Rapport Projet Bibliothèque C++

Par DJERIDI Rania, AMIRI Afek, AMAKRANE Ghada et CARLIN Esteban

Introduction

Nous avons entrepris la gestion complète d'un réseau de bibliothèques, couvrant la création, les acquisitions de livres, jusqu'aux emprunts des adhérents. À cette fin, nous allons mettre en œuvre diverses classes dotées de méthodes facilitant sa gestion. Un schéma UML a été élaboré pour une meilleure visualisation de notre travail.

Pour construire notre bibliothèque, nous avons suivi les étapes suivantes :

1) Création de la classe Livre : Une classe générale, à partir de laquelle d'autres classes dérivent : Pièce_Théâtre, BD, Recueil_poèmes, Roman et Album.
2) Création de la classe Noeud : Cette classe est utilisée pour construire la liste chaînée de ListeLivres.
3) Mise en place de la classe ListeLivres : Cette classe utilise une liste chaînée pour stocker les livres.
4) Création de la classe Bibliothèque : Cette classe vise à caractériser chaque bibliothèque.
5) Introduction de la classe Adhérent : Cette classe caractérise la personne qui empruntera le livre.

6) Création de la classe NotFoundException pour gérer l'exception "Livre non trouvé" dans les fonctions "Bibliotheque::supprimeLivre(Livre*)" et "Bibliotheque::demanderLivreAutreBib(String, Bibliotheque)".

(Malheureusement, à cause des problèmes techniques on a supprimé cette classe, vous trouverez ci-dessous le code correspondant)

```
#ifndef NOTFOUNDEXCEPTION_H_

#define NOTFOUNDEXCEPTION_H_



#include <stdexcept>

#include <string>



class NotFoundException : public std::exception {

public:

   const char* what() const noexcept override {

      return "Livre non trouve dans la bibliotheque.";

   }

};



#endif /* NOTFOUNDEXCEPTION_H_ */
```

Il existe plusieurs relations entre ces classes. Par exemple, LivresEmpruntes de la classe Adhérent est de type Livre. La bibliothèque d'origine de la classe Livre est de type Bibliothèque, et la listeLivres de Bibliothèque est de type ListeLivres. Les

pointeurs *tete et *suivant de ListeLivres sont de type Noeud.Les classes BD, Piece_Theatre, Recueil_poemes et Album héritent de la classe Livre.

Choix de la liste chaînée dans la classe Bibliothèque

Initialement, nous avons opté pour une liste standard pour stocker les livres dans notre bibliothèque. Cependant, cette approche a présenté quelques limitations, notamment en ce qui concerne la gestion dynamique des livres empruntés et des retours. Afin de remédier à ces problèmes et d'optimiser l'efficacité de notre système, nous avons choisi de migrer vers une structure de liste chaînée. Cette transition nous offre une flexibilité accrue pour l'ajout, la suppression et la recherche des livres, tout en facilitant la gestion spécifique des livres empruntés par chaque adhérent. La liste chaînée s'est avérée être une solution plus adaptée à nos besoins, offrant une meilleure optimisation de l'espace mémoire et une gestion plus efficace des opérations liées aux emprunts de livres.

Gestion d'exceptions

On a utilisé la gestion d'exception dans la classe Bibliotheque, en utilisant l'exception standard out-of-range de C++ dans la fonction Bibliotheque::ajouterLivre(Livre* nouveauLivre) et avec la classe NotFoundException qui hérite de la Classe exception comme expliqué précédemment.

Surcharge d'opérateurs

On a surchargé l'opérateur + dans la classe ListeLivres pour ajouter un livre à la liste. On l'a utilisé comme méthode de classe.

