POO

Compte Rendu

ITC313_TP2

Victor FLATTOT

Encadré par

GINHAC Dominique
TEL Steven
NKONDJOCK Waytehad

27/01/2023

Table des matières

Introduction

Ce TP a pour objectif de concevoir une application de gestion des réservations d'un hôtel.

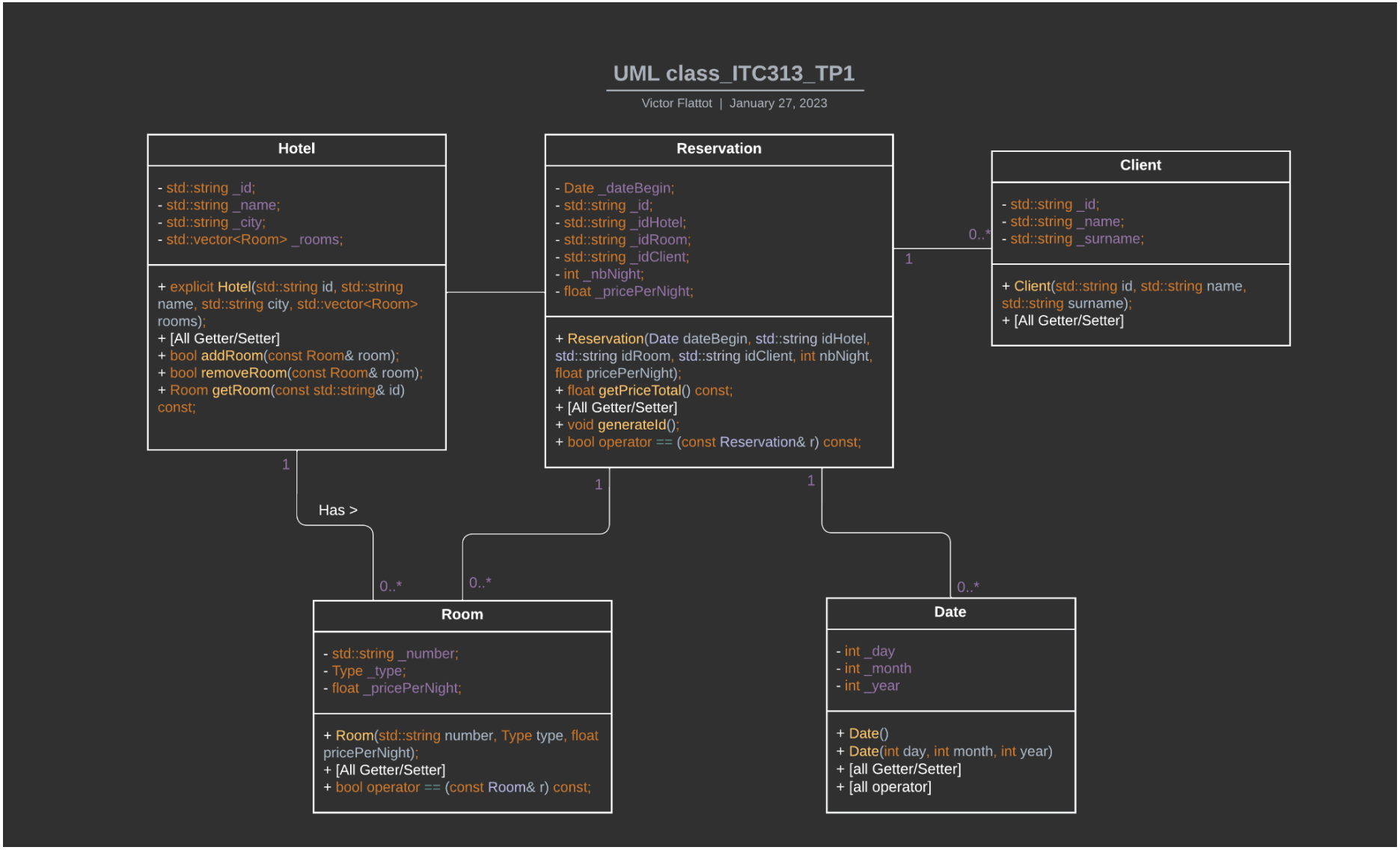
Pourquoi ce TP ?

Ce TP fait suite au TP précédent, il me semblait donc logique de continuer les TP dans l'ordre afin de voir une certaine progression. De plus, en découvrant le TP, j'ai remarqué que celui-ci se reposait en grande partie sur la gestion de container de la standard library et sur quelque algo que celle-ci comporte. Bref, un TP idéal pour renforcer ma compréhension de la "std".

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 le TP tournait principalement autour de la classe *Reservation* qui regroupe toutes les autres classes de l'application. La gestion des *Rooms* est aussi un point important car il faut filtrer celles-ci selon par exemple soit leur type mais également selon le fait qu'elles soit disponible ou non pendant certaine période.

Le diagramme UML suivant est un bon résumé de l'organisation de ces classes de comment elles interagissent entre elles. Il ne présente en revanche pas les nombreuses fonctions helper qui gravitent autour de chaque classe.


