UNIVERSITÉ DE SHERBROOKE

Faculté de génie

Département de génie électrique et génie informatique

RAPPORT APP1

Modèles de conception
APP 1

Présenté à

M. Domingo Palao Munoz

Présenté par

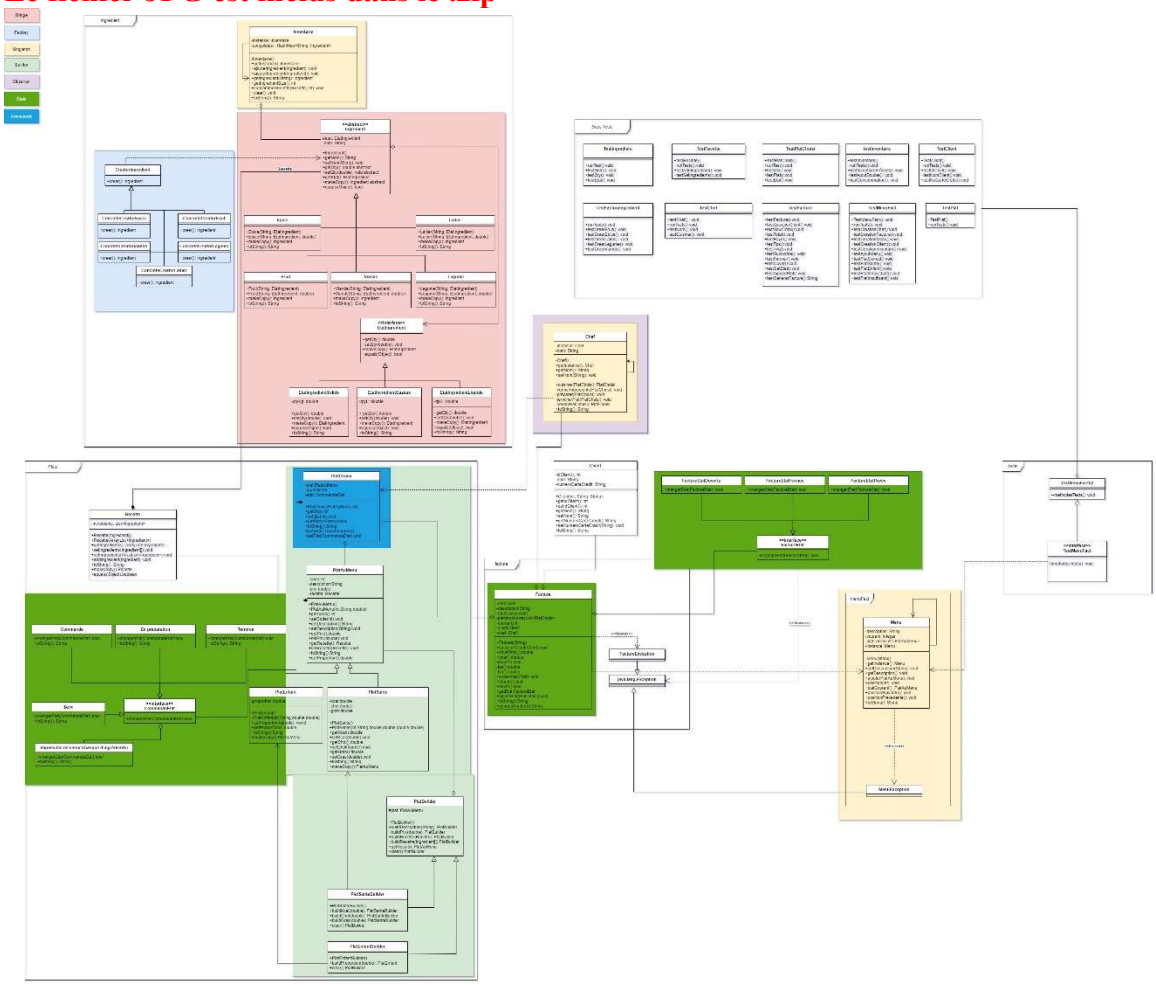
Anthony St-Laurent Cyr — STLA0801

Pascal-Emmanuel Lachance — LACP3102

Sherbrooke — 13 septembre 2021

Question 1 : Le nouveau diagramme de classes avec les Modèles de Conception à implémenter. Il serait très intéressant d'utilisés une couleur par pattern appliqué dans les classes.

Le fichier JPG est inclus dans le .zip



Question 2 : Un tableau pour expliquer l'utilisation de chaque Modèle de Conception et la justification de son utilisation.

