Rapport de Projet

Victoire CYROT & Fares EZZAOUIA

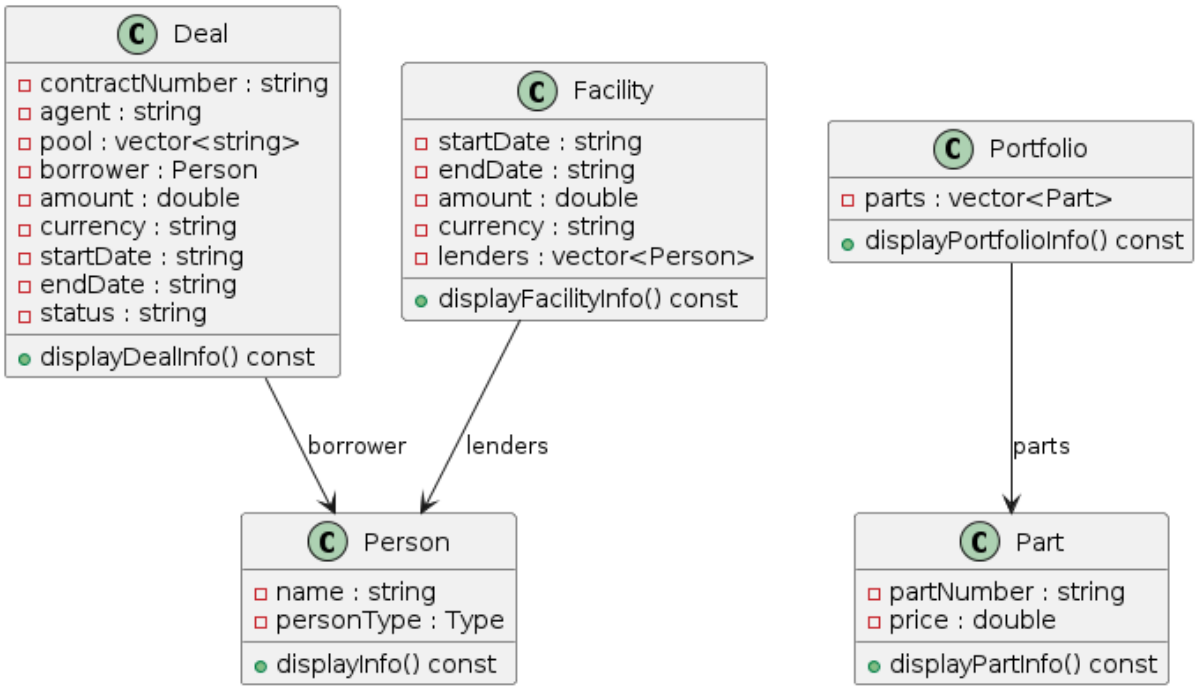
Introduction

Ce projet avait pour objectif de concevoir un système de gestion de transactions financières impliquant des deals, des facilities, des parts et des personnes (borrowers et lenders). Nous avons opté pour une architecture orientée objet afin de structurer et de gérer efficacement les entités et leurs relations. Le code a été écrit en C++, et nous avons adopté des conventions de nommage en anglais pour maintenir la cohérence et la lisibilité du code.

Lien du Github : https://github.com/vcyrot/CyrotEzzaouia

I - Architecture et Choix Techniques

Diagramme de Classes



Les principales classes utilisées dans ce projet sont Deal, Facility, Part, Person, et Portfolio.

- **Deal** : Représente un accord financier avec des attributs tels que le numéro de contrat, l'agent, le pool de participants, le borrower, le montant, la monnaie, les dates de début et de fin, et le statut.

- **Facility** : Représente une ligne de crédit ou un prêt avec des informations sur les dates de début et de fin, le montant, la monnaie, et les lenders participants.
- **Part** : Représente une part ou un composant d'un deal avec des attributs comme le numéro de part et le prix.
- **Person** : Représente une entité, soit un borrower soit un lender, avec des attributs tels que le nom et le type (borrower ou lender).
- **Portfolio** : Gère un ensemble de parts.

Le polymorphisme est par exemple utilisé dans la classe `Person` pour permettre le traitement uniforme des prêteurs et des emprunteurs. Ainsi, lors de l'ajout ou de l'affichage de personnes, le type de personne (prêteur ou emprunteur) est déterminé dynamiquement.

II - Difficultés Rencontrées

Gestion des Entrées Utilisateur

Une des principales difficultés a été la gestion des entrées utilisateur pour assurer la robustesse et la validation des données. Nous avons implémenté des contrôles de saisie pour valider les entrées, notamment pour vérifier l'existence des emprunteurs lors de l'ajout de deals et pour valider les types de personnes.

Mise à Jour Dynamique

La mise à jour dynamique des informations, comme la modification des deals et des personnes, a présenté des défis en termes de recherche efficace et de gestion des erreurs. Nous avons implémenté des recherches linéaires et des mises à jour conditionnelles pour traiter ces aspects.

