Pattern Adapter

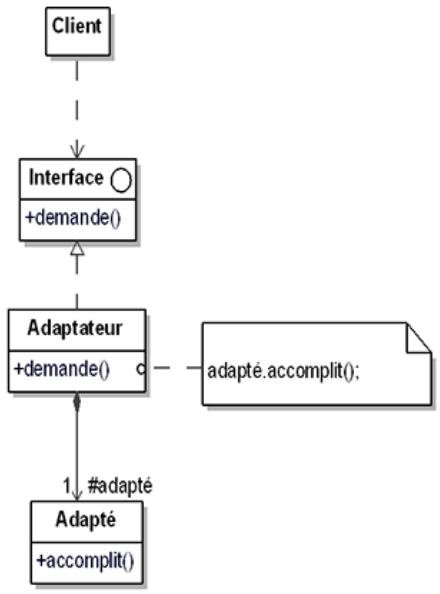
Le pattern **Adapter (structurel)** est utilisé pour permettre à des classes ayant des interfaces incompatibles de travailler ensemble. Il agit comme un "pont" entre deux interfaces incompatibles. Ce pattern est souvent employé lorsque l'on veut intégrer une nouvelle classe dans un système existant, mais que cette nouvelle classe n'a pas la même interface que les autres composants.

Un *adaptateur*. C'est un objet spécial qui convertit l'interface d'un objet afin qu'un autre objet puisse le comprendre.

Deux types de Structure:

-**Adaptateur objet** (comme ici)

-**Adaptateur de classe** (dans un cas d'héritage multiple comme en c++ par exemple)



Quand l'utiliser ?

- Lorsque tu souhaites intégrer une nouvelle classe dans un code existant, mais que les interfaces de ces classes ne sont pas compatibles.
- Lorsque tu souhaites réutiliser une classe qui a une interface différente de celle attendue par un autre module.

Principe SOLID	État
SRP	Respecté (Adaptateur = 1 responsabilité)
OCP	Respecté (Pour adapter une nouvelle classe, nouvel adaptateur (extension))
LSP	Respecté (L'adaptateur = « convertisseur » pour rendre les classes substituable)
ISP	Respecté (1 Adaptateur pour 1 interface)
DIP	Respecté (L'adapter dépend de l'interface (abstraction), le client dépend de l'abstraction fournit par l'adapter)

Adapter agit comme une couche intermédiaire pour rendre compatibles des interfaces incompatibles sans modifier leur code. Cela centralise les adaptations, ce qui facilite les mises à jour sans impacter le reste du système.

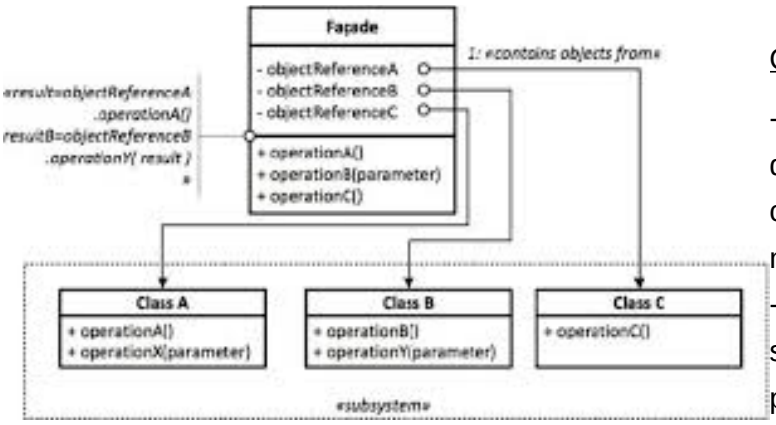
Limites du pattern Adapter

- **Complexité additionnelle** : La création d'un adaptateur pour chaque interface incompatible peut ajouter de la complexité au système.
- **Performance** : Les appels de méthode supplémentaires introduits par l'adaptateur peuvent légèrement affecter la performance, bien que ce soit souvent négligeable.
- **Rigidité en cas de changement** : Si l'interface cible ou celle de la classe adaptée change, l'adaptateur doit être modifié pour continuer à fonctionner, ce qui peut nuire à la flexibilité du système.

Pattern Façade

Façade est un patron de conception structurel qui procure une interface offrant un accès simplifié à une librairie, un framework ou à n'importe quel ensemble complexe de classes.

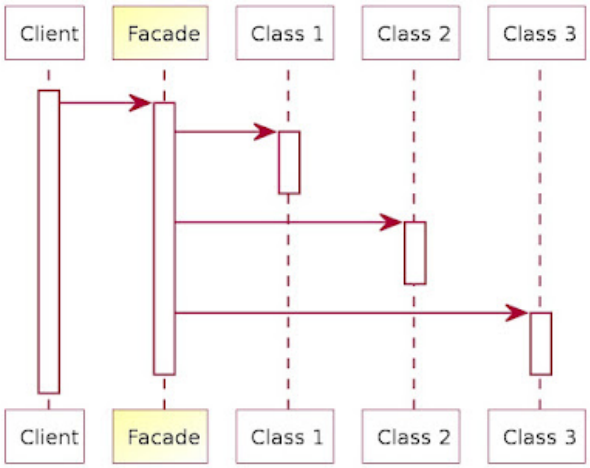
La **Façade** définit une nouvelle interface pour les objets existants, elle s'utilise pour un sous-système complet d'objets.



Quand l'utiliser ?

- Lorsque tu veux simplifier l'utilisation d'une bibliothèque ou d'un ensemble de classes complexes pour des utilisateurs non-experts.
- Lorsque tu souhaites décomposer des systèmes complexes en sous-systèmes pour une meilleure organisation.

Principe SOLID	État
SRP	Respecté elle a une responsabilité unique : fournir une interface simplifiée à un sous-système complexe
OCP	Respecté en partie la classe est ouverte aux extensions. Elle peut ne pas être complètement fermée aux modifications
LSP	Non applicable pas d'héritage
ISP	Non respecté car elle expose une interface globale qui peut contenir des méthodes inutiles pour certains clients
DIP	Respecté (Façade = couche d'abstraction entre client et sous-systèmes)



Façade masque la complexité d'un sous-système en offrant une interface simplifiée. Cela rend le code client plus clair et isolé des détails internes, ce qui facilite les modifications et réduit les risques de régression.

Limites du pattern Façade

- **Couplage indirect** : Bien que la Façade masque le sous-système, elle crée un point de dépendance unique. Si le sous-système change, la Façade doit être adaptée, ce qui entraîne un couplage indirect.
- **Complexité masquée** : En masquant la complexité, la Façade peut rendre plus difficile le débogage ou l'accès aux fonctionnalités avancées du sous-système, ce qui limite la flexibilité si des comportements spécifiques sont nécessaires.
- **Risque de surcharge** : Si la façade tente de couvrir trop de fonctionnalités, elle peut devenir trop complexe elle-même, annulant ainsi son but premier d'être une interface simplifiée.