Détail / Explication des Designs Patterns

Design Pattern choisi :

Le **MVC** va servir pour la structure générale de l’application, avec le modele, view, controller.

* Modèle : Le modèle représente la forme des données et la logique métier. Il maintient les données de l'application. Les objets de modèle récupèrent et stockent l'état du modèle dans une base de données.

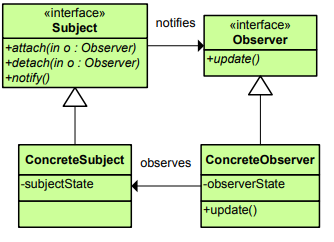
Le modèle est une logique de données et d'entreprise.

* View : View est une interface utilisateur. Afficher les données d'affichage en utilisant le modèle pour l'utilisateur et leur permet également de modifier les données.

La vue est une interface utilisateur.

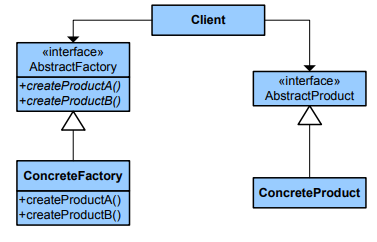
* Contrôleur : le contrôleur gère la demande de l'utilisateur. Généralement, l'utilisateur interagit avec View, ce qui déclenche à son tour la demande d'URL appropriée. Cette demande sera gérée par un contrôleur. Le contrôleur rend la vue appropriée avec les données du modèle en réponse.

Le contrôleur est un gestionnaire de demandes.



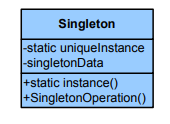
**Observer** présent dans le MVC. Il servira pour afficher dans les logs les différentes événements notifé.

Ce patron établit une relation un à plusieurs entre des objets, où lorsqu'un objet change, plusieurs autres objets sont avisés du changement.



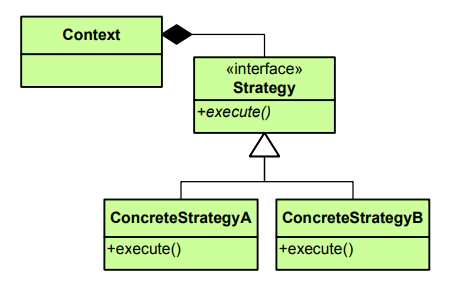
**Abstract Factory** pour la fabrication des clients.

Fournir une interface unique pour instancier des objets d'une même famille sans avoir à connaître les classes à instancier

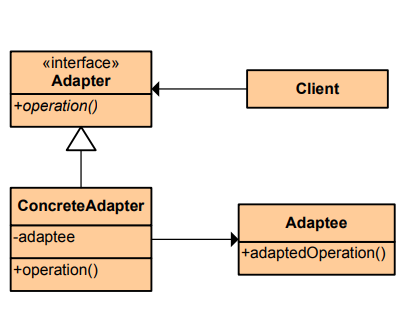


**Singleton** pour la cuisine et la salle

Ce patron vise à assurer qu'il n'y a toujours qu'une seule instance d'une classe en fournissant une interface pour la manipuler.

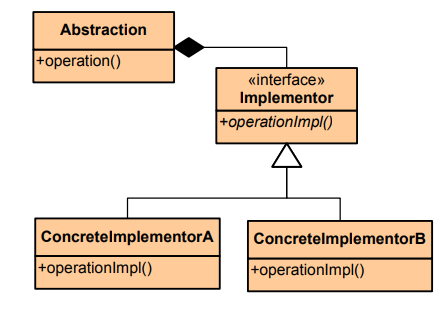
**Strategy**

Dans ce patron, une famille d'algorithmes est encapsulée de manière qu'ils soient interchangeables. Les algorithmes peuvent changer indépendamment de l'application qui s'en sert.



**Adapter**

Ce patron convertit l'interface d'une classe en une autre interface exploitée par une application. Permet d'interconnecter des classes qui sans cela seraient incompatibles.



**Bridge**

Ce patron permet de découpler une abstraction de son implémentation, de telle manière qu'ils peuvent évoluer indépendamment.