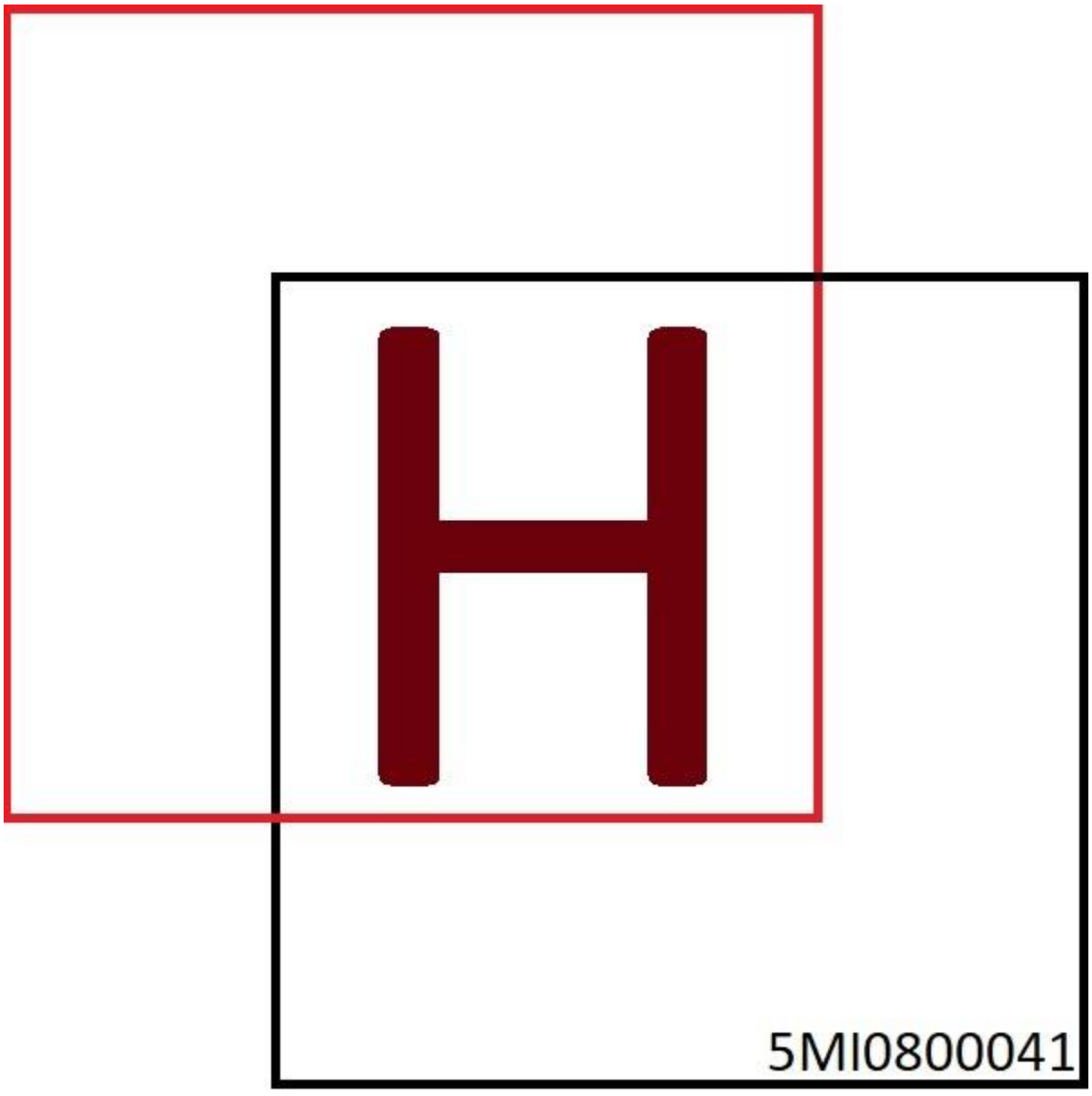
Документция за проект № 1

Тема: 6. Хотел



От: Данаил Иванов Димитров, 5MI0800041, КН, група 4, курс 1

Съдържание

Глава 1: Увод

1.1 Описание и идея на проекта

Идеята на проекта е да се направи софтуер, който позволява на служители на хотел да следят и управляват състоянието на стаите в хотела им.

