IFT3902 :
(Gestion de projet pour le) développement, (et la) maintenance des logiciels

Yann-Gaël Guéhéneuc

Professeur adjoint

guehene@iro.umontreal.ca, local 2345

(Cours inspiré du cours d'Olivier Motelet)

Département d'informatique et de recherche opérationnelle

Université de Montréal

© Yann-Gaël Guéhéneuc 2004

Les patrons de conception
=
bonnes pratiques
du génie logiciel à objets

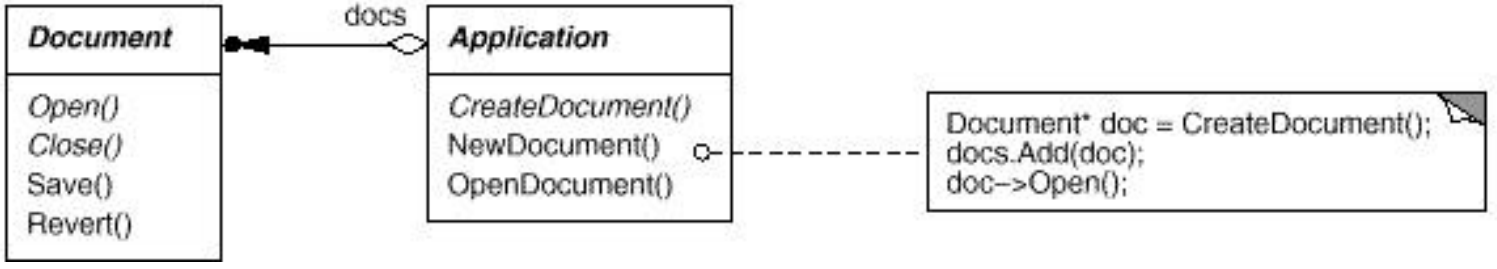
(= bonnes « recettes » ☺)

Les patrons de conception

- « Sentir »
  - La patron Factory Method
- « Voir »
  - Origine
  - Définition
  - Structure
- « Toucher »
  - Quand et comment les utiliser
  - Outils appuyant leur utilisation

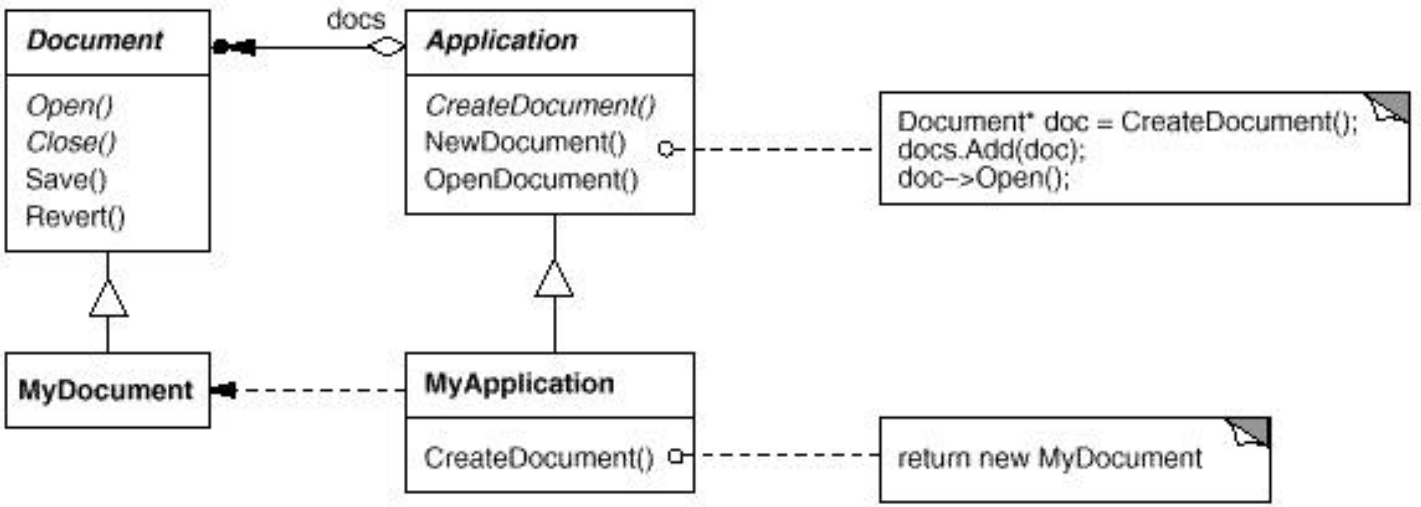
Méthode usine (1/3)

