Курсова работа

Тема: Хотел

Изготвена от: Владислав Иванов Георгиев

фак. номер:45699

Увод

Github хранилище:https://github.com/VGeorgiev1/fmi-course-projects/tree/master/Hotel

1.1 Описание и идея на проект

Целта на проекта е да се създаде програма, реализираща регистрация, менажиране и освобождаване на стаи в хотела. **Програмата не трябва да представя възможност за резервации, а само за регистрации в стаи от хотела.**

1.2 Цел и задачи на разработка

Програмата трябва да може да зарежда съдържанието на текстов файл съдържащ информацията за стаите в хотела както и регистрациите в него.
Една стая има следните характеристики:

- номер на стаята;
- брой легла;

Една регистрация в хотела представлява колекция от:

- дата за старт на регистрацията;
- дата за край на регистрацията;
- бележка към регистрацията;
- стаята, за която е регистрацията;

Главния набор от операции, които трябва да се поддържат от програмата са за вход/изход: open, close, save, save as, help и exit. Описанието на тези функции е тривиално. Тук важното е да се подчертае че **нито една друга команда не може да бъде извикана преди да бъде отворен файл с командата open.** Освен тези команди трябва да бъдат поддържани и следните команди за менажиране на хотела:

- checkin <room> <from> <to> <note> [<guests>] - Регистриране в стая с номер <room> от дата <from> до дата <to> и се добавя бележка <note>. Незадължителният параметър

<guests> задава броя на гостите, настанени в стаята. Ако той не е указан, се счита, че броят на настанените гости е равен на броя на леглата в стаята
- availability [<date>] - Извежда списък на свободните стаи на дата <date>, ако не е зададена, се използва текущата дата.
- checkout <room> - Освобождаване на заета стая с номер <room>.
- report <from> <to> - Извежда справка за използването на стаи в периода от дата <from> до <to>. Извежда се списък, в който за всяка стая, използвана в дадения период, се извежда и броя на дните, в които е била използвана.
- find <beds> <from> <to> - Намиране на подходяща свободна стая с поне <beds> на брой легла в периода от <from> до <to>. При наличие на повече свободни стаи се предпочитат такива с по-малко на брой легла.
- find! <beds> <from> <to> - Да се реализира алгоритъм, който предлага спешно намиране на стая за важен гост в случай на липса на свободни стаи за даден период. Алгоритъмът да предлага разместване на настанените от най-много две стаи.
- unavailable <room> <from> <to> <note> - Обявява стаята с номер <room> от дата <from> до дата <to> за временно недостъпна и се добавя бележка <note>. В стаята няма регистриран гост, но никой не може да бъде настанен в нея.

1.3 Структура на документацията

- Глава 1 - Увод в проекта
- Глава 2 - Преглед на предметната област на проекта
- Глава 3 - Проектиране на отделните компоненти на проекта
- Глава 4 - Начин на реализация и тестване
- Глава 5 - Заключение и бъдещи насоки

Преглед на предметната област

2.1 Основни концепции и алгоритми.

За решаване на задачата са използвани стандартни програмни структури в езика C++ като класове и вектори, символни низове и функционалности като входни/изходни операции, наследяване и полиморфизъм.

2.2 Проблеми, със които решението на задачата трябва да се справи

Най-важните задачи в проекта са:

- създаването на клас, който да менажира стаите и регистрациите в тях;
- създаването на лесен начин на за дефиниране на нови операции, които да изпълнява програмата
- четенето и записването на информацията за състоянието на хотела във файл

2.3 Подходи, методи за решаването

За решаването на задачата е дефиниран клас, който да съдържа вектор със стаите и регистрациите в хотела и дефинира методи за достъпване и проверки върху тези обекти. По този начин всяка една операция се нуждае само референция към обекта, за да извърши нужната функция върху него. За реализацията на командния интерфейс е използван модела Command pattern. Този модел позволява лесно добавяне на операции без да се усложнява логиката, на останалата част от програмата. По този начин всяка операция се грижи за това какво вижда потребителя при изпълнението и - останалата част от логиката няма нужда да се грижи за това.

