Université Nice Sophia Antipolis

Patrons de conception: composite

F. Mallet

miage.m1@gmail.com

http://deptinfo.unice.fr/~fmallet/

Université Nice Sophia Antipolis

Motivation

- Les besoins pour une bonne conception et du bon code :
  - Extensibilité
  - Flexibilité
  - Facilité à maintenir
  - Réutilisabilité
  - Les qualités internes
  - Meilleure spécification, construction, documentation

Université Nice Sophia Antipolis

Historique

- MVC
- Gang of Four : Erich Gamma, Richard Helm,Ralph Johnson, John Vlissides
  - Définition de 23 patterns
- Design Patterns – Elements of Reusable Object-Oriented Software, Addison Wesley, 1994
- Un Design Pattern nomme, abstrait et identifie les aspects essentiels d'une structuration récurrente, ce qui permet de créer une modélisation orientée objet réutilisable

Université Nice Sophia Antipolis

Classification

- Création
  - Comment un objet peut être créé
  - Indépendance entre la manière de créer et la manière d'utiliser
- Structure
  - Comment les objets peuvent être combinés
  - Indépendance entre les objets et les connexions
- Comportement
  - Comment les objets communiquent
  - Encapsulation de processus (ex : observer/observable)

Université Nice Sophia Antipolis

Patrons de structure

- **Adapter pattern**
  - adapter une interface à une autre
- **Bridge pattern**
  - conserver l'interface d'un programme client tandis que le comportement du programme serveur peut être changé.
- **Composite pattern**
  - composer des objets ensembles (structure d'arbre)
- **Decorator pattern**
  - encapsuler dynamiquement des objets et leur fournir de nouvelles fonctions
- **Façade pattern**
  - regrouper une hiérarchie complexe en une interface simple depuis l'extérieur
- **Flyweight pattern**
  - limiter la prolifération d'instances petites, simples et similaires en factorisant hors des classes des données passées en paramètre lors d'appel de méthode
- **Proxy pattern**
  - remplacer un objet le temps de le créer

Université Nice Sophia Antipolis

Intention

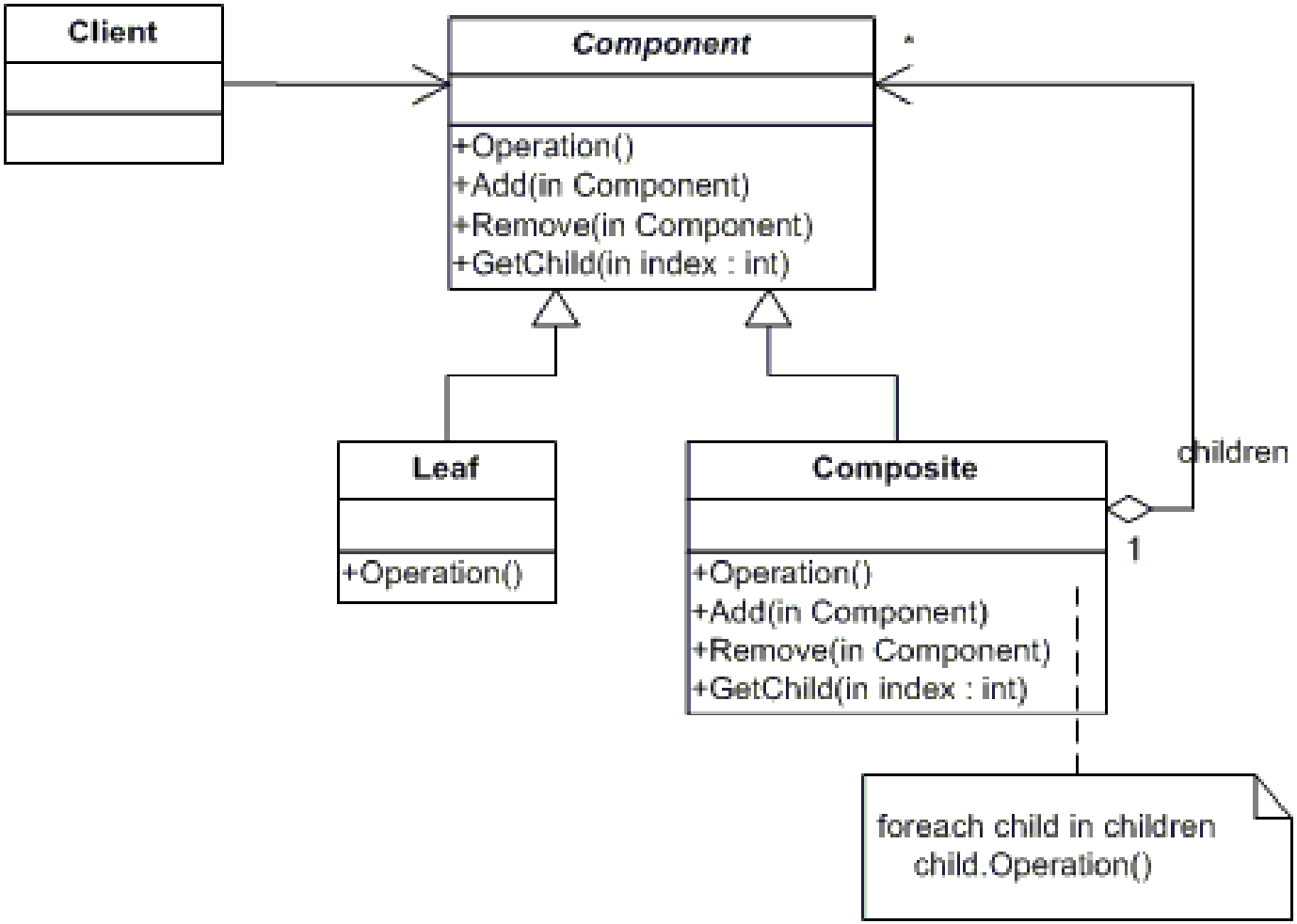
- Intention
  - Composer des objets dans des structures d'arbre pour représenter des hiérarchies composants/composés
  - Composite permet au client de manipuler uniformément les objets simples et les objets au sein de leurs compositions
- Exemple
  - Structure hiérarchique d'une entreprise : techniciens, cadres (employé)
  - JComponent / JButton, JLabel, etc. ou java.awt.Component / java.awt.Container