- Besoin : un cadre d'application qui gère plusieurs sortes de documents
- Problème : le cadre sait d'avance quand créer un document mais pas quel type de document créer



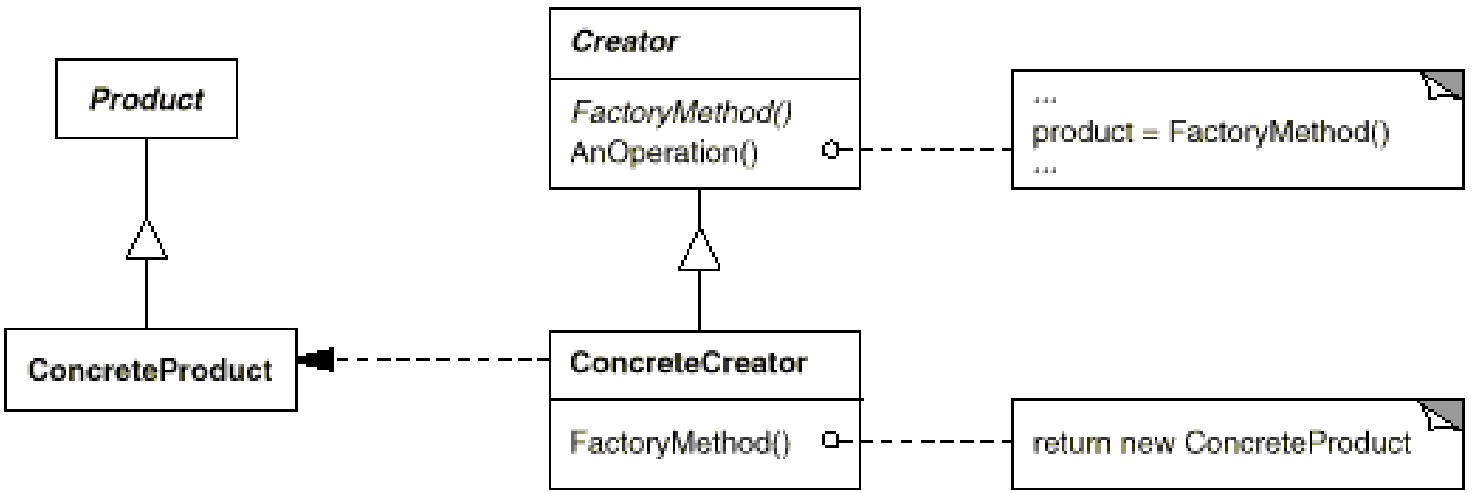
Méthode usine (2/3)

- Solution : isoler la connaissance du document à créer et « sortir » cette connaissance du programme



Méthode usine (3/3)

- Abstraction ?
- Caractéristiques de qualité ?



Origines (1/4)

« Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such way that you can use this solution a million times over, without ever doing it the same way twice »

« Each pattern is a three part rule, which express a relation between a context, a problem, and a solution »

Christopher Alexander [Alexander, 1977]

Origines (2/4)

« The strict modeling of the real world leads to reflect today’s realities but not necessarily tomorrow’s. The abstractions that emerge during design are key to making a design flexible »

Erich Gamma [Gamma et al., 1994]

Origines (3/4)

- Les programmes mettent en jeu de nombreuses classes et leur instances
- Vers un réutilisation d’ensemble de classes collaborantes
- Qu’est-ce qu’un bon style de programmation ?