1.2 Задачи на разработката

Поставени са 6 основни изисквания за проекта от условието, които за записани по-долу. Освен това програмата трябва да работи в диалогов режим, като трябва вижданото от потребителя да е максимално изчистено и разбираемо.

1.2.1 Регистрация на гост

При въведени номер на стая, времеви период, име на гост и коментар, които не е задължителен, трябва да се регистрира гост в стая за периода от време, ако стаята е свободна.

1.2.2 Извеждане на свободни стаи

При въведена желана дата трябва да се изкарат всички стай, които са свободни на съответната дата.

1.2.3 Освобождаване на стая

При въведен номер на стая, съответната стая трябва да се освободи ако в момента е заета.

1.2.4 Справка за заетост

При въведен времеви период трябва да се изведе справка за заетостта на всички стаи в хотела за този период. Справката трябва да се запише във файл с име “report-{година}-{месец}-{ден}”, където гофината, месеца и деня са началната дата.

1.2.5 Търсене на перфектната стая

При въведен минимален брой легла и времеви период, трябва да се изведе/изведадат стая/стаи (ако са няколко) която/които покриват изискванията. Като ако има такива стаи с различен брой легла се предпочитат тези, които имат по-малко на брой такива.

1.2.6 Затваряне на стая

При въведен номер на стая, тя трябва да се затвори. Като се премахнат всички резервации от сегашната дата нататък.

Глава 2: Преглед на предметната област

2.1 Основни дефиниции и концепции, които ще се използват

2.1.1 *PerfectRoom*

Среща се в кода на места, свързани с решаването на задачата описана в точка 1.2.5. Означава най-добрата стая спрямо посочените критерии.

2.1.2 Основен клас

По този начин ще наричам класовете *Room* и *RoomReservation*, които представляват обекти, които представляват обектите, които представят предмети от реален хотел.

2.1.3 Помощен клас

По този начин ще наричам класовете *String* и *Date*, които улесняват работата в проекта (Пример: благодарение на *String* не се налага да работим с масив от символи, да се грижим за размера му, терминиращата нула и тн., защото *String* прави това за нас). Ползват се като член-данни в основните класове. Представляват идеите за съответно текст и дата.

2.1.4 Архитектурен клас

По този начин ще наричам класовете Engine, *IOController* и *HelperController*, които осъществяват архитектурата на проекта.

2.1.5 *Singleton*

Архитектурен шаблон (*design pattern*), подсигурява, че от класа, които го имплементира, може да се създаде само един единствен обект.

2.1.6 Диалогов режим на работа

За себе си съм дефинирал този термин по следният начин. Начина по който даден потребител работи с програмата наподобява диалог. Потребителя въвежда команда (казва нещо на програмата) и тя дава резултат (отговаря).

2.2 Сложност и дефиниране на проблеми

2.2.1 Сложност

Най-сложната част на задачата е работата с файлове. При тях трябва по-голямо внимание и евентуално повече тестване, за да се изгладят възникнали проблеми.

2.2.2 Проблеми

1. Избора кога да се записва информация във файловете. Избирах между две възможности:

   а) При първата в началото на програмата създавам масиви от стай и резервации за стаи, които пълна с файловете. По време на работа се работи с масивите и накрая на програмата информацията се записва обратно във файловете. При този вариант се пести време и се предоставя директен достъп до всеки опект.

