**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

**к курсовой работе**

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

Тема: **«Разработка приложения на основе принципов объектно-ориентированного подхода»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2309 |  | Савин П. А. |
| Преподаватель |  | Новакова Н. Е. |

Санкт-Петербург

2023

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент: Савин П. А.

Группа: 2309

Тема работы: Разработка приложения на основе принципов объектно-ориентированного подхода

Исходные данные:

Реализовать структуры данных согласно заданию, реализовать вывод

результатов работы.

Содержание пояснительной записки:

Титульный лист, задание на курсовую работу, аннотация, содержание, введение, цель работы, разделы, заключение, список использованной литературы.

Содержание разделов:

Формулировка задания, теоретический аспект задачи, формализация задачи, спецификация классов, руководство оператора, руководство программиста, контрольный пример, листинг программы, выводы.

Предполагаемый объем пояснительной записки:

Не менее 40 страниц.

Дата выдачи задания: 22.10.2023

Дата сдачи курсовой работы: 18.12.2023

Дата защиты курсовой работы: 26.12.2023

|  |  |  |
| --- | --- | --- |
| Студент гр. 2309 |  | Савин П. А. |
| Преподаватель |  | Новакова Н. Е. |

**АННОТАЦИЯ**

Данная работа содержит в себе решение двух задач на основе объектно-ориентированного подхода. В первой задаче рассмотрены механизмы наследования и иерархия классов, во второй — разработка имитационной модели.

На основе этих моделей были разработаны приложения, включающие в себя различные пользовательские интерфейсы. Полученные результаты приведены в работе.

**SUMMARY**

This work contains the solution of two problems based on the object-oriented approach. The first task deals with inheritance mechanisms and class hierarchy, while the second task deals with the development of a simulation model.

Based on these models, applications including various user interfaces have been developed. The results obtained are summarized in the paper.

СОДЕРЖАНИЕ

[**АННОТАЦИЯ 3**](#_Toc154538407)

[**ВВЕДЕНИЕ 5**](#_Toc154538408)

[**ЦЕЛЬ РАБОТЫ 5**](#_Toc154538409)

[**1. ПЕРВЫЙ РАЗДЕЛ 6**](#_Toc154538410)

[**1.1. Формулировка задания 6**](#_Toc154538411)

[**1.2. Теоретический аспект задачи 6**](#_Toc154538412)

[**1.3. Формализация задачи 6**](#_Toc154538413)

[**1.4. Спецификация классов 10**](#_Toc154538414)

[**1.5. Руководство оператора 13**](#_Toc154538415)

[**1.6. Руководство программиста 13**](#_Toc154538416)

[**1.7. Контрольный пример 14**](#_Toc154538417)

[**1.8. Листинг программы 15**](#_Toc154538418)

[**2. ВТОРОЙ РАЗДЕЛ 19**](#_Toc154538419)

[**2.1. Формулировка задания 19**](#_Toc154538420)

[**2.2. Теоретический аспект задачи 19**](#_Toc154538421)

[**2.3. Формализация задачи 20**](#_Toc154538422)

[**2.4. Спецификация классов 24**](#_Toc154538423)

[**2.5. Руководство оператора 28**](#_Toc154538424)

[**2.6. Руководство программиста 30**](#_Toc154538425)

[**2.7. Контрольный пример 30**](#_Toc154538426)

[**2.8. Листинг программы 32**](#_Toc154538427)

[**ЗАКЛЮЧЕНИЕ 43**](#_Toc154538428)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 44**](#_Toc154538429)

# ВВЕДЕНИЕ

Курсовая работа направлена на создание приложений на основе объектно-ориентированного подхода на языке C#. В ней рассматриваются иерархии классов и наследования, а также имитационные модели. Курсовая работа состоит из 2 разделов.

# ЦЕЛЬ РАБОТЫ

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

Формулировка исходного задания:

* Первый раздел: разработать программу для обеспечения продажи театральных билетов.
* Второй раздел: создать программу, имитирующую карточную игру «Дурак переводной». Описать правила игры. Применить их в логике программы. Создать объектную структуру, подходящую для игры нескольких человек в карты. Пользовательский интерфейс должен обеспечивать ввод ходов и команд. Необходимо сформировать протокол игры с выдачей его по завершении и запоминанием «для истории» (Сбора статистики). Обеспечить вывод результатов сбора статистики в текстовый файл.

# **ПЕРВЫЙ РАЗДЕЛ**

## **Формулировка задания**

Вариант 19

Разработать программу для обеспечения продажи театральных билетов.

## **Теоретический аспект задачи**

Перед непосредственно реализацией необходимо определить структуру программы теоретически. Принято решение реализовать следующую структуру: описать абстрактный класс Ticket, являющийся общей моделью театрального билета, и имеющий свойства, определяющие билет как сущность — его идентификационный номер, название постановки и дата ее проведения, номер зала, номер места и тип места, также следует добавить метод Print, который будет реализовывать интерфейс IPrintable и организовывать вывод информации о билете в презентабельном виде. Для данного класса будут наследоваться пять классов, которые будут представлять модель билетов в конкретные места зала (партер, ложа, амфитеатр, бельэтаж, балкон), у которых дополнительным свойством будет цена, общая для всех экземпляров одного класса, а метод Print будет перегружаться, вызывая сначала Print у Ticket, и затем поверх этого метода выводить информацию о цене.

## **Формализация задачи**

Далее представлена диаграмма классов (Рисунок 1.1).

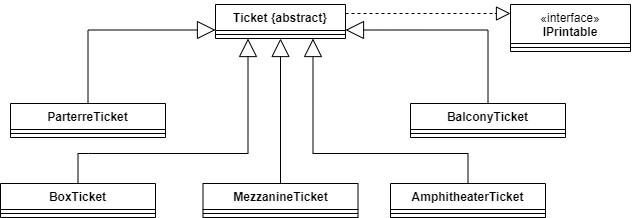


Рисунок 1.1 – Диаграмма классов для первого раздела

На рисунках 1.2 – 1.8 представлены диаграммы отдельных классов.

