**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая РАБОТА

**по дисциплине «Объективно-ориентированное программирование»**

Тема: Разработка приложения на основе объектно-ориентированного подхода

|  |  |  |
| --- | --- | --- |
| Студентка гр. 9301 |  | Русанова К.В. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2021

# Задание на курсовую работу

|  |  |  |
| --- | --- | --- |
| Студент Русанова К.В. | | |
| Группа 9301 | | |
| Тема работ: разработка приложения на основе объектно-ориентированного подхода | | |
| Исходные данные:  Задания для первой, второй и третьей части курсовой указаны на страницах …  Язык программирования C# версии .NET Core 3.1 | | |
| Содержание пояснительной записки:  «Содержание», «Аннотации», «Цель», «Введение», «Первый раздел. Создание иерархии классов», «Второй раздел. Определение остова наименьшего веса с помощью алгоритма ближайшего соседа», «Третий раздел. Имитационное моделирование», «Заключение», «Список использованных источников» | | |
| Предполагаемый объем пояснительной записки:  Не менее 40 страниц. | | |
| Дата выдачи задания: 08.02.2021 | | |
| Дата сдачи курсовой: 03.06.2021 | | |
| Дата защиты курсовой: 04.06.2021 | | |
| Студент |  | Русанова К.В. |
| Преподаватель |  | Новакова Н.Е. |

# Аннотация

Для выполнения работы были изучены материалы курса объектно-ориентированного программирования и дискретной математики. Реализация приложений осуществлена на языке C#. Для успешной разработки качественных программ был произведён контроль на соответствие принципам SOLID.

# Summary

To carry out the work, the materials of the course of object-oriented programming and discrete mathematics were studied. The applications are implemented in the C # language. For the successful development of high-quality programs, control was carried out for compliance with the SOLID principles.

**содержание**

[Задание на курсовую работу 2](#_Toc73581789)

[Аннотация 3](#_Toc73581790)

[Summary 3](#_Toc73581791)

[Цель 6](#_Toc73581792)

[Введение 6](#_Toc73581793)

[1 Первый раздел. Создание иерархии классов 7](#_Toc73581794)

[1.1 Задание 7](#_Toc73581795)

[1.2 Теоритический аспект 7](#_Toc73581796)

[1.3 Формализация задачи 7](#_Toc73581797)

[1.4 Спецификация программы 9](#_Toc73581798)

[1.5 Руководство пользователя 11](#_Toc73581799)

[1.6 Руководство программиста 11](#_Toc73581800)

[1.7 Контрольный пример 11](#_Toc73581801)

[1.8 Листинг 12](#_Toc73581802)

[2 Второй раздел. Гамильтоновы циклы 17](#_Toc73581803)

[2.1 Задание 17](#_Toc73581804)

[2.2 Теоретический аспект 17](#_Toc73581805)

[2.3 Формализация задачи 18](#_Toc73581806)

[2.4 Спецификация программы 19](#_Toc73581807)

[2.5 Руководство пользователя 21](#_Toc73581808)

[2.6 Руководство программиста 22](#_Toc73581809)

[2.7 Контрольный пример 22](#_Toc73581810)

[2.8 Листинг 23](#_Toc73581811)

[3 Третий раздел. Имитационное моделирование 27](#_Toc73581812)

[3.1 Задание 27](#_Toc73581813)

[3.2 Теоретический аспект 28](#_Toc73581814)

[3.3 Формализация задачи 28](#_Toc73581815)

[3.4 Спецификация программы 32](#_Toc73581816)

[3.5 Руководство пользователя 35](#_Toc73581817)

[3.6 Руководство программиста 36](#_Toc73581818)

[3.7 Контрольный пример 36](#_Toc73581819)

[3.8 Листинг 39](#_Toc73581820)

[Заключение 50](#_Toc73581821)

[Список использованных источников 51](#_Toc73581822)

# Цель

Целью курсовой работы является закрепление теоретических знаний и получение практических навыков разработки программного обеспечения на основе объектно-ориентированного подхода

# Введение

В данной работе применены навыки разработки приложений, основанных на принципах объектно-ориентированного подхода. Реализована иерархия классов, осуществлён поиск гамильтонова цикла и разработана имитационная модель.

Для выполнения курсовой работы использовался язык программирования C#. В работах присутствует пользовательский интерфейс, реализованный с помощью диалогов в консоли и элементов управления Windows Forms.

# Первый раздел. Создание иерархии классов

## Задание

Вариант №19

Разработать программу для обеспечения продажи театральных билетов

## Теоритический аспект

Театр — зрелищный вид искусства, представляющий собой синтез различных искусств — литературы, музыки, хореографии, вокала, изобразительного искусства и других.

Билет (фр. billet — записка, удостоверение) — документ, удостоверяющий наличие некоего права у какого-либо определённого лица или у предъявителя билета. Действие билета может распространяться на конкретное время или не иметь сроков.

В данном задании необходимо осуществить продажу билетов на спектакль, выбранный пользователем.

## Формализация задачи

Исходными данными являются классы касс, спектаклей и билетов. Программа должна выводить возможные спектакли, места и стоимость билетов для бронирования и покупки.

Диаграмма используемых классов представлена на рисунке 1.3.1

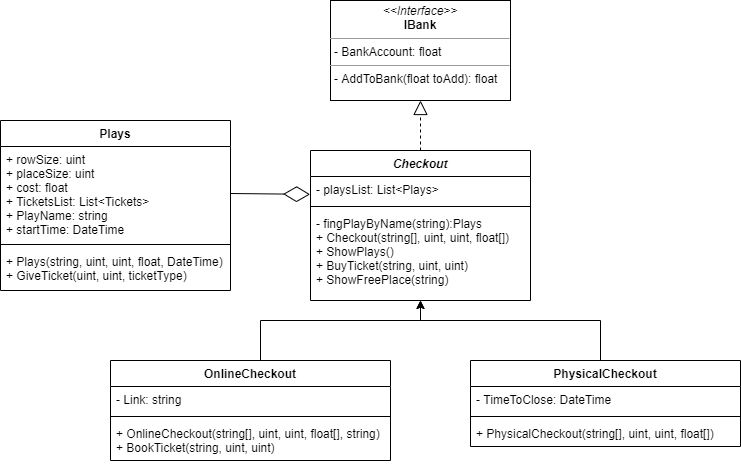


Рисунок 1.3.1 — UML диаграмма классов, необходимых для выдачи театральных билетов

Классы:

* abstract Checkout — описывает общий вид касс.
* Online Checkout — описывает онлайн кассы и наследуется от Checkout.
* Physical Checkout — описывает физические кассы и наследуется от Checkout.
* Plays — описывает спектакли

Интерфейс:

* IBank — хранит банковский счет и пополняет его

## Спецификация программы

Ниже представлены таблицы классов, структур интерфейсов, реализованных в программе.

Класс Plays

Таблица 1.1 — Описание методов класса Plays

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Plays | Создание элемента класса | - | Public | Название, количество мест, стоимость, время проведения |
| GiveTicket | Бронирование или покупка билета | void | Public | Название, место, тип операции |

Таблица 1.2 — Описание полей класса Plays

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| TicketsList | List<Tickets> | public readonly | Список билетов |
| PlayName | String | Public | Название спектакля |
| rowSize | uint | Public | Количество рядов |
| placeSize | Uint | Public | Количество мест |
| cost | float | public readonly | Стоимость билета |
| startTime | DateTime | public | Время начала |

Интерфейс IBank

Таблица 1.3 — Описание методов интерфейса IBank

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| AddToBank | Пополнение баланса банка | void | private protected static | Сумма к добавлению |

Таблица 1.4 — Описание полей интерфейса IBank

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| BankAccount | float | private static | Сумма в банке |

Класс Checkout

Таблица 1.5 — Описание методов класса Checkout

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Checkout | Создание элемента класса | - | Public | Названия спектаклей, размеры залов, стоимость |
| ShowPlays | Вывод списка спектаклей | void | Public | - |
| BuyTicket | Покупка билета | void | Public | Название спектакля, места |
| fingPlayByName | Поиск спектакля | Plays | private protected | Название спектакля |
| ShowFreePlace | Вывод свободных мест | void | Public | Название спектакля |

Таблица 1.6 — Описание полей класса Checkout

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| playsList | List<Plays> | private | Список спектаклей |

Описание методов и полей Online Checkout и Physical Checkout наследуете от Checkout.

## Руководство пользователя

Программа предназначена для осуществления покупки или бронирования билета в театральной кассе.

Сначала пользователю будет предоставлен список всех возможных спектаклей, на которые можно купить билет. Далее необходимо ввести название желаемого спектакля. После этого, программа выведет стоимость билета и свободные места и попросит ввести ряд и место для бронирования. При успешном бронировании выведется соответствующее сообщение. Для подтверждения осуществления бронирования, программа повторно выведет список свободных мест.

## Руководство программиста

Программа написана на языке C# версии .NET Core 3.1 с помощью среды разработки Microsoft Visual Studio Community 2019 Версии 16.9.5. Среда разработки установлена на операционную систему Windows 10.

Программа не получает данные при запуске. Все необходимые данные вводит пользователь в программе.

Все реализованные классы, их поля и методы находятся в разделе «1.4 Спецификация программы».

## Контрольный пример

На рисунке 1.7.1 показан пример бронирования билета в онлайн кассе с помощью реализованной программы. Пользователь ввел название спектакля Три сестры, на что программа вывела стоимость билета и список свободных мест. Далее было осуществлено бронирование 12 места 1 ряда.

