POO

Compte Rendu

ITC313_TP1

Victor FLATTOT

Encadré par

GINHAC Dominique
TEL Steven
NKONDJOCK Waytehad

06/01/2022

Table des matières

Introduction

Ce TP concerne la gestion d'une bibliothèque. Cette bibliothèque comporte des livres qui peuvent être empruntés par des lecteurs en respectant la règle suivante : « Un Livre peut être emprunté par un seul Lecteur à un instant donné et doit donc être rendu à la bibliothèque par ce Lecteur avant d'être à nouveau emprunté par le même lecteur ou par un autre lecteur ».

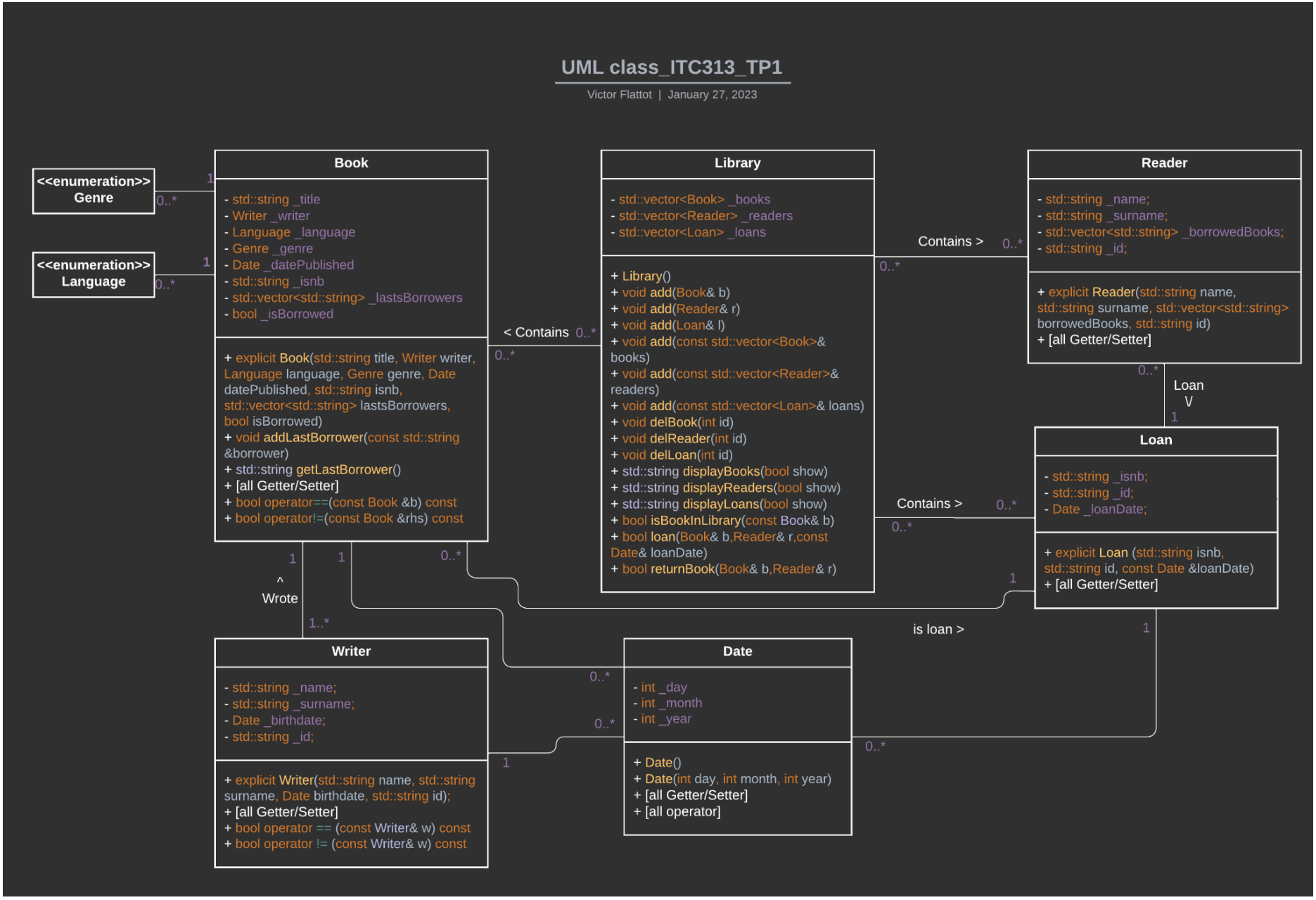
Pourquoi ce TP ?