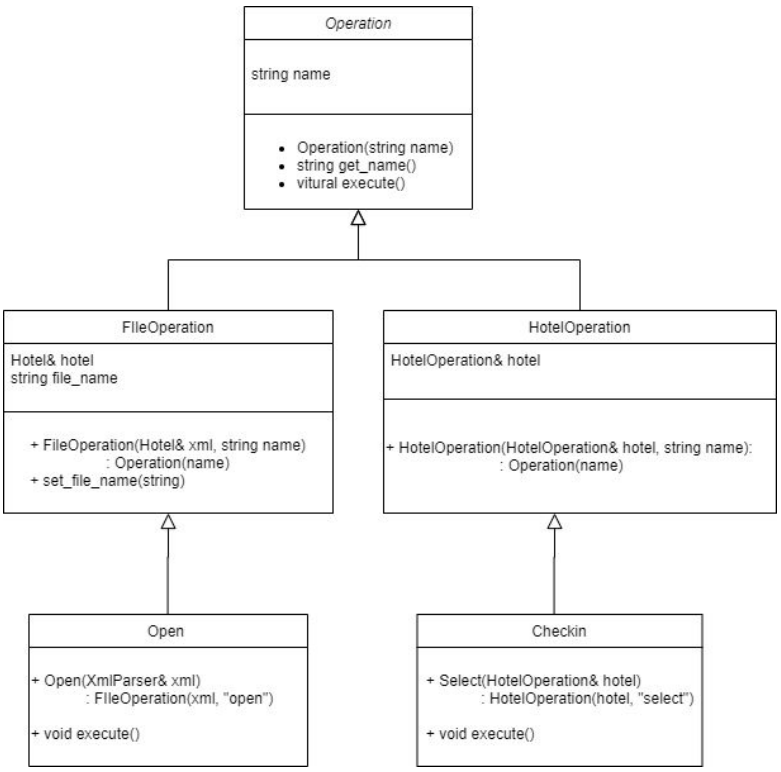
Проектиране

3.1 Обща архитектура реализация на проекта

Главните конструкции в проекта са следните класове:

- Hotel
- Operation

Hotel се грижи за, съхранението на стаите и регистрациите докато програмата работи с тях, добавянето и редактирането им и достъп до отделни стаи и регистрации. Operation е виртуален клас със две функции - едната за задаване на име на операцията и една за виртуална функция, която всяка операция трябва да имплементира. Конструктора на Operation приема референция към Hotel върху, който да извършва операциите. Този подход има своите недостатъци тъй като трябва да се гарантира че “живота” на Hotel трябва да е по-дълъг от този на Операциите закачени за него. Това е постигнато като Hotel съдържа вектор със *pointers* сочещи към инстанции на операции. След това в деструктора на Hotel тези инстанции биват изтрити. По-този начин е гарантирано, че инстанцията на всяка една операция за даден Hotel няма да “надживее” класа, към които държат референция. Това също улеснява начина на намиране на операции, при четенето но името им от стандартния вход. Класа Operation е наследен от два други класа - FileOperation и Hotel. Първият държи информация за в момента отворения файл, а втория за Hotel класа върху, който да извършва операциите по редактиране и селектиране на Hotel елементи. Диаграмата по-долу показва начина на наследяване и работа на операциите:



фигура 3.1.1 Дефиницията на операциите

При настъпване на грешка при изпълнение на някой от функциите е реализиран отделен клас, наследяващ std::logic_error, който да описва какъв проблем е настъпил:

```
#ifndef OPERATION_EXC_H
#define OPERATION_EXC_H


#include <stdexcept>
class OperationException : public std::logic_error {
public:
    OperationException(std::string exc);
};
#endif
```

За реализацията на проекта е създаден и специален клас който да репрезентира дата. Интерфейса на класа е следния:

```
class Date {
private:
    int year;
    int month;
    int date;
public:
    Date();
    Date(std::string date_string);
    static int monthsDays[12];

    bool operator <(const Date& other);
    bool operator >(const Date& other);
    bool operator==(const Date& other);
    bool operator!=(const Date& other);
    bool operator <=(const Date& other);
    bool operator >=(const Date& other);
    int operator-(const Date& other);
    friend std::ostream& operator <<(std::ostream& os, const Date& dt);

    int get_year() const;
    int get_month() const;
    int get_date() const;

};
```

