PARADOT Charles

LECLAIR Simon

DAVEAU Quentin

Explications de tous les DP utilisés dans l'application

Pour l'architecture générale :

MVC, le Modèle View Controler est un design pattern qui permet d'organiser le code source, il permet de séparer la logique du code en 3 parties distinctes.

- Le Modèle, il gère les données du code source. Son rôle principal est de récupérer les informations brutes dans la base de données, les organiser, et les assembler pour qu'elles soient traitées par le contrôleur. C'est dans cette partie du code que l'on retrouve par exemple nos requêtes SQL qui seront envoyées à la base de données SQL Server.
- Le Contrôleur, c'est la partie du code qui permet de « prendre des décisions ». C'est un intermédiaire entre le modèle (c'est un peu le côté serveur) et la vue (c'est un peu l'interface client). Il va demander au modèle les données à analyser, prendre des décisions, et renvoyer les objets à afficher sur la vue. Il gère aussi le droit d'accès, si l'utilisateur a le droit de voir la page ou non.
- La Vue, c'est l'affichage de l'application. La vue ne fait en soi presque aucun calcul, et se contente d'afficher les variables qu'elle doit afficher.

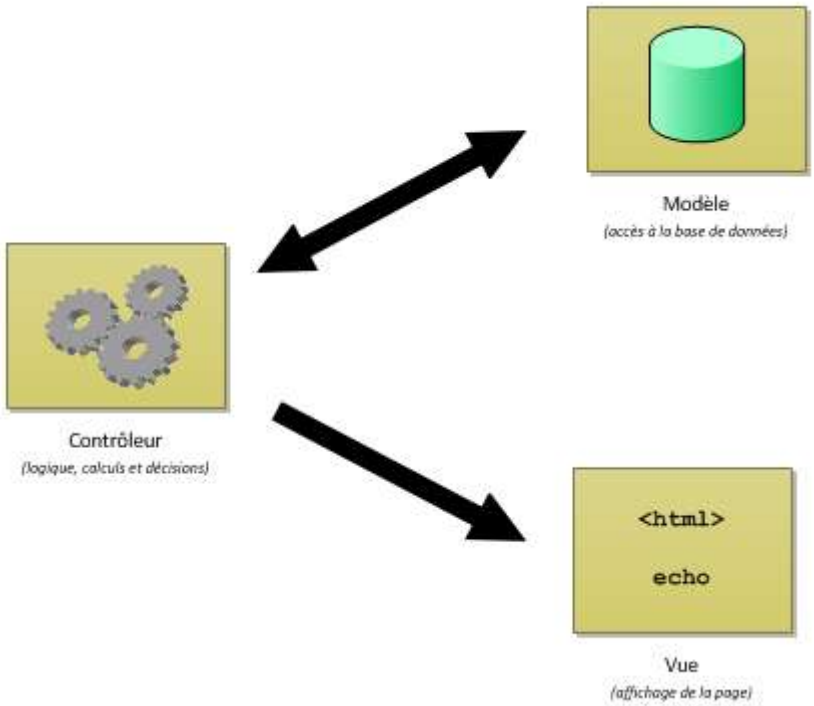

Figure 1: Schéma du fonctionnement de MVC

OBSERVER

Il est recommandé dès qu'il est nécessaire de gérer des évènements, quand une classe déclenche l'exécution d'une ou plusieurs autres.
Dans ce patron, le sujet observable se voit attribuer une collection d'observateurs qu'il notifie lors de changements d'états. Chaque observateur concret est chargé de faire les mises à jour adéquates en fonction des changements notifiés.

