PROJET PROGRAMMATION SYSTEME & OBJET

DESIGN PATTERNS & MVC

NOUMEN Darryl
SIKATI SAMUEL
ALLARAMADJI GHISLAIN
TANKWA JORDAN

Année Académique 2022-2023
PROMOTION X2025

24 NOVEMBRE 2022

Groupe 7

Sommaire

I. DETAILS ET EXPLICATION DES DESIGN PATTERNS

A. DESIGN PATTERN OBSERVER

L'Observer pattern est utilisé quand il y a une relation un-à-plusieurs entre les objets de sorte à ce que, si un objet est modifié, ses objets dépendants seront notifiés automatiquement.

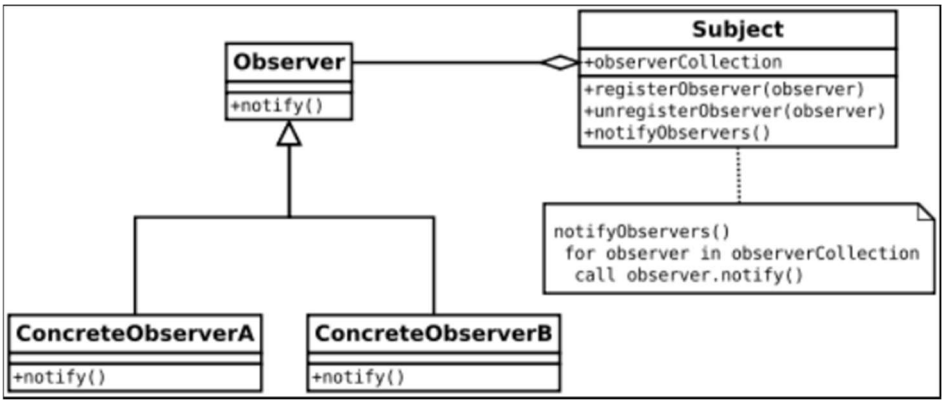

Figure 1 : Diagramme de Classe du Observer

L'Observer Pattern utilise trois classes d'objets qui sont ; le sujet, le client et l'Observateur. Le sujet est un objet qui comporte les méthodes pour mettre ou retirer un observateur sur un objet Client.

Nous comptons utiliser ce pattern pour notifier la vue de notre architecture MVC en cas d'épuisement d'un stock, d'insuffisance d'un ingrédient, de surcharge de travail sur un membre du personnel.

B. DESIGN PATTERN FACTORY

Ce design pattern vient sous les patterns de création car il offre la meilleure manière de créer un objet. Dans ce pattern, nous créons l'objet sans montrer au client la logique de création de ce dernier et fait référence aux objets récemment crées avec une interface commune. Ce pattern permet grandement de gérer des collections d'objets différents mais qui ont plusieurs caractéristiques similaires.

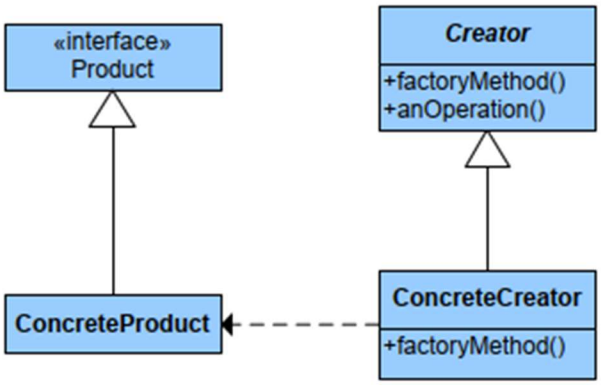

Figure 2 : Diagramme de Classe du Factory

Dans le cadre de notre projet, nous comptons l'utiliser pour faciliter la création des objets qui se répètent tels que les chaises ou les clients et ainsi ne pas avoir à les instancier pour chaque utilisation.

C. DESIGN PATTERN DECORATOR

Ce design pattern permet d'ajouter de nouvelles fonctionnalités à un objet existant sans avoir à modifier sa structure ou sa déclaration. Ce pattern crée une classe décorateur qui encapsule la classe originale et offre des fonctionnalités en plus tout en gardant les méthodes de la classe intactes.

