ESIREM INFOTRONIQUE 3A

Compte rendu TP1 & TP3 : Programmation C++

Auteurs :

DYNAK Tom, NTAWINIGA Baréké

2022-2023

TP1 : Gestion d'une bibliothèque

Le TP1 consiste à coder la gestion d'une bibliothèque, le fait de stocker les livres et leurs informations, leur auteur, de gérer les emprunts avec des lecteurs identifiés. On a des restrictions à respecter, tel que le fait qu'un livre ne soit emprunté que par un lecteur à la fois.

On s'est mis d'accord de faire le TP1 en premier, pour ne pas démarrer en difficultés.

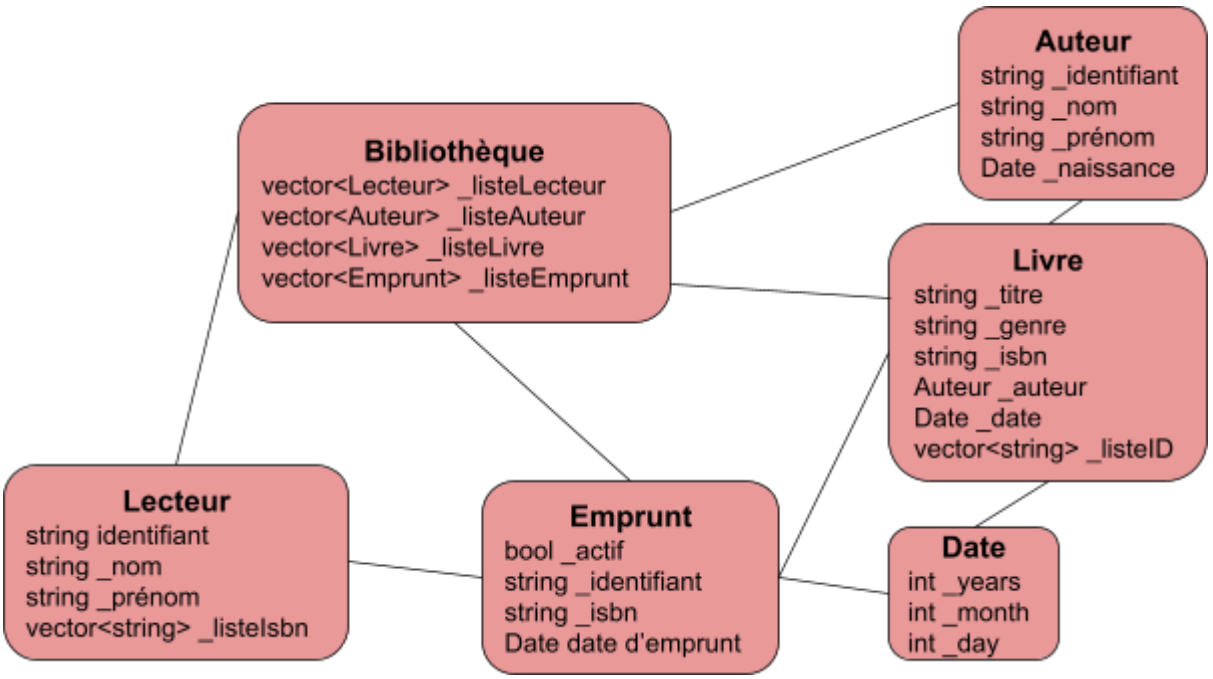

Schéma des classes et de leurs relations.

Tout d'abord, nous avons récupéré la classe *Date* et ajouté un paramètre correspondant à l'année. Cette classe interagit avec plusieurs autres classes comme la classe Livre et la classe Emprunt. Malgré tout, nous n'avons pas implémenté dans Date la capacité à lire la date de l'ordinateur pour le cas de la date d'emprunt.

