

INFO-H-200

Programmation orientée objet

Séance d'exercices 9
Design Patterns

Université libre de Bruxelles
École polytechnique de Bruxelles

Professeur : Hugues Bersini

2015-2016

Bonnes pratiques de conception OO

"Controlling complexity is the essence of computer programming" (Brian Kernighan)

Quelques principes :

- ① Expert d'information (celui qui sait le fait)
- ② Couplage faible
- ③ Forte cohésion
- ④ Loi de Demeter (Don't talk to strangers)

Ces principes sont tirés des principes GRASP.

Bonnes pratiques de conception OO

Couplage faible

Le couplage mesure la dépendance entre des classes.

Dépendance : héritage, attributs, arguments,...

Faible : réduit l'impact des changements dans une classe

Forte cohésion

La cohésion mesure la compréhensibilité des classes.

Cohésion : une classe doit avoir des responsabilités cohérentes.

Forte : faciliter la maintenance et l'utilisation d'une classe

Bonnes pratiques de conception OO

Loi de Demeter Cas particulier du couplage faible

La loi dit qu'on ne peut envoyer des messages qu'à :

- un argument passé
- un objet que l'on a créé
- self et super

Patron de conceptions

Design Patterns

Patron de conceptions

Concept de génie logiciel visant à résoudre des problèmes récurrents d'architecture et de conception logiciel.

Formalisation de bonnes pratiques.

Capitalisation de l'expérience appliquée à la conception.

Trois grandes familles :

- ❶ **Patrons de constructions** : Instancier et configurer des classes et des objets
- ❷ **Patrons structuraux** : Organiser les classes d'un programme en séparant l'interface de l'implémentation.
- ❸ **Patrons comportementaux** : Organiser les objets pour que ceux-ci collaborent.

Patron de conceptions

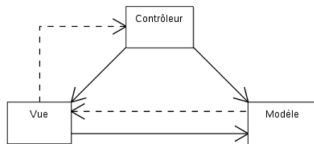
Modèle-Vue-Contrôleur

Patron de conceptions : Modèle-Vue-Contrôleur

Ce n'est pas un patron en tant que tel, mais une méthode de conception pour organiser l'architecture d'une application avec l'interface graphique.

On divise l'application entre :

- **Modèle** : Les données, la connection avec la DB et les notifications de la vue
- **Vue** : Présentation, interface, notification du contrôleur, répond aux notifications du modèle
- **Contrôleur** : Logique de contrôle, synchronisation, répond aux notifications de la vue.



En pointillés, les notifications et en plein, les associations.

Patron de conceptions : Modèle-Vue-Contrôleur

Modèle :

Gestion du comportement de l'application, avec le traitement des données, la gestion des interactions avec la base de données, la garantie de l'intégrité des données. Le modèle est le seul à modifier les données et est donc garant de ces données.

Il signale à la vue que les données ont changé (via un événement)

Interface :

Interface avec laquelle l'utilisateur interagit. Celle-ci a deux rôles :

- 1 Présenter les résultats renvoyés par le modèle.
- 2 Recevoir les actions des utilisateurs (click et autres...) et transmettre au contrôleur (via un événement). Elle n'effectue aucun traitement et ne fait qu'afficher les données du modèle.

Patron de conceptions : Modèle-Vue-Contrôleur

Contrôleur :

Gestion des événements pour mettre à jour la vue ou le modèle et les synchroniser. Il reçoit les notifications de la vue, met à jour le modèle au besoin et avertit la vue que les données ont changé pour que celle-ci se mette à jour.

Le contrôleur ne modifie aucune donnée et est composé des méthodes de gestion d'événements.

Patron de conceptions : Modèle-Vue-Contrôleur

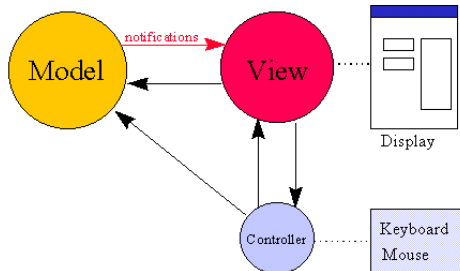
Exemple de cheminement :

- 1 L'utilisateur clique sur quelque chose (ce qui déclenche une notification)
- 2 La requête est analysée par le contrôleur.
- 3 Le contrôleur demande au modèle de faire les changements requis.
- 4 Le contrôleur renvoie la vue adaptée si le modèle ne l'a pas déjà fait.

