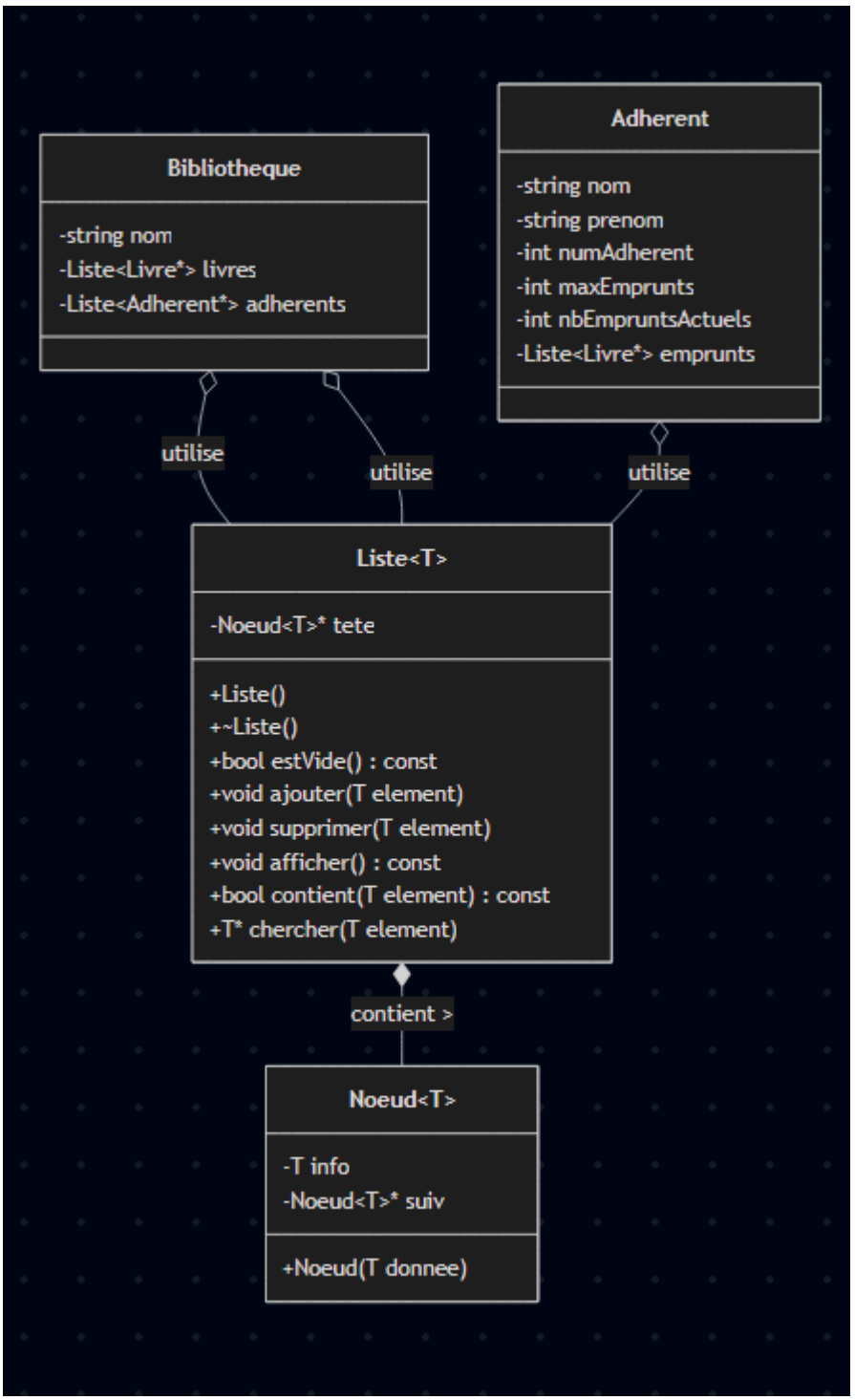
Rapport de Projet Electif POO : Gestion d’une Bibliothèque