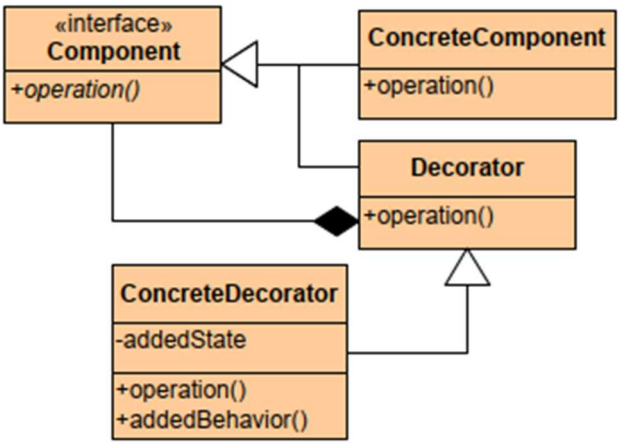

Figure 3 : Diagramme de Classe du Decorator

Dans notre projet, nous pensons l'utiliser pour la mise en forme des différents matériels, tels que les plats et autres, qui seront représentés par un objet auquel on va ajouter des fonctionnalités de mise en forme selon ce que l'on voudrait avoir.

D. DESIGN PATTERN STRATEGY

Dans ce pattern, le comportement de la classe peut être modifié lors de l'exécution. Ici, nous créons des objets qui représentent différentes stratégies et un objet contextuel dont le comportement varie selon ses objets de stratégies. L'objet de stratégie change l'algorithme d'exécution de l'objet contextuelle.

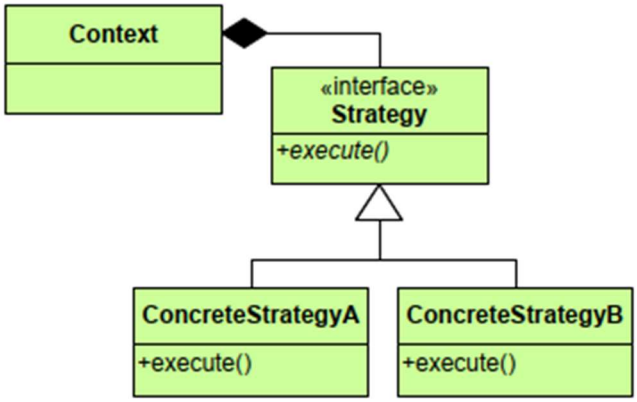

Figure 4 : Diagramme de Classe du Strategy

Dans notre programme, nous pensons utiliser ce design pattern pour l'implémentation des différents rôles des membres du restaurant, tel que le chef de rang, qui lorsqu'il n'y a pas de clients, peut donner un coup de main à un autre rang ou peut atteindre dans son carré ou à l'accueil du restaurant ou encore pour le comportement variant des clients ; ceux pressés et non pressés ; le choix de menu des différents clients ; ceux qui commande en un temps ou en deux temps.

E. DESIGN PATTERN SINGLETON

Ce pattern est l'un des plus simple car il consiste en la création d'une classe unique responsable de la création d'un objet tout en se rassurant qu'un seul objet a été créé. Cette classe offre un moyen d'accéder à cet unique objet qui peut être accédé directement par une autre classe sans avoir besoin d'instancier à nouveau l'objet.

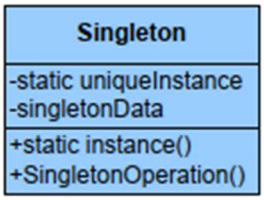

Figure 5 : Diagramme de Classe du Singleton

Dans notre programme, nous comptons utiliser le Singleton pattern pour faire la connexion à la base de données avec le driver SQL. L'instanciation de la base de données est contrôlée par ce même élément ce qui permet de s'assurer qu'il y ait bien un seul point d'accès à la BD et ainsi, nous garantissons une plus grande sécurité de nos données.

