Problème : Pizzeria

Thomas Osorio TDJ-A3

I/Explication du diagramme UML :

Le cahier des charges fait référence à différentes caractéristiques d'une pizzeria qu'il est nécessaire de transformer en diagramme UML. En lisant l'énoncé, j'ai pu dégager quatre classes principales **: Effectif**, **Client**, **Commande** et **Produits** dont certaines peuvent décomposer en sous-classe. Pour simuler les différentes actions réalisées dans mon Interface, j'utilise la classe **MaPizzeria**.

Tout d'abord l'**Effectif**, cet objet doit posséder des coordonnées : un nom, un prénom, une adresse et un numéro comme tout objet représentant un individu. Ce pourquoi on peut y implémenter l'interface **ICoordonnées**. Celle-ci implémente les quatre propriétés des variables précédentes. D'après la lecture, On doit aussi informer si ce salarié est en congé ou non. Cependant, il est plus judiciable de ne pas pouvoir instancier ce type d'objet car être de type effectif n'est pas une fonction. dans la pizzeria. Un salarié de type **Effectif** peut être un **Commis** ou un **Livreur**. Un **Commis** possède comme caractéristique une date d'embauche qui permettra de définir sa progression, mais il est aussi celui qui **gère** les commandes au sein de la pizzeria, ce pourquoi il est aussi défini par un nombre de commande gérées dont il est et a été responsable. Ce critère nous permettra d'analyser son rendement par la même occasion. L'autre sous-classe d'**Effectif** est **Livreur.** Comme le dit le cahier des charges nous l'indique nous voulons savoir si une instance de type **Livreur** est en route. En plus de cela il a pour fonction de livrée bien entendu et ramener le paiement de la commande livrée, voila pourquoi il a avec lui de l'argent et comme sa fonction l'indique il a avec lui une commande traite s'il est en route. Nous verrons par la suite que le Livreur est affilié à une ou plusieurs commandes dont il est responsable pour la livraison. On cherche aussi à calculer son rendement en calculant le nombre de livraisons qu'il peut effectuer. Après avoir effectué sa livraison avec succès, il ramène la **Facture**. Afin de réaliser les différentes actions de tri ou traité les évènements pour l'affichage WPF, on implémente les interfaces **IComparable** et **INotifyPropertyChanged** sur la classe mère **Effectif.** On part du principe que quand une pizzeria ouvre elle doit obligatoirement contenir un effectif pour son bon fonctionnement.

En plus de cela, même si une pizzeria peut ouvrir sans client à l'origine, elle possède bien entendu des objets de type **Client** au bout d'un certains temps**.** Les clients, comme tout individu, doivent pouvoir être identifiés par des coordonnées à l'aide l'interface **ICoordonnées.** D'après l'énoncé, on est aussi intéressé pour lui donnée un numéro client, la date de sa première commande et le montant cumulé de ses achats pour voir sa fidélité. Pour effectuer les différents traitements, on implémente les interfaces **IComparable** et **INotifyPropertyChanged.** De plus le **Client** réalise une commande auprès de la Pizzeria, il commande des produits avec le **Commis.** De même le Client règle la commande facture auprès du livreur.

La pizzeria est composée de commandes grâce à une classe **Commande.** Une commande doit posséder un numéro de commande, une heure de commande, une date, le nom du client, du commis responsable par cette commande, du nom des livreurs, l'état de la commande et l'état du solde. Pour effectuer les différents tris nécessaires demandés par le cahier des charges, j'implémente l'interface **IComparable.** Elle est aussi composée d'une **Facture** qui liste les différents **Produits** commandés. Voila pourquoi on implémente la classe **Facture** qui en plus des produits est caractérisée par un solde final, la somme à régler par le **Client.** Les **Produi**ts commandés par le **Client** sont caractérisés par un prix, un nom et la quantité de ce produit et sa taille. Dans notre pizzeria, il nous est demandé de proposer des

Pizzas et des **Boissons.** Les deux classes se différenties par le type qu'elles peuvent être et la manière de calculer le prix final du produit. On trouve aussi une liste de produits dans la classe MaPizzeria qui correspond au stock d'aliments.

Pour la partie innovation du cahier des charge, j'ai implémenté les classes **Fournisseurs**, **Viande**, **Fromage** et **Légume**. Si le 3 dernières classes sont des sous-classes de la classe Produit et représente la matière première, la classe **Fournisseur** représente les éventuels fournisseurs de la pizzeria on retrouve certaines interfaces qui me permettront de réaliser certains tris sur cette classe que j'expliquerai dans les prochaines parties. Une pizzeria peut avoir plusieurs fournisseurs et doit en avoir au moins un pour ouvrir.

Pour gérer les différentes classes citées ci-dessus et pouvoir simuler correctement les relations entre elles, j'ai pris la liberté de créer une classe **MaPizzeria** qui sera composé de **Commis**, **Livreur** obligatoirement et d'un menu qui propose la liste des produits vendu pour pouvoir ouvrir. En plus de cela s'ajoute un fichier client et un historique des commandes réalisés.