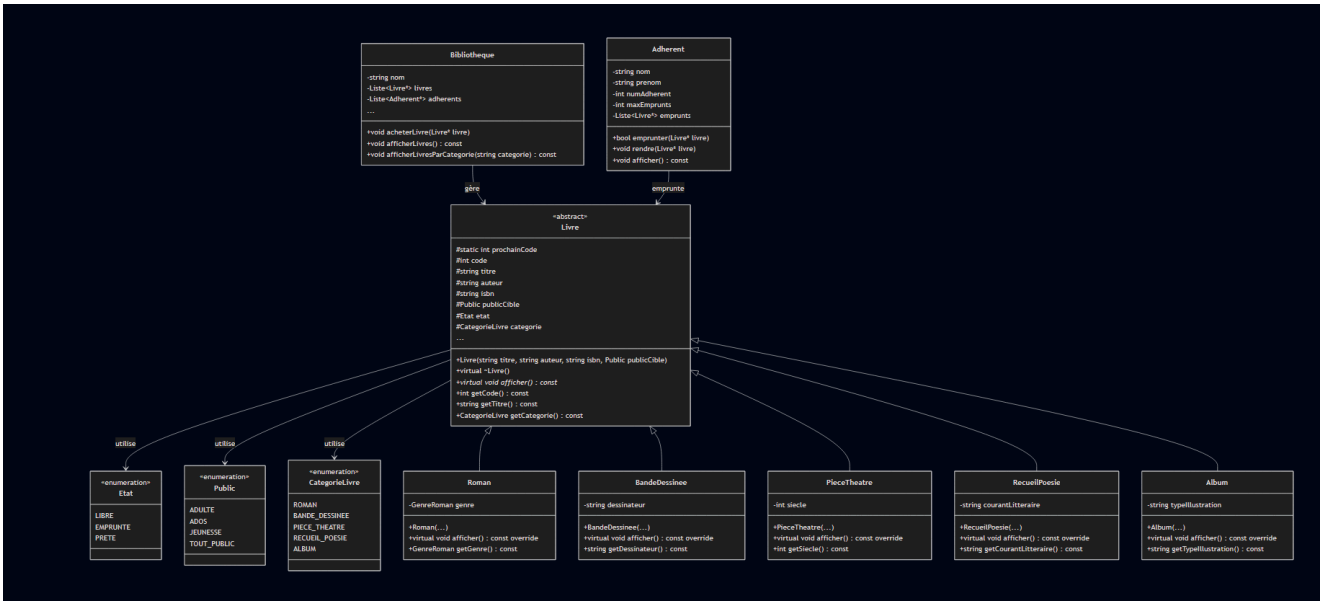
Alexandre JUGE
Louis JADOT

1. Introduction

Ce projet consiste en la réalisation d'une application de gestion de réseau de bibliothèques. L'objectif était de modéliser les interactions entre une bibliothèque, ses adhérents et son stock de livres. La contrainte technique majeure — l'interdiction d'utiliser la bibliothèque standard (STL) — nous a conduits à implémenter nos propres structures de données (listes chaînées templates).

2. Diagrammes UML


Bibliotheque
-string nom
-Liste<Livre*> livres
-Liste<Adherent*> adherents
Adherent
-string nom
-string prenom
-int numAdherent
-int maxEmprunts
-int nbEmpruntsActuels
-Liste<Livre*> emprunts
utilise
utilise
utilise
Liste<T>
-Noeud<T>* tete
+Liste()
+~Liste()
+bool estVide() : const
+void ajouter(T element)
+void supprimer(T element)
+void afficher() : const
+bool contient(T element) : const
+T* chercher(T element)
contient >
Noeud<T>
-T info
-Noeud<T>* suiv
+Noeud(T donnee)



3. Description des Classes

3.1. Structures de Données

- **`Liste<T>` (Template)** : Classe générique gérant une liste chaînée simple. Elle remplace `std::vector`.
    - *Rôle :* Gérer l'ajout, la suppression et le parcours d'éléments de n'importe quel type.
    - *Particularité :* Gestion manuelle de la mémoire (new/delete) et amitié avec `Bibliotheque` pour faciliter les recherches.
- **`Noeud<T>`** : Classe utilitaire représentant un maillon de la liste.

3.2. Classes principales

- **`Bibliotheque`**
    - *Attributs :* Une liste de livres (le catalogue) et une liste d'adhérents.
    - *Méthodes clés :* `acheterLivre()`, `inscrireAdherent()`, `echangerLivre()`.
- **`Adherent`**
    - *Rôle :* Porter la logique d'emprunt (vérification des quotas).
    - *Attributs :* Identité, quota max, et une `Liste<Livre*>` représentant ses emprunts en cours.

3.3. Héritage

- **`Livre` (Abstraite)** :
- **Classes Filles (`Roman`, `BD`, `PieceTheatre`, `Album`, `Poesie`)** :

- Elles héritent de `Livre` et ajoutent leurs spécificités (ex: `Dessinateur` pour la BD, `Genre` pour le Roman).
- Elles implémentent chacune leur propre version de `afficher()` qui est virtuelle pure dans Livre.

4. Choix Techniques & Justifications

4.1. Pourquoi une Liste Chaînée Manuelle ?

Nous avons choisi l'implémentation par liste chaînée pour sa flexibilité lors des insertions/suppressions dynamiques. Le choix du **Template** s'est imposé pour éviter la duplication de code entre la liste des adhérents et celle des livres.

4.2. Gestion de la Mémoire

L'absence de "Smart Pointers" (interdits par la consigne implicite de niveau) nous oblige à une rigueur totale.

Les livres sont alloués dynamiquement ou sur la pile (dans le main).

Le destructeur de `Liste` (`~Liste`) se charge de nettoyer les maillons (`Noeuds`) pour éviter les fuites de mémoire.

4.3. Enums

Pour l'état des livres (`LIBRE`, `EMPRUNTE`) et les catégories, nous utilisons des **`enum`**. Cela garantit qu'aucun état invalide ne peut être saisi par erreur dans le code, contrairement à l'utilisation de chaînes de caractères (`string`).

5. Fonctionnalités Implémentées

Le programme final permet de :

1. Créer une bibliothèque et un stock hétérogène (Romans, BDs...).
2. Gérer les emprunts avec contrôle de règles (Quotas, Livre déjà pris).
3. Simuler un échange de livre entre deux bibliothèques.
4. Filtrer l'affichage par catégorie ou afficher tout le catalogue.
5. Gérer les erreurs via des **Exceptions** (`try/catch`) pour ne pas faire planter le programme.

6. Difficultés & Conclusion

La principale difficulté a été la gestion des pointeurs dans la méthode `supprimer()` de la liste générique (recollage des maillons `precedent->suiv = courant->suiv`). Les

inclusions circulaires entre `Adherent` et `Bibliotheque` ont aussi été résolues grâce à la déclaration anticipée.