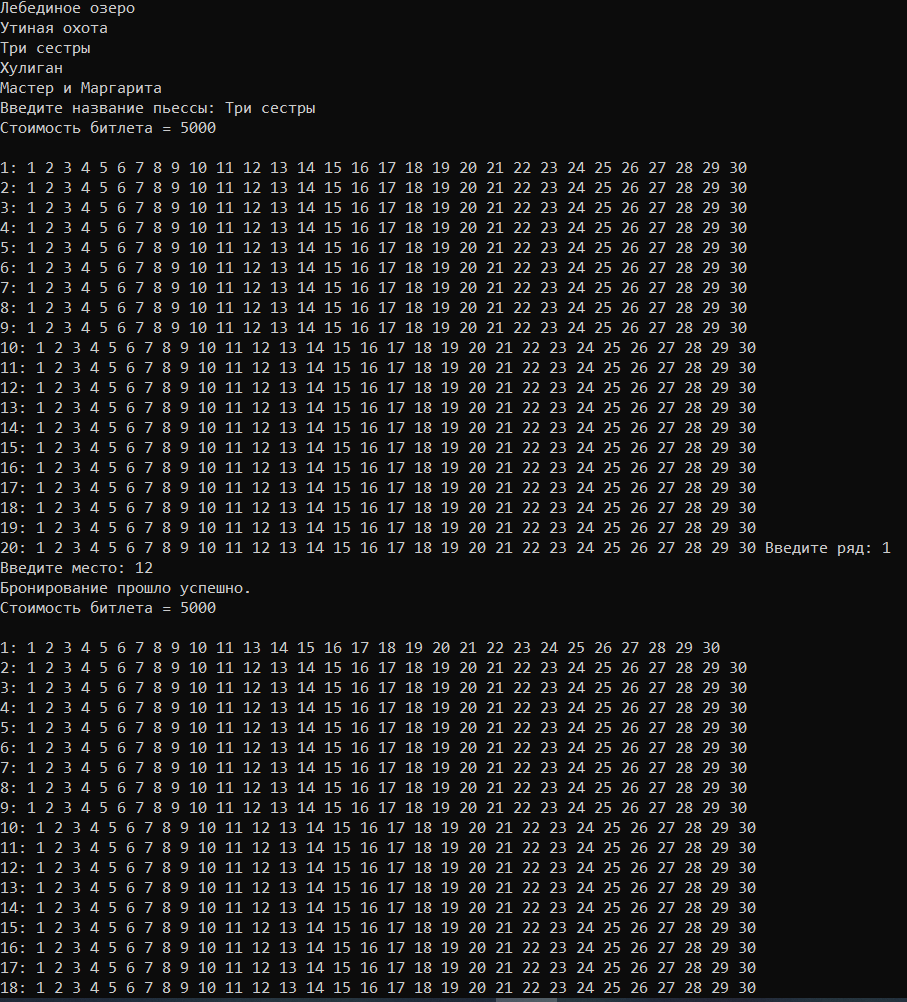


Рис. 1.7.1 — Результат работы программы

## Листинг

Лист. 1.8.1 — Tickets.cs

**using** System**;**

enum ticketType //типы мест

**{**

free**,** //свободное место

booked**,** //забронированное место

paid //оплаченное место

**}**

struct Tickets

**{**

**public** ticketType type**;** //тип билета

**public** **readonly** uint row**;** //ряд

**public** **readonly** uint place**;** //месито

**public** Tickets**(**uint \_row**,** uint \_place**)**

**{**

**this.**place **=** \_place**;**

**this.**row **=** \_row**;**

**this.**type **=** ticketType**.**free**;**

**}**

**}**

Лист. 1.8.2 — Plays.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

class Plays

**{**

**public** uint rowSize**;** //количество рядов

**public** uint placeSize**;** //количество мест

**public** **readonly** float cost**;** //стоимость билета

**public** **readonly** List**<**Tickets**>** TicketsList**;** //список билетов

**public** string PlayName**;** //название пьессы

**public** DateTime startTime**;** //время начала пьессы

**public** Plays**(**string \_PlayName**,** uint \_rowSize**,** uint \_placeSize**,** float \_cost**,** DateTime \_startTime**)**

**{**

rowSize **=** \_rowSize**;**

placeSize **=** \_placeSize**;**

TicketsList **=** **new** List**<**Tickets**>();**

cost **=** \_cost**;**

**for** **(**uint i **=** 0**;** i **<** rowSize**;** i**++)** //заплднение списка билетов

**{**

**for** **(**uint j **=** 0**;** j **<** placeSize**;** j**++)**

**{**

TicketsList**.**Add**(new** Tickets**(**i **+** 1**,** j **+** 1**));**

**}**

**}**

PlayName **=** \_PlayName**;**

startTime **=** \_startTime**;**

**}**

**public** void GiveTicket**(**uint RowToBuy**,** uint PlaceToBuy**,** ticketType whatDo**)** //выдача билета

**{**

**for** **(**int i **=** 0**;** i **<** TicketsList**.**Count**;** i**++)**

**{**

**if** **((**TicketsList**[**i**].**row **==** RowToBuy**)** **&&** **(**TicketsList**[**i**].**place **==** PlaceToBuy**)** **&&** **(**TicketsList**[**i**].**type **==** ticketType**.**free**))**

**{**

Tickets changeTic **=** TicketsList**[**i**];**

changeTic**.**type **=** whatDo**;**

TicketsList**[**i**]** **=** changeTic**;**

**return;**

**}**

**}**

**throw** **new** ArgumentException**(**"Ticket not found"**);** //не удалось найти билет

**}**

**}**

Лист. 1.8.3 — IBank.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**interface** IBank

**{**

**private** static float BankAccount **=** 0**;** //сумма находящаяся в банке

**private** **protected** static void AddToBank**(**float toAdd**)** //пополнение баланса банка

**{**

BankAccount **+=** toAdd**;**

**}**

**}**

Лист. 1.8.4 — Checkout.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**abstract** class Checkout **:** IBank

**{**

**private** List**<**Plays**>** playsList**;** //список пьес

**public** Checkout**(**string**[]** \_playsName**,** uint \_rowSize**,** uint \_placeSize**,** float**[]** \_cost**)**

**{**

playsList **=** **new** List**<**Plays**>();**

**for** **(**int i **=** 0**;** i **<** \_playsName**.**Length**;** i**++)**

**{**

DateTime start **=** **new** DateTime**(**1995**,** 1**,** 1**);**

Random gen **=** **new** Random**();**

int range **=** **((**TimeSpan**)(**DateTime**.**Today **-** start**)).**Days**;**

playsList**.**Add**(new** Plays**(**\_playsName**[**i**],** \_rowSize**,** \_placeSize**,** \_cost**[**i**],** start**.**AddDays**(**gen**.**Next**(**range**))));**

**}**

**}**

**public** void ShowPlays**()** //метод покапзываеющий пьессы

**{**

**for** **(**int i **=** 0**;** i **<** playsList**.**Count**;** i**++)**

**{**

Console**.**WriteLine**(**$"{playsList[i].PlayName}"**);**

**}**

**}**

**public** void BuyTicket**(**string NamePlayToBuy**,** uint RowToBuy**,** uint PlaceToBuy**)** //покупка билетов

**{**

Plays Buy **=** fingPlayByName**(**NamePlayToBuy**);**

Buy**.**GiveTicket**(**RowToBuy**,** PlaceToBuy**,** ticketType**.**paid**);**

IBank**.**AddToBank**(**Buy**.**cost**);** //добавляем стоимость билета в банк

**}**

**private** **protected** Plays fingPlayByName**(**string NamePlay**)** //поиск пьесы по имени

**{**

**for** **(**int i **=** 0**;** i **<** playsList**.**Count**;** i**++)**

**{**

**if** **(**playsList**[**i**].**PlayName **==** NamePlay**)**

**{**

**return** playsList**[**i**];**

**}**

**}**

**throw** **new** ArgumentException**(**"Play not found"**);**

**}**

**public** void ShowFreePlace**(**string NamePlay**)** //метод покапзываеющий свободные места

**{**

Plays Showing **=** fingPlayByName**(**NamePlay**);**

Console**.**WriteLine**(**$"Стоимость битлета = {Showing.cost}"**);**

uint row **=** 0**;**

**for** **(**int i **=** 0**;** i **<** Showing**.**TicketsList**.**Count**;** i**++)**

**{**

**if** **(**row **!=** Showing**.**TicketsList**[**i**].**row**)**

**{**

Console**.**WriteLine**();**

Console**.**Write**(**$"{Showing.TicketsList[i].row}: "**);**

row **=** Showing**.**TicketsList**[**i**].**row**;**

**}**

**if** **(**Showing**.**TicketsList**[**i**].**type **==** ticketType**.**free**)**

**{**

Console**.**Write**(**$"{Showing.TicketsList[i].place} "**);**

**}**

**}**

**}**

**}**

Лист. 1.8.5 — OnlineCheckout.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

class OnlineCheckout **:** Checkout

**{**

**public** **readonly** string Link**;** //ссылка на сайт онлайн кассы

**public** OnlineCheckout**(**string**[]** \_playsName**,** uint \_rowSize**,** uint \_placeSize**,** float**[]** \_cost**,** string \_Link**)** **:** **base(**\_playsName**,** \_rowSize**,** \_placeSize**,** \_cost**)**

**{**

Link **=** \_Link**;**

**}**

**public** void BookTicket**(**string NamePlayToBuy**,** uint RowToBuy**,** uint PlaceToBuy**)** //бронирование билетов

**{**

fingPlayByName**(**NamePlayToBuy**).**GiveTicket**(**RowToBuy**,** PlaceToBuy**,** ticketType**.**booked**);**

**}**

**}**

Лист. 1.8.6 — PhysicalCheckout.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

class PhysicalCheckout **:** Checkout

**{**

**public** **readonly** DateTime TimeToClose**;** //время закрытия

**public** PhysicalCheckout**(**string**[]** \_playsName**,** uint \_rowSize**,** uint \_placeSize**,** float**[]** \_cost**)** **:** **base(**\_playsName**,** \_rowSize**,** \_placeSize**,** \_cost**)**

**{**

TimeToClose **=** **new** DateTime**(**2021**,** 6**,** 1**,** 18**,** 0**,** 0**);**

**}**

**}**

Лист. 1.8.7 — Program.cs

**using** System**;**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

string**[]** NamePlay **=** **{** "Лебединое озеро"**,** "Утиная охота"**,** "Три сестры"**,** "Хулиган"**,** "Мастер и Маргарита" **};** //массив названий пьесс

float**[]** CostPlay **=** **{** 3000**,** 1200**,** 5000**,** 3500**,** 5000 **};** //массив стоимостей билетов

OnlineCheckout Yandex **=** **new** OnlineCheckout**(**NamePlay**,** 20**,** 30**,** CostPlay**,** "https://afisha.yandex.ru/"**);** //создание олайн кассы

PhysicalCheckout Mariinsky **=** **new** PhysicalCheckout**(**NamePlay**,** 20**,** 30**,** CostPlay**);** //создание физической кассы

Yandex**.**ShowPlays**();** //выводим названия пьесс

Console**.**Write**(**"Введите название пьессы: "**);**

string ReadingLine **=** Console**.**ReadLine**();**

Yandex**.**ShowFreePlace**(**ReadingLine**);** //показываем свободные места

Console**.**Write**(**"Введите ряд: "**);**

uint row **=** Convert**.**ToUInt32**(**Console**.**ReadLine**());**

Console**.**Write**(**"Введите место: "**);**

uint place **=** Convert**.**ToUInt32**(**Console**.**ReadLine**());**

Yandex**.**BookTicket**(**ReadingLine**,** row**,** place**);** //бронируем билеты

Console**.**WriteLine**(**"Бронирование прошло успешно."**);**

Yandex**.**ShowFreePlace**(**ReadingLine**);** //показываем изменения

Console**.**ReadKey**();**

**}**

**}**

# Второй раздел. Гамильтоновы циклы

## Задание

Задачу решить в общем виде. В качестве контрольного примера использовать граф, представленный на рисунке 2.1.1. Найти все гамильтоновы циклы графа.

