```

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

ESIEE

11

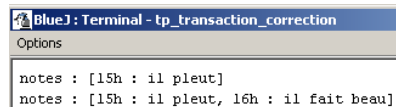
Usage : Un Client un memento, 2)

```
public class Client {  
    public static void main(String[] args) throws NotePadFullException  
    {  
        NotePad notes = new NotePad();  
        notes.addNote("15h : il pleut");  
        System.out.println("notes : " + notes);
```

```
        Caretaker gardien = new Caretaker();  
        gardien.setMemento(notes.createMemento()); // sauvegarde
```

```
        notes.addNote("16h : il fait beau");  
        System.out.println("notes : " + notes);
```

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



```
BlueJ: Terminal - tp_transaction_correction  
Options  
notes : [15h : il pleut]  
notes : [15h : il pleut, 16h : il fait beau]
```

ESIEE

12

Usage : Un Client un memento 3)

```
public class Client {
    public static void main(String[] args) throws NotePadFullException
    {
        NotePad notes = new NotePad();
        notes.addNote("15h : il pleut");
        System.out.println("notes : " + notes);
        Caretaker gardien = new Caretaker();
        gardien.setMemento(notes.createMemento()); // sauvegarde
        notes.addNote("16h : il fait beau");
        System.out.println("notes : " + notes);

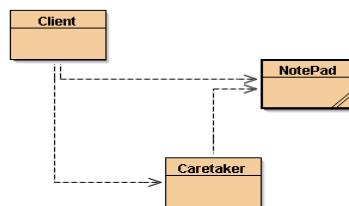
        notes.setMemento(gardien.getMemento()); // restitution
        System.out.println("notes : " + notes);
    }
}
```

```
BlueJ: Terminal - tp_transaction_correction
Options
notes : [15h : il pleut]
notes : [15h : il pleut, 16h : il fait beau]
notes : [15h : il pleut]
```

ESIEE

13

Memento exemple : NotePad/Agenda



- **NotePad** // *L'agenda, une liste par délégation*
 - Méthodes classiques d'un agenda : ajouter,....,
 - Et les méthodes de sauvegarde/restitution d'un Memento *createMemento, setMemento*
- **NotePad.Memento** // *Une classe interne, membre (la facilité ?)*
 - Méthodes de lecture/écriture d'un état de l'agenda, *getState, setState*
- **CareTaker** // *Source fourni*

ESIEE

14

La facilité

sans violer le principe d'encapsulation.

- Pas d'accès aux attributs en général
 - Structures internes supposées inconnues

- → sur cet exemple une classe interne et membre

Valable uniquement pour cette présentation ?

discussion

ESIEE

15

NotePad : l'agenda

```
public class NotePad {
    private List<String> notes = new ArrayList<String>();

    public void addNote(String note) throws NFE { notes.add(note);}
    public String toString(){return notes.toString(); }

    private List<String> getNotes(){return this.notes;}
    private void setNotes(List<String> notes){this.notes = notes;}

    public Memento createMemento() {
        Memento memento = new Memento();
        memento.setState(); // sauvegarde du NotePad, clone, copie...
                           // ici la « List<String> » notes
        return memento;
    }

    public void setMemento(Memento memento) {
        memento.getState(); // restitution du NotePad
    }
}
```

*NFE = NotePadFullException

ESIEE

16

NotePad.Memento

```
.....
// classe interne et membre... la facilité ?

public class Memento{
    private List<String> mementoNotes;

    public void setState(){// copie,clonage d'un notePad
        mementoNotes = new ArrayList<String>(getNotes());
    }

    public void getState(){
        setNotes(mementoNotes);
    }
}
.....
} // fin de la classe NotePad
```

ESIEE

17

Caretaker : le conservateur de memento

```
public class Caretaker {
    private NotePad.Memento memento;

