Rapport de Projet : Système de Financement Structuré

Master 2 MIAGE IF Apprentissage 2024/2025
Projet C++

DOUBABI Mustapha / HADDAD Chirine

Table des Matières

Table des matières

---

1. Introduction

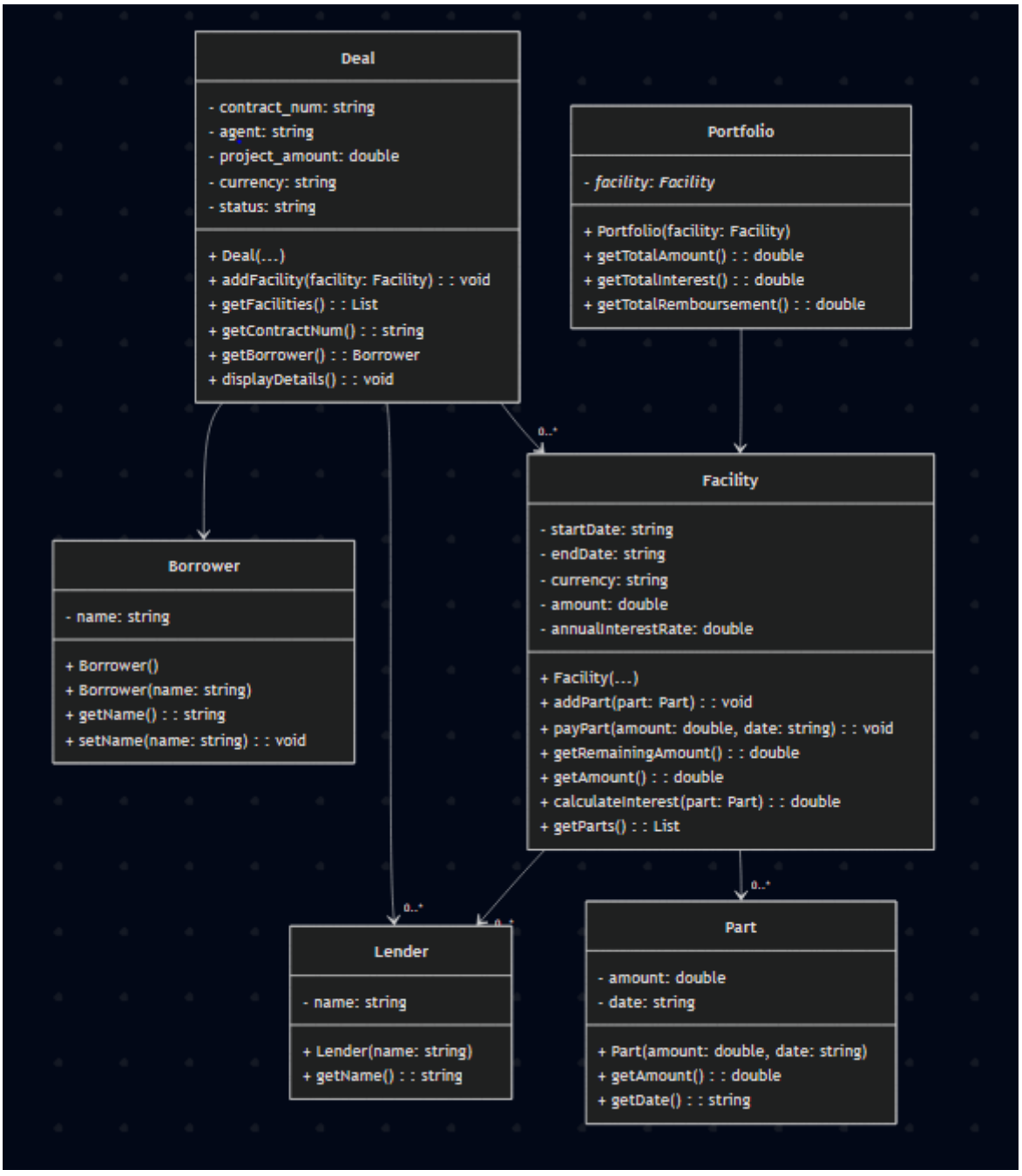
Le présent projet consiste en l'implémentation d'un système de gestion de financement structuré en C++. Dans le contexte bancaire moderne, lorsqu'une institution financière ne peut assumer seule le risque d'un projet de grande envergure, elle fait appel à un consortium de banques pour partager ce financement. Notre système modélise cette réalité complexe où une banque agent coordonne un pool de prêteurs pour financer les besoins d'un emprunteur.

Le défi technique résidait dans la création d'un modèle objet robuste, capable de gérer les relations complexes entre les différents acteurs financiers, tout en assurant l'intégrité des données et la cohérence des opérations financières. L'objectif était de produire un code maintenable, extensible et conforme aux standards de qualité industriels.

2. Architecture et Conception

Diagramme de Classes

Notre architecture s'articule autour de six classes principales, chacune ayant une responsabilité bien définie dans l'écosystème du financement structuré :



Principe de Conception Adopté

L'architecture repose sur le **principe de responsabilité unique** (Single Responsibility Principle). Chaque classe encapsule un concept métier distinct :

- **Deal** : Représente le contrat principal et orchestre l'ensemble du financement
- **Facility** : Modélise une tranche de financement avec ses spécificités
- **Part** : Encapsule un remboursement individuel avec validation temporelle
- **Portfolio** : Fournit une vue consolidée des calculs financiers
- **Borrower/Lender** : Représentent les acteurs financiers

Cette séparation facilite la maintenance et permet une évolution indépendante de chaque composant.