UML class_ITC313_TP1
Victor Flattot | January 27, 2023
Hotel
- std::string _id;
- std::string _name;
- std::string _city;
- std::vector<Room> _rooms;
+ explicit Hotel(std::string id, std::string name, std::string city, std::vector<Room> rooms);
+ [All Getter/Setter]
+ bool addRoom(const Room& room);
+ bool removeRoom(const Room& room);
+ Room getRoom(const std::string& id) const;
Reservation
- Date _dateBegin;
- std::string _id;
- std::string _idHotel;
- std::string _idRoom;
- std::string _idClient;
- int _nbNight;
- float _pricePerNight;
+ Reservation(Date dateBegin, std::string idHotel, std::string idRoom, std::string idClient, int nbNight, float pricePerNight);
+ float getPriceTotal() const;
+ [All Getter/Setter]
+ void generateId();
+ bool operator == (const Reservation& r) const;
Client
- std::string _id;
- std::string _name;
- std::string _surname;
+ Client(std::string id, std::string name, std::string surname);
+ [All Getter/Setter]
Room
- std::string _number;
- Type _type;
- float _pricePerNight;
+ Room(std::string number, Type type, float pricePerNight);
+ [All Getter/Setter]
+ bool operator == (const Room& r) const;
Date
- int _day
- int _month
- int _year
+ Date()
+ Date(int day, int month, int year)
+ [all Getter/Setter]
+ [all operator]
1
Has >
0..*
1
0..*
1
0..*
1
0..*

Fonction lambda

Lors de ce TP, on a besoin par exemple de filtrer les *Rooms* par leur type. Pour se faire deux solutions soit itérer en créant soit même une fonction tel que :

```
std::vector<Room> Hotel::getRooms(const Type &type) const {
   std::vector<Room> roomsAvailable;
   for(const auto & _room : _rooms){
       if(_room.getType() == type){
           roomsAvailable.push_back(_room);
       }
   }
   return roomsAvailable;
}
```

Mais de telles fonctions ne sont pas optimisées en termes de mémoire. Il y a donc une autre solution qui consiste à utiliser les algorithmes fournir par la “std” tel que ***copy_if*** :

```
std::vector<Room> Hotel::getRooms(const Type &type) const {
  std::vector<Room> roomsAvailable;
  std::copy_if(_rooms.begin(),_rooms.end(),
     std::back_inserter(roomsAvailable),
     [type](const Room& x) {return x.getType() == type;});
  return roomsAvailable;
};
```

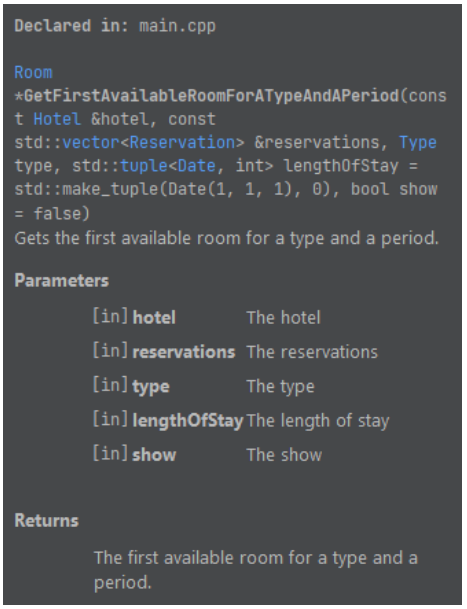
Pointeurs

Pour réaliser ce TP, j’ai dû parfois utiliser des pointeurs. Cette notion ne m’était pas inconnu étant donné que nous les avons étudiées en cours, mais la mise en pratique fut plus périlleuse. En effet, certaines de mes fonctions doivent retourner un objet mais parfois ces mêmes fonctions ne peuvent pas retourner cet objet, en cas d’absence de *Room* disponible après une recherche par exemple. J’ai donc créer des fonctions qui retourne, dans mon exemple, non pas un objet *Room* mais bien un pointeur *Room*.

```
Room * GetFirstAvailableRoomForATypeAndAPeriod(...){
....
auto roomsAvailable = getRoomsAvailableByTypeAndDate(hotel,reservations,
                                                     type, lengthOfStay);
Room *room = nullptr;
if (!roomsAvailable.empty()){
   room = new Room((roomsAvailable.at(0)));
   if(show) std::cout << "First Room available : \n" << *room;
}else if(show) std::cout << "No Room available\n";
return room;
}
```

Doxygen

Afin de documenter mon code j'ai utilisé Doxygen. Un générateur de documentation sous licence libre capable de produire une documentation logicielle à partir du code source d'un programme. Pour cela, il tient compte de la syntaxe du langage dans lequel est écrit le code source, ainsi que des commentaires s'ils sont écrits dans un format particulier. De plus, j'utilise CLion comme IDE et celui-ci prend en compte le format de Doxygen.




```
/**
 * @brief      Gets the first available room for a type and a period.
 *
 * @param[in]  hotel         The hotel
 * @param[in]  reservations  The reservations
 * @param[in]  type          The type
 * @param[in]  lengthOfStay  The length of stay
 * @param[in]  show          The show
 *
 * @return     The first available room for a type and a period.
 */
Room * GetFirstAvailableRoomForATypeAndAPeriod(const Hotel &hotel,
                                               const std::vector<Reservation> &reservations,
                                               Type type,
                                               std::tuple<Date, int> lengthOfStay =
                                               std::make_tuple( a1: Date( day: 1, month: 1, year: 1),  a2: 0),
                                               bool show = false){
```

Conclusion

Ce TP m’a permis de découvrir et surtout d’expérimenter les différents algorithmes de la “std” et également l’utilité des pointeurs. L’un des défauts principaux de mon code actuellement est le fait que j’ai mélangé dans une seule fonction, le fait de pouvoir rentrer itérativement les infos mais également le fait de pouvoir renseigner les informations dans les paramètres de la fonction. Ce système fonctionne pour moi mais je ne suis pas sûr que ce soit la manière la plus ergonomique de procéder.