J'ai déjà des notions de programmation dans d'autre langage de programmation orientée objet tel que le Java, Python ou encore le VB Net/C#. Mais je n'avais jamais fait de C++ et afin d'avoir de bonnes bases et de prendre de bonnes habitudes de programmation pour ce nouveau langage, j'ai choisi de commencer par le TP de plus bas niveau afin de me concentrer plus sur la syntaxe et les bonnes pratiques du C++.

Organisation des classes C++

Avant de commencer à coder j'ai analysé comment les classes interagissent entre elles. Cela m'a permis d'avoir une vue d'ensemble pour le TP. Je me suis donc rendu compte que la classe principale serait *Library* car en effet elle centralise les *Book*, les *Reader* ainsi que les *Loan*. Les *Loan* sont également très importants étant donné que ce sont ceux-ci que nous voulons manipuler et gérer de la meilleure façon possible. J'ai décidé que la gestion des *Loan* sera faite directement dans la classe *Library*. En effet, pour garder une trace de quel livre à été emprunté et par qui, nous avons besoin d'avoir accès à ces classes.

Le diagramme UML suivant est un bon résumé de l'organisation de ces classes de comment elles interagissent entre elles.


UML class_ITC313_TP1
Victor Flattot | January 27, 2023
<<enumeration>>
Genre
<<enumeration>>
Language
1
0..*
1
0..*
Book
- std::string _title
- Writer _writer
- Language _language
- Genre _genre
- Date _datePublished
- std::string _isnb
- std::vector<std::string> _lastsBorrowers
- bool _isBorrowed
+ explicit Book(std::string title, Writer writer, Language language, Genre genre, Date datePublished, std::string isnb, std::vector<std::string> lastsBorrowers, bool isBorrowed)
+ void addLastBorrower(const std::string &borrower)
+ std::string getLastBorrower()
+ [all Getter/Setter]
+ bool operator==(const Book &b) const
+ bool operator!=(const Book &rhs) const
< Contains
0..*
0..*
Library
- std::vector<Book> _books
- std::vector<Reader> _readers
- std::vector<Loan> _loans
+ Library()
+ void add(Book& b)
+ void add(Reader& r)
+ void add(Loan& l)
+ void add(const std::vector<Book>& books)
+ void add(const std::vector<Reader>& readers)
+ void add(const std::vector<Loan>& loans)
+ void delBook(int id)
+ void delReader(int id)
+ void delLoan(int id)
+ std::string displayBooks(bool show)
+ std::string displayReaders(bool show)
+ std::string displayLoans(bool show)
+ bool isBookInLibrary(const Book& b)
+ bool loan(Book& b,Reader& r,const Date& loanDate)
+ bool returnBook(Book& b,Reader& r)
Contains >
0..*
0..*
Reader
- std::string _name;
- std::string _surname;
- std::vector<std::string> _borrowedBooks;
- std::string _id;
+ explicit Reader(std::string name, std::string surname, std::vector<std::string> borrowedBooks, std::string id)
+ [all Getter/Setter]
0..*
Loan
V
1
Loan
- std::string _isnb;
- std::string _id;
- Date _loanDate;
+ explicit Loan (std::string isnb, std::string id, const Date &loanDate)
+ [all Getter/Setter]
Contains >
0..*
0..*
1
1
^
Wrote
1..*
1
0..*
is loan >
1
Writer
- std::string _name;
- std::string _surname;
- Date _birthdate;
- std::string _id;
+ explicit Writer(std::string name, std::string surname, Date birthdate, std::string id);
+ [all Getter/Setter]
+ bool operator == (const Writer& w) const
+ bool operator != (const Writer& w) const
0..*
0..*
1
Date
- int _day
- int _month
- int _year
+ Date()
+ Date(int day, int month, int year)
+ [all Getter/Setter]
+ [all operator]
0..*