   б) При втората възможност четем обектите направо от файловете и записваме в тях всеки път когато направим промяна или се създаде нов обект, които трябва да запазим. Така губим директния достъп и хабим повече време, но спестяваме памет(Във вариант едно пазим информация за обектите на две места - файла и масива) и ако програмата прекъсне по някаква причина няма да загубим

информация, тя вече ще е във файла(Има шанс програмата да прекъсне в момента между създаване на обекта и записване във файла, тогава ще загубим информация, но само за този обект и шанса да се случи това е по-малък). Поради тази причниа избрах втория вариант.

2. Как да се запазват резервациите във файл. Отново избирах между два варианта:

а) Всички резервации да са в един файл. Когато се иска резервации за дадена стая се обхождат всички резервации и се взимат тези, които за roomId имат номера на желаната стая.

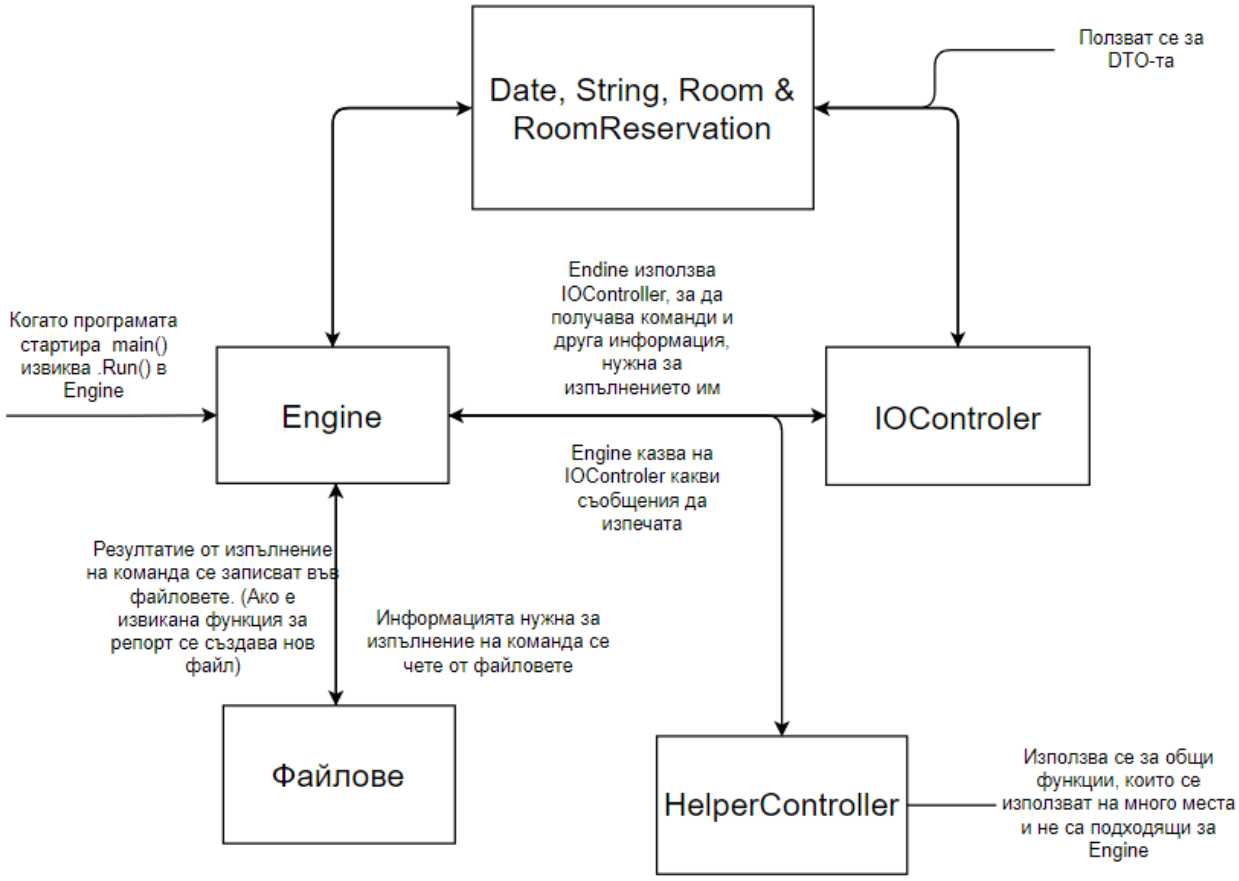
б) Резервациите за всяка стая се запазват в отделен файл. Когато се искат резервации за дадена стая се сглобява името на файла, където са записани резервациите, като се ползва номера на стаята.