III - Fonctionnalités Développées

1. **Ajout de Deals** : Permet d'ajouter de nouveaux deals avec validation des emprunteurs existants.
2. **Affichage des Deals** : Affiche tous les deals avec leurs détails.
3. **Ajout de Prêteurs et Emprunteurs** : Permet d'ajouter de nouveaux prêteurs et emprunteurs avec validation des entrées.
4. **Affichage des Prêteurs et Emprunteurs** : Affiche les listes de prêteurs et emprunteurs.
5. **Mise à Jour des Deals** : Permet de modifier les détails des deals existants.
6. **Suppression de Deals** : Permet de supprimer des deals existants.
7. **Recherche de Deals par Emprunteur** : Permet de rechercher et d'afficher des deals spécifiques à un emprunteur.
8. **Mise à Jour des Informations de Personnes** : Permet de mettre à jour les montants associés aux prêteurs ou emprunteurs.

Malgré le grand nombre de fonctionnalités implémentés, nous aurions voulu en rajouter plus, comme par exemple une fonction de conversion des devises pour les facilités. Mais nous n’avons pas réussi à l’implémenter par manque de temps.

IV - Choix et Pistes Abandonnées

Tentative de Création d'une Interface Graphique Utilisateur (GUI)

Initialement, nous avions envisagé de créer une interface graphique utilisateur (GUI) pour améliorer l'expérience utilisateur et rendre l'application plus intuitive et accessible. Une GUI aurait permis aux utilisateurs de manipuler les deals, prêteurs, emprunteurs, facilités de crédit, et parts de manière plus visuelle et interactive, par rapport à une interface en ligne de commande.

Pour implémenter la GUI, nous avons exploré deux frameworks disponibles pour le développement d'interfaces graphiques en C++ :

1. **Qt** : Qt est un framework puissant et largement utilisé pour développer des applications graphiques en C++.
2. **wxWidgets** : wxWidgets est une autre bibliothèque populaire pour le développement de GUI en C++. Elle est multiplateforme et fournit une interface native sur chaque système d'exploitation.

Après avoir évalué ces options, nous avons décidé de nous concentrer sur Qt en raison de sa popularité, de sa riche documentation, et de sa communauté active. Cependant, nous avons dû faire face à deux défis techniques majeurs :

1. **Intégration avec le Code Existant** : L'intégration de Qt avec notre code existant a été difficile. Adapter notre modèle de données basé sur des vecteurs et des objets C++ pour qu'il soit manipulé par une interface graphique nécessitait une refonte significative de certaines parties du code, notamment celles des classes.
2. **Gestion des Événements** : La gestion des événements utilisateur dans une application graphique est plus complexe que dans une application en ligne de commande. Il fallait gérer les clics, les entrées de formulaire, les mises à jour dynamiques de l'interface, et les interactions entre les différents composants de l'application.

Après plusieurs jours de tentatives, nous avons conclu que le développement de la GUI nécessitait un investissement de temps et de ressources que nous ne pouvions pas nous permettre dans le cadre de ce projet. Nous avons donc décidé d'abandonner la création de la GUI pour nous concentrer sur le développement d'une interface en ligne de commande robuste et complète.

Cette décision a été prise pour assurer que toutes les fonctionnalités essentielles du projet soient correctement implémentées et testées, tout en respectant les délais. Nous avons, cela dit, beaucoup apprit concernant Qt, ce qui pourra servir de base pour une future extension du projet ou pour d'autres projets nécessitant une interface graphique.

V - Répartition du travail & Qualité du code

- **Victoire Cyrot** : Conception des classes et implémentation des fonctionnalités de base (ajout et affichage des deals, prêteurs, et emprunteurs).
- **Fares EZZAOUIA** : Développement des autres fonctionnalités (mise à jour et suppression des deals, recherche par emprunteur, mise à jour des informations des personnes) et gestion des contrôles de saisie.

Les bugs ont été gérés en utilisant des techniques de débogage standard telles que l'ajout de points d'arrêt et l'utilisation de messages de journalisation pour suivre l'exécution du code. Un bug majeur concernant la mise à jour incorrecte des montants a été résolu en ajoutant une validation supplémentaire et en retravaillant la logique de mise à jour.

La redondance a été évitée en utilisant des fonctions utilitaires pour les tâches courantes, telles que la validation des entrées et la recherche d'éléments dans les collections. L'utilisation de la généralisation dans la classe Person a également contribué à réduire la duplication de code.

Conclusion

Le projet a été une réussite, avec toutes les fonctionnalités prévues implémentées et testées. Les choix de conception, tels que l'utilisation du polymorphisme, ont permis de créer un code propre, modulaire et extensible. Les difficultés rencontrées ont été surmontées grâce à une approche systématique et collaborative. Ce projet a permis d'approfondir nos compétences en C++ et en conception logicielle.