Fonctions amies

On a défini certaines méthodes de la classe Bibliothèque comme amies dans la classe Livre pour pouvoir accéder aux attributs privés facilement.

```
friend void Bibliotheque::ajouterLivre(Livre* );
friend void Bibliotheque::demanderLivreAutreBib(string, Bibliotheque);
friend void Bibliotheque::supprimerLivre(string);
friend void Bibliotheque::afficherLivres();
friend void Bibliotheque::acheterLivre(Livre*);
friend void Bibliotheque::RendreLivre(Livre*);
```

Explication des classes

1- La classe Bibliotheque:

La classe Bibliothèque est conçue pour représenter une bibliothèque et stocker des livres. Voici une amélioration de la présentation avec quelques précisions pour faciliter la compréhension:

Attributs :

nom: Nom de la bibliothèque.

adresse: Adresse de la bibliothèque.

code: Code d'identification unique de la bibliothèque.

listeLivres: Liste chaînée de livres pour stocker les livres de la bibliothèque (de type ListeLivres).

capaciteMax: Capacité maximale de la bibliothèque (par défaut 100).

nombreLivres: Nombre de livres disponibles dans la bibliothèque (par défaut 0).

Méthodes :

Bibliothque(): Constructeur par défaut.

Bibliotheque(string nom, string adresse, string code): Constructeur prenant comme paramètres le nom, l'adresse et le code de la bibliothèque.

ajouterLivre(Livre* nouveauLivre): Ajoute un livre à la liste chaînée des livres, sous réserve que la capacité maximale ne soit pas dépassée.

supprimerLivre(string codeLivre): Supprime un livre de la bibliothèque en fonction de son code.

afficherLivres(): Affiche tous les livres disponibles dans la bibliothèque.

acheterLivre(Livre* nouveauLivre): La bibliothèque achète un livre.

Dans le setter de livres, on vérifie que la liste de livres proposée est de bonne taille. Si sa taille dépasse la capacité maximale, on en prends que les premiers éléments.

Dans la méthode Bibliotheque::demanderLivreAutreBib(string isbn, Bibliotheque autreBibliotheque)
On a choisi de garder le livre dans sa bibliothèque d'origine et modifier son état à prêté. On a créer une copie dans la bibliothèque qui l'a emprunté ayant comme état "emprunté". Cette copie sera supprimée lorsque le livre est rendu à sa bibliothèque d'origine.

2- La classe Livre:

Un livre est caractérisé par plusieurs attributs essentiels qui définissent son identité au sein de notre bibliothèque.

Attributs :

ISBN : Code unique servant à identifier un livre.

code : Identifiant propre à chaque exemplaire du livre, permettant une distinction au sein d'une même bibliothèque.

auteur : Information sur l'auteur du livre.

titre : Titre du livre.

éditeur : Information sur l'éditeur du livre.

publicCible : Catégorie d'âge à laquelle le livre est destiné, avec les valeurs "00" pour tout le monde et "18" pour les lecteurs de plus de 18 ans.

état : Indication sur l'état du livre, pouvant être "libre" ou "emprunte" ou "prete".

bibliothequeOrigine : Bibliothèque d'origine du livre, indiquant où le livre a été initialement ajouté au système.

La raison de l'utilisation du type protected pour la plupart de ces attributs est de favoriser l'encapsulation des données au sein de la classe Livre et de ses classes dérivées, assurant ainsi une gestion contrôlée des informations liées à chaque livre. Cela permet également de garantir une cohérence interne dans la manipulation de ces données au sein de la hiérarchie de classes.

Cependant, l'attribut bibliothequeOrigine est déclaré comme public Bibliothèque. Cette décision découle du besoin d'accéder directement à cette information depuis l'extérieur de la classe Livre. En rendant cet attribut public, nous offrons la possibilité

d'obtenir des détails sur la bibliothèque d'origine sans passer par des méthodes spécifiques, simplifiant ainsi l'accès à cette information cruciale depuis d'autres parties du programme.

Méthodes :

Livre() :constructeur par défaut

Livre(int, string, string, string, string,string): constructeur

Livre(Livre*): constructeur de recopie

~Livre: déconstructeur

On a essayé d'optimiser les getters et les setters et on essayé de coder que celles qu'on a besoin

3- Sous-classes de Livre

BD (Bandes dessinées) :

La BD est une catégorie spécifique de livre qui se distingue par la présence d'un dessinateur en plus des caractéristiques standard d'un livre.