---

3. Justification des Choix Techniques

Absence d'Héritage Complexe

Contrairement à une approche traditionnelle qui aurait pu introduire une hiérarchie d'acteurs financiers (par exemple, une classe Actor dont hériteraient Borrower et Lender), nous avons opté pour des classes indépendantes. Cette décision se justifie par :

1. **Simplicité conceptuelle** : Borrower et Lender ont des comportements fondamentalement différents
2. **Évitement du couplage** : Pas de dépendances héritées complexes
3. **Facilité de test** : Chaque classe peut être testée isolément

Polymorphisme par Composition

Le polymorphisme est obtenu par composition plutôt que par héritage. Par exemple, la classe Deal contient une collection de Facility, permettant différents types de tranches sans hiérarchie complexe. Cette approche offre plus de flexibilité pour l'évolution future du système.

Encapsulation Stricte

Tous les attributs sont déclarés privés, l'accès se faisant exclusivement par des méthodes publiques. Cette décision garantit :

- **Contrôle d'intégrité** : Validation systématique des données
- **Évolutivité** : Possibilité de modifier l'implémentation interne sans impact sur les clients
- **Debugging facilité** : Points de contrôle centralisés

Gestion d'Erreurs par Exceptions

L'utilisation systématique des exceptions C++ (std::invalid_argument, std::runtime_error) permet une gestion d'erreurs robuste et explicite, abandonnant les codes de retour traditionnels au profit d'une approche plus moderne et lisible.

---

4. Difficultés Rencontrées

Dépendances Circulaires

Problème identifié : Les premières versions du code présentaient des dépendances circulaires entre Facility.h et Part.h, causant des erreurs de compilation.

Solution adoptée : Réorganisation des inclusions et utilisation de déclarations anticipées (forward declarations) quand nécessaire. Suppression des inclusions inutiles dans les fichiers header.

Leçon apprise : L'importance d'une analyse préalable des dépendances lors de la conception.

Double Gestion des Intérêts

Problème identifié : La classe Portfolio comportait initialement deux mécanismes de calcul d'intérêts : un automatique basé sur les Part et un manuel via addInterest(). Cette redondance créait des incohérences.

Solution adoptée : Suppression du mécanisme manuel et centralisation du calcul automatique. Cette approche garantit la cohérence et élimine les sources d'erreur.

Impact : Simplification de l'API et fiabilité accrue des calculs financiers.

Validation des Formats de Données

Défi technique : Assurer la cohérence des formats de dates et numéros de contrat selon les spécifications métier.

Approche retenue : Implémentation de méthodes de validation privées avec parsing strict et levée d'exceptions explicites. Par exemple, validation du format S1234 pour les numéros de contrat.

Performance et Gestion Mémoire

Optimisation réalisée : Passage systématique par référence constante (const std::string&) pour éviter les copies coûteuses. Utilisation de pointeurs pour Portfolio afin d'éviter la duplication de Facility.

5. Fonctionnalités Implémentées

Gestion Complète des Deals

- Création avec validation complète des paramètres
- Gestion des statuts (TERMINATED, CLOSED) via énumération type-safe
- Ajout contrôlé de facilities avec vérification des montants
- Affichage détaillé avec formatage professionnel

Système de Facilities Robuste

- Support multi-devise avec validation
- Calcul d'intérêts composés avec gestion temporelle précise
- Gestion des remboursements avec contrôle d'intégrité
- Suivi du montant restant en temps réel

Portfolio Intelligent

- Calculs financiers automatisés et cohérents
- Vue consolidée des intérêts et remboursements
- Synchronisation automatique avec les modifications de facility

Interface Utilisateur Complète

- Menu interactif avec gestion d'erreurs
- Validation des saisies utilisateur
- Messages d'erreur explicites et actions correctives

Persistance des Données

Structure préparée pour l'intégration MySQL avec méthodes CRUD commentées, permettant une évolution future vers une base de données sans refactoring majeur.

6. Fonctionnalités Non Implémentées

Intégration Base de Données

Statut : Non implémentée

Calculs Financiers Avancés

Éléments manquants : Amortissement, calculs actuariels, courbes de taux

Interface Graphique

Statut : Interface console uniquement

Justification : Concentration sur la logique métier et l'architecture. Une GUI pourrait être ajoutée en couche supérieure sans impacter le cœur fonctionnel.

---

7. Répartition du travail

Notre binôme a collaboré étroitement tout au long du projet, en répartissant équitablement les tâches et responsabilités. Nous avons d'abord défini ensemble les spécifications du financement structuré, en identifiant les classes et fonctionnalités essentielles.
Le travail a ensuite été réparti selon nos compétences respectives. Chacun a développé certaines parties du code en suivant les bonnes pratiques et les conventions établies. Des échanges réguliers nous ont permis de résoudre les difficultés, d'harmoniser notre approche et de prendre ensemble les décisions clés en matière d'architecture et de conception.

---

8. Conclusion

Ce projet de financement structuré a été une expérience formatrice, nous permettant de mobiliser nos compétences en programmation orientée objet et de relever les défis liés à la conception d'un système complexe.
Nous avons consolidé notre compréhension du financement structuré et de ses spécificités.
Malgré les obstacles, notre binôme a su collaborer efficacement, en partageant les tâches pour atteindre les objectifs fixés. Nous sommes satisfaits du résultat et de l'architecture logicielle souple et modulaire que nous avons conçue.