

Pattern PROXY



Noah GERMANEAU, Scotty DELMART,
Vianney BISTON, Roman JEFF

Sommaire

1. Introduction
 2. Mise en contexte
 3. Première modélisation (naïve)
 4. Deuxième modélisation
 5. Généralisation
 6. Détails techniques
 7. SOLID et maintenabilité
 8. Limites du pattern
 9. Comparaison
 10. Deuxième contexte d'application
 11. Live coding
 12. Bibliographie / Webographie
 13. QCM
-

Qu'est-ce qu'un DESIGN PATTERN ?

Design Patterns

Elements of Reusable
Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



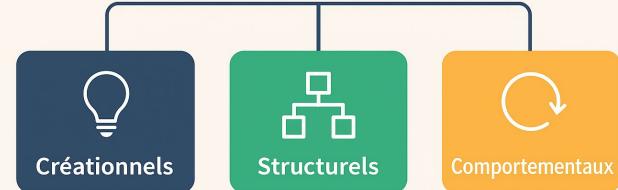
Foreword by Grady Booch

Elements of Reusable Object-Oriented Software

Définition :

“Solutions types à des problèmes récurrents
de conception orientée objet”

Design Patterns



Un besoin concret : gestion d'un petit hôtel



Client



Chambre d'hôtel (RealSubject)



Réceptionniste (proxy)

Vérifie l'identité



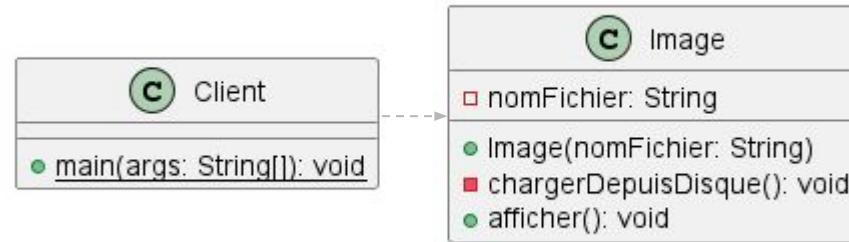
Contacte le service de nettoyage

Redirige le client si tout est ok

Une première conception inefficace

Liste des problèmes :

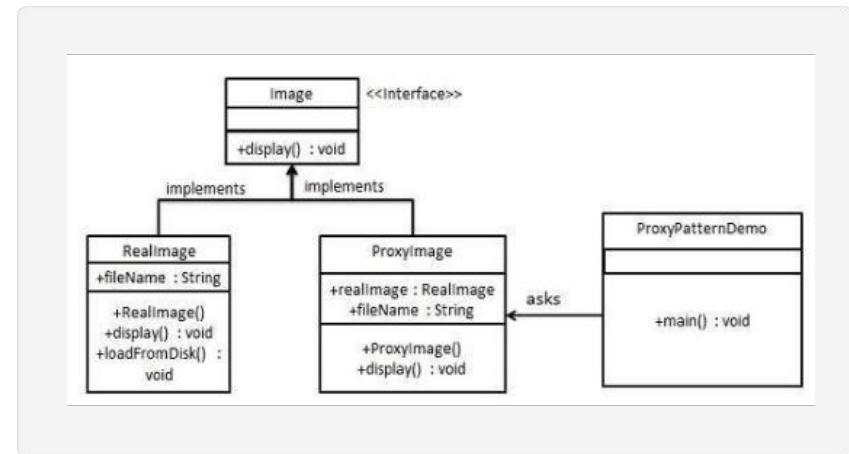
- Chargement inutile
- Couplage fort
- Difficulté à rajouter du contrôle d'accès



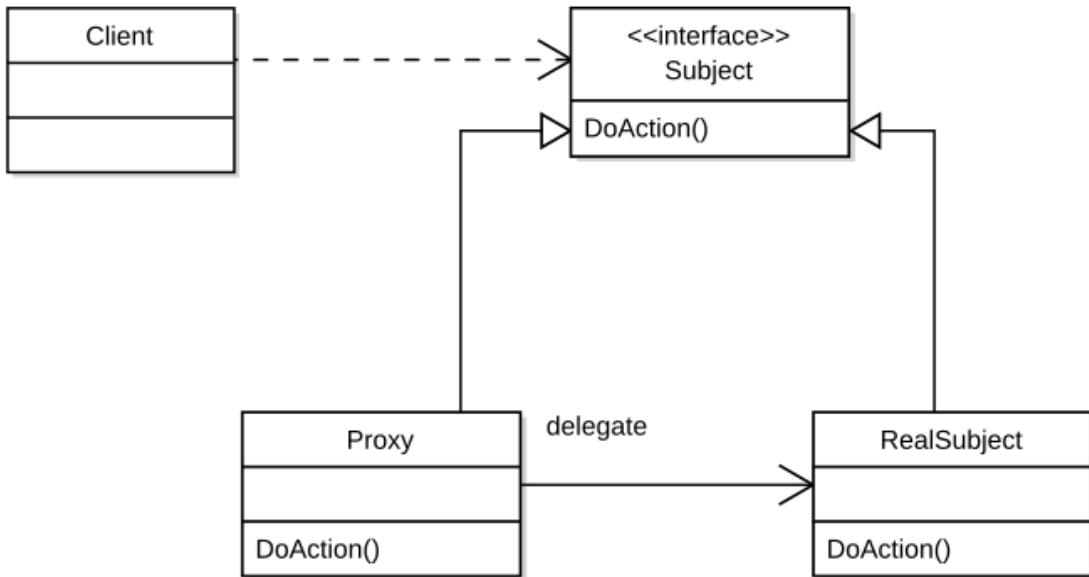
Amélioration avec le pattern Proxy

Avantages de la solution :