    public NotePad.Memento getMemento() {
        return memento;
    }

    public void setMemento(NotePad.Memento memento){
        this.memento = memento;
    }
}
```

Abstraction réussie ...n'est-ce pas ? Mais ...

ESIEE

18

Conclusion intermédiaire / Discussion

- **Memento**

- Objectifs atteints

- Mais

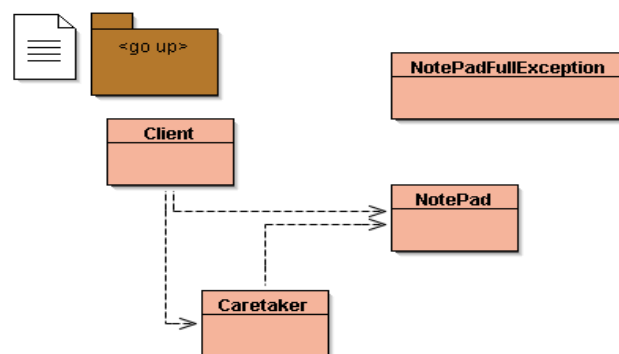
- Une classe dite « externe » sur cet exemple serait difficile à réaliser
 - Méthodes manquantes

- Prévoir les méthodes dès la conception de l' « Originator »
 - clone, copie en profondeur, Itérateur, Visiteur, ???, ...

ESIEE

19

Démonstration / discussion



- **Discutons ... Démonstration ...** *NFE est à l'affût*

ESIEE

20

Le Patron Command

- **Couplage faible opérations abstraites/réalisations concrètes**

- Sur un agenda nous avons « toujours » les opérations d'ajout, de retrait, d'affichage ... *que l'on pourrait qualifier d'opérations abstraites*
- Les noms des opérations concrètes sont dépendants d'une implémentation,
 - Ces noms peuvent changer,
- L'implémentation, les noms de classes peuvent changer,
- Etc. ...
- Comment assurer un couplage faible ? Opérations abstraites/concrètes

ESIEE

21

Commandes : une illustration



Télécommande Universelle Harmony 1000

Écran tactileGestion des dispositifs placés dans les placards, à travers les murs et les sols Grâce à la double transmission infrarouge (IR) et radiofréquence (RF), la télécommande Harmony 1000 peut contrôler des dispositifs sans pointage ni ligne de visée. Lorsque vous utilisez la télécommande conjointement avec extension Harmony RF Wireless

Dispositifs ou objets que l'on ne connaît pas !

ESIEE

22

Patron Command et notre agenda

- Les opérations de cet Agenda

Ajouter, Retirer, Afficher, ...

–Comment assurer un couplage faible entre l'agenda et ces(ses ?) opérateurs ?

–Usage du patron Command

- Ne connaît pas les actions à effectuer
- Ne connaît pas les effecteurs
- Ne connaît pas grand chose ...

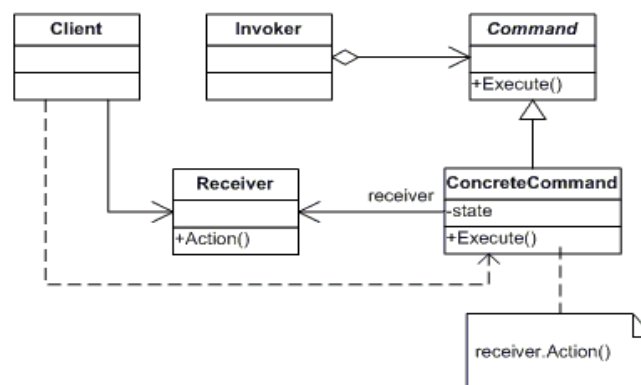
•Assemblage, configuration en fonction des besoins



ESIEE

23

Command, alias Action, Transaction ...

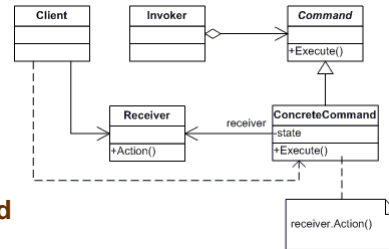


- **Abstraction des commandes effectuées**
 - Invoker ne sait pas ce qu'il commande Il exécute
 - Une commande concrète, et un récepteur concret
 - Souplesse attendue de ce couplage faible ...

ESIEE

24

L' exemple suite

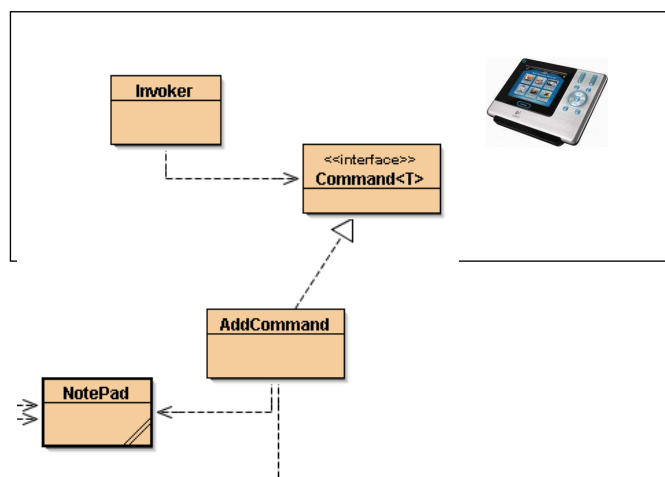


- **L'exemple précédent : le NotePad / agenda**
 - Receiver : NotePad : l'agenda
- **Abstraction des commandes**
 - Command et Invoker : La télécommande universelle
- **Ajouter, retirer**
 - ConcreteCommand

ESIEE

25

Command exemple



- **NotePad // est le Receiver**

ESIEE

26

L'interface Command

```
public interface Command<T>{  
    public void execute(T t) throws Exception;  
    public void undo();  
}
```



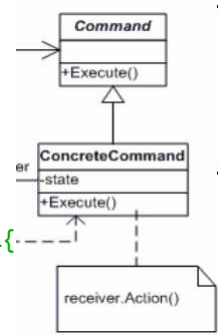
Une télécommande générique

ESIEE

27

Une « Command » concrète

```
public class AddCommand implements Command<String>{  
    private NotePad notes;  
  
    public AddCommand(NotePad notes){  
        this.notes = notes;  
    }  
  
    public void execute(String note) throws Exception{  
        notes.addNote(note);  
    }  
  
    public void undo(){}
```



// à faire, un Memento ??

ESIEE

28

Invoker

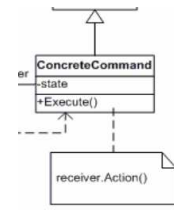
Abstraction de la commande effectuée,
relier la télécommande aux opérations dont on ignore tout ...

