|  |
| --- |
| Projet de design pattern |
| Conception d’un bar à salade |

Suivi du document

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | 25/06/2019 | V1.0 | |  |  | | |  |  | | --- | --- | | Création | V. Guénin | | Validation | XXXXXXX | |
| |  |  | | --- | --- | | 30/06/2019 | V1.1 | | |  |  | | --- | --- | | Modification | C. Célia | | Validation | XXXXXXX | |



Groupe 1 : Benard Alexis, Casagrande Célia, Guénin Vincent

ESGI – Design Pattern – Dorra Bartaguiz - 2018/2019

Contenu

[Présentation de l’application 1](#_Toc12808936)

[Explication des différents modules 1](#_Toc12808937)

[Interface utilisateur 1](#_Toc12808938)

[Création de la commande 2](#_Toc12808939)

[Construction de la salade 3](#_Toc12808940)

[Préparation de la commande 4](#_Toc12808941)

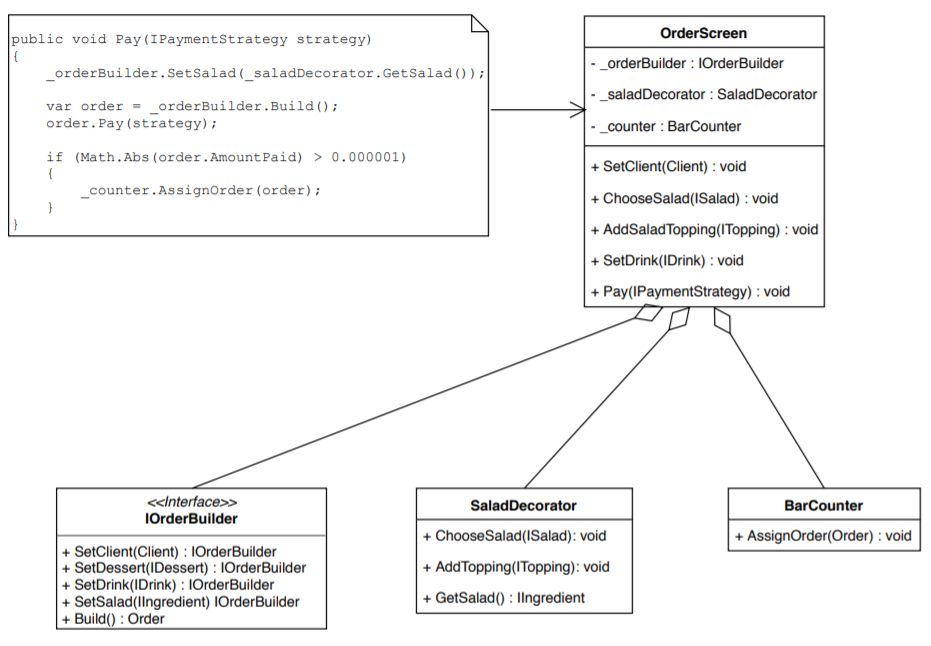
[Gestion du paiement 5](#_Toc12808942)

# Présentation de l’application

Notre application est destinée à gérer un bar à salade. Elle est composée de cinq modules permettant de symboliser un parcours client, de l’identification de ce dernier à la remise de la commande.

# Explication des différents modules

## Interface utilisateur



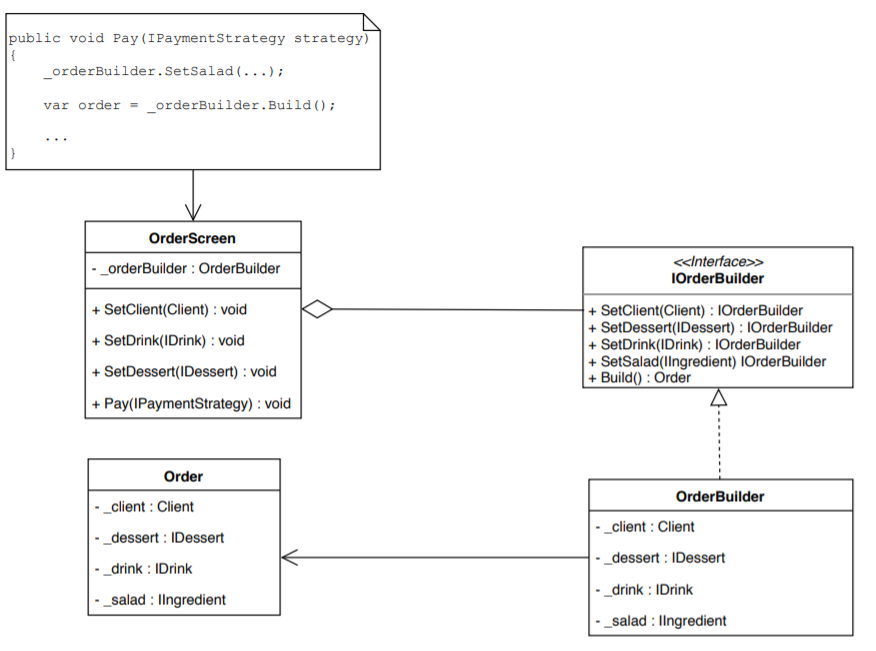
L’interface client de notre application est représentée par la classe OrderScreen. Elle symbolise l’écran de commande de l’application, à l’image de l’écran de commande utilisateur que l’on peut trouver dans les restaurants McDonald’s.