Използвах втория вариант, защото времето да се намери дадена резервация ще е много по-малко. В същото време не хабим повече памет, записваме същия брой обекти, просто са в няколко файла. Негативната страна на този подход е, че може да бъдат създадени много файлове, въпреки това намирам втория вариант за по-добър.

3. Дали да се използват бинарни или текстови файлове:

Избрах текстови, за да е по-лесно да се проследи дали програмата работи правилно и записва правилните неща във файловете, както и да мога да пиша ръчно в тях за тестови цели. Освен това по условие стайте трябва да са предварително зададени, с бинарен файл това би било по-трудно. Също така когато файловете са текстови потенциален потребител на програмата би могъл да ги отвори и да види информацията, от която се нуждае от файловете.

Глава 3: Архитектура



Цялата логика на програмата се намира в архитектурните класове *Engine* (отговаря за изпълнението на командите) и *IOController* (отговаря за приемането на команди и друга нужна информация, както и за извеждането на резултати и други съобщения).

В началото на програмата се вика метода .run() в *Engine*. Това е може би най-важният метод за програмата. Той използва *IOController*, за да получи команда и спрямо това каква е тя вика съответния метод в *Engine,* който отново чрез *IOController* получава информацията, нужна за изпълнението и, извършва нужните операции и накрая вика нужния метод в *IOController*, които да отпечата резултат. Процеса се повтаря до команда за приключване на програмата

Обектите от класовете Room, RoomReservation, Date и се ползват като преносители на информация (Data Transfer Object -DTO). Чрез тях се запазва и пренася информацията в програмата.

Ако се хвърли грешка (exception) в повечето случай ще се хване от try-catch блок в .run(), ще се изпечата съобщението в нея и програмата ще продължи. Важно е да се отбележи, че към този момент има грешки, които не се хващат. Например ако се въведе символ когато се въвежда дата (символ вместо число) програмата става неизползваема и поради начина по които е написан .run() се създава безкраен цикъл, в които се печата една и съща информация безкраен брой пъти, без да може да се въвежда. В този момент програмата трябва да се спре ръчно и да се пусне на ново, за да се продължи работа.

Повече за класовете в **следващата глава**.

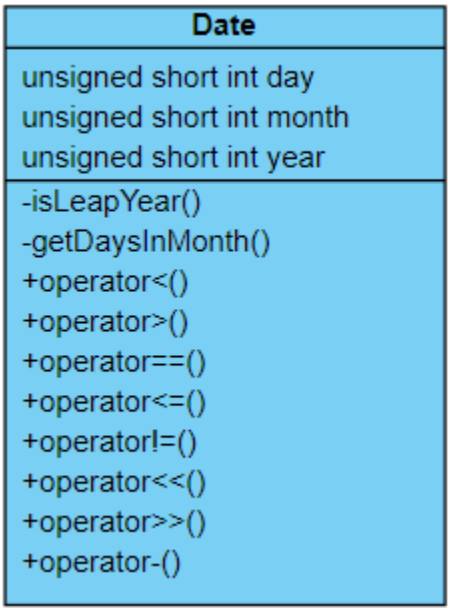
Глава 4: Реализация

Не всички методи ще бъдат обяснени подробно, доста от тях се разбират от имената си, ще обясня подробно по важните, тези, които може да бъдат объркани и тези, в които има някаква особеност, например валидации.

4.1 Реализация на помощни класове

4.1.1 Date

Представлява дата.