```
public class Invoker{
    private Command<String> cmd;

    public Invoker(Command<String> cmd){
        this.cmd = cmd;
    }

    public void addNotePad(String note){
        cmd.execute(note);
    }

    public void undo(){
        cmd.undo();
    }
}
```



ESIEE

29

Un Client

```
public class ClientCommand {

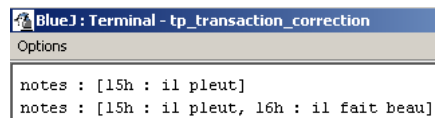
    public static void main(String[] args) throws Exception {

        NotePad notes = new NotePad();

        Invoker invoke = new Invoker(new AddCommand(notes));

        invoke.addNotePad(" 15h : il pleut");
        System.out.println(notes);

        invoke.addNotePad(" 16h : il fait beau");
        System.out.println(notes);
    }
}
```



```
BlueJ: Terminal - tp_transaction_correction
Options
notes : [15h : il pleut]
notes : [15h : il pleut, 16h : il fait beau]
```

ESIEE

30

Discussion

- **Patron Command**
 - Abstraction réussie
 - De l'invocateur et du récepteur
 - ...
 - Reste en suspend la Commande undo
 - Un memento ! Bien sûr

ESIEE

31

Command & Memento

```
public class AddCommand implements Command<String>{
    private NotePad notes;
    private Caretaker gardien; // le conservateur

    public AddCommand(NotePad notepad){
        this.notes = notepad;
        gardien = new Caretaker();
    }

    public void execute(String note){
        gardien.setMemento(notes.createMemento());
        try{
            notes.addNote(note);
        }catch(NotePadFullException e){} // rien en cas d'erreur ??
    }

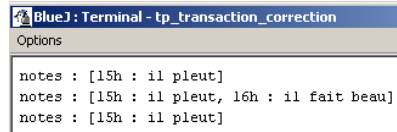
    public void undo(){
        notes.setMemento(gardien.getMemento());
    }
}
```

ESIEE

32

Un Client

```
public class ClientCommand2 {  
    public static void main(String[] args) {  
        NotePad notes = new NotePad();  
  
        Invoker invoke = new Invoker(new AddCommand(notes));  
        invoke.addNotePad("15h : il pleut");  
        System.out.println(notes);  
  
        invoke.addNotePad(" 16h : il fait beau ");  
        System.out.println(notes);  
  
        invoke.undo();  
        System.out.println(notes);  
    }  
}
```



```
BlueJ: Terminal - tp_transaction_correction  
Options  
notes : [15h : il pleut]  
notes : [15h : il pleut, 16h : il fait beau]  
notes : [15h : il pleut]
```

ESIEE

33

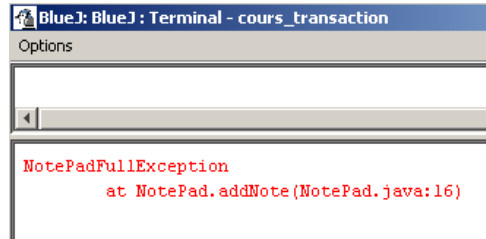
Démonstration ...

ESIEE

34

Une démonstration qui échoue ...

- **Comment est-ce possible ?**



```
BlueJ: BlueJ : Terminal - cours_transaction
Options
NotePadFullException
  at NotePad.addNote(NotePad.java:16)
```

- **Lorsque l'agenda est rempli !**

À suivre... de près

ESIEE

35

Combinaison de pattern

- **Commande + Memento**

- Fructueuse
- Discussion

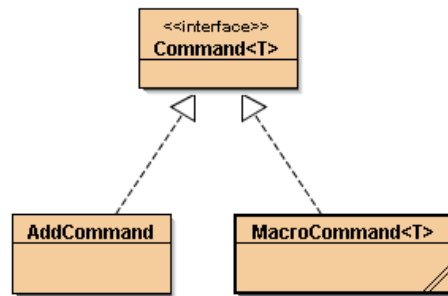
ESIEE

36

Command : un peu plus

- **MacroCommande**

- Un ensemble de commandes à exécuter
 - **MacroCommand<T>** implements **Command<T>**



ESIEE

37

La classe MacroCommand

