Group 8 :

SPECIFICATIONS DES DESIGN PATTERN

Nous avons opté pour les différents patterns suivant :

1. **Pattern Singleton :**

En génie logiciel, le singleton est un patron de conception (design pattern) dont l'objectif est de restreindre l'instanciation d'une classe à un seul objet (ou bien à quelques objets seulement). Il est utilisé lorsqu'on a besoin exactement d'un objet pour coordonner des opérations dans un système.

Le pattern Singleton peut être utilisé pour garantir qu'il n'y ait qu'une seule instance de certaines classes clés dans l'application, comme le chef de cuisine. Cela permet de centraliser la gestion de certaines ressources critiques et d'éviter les conflits potentiels. Par exemple, une seule instance du chef de cuisine peut être créée et partagée par tous les autres acteurs de la cuisine pour garantir une coordination efficace.

2. **Pattern Observer :**

Le Design Pattern observer nous permettra de faire interagir le package Model avec celui de la Vue. On pourra à chaque changement d'état d'un attribut dans le modèle, notifier la vue de ce changement et ainsi la vue pourra afficher les bonnes données aux bons moments.

Ce pattern peut être utilisé pour gérer les interactions entre les différents acteurs du restaurant, tels que le maître d'hôtel, le chef de rang, les serveurs, etc. Lorsqu'un événement important se produit dans la salle de restauration ou dans la cuisine, cet événement peut être notifié aux observateurs pertinents, qui peuvent ensuite prendre des mesures appropriées. Par exemple, lorsque le maître d'hôtel attribue une table à un client, il peut notifier le chef de rang concerné pour qu'il puisse organiser le placement sur la table.

3. **Pattern Factory :**

Le pattern Factory peut être utilisé pour la création d'objets du restaurant, tels que les tables, les plats, les boissons, etc. Une fabrique peut être mise en place pour créer des instances appropriées de ces objets en fonction des besoins et des paramètres spécifiques. Cela permet de simplifier la création d'objets complexes et de centraliser la logique de création.

4. **Design Pattern Bridge :**

Le Pont est un patron de conception structurel qui permet de séparer une grosse classe ou un ensemble de classes connexes en deux hiérarchies — abstraction et implémentation — qui peuvent évoluer indépendamment l'une de l'autre.

Le pattern Bridge peut intervenir au niveau de la communication et de la coordination entre les différents systèmes du restaurant, tels que le système de réservation, le système de gestion des commandes et le système de gestion des stocks. Il permet de séparer l'abstraction (les fonctionnalités offertes par les systèmes) de l'implémentation (les détails spécifiques de chaque systèmeJe m'excuse, je voudrais corriger ma réponse précédente. Dans le contexte du projet de programmation système pour le restaurant, voici comment les design patterns Decorator, Prototype et Bridge pourraient intervenir :

5. Le design pattern MVC (Modèle-Vue-Contrôleur) peut être utilisé dans le projet de développement de l'application de gestion et supervision du restaurant pour séparer la logique métier, la présentation et la gestion des interactions.

A. Modèle (Model) :

Le modèle représente la logique métier et les données du système. Dans le contexte du restaurant, le modèle pourrait inclure des classes telles que "Table" pour gérer les informations sur les tables du restaurant, "Commande" pour représenter les commandes passées par les clients, "Menu" pour gérer les options de menu, etc. Le modèle encapsule les données et les fonctionnalités liées à ces entités, et peut inclure des méthodes pour effectuer des opérations telles que la réservation de tables, la gestion des commandes, etc.

B. Vue (View) :

La vue est responsable de la présentation des données au client. Dans le contexte du restaurant, les vues pourraient inclure l'interface utilisateur graphique (GUI) affichant les menus, les tables disponibles, les commandes en cours, etc. Les vues se mettent à jour en fonction des changements survenus dans le modèle et permettent aux utilisateurs d'interagir avec le système, par exemple en passant des commandes ou en réservant une table.

C. Contrôleur (Controller) :

Le contrôleur fait le lien entre le modèle et la vue, gérant les interactions et les actions de l'utilisateur. Dans le contexte du restaurant, le contrôleur pourrait recevoir les requêtes de l'utilisateur pour la réservation de tables, la passation de commandes, etc. Il interagit avec le modèle

pour effectuer les opérations correspondantes et met à jour la vue en conséquence. Par exemple, lorsque le client sélectionne un plat dans le menu, le contrôleur récupère les détails du plat à partir du modèle et les affiche dans la vue.

Le pattern MVC permet de séparer clairement les responsabilités et de favoriser la maintenabilité et l'évolutivité du système. Il facilite également la réutilisation du code, car les vues et les contrôleurs peuvent être réutilisés avec différents modèles. Dans le contexte du projet de programmation système pour le restaurant, le modèle MVC faciliterait la gestion des interactions utilisateur, la présentation des données et la manipulation des opérations métier, tout en maintenant une structure claire et modulaire.

6. Strategy design pattern

Ce design pattern décrit dans le document, le design pattern Strategy peut être appliqué pour gérer différentes stratégies de gestion des tâches et des opérations dans le système.

Prenons l'exemple de la gestion des tâches dans la cuisine. Dans la cuisine, il y a plusieurs postes tels que le chef de cuisine, les chefs de partie et les commis. Chaque poste a une fonction spécifique et exécute des tâches particulières. En utilisant le design pattern Strategy, nous pouvons créer une interface ou une classe abstraite appelée "TaskStrategy" qui définit les méthodes communes pour exécuter une tâche.

Ensuite, nous pouvons implémenter des classes concrètes telles que "ChefDeCuisineStrategy", "ChefDePartieStrategy" et "CommisStrategy", qui héritent de la classe abstraite "TaskStrategy" et implémentent les méthodes spécifiques à chaque poste. Chaque classe concrète peut encapsuler la logique spécifique à son poste, par exemple, la préparation des plats, la gestion des stocks, etc.

Au moment de l'exécution, le système peut utiliser une instance de la classe de stratégie appropriée en fonction du poste ou de la tâche à exécuter. Cela permet une flexibilité et une extensibilité du système, car de nouvelles classes de stratégie peuvent être ajoutées pour gérer de nouveaux postes ou de nouvelles tâches sans affecter le reste du code.

En utilisant le design pattern Strategy, nous pouvons donc encapsuler les différentes stratégies de gestion des tâches dans des classes séparées, favorisant ainsi la modularité, la flexibilité et la réutilisabilité du code.