Problème POO Avancée

1. Introduction

Au cours du 1er semestre de cette 3ème année au sein de l’ESILV, nous avons pu découvrir deux matières liées à la programmation C#. Les Structures de Données et Algorithmes, et la Programmation Orientée Objet, c’est dans cette dernière que nous avons eu l’occasion de réaliser ce projet.

Nous avons donc dû réaliser un projet dont le but est de simuler une application permettant de prendre en charge les commandes au sein d’une pizzeria. Même si ce sujet peut paraitre assez simple à son intituler, il est en réalité très complexe. Nous avons dû acquérir de nouvelles compétences telles que la création d’une solution se présentant sous la forme d’une interface graphique (WPF) que nous n’avions encore jamais rencontré auparavant, et la conception d’un projet complexe en binôme.

Nous allons donc vous expliquer et vous présenter ce que nous avons réalisé lors de la production de cette solution.

1. Diagramme UML

Pour réaliser ce diagramme UML, nous avons commencé par bien identifier toutes les classes dont nous allions avoir besoin pour mener à bien le projet. Pour ce faire nous avons lu attentivement le sujet du problème et fait attention à bien comprendre comment nous allions devoir construire notre solution.

Nous avons donc choisi de commencer par réaliser les classes en partant « du bas de la pyramide », c’est-à-dire que nous avons commencé par créer les classes qui sont à la base du code, la classe Pizza, la classe Client, la classe ProduitAnnexe et la classe Personne qui sont des classes qui vont nous permettre de créer par la suite l’ensemble des classes qui vont mener à l’aboutissement de notre projet.

* Attributs de la classe Pizza : « string : nomPizza » pour pouvoir différencier les différents types de pizza, « double prixPizza » qui nous permettra par la suite de donner le total du prix de la commande au client et d’étudier des statiques sur les chiffres de vente de la pizzéria, « string taille » qui nous permet de donner un prix différent selon si la pizza est plus ou moins grande.
* Attributs de la classe Client : « string nom, string prenom, string adresse, string ville, string mail, int numeroTel » qui nous permettent de donner des caractéristiques propres à chaque client pour pouvoir les différencier.
* Attributs de la classe Employe : « string nom, string prenom, string adresse, DateTime recrutement, int identifiant » pour décrire chacun des employés. Cette classe sera une classe Abstract car nous ne créerons jamais d’instance de la classe personne directement.
* Attributs de la classe ProduitAnnexe : un prix de type double, un nom de type string

Nous avons donc logiquement enchaîné avec la classe commande qui sera composée de liste de classe précédente (Pizza/ProduitAnnexe) mais qui aura aussi comme attribut un numéro de commande pour que l’on puisse différencier les commandes les unes des autres même si elles sont composées des mêmes produits. Elle sera évidemment liée à un client pour que nous puissions retrouver à qui appartient chaque commande. Ainsi qu’un attribut Facture de type double pour que nous puissions dire au client combien il doit régler mais aussi pour que la pizzeria ne se perde pas dans sa comptabilité.

Nous pouvons enfin passer au classe Cuisinier, Commis et Livreur qui seront des classes filles de la classe Employe qui représente leur classe mère. Ces trois classes ont chacune leurs attributs propres en plus de ceux hérités par la classe mère.

* Commis : « string état » pour savoir si le commis est présent ou non, disponible ou non et « DateTime embauche » pour connaitre son ancienneté au sein de la pizzeria.
* Livreur : « Commande commandeencours » pour savoir quelle commande le livreur est-il en train de livrer. « string etat » pour savoir s’il est présent ou non. « double argentpaiement » pour connaitre la somme exacte avec laquelle le livreur est rentré à la pizzeria.
* Cuisinier : « string niveau » pour connaitre le nombre de pizza que le cuisinier peut préparer en même temps et « pizza[] nbrpizzapreparation» qui nous permet de savoir combien de pizza le cuisinier prépare actuellement.