Durant la création de la classe Livre, de ses getters, on a différencié *getAuteur()* donnant seulement le nom entier de l'auteur et *getAuteurC()* donnant une copie de la classe. Le livre contient une copie de auteur car lors de l'instanciation, si c'est une référence, alors on ne peut pas instancier la variable contenant l'auteur du livre.

Pour la classe Lecteur on a choisi d'inclure un vector de string contenant une liste des isbn des livres qu'il a emprunté.

La classe Emprunt contient l'isbn du livre emprunté et l'identifiant du lecteur, ce qui permet de les retrouver avec une fonction de recherche.

On a ensuite créé la classe Bibliothèque en choisissant de faire des std::vector de Lecteur Emprunt, Livre et Auteur, les instances de ces classes se feront copier mais toutes les méthodes de Bibliothèque utilisent les listes de la même classe. De cette manière on n'aura pas à se soucier du fait que les changements dans la liste ne se font pas à l'instance de classe originale. La classe comprend toutes les méthodes gérant les autres classes telle que l'emprunt de livre et leur restitutions.

On a implémenté l'opérateur << aux classes Livre, Auteur et Lecteur qui seront utilisés dans les fonctions de Bibliothèque visant à afficher des informations

Dans le main.cpp on va tester si notre bibliothèque fonctionne, d'abord on crée les instances de Bibliothèque, puis des auteurs qui vont être utilisés pour ajouter des livres. Ces mêmes livres qui sont uniques à cette bibliothèque et ne se trouveront jamais en stock autre part. On s'est aussi assuré que l'on ne peut pas emprunter un livre déjà pris. On teste les méthodes permettant de rechercher un livre selon son auteur, sa disponibilité ou si il est emprunté par un lecteur, toutes ses méthodes affichent leur résultat dans la console.

Quant à l'utilisation de git, des commit on été fait a chaque question du sujet finis, ou lors de l'ajout de nouvelle méthodes ou classes.

Logiciels utilisés:

Sublime Text avec le terminal linux ou le site replit.com ou Vscode pour Windows.



Problèmes rencontrés:

On a eu beaucoup de problèmes à gérer les github merge et récupérer ce qu'on a mis dans github stash, ce qui nous a poussé à éviter de travailler sur les mêmes fichiers. Nous avions donc décidé de travailler sur différents TP en parallèle après 3 séances de TP. D'autres problèmes incluent de faire fonctionner g++ et git en changeant d'ordinateur. Ainsi que le fait que replit.com ajoute un "initial commit" sur la branche à chaque fois que l'on se reconnecte au dépôt github. Sous Windows, nous avons utilisé replit.com au lieu de Vscode car nous n'avions pas réussi à faire fonctionner make.

Cependant, ce 1er TP nous a permis d'apprendre à utiliser Github, et de manipuler git sur un terminal linux. On a aussi appris en C++ de coder avec des classes, de faire des vectors de classes et les erreurs qui peuvent arriver, par exemple que si on insère une classe dans le vector elle est copié est donc les changements ne se feront pas sur l'original.

TP3 : Application de magasin en ligne (EasyStore)

Ce TP consiste à concevoir une application de magasin en ligne. Dans ce TP, un certain nombre de classes vont interagir entre elles.

J'ai choisi ce TP car je ne me sentais pas encore bien à l'aise sur la programmation C++, et le choix s'est porté sur un TP de niveau intermédiaire, afin de voir si nous étions capable tout de même d'augmenter le niveau du TP d'un cran, par rapport au TP1 qui était de niveau débutant.

Tout d'abord, pendant que l'un terminait le TP1, nous avons pensé qu'il valait mieux que l'autre commence un nouveau TP pendant ce temps (car nous avions pris un peu de retard, le temps de s'habituer au langage et à utiliser GitHub).

Puis, après que le TP1 soit terminé, nous nous sommes entraidés pour avancer au mieux sur le TP3. Cependant, nous avons eu un problème avec le git push (détaillé sur **Problèmes rencontrés**).

Logiciels utilisés:

Sublime Text avec le terminal linux ou le site replit.com ou Vscode pour Windows.