```
public class MacroCommand<T> implements Command<T>{
    private Command<T>[] commands;

    public MacroCommand(Command<T>[] commands){
        this.commands = commands;
    }

    public void execute(T t){
        for(Command<T> cmd : commands)
            cmd.execute(t);
    }
    public void undo(){ ...}
}
```

ESIEE

38

Patron Command & AWT

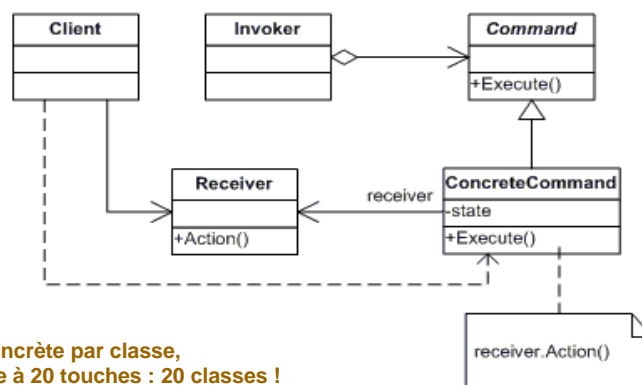
- Patron déjà présent dans l'AWT ?

- À chaque clic sur un « JButton »,
 - C'est le patron Command qui est employé
 - Soit Command + Observateur/Observé
 - Voir en annexe

ESIEE

39

Command



- Critiques
 - Une commande concrète par classe,
 - Une télécommande à 20 touches : 20 classes !

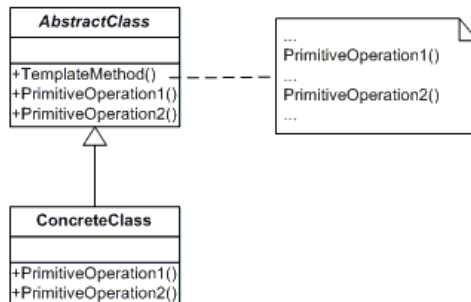
Mais
– la Factorisation du code des commandes est difficile ...

– Sauf si ...
`// on utilise le patron Template Method ...`

ESIEE

40

Template Method



- Nous laissons aux sous-classes de grandes initiatives ...

ESIEE

41

Template Method, exemple !

```
public class AbstractClass{
    public abstract void avant();
    public abstract void après();

    public void operation(){
        avant();
        // du code
        après();
    }
}
```

avant/après

Analogue à un **begin/end/rollback** d'une Transaction ?

ESIEE

42

Command + Template Method = AbstractTransaction ?

```
public void execute(T t){
    try{
        beginTransaction();
        // un appel de méthodes ici
        endTransaction();
    }catch(Exception e){
        rollbackTransaction();
    }
}

public abstract void beginTransaction();
public abstract void endTransaction();
public abstract void rollbackTransaction();
```

ESIEE

43

Command + Template Method

```
public interface Command<T>{
    public abstract void execute(T t);
}

public abstract class Transaction<T> implements Command<T>{

    public abstract void beginTransaction();
    public abstract void endTransaction();
    public abstract void rollbackTransaction();

    public void execute(T t){
        try{
            beginTransaction();
            atomic_execute(t);           // instruction dite atomique
            endTransaction();
        }catch(Exception e){
            rollbackTransaction();
        }
    }
}
```

ESIEE

44

Transaction « Sûre »

```
public class TransactionSure<T> extends Transaction<T>{

    public void beginTransaction(){
        // sauvegarde de l'état      (un Memento, bien sûr)
    }

    public void endTransaction(){
        // fin normale
    }

    public void rollbackTransaction(){
        // restitution de l'état(Memento)
    }
}
```

ESIEE

45

Et le patron Memento , souvenons-nous ...

```
public class TransactionSure<T> extends Transaction<T>{
    private Contexte ctxt;
    private CareTaker gardien;

    public class TransactionSure(Contexte ctxt){ ...}

    public void beginTransaction(){
        gardien.setMemento(ctxt.createMemento());
    }

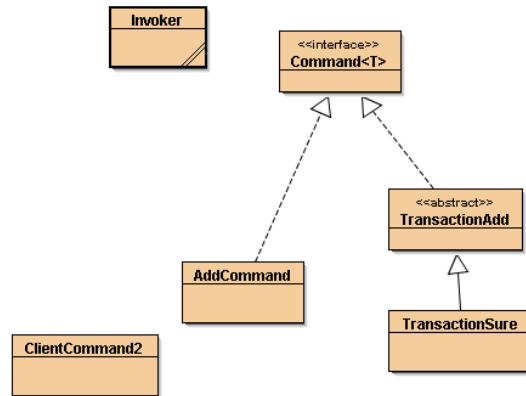
    public void endTransaction(){
        //
    }

    public void rollbackTransaction(){
        ctxt.setMemento(gardien.getMemento());
    }
}
```

ESIEE

46

Avec l'exemple initial, une nouvelle commande



- TransactionAdd et TransactionSure

ESIEE

47

Add transactionnel