II/Structure du code C# et Utilisation de WPF

1. Les différentes structures de données :

Afin de pouvoir gérer les différentes classes traiter précédemment, nous allons utiliser dans la classe MaPizzeria deux **List**, pour les commis et les livreurs. Comme nous devons identifier la commande par un numéro, nous allons identifier l'ensemble des commandes par une **SortedList<int,Commande>.** De même pour les clients que nous devons être capable d'identifier grâce à leurs numéros de téléphone grâce à la collection **SortedList<string,Client>.**

2 . La lecture de fichier :

Pour lire les différents fichiers csv, j'utilise la classe StreamReader. Elle me permet de lire chaque ligne une à une en séparant les données des cases dans une chaine de string à l'aide de la méthode **String.Split()**.

3 . Gestion des Commandes:

Le commis saisit le numéro du client comme demandé par le cahier des charges. Dans un premier temps le programme vérifie si ce client existe à l'aide de la méthode Keys.IndexOf(). S'il existe dans le fichier on envoie un message pour savoir si l'adresse est correcte sinon on commence une nouvelle commande en ouvrant une nouvelle fenêtre. Dans cette fenêtre on pourra créer une nouvelle facture, en ajoutant au panier tous les produits que l'on veut, que ce soient des boissons ou des pizzas. Pour ajouter un produit au panier on fait appel à une méthode d'ajout pour chaque sous-classe. Dans ces méthodes on trouve à l'aide d'une **méthode lambda** sur la méthode **.Find()** de la liste de produit du menu que l'on veut. Ensuite on regarde s'il est déjà dans la commande à l'aide de cette même méthode **.Find()** mais cette fois ci avec **une méthode anonyme**. Si le produit est déjà dans la commande alors on met à jour la quantité et le prix total dans la facture, sinon on ajoute un nouveau produit dans la facture. Pour calculer le prix d'un produit selon sa taille j'ai implémenté un **delegate** dans la classe **Produit** qui permet de calculer le prix d'un produit. Deux méthodes respectent cette définition, une dans la classe Pizza et l'autre dans la classe **Boisson.** Elles peuvent être appelé depuis une autre méthode qui passe un paramètre de type **CalculPrix().**

Une fois la commande terminer, on cherche à l'envoyer. L'algorithme vérifie si la commande est composée d'au moins une pizza et que le nom du commis responsable a bien été choisi. Pour définir le livreur disponible, on regarde parmi les livreurs ceux qui sont actuellement en route et on choisit

celui qui ne l'est pas. Si aucun n'est disponible, il faut malheureusement attendre avant de passer la commande. Pour finir on ajoute la nouvelle commande dans l'historique et on modifie sur l'écran de gestion principale de l'interface.

Dans cette interface on peut accéder de différente manière à la commande. On peut voir son état grâce à la **listView** et on peut accéder aux différentes commandes de celle-ci en les recherchant par son numéro, grâce à la méthode **SortedList.GetKey().** Pour modifier l'état de la commande il suffit de cliquer sur le bouton prête, lorsque le cuisinier termine la préparation et en livraison, en modifiant l'état de la commande sélectionnée et activant la caractéristique en route du livreur. Lorsque le livreur revient avec la commande on écrit le numéro de commande, le numéro du livreur pour ensuite avoir le paiement. Le livreur est retrouvé grâce à la **méthode lambda** de **List.Find().**

Nous pouvons aussi voir le temps passé depuis la commande passé grâce au bouton durée. Dans l'optique ou le fichier client est conséquent j'ai implémenté l'interface **IComparable** dans commande qui permet de modifier la méthode **CompareTo** selon le délai depuis commande. On peut appeler ce tri depuis le bouton trier par délais.

Afin de pouvoir afficher les factures à imprimer, en cliquant sur une commande, on peut afficher son fichier.txt qui représente sa facture, de même que pour la gestion des clients et livreurs qui sont accessible en txt afin d'avoir une vision plus simple de nos effectifs.

4 .Gestion des clients :

Lors de l'appel, au cas où le client n'existe pas encore dans le fichier client, on lui crée une fiche client que l'on enregistre.

Dans la fenêtre de gestion de clients il nous est demandé de pouvoir ajouter-supprimer-modifier le client, ce pourquoi je réalise différentes actions sur mes clients qui me permettent de mieux les traiter. Pour rechercher un client on écrit son numéro et par la **SortedList** on a accès à la valeur de ce numéro de téléphone. On peut modifier et supprimer un client en le sélectionnant au préalable. Pour la modification, on peut changer seulement une donnée du client en le sélectionnant et non toutes.

Pour les problèmes de tris, j'utilise l'implémentation de la méthode **CompareTo** pour trier les clients selon leurs dépenses. Pour les noms et les villes j'utilise la délégation du **List.Sort()** avec deux méthodes static permettant de trier selon le nom et la ville.

