Compte rendu de programmation: TP3

ESIREM DIJON

CHAPUS Louka

PAUZIE Laureline

Groupe 3A IE

TP n°3

Pour commencer, suite à notre premier TP, nous avons choisi de continuer par le TP3. La raison en est qu'après le TP1, nous avons remarqué que notre niveau doit être environ entre le débutant et l'intermédiaire. Alors, nous avons décidé de continuer de manière croissante dans le niveau des TP. Le TP3 se trouvait donc parfait pour tester notre niveau et vérifier donc si notre niveau était bien dans l'intermédiaire et non seulement débutant.

L'objectif de ce TP consistait à concevoir une application de magasin en ligne nommée EasyStore qui pouvait gérer les différents produits, clients et commandes.

Comme au TP1, nous avons fait le choix d'ajouter la commande suivante:

- *#pragma once*

Cette commande servait à n'inclure qu'une seule fois les autres classes nécessaires à l'implémentation ce qui nous permettait d'économiser de la mémoire. De même, nous avons aussi ajouté les commandes suivant dans les fichiers .h pour ne pas avoir à définir une classe plusieurs fois au mauvais endroit.:

- *#ifndef*
- *#define*
- *#endif*

Pareil au TP1, nous avons aussi ajouté pour chaque classe getters constants.

Dans ce TP, nous avons réparti chacun un travail, Louka créait des classes **Produit** et **Magasin** tandis que Laureline créait les classes **Commande** et **Client**. Puis nous vérifions s'il y avait des fautes et les remettons ensemble. C'était aussi la raison pour laquelle nous avons 2 main: le main pour Louka et le main1 pour Laureline. La décision pour créer deux main est prise par le fait qu'à chaque fois un de nous deux change le main, nous n'arrivons pas à faire le push dans Git. Nous devrions faire un pull d'abord puis modifier notre main pour pouvoir par la suite faire un push. Ainsi, nous avons décidé de créer deux main pour faciliter la programmation à deux, puisque maintenant chacun pouvait tester ses fonctions comme il le voudrait sans avoir à déranger l'autre.

Puis, Louka commença à créer la classe **Magasin** tandis que Laureline créait la classe **Produit**. Dans la classe **Magasin**, nous avons créé plusieurs fonctions pour pouvoir bien gérer le magasin, chaque fonction correspondait à une question précise demandée dans l'énoncé. Tous les paramètres de cette classe étaient des vecteurs qui prennent en arguments des autres classes comme **Produit**, **Client** et **Commande**. Les fonctions concernant le panier du client étaient reprises des fonctions de gestion du panier du client dans la classe **Client**. Ici, nous sommes dans le cas où nous acceptons tous les produits et nous ne vérifions pas si ce produit est correct. De même pour des clients, nous acceptons tous les noms et prénoms. La classe Magasin a été créer au tout début et par la suite nous la complétons avec la création des autres classes et en fonction de l'énoncé.

Pour pouvoir exporter et importer les données sur un .txt, on le fait dans la classe **Magasin** avec la fonction *mise_a_jour()* qui consiste à écrire dans chaque fichier (produit et client) ,avec un formalisme définie à l'avance, toutes les informations stocké dans les variables membres pour pouvoir réutiliser les informations lors d'un autre lancement du programme. Il y a aussi la fonction *lecture_donne()* qui permet de faire l'inverse en prenant

les informations stockées dans les .txt et recréé les listes dans les variables de la classe. Pour les deux fonctions on utilise l'objet std::ifstream qui permet d'interagir avec des fichiers .txt.

Nous avons un détail dans cette classe qui peut être dérangeant lorsque nous voudrions afficher les détails d'un produit à l'écran, nous devrions écrire correctement le nom du produit sélectionné. C'est-à-dire que si nous faisons une faute de frappe, le produit n'apparaît pas et nous recevons un message d'erreur. Il faut donc respecter les majuscules et les minuscules dans le nom du produit. Cette erreur est la même pour l'affichage des clients et des commandes.