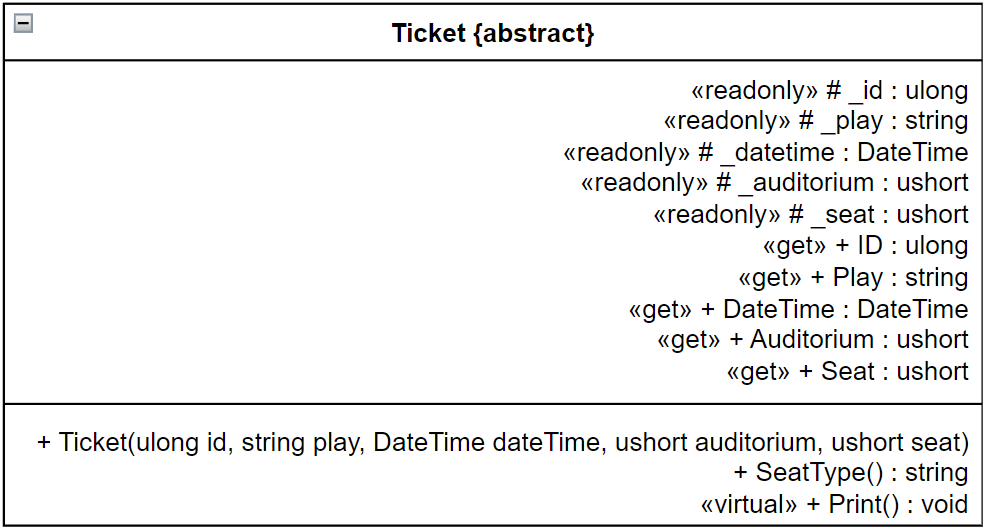


Рисунок 1.2 – Диаграмма класса Ticket

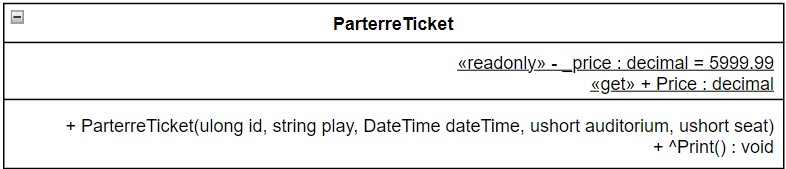


Рисунок 1.3 – Диаграмма класса ParterreTicket

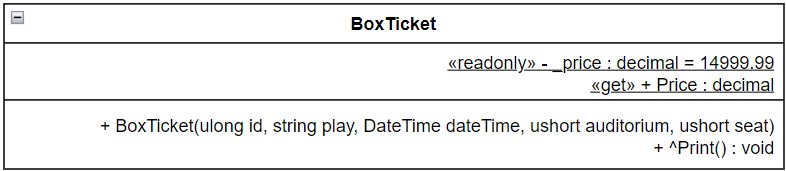


Рисунок 1.4 – Диаграмма класса BoxTicket

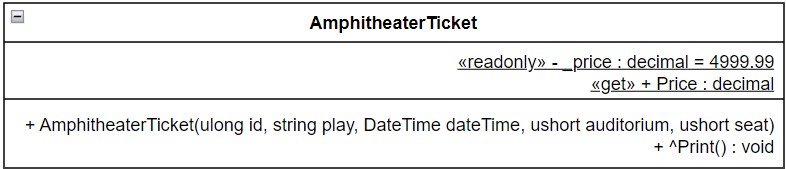


Рисунок 1.5 – Диаграмма класса AmphitheaterTicket

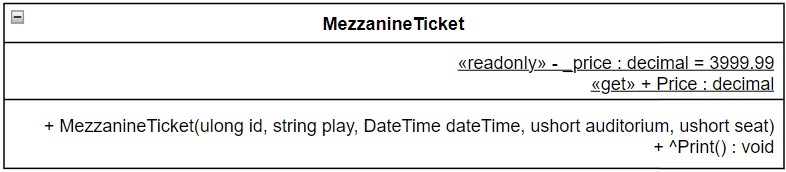


Рисунок 1.6 – Диаграмма класса MezzanineTicket

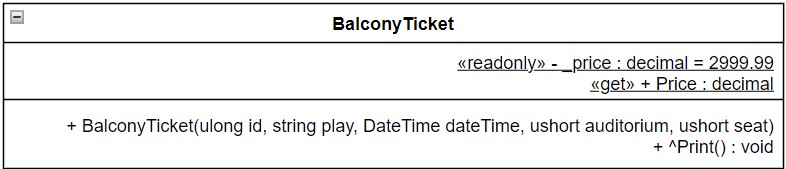


Рисунок 1.7 – Диаграмма класса BalconyTicket

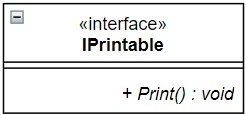


Рисунок 1.8 – Диаграмма интерфейса IPrintable

Основой программы является класс Ticket – абстрактный класс, описывающий общую модель театрального билета. Полями этого класса являются свойства театрального билета (не учитывая цены). У этого класса реализован конструктор, присваивающий при инициализации свойствам значения (только здесь свойства могут получить свои значения, так как они используют стереотип «только для чтения»), метод определения типа места в зале, который из имени типа высчитывает название типа места, и виртуальный метод Print, реализующий интерфейс IPrintable презентабельного вывода информации об объекте. От Ticket наследуются пять классов: ParterreTicket, BoxTicket, AmphitheaterTicket, MezzanineTicket и BalconyTicket. Помимо описания Ticket они имеют статическое поле \_price, также использующее стереотип «только для чтения» и являющееся свойством цены билета на конкретный тип места, для каждого класса это поле свое и оно имеет свое значение. У производных классов конструктор перенаправляет на конструктор базового класса, так как у самих классов ничего нового нет при инициализации, также перегружается интерфейсный метод Print — сначала вызывается Print базового класса, затем дополнительно выводится информация о цене билета.

## ****Спецификация классов****

Описание полей класса Ticket представлено в таблице 1.

**Таблица 1**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_id | ulong | protected | Идентификационный номер билета |
| \_play | string | protected | Название постановки |
| \_datetime | DateTime | protected | Дата и время постановки |
| \_auditorium | ushort | protected | Номер зала |
| \_seat | ushort | protected | Номер места |

Описание методов класса Ticket представлено в таблице 2.

**Таблица 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Ticket | - | public | ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat | - | Конструктор класса |
| SeatType | string | public | - | - | Определение типа места в зале |
| Print | void | public | - | - | Вывод информации о билете в презентабельном виде |

Описание полей класса ParterreTicket представлено в таблице 3.

**Таблица 3**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_price | decimal | private | Цена билета в партер |

Описание методов класса ParterreTicket представлено в таблице 4.

**Таблица 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| ParterreTicket | - | public | ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat | - | Конструктор класса |
| Print | void | public | - | - | Вывод информации о билете в презентабельном виде |

Описание полей класса BoxTicket представлено в таблице 5.

**Таблица 5**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_price | decimal | private | Цена билета в ложу |

Описание методов класса BoxTicket представлено в таблице 6.

**Таблица 6**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| BoxTicket | - | public | ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat | - | Конструктор класса |
| Print | void | public | - | - | Вывод информации о билете в презентабельном виде |

Описание полей класса AmphitheaterTicket представлено в таблице 7.

**Таблица 7**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_price | decimal | private | Цена билета в амфитеатр |

Описание методов класса AmphitheaterTicket представлено в таблице 8.

**Таблица 8**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| AmphitheaterTicket | - | public | ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat | - | Конструктор класса |
| Print | void | public | - | - | Вывод информации о билете в презентабельном виде |

Описание полей класса MezzanineTicket представлено в таблице 9.

**Таблица 9**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_price | decimal | private | Цена билета на бельэтаж |

Описание методов класса MezzanineTicket представлено в таблице 10.

**Таблица 10**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| MezzanineTicket | - | public | ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat | - | Конструктор класса |
| Print | void | public | - | - | Вывод информации о билете в презентабельном виде |

Описание полей класса BalconyTicket представлено в таблице 11.

**Таблица 11**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_price | decimal | private | Цена билета на балкон |

Описание методов класса BalconyTicket представлено в таблице 12.

**Таблица 12**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| BalconyTicket | - | public | ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat | - | Конструктор класса |
| Print | void | public | - | - | Вывод информации о билете в презентабельном виде |

## **Руководство оператора**

Программа для пользователя является демонстрационной. Все данные, выводящиеся на экран, задаются программистом, пользователю остается лишь запустить программу и увидеть результаты работы программы на экране.

## **Руководство программиста**

Все логически отдельные структуры находятся в раздельных файлах. Название файлов интерфейсов начинается с буквы I. Для получения данных о полях экземпляров классов извне следует использовать реализованные геттеры. Также поля используют стереотип «только для чтения», поэтому присваивание значений новому экземпляру извне происходит лишь однажды при инициализации. Класс Ticket является абстрактным, поэтому создание экземпляра класса Ticket является недопустимым.

## Контрольный пример

Далее представлены результаты работы программы (Рисунок 2).

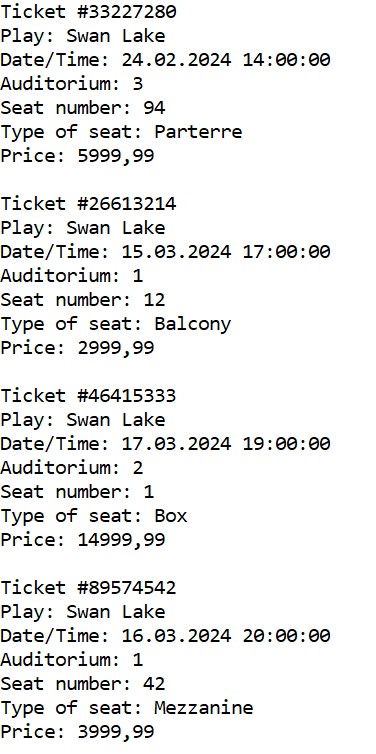


Рисунок 2 - Контрольный пример для работы

Пояснение: в методе Main создаются 4 различных билета (в партер, на балкон, в ложу, на бельэтаж) и через конструктор им присваиваются свои данные, после чего у них по очереди вызывается метод Print, выводящий на экран пользователя информацию о созданных билетах (через строку): идентификационный номер, название постановки, дата и время постановки, номер зала, номер места, тип места, цена билета.