- `void setDay(const unsigned short day)`

Това е мутаторът за ден. Туй като трябва да сравни деня спрямо годината и месеца, ако те не са зададени (тоест са 0) ще се хвърли грешка. Ако стойността за ден, която се задава е по-голяма от максималната за дадения месец се хвърля грешка.

- `void setMonth(const unsigned short month)`

Мутатор за месец. Ще се хвърли грешка ако стойността, която се задава е 0 (нулата показва, че дадената член-данна не е зададена) или ако е по-голяма от 12 (има 12 месеца в година).

- `void setYear(const unsigned short year)`

Мутатор за година. В него има две константи, които определят минималната и максималната стойност за година. Ще се хвърли грешка ако стойността е извън зададената стойност.

- `unsigned operator-(const Date& other) const`

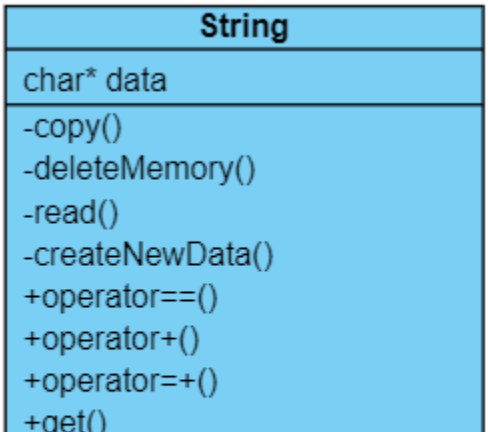
Презаписване на оператора -. Връща броя на дните между две дати. Ще се хвърли грешка ако датата other е по-голяма от сегашния обект, защото ако се продължи докато other е по-голяма цикъла в метода няма да работи правилно.

- ```
  std::ostream& operator<<(std::ostream& stream,
                           const Date& obj)
  ```

Оператор за извеждане. Важното за отбелязване е, че има константа, която определя символът, които разделя дните, месеците и годините.