Pour les méthodes, nous avons mis en place deux constructeurs :

- BD(Livre*, string) : Prend en arguments le livre et le dessinateur.
- BD(int, string, string, string, string, string, string) : Un constructeur alternatif prenant les attributs du livre ainsi que l'information sur le dessinateur. L'attribut du dessinateur est déclaré comme private.

Pièce de théâtre :

La pièce de théâtre est une autre catégorie de livre avec une caractéristique unique, le siècle (de type string).

Pour les méthodes, nous avons mis en place deux constructeurs :

- Pièce_théatre(Livre*, string) : Prend en arguments le livre et le siècle.
- Pièce_théatre(int, string, string, string, string, string, string) : Un constructeur alternatif prenant les attributs du livre ainsi que l'information sur le siècle. L'attribut du siècle est déclaré comme private.

Roman :

Le roman est une catégorie de livre avec une caractéristique supplémentaire, le genre (de type string).

Pour les méthodes, nous avons mis en place deux constructeurs :

- Roman(Livre*, string) : Prend en arguments le livre et le genre.
- Roman(int, string, string, string, string, string, string) : Un constructeur alternatif prenant les attributs du livre ainsi que l'information sur le genre. L'attribut du genre est déclaré comme private.

Album :

L'album est une catégorie de livre avec une caractéristique distincte, l'illustration (de type string).

Pour les méthodes, nous avons mis en place deux constructeurs :

- Album(Livre*, string) : Prend en arguments le livre et l'illustration.
- Album(int, string, string, string, string, string, string) : Un constructeur alternatif prenant les attributs du livre ainsi que l'information sur l'illustration. L'attribut de l'illustration est déclaré comme private.

Recueil de poèmes :

Le recueil de poèmes est une catégorie de livre avec une caractéristique particulière, l'indicateur (de type string).

Pour les méthodes, nous avons mis en place deux constructeurs :

- Recueil_poèmes(Livre*, string) : Prend en arguments le livre et l'indicateur.
- Recueil_poèmes(int, string, string, string, string, string, string) : Un constructeur alternatif prenant les attributs du livre ainsi que l'information sur l'indicateur. L'attribut de l'indicateur est déclaré comme private.

L'utilisation de private pour les attributs dans les sous-classes (BD, Pièce_théâtre, Roman, Album, Recueil_poèmes) assure une encapsulation, limitant l'accès direct aux attributs, favorisant la modularité et préservant l'intégrité des données, tout en permettant un contrôle d'accès via des méthodes publiques (getters et setters).

4- La classe Adhérent

Un Adhérent est défini par :

Attributs :

nom : le nom de l'adhérent.

prénom : le prénom de l'adhérent.

adresse : l'adresse de l'adhérent.

age: l'âge de l'adhérent

numéroAdhérent : un code unique permettant d'identifier chaque adhérent.

nombreLivresEmprunts : le nombre de livres actuellement empruntés par l'adhérent (variable entre 0 et capaciteMaxLivresEmprunts).(par défaut c'est 0)

capaciteMaxLivresEmprunts : la capacité maximale de livres qu'un adhérent peut emprunter (par défaut, c'est 10).

livresEmprunts : une liste pour stocker les livres empruntés par l'adhérent.

L'utilisation de private pour ces attributs garantit l'encapsulation, limitant l'accès direct à ces détails d'identité depuis l'extérieur de la classe. Cela favorise une gestion sécurisée des informations liées à l'adhérent en permettant un contrôle strict sur l'accès et la modification de ces attributs via des méthodes publiques (getters et setters).

Méthodes :

Adherent(): constructeur par défaut.

Adherent(string, string, string, int, int): constructeur prenant en paramètre le nom, le prénom, l'adresse, le numéro d'adhérent et la capacité maximale d'emprunts.

