Intention métier

L'intention principale du pattern Proxy défini par le "Gang of Four" (GoF), est de "Fournir un substitut ou un représentant d'un autre objet afin de contrôler l'accès à celui-ci."

Il s'agit d'intercaler un objet (le proxy) entre un client et un objet "réel" (souvent coûteux à créer ou à accéder) pour gérer cet accès.

UML Diagramme de classes (Générique)

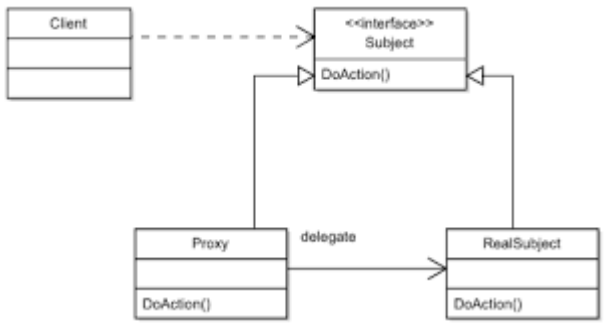
Le diagramme de classes générique du pattern tourne autour de trois composants principaux :

1. Un Client qui souhaite accéder à un objet.
2. Une interface Subject (ou classe abstraite) qui définit les opérations communes.
3. Deux classes concrètes qui implémentent Subject :

    -RealSubject : L'objet réel qui contient la logique métier ou la ressource coûteuse.

    -Proxy : Le substitut qui détient une référence vers le RealSubject.

Le Client interagit uniquement avec l'interface Subject, sans savoir s'il s'adresse au Proxy ou au RealSubject.



Explication des classes participantes

Subject (Sujet)

-Rôle : Définit l'interface commune que le RealSubject et le Proxy doivent implémenter.

-Objectif : Permettre au Client de traiter le Proxy de la même manière que le Sujet Réel (respectant le principe de substitution de Liskov).

RealSubject (Sujet Réel)

-Rôle : C'est l'objet final qui réalise le "vrai travail".

-Objectif : Contenir la logique métier ou la ressource dont l'accès doit être contrôlé (par exemple, un objet coûteux à instancier, une connexion réseau, etc.).

Proxy (Substitut)

-Rôle : C'est l'intermédiaire qui contrôle l'accès au RealSubject. Il implémente la même interface Subject.

-Objectif : Il décide s'il doit ou non déléguer l'appel au RealSubject. Il peut ajouter des traitements complémentaires comme :

-Le chargement paresseux (Lazy Loading) : ne créer le RealSubject qu'au moment où il est vraiment nécessaire.

-Le contrôle d'accès (Proxy de protection) : vérifier si le client a les droits.

-La mise en cache (Proxy de cache) : stocker les résultats d'opérations coûteuses.

-La gestion réseau (Proxy distant) : masquer la complexité d'un appel réseau.

Exemple en contexte métier (Galerie d'images)

Un exemple concret est une application de gestion d'images.

Problème : Si une galerie doit afficher des dizaines d'images, et que chaque objet Image charge un fichier lourd (plusieurs Mo) lors de sa création, l'application sera très lente au démarrage et consommera énormément de mémoire.

-> Solution (Proxy) :

-On crée une interface Image avec une méthode afficher().

-On crée la classe ImageReelle (le RealSubject) qui implémente Image et charge réellement le fichier lourd depuis le disque lors de son instanciation.

-On crée la classe ProxyImage (le Proxy) qui implémente aussi Image.

-La galerie (le Client) manipule une liste d'objets ProxyImage.

-Lorsqu'un ProxyImage doit afficher(), il vérifie s'il a déjà une instance de ImageReelle.

- Si non, il la crée (charge le fichier lourd à ce moment précis).
- Il délègue ensuite l'appel afficher() à ImageReelle.

Résultat : Les images ne sont chargées qu'au moment où elles sont réellement affichées, optimisant ainsi les performances.

Démonstration de la SOLIDité

Le pattern Proxy illustre plusieurs principes SOLID :

- S (Single Responsibility Principle - Responsabilité unique) :

  -Le RealSubject a la responsabilité unique de la logique métier (ex: le traitement de l'image).

  -Le Proxy a la responsabilité unique de la gestion de l'accès (ex: le chargement paresseux, le cache, la sécurité).

- O (Open/Closed Principle - Ouvert/Fermé) :

  -On peut ajouter de nouvelles logiques de contrôle (un proxy de cache, un proxy de sécurité) sans jamais modifier le code du RealSubject. Le système est ouvert à l'extension (ajout de proxys) mais fermé à la modification (le RealSubject est stable).

- L (Liskov Substitution Principle - Substitution de Liskov) :

  -Le Client dépend uniquement de l'interface Subject. Il peut donc manipuler un Proxy ou un RealSubject de manière interchangeable, sans que cela ne modifie le comportement attendu du programme.