Elle symbolise un point d’entrée unique pour l’utilisateur. Nous avons donc décidé d’implémenter le design pattern facade pour représenter ce module.

Techniquement, les méthodes de cette classe délèguent le travail aux services correspondants. L’utilisateur va renseigner son nom afin que sa commande puisse lui être remise (à l’image de Starbucks), définir les éléments de sa commande puis payer, ceci au sein de cette même interface.

La méthode Pay permet d’ordonner la commande : elle finalise sa construction pour permettre le paiement et l’envoi en cuisine.

## Création de la commande



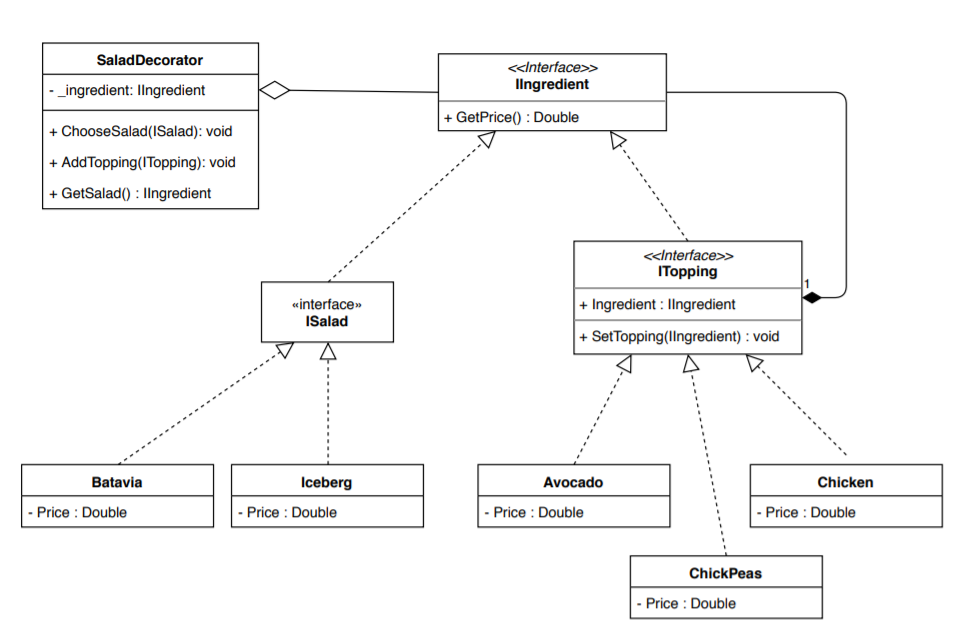
Une commande dans notre bar à salade est un objet plutôt complexe qui peut être représenté de multiples manières différentes : par exemple, le client a le choix de prendre ou non un dessert ou une boisson. Il pourrait également s’il le souhaite ne prendre qu’un dessert.

Le design pattern Builder nous a donc paru être le plus adéquat pour conceptualiser ce module, ce dernier permettant de travailler plus aisément avec les champs optionnels et les objets complexes.

Nous considérons également que l’ordre de création des éléments de la commande n’a aucune importance, ainsi, le design pattern builder nous permet de créer notre commande au fur et à mesure des actions de l’utilisateur.

Nous vérifions dans la méthode Build() si l’utilisateur a bien renseigné son identité et a bien commandé au moins un élément. Dans le cas contraire nous retournons des exceptions personnalisées.

## Construction de la salade



La salade est un objet complexe à part entière et nous avons considéré qu’il était préférable de le détacher de notre « builder » de commande. Le cas contraire était pour nous une entorse au principe de responsabilité unique d’une classe.

Alors, en quoi une salade est un objet complexe ?

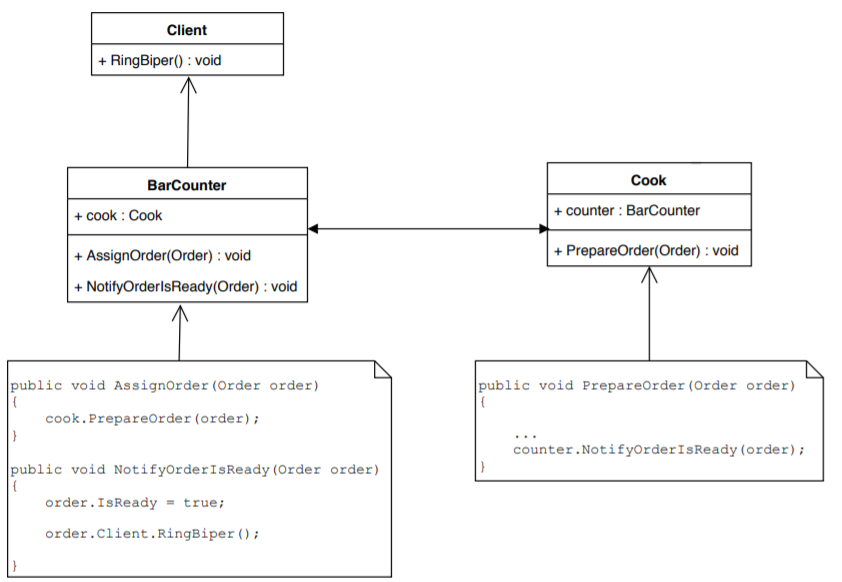
Notre application considère que les clients peuvent commander la salade qu’ils veulent parmi les différents choix et y mettre tous les ingrédients dont ils ont envie. Nous souhaitions qu’un client puisse choisir entre différents types de salade et ajouter autant de toppings qu’il le désire, en quantité illimitée.

