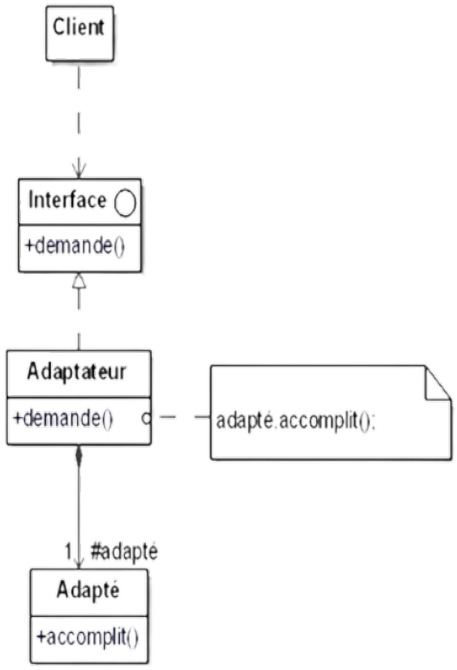
Fiche résumé

Pattern Adapter

Le pattern Adapter permet de rendre compatible deux interfaces qui ne le sont pas initialement.

Il agit comme un convertisseur entre deux classes, afin que le client puisse utiliser un service dont l'interface ne correspond pas à ce qu'il attend.

Il permet donc à des composants déjà existants d'être réutilisés sans modification.



Il y a 2 types de structures pour ce pattern :

- Adapter de classe qui est utiliser dans un cas d'héritage multiple
- Adapter objet qui est celui qu'on utilise le plus souvent

Quand utiliser ce pattern ?

- Lorsqu'on veut intégrer une classe existante dans un système dont l'interface est différente.
- Quand on ne peut pas modifier le code d'origine (par exemple, une bibliothèque externe).
- Quand on veut réutiliser du code sans le dupliquer ou le réécrire.

Principe SOLID	
SRP	L'Adapter a une seule responsabilité : convertir une interface en une autre.
OCP	On peut ajouter de nouveaux Adapters sans modifier le code existant
LSP	L'Adapter peut remplacer n'importe quelle implémentation de l'interface cible sans problème.
ISP	Le client utilise uniquement les méthodes dont il a besoin via une interface adaptée.
DIP	Le client dépend d'une abstraction, pas d'une implémentation concrète.

-

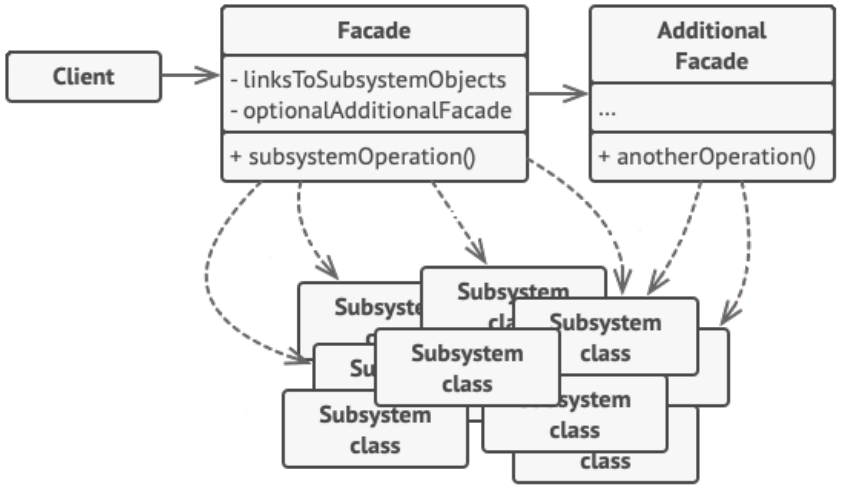
Limites du pattern Adapter :

- Peut alourdir le code si les interfaces sont très proches.
- Risque de multiplier les adaptateurs si plusieurs systèmes externes doivent être intégrés.
- Possibilité de chaînes d'adaptations complexes, difficiles à maintenir.

- Peut masquer un problème d’architecture si utilisé pour compenser un mauvais design global.

Pattern Façade

Le pattern Façade fournit une interface simplifiée et unifiée à un ensemble de classes ou de sous-systèmes complexes.
Il permet au client d’interagir avec tout un système via un point d’accès unique, sans connaître les détails internes.
C’est une manière de rendre le code plus clair, plus lisible et plus facile à maintenir.



Quand utiliser ce pattern ?

- Quand un système devient trop complexe à manipuler directement.
- Lorsqu’on veut masquer la complexité d’un sous-système derrière une interface unique.
- Pour réorganiser du code et réduire les dépendances entre les différentes classes.

Principe SOLID	
SRP	la Façade a une seule responsabilité, simplifier l’accès à un système.
OCP	on peut ajouter ou modifier des classes internes sans toucher à l’interface de la façade.
LSP	respecte automatiquement ce principe, car il n’est pas destiné à être hérité ou substitué dans une hiérarchie de classes.
ISP	Il expose souvent une interface large et rempli pour simplifier l’usage du client, ce qui peut aller à l’encontre de ce principe. Ce principe est donc généralement non suivi, tout dépend de l’implémentation faite.
DIP	le client dépend de la façade (abstraction), pas des classes internes.

Limites du pattern Façade :

- Peut devenir trop complexe si elle regroupe trop de fonctionnalités.
- Risque de surcharger la classe principale en centralisant trop d’appels internes.
- La création de plusieurs façades secondaires peut compliquer la structure globale.
- Peut augmenter la complexité architecturale au lieu de réellement la réduire.

Lien entre Adapter et Façade

L’Adapter sert à rendre une interface existante compatible avec celle qu’attend le client, tandis que la Façade crée une interface simplifiée pour faciliter l’accès à un ensemble de classes. L’Adapter agit généralement sur un seul composant, alors que la Façade s’applique à un sous-système complet.