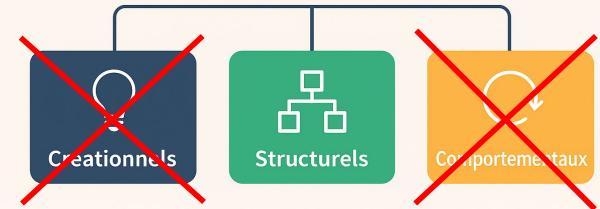
- **Contrôle d'accès** : Le Proxy intercepte l'appel et décide quand instancier l'objet réel.
- **Découplage** : Le client n'interagit qu'avec l'interface, ignorant l'implémentation réelle.
- **Flexibilité** : Le Proxy peut intégrer du Lazy Loading, du Cache, du Logging, etc.



Le pattern Proxy



Design Patterns



Fonctionnement du Proxy

Le Déroulement d'un appel :

■ PHASE 0 : OBTENTION DU PROXY

Client → Factory : getService() ou createProxy()

Factory : Crée et configure le Proxy

Factory → Client : Retourne l'instance du Proxy

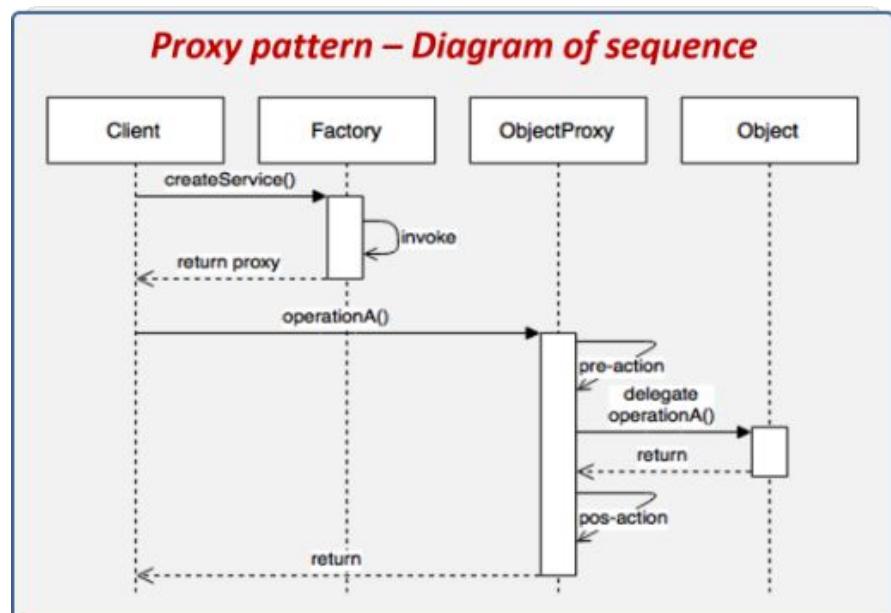
■ PHASE 1 : APPEL MÉTIER

Client → Proxy : Appel méthode via l'interface

Proxy : Exécute son rôle (vérification, cache, lazy loading)

Proxy → RealSubject : Instancie et appelle si nécessaire

Proxy → Client : Transmet le résultat



Proxy et principes SOLID

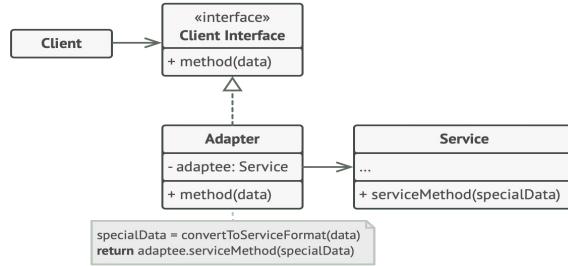
Le Proxy contribue à un code robuste et maintenable :

- S** **Responsabilité Unique** : Le Proxy gère l'accès, le “RealSubject” gère la logique métier.
Séparation des préoccupations.
- O** **Ouvert/Fermé** : On peut ajouter un Proxy pour changer le comportement d'accès sans modifier la classe du “RealSubject”.
- L** **Substitution de Liskov** : Ils partagent la même interface (“Subject”), le Proxy est donc interchangeable avec le “RealSubject”.