Origines (4/4)

- Un patron de conception est une solution nommée et réutilisable à un problème récurrent de conception dans un contexte particulier

Définition (1/2)

« The description of communicating objects and classes customized to solve general design problem in a particular context »

« Each design pattern lets some aspect of system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change »

[Gamma et al., 1994]

Définition (2/2)

- Une manière d’améliorer et d’encourager la réutilisation
- Une manière de capture l’expérience de conception
- Un vocabulaire commun aux développeurs

Structure (1/3)

- Nom
- Problème
- Solution
- Conséquences

Et bien plus… (2/3)

- Problème + conséquence = contexte
  - Intention, application, conséquences
- Solution + conséquence = stratégies
  - Structure, participants, collaborations
- Compréhension
  - Motivation, patrons liés, usages connus
- Utilisation
  - implantations et exemples de code source

Mais… (3/3)

- Information éparpillée
  - Texte informelle

- Un exemple général plutôt qu'une règle générale

Tous les interpréter...

Quant utiliser les patrons ?

- Face à un problème complexe ?
  - Nombreux patrons de conception (existe-t-il une liste complète ?)
  - Granularité
    - Besoins, analyse, architecture
    - Conception, implantation (idiomes)
    - Restructuration, test
    - ...

Tous les connaître...

Comment utiliser les patrons ?

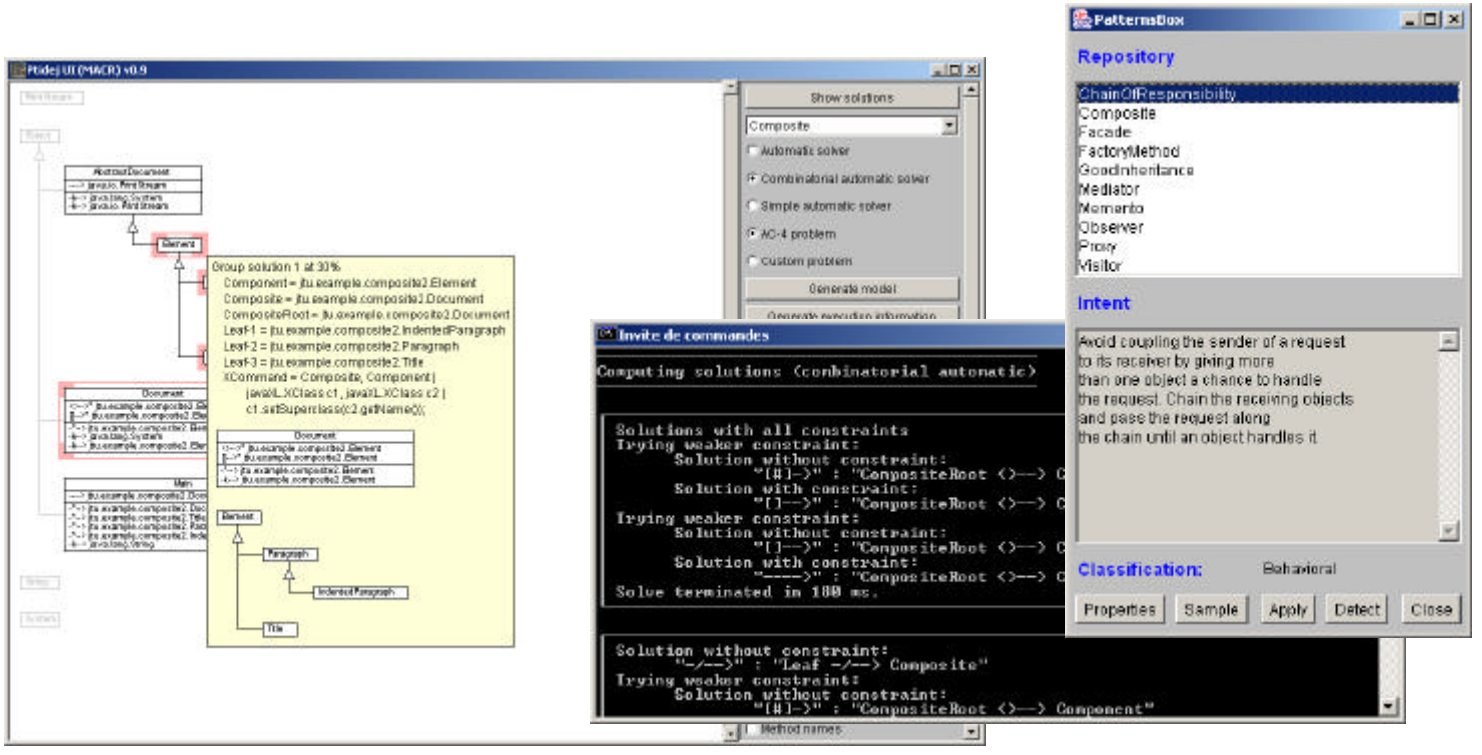
- Un processus itératif d'induction
  - D'un exemple à une abstraction à une application à un exemple à …
  - Processus de validation ?
- Catégories
  - Comportementale
  - De création
  - Structurelle