Le design pattern decorator et son principe « d’enveloppement » correspondait donc parfaitement à notre besoin. Il nous permet de créer des salades avec tous les toppings disponibles, à quantité variable. Et si le client souhaite seulement de la laitue Iceberg, cela lui est permit.

Lors du paiement, le prix de la salade est calculé à partir du prix de chaque topping présent dans cette dernière ainsi que du prix de la laitue choisie. A noter que l’on nomme topping tout ingrédient de la salade (pois-chiche, avocat…).

Nous envoyons une exception personnalisée si l’utilisateur ajoute un topping avant de choisir sa salade.

## Préparation de la commande



Une fois que la commande est envoyée au cuisiner et que ce dernier finit de la cuisiner, le client doit être informé. Nous avons cependant considéré que ce n’était pas au cuisinier d’informer le client de la disponibilité de sa commande.

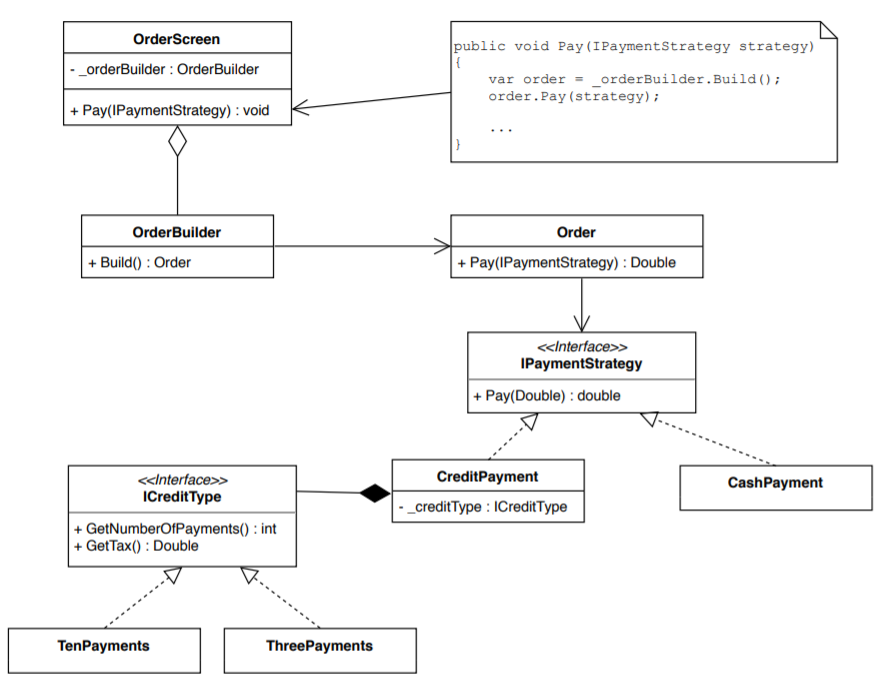
Pour conceptualiser ce problème, nous avons alors mis en place deux observers :

Le premier est déclenché par le cuisiner qui indique au comptoir du bar que la commande est finie d’être préparée.

Le bar du comptoir recevant cette dernière alerte, il peut alerter le client en lui faisant sonner son bipper.

Le client est alors servi !

## Gestion du paiement



Lorsque la commande du client est complétée, il doit payer afin qu’elle soit transmise aux cuisiniers.

Nous avons trouvé intéressant de permettre plusieurs moyens de paiements. Nous avons donc opté pour le design pattern strategy pour conceptualiser ce problème car il permet de faire faire varier le traitement facilement, indépendamment de l’objet qui l’utilise.

Egalement, il est de cette manière particulièrement aisé d’ajouter de nouveaux moyens de paiement en fonction des possibilités de l’entreprise.

Deux moyens de paiement sont actuellement disponibles :

* En liquide : le montant de la commande est réglé en une fois.
* Par crédit : le montant de la commande est réglé en plusieurs fois. Le nombre de paiements (IcreditType) est passé en argument du constructeur et fait varier le prix final de la commande : le paiement par crédit est taxé, la taxe étant propre à chaque type de crédit.