**RAPPORT**

*Un projet fait par :* ***BURY Jean-Baptiste & BENYEMNA Hamza***

**Diagramme UML :**

**Liste de classes (leur type ; attributs):**

* **Pizzeria** (normale ; nom, adresse, téléphone, liste de **clients**, liste de **livreurs**, liste de **commis**, liste des **commandes**)
* **Personne** (abstraite ; nom, prénom, ville)
* **Effectif** (mère et fille ; date d’embauche, état (= ‘*sur place’*, ‘*en congé’* ou en ‘*livraison’*))
* **Commis** (fille sealed ; )
* **Livreur** (fille sealed ; véhicule)
* **Client** (fille ; téléphone, adresse, date de 1ere **commande**)
* **Produit** (abstract ; )
* **Pizza** (fille ; type(=’*Margherita*’, ‘*Royale*’, *’Chèvre Miel’* ou ‘*Quatre Fromages’*), taille(=’*Grande*’, ‘*Moyenne*’ ou ‘*Petite*’))
* **Boisson** ( fille ; type(=’*Eau*’, ‘*Coca*’, ‘*Fanta*’ ou ‘*Oasis*’), volume(=’*0.5L’* ou ‘*1.5L’*))
* **Commande** (normale ; liste de **pizza** ( != null ), liste de **boissons**, numéro de **commande**, nom du **client**, nom du **commis,** nom du **livreur,** état(=’en préparation’, ‘en livraison’ ou ‘fermée’), solde(=’non payée’ ou ‘payée’), facture)
* **IPromo** (interface ; booléen qui vérifie qu’un **client** dispose d’une réduction)
* **ICopie** (interface; string qui retourne une copie de la facture de la **commande**)
* **IVerifieEtat** (interface ; booléen qui vérifie l’état d’une commande)
* **IVerifiePayee** (interface ; booléen qui vérifie qu’un client dispose d’une réduction)

**Lien entre les classes :**

* **Effectif** et **Clients** héritent de **Personne**
* **Livreur** et **Commis** héritent de **Effectif**
* **Pizza** et **Boisson** héritent de **Produit**
* L’interface **IPromo** est implémentée dans **Client**
* Les interfaces **IVerifiePayee**, **IVerifieEtat** et **ICopie** sont implémentées dans **Commande**

**Explications :**

**Personne** et **Produit** sont abstract car elles n’héritent d’aucune classe et seules leurs classes filles sont instanciés.

**Commis** et **Livreur** sont sealed car ce sont les dernières classes de l’héritage (aucune classe n’héritent d’elles).

**Explication du programme :**

**Pizzeria :**

On crée des listes de **Commandes**. C’est pratique pour avoir un historique de toute les commandes que l’on trie par date (du plus au moins récent). On possède aussi des méthodes qui peuvent afficher l’historique des commandes concernant une période spécifique. Enfin, il y a une méthode qui permet de retourner la moyenne du prix d’un commande.

De la même manière, on créer des listes de Commis, de Livreur travaillant dans la pizzeria et de Client qui ont commandé. Il existe des méthodes qui permettent d’ajouter ces Personnes dans les listes respectives.

Toutes ces méthodes sont simples grâce au fait d’établir des listes pour les manipuler.

**Commande :**

Nous avons ajouté en argument la facture qui est pratique pour résumé les informations de la **Commande** et facilement manipulable car il suffit de l’afficher

On y implémente les interfaces **IComparable** pour le tri, **IVerifiePayee** qui vérifie si la commande est payée, **IVerifieEtat** qui vérifie l’avancement de la commande et **ICopie** qui permet de faire la copie de la facture donnée au **Client** pourle **Commis**.

Les interface de **IVerifieEtat** permettent de savoir si la commande est préparée, auquel cas on l’ajoute à la liste des **commandes** préparées par le **Commis**, ou si elle est livrée, auquel cas on l’ajoute à la liste des **commandes** livrées par un Livreur.

De plus, Nous avons ajouté la notion de promotion qui pour tout client de plus de 10 **commandes**, lui réduit le prix de ses **commandes** à venir de 5%. Il y a donc 2 méthodes de calcul du prix : avec ou sans promotion.

**Produit :**

Un **Produit** est obligatoirement une **Boisson** ou une **Pizza** donc nous elle est en abstract. Cependant, il est possible d’instancier un **Produit** en tant que **Pizza** ou **Boisson** *(ex : Produit maPizza = new Pizza(« Magherita », « Grande »).* C’est pourquoi nous avons établi un *virtual/override* pour la méthode de calcul de prix qui renvoie à la bonne méthode selon s’il s’agit d’une **Boisson** ou d’une **Pizza**.

**Boisson :**

Il y a 4 types de boissons possibles dans 2 volumes différents. Ainsi, le prix diffère selon le type et le volume. Pour cela, nous procédé à un *switch/case* concernant le choix du type, suivi d’un simple *if/else* pour la saisie du choix du volume. Cette méthode est à saisie sécurisée, tant dans le type que le volume.

**Pizza :**

Le raisonnement pour la **Pizza** est le même que pour la **Boisson**.

**Personnes :**

Cette classe permet grâce à l’interface **IComparable** d’effectuer les tris (par nom et par ville) qui permettent le tri de n’importe quel entité (**Livreur**, **Commis** ou **Client**) qui hérite de **Personne** grâce à l’héritage. Elle permet aussi le polymorphisme concernant l’affichage des attributs à l’aide du *override*.

**Commis :**

Nous avons ajouté une liste de **commandes** préparées. Grâce à la méthode d’interface **IVerifieEtat**, il est facile d’ajouter une **commande** lorsqu’elle a fini d’être préparée. De plus, la taille de sa liste nous donne accès directement au nombre de **commandes** que le **Commis** a préparé.