Outils appuyant l'utilisation des patrons de conception

- Livre du « GoF »
  - Listes, classifications, relations
  - [Gamma et al., 1996]
- Outils d'aide à la conception
  - Fragments [Florijn et al., 1997]
  - PatternsBox et Ptidej [Albin et al., 2001]
- Navigation
  - Tutor [Motelet, 2000]

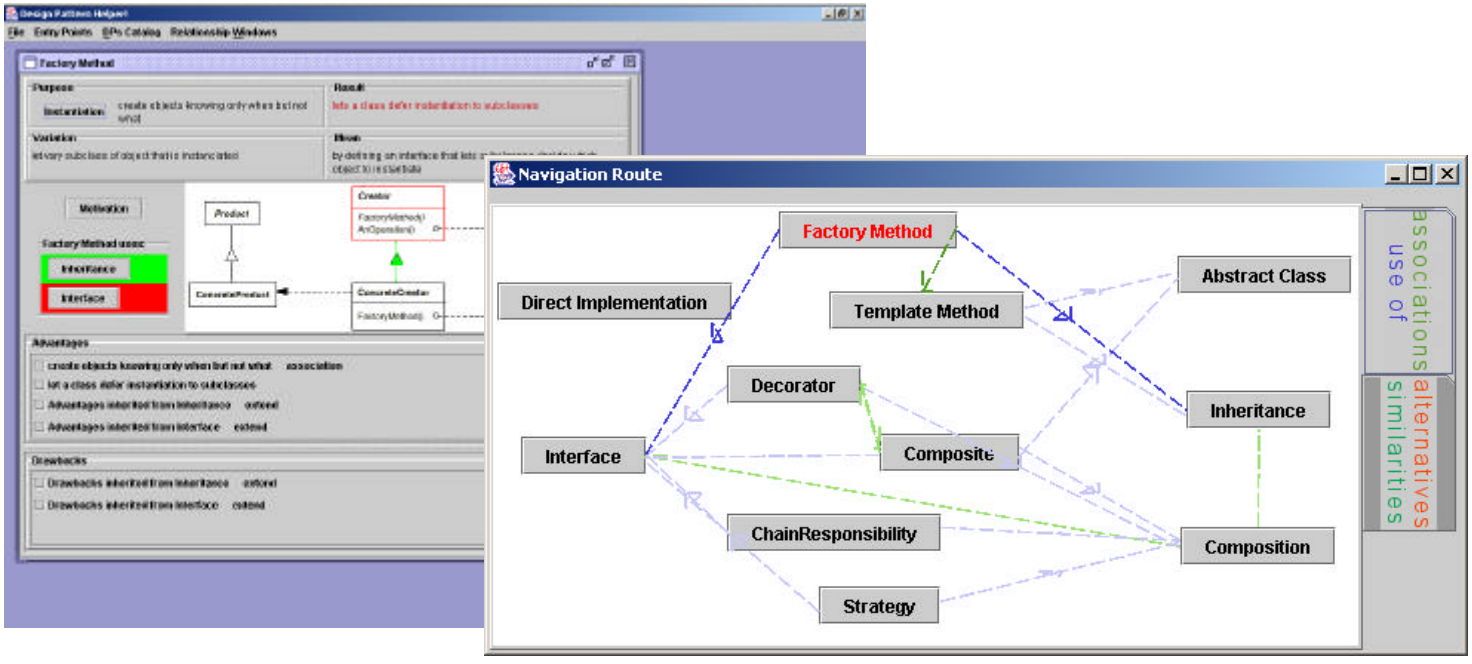
PatternsBox et Ptidej

- Démos ?



Tutor

- Demos ?


Navigation Route
Factory Method
Direct Implementation
Template Method
Abstract Class
Decorator
Inheritance
Interface
Composite
ChainResponsibility
Composition
Strategy
associations
use of
alternatives
similarities

Références