5. Gestion des Effectifs :

L'effectif est composé de livreur et commis. Afin d'exploiter au plus possible le WPF, j'ai créé dans ma fenêtre pour le personnel deux pages. Les deux comportent une ListView avec les différentes informations nécessaires à l'affichage. Tous les objets de type **Effectif** peuvent être trier de deux manières. Tout d'abord selon leurs noms grâce à l'implémentation de l'interface **IComparable** et de sa méthode **CompareTo.** Dans un second temps selon leurs villes. Notons que comme pour les clients, j'ai structuré l'adressage afin que chaque donnée soit lisible en utilisant split sur une virgule. On peut donc comparer les types selon leurs villes de domicile grâce à la fonction **Ville()** et la méthode **static myCompareVille()** qui respecte le delegate **Comparison()** que l'on appelle ensuite dans le **List. Sort()** de chaque type d'effectif.

Dans les pages commis et livreurs on peut ajouter, retirer, modifier et rechercher un commis ou un livreur. Les principales fonctions demandées pour les commis peuvent être appelé à l'aide de bouton. Quand l'utilisateur rentre le numéro du commis ou du livreur on vérifie s'il n'y a pas de

caractère littéraire grâce à la fonction **Int32.Parse().** On peut modifier et rechercher une de ces personnes juste à partir de l'une de ses caractéristiques sauf pour ajouter un nouveau commis ou livreur, où bien entendu il faut que l'on ait les coordonnées complètes. De plus nous pouvons aussi trier les commis et livreurs selon leurs rendements grâce l'utilisation des méthodes **myCompareNbCommandes//myCompareNbLivraison** qui sont appelés dans la méthode **List.Sort().** Pour finir on peut aussi préciser si le commis est en congé grâce au bouton congé.

6.Innovation et Optimisation :

Pour aller plus loin j'ai voulu réaliser quelques créations afin de rendre mon interface plus interactive. Dans les fonctionnalités déjà existantes j'ai permis à l'utilisateur, le commis, de pouvoir visualiser les pizzas préférées des clients en parcourant toutes les commandes, en comptant le nombre d'occurrences de chaque type de pizza et en utilisant le principe de récursivité pour ceux-ci. Dans un second temps j'ai aussi attribué une réduction automatique de la commande si le client commande plus de 3 pizzas.

Mon innovation la plus importante est celle de la classe **Fournisseur.** J'ai voulu mettre en relation ma pizzeria avec une certaines gestion des stocks. Dans cette classe fournisseur, je peux acheter de nouveaux produits afin d'augmenter la quantité du stock, je peux rechercher un fournisseur par son nom mais aussi par le type de produit qu'il vend. On peut aussi comparer les fournisseurs selon les achats effectués chez eux et avoir une idée des paiements que l'on a effectué. Il est aussi possible de trier le stock par la quantité de chaque produit afin de pouvoir voir lesquels il est nécessaire de commander. Il faut faire aussi attention à l'état des stocks avant de commander. Une amélioration pourrait être une commande automatique au cas où les stocks soient vide, mais aussi de pouvoir vider tel ou tel stock selon le type de produit.

III/Conclusion et Améliorations possible :

Pour finir je pense avoir respecter la plupart des contraintes du cahier des charges car mon programme réalise toutes les tâches correctement. J'ai essayé d'être le plus juste possible dans l'usage de WPF. J'ai cherché à ne pas effectuer de traitement sur les données dans les commandes WPF (boutons,textbox etc..) sauf dans les cas où les appels des méthodes de **List** permettent de le faire en une ligne. Toujours de manière à rendre mon projet le plus optimale possible, j'ai porté une attention particulière sur l'écriture utilisateur (un nom à la place du numéro etc…) pour que les fenêtres soient le plus lisibles possibles et le plus simple à manipuler. Afin de lier mes différentes fenêtres et les données qui les composent entre elles, j'ai mis en place des relations d'évènements entre mes différentes classes de fenêtre lorsque qu'une modification de la pizzeria apparaissait. Enfin j'ai essayé d'utiliser le plus d'éléments possibles de WPF afin de montrer un certain panel de fonctionnalité en utilisant des ListView, ComboBox, textBox, slider, fenêtres, pages et autres….

Néanmoins certaines améliorations pourraient être apporté à ma simulation. Je pense que j'aurai pu rendre plus performant mon innovation en pouvant choisir un menu ou une carte des produits proposés par chaque fournisseur, ou encore mieux gérer le stock. Un autre point à souligner qui peut s'avérer problématique est qu'un livreur ne peut prendre qu'une commande à la fois. Pour optimiser les livraisons, j'auraient pu lui permettre d'effectuer plusieurs livraisons à la fois.

Pour terminer ce projet m'a permis de découvrir de nouvelles chose en C# et WPF et d'améliorer mes capacités en programmation.