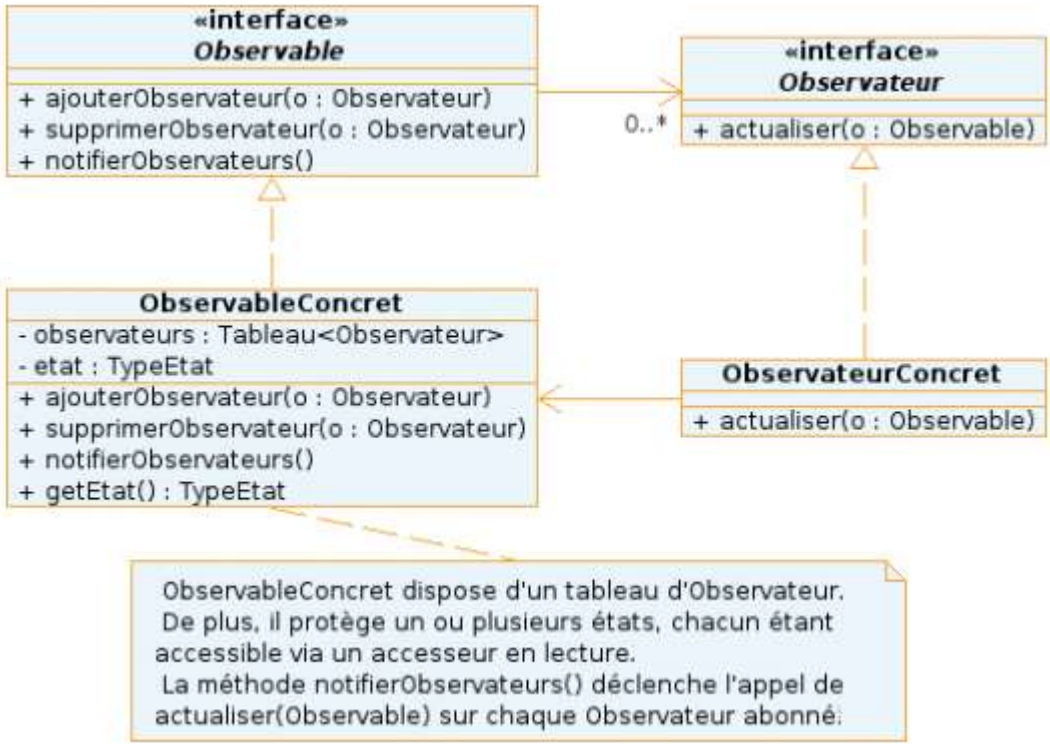

Figure 2: Schéma du fonctionnement d'OBSERVER

Dans notre cas, il va être utilisé pour la gestion du stock, le fait que les plongeurs peuvent aider les commis de cuisine dans leurs tâches…

SINGLETON

On implémente le singleton en écrivant une classe contenant une méthode qui crée une instance uniquement s'il n'en existe pas encore. Sinon elle renvoie une référence vers l'objet qui existe déjà.

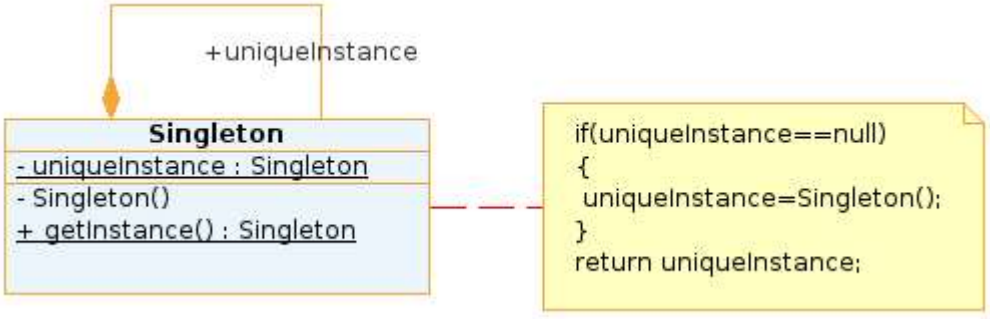

Figure 3: Schéma de fonctionnement de SINGLETON

Dans notre cas, il sera utilisé pour le DP factory.

FACTORY

Elle permet d'instancier des objets dont le type est dérivé d'un type abstrait. La classe exacte de l'objet n'est donc pas connue par l'appelant.
Plusieurs fabriques peuvent être regroupées en une fabrique abstraite permettant d'instancier des objets dérivant de plusieurs types abstraits différents.