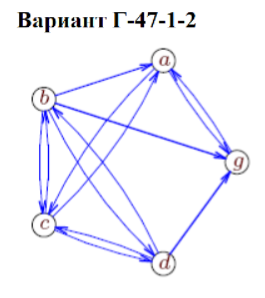


Рисунок 2.1.1 — задание для контрольного примера

## Теоретический аспект

Гамильтонов путь — простой путь, проходящий через все вершины графа ровно один раз. Гамильтонов цикл — путь, конечная и начальная вершины которого совпадают. Гамильтоновым графом является граф, содержащий гамильтонов цикл.

Для не ориентированных графов применимы теоремы Дирака и Оре. Теорема Дирака утверждает, что если в графе больше двух вершин и степени всех вершин больше n/2, где n — количество вершин, то граф Гамильтонов. Теорема Оре утверждает, что что если в графе больше двух вершин, и сумма степеней любых двух не смежных вершин больше или равна n, то граф Гамильтонов.

Существует несколько способов поиска гамильтоновых циклов. К ним относится перебор всех перестановок и динамическое программирование по подмножествам.

## Формализация задачи

Информацией, поступающей на вход является путь к файлу и сам файл, в котором записана матрица смежности графа.

Блок-схема алгоритма поиска Гамильтонова пути представлена на рисунке 2.3.1.

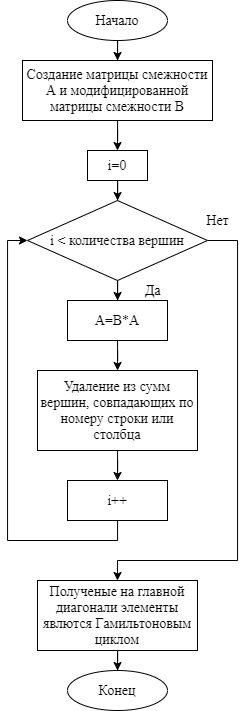


Рисунок 2.3.1 — Диаграмма деятельности

Взаимосвязь и деятельность классов представлена на рисунке 2.3.2

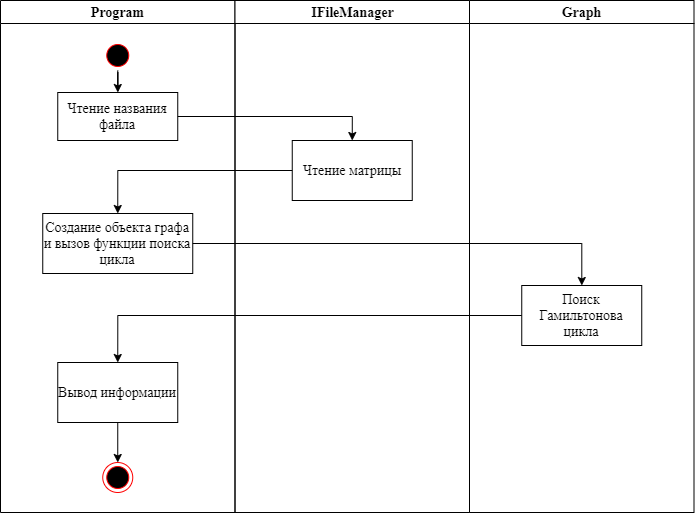


Рисунок 2.3.2 — Диаграмма деятельности

Классы:

* Graph — описывает граф и выполняет поиск Гамильтоново цикла.
* Program — класс, содержащий метод Main, с которого начинает работать программа.
* FileManager — необходим для сохранения информации в файл и чтения информации из файла

Важным аспектом задачи является обработка исключений некорректного ввода и ошибок чтения файла. Также для удобного тестирования используются модульные тесты.

## Спецификация программы

Ниже представлены таблицы классов, реализованных в программе.

Класс Program

Таблица 2.1 — Описание методов класса Program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Main | Начало работы программы | - | static | string[] |

Класс FileManager

Таблица 2.2 — Описание методов класса FileManager

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| ReadMatrix | Чтение матрицы из файла | Int[,] | public static | Имя файла |
| WriteString | Запись строки в файл | Void | public static | Имя файла, сообщение |

Класс Graph r

Таблица 2.3 — Описание методов класса Graph

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Graph | Заполнение полей класса | - | public | Матрица смежности |
| GamAlg | Поиск Гамильтоновых циклов | string | public | - |
| Mupl | Умножение матриц, состоящих из строк | string[,] | private | Матрицы для умножения |

Таблица 2.4 — Описание полей класса Graph

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| GraphMatrix | int[,] | public readonly | Матрица смежности |
| GraphSize | int | public readonly | Количество вершин |

## Руководство пользователя

Программа предназначена для поиска Гамильтонова цикла по матрице смежности, заданной в файле.

Сначала программа попросит ввести имя файла, в котором записана матрица смежности (рисунок 2.5.1).

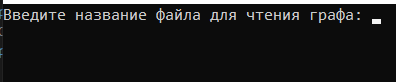


Рисунок 2.5.1 — Программа после запуска

При не правильном вводе или не найденном файле, программа выдаст соответствующее сообщение и прекратит работу. Для выполнения чтения, файл должен находится в папке где находится exe файл программы.

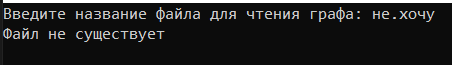


Рисунок 2.5.2 — Ошибка не найденного файла

Информация в файле должна выглядеть следующим образом: в первой строке записано количество вершин в матрице. В последующих строках матрица смежности. Значения рёбер разделены пробелом. Пример входного файла на рисунке

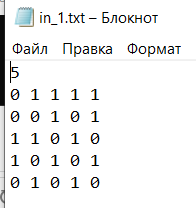


Рисунок 2.5.3 — Пример данных в файле

Если все данные введены правильно, программа выдаст последовательность вершин, соответствующая Гамильтонову циклу.



Рисунок 2.5.4 — Корректная работа программы

Номера вершин в выводи начинаются с 0.

## Руководство программиста

Программа написана на языке C# версии .NET Core 3.1 с помощью среды разработки Microsoft Visual Studio Community 2019 Версии 16.9.5. Среда разработки установлена на операционную систему Windows 10.

Программа не получает данные при запуске. Все необходимые данные вводит пользователь в программе.

Все реализованные классы, их поля и методы находятся в разделе «3.4 Спецификация программы».

В программе реализованы юнит тесты, проверяющие обработку исключений и корректность выводимых данных.

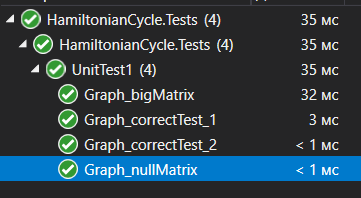


Рисунок 2.6.1 — Выполненные тесты

## Контрольный пример

В качестве контрольного примера выступает граф, данный в задании (Рисунок 2.1.1). Содержимое файла, из которого будет осуществляться чтение изображен на рисунке 2.7.1.

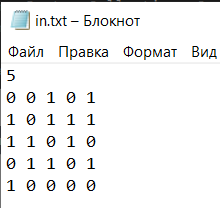


Рисунок 2.7.1 — Матрица смежности

Вывод гамильтоного цикла после запуска программы изображен на рисунке 2.7.2.

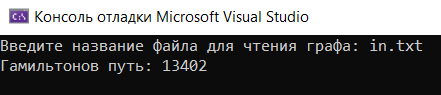


Рисунок 2.7.2 — Работа программы

## Листинг

Листинг 2.8.1 — FileManager.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**using** System**.**IO**;**

**namespace** HamiltonianСycle

**{**

static class FileManager

**{**

**public** static int**[,]** ReadMatrix**(**string fileName**)**

**{**

**if** **(!**File**.**Exists**(**fileName**))**

**throw** **new** FileNotFoundException**(**"Файл не существует"**);**

StreamReader file **=** **new** StreamReader**(**fileName**);** //открываем поток для чтения

**try**

**{**

int size **=** Convert**.**ToInt32**(**file**.**ReadLine**());** //размер матрицы

**if** **(**size **<=** 0**)**

**{**

**throw** **new** ArgumentOutOfRangeException**(**"Не корректный размер матрицы"**);**

**}**

int**[,]** matrix **=** **new** int**[**size**,** size**];**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)** //считываем элементы матрицы

