INF5153 – Génie logiciel : conception

Patrons GRASP

Jacques Berger

Objectifs

Introduire des principes et patrons de base

Prérequis

Aucun

GRASP

General Responsibility Assignment Software Patterns

Créateur

Nom anglais : Creator

Problème : Qui doit créer un objet A?

Créateur

Solution : La classe qui répond à un (ou plusieurs) des critères suivants

1) contient (ou agrège) des objets A
2) enregistre des objets A
3) possède les données d'initialisation de A
4) utilise étroitement des objets A

Source : UML2 et les designs patterns (Craig Larman)

Créateur

Il est généralement reconnu que les contenants créent les objets contenus

Expert en information

Nom anglais : Information Expert

Problème : Quel objet doit avoir une responsabilité en particulier?

Expert en information

Solution : L'objet qui détient l'information requise pour remplir cette responsabilité

Expert en information

Principe élémentaire de l'affectation de responsabilités

Expert en information

Expert en information ne convient pas toujours

Il existe des cas où un autre objet serait mieux placé pour effectuer une opération, pour ne pas briser la cohésion de la classe

Faible couplage

Nom anglais : Low Coupling

Problème : Comment réduire l'impact des modifications?

Faible couplage

Solution : Affecter les responsabilités de façon à minimiser le couplage

Faible couplage

Sert à l'évaluation de solutions possibles

Devant plusieurs solutions, celle avec le moins de couplage sera privilégiée

Faible couplage

Expert en information entraîne souvent un couplage faible

Faible couplage

Un couplage fort à des objets stables ou omniprésents est moins problématique

Ex. : API Java

Forte cohésion

Nom anglais : High Cohesion

Problème : Comment s'assurer que les objets demeurent compréhensibles et simples, tout en contribuant à diminuer le couplage?

Forte cohésion

Solution : Affecter les responsabilités de façon à maximiser la cohésion

Forte cohésion

Sert à l'évaluation de solutions possibles

Devant plusieurs solutions, celle avec le plus de cohésion sera privilégiée

Forte cohésion

Une faible cohésion peut parfois être justifiable

Ex. : performance, interface distante à forte granularité (charge)

Polymorphisme

Nom anglais : Polymorphism

Problème : Comment gérer des alternatives dépendantes des types?

Polymorphisme

Solution : Quand des fonctions ou des comportements connexes varient en fonction du type (classe), affectez les responsabilités – en utilisant des opérations polymorphes – aux types pour lesquels le comportement varie

Source : UML2 et les design patterns (Craig Larman)

Polymorphisme

Ne jamais tester le type d'un objet

Pas de condition sur les types pour changer le comportement

Polymorphisme

Implique la présence d'interfaces

L'interface permet d'utiliser le polymorphisme sans être coincé par une hiérarchie de classes

Polymorphisme

Il est facile d'ajouter des nouvelles implémentations sans toucher les utilisations déjà en place

Fabrication pure

Nom anglais : Pure Fabrication

Problème : Quel est l'objet qui doit être responsable lorsqu'on ne veut pas transgresser les principes de Faible couplage et Forte cohésion mais que les solutions offertes par Expert en information ne sont pas appropriées?

Source : UML2 et les design patterns (Craig Larman)

Fabrication pure

Solution : Créer une classe artificielle (non présente dans le modèle du domaine), fortement cohésive, qui contient les responsabilités souhaitées

Fabrication pure

Préserve la cohésion

Augmente le potentiel de réutilisation

Attention : Abuser de ce patron fait exploser le nombre de classes qui ne sont pas Experts, ce qui augmente le couplage

Indirection

Nom anglais : Indirection

Problème : Comment éviter le couplage entre deux objets?

Indirection

Solution : Ajouter un objet intermédiaire entre les deux objets

L'intermédiaire crée une indirection entre les objets

Indirection

Généralement motivé par un désir de Faible Couplage

Protection des variations

Nom anglais : Protected Variations

Problème : Comment concevoir des objets pour que leur instabilité n'ait pas d'impact sur les autres objets?

Protection des variations

Solution : Identifier les points de variation ou d'instabilité prévisibles. Affecter les responsabilités pour créer une interface stable autour d'eux.

Source : UML2 et les design patterns (Craig Larman)

Protection des variations

Concept ancien : masquage des informations

Contrôleur

Selon le principe de Séparation Modèle-Vue, les objets de l'interface utilisateur ne doivent pas contenir les objets du domaine

L'interface utilisateur doit déléguer le traitement de la requête à la couche du domaine

Contrôleur

Nom anglais : Controller

Problème : Quel objet a la responsabilité de recevoir les instructions de la couche Présentation?

Contrôleur

Solution : Donner la responsabilité à un objet qui représente le système ou qui représente un cas d'utilisation

Contrôleur

L'objet Contrôleur reçoit les commandes de la présentation et manipule les objets du domaine

Contrôleur

Potentiel de réutilisation plus élevé puisque la logique d'affaires n'est pas manipulée par la couche de présentation

Contrôleur

Un contrôleur avec trop de responsabilités deviendra peu cohésif

Ex. :

- Un seul contrôleur pour gérer tous les événements
- Le contrôleur effectue des tâches lui-même
- Le contrôleur possède plusieurs attributs et de la logique applicative

Contrôleur

Il est pertinent d'avoir plusieurs contrôleurs dans un logiciel

On pourrait avoir un contrôleur par cas d'utilisation

Plus loin...

UML 2 et les design patterns, 3ème édition
Craig Larman
Pearson, 2005
Chapitres 16 et 22