## Листинг программы

Program.cs

namespace Theater

{

internal class Program

{

static void Main(string[] args)

{

ParterreTicket p\_ticket = new ParterreTicket(33227280, "Swan Lake", new DateTime(2024, 02, 24, 14, 00, 00), 3, 94);

BalconyTicket blc\_ticket = new BalconyTicket(26613214, "Swan Lake", new DateTime(2024, 03, 15, 17, 00, 00), 1, 12);

BoxTicket bx\_ticket = new BoxTicket(46415333, "Swan Lake", new DateTime(2024, 03, 17, 19, 00, 00), 2, 1);

MezzanineTicket m\_ticket = new MezzanineTicket(89574542, "Swan Lake", new DateTime(2024, 03, 16, 20, 00, 00), 1, 42);

p\_ticket.Print();

Console.WriteLine();

blc\_ticket.Print();

Console.WriteLine();

bx\_ticket.Print();

Console.WriteLine();

m\_ticket.Print();

}

}

}

IPrintable.cs

namespace Theater

{

internal interface IPrintable

{

void Print();

}

}

Ticket.cs

namespace Theater

{

internal abstract class Ticket : IPrintable // абстрактный класс Ticket, описывающий общую модель билета и реализующий интерфейс IPrintable для презентабельного вывода информации о билете

{

protected readonly ulong \_id; // идентификационный номер билета

protected readonly string \_play; // название постановки

protected readonly DateTime \_datetime; // дата и время постановки

protected readonly ushort \_auditorium; // номер зала

protected readonly ushort \_seat; // номер места в зале

public ulong ID { get { return \_id; } } // геттер для \_id

public string Play { get { return \_play; } } // геттер для \_play

public DateTime DateTime { get { return \_datetime; } } // геттер для \_datetime

public ushort Auditorium { get { return \_auditorium; } } // геттер для \_auditorium

public ushort Seat { get { return \_seat; } } // геттер для \_seat

public Ticket(ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat) // конструктор класса

{

\_id = id;

\_play = play;

\_datetime = dateTime;

\_auditorium = auditorium;

\_seat = seat;

}

public string SeatType()

{

string type = GetType().Name.ToString();

byte index = 1;

while (!char.IsUpper(type[index]))

{

++index;

}

return type.ToString().Substring(0, index);

}

public virtual void Print() // реализация интерфейсного метода Print; выводит информацию о билете

{

Console.WriteLine("Ticket #{0}\nPlay: {1}\nDate/Time: {2}\nAuditorium: {3}\nSeat number: {4}\nType of seat: {5}", ID, Play, DateTime, Auditorium, Seat, SeatType());

}

}

}

ParterreTicket.cs

namespace Theater

{

internal class ParterreTicket : Ticket // класс ParterreTicket, описывающий билет в партер

{

private static readonly decimal \_price = 5999.99M; // цена билета в партер; для всех билетов в партер она одинаковая

public static decimal Price { get { return \_price; } } // геттер для \_price

public ParterreTicket(ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat) : base(id, play, dateTime, auditorium, seat) { }

public override void Print()

{

base.Print();

Console.WriteLine("Price: {0}", Price);

}

}

}

BoxTicket.cs

namespace Theater

{

internal class BoxTicket : Ticket // класс BoxTicket, описывающий билет в ложу

{

private static readonly decimal \_price = 14999.99M; // цена билета в ложу; для всех билетов в ложу она одинаковая

public static decimal Price { get { return \_price; } } // геттер для \_price

public BoxTicket(ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat) : base(id, play, dateTime, auditorium, seat) { }

public override void Print()

{

base.Print();

Console.WriteLine("Price: {0}", Price);

}

}

}

AmphitheaterTicket.cs

namespace Theater

{

internal class AmphitheaterTicket : Ticket // класс AmphitheaterTicket, описывающий билет в амфитеатр

{

private static readonly decimal \_price = 4999.99M; // цена билета в амфитеатр; для всех билетов в амфитеатр она одинаковая

public static decimal Price { get { return \_price; } } // геттер для \_price

public AmphitheaterTicket(ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat) : base(id, play, dateTime, auditorium, seat) { }

public override void Print()

{

base.Print();

Console.WriteLine("Price: {0}", Price);

}

}

}

MezzanineTicket.cs

namespace Theater

{

internal class MezzanineTicket : Ticket // класс MezzanineTicket, описывающий билет на бельэтаж

{

private static readonly decimal \_price = 3999.99M; // цена билета на бельэтаж; для всех билетов на бельэтаж она одинаковая

public static decimal Price { get { return \_price; } } // геттер для \_price

public MezzanineTicket(ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat) : base(id, play, dateTime, auditorium, seat) { }

public override void Print()

{

base.Print();

Console.WriteLine("Price: {0}", Price);

}

}

}

BalconyTicket.cs

namespace Theater

{

internal class BalconyTicket : Ticket // класс BalconyTicket, описывающий билет на балкон

{

private static readonly decimal \_price = 2999.99M; // цена билета на балкон; для всех билетов на балкон она одинаковая

public static decimal Price { get { return \_price; } } // геттер для \_price

public BalconyTicket(ulong id, string play, DateTime dateTime, ushort auditorium, ushort seat) : base(id, play, dateTime, auditorium, seat) { }

public override void Print()

{

base.Print();

Console.WriteLine("Price: {0}", Price);

}

}

}

# ВТОРОЙ РАЗДЕЛ

## **Формулировка задания**

Вариант 23

Имитация карточной игры «Дурак переводной»

Создать программу, имитирующую карточную игру «Дурак переводной». Описать правила игры. Применить их в логике программы.

Создать объектную структуру, подходящую для игры нескольких человек в карты. Пользовательский интерфейс должен обеспечивать ввод ходов и команд. Необходимо сформировать протокол игры с выдачей его по завершении и запоминанием «для истории» (Сбора статистики).

Обеспечить вывод результатов сбора статистики в текстовый файл.

## **Теоретический аспект задачи**

Для решения поставленной задачи требуется изучить правила игры «Дурак», дополнения в варианте игры «Дурак переводной», и применить полученные сведения в логике программы.

«Дурак» — игра для 2-4 человек с колодой из 36 игральных карт старшинством от 6 до туза, где каждому в начале раздается по 6 карт, а нижняя карта из колоды становится козырной, видимой для всех игроков. Все карты, у которых масть совпадает с козырной, также являются козырными. Игру начинает игрок с самым младшим козырем (или самый младший игрок, если ни у кого нет в руке козырей), далее ход идет по часовой стрелке. Ведущий подкидывает своему соседу слева карты из своей руки, при этом он может подкинуть несколько карт, если они одного старшинства, на что оппоненту требуется ответить, покрыв карты из своей руки — покрывать карту можно одной масти и старшинством выше, либо козырем (однако если был подкинут козырь, отвечать можно только козырем большего старшинства, по этой причине козырной туз является в игре небьющейся картой); если же ответить нечем, игрок должен взять карты и пропустить свой ход, если все карты успешно были покрыты, ведущий может продолжать подкидывать карты, которые совпадают по старшинству с уже выложенными, или, если такой возможности нет, оставить подкидывание карт остальным игрокам за столом, но не более 6 карт на покрытие или даже меньше, если у отвечающего меньше 6 карт. Битые карты уходят в сброс — они больше не участвуют в игре, после чего каждому игроку раздаются карты из колоды, пока их не станет по 6 штук в руке (игрокам с большим количеством карт или не участвовавшим в подкидывании карты не раздают). Раздача происходит в следующем порядке: сначала ведущий (чей был ход), затем все остальные игроки, подкидывавшие карты в стол, и затем только отвечающий. Конечная цель игры — полностью избавиться от карт в руке, соответственно, оставшийся с картами в руке проигрывает (остается в «дураках»).

Дополнения «Дурака переводного»: пока отвечающий не начал отбиваться, он может перевести подкинутые карты на следующего по кругу игрока и сделать его отвечающим, если у него есть карты одного старшинства с выложенными в стол, при этом можно выложить как одну, так и несколько карт сразу.

Соответственно, при игре на двоих переводы происходят друг на друга, и круг оппонентов сужен до одного человека. Эти условия и были учтены при создании имитационной модели.

## **Формализация задачи**

**Описание классов.**

Далее представлена диаграмма классов (Рисунок 3.1).

