Table des matières

Binôme : Yufei LIU & Giovanni AMOUSSOU

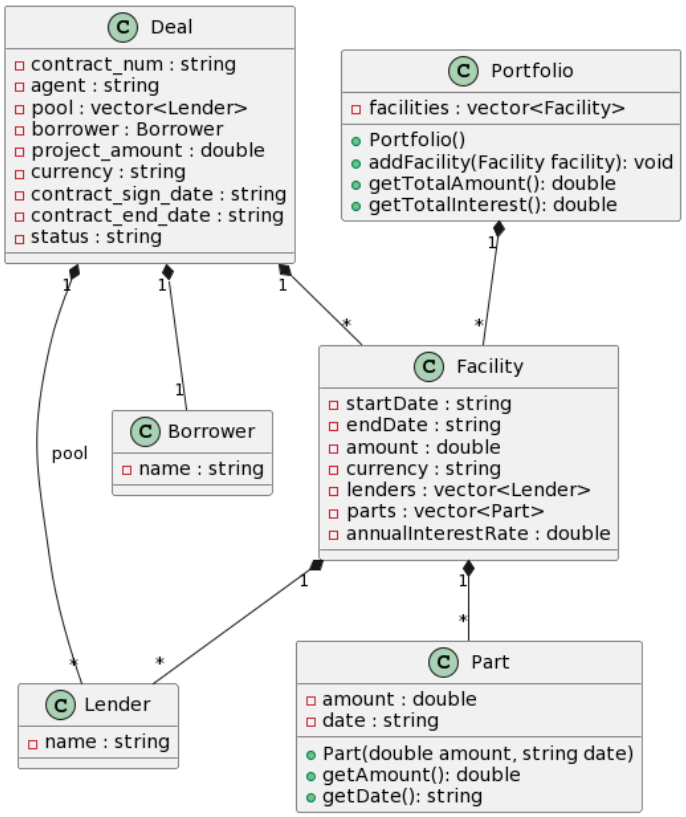
I. Introduction

Ce rapport présente le projet de financement structuré réalisé par notre binôme. L'objectif du projet était de concevoir et d'implémenter une solution logicielle pour la gestion de transactions de financement structuré. Nous avons développé une architecture basée sur des classes pour représenter les différents éléments impliqués dans le processus de financement structuré, tels que les prêts, les emprunteurs, les facilités de financement, les prêteurs, etc.

II. Architecture choisie

Notre architecture repose sur une conception orientée objet, utilisant des concepts tels que l'héritage, la composition et le polymorphisme. Nous avons identifié plusieurs classes clés pour modéliser les entités impliquées dans le financement structuré, notamment les classes Borrower, Deal, Facility, Lender, Part et Portfolio. Ces classes sont interconnectées pour représenter les relations et les flux de données entre les différentes entités.

Nous avons créé un diagramme de classes pour visualiser l'architecture du projet, montrant les classes, leurs attributs et leurs méthodes. Vous pouvez trouver le diagramme de classes dans le dossier du projet sous le nom "Diagramme de classes.png".



III. Choix de conception

1. Héritage et généralisation

Nous avons utilisé l'héritage et la généralisation pour modéliser les différents acteurs du financement structuré. Par exemple, les classes "Borrower" (emprunteur) et "Lender" (prêteur) héritent toutes deux de la classe "User" (utilisateur). Cela reflète le fait qu'un emprunteur et un prêteur sont tous deux des types d'utilisateurs, partageant certaines caractéristiques communes (comme un nom), mais ayant également leurs propres caractéristiques uniques (par exemple, un emprunteur pourrait prêter de l'argent dans un Deal, tandis qu'un prêteur pourrait être un Agent ou lender dans le Pool).

L'utilisation de l'héritage nous a permis de réutiliser du code commun aux emprunteurs et aux prêteurs, tout en nous permettant de définir des comportements spécifiques à chaque type d'utilisateur. Cela a favorisé la modularité et la maintenabilité de notre code.

2. Encapsulation

L'encapsulation est un autre concept clé de la programmation orientée objet que nous avons largement utilisé dans notre projet. En encapsulant les données (c'est-à-dire en rendant les variables membres privées) et en fournissant des accesseurs publics (getters) et des mutateurs (setters), nous avons pu contrôler l'accès aux données de nos objets et garantir que ces données restent dans un état valide.

Par exemple, dans notre classe "Deal", nous avons rendu l'attribut "projectAmount" privé et fourni des méthodes "getProjectAmount" et "setProjectAmount". Cela nous a permis de valider les nouvelles valeurs fournies à "setProjectAmount" avant de modifier l'état de l'objet, ce qui évite que l'objet ne se retrouve dans un état incohérent.

L'encapsulation nous a permis de construire des classes robustes qui résistent aux erreurs et facilitent le débogage.