Limites du pattern Proxy

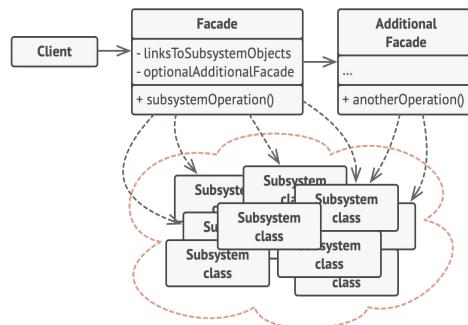
-  **Surcoût en performance (Indirection)** : Le fait de passer par une couche supplémentaire (le Proxy) introduit une micro-latence.
-  **Complexité accrue** : Plus de classes à maintenir et à comprendre dans la structure du projet.
-  **Difficulté de débogage** : Suivre le flux d'exécution devient plus complexe (Client -> Proxy -> RealSubject).

Proxy vs Adaptateur, Façade, Décorateur



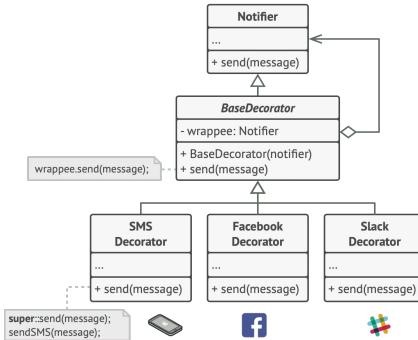
Adaptateur

Transforme une interface pour la rendre compatible avec une autre, mais ne contrôle pas l'accès.



Façade

Simplifie l'accès à un ensemble complexe de classes, mais ne se substitue pas à objet.



Décorateur

Ajoute des comportements dynamiques à un objet sans modifier sa structure interne, alors que Proxy intercepte les appels.

Application dans un autre contexte

Proxy dans un jeu vidéo

Contexte : Communication avec un serveur distant



Objectif: Le ProxyServeur contacte le ServeurJeu que lorsque c'est nécessaire



Live coding

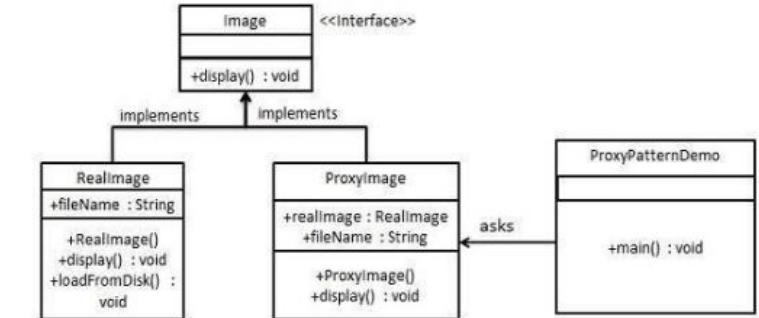
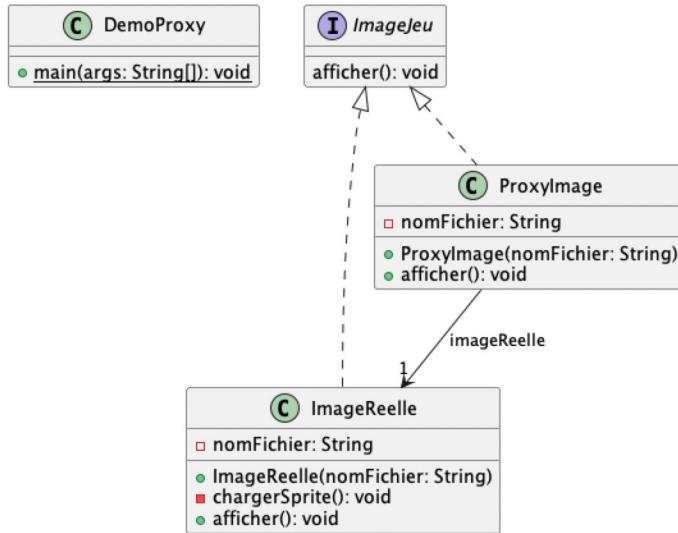
The screenshot shows an IDE interface with a Java code editor. The code implements the `Subject` interface, which includes methods for `attach` and `detach`. The `attach` method creates a new `Imageable` object and adds it to a `list`. The `detach` method removes the last element from the `list`. A tooltip is visible over the `Imageable` type in the `attach` method, listing several options such as `Imageable`, `Imageable`, `Imageable`, `Imageable`, `Imageable`, `Imageable`, `Imageable`, `Imageable`, and `Imageable`.

```
1 package test;
2
3 public class ProxyImage implements Subject {
4     private Imageable monitor;
5     private Imageable imageable;
6
7     public void attach() {
8         this.imageable = monitor;
9     }
10
11     public void detach() {
12         if (imageable != null) {
13             System.out.println("Detaching " + imageable);
14         }
15     }
16 }
```

Live coding

Comparaison des diagrammes

java:jeu.ProxyImage



Bibliographie / Webographie

- *Design Patterns: Elements of Reusable Object-Oriented Software*, Gamma et al., 1994
- *Refactoring.Guru* – Proxy Pattern
- *SourceMaking.com* – Design Patterns
- Cours R304 – IUT Informatique



QCM