4.1.2 String

Представлява текст.


  ```

- `std::istream& String::read(const char separator, std::istream& stream, unsigned capacity)`

Методът за четене, използван от публичните методи operator>>() и get(). Чете до даден разделител от даден поток и задавава прочетената стойност на член-данната.

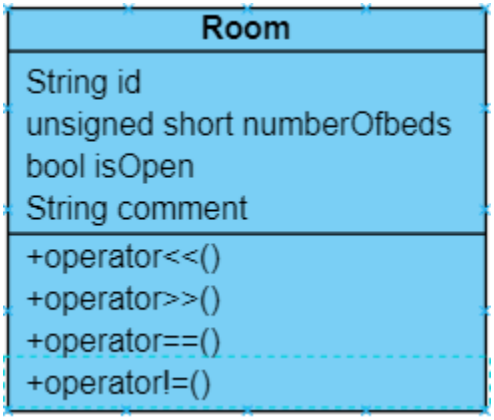
- `char* creatNewData(const String& other) const;`

Конкатенира два стринга. Методът се вика от операторите + и +=. Методът връща новополучения стринг.

4.2 Реализация на основни класове

4.2.1 Room

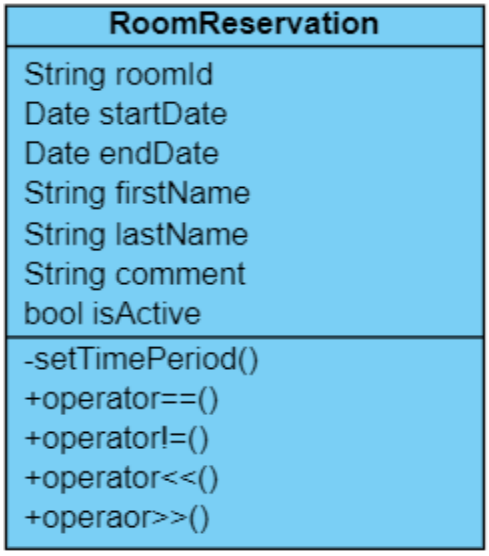
Представлява стая в хотела.



В този клас методите са лесни да се разберат само от имената си.

4.2.2 RoomReservation

Представлява резервация за стая.



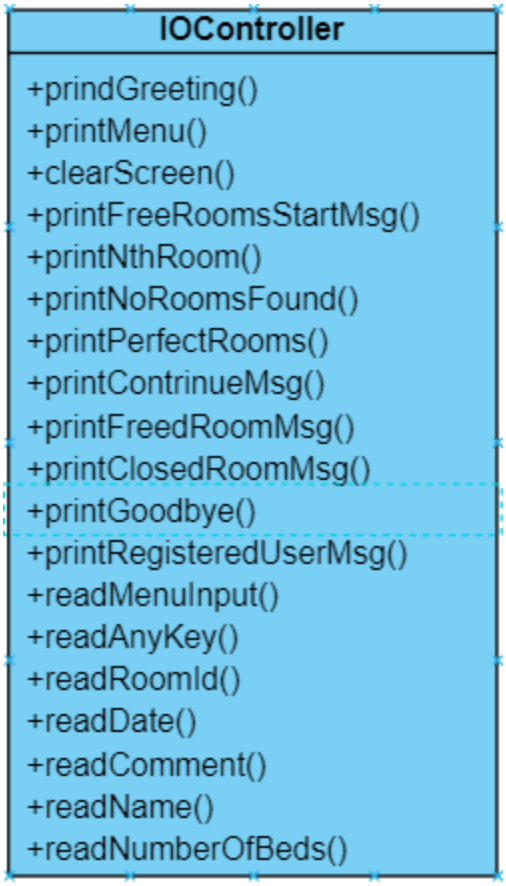
- `void setTimePeriod(const Date& startDate,`
  `const Date& endDate)`

Мутатор за начална и крайна дата. Той е общ, защото трябва да се сравнят двете дати една с друга. Ще се хвърли грешка крайната дата е преди началната.

4.3 Реализация на архитектурни класове

4.3.1 IOController

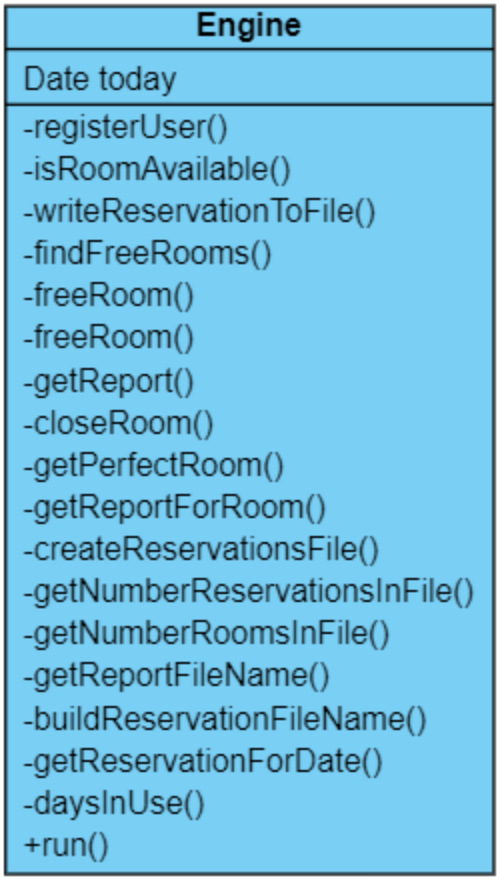
Класът отговаря за въвеждането и извеждането на информация. Всички негови методи са статични, защото искам да могат да се достъпят без да е нужно да се създаде обект от класа. Не имплементира Singleton, защото няма да е проблем да се създадат повече от един обект.



В този клас също методите са доста лесни за разбиране, достатъчно къси са да няма проблеми с разбирането. Важно е да се отбележи, че всеки метод позволява да се подадат потоци за писане и/или извеждане вместо да ограничава само до std::cin и std::cout.

4.3.2 Engine

Основния клас за програмата, които изпълнява командите и извършва всички нужни действия. Класа имплементира Singleton, тъй като той диктува целия ход на действие на програмата и ако съществуват два обекта ще е по-лесно да се получи неочаквано поведения от страна на програмата.



- `void run()`

Най-важният метод за програмата и единственият публичен в класа. Използва IOController, за да получи днешната дата. След това започва цикъл, които се повтаря докато не се получи команда за край на програмата. Метода вика други такива от класа на база получената команда. Ако се хвърли грешка в някой от тях ще се хване в try-catch блок и програмата ще продължи.

- `unsigned daysInUse(const RoomReservation& reservation,`
  `const Date& firstDate, const Date& secondDate)`

Връща броя дни, в които стаята е заета в даден период.

- `void createReservationsFile(const String& fileName)`