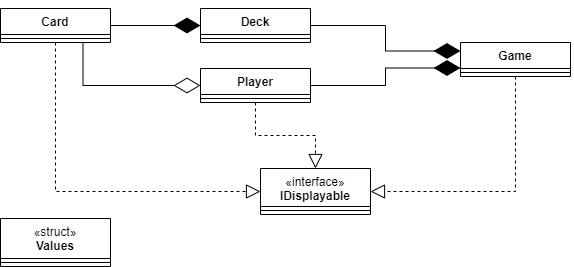


Рисунок 3.1 – Диаграмма классов для второго раздела

На рисунках 3.2 – 3.7 представлены диаграммы отдельных классов.

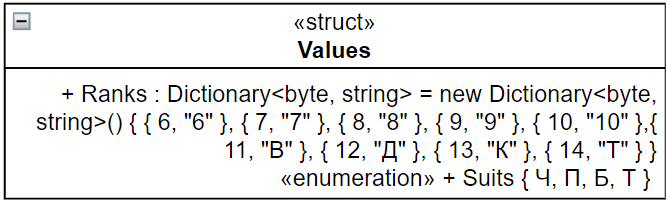


Рисунок 3.2 – Диаграмма структуры Values

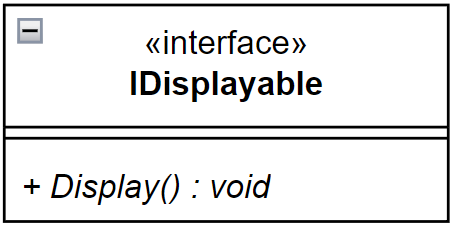


Рисунок 3.3 – Диаграмма интерфейса IDisplayable

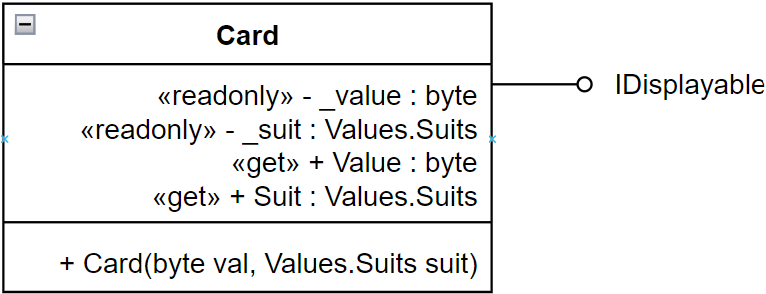


Рисунок 3.4 – Диаграмма класса Card

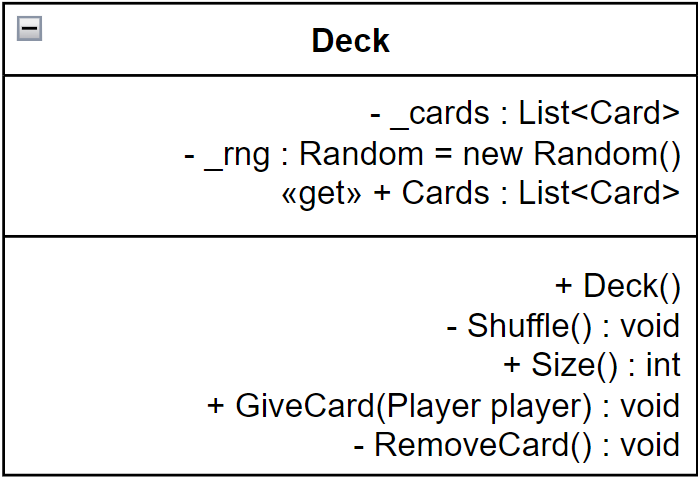


Рисунок 3.5 – Диаграмма класса Deck

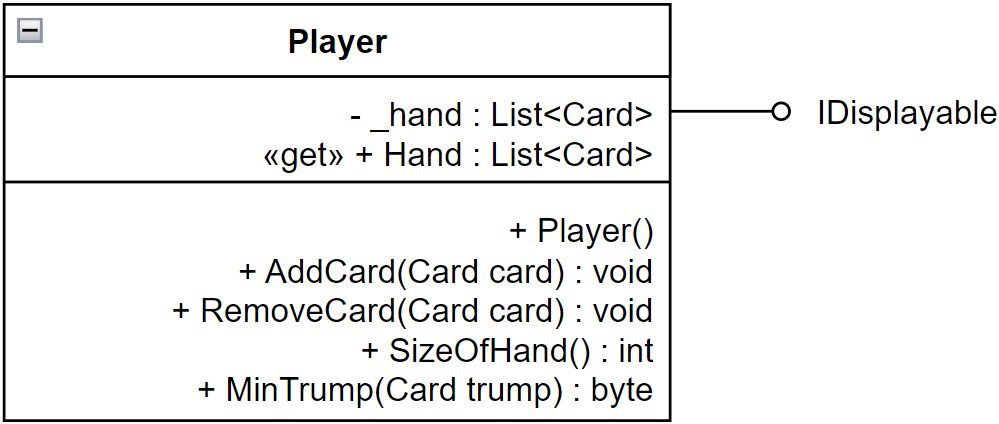


Рисунок 3.6 – Диаграмма класса Player

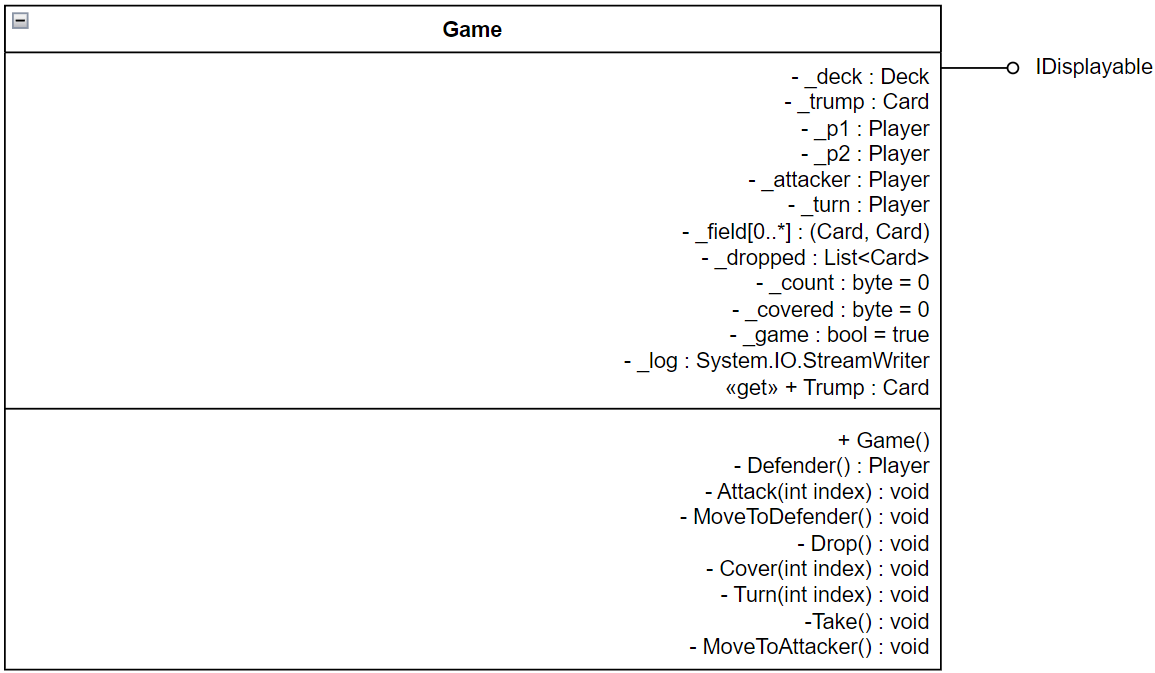


Рисунок 3.7 – Диаграмма класса Game

Игра является композицией базовых классов. Класс Card является самым базовым и описывает модель игральной карты. Полями этого класса являются свойства игральной карты (старшинство и масть). Card является агрегируемой частью класса Player, описывающего модель игрока, у которого есть одно поле — рука игрока, реализованная как список карт. Также Card явяляется частью композиции класса Deck, описывающей модель колоды карт, полями которого являются список карт — собственно, сама колода карт, и генератор случайных чисел, необходимый для перемешивания колоды. В свою очередь классы Player и Deck являются частями композитного класса Game, который является описанием игровой логики и самой игрой, то есть, весь игровой процесс происходит в этом классе. Полями этого класса являются игровые параметры, необходимые для достаточной реализации игры — колода карт, козырная карта, игроки, кто нападает, кто ходит, игровой стол, сброс, сколько карт было подкинуто, сколько карт было побито, идет ли игра. Классы Card, Player и Game реализуют интерфейс IDisplayable для красивого вывода объекта на экран.