**{**

string line **=** file**.**ReadLine**();**

line **=** line**.**Trim**();**

string**[]** bits **=** line**.**Split**(**' '**);**

**for** **(**int j **=** 0**;** j **<** size**;** j**++)**

matrix**[**i**,** j**]** **=** Convert**.**ToInt32**(**bits**[**j**]);**

**}**

file**.**Close**();** //закрытие файла

**return** matrix**;**

**}**

**catch** **(**Exception e**)**

**{**

Console**.**WriteLine**(**e**.**Message**);**

**return** **null;**

**}**

**}**

**public** static void WriteString**(**string fileName**,** string line**)**

**{**

**if** **(!**File**.**Exists**(**fileName**))**

**throw** **new** FileNotFoundException**(**"Файл не существует"**);**

StreamWriter file = new StreamWriter(fileName); //открытие потока для записи

file.WriteLine(line); //запись

file.Close(); //закрытие файла

}

}

}

Листинг 2.8.2 — Program.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**namespace** HamiltonianСycle

**{**

class Program

**{**

static void Main**(**string**[]** args**)**

**{**

**try**

**{**

Console**.**Write**(**"Введите название файла для чтения графа: "**);**

string fileName **=** Console**.**ReadLine**();** //чтение имени файла

int**[,]** matrix **=** FileManager**.**ReadMatrix**(**fileName**);** //получение матрицы графа

Graph graph **=** **new** Graph**(**matrix**);** //задание матрицы

Console**.**WriteLine**(**"Гамильтонов путь: " **+** graph**.**GamAlg**());** //вывод результата

**}**

**catch** **(**Exception e**)**

**{**

Console**.**WriteLine**(**e**.**Message**);** //вывод ошибок

**}**

**}**

**}**

**}**

Листинг 2.8.3 — Graph.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** HamiltonianСycle

**{**

**public** class Graph

**{**

**public** **readonly** int**[,]** GraphMatrix**;**

**public** **readonly** int GraphSize**;**

**public** Graph**(**int**[,]** matrix**)** //задание матрицей

**{**

**if** **(**matrix **==** **null)**

**{**

**throw** **new** ArgumentNullException**(**"Ошибка передаваемой матрицы"**);**

**}**

**if** **(**matrix**.**GetLength**(**0**)>**10**)**

**{**

**throw** **new** ArgumentOutOfRangeException**(**"Задана слишком большая матрица"**);**

**}**

GraphSize **=** matrix**.**GetLength**(**0**);**

GraphMatrix **=** **new** int**[**GraphSize**,** GraphSize**];**

**for** **(**int i **=** 0**;** i **<** GraphSize**;** i**++)**

**for** **(**int j **=** 0**;** j **<** GraphSize**;** j**++)**

GraphMatrix**[**i**,** j**]** **=** matrix**[**i**,** j**];** //копирование матрицы

**}**

**public** string GamAlg**()** //поиск Гамильтоновых циклов

**{**

string**[,]** A **=** **new** string**[**GraphSize**,** GraphSize**];** //матрица смежности

string**[,]** B **=** **new** string**[**GraphSize**,** GraphSize**];** //модифицированная матрица смежности

**for** **(**int i **=** 0**;** i **<** GraphSize**;** i**++)**

**{**

**for** **(**int j **=** 0**;** j **<** GraphSize**;** j**++)**

**{**

**if** **(**GraphMatrix**[**i**,** j**]** **==** 0**)** //преобразование исходной матрицы

**{**

A**[**i**,** j**]** **=** "F"**;** //пустой символ

B**[**i**,** j**]** **=** "F"**;**

**}**

**else**

**{**

A**[**i**,** j**]** **=** Convert**.**ToString**(**GraphMatrix**[**i**,** j**]);**

B**[**i**,** j**]** **=** "+" **+** Convert**.**ToString**(**j**);**

**}**

**}**

}

string[,] res = new string[GraphSize, GraphSize];

for (int i = 0; i < GraphSize; i++) //первое перемножение матриц

{

for (int j = 0; j < GraphSize; j++)

{

res[i, j] = "";

for (int k = 0; k < GraphSize; k++)

{

if (A[k, j] == "1")

{

if (B[i, k] != "F") // не пустой символ

{

res[i, j] += B[i, k];

}

}

}

if (res[i, j] == "") //не произведены операции

res[i, j] = "F";

}

}

A = res;

for (int j = 0; j < GraphSize; j++) //обнуление главной диагонали

A[j, j] = "F";

for (int k = 2; k < GraphSize; k++)

{

A = Mupl(B, A);

}

for (int i = 0; i < GraphSize; i++)

{

if (A[i, i] != "F") //поиск ячейки с результатом на главной диагонали

{

return Convert.ToString(i) + A[i, i].Substring(1, GraphSize - 1); //вырезаем путь

}

}

return null; //не удалось найти гамильтонов цикл

}

private string[,] Mupl(string[,] Arr1, string[,] Arr2) //перемножение матриц вершин

{

string[,] res = new string[GraphSize, GraphSize];

for (int i = 0; i < GraphSize; i++)

{

for (int j = 0; j < GraphSize; j++)

{

res[i, j] = "";

for (int k = 0; k < GraphSize; k++)

{

/\*ячейки не пустые и не содержат вершин i и j\*/

if ((Arr1[i, k] != "F") && (Arr2[k, j] != "F") && !Arr1[i, k].Contains(Convert.ToString(i)) && !Arr1[i, k].Contains(Convert.ToString(j)) && !Arr2[k, j].Contains(Convert.ToString(i)) && !Arr2[k, j].Contains(Convert.ToString(j)))

{

res[i, j] += "+" + Arr2[k, j].Replace("+", Arr1[i, k]);

res[i, j]=res[i, j].Replace("++", "+"); //удаление лишнего +

}

}

if (res[i, j] == "") //не произведены операции

res[i, j] = "F";

}

}

return res;

}

}

}

# Третий раздел. Имитационное моделирование

## Задание

Вариант 25

Система поддержки бронирования и заселения гостиницы

Небольшая гостиница содержит K номеров (20≤ K ≤ 30), которые различаются степенью комфорта, максимальным количеством человек, которые могут поселится в номере, а, следовательно, и стоимостью. Названия номеров традиционное: «люкс», «полулюкс», одноместные, простые двухместные, двухместные с раскладным диваном и т. д. (например, 70 у.е. за день проживания в одноместном номере, 120 у.е. – за номер «люкс»).

Требуется создать компьютерную систему, автоматизирующую управление занятостью номеров гостиницы. Система должна обрабатывать входной поток заявок двух видов:

• заявки, бронирующие определенные типы номеров на определенный срок;

• заявки на заселение в текущий момент.

Система должна хранить информацию о фактической занятости всех номеров и о их занятости в ближайшие дни (учитываются уже оплаченные вперед дни), а также сведения о произведенной брони номеров, и должна использовать все эти данные при обработке заявок. При бронировании номеров система должна автоматически формировать сообщение-подтверждение брони, а при выезде постояльцев она оформляет им счета.

Стратегия обработки заявок строится так, чтобы добиться максимальной занятости гостиницы с целью увеличения ее прибыли. Для этого система гибко распоряжается номерным фондом: в частности, при нехватке нужных номеров можно использовать пустующие номера большей комфортности (по меньшей цене), например, при нехватке одноместных номеров можно поселить одного человека в двухместный номер (за 70% его стоимости).

Для тестирования построенной системы необходимо смоделировать входной поток заявок на бронирование и поселение. Вид и параметры каждой заявки определяются случайным образом. Интервал между появлением двух заявок следует моделировать как случайную величину из определенного диапазона (например, от 1 до 5 часов).

Период моделирования – М дней (12<М<30), шаг – несколько часов.

Цель моделирования – изучение стратегий обработки заявок на заселение. В параметры моделирования следует включить: числа K и М, количество номеров каждой категории, характеристики используемых случайных величин.

В ходе моделирования система должна предоставлять всю необходимую информацию о занятости номеров гостиницы. По окончании моделирования должна выводится статистика заселения номеров, выполненных заявок, процент загруженности отдельных категорий номеров и гостиницы в целом.

## Теоретический аспект

Гостиница — дом с меблированными комнатами для приезжающих. Заселение в гостинцу происходит по заявкам. Также, в гостиницах есть возможность забронировать номер заранее на определённы срок. Для осуществления этих операций необходимо указать количество человек в номере и срок проживания.

## Формализация задачи

Исходными данными являются заявки на бронирование и заселение, включающие даты проживания и число людей. Программа должна уметь обрабатывать заявки, выводить даты занятости номеров и пополнение банковского счета.

Имитационная модель гостиницы включает в себя систему подачи заявок со стороны формы, их обработки в классе гостиницы и пополнение баланса

Диаграмма используемых классов представлена на рисунке 3.3.1

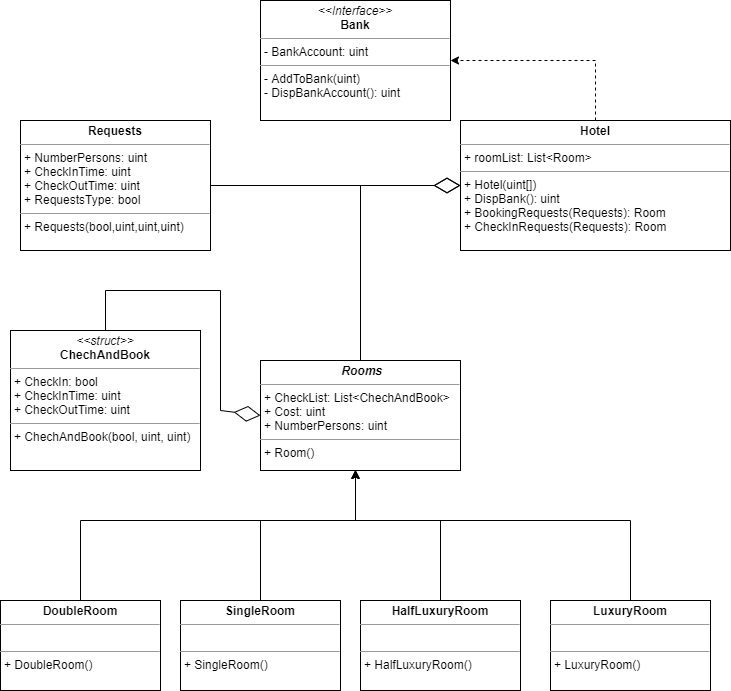


Рисунок 3.3.1 — UML диаграмма классов, необходимых для обработки заявок в гостинице

Структуры:

* ChechAndBook — описывает время и тип операций над номерами.