IV. Difficultés rencontrées

Lors du développement du projet, nous avons rencontré plusieurs difficultés. L'une des principales difficultés était de comprendre en détail les spécifications du financement structuré et de traduire ces connaissances en une architecture logicielle cohérente. Nous avons dû effectuer une recherche approfondie et consulter des ressources spécialisées pour acquérir une compréhension solide du domaine.

Une autre difficulté était la complexité croissante du projet à mesure que nous ajoutions de nouvelles fonctionnalités et gérions les interactions entre les différentes classes. Nous avons dû maintenir une structure de code organisée, éviter la duplication de code et mettre en œuvre des méthodes et des classes modulaires pour faciliter la maintenance et les modifications ultérieures.

Nous avons aussi essayé de mettre en place la base de données MySQL, cependant nous avons eu un problème lié aux librairies que nous n'avons pas eu le temps de résoudre. Nous avons quand-même implémenter les tables. Ci-dessous vous avez une capture d'écran du message d'erreur.

V. Répartition du travail

Notre binôme a travaillé en étroite collaboration tout au long du projet, partageant les tâches et les responsabilités de manière équilibrée. Nous avons commencé par discuter ensemble des spécifications et des exigences du financement structuré, en identifiant les classes et les fonctionnalités clés.

Ensuite, nous avons réparti le travail en fonction de nos compétences et de nos intérêts. Chacun d'entre nous a pris en charge la mise en œuvre de certaines classes et fonctionnalités, en se référant aux spécifications et en respectant les conventions de nommage et les bonnes pratiques de programmation.

Nous avons maintenu une communication régulière pour discuter des problèmes rencontrés, partager nos idées et harmoniser le projet dans son ensemble. Les décisions importantes concernant l'architecture et les choix de conception ont été prises en collaboration, en tenant compte des avantages et des inconvénients de chaque approche.

VI. Fonctionnalités implémentées et non implémentées

Nous avons réussi à implémenter avec succès les fonctionnalités essentielles du financement structuré dans notre projet. Cela comprend :

- **Créer un Deal**
- **Créer un Borrower**
- **Créer des Lenders**
- **Ajouter des Lenders à un Pool** : Cette fonctionnalité permettra à l'utilisateur d'ajouter un ou plusieurs prêteurs à un pool de prêteurs pour un contrat de financement spécifique.
- **Créer un Facility** : Cette fonctionnalité permettra à l'utilisateur de créer une nouvelle tranche de financement (Facility) pour un contrat de financement spécifique. L'utilisateur sera en mesure d'entrer des détails tels que la date de début, la date de fin, le montant, la devise et les prêteurs qui participent à la tranche.
- **Rembourser une Part d'un Facility** : Cette fonctionnalité permettra à l'emprunteur de rembourser une part spécifique d'une tranche de financement. En fonction de la quantité remboursée, le montant de la tranche sera ajusté et l'intérêt calculé.
- **Calculer l'intérêt** : Il s'agit d'une fonctionnalité qui permettra de calculer l'intérêt dû sur une tranche de financement en fonction du taux d'intérêt annuel et de la période.
- **Gérer le Portfolio** : Cette fonctionnalité permettra à l'utilisateur de voir tous les intérêts accumulés sur un portefeuille pour un contrat de financement spécifique.
- **Terminer un Deal** : Cette fonctionnalité permettra de marquer un contrat de financement comme terminé une fois que tout l'argent a été utilisé et remboursé.
- **Afficher les détails d'un Deal** : Cette fonctionnalité permettra à l'utilisateur de voir tous les détails d'un contrat de financement spécifique, y compris les informations sur l'emprunteur, les prêteurs, les tranches de financement, etc.

Cependant, en raison de contraintes de temps et de priorités, certaines fonctionnalités n'ont pas été implémentées, notamment :

- Les fonctionnalités avancées de filtrage et de recherche dans le portefeuille de financement.
- La persistance des données dans une base de données (Mysql).

Ces fonctionnalités n'étaient pas essentielles pour l'objectif principal du projet, mais pourraient être ajoutées ultérieurement pour améliorer la solution.

VII. Conclusion

Notre projet de financement structuré a été une expérience enrichissante, nous permettant d'appliquer nos connaissances en programmation orientée objet et de relever les défis de conception et d'implémentation d'un système complexe. Nous avons acquis une bonne compréhension du financement structuré et de ses exigences spécifiques.

Malgré les difficultés rencontrées, notre binôme a travaillé efficacement en équipe, en partageant les responsabilités et en collaborant étroitement pour atteindre les objectifs du projet. Nous sommes fiers du résultat obtenu et de notre capacité à concevoir une architecture logicielle flexible et modulaire pour la gestion du financement structuré.

Ce projet a renforcé notre compréhension des bonnes pratiques de programmation, de la réflexion et de l'attention aux détails nécessaires pour développer un code propre, maintenable et de qualité.