F. DESIGN PATTERN BUILDER

Ce pattern construit un objet complexe en utilisant de simples objets avec une approche par étape. Il permet aussi de produire différents types et représentations d'un objet à base du même code de construction.

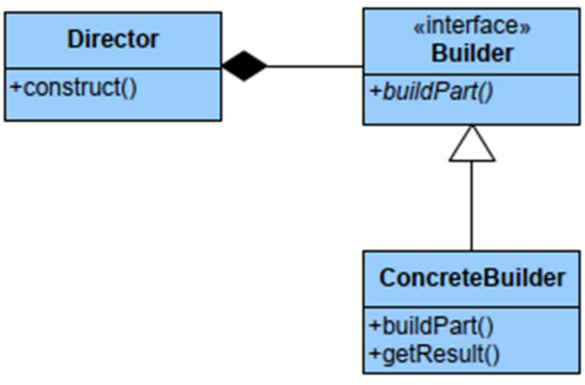

Figure 6 : Diagramme de Classe du Builder

Dans notre programme, nous pensons pouvoir utiliser ce pattern afin de créer les différents rôles ou membres qui travaille dans le restaurant, tel que ; le chef de rang, maître d'hôtel etc…

II. DETAILS ET EXPLICATIONS SUR L'ARCHITECTURE MVC

L'Architecture MVC est une façon d'organiser une interface graphique d'un programme en la distinguant en trois entités qui sont ; le modèle, la vue et le contrôleur ayant chacun un rôle précis.

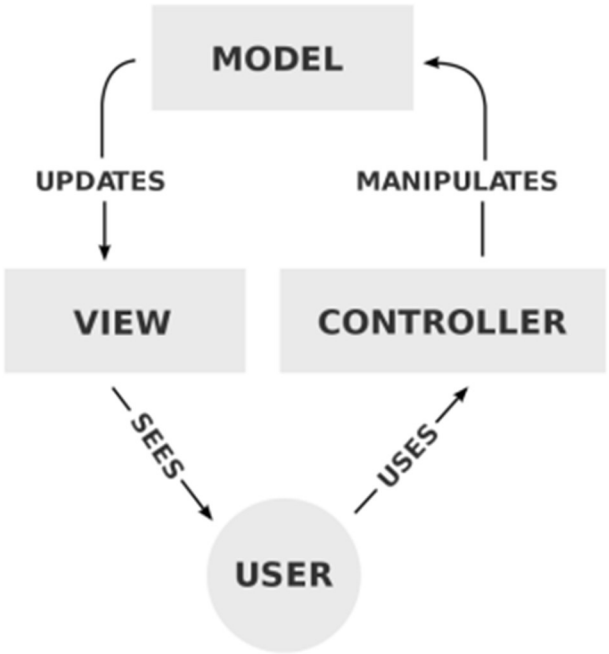

Figure 7 : Architecture MVC avec un utilisateur

A. MODELE

C'est la partie qui gère la logique métier de l'application et comprend notamment la gestion des données qui sont stockées, garantit leur intégrité, mais aussi tout le code qui interagit avec ces données. Son objectif est de fournir une interface d'action la plus simple possible au

contrôleur. Le modèle offre des méthodes pour mettre à jour les données (insertion, suppression, changement de valeur) dans une base de données.

B. VUE

C'est la partie qui sert d'interface avec l'utilisateur. Sa tâche première est d'afficher les données qu'elle a récupérées auprès du modèle. Aussi, elle a pour rôle de recevoir toutes les actions de l'utilisateur (clic sur un bouton, entrée à partir du clavier etc…) qui sont ensuite transmissent au contrôleur.

C. CONTROLEUR

C'est la partie qui gère les échanges avec l'utilisateur et permet de synchroniser le modèle et la vue. Il reçoit tous les événements de l'utilisateur et enclenche des actions en conséquence. Si une action nécessite un changement de données, le contrôleur demande la modification de celles-ci au modèle, qui va ensuite notifier la vue que les données ont changées pour une mise à jour. Dans le cas où les événements ne concernent pas les données, le contrôleur demande à la vue de se modifier directement.