Классы:

* abstract Room — описывает общий вид номеров.
* LuxuryRoom — описывает номера люкс класса и наследуется от Room.
* HalfLuxuryRoom — описывает номера класса полулюкс и наследуется от Room.
* SingleRoom — описывает одноместные номера и наследуется от Room.
* DoubleRoom— описывает двухместные номера и наследуется от Room.
* Requests — описывает заявки.
* Hotel — описывает гостиницу и обрабатывает заявки разных типов. Наследует интерфейс IBank.
* FileManipulation — необходим для сохранения информации в файл

Интерфейс:

* IBank — хранит банковский счет и пополняет его

У каждого гостиничного номера есть список, отвечающий за промежутки занятости, то есть за время, когда в нем кто-то проживает. Изображение этих промежутков представлено на рисунке \*\*. Основной алгоритм заключается в поиске соответствующего для вставки в этот список места. Все элементы в списке должны оставаться отсортированы по времени начала заселения

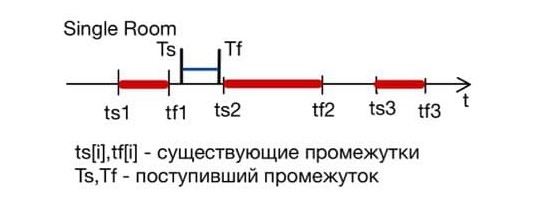


Рисунок 3.3.2 — схема промежутков времени проживания в номере

Блок-схема, отображающая алгоритм проверки возможности вставки поступившего промежутка заселения представлен на рисунке 3.3.2

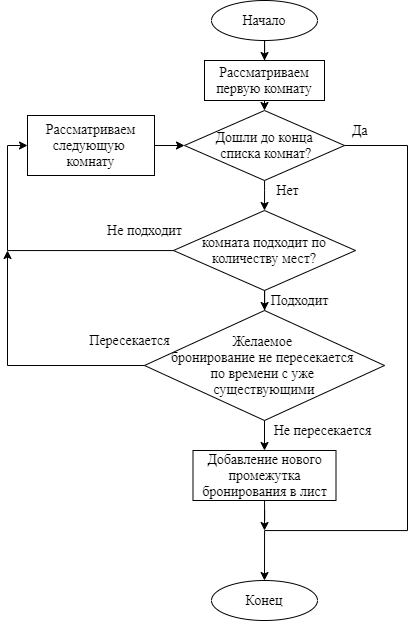


Рисунок 3.3.3 — схема поиска номера для бронирования

В программе реализованы принципы объектно-ориентированного программирования. Каждый класс выполняет определённую задачу, будь то хранение информации об объекте или осуществление сохранения данных в файл. В программу можно легко добавить номера других видов, что поддерживает принцип открытости-закрытости. Также, вместо общего класса Room, в заселении могут участвовать только подклассы. Это соответствует принципу подстановки. Класс Hotel может работать и без интерфейса IBank. Помимо этого, абстрактный класс Room не зависит от своих наследователей и может работать без них.

## Спецификация программы

Ниже представлены таблицы классов, структур интерфейсов, реализованных в программе.

Класс Room

Таблица 3.1 — Описание методов класса Room

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Room | Создание элемента класса | - | Public | - |

Таблица 3.2 — Описание полей класса Room

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| CheckList | List<ChechAndBook> | public | Брони |
| NumberPersons | Uint | Public | Вместимость |
| Cost | Uint | Public | Стоимость проживания |

Интерфейс IBank

Таблица 3.3 — Описание методов интерфейса IBank

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| AddToBank | Пополнение баланса банка | void | private protected static | Сумма к добавлению |
| DispBankAccount | Отображение баланса аккаунта | uint | private protected static | - |

Таблица 3.4 — Описание полей интерфейса IBank

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| BankAccount | uint | private static | Сумма в банке |

Класс Hotel

Таблица 3.5 — Описание методов класса Hotel

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Hotel | Создание элемента класса | - | Public | Количество номеров каждого типа |
| BookingRequests | Обработка заявки на бронирование | Room | Public | Заявка |
| CheckInRequests | Обработка заявки на заселение | Room | Public | Заявка |

Таблица 3.6 — Описание полей класса Hotel

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| roomList | List<Room> | Public readonly | Список номеров |

Структура ChechAndBook

Таблица 3.7 — Описание методов структуры ChechAndBook

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| ChechAndBook | Присваивание значений полей | - | Public | Тип заявки, время проживания |

Таблица 3.8 — Описание полей структуры ChechAndBook

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| CheckIn | bool | public | Тип заявки |
| CheckInTime | uint | public | Время заселения |
| CheckOutTime | uint | public | Время выезда |

Класс Requests

Таблица 3.9 — Описание методов класса Requests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Requests | Создание элемента класса | - | Public | Тип заявки, время проживания, количество человек |
| Requests | Создание элемента класса с случайно заданными значениями полей | - | public | Время заявки |

Таблица 3.10 — Описание полей класса Requests

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор доступа | Назначение |
| RequestsType | bool | public | Тип заявки |
| CheckInTime | uint | public | Время заселения |
| CheckOutTime | uint | public | Время выезда |
| NumberPersons | uint | public | Количество человек |

Класс FileManipulation

Таблица 3.11 — Описание методов класса Room

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| SaveToFile | Сохранение информации в файл | void | public static | Объект гостиницы, путь к файлу |

Описание методов и полей LuxuryRoom, HalfLuxuryRoom, SingleRoom и DoubleRoom наследуете от Room.

## Руководство пользователя

Для корректной работы программы, её необходимо запускать на операционной системе Windows. Минимальный объем оперативной памяти 40 Мбайт. Минимальный объем свободного места на диске 200 Кбайт.

Программа предназначена для имитации обработки заявок на заселение и бронирование номеров в гостинице. Пользовательский интерфейс представлен на рисунке 3.5.1.

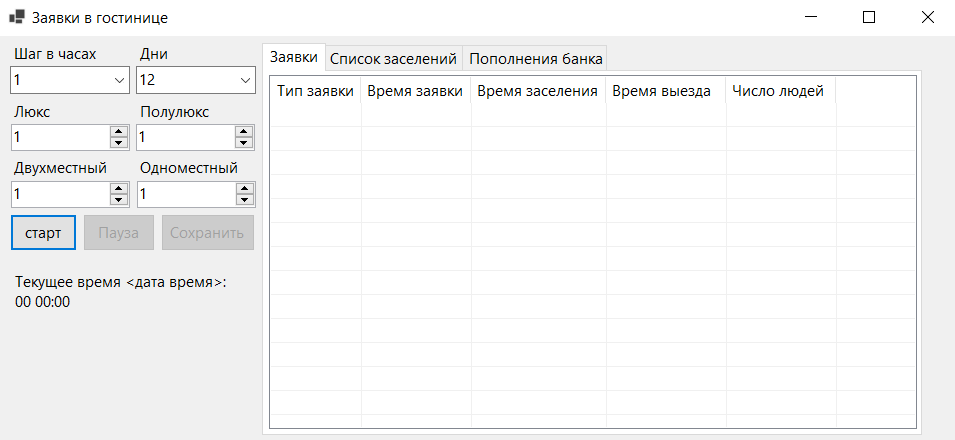


Рисунок 3.5.1 — Форма до запуска имитации

Перед началом работы пользователь может выбрать шаг от 1 до 5 часов и последний день имитации от 12 до 30. Также, для успешной имитации необходимо задать количество номеров разного типа. После нажатия на кнопку «старт», запустится таймер. Текущее время имитации будет отображено в соответствующей графе.

После запуска имитации, пользователь может переключаться между вкладками на панели. На первой вкладке отображаются заявки, а именно их время поступления, тип, даты проживания и количество человек. На второй вкладке отображаются успешно обработанные заявки, какой тип номера и на какие даты занят. В последней вкладке отображается сумма на банковском аккаунте после совершения успешного заселения и время совершения этой операции.

Помимо этого, при нажатии на кнопку «Пауза», работа имитации останавливается. После повторного нажатия на кнопку, имитация продолжит работу с точки остановки.

Для сохранения текущей информации о номерах и заселении, необходимо нажать на кнопку «Сохранить». После этого, в папке где находится exe файл программы, создастся файл «save.txt» в который будет записана вся информация.

## Руководство программиста

Программа написана на языке C# версии .NET Core 3.1 с помощью среды разработки Microsoft Visual Studio Community 2019 Версии 16.9.5. Среда разработки установлена на операционную систему Windows 10.

Программа не получает данные при запуске. Все необходимые данные вводит пользователь в форме.

Все реализованные классы, их поля и методы находятся в разделе «3.4 Спецификация программы».

## Контрольный пример

На рисунке 3.7.1 показано появление заявок при шаге иммитации в 1 час. Номеров люкс класса 10, полулюкс 3, двухместных 3, одноместных 1.

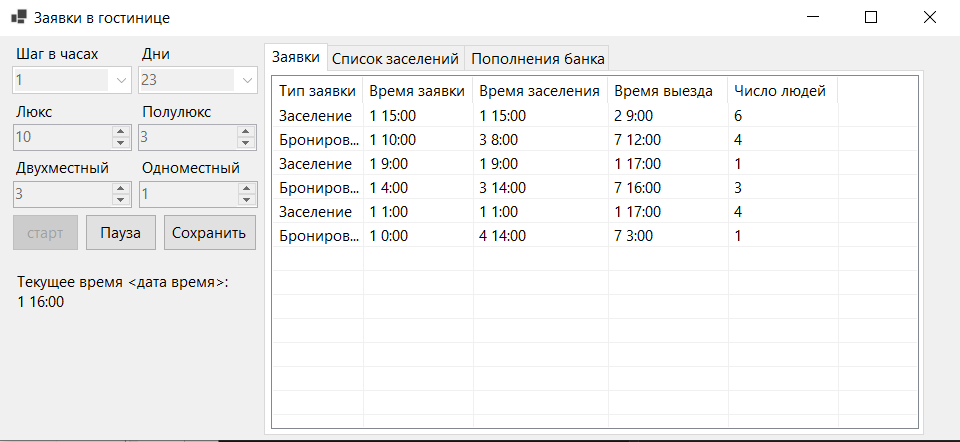


Рисунок 3.7.1 — Появление заявок

На рисунке 3.7.2 показан список принятых заявок при тех же начальных условиях.