фиг 3.1.2 Дефиниция на клас Date

По този начин сравнението и работата със датите от останалата част от програмата става по-лесна. Дефиницията на класа Hotel включва методи за добавяне, на стаи и регистрации. Освен това има методи като - проверки дали съществува регистрация за дадена стая в даден период или дали дадена стая е обявена за недостъпна за даден период.

```
Room* get_room(int number) const;
Room* get_most_fitting_room(int requested_beds, Date start, Date end, int room_to_skip, bool check_for_checkin) const;
bool is_unavailable_for_period(int room, Date start, Date end) const;

void add_record(Record r);
```

фиг. 3.1.2 Част от функциите на клас Hotel

Важно е да се отбележи, че стаите в Хотел-а се пазят под формата на pointers. А всяка една от стаите се държи в HEAP паметта. Така регистрациите се нуждаят само от този pointer и не налага копиране на обекти. Деструктора на Хотел се грижи за изтриването на стаите след като обекта приключи своето ползване. По този начин всяка една стая “живее” колкото хотела в, който е и след това бива записана във файл. Важна функция във Hotel е тази за намирането на най-подходяща стая в хотела за даден брой гости - намира стая с минималния брой легла, в даден интервал от време, които отговарят на заявката при регистрация Като допълнителни аргументи тази функция приема номер на стая, която да бъде пропусната при търсенето на стая (ако той е 0 не се пропуска нито една стая при търсенето) и дали да се търсят стаи, които вече имат регистрации.

```
Room* Hotel::get_most_fitting_room(int requested_beds, Date start, Date end, int room_to_skip, bool check_for_checkin) const {
    int min_beds = INT_MAX;
    Room* r = nullptr;

    for (std::vector<Room*>::const_iterator it = rooms.begin(); it != rooms.end(); ++it) {
        int beds = (*it)->get_beds();

        if (beds >= requested_beds && beds <= min_beds && !is_unavailable_for_period((*it)->get_number(), start, end)) {
            if (room_to_skip != 0 && room_to_skip == (*it)->get_number()) {
                continue;
            }
            if (check_for_checkin && room_is_cheked_in((*it)->get_number(), start, end)) {
                continue;
            }

            min_beds = beds;
            r = (*it);
        }
    }

    return r;
};
```

фиг. 3.1.3 Дефиниция на функция get_most_fitting_room

Друг важен момент от реализацията на проекта е командата *find!*:

```
Room* Findem::create_room(int beds, Date start, Date end, int room_to_miss, int step) {


    Room* most_fitting_room = hotel_.get_most_fitting_room(beds, start, end, room_to_miss, false);

    if (most_fitting_room == nullptr) {
        throw OperationException("No rooms with the needed beds were found and no moves were possible to be made!");
    }

    if (hotel_.room_is_cheked_in(most_fitting_room->get_number(), start, end)) {
        Record* r = hotel_.get_check_in_for_room(most_fitting_room->get_number());

        if (most_fitting_room == nullptr && step == 2) {
            throw OperationException("No rooms with the needed beds were found and no moves were possible to be made!");
        }

        Room* room_to_change = create_room(r->get_beds_taken(), r->get_start_date(), r->get_finish_date(), r->get_room()->get_number(), step++);

        int room_numb = r->get_room()->get_number();

        if (room_to_change != nullptr) {
            hotel_.change_room(most_fitting_room,room_to_change);
        }

        return most_fitting_room;
    }
    return most_fitting_room;
}
```

Условието на задачата лимитира търсенето в хотела на стая до две стъпки най-много, но чрез рекурсивното търсене, ако на по-късен етап това условие бъде премахнато търсенето на подходяща стая може да бъде направено за целия хотел.

Заключение

4.1 Обобщение на изпълнените задачи

Програмата реализира регистрационна система за хотел. Нужните функционалности от условието са реализирани и информацията за системата се пазят в отделни от програмата файлове.