[Alexander, 1977] Christopher Alexander ; *A Pattern Language* ; New York Oxford University Press, 1977.

[Gamma et al., 1994] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides ; *Design Patterns – Elements of Reusable Object-Oriented Software* ; Addison Wesley 1994.

[Florijn et al., 1997] Gert Florijin, Marco Meijers, and Pieter van Winsen ; *Tool Support for Object-Oriented Pattern* ; Proceedings of ECOOP, 1997.

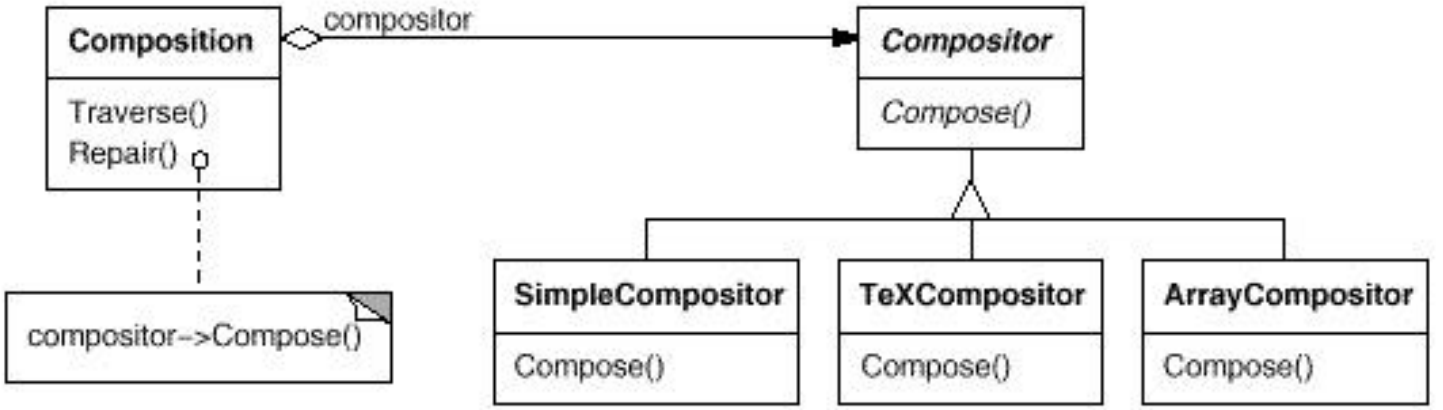
[Albin et al., 2001] Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, and Narendra Jussien ; *Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together* ; Proceedings of ASE, 2001.