Problèmes rencontrés:

Le plus gros problème est que replit.com , l'outil que l'on a utilisé pour coder sur Windows et qui pouvait compiler le code contrairement à Vscode, ne permettait pas de push sur les dépôts dont on est pas le propriétaire. Le problème a persisté après avoir donné toutes les autorisations possibles sur github et en partageant bien l'invitation de collaborer sur le github. L'un de nous deux ne pouvait donc pas push à certains moments, et par conséquent ce dernier devait copier le code et l'envoyer par message au propriétaire en question du github.

Développement du code:

Tout d'abord, il est dans un premier temps important de décomposer le problème en sous-problème et y réfléchir avant d'écrire du code.

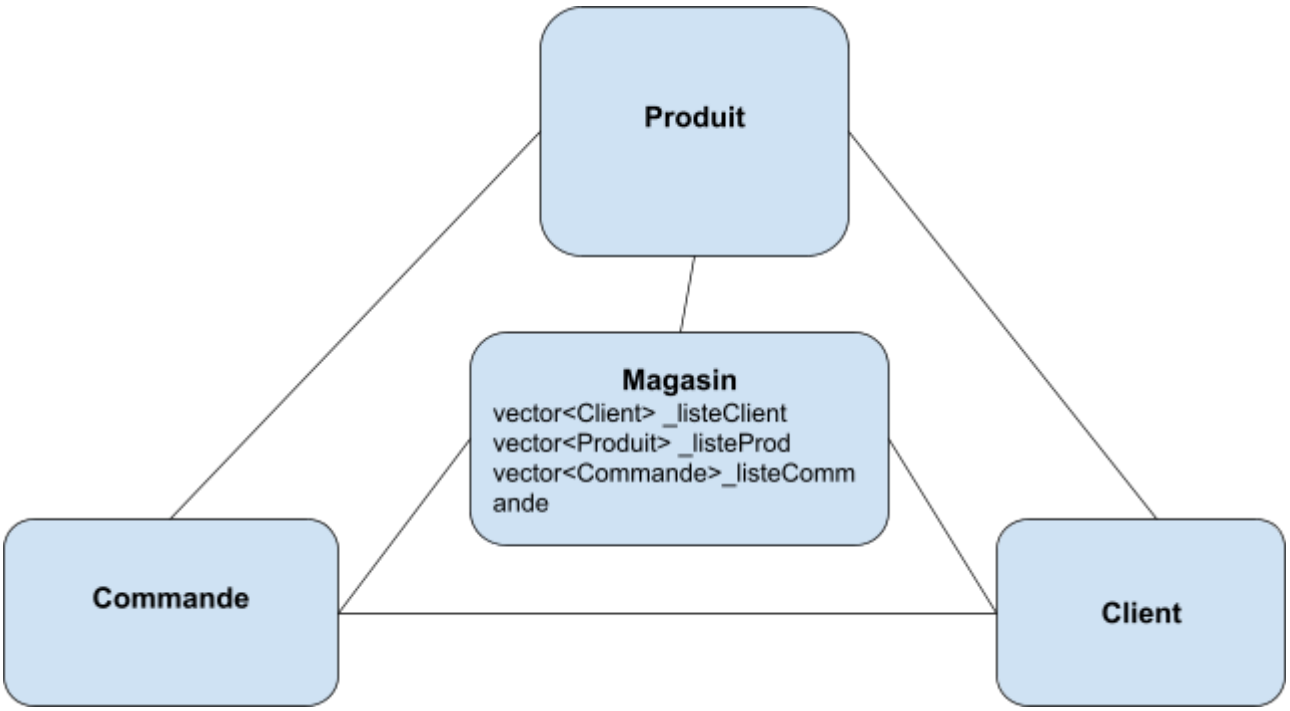

Schéma des classes et de leurs relations.

Comme nous pouvons le remarquer, nous allons probablement utiliser très peu de classes (au minimum 4), contrairement au TP1.