**Описание алгоритмов.**

Что касается обработки различных внутриигровых ситуаций — в классе Game происходит подготовка к игре — перемешивание колоды, раздача карт игрокам, определение козырной карты колоды и игрока, чей ход будет первым по младшему козырю, после чего запускается сам игровой процесс, в котором, путем нажатия требуемых от пользователя клавиш, совершаются действия. Также предусмотрено разделение обработки на действия, которые может совершать нападающий игрок — подкидывать карты из своей руки, передавать хода отбивающемуся игроку и объявлять бито, и действия, которые может совершать отбивающийся игрок — покрывать выложенные карты из своей руки, переводить карты одного старшинства на оппонента (также из своей руки), забирать карты со стола в случае неудачного ответа и передавать ход нападающему игроку в случае удачного ответа.

За каждое из вышеперечисленных действий в игре отвечают отдельные методы класса Game, которые до исполнения самого действия в соответствии с правилами игры «Дурак переводной» проверяют, а позволяет ли текущая игровая ситуация (карты на столе, карты у игроков, карты в колоде) совершить действие. После выполнения действий «объявить бито» и «забрать карты со стола» дополнительно проверяются колода и рукава игроков: если после объявления бито у игроков не осталось ни одной карты, и в колоде тоже не осталось карт, игра считается завершенной вничью; если после взятия карт со стола у нападавшего игрока не осталось ни одной карты, и в колоде тоже не осталось карт, игра считается завершенной с признанием нападавшего игрока победителем.

## Спецификация классов

Описание полей структуры Values представлено в таблице 13.

**Таблица 13**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| Ranks | Dictionary<byte, string> | public | Перевод числовых значений карт в старшинства |
| Suits | enum | public | Перевод букв, соответствующих мастям, в числа |

Описание полей класса Card представлено в таблице 14.

**Таблица 14**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_value | byte | private | Старшинство карты в числовом виде |
| \_suit | Values.Suits | private | Масть карты |

Описание методов класса Card представлено в таблице 15.

**Таблица 15**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Card | - | public | byte val, Values.Suits suit | - | Конструктор класса |
| Display | void | public | - | - | Вывод карты в презентабельном виде |

Описание полей класса Deck представлено в таблице 16.

**Таблица 16**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_cards | List<Card> | private | Список-колода карт |
| \_rng | Random | private | Генератор случайных чисел для перемешивания колоды |

Описание методов класса Deck представлено в таблице 17.

**Таблица 17**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Deck | - | public | - | - | Конструктор класса |
| Shuffle | void | private | - | - | Перемешивание колоды карт |
| Size | int | public | - | - | Возврат размера колоды |
| GiveCard | void | public | Player player | - | Отдать карту игроку из колоды |
| RemoveCard | void | private | - | - | Убрать карту из колоды |

Описание полей класса Player представлено в таблице 18.

**Таблица 18**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_hand | List<Card> | private | Рука игрока |

Описание методов класса Player представлено в таблице 19.

**Таблица 19**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Player | - | public | - | - | Конструктор класса |
| AddCard | void | public | Card card | - | Добавление карты в руку игрока |
| RemoveCard | void | public | int index | - | Удаление карты из руки игрока по переданному индексу |
| SizeOfHand | int | public | - | - | Подсчет количества карт в руке игрока |
| MinTrump | byte | public | Card trump | - | Определение младшего козыря |
| Display | void | public | - | - | Вывод игрока в презентабельном виде |

Описание полей класса Game представлено в таблице 20.

**Таблица 20**

| Имя | Тип | Модификатор доступа | Назначение |
| --- | --- | --- | --- |
| \_deck | Deck | private | Колода карт |
| \_trump | Card | private | Козырная карта |
| \_p1 | Player | private | Игрок 1 |
| \_p2 | Player | private | Игрок 2 |
| \_attacker | Player | private | Кто нападает в данный момент |
| \_turn | Player | private | Кто ходит в данный момент |
| \_field | (Card, Card)[] | private | Игровой стол |
| \_dropped | List<Card> | private | Сброс |
| \_count | byte | private | Сколько карт было подкинуто |
| \_covered | byte | private | Сколько карт было отбито |
| \_game | bool | private | Идет ли игра |
| \_log | System.IO.StreamWriter | private | Протоколирование игры |

Описание методов класса Game представлено в таблице 21.

**Таблица 21**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Возвращаемый тип | Модификатор доступа | Входные параметры | Выходные параметры | Назначение |
| Game | - | public | - | - | Конструктор класса, по совместительству, и сама игра |
| Defender | Player | private | int index | - | Определение отбивающегося игрока в текущем коне |
| Attack | void | private | - | - | Подкинуть карту в стол |
| MoveToDefender | void | private | - | - | Передать ход отбивающемуся игроку |
| Drop | void | private | - | - | Скинуть карты со стола в сброс |
| Cover | void | private | int index | - | Покрыть следующую непокрытую карту на столе |
| Turn | void | private | int index | - | Перевести стол на оппонента |
| Take | void | private | - | - | Взять карты |
| MoveToAttacker | void | private | - | - | Передать ход нападающему игроку |
| Display | void | public | - | - | Вывод игрового стола в презентабельном виде |

## **Руководство оператора**

При запуске пользователя встречает консольный интерфейс, где ему отображается, какие действия он может сделать, в зависимости от его хода, и на какие клавиши ему нужно нажимать. Примеры интерфейса показаны на рисунках 4.1-4.3.

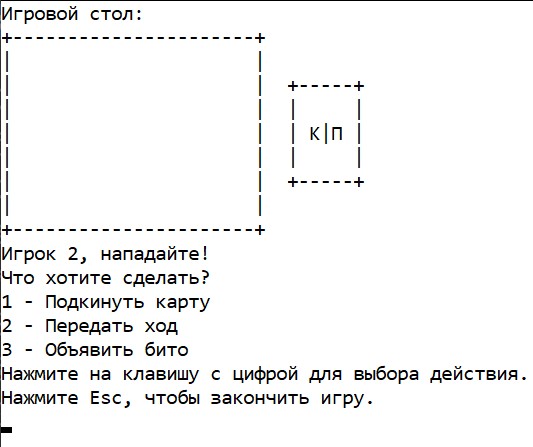


Рисунок 4.1 – Интерфейс нападающего игрока

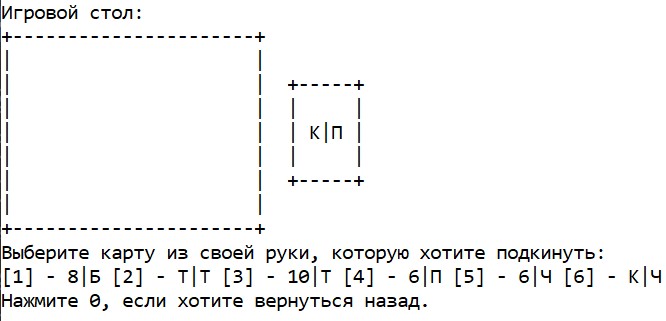


Рисунок 4.2 – Интерфейс выбора карты (доступен при подкидывании, переводе и покрытии)

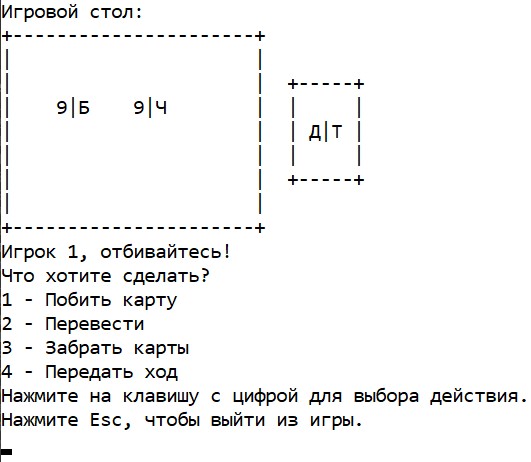


Рисунок 4.3 – Интерфейс отбивающегося игрока

Как можно заметить на рисунках, все действия обозначены клавишей, которую пользователю нужно нажать для совершения желаемого действия, при этом если выбранное действие в данный момент недопустимо, пользователя перебросит обратно на главный интерфейс, где он вновь может выбирать действия.