[Motelet, 2000] Olivier Motelet ; *A Contextual Help System for Assisting Object-Oriented Software Designers in using Design Patterns* ; EMOOSE Master Thesis, 2000

Atelier

- Trois patrons de conception
  - Stratégie
  - Décorateur
  - Composite
- Objectifs
  - Questions
  - Idées
  - Explications

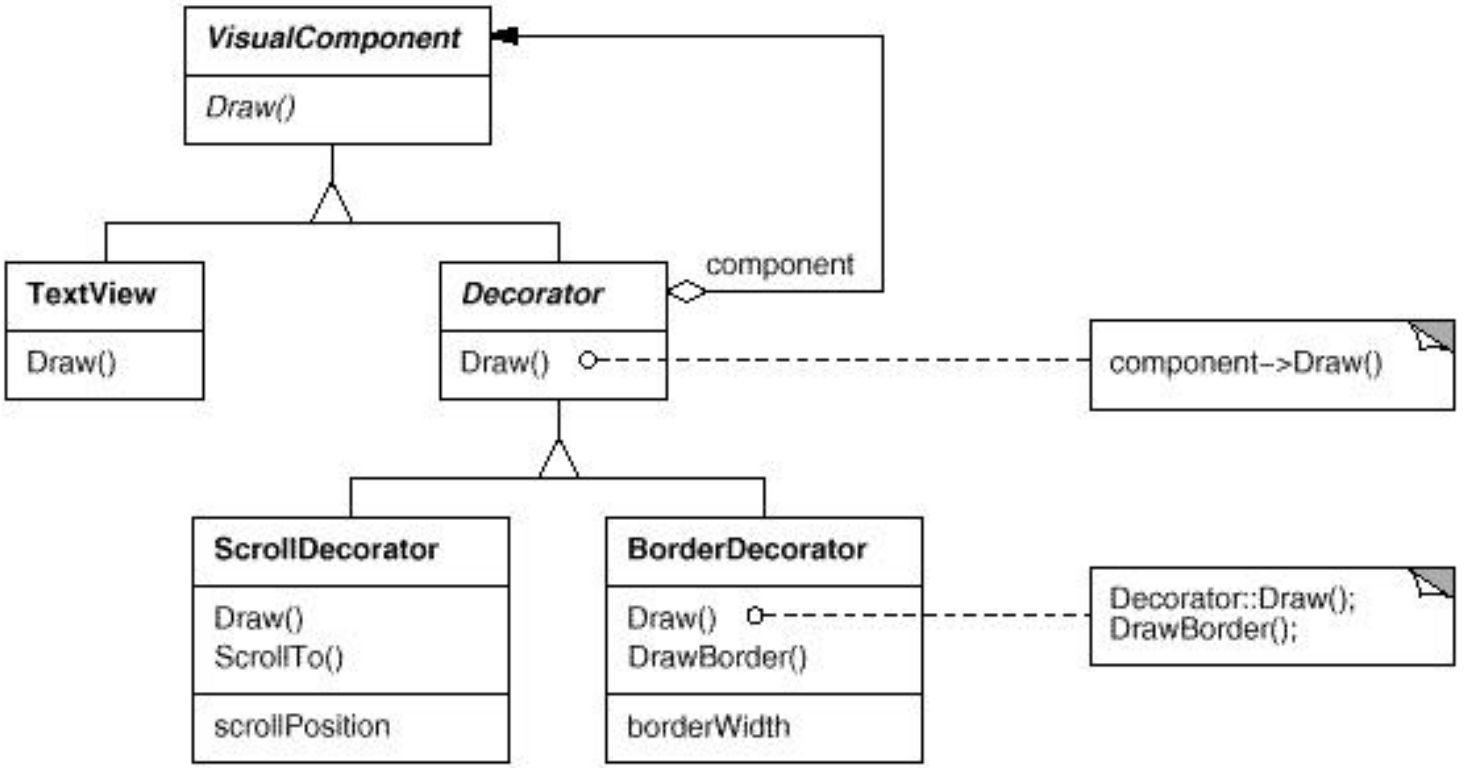
Stratégie (1/2)



Stratégie (2/2)

- *What happens when a system has an explosion of Strategy objects? Is there some way to better manage these strategies?*
- *In the implementation section of this pattern, the authors describe two ways in which a strategy object can get the information it needs to do its job. One way describes how a strategy object could get a reference from the context object, thereby giving it access to context data. But is it possible that the data required by the strategy are not available from the context interface. How could you solve this problem?*