Règles de base

- Le modèle ne peut pas connaître la vue.
- La vue peut connaître le modèle et l'utilise en lecture seule.

Patron de conceptions : Modèle-Vue-Contrôleur



https://www.mimuw.edu.pl/sl/teaching/00_01/Delfin_EC/Patterns/Figures/MVC1.gif

Patron de conceptions

Observateur

Patron de conceptions : Observateur (structurel)

Problème :

Plusieurs objets A, les observateurs sont notifiés lorsqu'un objet B, le sujet, change d'état.

Solution :

Remplacer la dépendance de B vers A par une dépendance sur une interface minimaliste.

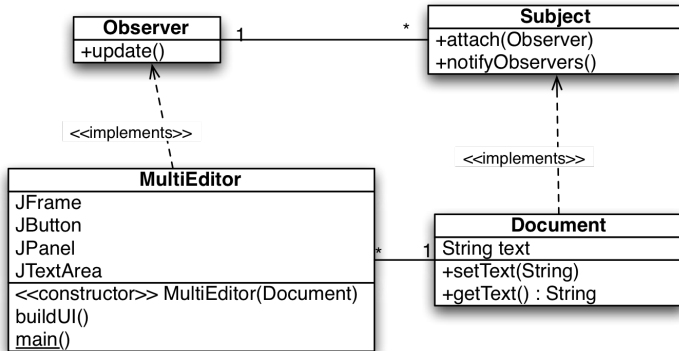
Conséquences :

Couplage abstrait, broadcast, mise à jour en cascade

L'observateur observe des observables. En cas de notification, les observateurs effectuent une action en fonction des informations qui viennent des observables.

La notion d'observateur/observable permet de coupler des modules de façon à réduire les dépendances aux seuls phénomènes observés. (voir aussi le cours sur les événements)

Patron de conceptions : Observateur (structurel)



Patron de conceptions

Commande

Patron de conceptions : Command (comportemental)

Problème :

Annuler et/ou sauvegarder des actions sur des objets.

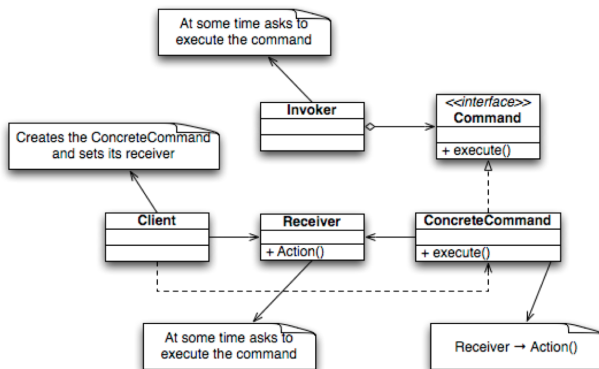
Solution :

Un objet encapsule toute l'information nécessaire pour appeler une méthode plus tard (nom, propriétaire, paramètres,...)

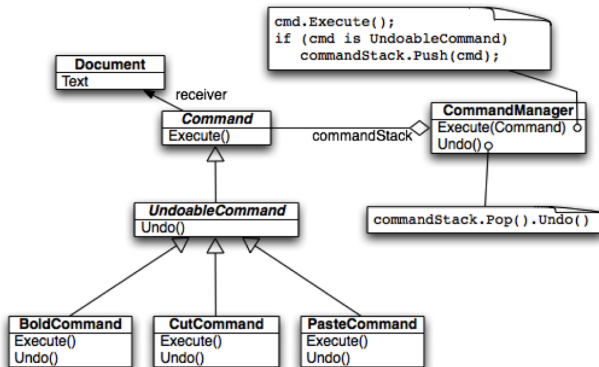
Conséquences :

Support de l'historique, de l'annulation de commandes,...

Patron de conceptions : Command (comportemental)



Patron de conceptions : Command (comportemental)



Patron de conceptions : Command (comportemental)