```
public abstract class TransactionAdd implements Command<String>{

    public abstract void beginTransaction();
    public abstract void endTransaction();
    public abstract void rollbackTransaction();

    protected NotePad notes;

    public TransactionAdd(NotePad notepad){ this.notes = notepad;}

    public void execute(String note){
        try{
            beginTransaction();
            notes.addNote(note);
            endTransaction();
        }catch(Exception e){
            rollbackTransaction();
        }
    }
}
```

ESIEE

48

TransactionSure

```
public class TransactionSûre extends TransactionAdd{
    private Caretaker gardien;
    public TransactionSûre(NotePad notes){
        super(notes);
        this.gardien = new Caretaker();
    }
    public void beginTransaction(){
        gardien.setMemento(notes.createMemento());
    }
    public void endTransaction(){
        //gardien.oublie();
    }

    public void rollbackTransaction(){
        notes.setMemento(gardien.getMemento());
    }
    public void undo(){
        notes.setMemento(gardien.getMemento());
    }
}
```

ESIEE

49

Le client

```
public class ClientCommand2 {

    public static void main(String[] args) throws Exception {
        NotePad notes = new NotePad();

        Invoker invoke = new Invoker(new TransactionSûre(notes));
        invoke.addNotePad("15h : il pleut");
        System.out.println(notes);

        invoke.addNotePad(" 16h : il fait beau ");
        System.out.println(notes);
    }
}
```

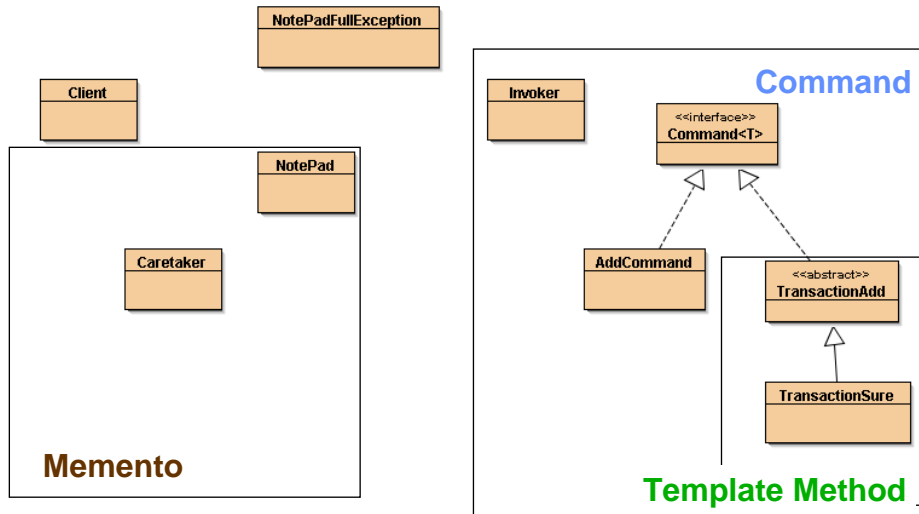
Couplage faible ...

ESIEE

50

Conclusion à Trois Patrons

- **Command + Memento + Template Method**



ESIEE

51

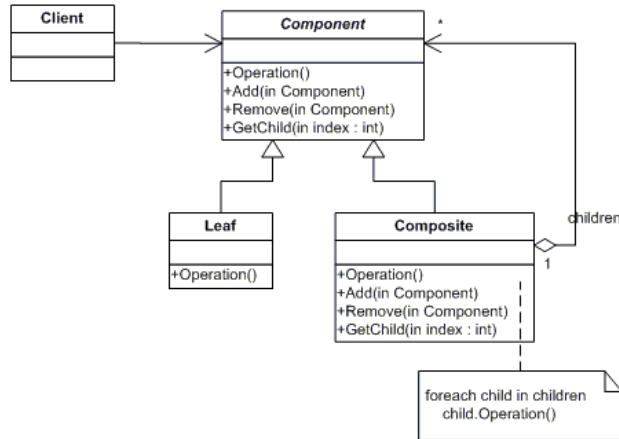
Plusieurs NotePad ?

- **Plusieurs NotePad/Agenda** mais combien ?
- **L'agenda d'une équipe, c.f. Google Agenda**
 - Synchronisés ?
 - Quelle structuration ?
 - Quel parcours ?
 - Une solution possible :
 - Utilisation du Patron **Composite**
 - Et les transactions ?
 - Et le parcours ?, **Itérateur, Visiteur** ?

ESIEE

52

Le patron Composite, rappel

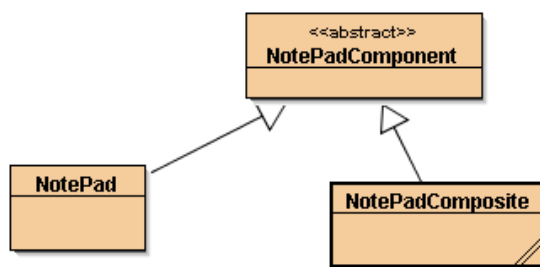


- Structures de données récursives
 - Hiérarchiques, arborescentes, ...

ESIEE

53

Plusieurs agendas : usage du Composite