## **Руководство программиста**

Все логически отдельные структуры находятся в раздельных файлах. Особенности каждого типа данных отображены с помощью комментариев в программе (представлено в листинге). Промежуточная информация о ходах игроков выводится в файл log.txt.

## Контрольный пример

Далее представлены результаты работы программы (Рисунки 5.1 и 5.2).

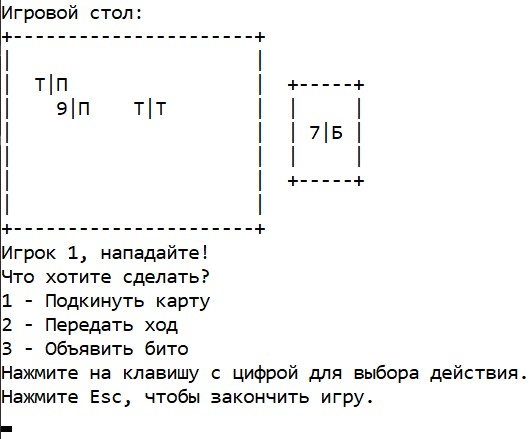


Рисунок 5.1 - Контрольный пример для второго раздела

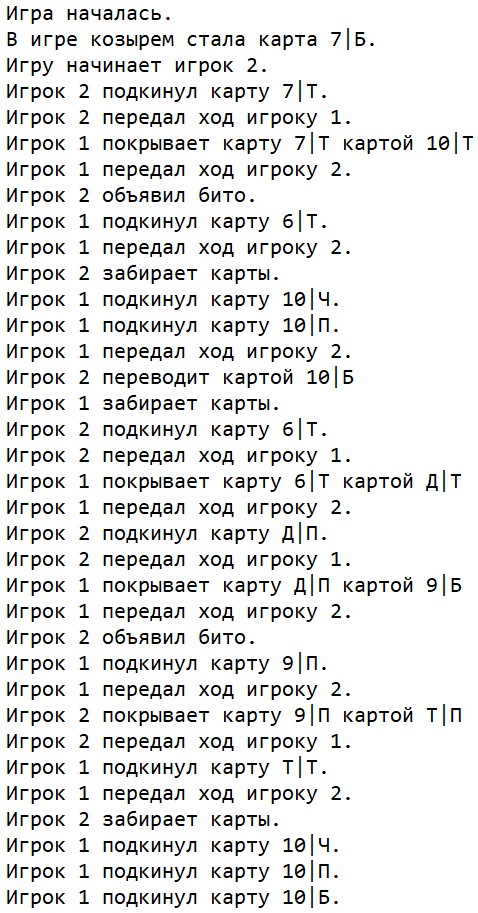


Рисунок 5.2 – протокол для контрольного примера

Пояснение: была запущена демонстративная партия и сыграна от начала до конца, после чего данные игры были записаны в протокол. На рисунке 5.2 можно увидеть, как началась игра, и какие ходы делали игроки.

## Листинг программы

IDisplayable.cs

namespace Game

{

// интерфейс красивого вывода объекта на экран

internal interface IDisplayable

{

void Display();

}

}

Values.cs

namespace Game

{

internal struct Values // структура Values,

// дающая доступ к старшинствам и мастям

{

// словарь перевода числовых значений в привычные нам старшинства

public static Dictionary<byte, string> Ranks = new()

{

{ 6, "6" },

{ 7, "7" },

{ 8, "8" },

{ 9, "9" },

{ 10, "10" },

{ 11, "В" },

{ 12, "Д" },

{ 13, "К" },

{ 14, "Т" }

};

public enum Suits { Ч, П, Б, Т }; // масти

}

}

Card.cs

namespace Game

{

internal class Card : IDisplayable // класс карты

{

private readonly byte \_value; // старшинство карты в числовом формате;

// словарь старшинств содержится в Values

private readonly Values.Suits \_suit; // масть карты

public byte Value { get { return \_value; } } // геттер для \_value

public Values.Suits Suit { get { return \_suit;} } // геттер для \_suit

public Card(byte val, Values.Suits suit) // конструктор класса Card;

// присвоение значений происходит

// только здесь

{

\_value = val;

\_suit = suit;

}

public void Display() // метод вывода карты

{

string toPrint = (\_value == 10 ? "" : " ") +

Values.Ranks[\_value] + "|"

+ \_suit.ToString(); // сформировать строку

Console.Write(toPrint); // и передать ее в вывод

}

}

}

Deck.cs

namespace Game

{

internal class Deck // класс колоды карт

{

private List<Card> \_cards; // список карт

private Random \_rng = new Random(); // рандомайзер

public List<Card> Cards { get { return \_cards; } } // геттер для \_cards

public Deck() // конструктор класса Deck; создает колоду и перемешивает ее

{

\_cards = new List<Card>();

for (int i = 6; i < 15; ++i)

{

for (int j = 0; j < 4; ++j)

{

\_cards.Add(new Card((byte)i, (Values.Suits)j));

}

}

Shuffle();

}

private void Shuffle() // метод перемешивания колоды

{

int n = \_cards.Count;

while (n > 1)

{

--n;

int k = \_rng.Next(n + 1);

Card value = \_cards[k];

\_cards[k] = \_cards[n];

\_cards[n] = value;

}

}

public int Size() // метод возврата текущего размера колоды

{

return \_cards.Count;

}

public void GiveCard(Player player) // метод раздачи карты из колоды игроку

{

player.AddCard(\_cards.First());

RemoveCard();

}

private void RemoveCard() // метод удаления карты из колоды

{

\_cards.Remove(\_cards.First());

}

}

}

Player.cs

namespace Game

{

internal class Player : IDisplayable // класс игрока

{

private List<Card> \_hand; // рука игрока

public List<Card> Hand { get { return \_hand; } } // геттер для \_hand

public Player() // конструктор класса Player

{

\_hand = new List<Card>();

}

public void AddCard(Card card) // метод добавления карты в руку игрока

{

\_hand.Add(card);

}

public void RemoveCard(int index) // метод удаления карты с руки игрока по // индексу

{

if (index <= \_hand.Count - 1)

{

\_hand.RemoveAt(index);

}

}

public int SizeOfHand() // метод, возвращающий кол-во карт в руке игрока

{

return \_hand.Count;

}

public byte MinTrump(Card trump) // метод, подсчитывающий младший козырь в // руке игрока

{

byte min = 15; // туз имеет значение 14, значит, если у игрока не было // обнаружено

// ни одного козыря, вернется 15

foreach (Card card in \_hand)

{

if (card.Suit == trump.Suit && card.Value < min) // если очередная // карта совпадает

// по масти с козырем

// и ее значение //меньше 15,

// это новый младший // козырь

{

min = card.Value;

}

}

return min; // в конце вернуть найденное значение

}

public void Display() // метод вывода имеющихся у игрока карт

{

if (\_hand.Count == 0) // если у игрока нет карт, ничего не выводим

{

Console.Write("");

return;

}

for (int i = 0; i < \_hand.Count; ++i)

{

Console.Write("[" + (i + 1).ToString() + "]"

+ " - " + Values.Ranks[\_hand[i].Value] + "|"

+ \_hand[i].Suit.ToString() + " ");

}

}

}

Game.cs

namespace Game

{

internal class Game : IDisplayable // класс игры

{

private Deck \_deck; // колода

private Card \_trump; // козырь

private Player \_p1; // игрок 1

private Player \_p2; // игрок 2

private Player \_attacker; // кто нападает

private Player \_turn; // кто ходит

private (Card, Card)[] \_field; // игровой стол

private List<Card> \_dropped; // бито

private byte \_count = 0; // сколько карт было выложено на стол

private byte \_covered = 0; // сколько карт было покрыто

private bool \_game = true; // идет ли игра или нет

private StreamWriter \_log; // запись логов игры

// конструктор класса Game; здесь же и происходит игра

// и обработка внутриигровых событий

public Game()