Champs d'application

- Une classe abstraite qui représente à la fois les primitives (contenues) et les containers
- Représentation de hiérarchie composants / composés
- Les clients doivent ignorer la différence entre les objets simples et leurs compositions (uniformité apparente: *forme spéciale de décorateur*)

```
class MultipleListener implements ActionListener {
  ArrayList<ActionListener> children;
  …
  public void actionPerformed(ActionEvent ae) {
    for(ActionListener child : children)
      child.actionPerformed(ae);
  }
}
```

Example


Client
Component
*
+Operation()
+Add(in Component)
+Remove(in Component)
+GetChild(in index : int)
Leaf
+Operation()
Composite
+Operation()
+Add(in Component)
+Remove(in Component)
+GetChild(in index : int)
children
1
foreach child in children
child.Operation()

Université Nice Sophia Antipolis

Structure

- **Component** (`java.awt.Component`)
  - déclare l'interface commune à tous les objets
  - implante le comportement par défaut pour toutes les classes
  - déclare l'interface pour gérer les composants fils
  - Définit l'interface pour accéder au composant parent (optionnel)
- **Leaf** représente une feuille (`java.awt.Button`)
  - Implantation du comportement élémentaire

Université Nice Sophia Antipolis

Structure

- **Composite** (`java.awt.Container`) définit le comportement des composants ayant des fils, stocke les fils et implémente les opérations nécessaires à leur gestion
  - Lien dans la hiérarchie
  - Comportement : fusion des comportements des fils
- Les clients (affichage graphique) utilise l'interface **Component**
  - si le receveur est une feuille la requête est directement traitée
  - sinon le **Composite** retransmet habituellement la requête à ses fils en effectuant éventuellement des traitements supplémentaires avant et/ou après

Université Nice Sophia Antipolis

Résumé

- Conséquence
  - Structure hiérarchique, simple, uniforme, général et facile à étendre pour de nouveaux objets
- Implantation
  - Maximiser l'interface de **Component**
    - Déclaration des opérations de gestion des fils
  - Ordonnancement des fils => **Iterator**
  - Parcourir la structure en profondeur => **Visitor**
  - Optionnel : Référence explicite aux parents