```

public abstract class NotePadComponent{
    public abstract void addNote(String note) throws NotePadFullException;
    public abstract void remove(String note);
    public abstract String toString();
}
    
```

Rappel tout est **NotePadComponent**

ESIEE

54

Classe NotePadComposite

```
public class NotePadComposite extends NotePadComponent{
    private List<NotePadComponent> list;
    public NotePadComposite(){
        list = new ArrayList<NotePadComponent>();
    }
    public void addNote(String note) throws NotePadFullException{
        for(NotePadComponent n : list)
            n.addNote(note);
    }

    public void remove(String note){...}
    public String toString(){...}

    public void addChild(NotePadComponent notePad){
        list.add(notePad);
    }
    public List<NotePadComponent> getChildren(){
        return list;
    }
}
```

ESIEE

55

Hiérarchie d'agendas ? Pierre Pol et Jak

- 3 agendas une seule instance
 - Un composite d'agendas

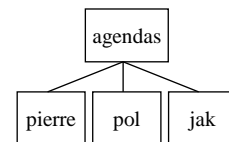
```
public static void testSimple() throws NotePadFullException{
    NotePadComposite agendas = new NotePadComposite();

    NotePad agendaDePierre = new NotePad(5); // 5 rdv possibles
    NotePad agendaDePol = new NotePad(15);
    NotePad agendaDeJak = new NotePad(7);

    agendas.addChild(agendaDePierre);
    agendas.addChild(agendaDePol);
    agendas.addChild(agendaDeJak);

    agendas.addNote("15/10/10,21:30 au KIBELE, 12 rue de l'échiquier");

    System.out.println(agendas);
}
```



ESIEE

56

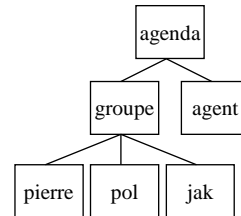
Une autre composition, l'agent et Pierre Pol Jak

```
public static void testEncoreSimple() throws NotePadFullException{
    NotePadComposite groupe = new NotePadComposite();

    NotePad agendaDePierre = new NotePad(5);
    NotePad agendaDePol = new NotePad(15);
    NotePad agendaDeJak = new NotePad(7);

    groupe.addChild(agendaDePierre);
    groupe.addChild(agendaDePol);
    groupe.addChild(agendaDeJak);

    NotePadComposite agenda = new NotePadComposite();
    NotePad agent = new NotePad(15);
    agenda.addChild(agent);
    agenda.addChild(groupe);
    agenda.addNote("21h : à l'entrepôt, Paris XIVème");
    System.out.println(agenda);
}
```



ESIEE

57

Mais ...

• Équité entre les agendas (leur possesseur ?)

- Si l'un d'entre eux est rempli (une exception est levée ... NFE*)
- Que fait-on ?
 - La réunion reste programmée ...
 - Ou bien
 - La réunion est annulée ...
 - Un groupe sans le chanteur ? (Dire Straits sans Mark Knopfler)
- Annulation, assurance
- Transaction à la rescousse

* NotePadFullException

ESIEE

58

Transaction ?

- L'un des agendas est « rempli »
 - Levée de l'exception **NotePadFullException**
 - Opération atomique :
Alors
 - Template Method + Command + Composite + Memento
 - C'est tout ? ...

ESIEE

59

Template & Command

```
public interface Command<T>{
    public void execute(T t);
    public void undo();
}

public abstract class Transaction extends Command<String>{
    protected NotePadComponent notePad;

    public abstract void beginTransaction();
    public abstract void endTransaction();
    public abstract void rollbackTransaction();

    public Transaction (NotePadComponent notePad){
        this.notePad = notePad;
    }
}
```

ESIEE

60

Transaction suite

```
public void execute(String note){
    try{
        beginTransaction();
        notePad.addNote(note);
        endTransaction();
    }catch(Exception e){
        rollbackTransaction();
    }
}
```

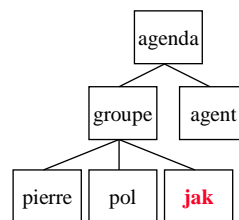
Du déjà vu ...

ESIEE

61

Mais pourquoi une Transaction ?

- Exemple : c'est **jak** qui est surbooké ...



- `agenda.addNote("15/10/10, A Bercy ")`
 - `groupe.addNote("15/10/10, A Bercy ")`
 - `pierre.addNote("15/10/10, A Bercy ")` // réussi, rdv pris
 - `pol.addNote ("15/10/10, A Bercy ")` // réussi, rdv pris
 - `jak.addNote("15/10/10, A Bercy ")` // échec, exception NFE
 - -> restitution de l'agenda initial de pierre et pol // transaction de la commande

ESIEE

62

Parfois le besoin d'un itérateur est pressant ...

- **Patron Composite + Itérateur**
 - Vers un Memento de composite