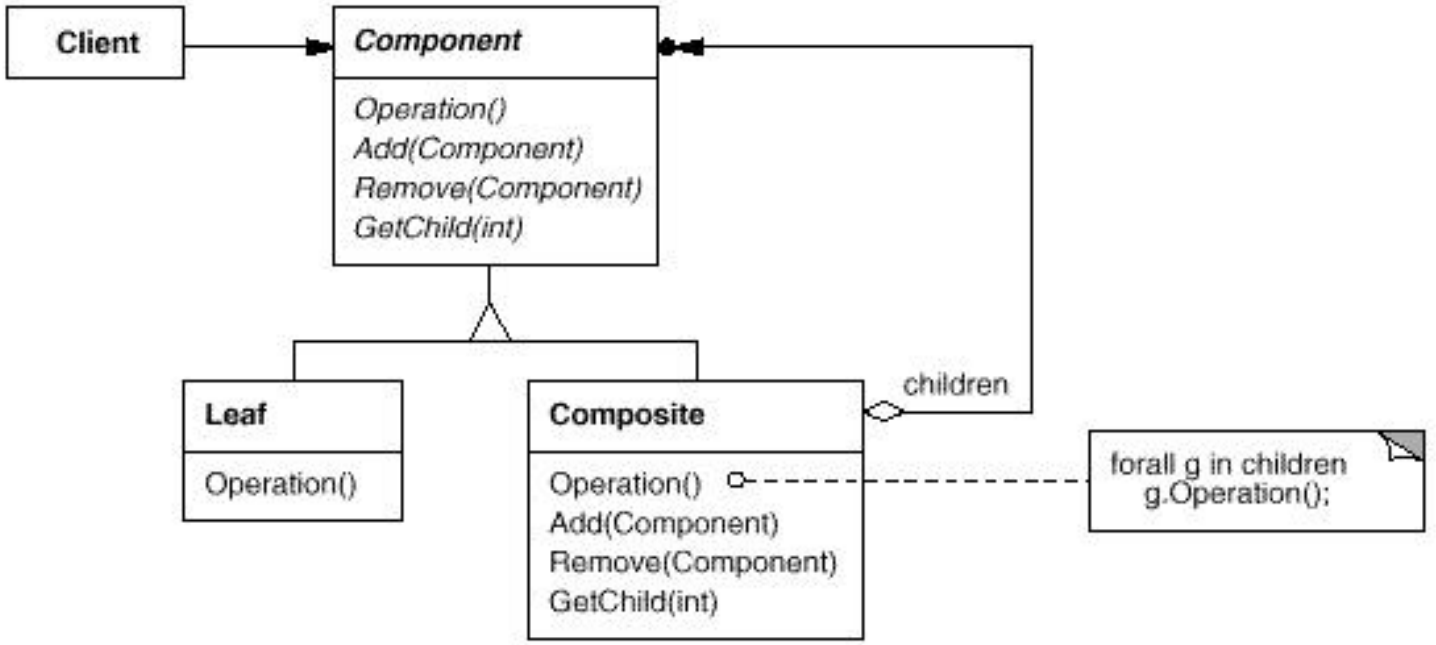
Décorateur (1/2)



Décorateur (2/2)

- *The implementation section of this pattern states that a decorator object's interface must conform to the interface of the object it decorates. Consider an object O, that is decorated with an object D. Object D shares an interface with object O because object D « decorates » object O. If some instance of this decorator attempts to call a method m that is not part of O's interface, does it mean that the object is no longer a decorator? Why is it important that a decorator object 's interface conforms to the interface of the object it decorates?*

Composite (1/2)



Composite (2/2)

- *How does the Composite design pattern help to consolidate system–wide conditional logic?*
- *Would you use this pattern if you do not have a part–whole hierarchy? In other words, if only a few objects have children and almost everything else in your collection is a leaf (a leaf has no children), would you still use this pattern to model these objects?*

Relations

- Décorateur – Stratégie
- Décorateur – Composite
- Composite – Décorateur

- Autres
  - Usine Abstraite – Singleton
  - …