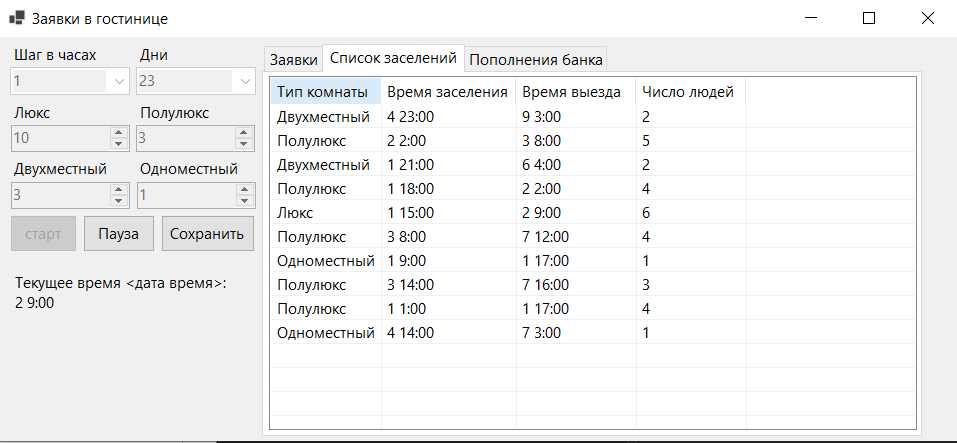


Рисунок 3.7.2 — Появление заселённых номеров

На рисунке 3.7.3 показано состояние после обновлений банковского аккаунта.

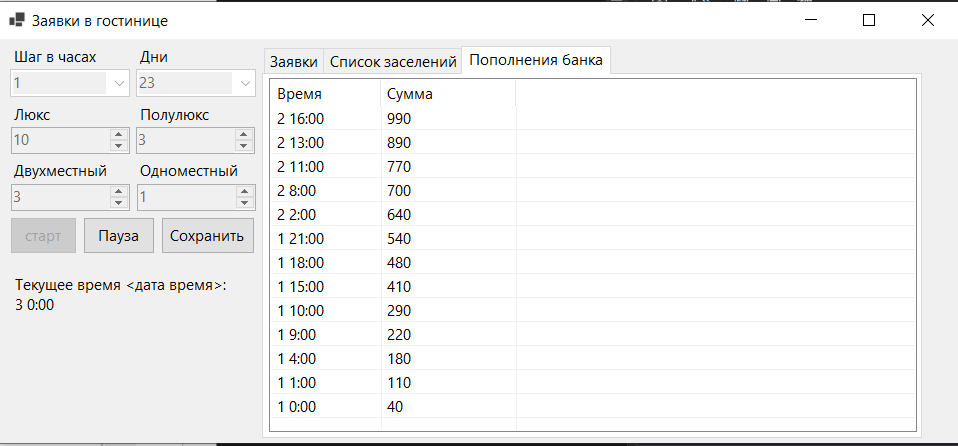


Рисунок 3.7.3 — Состояние банковского счета

На рисунке 3.7.4 показана информация, записанная в файл при нажатии на кнопку «Сохранить».

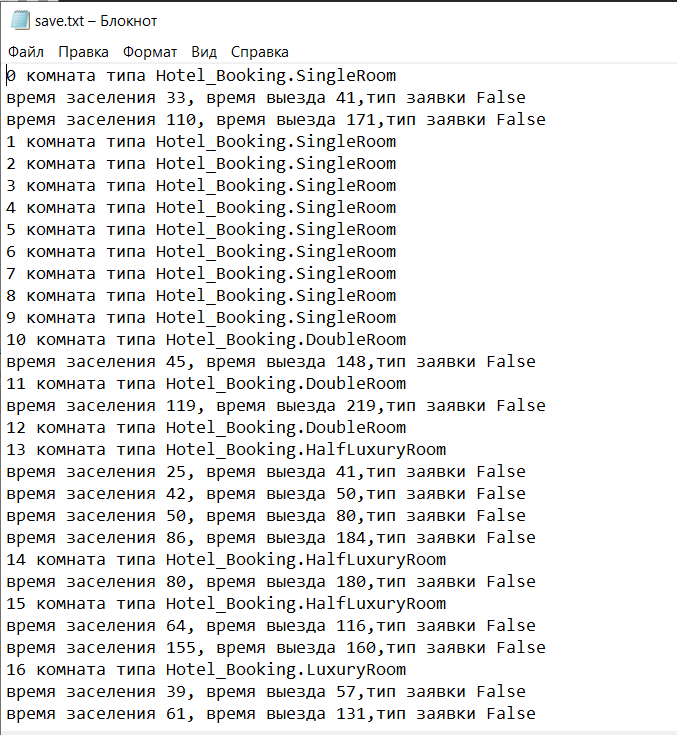


Рисунок 3.7.4 — Информация в файле

## Листинг

Листинг 3.8.1 — IBank.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**namespace** Hotel\_Booking

**{**

**interface** IBank

**{**

**private** static uint BankAccount **=** 0**;** //сумма находящаяся в банке

**private** **protected** static void AddToBank**(**uint toAdd**)** //пополнение баланса банка

**{**

BankAccount **+=** toAdd**;**

**}**

**public** static uint DispBankAccount**()** //отображение баланса

**{**

**return** BankAccount**;**

**}**

**}**

**}**

Листинг 3.8.2 — Requests.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Hotel\_Booking

**{**

class Requests

**{**

**public** uint NumberPersons **{** **get;** **protected** **set;** **}** // количество человек вв номере

**public** uint CheckInTime **{** **get;** **protected** **set;** **}** //дата заселения

**public** uint CheckOutTime **{** **get;** **protected** **set;** **}** //дата выселения

**public** bool RequestsType **{** **get;** **protected** **set;** **}** //тип заявки false -- бронированиек, true -- pfctktybt

**public** Requests**(**bool \_RequestsType**,**uint \_NumberPersons**,** uint \_CheckInTime**,** uint \_CheckOutTime**)**

**{**

RequestsType **=** \_RequestsType**;**

NumberPersons **=** \_NumberPersons**;**

CheckInTime **=** \_CheckInTime**;**

CheckOutTime **=** \_CheckOutTime**;**

**}**

**public** Requests**(**uint \_time**)** //заявка с случайно заданными полями

**{**

Random randT **=** **new** Random**();**

Random randB **=** **new** Random**();**

Random randP **=** **new** Random**();**

Random randType **=** **new** Random**();**

uint deltaT **=** Convert**.**ToUInt32**(**randT**.**Next**(**0**,** int**.**MaxValue**)** **%** 100**)** **+** 5**;** //длительность проживания

uint deltaB **=** Convert**.**ToUInt32**(**randB**.**Next**(**0**,** int**.**MaxValue**)** **%** 100**)** **+** 1**;** //время заселения

NumberPersons **=** Convert**.**ToUInt32**(**randP**.**Next**(**0**,** int**.**MaxValue**)** **%** 6**)** **+** 1**;** // количество человек

uint deltaType **=** Convert**.**ToUInt32**(**randType**.**Next**(**0**,** int**.**MaxValue**)** **%** 2**);** // количество человек

**if** **(**deltaType **==** 0**)**

**{**

RequestsType **=** **false;**

CheckInTime **=** \_time **+** deltaB**;**

CheckOutTime **=** \_time **+** deltaT **+** deltaB**;**

**}**

**else**

**{**

RequestsType **=** **true;**

CheckInTime **=** \_time**;**

CheckOutTime **=** \_time **+** deltaT**;**

**}**

**}**

**}**

**}**

Листинг 3.8.3 — Room.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Hotel\_Booking

**{**

**abstract** class Room

**{**

**public** List**<**ChechAndBook**>** CheckList**;**

**public** uint Cost **{** **get;** **protected** **set;** **}** // стоимость

**public** uint NumberPersons **{** **get;** **protected** **set;** **}** // количество человек в номере

**public** Room**()**

**{**

CheckList **=** **new** List**<**ChechAndBook**>();**

Cost **=** 0**;**

NumberPersons **=** 0**;**

**}**

**}**

**}**

Листинг 3.8.4 — RoomHeirs.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Hotel\_Booking

**{**

struct ChechAndBook

**{**

**public** bool CheckIn**;** //false -- бронь, true -- заселён

**public** uint CheckInTime **{** **get;** **set;** **}** //дата заселения

**public** uint CheckOutTime **{** **get;** **set;** **}** //дата выселения

**public** ChechAndBook**(**bool \_CheckIn**,** uint \_CheckInTime**,** uint \_CheckOutTime**)**

**{**

CheckIn **=** \_CheckIn**;**

CheckInTime **=** \_CheckInTime**;**

CheckOutTime **=** \_CheckOutTime**;**

**}**

**}**

class LuxuryRoom **:** Room

**{**

**public** LuxuryRoom**()** **:** **base()**

**{**

Cost **=** 120**;** //назначение стоимости номера

NumberPersons **=** 6**;** //назначение количества человек в номере

**}**

**}**

class HalfLuxuryRoom **:** Room

**{**

**public** HalfLuxuryRoom**()** **:** **base()**

**{**

Cost **=** 100**;** //назначение стоимости номера

NumberPersons **=** 5**;** //назначение количества человек в номере

**}**

**}**

class SingleRoom **:** Room

**{**

**public** SingleRoom**()** **:** **base()**

**{**

Cost **=** 40**;** //назначение стоимости номера

NumberPersons **=** 1**;** //назначение количества человек в номере

**}**

**}**

class DoubleRoom **:** Room

**{**

**public** DoubleRoom**()** **:** **base()**

**{**

Cost **=** 60**;** //назначение стоимости номера

NumberPersons **=** 2**;** //назначение количества человек в номере

**}**

**}**

**}**

Листинг 3.8.5 — FileManipulation.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**using** System**.**IO**;**

**namespace** Hotel\_Booking

**{**

class FileManipulation

**{**

**public** static void SaveToFile**(**Hotel hotel**,** string filename**)**

**{**

StreamWriter file **=** **new** StreamWriter**(**filename**);** //открытие файла для записи