Лесен за разбиране метод. Важно е да се спомене, че той съществува, защото когато се отвари файл за четене, ако не съществува няма да се създате. С този метод се подсигурява, че това няма да е проблем.

- `void Engine::getReportForRoom(const Room& room, const Date& startDate, const Date& endDate, std::ofstream& reportFile)`

Метода строй частта от доклада за дадена стая и го предава на IOController за печатане.

- `void freeRoom(const Room& room)`

Освобождава дадена стая, която се чете чрез IOController. Промените се запазват като се презапише файла с резервации за стаята.

- `void closeRoom()`

Методът затваря дадена стая, която се чете чрез IOController. Резервациите за нея, които започват след днешната дата (ако има резервация в момента тя също се трие).

- `void getReport()`

Създава доклад за даден период, които се чете от IOController. Вика getReportForRoom() за всяка стая във файла на стаите.

- `void getPerfectRoom()`

Намира перфектната стая, по критерии, които се четат с IOController.

- `void findFreeRooms()`

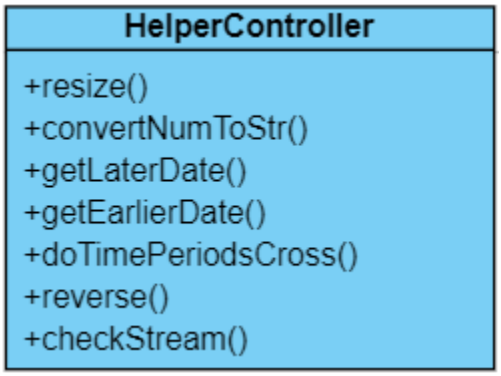
Намира всички свободни стаи на дата, прочетена с IOController.

- `void registerUser()`

Регистрира гост в стая. Нужната информация се чете от IOController.

4.3.3 HelperController

В този клас се намират методи, които помагат с извършването на различни действия, и не би било подходящо да седят в другите класове. Те са статични, за да могат да се викат без да трябва да се създава обект когато искаме да ги достъпим.



- `static void checkStream(std::ios& stream)`

Проверява дали даден файл е успешно отворен. Ако не е се хвърля грешка.

4.4 Управление на паметта

Динамична памет има само в String, които имплементира голямата четворка.

4.5 Тестване

На този етап са правени само ръчни тестове.

Глава 5: Заключение

5.1 Изпълнение на началните цели

Програмата изпълнява всички поставени задачи и предоставя лесен начин за работа с нея.

5.2 Бъдещо развитие усъвършенстване

5.2.1 Отделянето на работата с файлове в отделен архитектурен клас

На този етап цялата работа с файлове се извършва в Engine. Това е леко нарушение на *single responsibility* принципът. Отделянето ще доведе до създаването на трислоен модел, което ще доведе до улесняване на поддръжката на кода и ще го направи по-лесен за разбиране.

5.2.2 Оправяне на проблеми с въвеждането

В редки случай при въвеждане на нов ред например, когато не се очаква такъв вход може да се достигне до неочаквано поведение. Друг проблем с входа е например описаното тук.