Génie logiciel pour l'entreprise: Outils pour l'ajout de fonctionnalités à un code existant (Patrons de conception ou design patterns)

Jean-Baptiste Apoung Kamga

Cours M2 IM
Université Paris-sud XI
This is a draft version

Plan

Définition et exemples

Dans le développement logiciel, il peut être quelques fois judicieux d'exploiter les solutions proposées par d'autres développeurs. Ces solutions ne se réduisent pas seulement en un ensemble de codes, mais aussi en un ensemble de modèles ou de raisonnements semblables observées lors de la phase de conception. L'objectif principale étant la réduction de la phase de conception qui est très souvent coûteuse.

Les motifs de conceptions sont groupés en trois grandes classes:

Modèles de création

Ils permettent de rendre le système indépendant de la façon dont ses objets sont créés, combinés et concrétisés.

Modèles structuraux

Ils étudient la façon de composer les classes et les objets pour réaliser des structures plus importantes.

Modèles comportementaux

Ils traitent des algorithmes et de l'affectation des responsabilités entre les objets. Ils ne décrivent pas seulement, des modèles d'objets et de classes, mais également des modèles de communication entre eux.

Modèles Créateurs (5)

- Fabrique Abstraite (*Abstract Factory*) Fournit une interface, pour créer des familles d'objets apparentés ou dépendants, sans avoir à spécifier leurs classes concrètes.
- Monteur (*Builder*) Dans un objet complexe, dissocie sa construction de sa représentation, de sorte que, le même procédé de construction puisse engendrer des représentations différentes.
- Fabrication (*Factory Method*) Définit une interface pour la création d'un objet, tout en laissant à des sous-classes le choix de la classe à instancier. Une fabrication permet de déférer à des sous-classes les instanciations d'une classe.
- Prototype (*Prototype*) Spécifie les espèces d'objets à créer, en utilisant une instance de type prototype, et crée de nouveaux objet par copies de ces prototype.
- Singleton (*Singleton*) Garantit qu'une classe n'a qu'une seule instance, et fournit à celle-ci un point d'accès de type global.

Modèles Structuraux (7)

- Adaptateur (*Adaptor*) Convertit l'interface d'une classe en une interface distincte, conforme à l'attente de l'utilisateur. L'adaptateur permet à des classes de travailler ensemble, qui n'auraient pas pu le faire autrement pour des causes d'interfaces incompatibles.
- Pont (*Bridge*) Découpe l'abstraction de son implémentation associée, afin que les deux puisse être modifiées indépendamment.
- Objet Composite (*Composite*) Organise les objets en structure arborescente représentant la hiérarchie de bas en haut. Le composite permet aux utilisateurs de traiter les objets individuels et des ensembles organisés de ces objets de la même façon.
- Décorateur (*Decorator*) Attache des responsabilités supplémentaires à un objet de manière dynamique. Il permet une solution alternative pratique pour l'extension de fonctionnalités, à celle de dérivation de classes.
- Façade (*Facade*) Fournit une interface unifiée pour un ensemble d'interfaces d'un sous-système. Façade définit une interface de plus haut niveau, qui rend le sous-système plus facile à utiliser.
- Poids-Mouche (*Flyweight*) Assure en mode partage le support efficace d'un grand nombre d'objets à fine granularité.
- Procuration (*Proxy*) Fournit un subrogé ou un remplaçant d'un objet pour en contrôler l'accès.

- Chaîne de Responsabilité (*Chain of Responsability*) Permet d'éviter de coupler l'expéditeur d'une requête à son destinataire, en donnant la possibilité à plusieurs objets de pendre en charge la requête. Pour ce faire, il chaîne les objets récepteurs, et fait passer la requête tout au long de cette chaîne jusqu'à ce qu'un des objets la prenne en charge.
- Commande (*Command*) Encapsule une requête comme un objet, ce qui permet de faire un paramétrage des clients avec différentes requêtes, files d'attente, ou historique de requêtes et d'assurer le traitement des opérations réversibles.
- Interprète (*Interpreter*) Dans un langage donné, il définit une représentation de sa grammaire, ainsi qu'un interprète qui utilise cette représentation pour analyser la syntaxe du langage.
- Itérateur (*Iterator*) Fournit un moyen pour accéder en séquence aux éléments d'un objet de type agrégat sans révéler sa représentation sous-jacente.
- Médiateur (*Mediator*) Définit un objet qui encapsule les modalités d'interaction de divers objets. Le mediateur favorise les couplages faibles, en dispensant les objets d'avoir à faire référence explicite les uns des autres; de plus il permet de modifier une relation indépendamment des autres.

- Mémento (*Memento*) Sans violer l'encapsulation, acquiert et délivre à l'extérieur une information sur l'état interne d'un objet, afin que celui-ci puisse être rétabli ultérieurement dans cet état.
- Observateur (*Observator*) Définit une corrélation entre objets du type un à plusieurs, de façon que, lorsqu'un objet change d'état, tous ceux qui en dépendent en soient notifiés et mis à jour automatiquement.
- État (*State*) Permet à un objet de modifier son comportement lorsque son état interne change. L'objet paraîtra changer de classe.
- Stratégie (*Strategy*) Définit une famille d'algorithmes, encapsule chacun d'entre eux, et les rend interchangeables. Une stratégie permet de modifier un algorithme indépendamment de ses clients.
- Patron de Méthode (*Template Method*) Définit le squelette de l'algorithme d'une opération, en déléguant le traitement de certaines étapes à des sous-classes. Le patron de méthode permet aux sous-classes de redéfinir certaines étapes d'un algorithme sans modifier la structure de l'algorithme.
- Visiteur (*Visitor*) Représente une opération à effectuer sur les éléments d'une structure d'objets. Le visiteur permet de définir une nouvelle opération sans modifier les classes des éléments sur lesquels il opère.

Design patterns, les plus courants et leur diagramme

Les designs les plus courants

Voir TP

Les diagrammes

Voir TP