```
public class NotePadComposite
    extends NotePadComponent
    implements Iterable<NotePadComponent>{

    public Iterator<NodePadComponent> iterator(){
        return ...
    }
}
```

ESIEE

63

Discussion, petite conclusion à 4 patrons

Template Method

+

Command

+

Composite/Itérateur *

+

Memento de Composite

=

?

** voir en annexe, un itérateur de composite extrait de
<http://www.oreilly.com/catalog/hfdesignpat/>*

ESIEE

64

L'Itérateur : parfois complexe ... un extrait

```
public Iterator<NotePadComponent> iterator(){ return new Compositeliterator(list.iterator()); }

private class Compositeliterator implements Iterator<NotePadComponent>{
    private Stack<Iterator<NotePadComponent>> stk;

    public Compositeliterator(Iterator<NotePadComponent> iterator){
        stk = new Stack<Iterator<NotePadComponent>>();
        stk.push(iterator);
    }

    public boolean hasNext(){
        if(stk.empty()) return false;
        while( !stk.empty() && !stk.peek().hasNext()) stk.pop();           // Lisible ?
        return !stk.empty() && stk.peek().hasNext();
    }

    public NotePadComponent next(){
        if(hasNext()){
            ...
        }
    }
}
```

un extrait de tête la première ... <http://www.oreilly.com/catalog/hfdesignpat/>



ESIEE

65

À visiter ... vaut le détour

- **Patron Iterator**

- Parcours du Composite (*le code complet est en annexe*)

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

- **Patron Visitor**

- Parcours, actions définis par le client,
- Une visite par nœud concret du Composite

ESIEE

66

Le patron Visitor

```
public abstract class Visitor<T>{  
  
    public abstract T visit(NotePad notePad);  
    public abstract T visit(NotePadComposite notePad);  
}
```

- **La Racine du Composite, contient la méthode accept**

```
public abstract <T> T accept(Visitor<T> visitor);
```

- **Toutes les classes, feuilles du composite ont cette méthode**

```
public <T> T accept(Visitor<T> visitor){  
    return visitor.visit(this);  
}
```

ESIEE

67

Le Composite nouveau

La classe Racine du Composite s'est enrichie

```
public abstract class NotePadComponent {  
    public abstract void addNote(String note) throws NotePadFullException;  
    public abstract void remove(String note);  
    public abstract List<String> getNotes();  
    public abstract String toString();  
  
    public abstract <T> T accept(Visitor<T> visitor);  
}
```

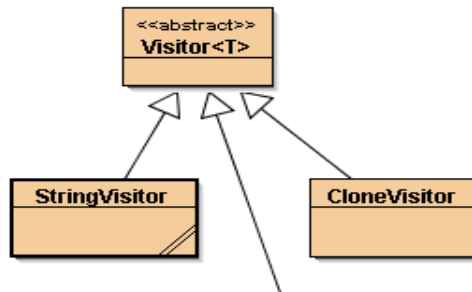
Chaque classe concrète Nœud déclenche la visite appropriée

```
public class NotePad extends NotePadComponent {  
    ...  
  
    public <T> T accept(Visitor<T> visitor){  
        return visitor.accept(this);  
    }  
}
```

ESIEE

68

Les visiteurs



- Tout type de visite à la charge du client devient possible ...
- *Plus simple : c'est au « client » de le faire ... une bonne idée...*

ESIEE

69

Un CloneVisitor

```
public class CloneVisitor extends Visitor<NotePadComponent>{

    public NotePadComponent visit(NotePad notePad){
        NotePad clone = new NotePad(notePad.getCapacity());
        try{
            for(String note : notePad.getNotes()){
                clone.addNote(note);
            }
            return clone;
        }catch(NotePadFullException e){return null;}
    }

    public NotePadComponent visit(NotePadComposite notePad){
        NotePadComposite clone = new NotePadComposite();
        for( NotePadComponent n : notePad.getChildren()){
            clone.addChild(n.accept(this));
        }
        return clone;
    }
}
```

ESIEE

70

Un test parmi d'autres

```
public void testAgentPierrePolJak_Visitors(){
    try{
        NotePadComposite groupe = new NotePadComposite();
        NotePad agendaDePierre = new NotePad(5);
        NotePad agendaDePol = new NotePad(15);
        NotePad agendaDeJak = new NotePad(7);
        groupe.addChild(agendaDePierre);groupe.addChild(agendaDePol);
        groupe.addChild(agendaDeJak);

        NotePadComposite agenda = new NotePadComposite();
        NotePad agent = new NotePad(15);
        agenda.addChild(agent);
        agenda.addChild(groupe);

        NotePadComponent clone = agenda.accept(new CloneVisitor());

        System.out.println("clone.toString() : " + clone);
        System.out.println("clone_visitor : " + clone.accept(new StringVisitor()));