Cependant, ici la difficulté résidait sur les nombreuses intéractions entre chaque classe. Le risque était de se perdre sur les nombreuses méthodes et fonctions getters/setters utilisés sur certaines classes, et ne pas oublier pour chaque nouvelle méthode utilisée, vérifier si les paramètres sont toujours bons.

Tout d'abord, nous avons créé une première classe "Magasin" qui sera notre classe principale (elle sera la principale source entre chaque interaction des différentes classes).

Ensuite, nous avons créé une classe Produit avec 4 variables membres qui seront utilisés pour :

- Le nom du produit (std::string)
- Une description du produit (std::string)
- Une quantité (unsigned int)
- Un prix (Float)

Pour l'affichage du prix, nous avons décidé de créer 2 variables int (euro et centime), dans le but de toujours afficher le prix de cette façon : xx.xx. Puis, nous avons ajouté une méthode pour modifier la quantité d'un produit.

On commence par réaliser des intéractions entre la classe Produit et Magasin, dans le but de référencer des produits grâce à une méthode (le produit rentre dans le std::vector<Produit> de classe Magasin), puis nous avons réalisé les fonctions helper/méthodes demandées.

Pour la création de la classe Client, nous avons décidé de décomposer le produit lorsqu'il rentre dans le panier. C'est-à-dire, le panier est composé des 4 mêmes variables qu'un produit mais répartis sur 4 std::vector. En somme, pour 1 produit entré dans le panier d'achat du client correspond à 1 itération des tableurs std::vector (ce qui fait que le premier produit entré dans le panier sera à l'emplacement 0 de chaque vecteur, puis pour un deuxième produit à l'emplacement 1 de chaque std::vector et ainsi de suite).

Il est vrai cependant qu'on aurait pu directement reprendre seulement un std::vector<Produit> pour créer le panier d'achat du client. Puis nous avons réalisé les méthodes/helper demandés et surcharger l'opérateur du client.

Comme la classe Client est réalisée, il est naturel de revenir sur la classe principale (Magasin) afin de refaire les étapes similaires à ceux de l'implémentation d'un Produit dans la liste des produits du magasin. Cependant, nous savons que sur ce code actuel, malgré que les méthodes demandées soient présentes, il manque les conditions comme le fait de restituer la quantité d'un produit dans le magasin lorsque le produit dans le panier est supprimé (dans le cas où ce n'est pas une commande, bien entendu) par exemple.

Ensuite, nous avons créé la dernière classe, la classe Commande. Afin d'éviter de traîner un objet Client comme variable membre, nous avons préféré le définir par un std::string ident où grâce à la méthode searchClient() réalisée plus tôt, nous pouvons directement récupérer le client issu de la commande en question, avec une liste de produits, codée de la même manière que pour le panier du client.

Enfin, en lien avec la classe Commande, nous avons ajouté des fonctionnalités à Magasin et nous avons commencé à créer une interface graphique comprenant un menu et des sous-menus. Pour cela, nous avons utilisé 2 variables, l'une qui permet de pointer le menu au ou sous-menu auquel on veut accéder et une deuxième variable qui permet d'indiquer dans quelle "couche" des menus nous sommes. Par exemple, le menu principal serait la couche 1 et après accès à un premier menu, nous sommes à la couche 2.

Conclusion:

Ce TP programmé, malgré qu'il ne soit pas totalement abouti et terminé, nous a permis d'acquérir des connaissances sur la programmation orientée objet, tout en renforçant nos connaissances en mettant en pratique ce qui avait été vu en cours magistral. Pour ma part (Tom Dynak), durant ces 2 dernières années, j'avais seulement appris la programmation avec Arduino en C/C++ (avec une approche plutôt rudimentaire).

(Tom Dynak) Ce qu'il me manque à acquérir pour le futur métier d'ingénieur est la vitesse à traiter rapidement et efficacement une problématique, notamment en programmation.