**for** **(**int i **=** 0**;** i **<** hotel**.**roomList**.**Count**;** i**++)** //идём по всем комнатам

**{**

file**.**WriteLine**(**$"{i} комната типа {hotel.roomList[i].GetType().ToString()}"**);**

**for** **(**int j **=** 0**;** j **<** hotel**.**roomList**[**i**].**CheckList**.**Count**;** j**++)** //записываем все забронированные промежутки

**{**

file**.**WriteLine**(**$"время заселения {hotel.roomList[i].CheckList[j].CheckInTime}, время выезда {hotel.roomList[i].CheckList[j].CheckOutTime},тип заявки {hotel.roomList[i].CheckList[j].CheckIn}"**);**

**}**

**}**

file**.**Close**();** //закрытие файла

**}**

**}**

**}**

Листинг 3.8.6 — Hotel.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**using** System**.**IO**;**

**namespace** Hotel\_Booking

**{**

class Hotel **:** IBank

**{**

**public** **readonly** List**<**Room**>** roomList**;** //список комнат

**public** Hotel**(**uint**[]** countRoom**)**

**{**

roomList **=** **new** List**<**Room**>();**

**for** **(**int i **=** 0**;** i **<** countRoom**[**0**];** i**++)** //0 - одиночные, 1 - двойные, 2 - полулюкс, 3 - люкс

roomList**.**Add**(new** SingleRoom**());**

**for** **(**int i **=** 0**;** i **<** countRoom**[**1**];** i**++)**

roomList**.**Add**(new** DoubleRoom**());**

**for** **(**int i **=** 0**;** i **<** countRoom**[**2**];** i**++)**

roomList**.**Add**(new** HalfLuxuryRoom**());**

**for** **(**int i **=** 0**;** i **<** countRoom**[**3**];** i**++)**

roomList**.**Add**(new** LuxuryRoom**());**

**}**

**public** Room BookingRequests**(**Requests req**)**

**{**

**for** **(**int i **=** 0**;** i **<** roomList**.**Count**;** i**++)** //проверка всех комнат

**{**

**if** **(**req**.**NumberPersons **<=** roomList**[**i**].**NumberPersons**)** //проверка вместимости

**{**

**if** **(**roomList**[**i**].**CheckList**.**Count **==** 0**)** //заселений ещё нет

**{**

ChechAndBook toAdd **=** **new** ChechAndBook**(false,**req**.**CheckInTime**,**req**.**CheckOutTime**);** //добавляем новое заселение

roomList**[**i**].**CheckList**.**Add**(**toAdd**);**

**if** **(**req**.**NumberPersons **==** roomList**[**i**].**NumberPersons**)** //если число людей совпадает с размером комнаты

**{**

IBank**.**AddToBank**(**roomList**[**i**].**Cost**);** //полная стоимость

**}**

**else**

**{**

IBank**.**AddToBank**(**Convert**.**ToUInt32**(**roomList**[**i**].**Cost**\***0.7**));** //70% стоимости

**}**

**return** roomList**[**i**];**

**}**

**else**

**{**

int j**;**

uint Ts **=** req**.**CheckInTime**,** Tf **=** req**.**CheckOutTime**;**

bool ifCan **=** **true;**

**for** **(**j **=** 0**;** **(**j **<** roomList**[**i**].**CheckList**.**Count**)** **&&** **!(**Tf **<** roomList**[**i**].**CheckList**[**j**].**CheckInTime**);** j**++)**

**if** **(((**Ts **<** roomList**[**i**].**CheckList**[**j**].**CheckInTime**)** **&&** **(**Tf **>** roomList**[**i**].**CheckList**[**j**].**CheckOutTime**))** **||**

**((**roomList**[**i**].**CheckList**[**j**].**CheckInTime **<** Ts**)** **&&** **(**Ts **<** roomList**[**i**].**CheckList**[**j**].**CheckOutTime**))** **||**

**((**roomList**[**i**].**CheckList**[**j**].**CheckInTime **<** Tf**)** **&&** **(**Tf **<** roomList**[**i**].**CheckList**[**j**].**CheckOutTime**)))**

**{**

ifCan **=** **false;** //временные отрезки пересекаются

**break;**

**}**

**if** **(!**ifCan**)** //если пересекаются

**continue;**

**if** **((**j **==** roomList**[**i**].**CheckList**.**Count**)** **&&** **(**Ts **>** roomList**[**i**].**CheckList**[**j**-**1**].**CheckInTime**))**

**{**

ChechAndBook toAdd **=** **new** ChechAndBook**(false,** req**.**CheckInTime**,** req**.**CheckOutTime**);**

roomList**[**i**].**CheckList**.**Add**(**toAdd**);** //добавляем новое заселение

**if** **(**req**.**NumberPersons **==** roomList**[**i**].**NumberPersons**)** //если число людей совпадает с размером комнаты

**{**

IBank**.**AddToBank**(**roomList**[**i**].**Cost**);** //полная стоимость

**}**

**else**

**{**

IBank**.**AddToBank**(**Convert**.**ToUInt32**(**roomList**[**i**].**Cost **\*** 0.7**));** //70% стоимости

**}**

**}**

**else**

**{**

ChechAndBook toAdd **=** **new** ChechAndBook**(false,** req**.**CheckInTime**,** req**.**CheckOutTime**);**

roomList**[**i**].**CheckList**.**Insert**(**j**,** toAdd**);** //добавляем новое заселение на место j

**if** **(**req**.**NumberPersons **==** roomList**[**i**].**NumberPersons**)** //если число людей совпадает с размером комнаты

**{**

IBank**.**AddToBank**(**roomList**[**i**].**Cost**);** //полная стоимость

**}**

**else**

**{**

IBank**.**AddToBank**(**Convert**.**ToUInt32**(**roomList**[**i**].**Cost **\*** 0.7**));** //70% стоимости

**}**

**}**

**return** roomList**[**i**];**

**}**

**}**

**}**

**return** **null;** //не получилось заселить

**}**

**public** Room CheckInRequests**(**Requests req**)**

**{**

**for** **(**int i **=** 0**;** i **<** roomList**.**Count**;** i**++)** //проверка всех комнат

**{**

**if** **(**req**.**NumberPersons **<=** roomList**[**i**].**NumberPersons**)** //проверка вместимости

**{**

**if** **(**roomList**[**i**].**CheckList**.**Count **==** 0**)** //заселений ещё нет

**{**

ChechAndBook toAdd **=** **new** ChechAndBook**(true,** req**.**CheckInTime**,** req**.**CheckOutTime**);**

roomList**[**i**].**CheckList**.**Add**(**toAdd**);** //добавляем новое заселение

**if** **(**req**.**NumberPersons **==** roomList**[**i**].**NumberPersons**)** //если число людей совпадает с размером комнаты

**{**

IBank**.**AddToBank**(**roomList**[**i**].**Cost**);** //полная стоимость

**}**

**else**

**{**

IBank**.**AddToBank**(**Convert**.**ToUInt32**(**roomList**[**i**].**Cost **\*** 0.7**));** //70% стоимости

**}**

**return** roomList**[**i**];**

**}**

**else**

**{**

int j**;**

uint Ts **=** req**.**CheckInTime**,** Tf **=** req**.**CheckOutTime**;**

bool ifCan **=** **true;**

**for** **(**j **=** 0**;** **(**j **<** roomList**[**i**].**CheckList**.**Count**)** **&&** **!(**Tf **<** roomList**[**i**].**CheckList**[**j**].**CheckInTime**);** j**++)**

**if** **(((**Ts **<** roomList**[**i**].**CheckList**[**j**].**CheckInTime**)** **&&** **(**Tf **>** roomList**[**i**].**CheckList**[**j**].**CheckOutTime**))** **||**

**((**roomList**[**i**].**CheckList**[**j**].**CheckInTime **<** Ts**)** **&&** **(**Ts **<** roomList**[**i**].**CheckList**[**j**].**CheckOutTime**))** **||**

**((**roomList**[**i**].**CheckList**[**j**].**CheckInTime **<** Tf**)** **&&** **(**Tf **<** roomList**[**i**].**CheckList**[**j**].**CheckOutTime**)))**

**{**

ifCan **=** **false;** //временные отрезки пересекаются

**break;**

**}**

**if** **(!**ifCan**)**

**continue;**

**if** **((**j **==** roomList**[**i**].**CheckList**.**Count**)** **&&** **(**Ts **>** roomList**[**i**].**CheckList**[**j**-**1**].**CheckInTime**))**

**{**

ChechAndBook toAdd **=** **new** ChechAndBook**(true,** req**.**CheckInTime**,** req**.**CheckOutTime**);**

roomList**[**i**].**CheckList**.**Add**(**toAdd**);** //добавляем новое заселение

**if** **(**req**.**NumberPersons **==** roomList**[**i**].**NumberPersons**)** //если число людей совпадает с размером комнаты

**{**

IBank**.**AddToBank**(**roomList**[**i**].**Cost**);** //полная стоимость

**}**

**else**

**{**

IBank**.**AddToBank**(**Convert**.**ToUInt32**(**roomList**[**i**].**Cost **\*** 0.7**));** //70% стоимости

**}**

**}**

**else**

**{**

ChechAndBook toAdd **=** **new** ChechAndBook**(true,** req**.**CheckInTime**,** req**.**CheckOutTime**);**

roomList**[**i**].**CheckList**.**Insert**(**j**,** toAdd**);** //добавляем новое заселение на место j

**if** **(**req**.**NumberPersons **==** roomList**[**i**].**NumberPersons**)** //если число людей совпадает с размером комнаты

**{**

IBank**.**AddToBank**(**roomList**[**i**].**Cost**);** //полная стоимость

**}**

**else**

**{**

IBank**.**AddToBank**(**Convert**.**ToUInt32**(**roomList**[**i**].**Cost **\*** 0.7**));** //70% стоимости