{

// интерфейс для нападающего игрока

string msg\_attack = "Что хотите сделать?\n1 - Подкинуть карту\n" +

"2 - Передать ход\n3 - Объявить бито\n" +

"Нажмите на клавишу с цифрой для выбора действия.\n" +

"Нажмите Esc, чтобы закончить игру.";

// интерфейс для отбивающегося игрока

string msg\_defend = "Что хотите сделать?\n1 - Побить карту\n2 - Перевести\n" +

"3 - Забрать карты\n4 - Передать ход\n" +

"Нажмите на клавишу с цифрой для выбора действия.\n" +

"Нажмите Esc, чтобы выйти из игры.";

// инициализация полей класса

\_deck = new Deck();

\_trump = \_deck.Cards.Last();

\_p1 = new Player();

\_p2 = new Player();

\_field = new (Card, Card)[6];

\_dropped = new List<Card>();

\_log = new StreamWriter(Directory.GetCurrentDirectory()

+ "\\" + "log.txt");

\_log.WriteLine("Игра началась.");

\_log.WriteLine("Козырем стала карта " +

Values.Ranks[\_trump.Value] + '|' +

\_trump.Suit.ToString() + ".");

// раздача карт и определение младшего козыря

for (int i = 0; i < 6; ++i)

{

\_deck.GiveCard(\_p1);

\_deck.GiveCard(\_p2);

}

if (\_p1.MinTrump(\_trump) <= \_p2.MinTrump(\_trump)) // у игрока 1 есть // козырь

// младше козырей игрока // 2,

{ // либо у них обоих нет // козырей

\_turn = \_p1; // передаем ход игроку 1

\_attacker = \_p1; // и делаем его нападающим

\_log.WriteLine("Начинает игрок 1.");

}

else

{

\_turn = \_p2; // передаем ход игроку 2

\_attacker = \_p2; // и делаем его нападающим

\_log.WriteLine("Начинает игрок 2.");

}

ConsoleKey key\_main, key\_second; // у меню два уровня вложенности,

// поэтому и код имеет две переменные

// сама игра

while (\_game)

{

Display();

Console.Write("Игрок {0}, ", (\_turn == \_p1 ? 1 : 2));

// обработка действий нападающего игрока

if (\_turn == \_attacker)

{

Console.WriteLine("нападайте!");

Console.WriteLine(msg\_attack);

key\_main = Console.ReadKey().Key;

switch (key\_main)

{

case (ConsoleKey.Escape):

\_game = false;

\_log.WriteLine("Игра завершилась досрочно.");

break;

case (ConsoleKey.D1):

Display();

Console.WriteLine("Выберите карту из своей руки, " +

"которую хотите подкинуть:");

\_turn.Display();

Console.WriteLine("\nНажмите 0, если хотите вернуться назад.");

key\_second = Console.ReadKey().Key;

if (key\_second == ConsoleKey.D0)

{

break;

}

Attack((int)key\_second % 49);

break;

case (ConsoleKey.D2):

MoveToDefender();

break;

case (ConsoleKey.D3):

Drop();

break;

}

}

// обработка действий отбивающегося игрока

else

{

Console.WriteLine("отбивайтесь!");

Console.WriteLine(msg\_defend);

key\_main = Console.ReadKey().Key;

switch (key\_main)

{

case (ConsoleKey.Escape):

\_game = false;

\_log.WriteLine("Игра завершилась досрочно.");

break;

case (ConsoleKey.D1):

Display();

Console.WriteLine("Выберите карту из своей руки, " +

"которой хотите покрыть:");

\_turn.Display();

Console.WriteLine("\nНажмите 0, если хотите вернуться назад.");

key\_second = Console.ReadKey().Key;

if (key\_second == ConsoleKey.D0)

{

break;

}

Cover((int)key\_second % 49);

break;

case (ConsoleKey.D2):

Display();

Console.WriteLine("Выберите карту из своей руки, " +

"которой хотите перевести:");

\_turn.Display();

Console.WriteLine("\nНажмите 0, если хотите вернуться назад.");

key\_second = Console.ReadKey().Key;

if (key\_second == ConsoleKey.D0)

{

break;

}

Turn((int)key\_second % 49);

break;

case (ConsoleKey.D3):

Take();

break;

case (ConsoleKey.D4):

MoveToAttacker();

break;

}

}

}

// после игры закрываем файл протокола

\_log.Close();

}

// метод, определяющий отбивающегося игрока на текущем кону

private Player Defender()

{

// если ходит нападающий, нужно вернуть другого игрока

if (\_turn == \_attacker)

{

return (\_turn == \_p1 ? \_p2 : \_p1);

}

// в противном случае возвращаем ходящего игрока

return \_turn;

}

// метод, производящий попытку подкинуть карту в стол

private void Attack(int index)

{

/\* Условия допустимости подкидывания:

\* 1. Подкинутых карт меньше 6

\* 2. У отбивающегося игрока хватит карт покрыться

\* 3. Карта совпадает по старшинству с любой из выложенных в стол

\*/

if (\_count < 6

&& \_count - \_covered <= Defender().SizeOfHand()

&& index <= \_attacker.SizeOfHand() - 1

&& (\_count == 0

|| \_field.Any(x => (x.Item1?.Value == \_attacker.Hand[index].Value

|| x.Item2?.Value == \_attacker.Hand[index].Value))))

{

// сформировать сообщение о сделанном ходе

string log = "Игрок " + (\_attacker == \_p1 ? "1" : "2")

+ " подкинул карту " + Values.Ranks[\_p1.Hand[index].Value]

+ '|' + \_p1.Hand[index].Suit.ToString() + ".";

// и записать его в протокол

\_log.WriteLine(log);

// добавить на стол выбранную карту

\_field[\_count].Item1 = \_attacker.Hand[index];

// и убрать ее у игрока

\_attacker.RemoveCard(index);

}

}

// метод, производящий попытку передать ход отбивающемуся игроку

private void MoveToDefender()

{

/\* Условия допустимости передачи хода отбивающемуся:

\* 1. Нападающий подкинул хотя бы одну карту

\* 2. На стол еще не было выложено 12 карт

\*/

if (\_count > 0 && (\_count + \_covered != 12))

{

// сформировать сообщение о сделанном ходе

string log = "Игрок" + (\_attacker == \_p1 ? "1" : "2")

+ " передал ход игроку" + (\_attacker == \_p1 ? "2" : "1") + ".";

// и записать его в протокол

\_log.WriteLine(log);

// передать ход отбивающемуся

\_turn = Defender();

}

}

// метод, производящий попытку сбросить карты в бито и закончить кон

private void Drop()

{

/\* Условия допустимости объявления бито:

\* 1. Нападающий подкинул хотя бы одну карту

\* 2. Все подкинутые карты были покрыты

\*/

if (\_count > 0 && \_count == \_covered)

{

// сформировать сообщение о сделанном ходе

string log = "Игрок "

+ (\_attacker == \_p1 ? "1" : "2") + " объявил бито.";

// и записать его в протокол

\_log.WriteLine(log);

// определим отбивающегося - он понадобится дальше

Player \_defender = Defender();

// по правилам первый получает карты нападавший

while (\_deck.Size() > 0 && \_attacker.SizeOfHand() < 6)

{

\_deck.GiveCard(\_attacker);

}

// затем отбивавшийся

while (\_deck.Size() > 0 && \_defender.SizeOfHand() < 6)

{

\_deck.GiveCard(\_defender);

}

// если в колоде не осталось карт

// и у игроков не осталось карт, то это ничья, игра заканчивается

if (\_deck.Size() == 0

&& \_attacker.SizeOfHand() == 0

&& \_defender.SizeOfHand() == 0)

{

Console.WriteLine("Ничья!");

\_log.WriteLine("Игра закончилась ничьей.");

\_game = false;

return;

}

// сбросить все карты со стола в бито

\_dropped.AddRange(\_field.Select(x => x.Item1).Where(x => x != null).ToList());

\_dropped.AddRange(\_field.Select(x => x.Item2).Where(x => x != null).ToList());

// и очистить стол

for (int i = 0; i < 6; ++i)

{

\_field[i] = (null, null);

}

// передать ход

\_attacker = \_defender;

\_turn = \_attacker;

// обнулить счетчики карт

\_count = 0;

\_covered = 0;

}

}

// метод, производящий попытку покрыть картой

private void Cover(int index)

{

/\* Условия допустимости покрытия картой:

\* 1. Не все карты еще были покрыты

\* 2. Выбранная карта совпадает с покрываемой по масти и старше нее

\* ИЛИ

\* 3. Выбранная карта козырная, покрываемая не козырная

\*/

if (\_covered < \_count && index <= \_turn.SizeOfHand() - 1

&& ((\_turn.Hand[index].Suit == \_field[\_covered].Item1.Suit

&& \_turn.Hand[index].Value > \_field[\_covered].Item1.Value)

|| (\_turn.Hand[index].Suit == \_trump.Suit

&& \_field[\_covered].Item1.Suit != \_trump.Suit)))

{

// сформировать сообщение о сделанном ходе

string log = "Игрок " + (\_turn == \_p1 ? "1" : "2") + " покрыл карту"

+ Values.Ranks[\_field[\_covered].Item1.Value] + '|'

+ \_field[\_covered].Item1.Suit.ToString() + " картой "

+ Values.Ranks[\_turn.Hand[index].Value] + '|'

+ \_turn.Hand[index].Suit.ToString();

// и записать его в протокол

\_log.WriteLine(log);

// выложить выбранную карту на стол

\_field[\_covered].Item2 = \_p1.Hand[index];

// убрать ее из руки игрока

\_turn.RemoveCard(index);

++\_covered;

}

}

// метод, производящий попытку перевести карты

private void Turn(int index)

{

/\* Условия допустимости перевода:

\* 1. Еще ни одной карты не было покрыто

\* 2. Выбранная карта имеет то же старшинство, что и подкинутые

\*/

if (\_covered == 0

&& index <= \_turn.SizeOfHand() - 1

&& \_turn.Hand[index].Value == \_field[0].Item1.Value)

{

// сформировать сообщение о сделанном ходе

string log = "Игрок " + (\_turn == \_p1 ? "1" : "2")

+ " перевел картой " + Values.Ranks[\_turn.Hand[index].Value]

+ '|' + \_p1.Hand[index].Suit.ToString();

// и записать его в протокол

\_log.WriteLine(log);

// добавить выбранную карту на стол

\_field[\_count].Item1 = \_turn.Hand[index];

// убрать ее из руки игрока

\_turn.RemoveCard(index);

// перевести стрелки и передать ход

++\_count;

Player \_defender = Defender();

\_turn = \_attacker;

\_attacker = \_defender;

}

}

// метод, производящий попытку взять карты и закончить кон

private void Take()

{

/\* Условия, допускающие взятие карт:

\* 1. Еще не все карты были побиты

\*/

if (\_count != \_covered)

{

// если колода пуста и рука нападавшего пуста, нападавший победил

if (\_deck.Size() == 0 && \_attacker.SizeOfHand() == 0)

{

// вывести сообщение о победе на экран

Console.WriteLine("Игрок {0} победил!", \_attacker == \_p1 ? "1" : "2");

// сформировать сообщение о победе

string msg = "В игре победил игрок " + (\_attacker == \_p1 ? "1" : "2") + ".";

// и записать его в протокол

\_log.WriteLine(msg);

\_game = false;

return;

}

// сформировать сообщение о сделанном ходе

string log = "Игрок " + (\_turn == \_p1 ? "1" : "2") + " забрал карты.";

// и записать его в протокол

\_log.WriteLine(log);

// отдать отбивавшемуся игроку все карты со стола

\_turn.Hand.AddRange(\_field.Select(x => x.Item1)

.Where(x => x != null).ToList());

\_turn.Hand.AddRange(\_field.Select(x => x.Item2)

.Where(x => x != null).ToList());

// докинуть карты нападавшему

while (\_deck.Size() > 0 && \_attacker.SizeOfHand() < 6)

{

\_deck.GiveCard(\_attacker);

}

// докинуть карты отбивавшемуся

while (\_deck.Size() > 0 && \_turn.SizeOfHand() < 6)

{

\_deck.GiveCard(\_turn);

}

// очистить стол

for (int i = 0; i < 6; ++i)

{

\_field[i] = (null, null);

}

// обнулить счетчики, передать ход

\_count = 0;

\_covered = 0;

\_turn = \_attacker;

}

}

// метод, производящий попытку передать ход нападающему игроку

private void MoveToAttacker()

{

/\* Условия допустимости передачи хода:

\* 1. Все выложенные карты были побиты

\*/

if (\_covered == \_count)

{

// сформировать сообщение о сделанном ходе

string log = "Игрок " + (\_turn == \_p1 ? "1" : "2")

+ " передал ход игроку " + (\_turn == \_p1 ? "2" : "1") + ".";

// и записать его в протокол

\_log.WriteLine(log);

\_turn = \_attacker;

}

}

// метод вывода игрового стола

public void Display()

{

Console.Clear();

// строка 1

Console.WriteLine("Игровой стол:");

// строка 2

Console.WriteLine("+----------------------+");

// строка 3

Console.WriteLine("| |");

// строка 4

Console.Write("| ");

if (\_field[0].Item2 != null) { \_field[0].Item2.Display(); }

else { Console.Write(" "); }

Console.Write(" ");

if (\_field[1].Item2 != null) { \_field[1].Item2.Display(); }

else { Console.Write(" "); }

Console.Write(" ");

if (\_field[2].Item2 != null) { \_field[2].Item2.Display(); }

else { Console.Write(" "); }

Console.WriteLine(" | +" + (\_trump.Value == 10 ? "------+" : "----- +"));

// строка 5

Console.Write("| ");

if (\_field[0].Item1 != null) { \_field[0].Item1.Display(); }

else { Console.Write(" "); }

Console.Write(" ");

if (\_field[1].Item1 != null) { \_field[1].Item1.Display(); }

else { Console.Write(" "); }

Console.Write(" ");

if (\_field[2].Item1 != null) { \_field[2].Item1.Display(); }

else { Console.Write(" "); }

Console.WriteLine(" | |" + (\_trump.Value == 10 ? " |" : " |"));

// строка 6

Console.WriteLine("| | | " +

Values.Ranks[\_trump.Value] + "|" + \_trump.Suit.ToString() + " |");

// строка 7

Console.Write("| ");

if (\_field[3].Item2 != null) { \_field[3].Item2.Display(); }

else { Console.Write(" "); }

Console.Write(" ");

if (\_field[4].Item2 != null) { \_field[4].Item2.Display(); }

else { Console.Write(" "); }

Console.Write(" ");

if (\_field[5].Item2 != null) { \_field[5].Item2.Display(); }

else { Console.Write(" "); }

Console.WriteLine(" | |" + (\_trump.Value == 10 ? " |" : " |"));

// строка 8

Console.Write("| ");

if (\_field[3].Item1 != null) { \_field[3].Item1.Display(); }

else { Console.Write(" "); }

Console.Write(" ");

if (\_field[4].Item1 != null) { \_field[4].Item1.Display(); }

else { Console.Write(" "); }

Console.Write(" ");

if (\_field[5].Item1 != null) { \_field[5].Item1.Display(); }

else { Console.Write(" "); }

Console.WriteLine(" | +" + (\_trump.Value == 10 ? "------+" : "-----+"));

// строки 9 и 10

Console.WriteLine("| |\n+----------------------+");

}

}

}

Program.cs

namespace Game

{

internal class Program

{

static void Main(string[] args)

{

Game game = new Game(); // создаем новый объект и начинаем играть

}

}

}

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были разработаны две программы, основанные на принципах объектно-ориентированного подхода.

Были закреплены навыки работы с файлами, интерфейсами, абстрактными классами, также были закреплены знания о механизмах наследования. Получен опыт в разработке приложений, моделирующих сложные и непостоянные процессы.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Microsoft Learn – сеть разработчиков Microsoft. URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 20.12.2023)

2. Горячев А. В., Кравчук Д. К., Новакова Н. Е. Объектно-ориентированное моделирование. Учеб. Пособие. СПб.: Изд-во СПбГЭТУ “ЛЭТИ”, 2010.

3. Новакова Н. Е., Горячев А. В. Моделирование коммуникативных процессов в распределенных САПР. Учеб. Пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2010.

4. Язык UML = The Unified Modeling Language User Guide : руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон.

5. Полное руководство по языку программирования С# 12 и платформе .NET 8. URL: <https://metanit.com/sharp/tutorial/> (дата обращения: 09.12.2023)