LEHNERT William

Projet EasyStore (TP3)

Présentation des Classes

Classe Product

Attributs		
Type	**Nom**	**Explication**
String	***Title***	Nom du **produit**
String	***Description***	Description brève du **produit**
Int quantity	***Quantity***	Quantité disponible (dans le **Store** ou dans un panier)
Double price	***Price***	Prix du **produit** (en euros)

On retrouve des constructeurs classiques, un par défaut et un selon les attributs. On a aussi des Getters/Setters simples et une surcharge de l'opérateur << pour afficher le **produit**. Cette classe est extrêmement simple, elle sert surtout à représenter un objet pour les autres classes.

Classe Client

Attributs		
Type	**Nom**	**Explication**
Int	***Id***	N° d'identifiant du **Client**, unique, évite les homonymes
String	***First_name***	Prénom du **Client**
String	***Last_name***	Nom du **Client**
Vector<Product>	***_cart***	Panier du **Client**

On retrouve des constructeurs classiques, un par défaut et un selon les attributs. On a aussi des Getters/Setters simples et une surcharge de l'opérateur << pour afficher les informations du **Client**.

Pour représenter le panier, j'ai choisi d'utiliser des vectors pour leur dynamisme (on considère que le panier n'a pas de limites) et car j'ai l'habitude de travailler avec des vectors (plutôt que des arrays par exemple).

Methods		
Retour type	**Nom**	**Explication**
Void	***AddProductToCart(…)***	Ajoute un **produit** au panier du **Client** selon sa quantité
Void	***ClearCart(…)***	Vide le panier du **Client**

Void	***Update…Cart(…)***	Modifie la quantité d'un **produit** dans le panier du **Client**. Pour cela on parcourt le panier du client en comparant les noms des **produits**. S'ils sont identiques, alors on modifie sa quantité.
Void	***Remove...Cart(…)***	Retire complètement un **produit** du panier du **Client**

Dans la même idée que la classe **Produit**, cette classe **Client** est surtout utilisée par les autres classes comme un simple objet. J'ai seulement ajouté un vector de **produits** pour simuler un panier et quelques méthodes de gestions sur ce panier.

Classe Order

Attributs		
Type	**Nom**	**Explication**
Client	***Client***	A chaque **commande** est toujours attaché un **Client**
Vector<Product>	***productsToBuy***	Liste des **produits** commandés par le **Client**
bool	***isDelivered***	Booléen représentant le statut de la **commande** (livrée ou non)

On retrouve des constructeurs classiques, un par défaut et un selon les attributs. On a aussi des Getters/Setters simples et une surcharge de l'opérateur << pour afficher les informations de la **commande**.

Pas de méthodes particulières. La **commande** est vue comme un simple "historique" d'une liste de **produits** commandés/achetés par le **Client**. On consulte ou choisit de modifier leur validité depuis l'historique des **commandes** du **Store**.

Classe Store

Attributs		
Type	**Nom**	**Explication**
Vector<Product>	***_products***	Liste des **produits** disponibles à l'achat dans le **Store**.
Vector<Client>	***_clients***	Liste des **Client** enregistrés dans le **Store**.
Vector<Order>	***_orders***	Liste des **commandes** effectuées dans ce **Store**.

On retrouve des constructeurs classiques, un par défaut et un selon les attributs. On a aussi des Getters/Setters simples et une surcharge de l'opérateur << pour afficher les informations du **Store**.

Pour représenter toutes ces listes, j'ai choisi d'utiliser des vectors pour leur dynamisme (on considère que le **Store** n'a pas de limites de **produits** à présenter à l'achat, qu'il n'a pas de limite de **Client**, idem pour les **commandes**).

Pour afficher tous ces éléments, on dispose de fonctions "display" (One/All) qui affiche un ou tous les éléments. Pour accéder ou afficher des **commandes**, j'ai choisi de renseigner l'Id du **Client** (pour trouver toutes les **commandes** associées à ce **Client**).

Methods		
Retour type	**Nom**	**Explication**
Void	***addProduct(…)***	Ajoute un **produit** à la liste des produits
Void	***addNewProduct(…)***	Idem ci-dessus, vérifie son existence avant
Void	***productQuantityUpdate(…)***	Modifie la quantité de **produit** disponible
Void	***addNewClient(…)***	Ajoute un **Client** à la liste des **Client** du **Store**
Void	***addProductToClientCart(…)***	Ajoute un **produit** au panier du **Client**
Void	***Remove…ClientCart(…)***	Supprime (en totalité) un **produit** dans un panier
Void	***Update…Cart(…)***	Modifie la quantité de **produit** dans un panier
Void	***ValidateOrder(…)***	Valide une **commande** (voir ci-dessous)
Void	***UpdateOrderStatus(…)***	Met à jour le statut d'une **commande**
Client	***getOneClient(…)***	Obtenir un **Client** du **Store** selon son Id
Order	***getOneOrder(…)***	Obtenir une **commande** selon l'Id du **Client** qui a passé cette **commande**.

Certaines méthodes se ressemblent dans leur intérêt mais elles diffèrent dans leur vérification : Par exemple, addProduct(…) et addNewProduct(…) ajoutent toutes les deux un **Product** au stock du **Store** mais la deuxième vérifie si le **produit** existe déjà. Si oui, elle ajoute juste la quantité de **produit** renseigné en paramètre.

Pour valider une **commande** depuis le panier d'un **Client**, on utilise la méthode validateOrder(…). Elle vérifie déjà si le **Client** existe. Si non, la **commande** sera déjà refusée (via un booléen false utilisé ensuite). De même, on vérifie que le panier du **Client** n'est pas vide (sinon on ferait une **commande** vide, donc inutile, il n'y a aucun intérêt à enregistrer des **commandes** nulles). Une fois toutes ces vérifications faites, le statut de la **commande** courante est validé. Ensuite, on supprime la quantité "achetée" du stock du **Store** et on vide le panier du **Client**.

Dans la partie mise en commentaire, on peut effectuer/ajouter des opérations de virements avec d'autres classes (banque…) (tout en faisant des vérifications sur la possibilité de paiements avant : argent disponible, confirmation de la fiabilité du site/ **Store** par la banque, etc…).

Pistes d'amélioration et poursuites possibles du projet

Plusieurs remarques et indications sont écrites de manière éparpillée entre les lignes de code. En voici, un condensé :

- Ajouter des vérifications diverses : sur les quantités / prix positifs des **produits** par exemple.
- Le "menu" décrit dans le main peut être une classe à part prenant en compte un **Store** dans ses attributs ou en paramètre dans ses méthodes.
- La dernière question subsidiaire n'a pas été traitée. Elle consiste à l'enregistrement ou à la lecture des données d'un Store dans un fichier texte (données qu'on peut récupérer et réutiliser lors d'un prochain lancement du programme, par exemple).

Conclusion / Expérience de programmation sur ce projet

Durant l'écriture de ce programme, j'ai renforcé mes connaissances sur la programmation objet, notamment sur la gestion des classes. J'ai découvert de nouvelles fonctionnalités simplifiant l'écriture du code comme la surcharge de l'opérateur <<. Au fur et à mesure de son écriture, de nouveaux problèmes se présentaient devant moi. J'estime avoir trouver des idées suffisamment intéressantes pour répondre à ces difficultés, tout en me permettant d'élargir les possibilités (=ouverture) du programme. J'ai ainsi appris à mieux m'adapter aux demandes de l'énoncé tout en continuant à conclure la réalisation de ce projet.