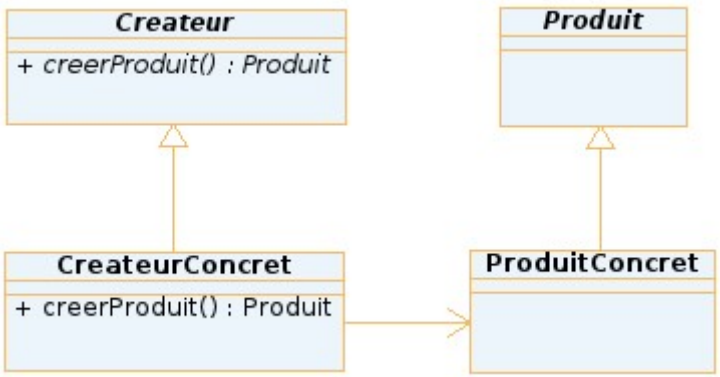

Figure 4: Schéma de fonctionnement de FACTORY

Dans notre cas, il va être utilisé pour gérer les clients, leur création.

DECORATOR

Decorator nous permet d'ajouter des fonctionnalités nouvelles à une classe de façon dynamique sans impacter les classes qui l'utilisent ou en héritent.

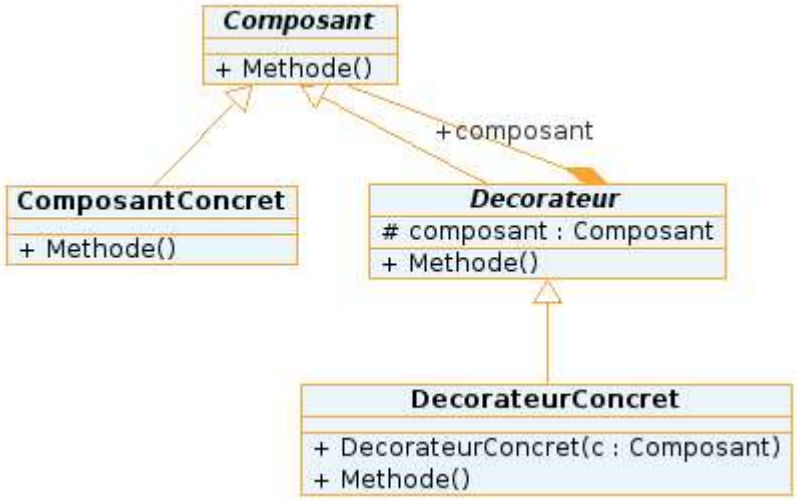

Figure 5:Schéma de fonctionnement de DECORATOR

Dans notre cas, il sera utilisé pour limiter l'héritage dans notre application, mais aussi comme le cas de l'Observer pour donner au plongeur le droit d'aider le commis de cuisine.

STRATEGY

L'idée principale est de pouvoir mettre en place une certaine stratégie (par exemple, une méthode qui réalise une certaine action) et de pouvoir changer dynamiquement de stratégie au runtime. On créer donc une interface de base, appelée ici Strategy et on y ajoute une méthode qui sera la méthode qui applique notre stratégie.

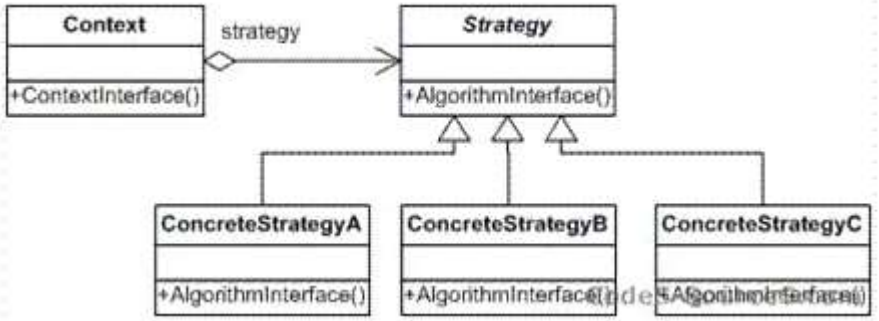

Figure 6: Schéma de fonctionnement de STRATEGY

Dans notre cas, il peut être utilise pour la gestion des commandes, des recettes, pour faire différents choix dans l'application.