Et vient enfin la classe Pizzeria, dans laquelle nous allons mettre 4 attributs qui sont : « List<Commande> listeCommande» qui sera la liste qui contiendra toutes les commandes passé via l’application. « List<Client> listeClient » qui nous permettra de stocker tous les clients qui ont réalisé une commande via l’application. « List<Employe> listeEmploye » où seront stockés de la même manière tous les employés de la pizzeria. « string nomPizzeria » que nous initialiserons à « Buddy’s Pizzeria » que nous avons choisi comme nom de Restaurant.

1. Fonctions

La fonction la plus complexe que nous avons utilisée consiste à lire un fichier externe à Visual Studio, puis à réécrire ce que nous avons récupérer dans le fichier, Je m’explique : pour pouvoir avoir une liste de client qui reste dans notre application même dans le cas ou nous la fermons, il a fallu passer par un fichier externe. Nous avons choisi de passer par un fichier.csv, qui nous a semblé être le plus judicieux pour pouvoir écrire et lire le fichier. Nous avons divisé la fonction en deux fonctions indépendantes l’une de l’autre. Une première qui nous permet de lire le fichier externe qui contient déjà une liste de client de client qui ne sont pas en train d’utiliser l’application à l’instant T, pour la mettre dans une liste de client List<Client> listeClient dans notre application, et une seconde qui nous permet d’écrire ce que nous avons récupéré la liste de client de l’application avec en plus les informations du nouveau client qui vient de réaliser sa commande. Nous les avons appelés EntreClient() et EnregistrementClient()

Nous avons raisonné exactement de la même manière pour la liste de Commande, soit les fonctions EntreCommande() et EnregistrementCOmmande().

Les fonctions que nous vous avons présenté précédemment se situe dans la classe pizzeria qui symbolise l’intégralité de notre restaurant.

Nous avons créé une fonction AfficheCommande() dans la classe Commande qui nous permet d’afficher l’intégralité de la commande du client qui est en train d’utiliser l’application.

1. Interface WPF

Nous avons créer 6 Window afin de satisfaire la navigation et la prise de commande de l’utilisateur.

Identification: L'utilisateur se retrouve sur cette window pour créér son profil client avec les informations qu’il aura rentré dans les textbox.

MainWindow: Une fois que l’utilisateur a validé son profil client avec un numéro de téléphone exploitable il a le choix pour aller sur la page Commander,Contact,Adresse.

WindowAdresse: L’utilisateur retrouve sur cette page l’adresse de Buddy’s Pizza ainsi une photo du restaurant.

WindowContact: Sur cette page l’utilisateur obtiendra le numéro de la pizzeria ainsi qu’un endroit écriture où il pourra envoyer un message à l’entreprise. Un message apparaitra pour confirmer la prise du message et le retour sur sa boite mail.

WindowCommander: C’est ici que le l’utilisateur pourra sélectionner les pizzas qu’il voudra ajouter à sa commande s’il a bien renseigné la taille avec la Comboxbox. Un message apparaitra pour confirmer la sélection de la pizza.

WindowPanier: Cette page window est uniquement accessible depuis la page WindowCommander. Elle permet de rendre un visuel à l’utilisateur sur l’état de sa commande avec la fonction Affichecommande() dans une textbox.

Pour effectuer le visuel en arrière plan commun à toutes ces pages, nous avons mis une image( Source=””)avec la commande dans la Grid qui prenait toute la fenêtre. Le titre a été crée avec un Label. L’utilisateur à pu interagir avec l’interface à travers différents outils wpf qui sont les Textbox, les bouttons , les bouttons des messagesbox.show ( avec la commande MessageBoxResult. ), les choix ComboBoxItem de la combobox. Les textbox ont été utilisés à fois pour recevoir un message de l’utilisateur ainsi que pour lui en délivrer un. Les boutons ont permis à la fois de passer de naviguer entre les pages de navigation, d’effectuer une sélection de pizza ainsi que de supprimer des éléments de la commande.