Unicité de l’identifiant d’un objet

L’une des question que je me suis posé à été le fait de garantir l’unicité de l’identifiant d’un objet c’est à dire que par exemple que la propriété *_id* de la classe *Writer* permette d’identifier avec certitude et sans ambiguïté un et un seul *Writer.*

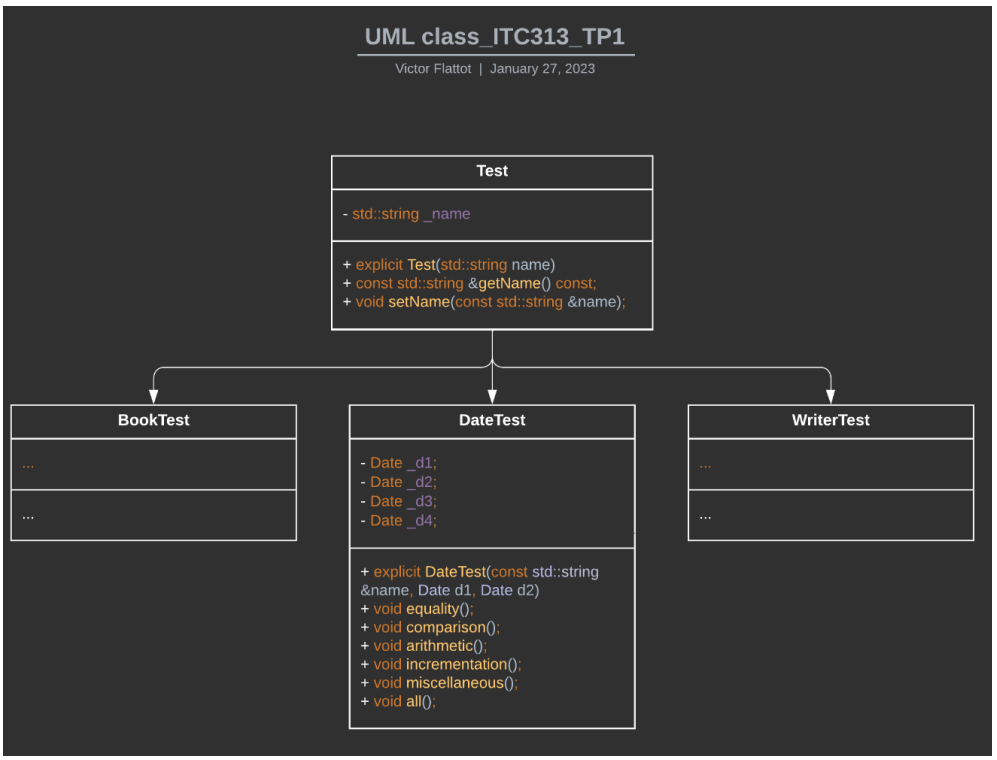
Pour ce faire, j’ai décidé de créer dans chaque classe un *vector* d’*id*. A chaque fois que l’on crée un objet l’id de celui-ci est ajouté dans ce *vector* si et seul si celui n’est pas déjà présent dans le *vector*.

```
std::vector<std::string> existingId = {};

bool isIdValid(const std::string& id) {
   if (std::find(existingId.begin(), existingId.end(), id) !=
existingId.end() )
       return false;
   return true;
}
```

Création de classe de Test

Afin de pouvoir tester mes différentes classes j’ai décidé de créer un namespace test. Dans celui-ci, on retrouve une classe mère *Test* et des classe fille pour chaque classe.



Pour ce TP j'ai juste fait une classe pour tester la classe Date comme *proof of concept*. Lorsque je lance le programme de test en mode *coverage* j'obtiens pour la classe *Date*.

Je peux donc encore affiner mes tests afin de tester à 100% des branches. Mais je n'ai pas encore eu le temps de me pencher sur le concept de "Vrai" test pour le C++.

Conclusion

Lors de la réalisation de ce TP, j'ai pu mettre en pratique de nombreux concepts vus lors des cours. Notamment le concept de référence que je n'avais pas tout à fait compris mais qui s'est éclairci en pratiquant. De plus, lors des séances de TP, j'ai pu mettre à contribution mes compétences en programmation au service des autres élèves qui n'avait pour une grande partie jamais faire de la programmation orienté objet voir de la programmation tout court.