Design Pattern	Explication de l'utilisation et justification de son utilisation
Singleton	Le Singleton est utilisé dans le but de s'assurer qu'il n'y a qu'une seule instance d'une même classe en même temps. Dans notre cas, on l'utilise sur le chef pour s'assurer qu'il n'y est qu'un chef qui produira ses fonctions. Le pattern Singleton est aussi utilisé dans le menu et l'inventaire dans le même but.
Observer	L'Observer est utilisé pour notifier un objet d'un évènement, évitant ainsi le polling, et permettant de combiner des notifications et d'avertir plusieurs objets uniquement lorsque nécessaire. Il était utilisé dans la problématique en conjonction avec une Commande pour notifier le chef qu'un plat avait été ajoutée à la facture et qu'il devait commencer à cuisiner.
State	Le State est utilisé pour assurer les transitions légales d'état dans une suite d'état. Dans notre situation, on utilise des states pour s'assurer que la progression des états par exemple dans la préparation d'un plat se fait dans le bon ordre. Le pattern State est aussi utilisé pour la facture.
Factory	La Factory permet de créer des éléments de divers types par une classe mère. Ensuite les sous-classes définissent leur propre type spécifique. L'utilisation de ce pattern est pertinente dans notre cas, pour universaliser la création de différents types d'ingrédient par une seule interface d'ingrédient qui rappelle les sous-classes de tous les types.
Builder	Le Builder permet de créer des éléments en plusieurs étapes consécutives simples. Dans notre cas, on utilise un petit Builder pour fabriquer les ingrédients, qui sont des objets complexes, qui profitent à être construits en plusieurs petites étapes.
Commande	La commande permet de traiter une action à faire pour la changer en objet distinct. Ce qui permet d'apporter l'information d'une classe à une autre au travers d'un objet. Dans notre situation le chef peut accéder à toute l'information de la préparation du plat par son Observer en prenant l'objet de la commande. Pour ensuite pouvoir lancer la préparation du plat.
Bridge	Le Bridge nous permet de séparer des informations de classes reliées en un attribut ; les mêmes informations sont présentes, mais utilisent de la composition plutôt que de l'héritage, et permettent d'éviter de la répétition et favorisent l'ajout d'autres classes dans le futur. Il était utilisé dans notre cas pour les états physiques (solide ou liquide) des différents ingrédients.

Question 3 : Quel est l'avantage d'avoir utilisé des Modèles de Conception pour développer l'application ? Est-ce que vous recommandez cette manière de créer une application ?

Les modèles de conception offrent des solutions entendues et optimisées pour résoudre des problèmes communément rencontrés dans la conception de logiciel. Ces solutions sont relativement standardisées, ce qui permet d'autodocumenter le code, permettant à de nouveaux programmeurs de rejoindre le projet et de comprendre les interactions entre les différentes classes au travers des modèles de conceptions. Les problèmes communément rencontrés lors de l'application des modèles à un code sont également bien documentés, ce qui permet de facilement les éviter lors de leur implémentation. Les modèles de conception permettent d'accélérer et de faciliter la conception (en offrant des solutions standards déjà pensées dans un langage entendu), l'implémentation (en étant bien documentés avec de nombreux exemples, et bien connus par les programmeurs) et la maintenance (en autodocumentant le code et en étant reconnus par les programmeurs).

Il est important de réfléchir aux avantages et aux désavantages de l'emploi des modèles de conception, et un modèle de conception forcé là où il ne devrait pas être est tout aussi nuisible qu'un code fait sans appliquer aucun modèle de conception. En revanche, utilisés judicieusement, les modèles de conception sont un outil puissant pour la conception, le développement et la maintenance du code, que nous recommandons fortement pour le développement d'une application.

Les modèles de conceptions représentent des solutions à des problèmes rencontrés communément, qui doivent être résolus d'une façon ou d'une autre lors de la conception et du développement d'un logiciel, alors autant choisir les solutions pensées et standardisées pour remplir ce rôle plutôt que de réinventer la roue.

Question 4 : Quel est l'avantage d'avoir utilisé les Test Unitaires dans le développement de l'application ? Est-ce que les Tests Unitaires ont changé votre manière de développer l'application ?

Les Tests Unitaires permettent de tester individuellement chaque fonctionnalité du logiciel indépendamment, ce qui permet de repérer les problèmes très rapidement en ce qui concerne l'implémentation initiale du programme. Idéalement, ces derniers seraient développés en même temps que l'application pour pouvoir tester le comportement demandé et trouver les erreurs et comportements erronés. En théorie, un code qui passe tous les tests unitaires devrait bien fonctionner lors des tests d'intégrations complètes et du déploiement final du logiciel (la pratique n'est pas aussi simple, mais passer tous les tests est un excellent signe et indique que 99 % de l'application peut fonctionner).

Les tests unitaires changent également comment on pense au programme, et aident à vouloir créer un programme plus divisible pour rendre les tests plus faciles à implémenter, ce qui est une bonne pratique à prendre dans tous les cas. Ils nous aidaient aussi à penser aux cas extrêmes (telles que des valeurs négatives ou nulles) à prendre en compte lors des tests (et donc lors de l'implémentation des fonctionnalités pour couvrir les tests). Les Tests Unitaires ont donc changé notre manière de développer l'application, en apportant une vision différente des fonctionnalités. Celle-ci amenant un recentrage sur la fonctionnalité première des diverses méthodes indépendamment de l'implémentation.