    }catch(NotePadFullException e){
        fail(" agenda plein ? ");
    }catch(Exception e){
        fail("exception inattendue ??? " + e.getMessage());
    }
}
```

ESIEE

71

Conclusion à 5 Patrons

Template Method

+

Command

+

Composite/Iterator

+

Memento/Visitor (CloneVisitor)

=

?

ESIEE

72

Conclusion

- **Assemblage de Patrons**

- Transaction

ESIEE

73

Annexes

- **Patron Composite et parcours**

- Souvent récursif, « intra-classe »

- Avec un itérateur

- Même recette ...

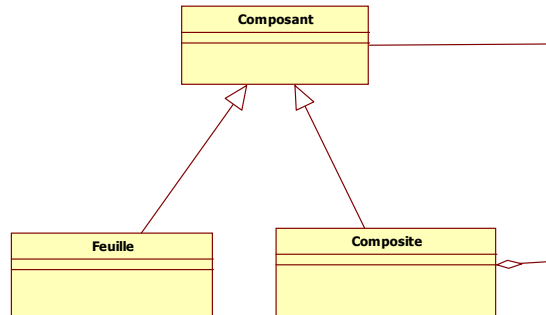
- Une pile mémorise l'itérateur de chaque classe « composite »

- Avec un visiteur

ESIEE

74

Composite et Iterator



- Structure réursive « habituelle »

ESIEE

75

En partie extrait de
<http://www.oreilly.com/catalog/hfdesignpat/>



- 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

76

Compositeliterator : 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

77

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

78

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

79

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

80

Composite & Iterator : l'exemple NotePad

```
public Iterator<NotePadComponent> iterator(){
    return new CompositeIterator(list.iterator());
}

private class CompositeIterator implements Iterator<NotePadComponent>{
    private Stack<Iterator<NotePadComponent>> stk;

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

    public boolean hasNext(){
        if(stk.empty()) return false;
        while( !stk.empty() && !stk.peek().hasNext()) stk.pop();
        return !stk.empty() && stk.peek().hasNext();
    }
}
```

ESIEE

81

NotePadComposite

```
public NotePadComponent next(){
    if(hasNext()){
        Iterator<NotePadComponent> iterator = stk.peek();
        NotePadComponent notepad = iterator.next();
        if(notepad instanceof NotePadComposite){
            NotePadComposite composite = (NotePadComposite)notepad;
            stk.push(composite.list.iterator());
        }
        return notepad;
    }else{
        throw new NoSuchElementException();
    }
}

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

ESIEE

82

Avec un visiteur

```
public abstract class Visitor<T>{  
  
    public abstract T visit(NotePad notePad);  
    public abstract T visit(NotePadComposite notePad);  
}
```

ESIEE

83

CloneVisitor

```
public class CloneVisitor extends Visitor<NotePadComponent>{  
  
    public NotePadComponent visit(NotePad notePad){  
        NotePad clone = new NotePad(notePad.getCapacity());  
        try{  
            for(String note : notePad.getNotes()){  
                clone.addNote(note);  
            }  
            return clone;  
        }catch(NotePadFullException e){ return null;}  
    }  
  
    public NotePadComponent visit(NotePadComposite notePad){  
        NotePadComposite clone = new NotePadComposite();  
        for( NotePadComponent n : notePad.getChildren()){  
            clone.addChild(n.accept(this));  
        }  
        return clone;  
    }  
}}
```

ESIEE

84

Command et java.awt.*

- **Button, MenuItem sont les invocateurs**
- **ActionListener la Commande**
- **Une implémentation d'ActionListener :**
 - **une commande concrète**
- **Le code d'ActionListener contient l'appel du récepteur**
- Exemple qui suit est extrait de
 - http://www.cs.wustl.edu/~levine/courses/cs342/patterns/compounding-command_4.pdf

ESIEE

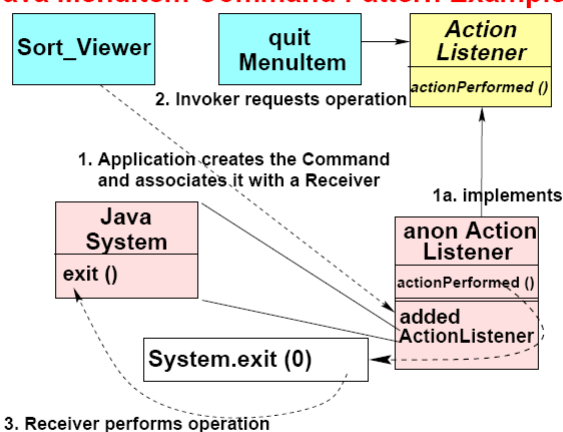
85

Command ActionListener ...

CS 342: OO Software Development Lab

OO Patterns

Java MenuItem Command Pattern Example



Copyright © 1997-2000 Dept. of Computer Science, Washington University

18

Extrait de http://www.cs.wustl.edu/~levine/courses/cs342/patterns/compounding-command_4.pdf

ESIEE

86

Est-ce bien le patron command ?

- « Quit » est l'invocateur
- ActionListener : Command
 - actionPerformed comme execute,...
- System.exit(0) : Receiver