Par la suite, la classe **Produit** prend en paramètre des chaînes de caractères, un int et un float pour pouvoir afficher en détails le prix et la quantité du produit. Dans cette classe, nous avons vérifié que les prix et les quantités ne descendent pas plus bas que 0, d'où nous recevons un message d'erreur qui indique que nous avons introduit une quantité ou un prix négatif. Enfin nous avons fini par surcharger l'opérateur << pour un affichage correct des informations suivant d'un produit:

- Le nom
- La quantité
- Le prix

Ces détails seront affichés sur une même colonne selon leur type.

Ensuite, nous avons implémenté la classe **Client**, cette classe ne prenait en paramètre soit des chaînes de caractères soit un vecteur de type **Produit**. Dans la classe Client, nous n'avons pas fait une vérification pour assurer un objet valide, ici nous acceptons tous les noms, les prénoms et les identifiants. Et comme la classe précédente, nous avons aussi surcharger l'opérateur << pour afficher proprement les informations d'un client dans le format demandé par l'énoncé, c'est-à-dire bien encadré.

Puis, nous avons créé la classe **Commande**, qui reprend les arguments des classes **Client** et **Produit**. Cette classe prend en compte un vecteur de type **Produit** car nous ne savons pas combien de produits chaque commande va avoir et nous avons besoin de pouvoir modifier des éléments dans une commande à tout moment. Un vecteur est donc indispensable pour pouvoir avoir de la flexibilité dans la gestion de la commande. De plus, nous avons créé un variable booléen pour la vérification lorsqu'une commande est validée ou non. Par défaut, lorsque nous créons une commande, elle est non validée et nous validons la commande de manière manuelle. Cependant, nous n'avons pas pu utiliser cette vérification car nous n'avons pas eu assez de temps pour terminer le terminal graphique pour la gestion des commandes.

Nous avons aussi créé la fonction panier qui servait à montrer tous les produits ajoutés dans le panier lors de la commande. Lorsqu'un client ajouté un produit dans son panier, la commande associée à ce client va automatiquement avoir ce produit, de même lorsqu'un client modifie la quantité du produit qu'il a ajouté ou lorsqu'il supprime un produit. Cela nous permettait de voir à tout moment les produits choisis par le client ainsi que lorsqu'il modifiait sa commande. Cette fonction ressemble fortement à la fonction *afficher_panier_achat()* dans la classe **Client**. La raison pour laquelle nous avons créé cette fonction est de vérifier si la classe Commande fonctionne bien. La fonction *panier()* nous permettait de voir si les commandes de la classe **Commande** fonctionnent sans erreur. Encore une fois, nous avons surchargé l'opérateur << pour avoir les détails d'une

commande avec deux cas pour la validité de la commande, une lorsqu'elle n’est pas valide et une lorsqu’elle est valide.

Finalement, dans le main nous testons toutes les commandes et des classes créées. Les deux mains contiennent les mêmes fonctions pour le terminal de graphique, cependant c’est dans le main de Louka que nous avons décidé de mettre toutes les commandes pour tester. Avant de créer les éléments du menu, nous avons testé le bon fonctionnement des fonctions créées. Nous avons commencé par la création d’un magasin qui s’appelle EasyStore, puis nous avons créé et ajouté des produits dans ce magasin. Nous avons ensuite créé des clients et vérifié si nous pouvions modifier, afficher et supprimer des produits dans le magasin. Nous avons aussi vérifié que les affichages des produits, des clients, les commandes et le magasin étaient sans faute avec la surcharge. Puis nous commençons à créer les fonctions nécessaires dans le main pour l’affichage et la gestion du menu.

Pour conclure, ce TP nous a permis de connaître plus concrètement notre niveau en programmation. Nous avons pu voir les différentes applications plus complexes pour programmer comme les fonctions *offstream()*. Ce TP nous permet d’appliquer et améliorer notre niveau dans la programmation, nous pouvons coder plus facilement et aussi plus rapidement comparé au TP1. Malheureusement, nous n’avons pas pu terminer notre TP en entier et donc le menu restera non fini.