**Livreur :**

On y a ajouté une liste de **commandes** livrées avec le même raisonnement que pour le **Commis** concernant l’ajout des commandes finies d’être livrées grâce aux méthodes d’interfaces et le nombre de **commandes** livrées grâce à la taille de cette liste.

**Client :**

Pour les mêmes raison pratiques que **Livreur** et **Commis**, on ajoute une liste de commandes qui prend en compte que les commandes payées, vérifiée à l’aide de la méthode d’interface **IVerifiePayee**. Ainsi, le calcul du prix cumulé des **commandes** par **clients** est plus simple, ainsi que le calcul de la moyenne du prix de ses **commandes**.

D’une part, la méthode d’interface **IComparable** y est implémentée car elle est utile pour le tri des **clients** par prix cumulé qui permet de savoir quel **Client** dépense le plus ou non. D’autre part, **IPromo** pour savoir si l’on applique la promotion ou non au **Client**.

**Remarque :**

Les délégations ont été utiles pour rechercher les **commandes** dont la date était comprise entre 2 périodes. Par ailleurs, elles servent aussi à l’affichage des listes dans le Program.cs en codant la fonction en *lambda.*

**WPF :**

**Menu Principal**

Le menu principal se compose de 5 modules :

* Nouvelle **Commande**
* **Commandes** (incomplet, il y a juste l’affichage de la liste des)
* **Clients**/**Effectif** (incomplet)
* **Statistiques** (non implémenté)
* **Autre** (non implémenté)

Le code du projet sans WPF est complet (à l’exception du chargement de fichier de commandes) mais nous n’avons pas pu exploiter la totalité de celui-ci en WPF.

**Module nouvelle commande :**

Ce module permet de saisir une nouvelle **Commande**.

Tout d’abord une fenêtre permettant de rechercher un **Client** à l’aide de son numéro de téléphone s’affiche (RechercheClient.xaml). Si le **Client** commande pour la première fois, on peut alors cliquer sur le bouton ***ajouter*** qui va alors afficher une nouvelle fenêtre permettant de saisir les informations du **Client** afin de l’ajouter à la liste de **clients** de la **Pizzeria** (AjouterClient.xaml). Une fois le **Client** ajouté ou trouvé, on accède alors la fenêtre d’affichage des informations du **Client** (AffichageDuClient.xaml). Le bouton ***valider*** permet de confirmer la sélection du **Client** et d’ouvrir la fenêtre de choix des **pizzas** (ChoixDeLaPizza.xaml). Cette dernière permet de sélectionner le type et la taille d’une **Pizza** à l’aide de menus déroulants. Le bouton ***reset*** permet de réinitialiser la **Commande** (i.e. La liste de **pizzas** et la liste de **boissons** sont vidées) alors que le buton ***supprimer*** permet de retirer la dernière **Pizza** de la **Commande**. Le bouton ***suppléments*** permet d’accéder à la fenêtre de choix des **boissons** (ChoixBoisson.xaml). Sur cette page le buton ***reset*** retire toutes les **boissons** de la **Commande** alors que le bouton ***supprimer*** permet de retirer la dernière **Boisson** de la **Commande**. Sur ces deux dernières pages, le bouton ***valider*** permet d’accéder à la fenêtre d’affichage de la **Commande** (AffichageCommande.xaml). Cette fenêtre est un récapitulatif des **produits** composant la **Commande** avec leur prix unitaire ainsi que le prix total de la **Commande**. Le buton ***suivant*** permet d’accéder à la fenêtre de sélection du **Commis** (SelectionCommis.xaml). Cette fenêtre permet de sélectionner le **Commis** qui va s’occuper de préparer la **Commande**, une fois le **Commis** sélectionné, le bouton ***terminer*** va valider la **Commande** et l’ajouter à la liste de **commandes** de la **Pizzeria**. Ilpermet aussi de retourner sur le menu principal.

Chaque fenêtre est munie d’un bouton ***retour*** qui permet de revenir sur la page précédente.

**Module Commande :**

Ce module permet d’afficher la liste de toutes les **commandes.**

Mais il permettrait aussi d’afficher la liste des **commandes** en préparation, celles en cours de livraison, celles payées ou non et celles fermées (payée ou non) grâce à des *listview*.

On pourrait ainsi sélectionner une **Commande** en préparation et la passer en livraison, sélectionner une **Commande** en livraison et la passer en payer et fermée ou non payée et fermée à l’aide de boutons et de *listview*.

**Module Clients/Effectif :**

Ce module permet d’afficher la liste des **clients**, celle des **livreurs** et celle des **commis**.

Mais ce module permettrait aussi d’entrer, supprimer ou modifier un nouveau **Client** ou un nouveau **Commis** ou un nouveau **Livreur** ainsi que d’afficher la liste des **clients**, celle des **livreurs** et celle des **commis** par ordre alphabétique, par villes et par montant des achats cumulés pour les **clients** (toutes ces méthodes sont disponibles dans le projet sans WPF).

**Module Statistiques :**

Ce module permettrait d’afficher par **Commis**, le nombre de **commandes** gérées, d’afficher par **Livreur** le nombre de **livraisons** effectuées, d’afficher les **commandes** selon une période de temps d’afficher la moyenne des prix des commandes d’afficher la moyenne des comptes **clients** (toutes ces méthodes sont disponibles dans le projet sans WPF).

**Module Autre :**

Ce module permettrait d’implémenter d’autres fonctionnalités qui ne font pas parties des modules précédents.

**Créativité :**

* **Promotion : affecter une remise de 5% sur toutes les commandes d’un client qui a commandé (et payé sa commande) plus de 10 fois.**