**}**

**}**

**return** roomList**[**i**];**

**}**

**}**

**}**

**return** **null;** //не получилось заселить

**}**

**}**

**}**

Листинг 3.8.7 — Form1.cs

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**ComponentModel**;**

**using** System**.**Data**;**

**using** System**.**Drawing**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**Windows**.**Forms**;**

**namespace** Hotel\_Booking

**{**

**public** **partial** class Form1 **:** Form

**{**

**public** Form1**()**

**{**

InitializeComponent**();**

**}**

uint time **=** 24**;** //время имитации

uint time\_disp **=** 24**;** //отображаемое время

uint delta\_disp **=** 0**;**

uint scale**;**

bool Moving **=** **false;** //флаг запуска таймера

uint endTime**;** //время конца имитации

Hotel hotel**;**

**private** SortedList**<**uint**,** Requests**>** pendingRequests**;** //ожидающие заявки сортированные по порядку поступления

**private** void button1\_Click**(object** sender**,** EventArgs e**)**

**{**

button1**.**Enabled **=** **false;** //отключение кнопки старта

button2**.**Enabled **=** **true;** //всключение кнопок паузы и сохранения

button3**.**Enabled **=** **true;**

Moving **=** **true;** //переключение флага таймера

endTime **=** Convert**.**ToUInt32**(**comboBox2**.**SelectedItem**)\***24**;** //определение конечного времени

scale **=** Convert**.**ToUInt32**(**comboBox1**.**SelectedItem**);** //определение масштаба шага

timer1**.**Interval **=** **(**int**)(**timer1**.**Interval**/**scale**);**

// создание массива количества номеров каждого типа

uint**[]** arr **=** **{** Convert**.**ToUInt32**(**numericUpDown1**.**Value**),** Convert**.**ToUInt32**(**numericUpDown2**.**Value**),** Convert**.**ToUInt32**(**numericUpDown3**.**Value**),** Convert**.**ToUInt32**(**numericUpDown4**.**Value**)** **};**

CreateReq**();** //создание списка заявок

hotel **=** **new** Hotel**(**arr**);**

listViewRequests**.**Items**.**Clear**();**

comboBox1**.**Enabled **=** **false;** //блокировка элементов управления

comboBox2**.**Enabled **=** **false;**

numericUpDown1**.**Enabled **=** **false;**

numericUpDown2**.**Enabled **=** **false;**

numericUpDown3**.**Enabled **=** **false;**

numericUpDown4**.**Enabled **=** **false;**

labelTime**.**Text **=** time **/** 24 **+** " " **+** time **%** 24 **+** ":00"**;** //отображение времени

timer1**.**Start**();** //таймер для дискретного шага

**}**

**private** void timer1\_Tick**(object** sender**,** EventArgs e**)**

**{**

Requests newReq**;**

string**[]** row**;**

ListViewItem liv**;**

**while** **(**pendingRequests**.**TryGetValue**(**time**,out** newReq**))** //пока есть заявка на настоящее время

**{**

**if** **(**newReq**.**RequestsType **==** **false)** //заявка на бронирование

row **=** **new** string**[]** **{** "Бронирование"**,** time **/** 24 **+** " " **+** time **%** 24 **+** ":00"**,** newReq**.**CheckInTime **/** 24 **+** " " **+** newReq**.**CheckInTime **%** 24 **+** ":00"**,** newReq**.**CheckOutTime **/** 24 **+** " " **+** newReq**.**CheckOutTime **%** 24 **+** ":00"**,** newReq**.**NumberPersons**.**ToString**()** **};**

**else** //заявка на заселение

row **=** **new** string**[]** **{** "Заселение"**,** time **/** 24 **+** " " **+** time **%** 24 **+** ":00"**,** newReq**.**CheckInTime **/** 24 **+** " " **+** newReq**.**CheckInTime **%** 24 **+** ":00"**,** newReq**.**CheckOutTime **/** 24 **+** " " **+** newReq**.**CheckOutTime **%** 24 **+** ":00"**,** newReq**.**NumberPersons**.**ToString**()** **};**

liv **=** **new** ListViewItem**(**row**);** //обновление информации в списках

liv**.**Tag **=** newReq**;**

listViewRequests**.**Items**.**Insert**(**0**,** liv**);**

Room roomForReq **=** hotel**.**BookingRequests**(**newReq**);**

**if** **(**roomForReq **!=** **null)** //удалось заселить

**{**

//поля для добавления в таблицу

**if** **(**roomForReq**.**GetType**().**ToString**()** **==** **typeof(**LuxuryRoom**).**ToString**())**

**{**

row **=** **new** string**[]** **{** "Люкс"**,** newReq**.**CheckInTime **/** 24 **+** " " **+** newReq**.**CheckInTime **%** 24 **+** ":00"**,** newReq**.**CheckOutTime **/** 24 **+** " " **+** newReq**.**CheckOutTime **%** 24 **+** ":00"**,** newReq**.**NumberPersons**.**ToString**()** **};**

**}**

**else** **if** **(**roomForReq**.**GetType**().**ToString**()** **==** **typeof(**HalfLuxuryRoom**).**ToString**())**

**{**

row **=** **new** string**[]** **{** "Полулюкс"**,** newReq**.**CheckInTime **/** 24 **+** " " **+** newReq**.**CheckInTime **%** 24 **+** ":00"**,** newReq**.**CheckOutTime **/** 24 **+** " " **+** newReq**.**CheckOutTime **%** 24 **+** ":00"**,** newReq**.**NumberPersons**.**ToString**()** **};**

**}**

**else** **if** **(**roomForReq**.**GetType**().**ToString**()** **==** **typeof(**DoubleRoom**).**ToString**())**

**{**

row **=** **new** string**[]** **{** "Двухместный"**,** newReq**.**CheckInTime **/** 24 **+** " " **+** newReq**.**CheckInTime **%** 24 **+** ":00"**,** newReq**.**CheckOutTime **/** 24 **+** " " **+** newReq**.**CheckOutTime **%** 24 **+** ":00"**,** newReq**.**NumberPersons**.**ToString**()** **};**

**}**

**else** **if** **(**roomForReq**.**GetType**().**ToString**()** **==** **typeof(**SingleRoom**).**ToString**())**

**{**

row **=** **new** string**[]** **{** "Одноместный"**,** newReq**.**CheckInTime **/** 24 **+** " " **+** newReq**.**CheckInTime **%** 24 **+** ":00"**,** newReq**.**CheckOutTime **/** 24 **+** " " **+** newReq**.**CheckOutTime **%** 24 **+** ":00"**,** newReq**.**NumberPersons**.**ToString**()** **};**

**}**

liv **=** **new** ListViewItem**(**row**);**

liv**.**Tag **=** roomForReq**;**

listViewRoomNow**.**Items**.**Insert**(**0**,**liv**);** //добавление в начало таблицы

row **=** **new** string**[]** **{** time **/** 24 **+** " " **+** time **%** 24 **+** ":00"**,** IBank**.**DispBankAccount**().**ToString**()** **};**

liv **=** **new** ListViewItem**(**row**);**

liv**.**Tag **=** roomForReq**;**

listViewBank**.**Items**.**Insert**(**0**,** liv**);** //добавление в начало таблицы

**}**

pendingRequests**.**Remove**(**time**);** //удаление заявки из ожидающийх

**}**

time**++;** //следующий щаг

delta\_disp**++;**

**if** **(**delta\_disp **==** scale**)**

**{**

time\_disp **+=** delta\_disp**;**

labelTime**.**Text **=** time\_disp **/** 24 **+** " " **+** time\_disp **%** 24 **+** ":00"**;**

delta\_disp **=** 0**;**

**}**

**if** **(**time **==** endTime**)** //дошли до конца имитации

**{**

timer1**.**Stop**();**

**}**

**}**

**private** void CreateReq**()**

**{**

pendingRequests **=** **new** SortedList**<**uint**,** Requests**>();**

Random randStep **=** **new** Random**();**

uint \_time **=** 24**;**

uint step**;**

**while** **(**\_time **<** endTime**)**

**{**

pendingRequests**.**Add**(**\_time**,** **new** Requests**(**\_time**));**

step **=** Convert**.**ToUInt32**(**randStep**.**Next**(**0**,** int**.**MaxValue**)** **%** 5**)** **+** 1**;** //шаг до следующей заявки

\_time **+=** step**;**

**}**

**}**

**private** void button3\_Click**(object** sender**,** EventArgs e**)**

**{**

FileManipulation**.**SaveToFile**(**hotel**,**"save.txt"**);**

**}**

**private** void button2\_Click**(object** sender**,** EventArgs e**)**

**{**

Moving **=** **!**Moving**;**

**if** **(**Moving**==false)** //смена состояния таймера

**{**

timer1**.**Stop**();**

**}**

**else**

**{**

timer1**.**Start**();**

**}**

**}**

**private** void Form1\_Load**(object** sender**,** EventArgs e**)**

**{**

comboBox1**.**SelectedIndex **=** 0**;**

comboBox2**.**SelectedIndex **=** 0**;**

**}**

**}**

**}**

# Заключение

В результате выполнения курсовой работы были созданы три приложения. Программа, в которой используется механизм наследования, реализующая продажу театральных билетов, программа, осуществляющая поиск гамильтоновых циклов, и программа, выполняющая имитацию обработки заявок для заселения в гостиницу. При выполнении были продемонстрированы принципы объектно-ориентированного подхода к разработке программного обеспечения. Также были закреплены на практике знания, полученные за курс объектно-ориентированного программирования. Разработка консольных приложений и приложений Windows Forms позволила раскрыть возможности языка C#.

# Список использованных источников

1. Сайт разработчиков Microsoft. // Microsoft Developer Network. URL: https://msdn.microsoft.com/en-us (дата обращения: 20.04.2021)
2. Горячев А. В., Кравчук Д. К., Новакова Н. Е. Объектно-ориентированное моделирование. Учеб. Пособие. СПб.: Изд-во СПбГЭТУ “ЛЭТИ”, 2010.
3. Горячев А. В., Новакова Н. Е. Моделирование коммуникативных процессов в САПР. Учеб. пособие, СПб. Изд-во СПбГЭТУ «ЛЭТИ», 2010. 63 с.
4. Полное руководство по языку программирования С# 9.0 и платформе .NET 5. // URL: https://metanit.com/ (дата обращения: 20.04.2021)