~Adherent(): destructeur.

rendreLivre(string, string): permet à l'adhérent de rendre un exemplaire à la bibliothèque en utilisant le code ISBN (identifiant du livre) et le code (identifiant de l'exemplaire). Cette fonction évite qu'un adhérent emprunte deux exemplaires du même

livre et rende un exemplaire au lieu de l'autre. Une fois l'exemplaire rendu, son état est changé à "libre" pour que d'autres adhérents puissent l'emprunter.

emprunterLivre(Livre): permet à l'adhérent d'emprunter un livre s'il est disponible.

afficherInfosAdherent(const Adherent&): affiche toutes les informations sur l'adhérent.

operator%(string): prend en paramètre un ISBN et retourne vrai (true) si l'adhérent a emprunté le livre correspondant et si ce livre est en sa possession, sinon retourne faux (false).

5- La classe Noeud

Attributs :

Livre* info : pointeur vers un livre.
Noeud* suivant : pointeur vers le prochain noeud dans la liste.
Méthodes :
Noeud() : constructeur par défaut
Noeud(Livre*) : constructeur avec attribut info

6- La classe ListeLivres

Attributs :

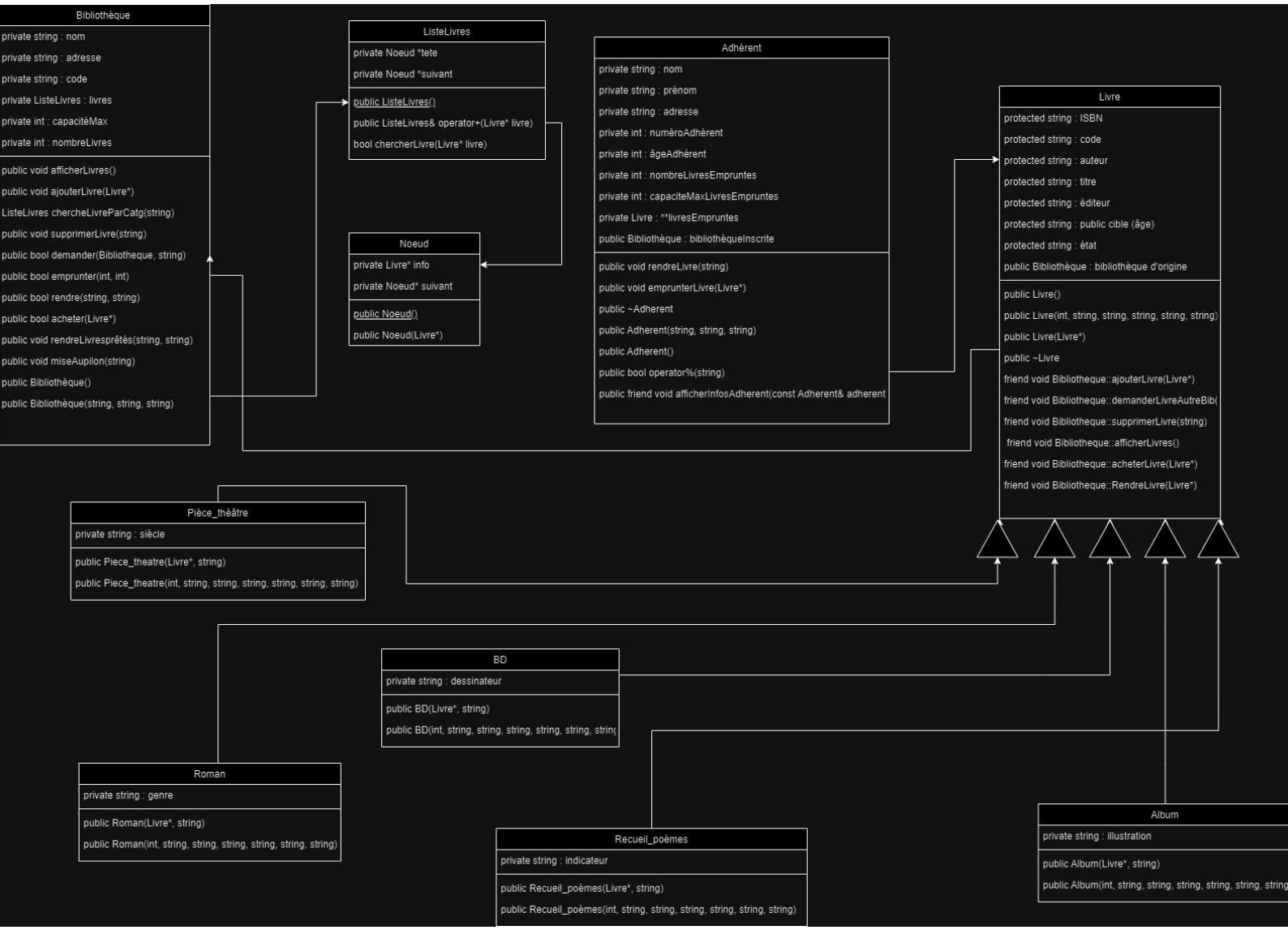
taille : taille de la liste
Noeud* tete : pointeur vers la tête.

Méthodes:

ListeLivres() : constructeur par défaut
ListeLivres& operator+(Livre* livre) : ajouter un livre
bool chercherLivre(Livre* livre) : chercher un livre

Diagramme UML

Par souci de simplification, on a pas inclus la classe NotFoundException dans notre diagramme UML.


Bibliothèque
private string : nom
private string : adresse
private string : code
private ListeLivres : livres
private int : capacitéMax
private int : nombreLivres
public void afficherLivres()
public void ajouterLivre(Livre*)
ListeLivres chercheLivreParCatg(string)
public void supprimerLivre(string)
public bool demander(Bibliotheque, string)
public bool emprunter(int, int)
public bool rendre(string, string)
public bool acheter(Livre*)
public void rendreLivresprêtés(string, string)
public void miseAupilon(string)
public Bibliothèque()
public Bibliothèque(string, string, string)
ListeLivres
private Noeud *tete
private Noeud *suivant
public ListeLivres()
public ListeLivres& operator+(Livre* livre)
bool chercherLivre(Livre* livre)
Noeud
private Livre* info
private Noeud* suivant
public Noeud()
public Noeud(Livre*)
Adhérent
private string : nom
private string : prénom
private string : adresse
private int : numéroAdhérent
private int : âgeAdhérent
private int : nombreLivresEmpruntes
private int : capaciteMaxLivresEmpruntes
private Livre : **livresEmpruntes
public Bibliothèque : bibliothèqueInscrite
public void rendreLivre(string)
public void emprunterLivre(Livre*)
public ~Adherent
public Adherent(string, string, string)
public Adherent()
public bool operator%(string)
public friend void afficherInfosAdherent(const Adherent& adherent
Livre
protected string : ISBN
protected string : code
protected string : auteur
protected string : titre
protected string : éditeur
protected string : public cible (âge)
protected string : état
public Bibliothèque : bibliothèque d'origine
public Livre()
public Livre(int, string, string, string, string, string)
public Livre(Livre*)
public ~Livre
friend void Bibliotheque::ajouterLivre(Livre*)
friend void Bibliotheque::demanderLivreAutreBib(
friend void Bibliotheque::supprimerLivre(string)
friend void Bibliotheque::afficherLivres()
friend void Bibliotheque::acheterLivre(Livre*)
friend void Bibliotheque::RendreLivre(Livre*)
Pièce_théâtre
private string : siècle
public Piece_theatre(Livre*, string)
public Piece_theatre(int, string, string, string, string, string, string)
BD
private string : dessinateur
public BD(Livre*, string)
public BD(int, string, string, string, string, string, strin
Roman
private string : genre
public Roman(Livre*, string)
public Roman(int, string, string, string, string, string, string)
Recueil_poèmes
private string : indicateur
public Recueil_poèmes(Livre*, string)
public Recueil_poèmes(int, string, string, string, string, string)
Album
private string : illustration
public Album(Livre*, string)
public Album(int, string, string, string, string, string, string