

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В. И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра информационной безопасности**

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
**к курсовой работе**  
**по дисциплине «Объектно-ориентированное программирование»**  
**Тема: «Разработка приложения на основе принципов объектно-**  
**ориентированного подхода»**

Студент гр. 3363

\_\_\_\_\_

Минко Д. А.

Преподаватель

\_\_\_\_\_

Новакова Н. Е.

Санкт-Петербург

2024

## ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент: Минко Д. А.

Группа: 3363

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 80 страниц.

Дата выдачи задания:

Дата сдачи курсовой работы:

Дата защиты курсовой работы:

Студент гр. 3363

\_\_\_\_\_

Минко Д. А.

Преподаватель

\_\_\_\_\_

Новакова Н. Е.

## **АННОТАЦИЯ**

Данная работа содержит в себе решение трех задач на основе объектно-ориентированного подхода. В первой задаче рассмотрены механизмы наследования и иерархия классов, во второй – работа с графами, в третьей – 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 three problems based on an object-oriented approach. In the first task, inheritance mechanisms and class hierarchy are considered, in the second - working with graphs, in the third - developing a simulation model.

Based on these models, applications have been developed that include various user interfaces. The results obtained are presented in the work.

## СОДЕРЖАНИЕ

АННОТАЦИЯ .....	3
ВВЕДЕНИЕ.....	5
ЦЕЛЬ РАБОТЫ.....	5
1. ПЕРВЫЙ РАЗДЕЛ.....	7
1.1. Формулировка задания .....	7
1.2. Теоретический аспект задачи .....	7
1.3. Формализация задачи.....	8
1.4. Спецификация классов .....	9
1.5. Руководство оператора.....	11
1.6. Руководство программиста .....	12
1.7. Контрольный пример .....	12
1.8. Листинг программы.....	13
2. ВТОРОЙ РАЗДЕЛ.....	17
2.1. Формулировка задания .....	17
2.2. Теоретический аспект задачи .....	17
2.3. Формализация задачи.....	19
2.4. Спецификация классов .....	22
2.5. Руководство оператора.....	24
2.6. Руководство программиста .....	27
2.7. Контрольный пример .....	Ошибка! Закладка не определена.
2.8. Листинг программы.....	67
ЗАКЛЮЧЕНИЕ .....	84
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	85

## **ВВЕДЕНИЕ**

Курсовая работа направлена на создание приложений на основе объектно-ориентированного подхода на языке C#. В ней рассматриваются иерархии классов и наследования, работа с графами, а также имитационные модели. Курсовая работа состоит из 3 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 туров.
- Второй раздел: для заданного двудольного графа найти число совершенных паросочетаний  $P$  и одно из наибольших паросочетаний. Задачу решить в общем виде. В качестве контрольного примера использовать задание своего варианта.
- Третий раздел: требуется создать компьютерную модель обслуживания потока заявок на разгрузку, поступающих от грузовых судов (сухогрузов и танкеров), прибывающих в морской порт. Грузовые суда прибывают в порт согласно расписанию, но возможны опоздания и досрочные прибытия. Расписание включает день и время прибытия, название судна, вид груза и его вес, а также планируемый срок стоянки в порту для разгрузки.

Для разгрузки судов в порту используются три вида разгрузочных кранов, соответствующих трем видам грузов: сыпучим и жидким грузам, контейнерам. Число разгрузочных кранов каждого вида ограничено, так что поступающие заявки на разгрузку одного вида груза образуют очередь. Длительность разгрузки судна зависит от вида и веса его груза, а также некоторых других факторов, например, погодных условий. Любой дополнительный (сверх запланированного срока) день стояния судна в порту (из-за ожидания разгрузки

в очереди или из-за задержки самой разгрузки) влечет за собой выплату штрафа (например, 2 тыс. у.е. за каждый дополнительный день простоя судна).

При моделировании прибытия судов отклонение их от расписания рассматривается как случайная величина с равномерным распределением в некотором интервале (например, от -2 до 9 дней). Еще одной случайной величиной, изменяющейся в фиксированном диапазоне (например, от 0 до 12 дней), является время задержки окончания разгрузки судна по сравнению с обычным (зависящим только от вида груза и его веса).

Цель моделирования работы морского порта – определение для заданного расписания прибытия судов минимально достаточного числа кранов в порту, позволяющего уменьшить штрафные суммы. Период моделирования – месяц, шаг моделирования – 1-3 дня. В параметры моделирования следует включить расписание прибытия судов, количество кранов каждого вида, диапазоны разброса случайных величин (отклонения от расписания прибытия и отклонения от обычного времени разгрузки), а также шаг моделирования.

Визуализация моделируемого процесса должна предусматривать показ очередей у разгрузочных кранов, приход судов в порт и их отход после разгрузки. Должен быть показан также список произведенных разгрузок, в котором указывается название разгруженного судна, время его прихода в порт и время ожидания в очереди на разгрузку, время начала разгрузки и ее продолжительность.

По окончании моделирования должна быть выведена итоговая статистика: число разгруженных судов, средняя длина очереди на разгрузку, среднее время ожидания в очереди, максимальная и средняя задержка разгрузки, общая сумма выплаченного штрафа. Результат сбора статистики должен быть выведен в текстовый файл.

## **1. ПЕРВЫЙ РАЗДЕЛ**

### **1.1. Формулировка задания**

Вариант 22

Разработать программу для обеспечения продажи туров.

### **1.2. Теоретический аспект задач**

Перед реализацией программы необходимо теоретически определить структуру. Для этого принято решение создать абстрактный класс «Tour», который будет представлять общую модель тура. Этот класс будет содержать свойства, определяющие тур как сущность: его идентификационный номер, название тура, дата начала, продолжительность тура и его цена. Кроме того, класс «Tour» будет реализовывать метод «Print», который будет выводить информацию о туре в презентабельном виде, а также реализует интерфейс «IPrintable».

Далее будут созданы три наследующих класса от «Tour», каждый из которых будет представлять отдельный тип тура (пляжный тур, городской тур, приключенческий тур). Эти классы будут содержать дополнительную информацию о типе тура, а метод «Print» будет переопределен для каждого из них. В каждом из этих классов будет добавлено статическое поле, представляющее цену тура для конкретного типа, и это поле будет иметь уникальное значение для каждого типа тура.

### 1.3. Формализация задачи

Далее представлена диаграмма классов (Рисунок 2).

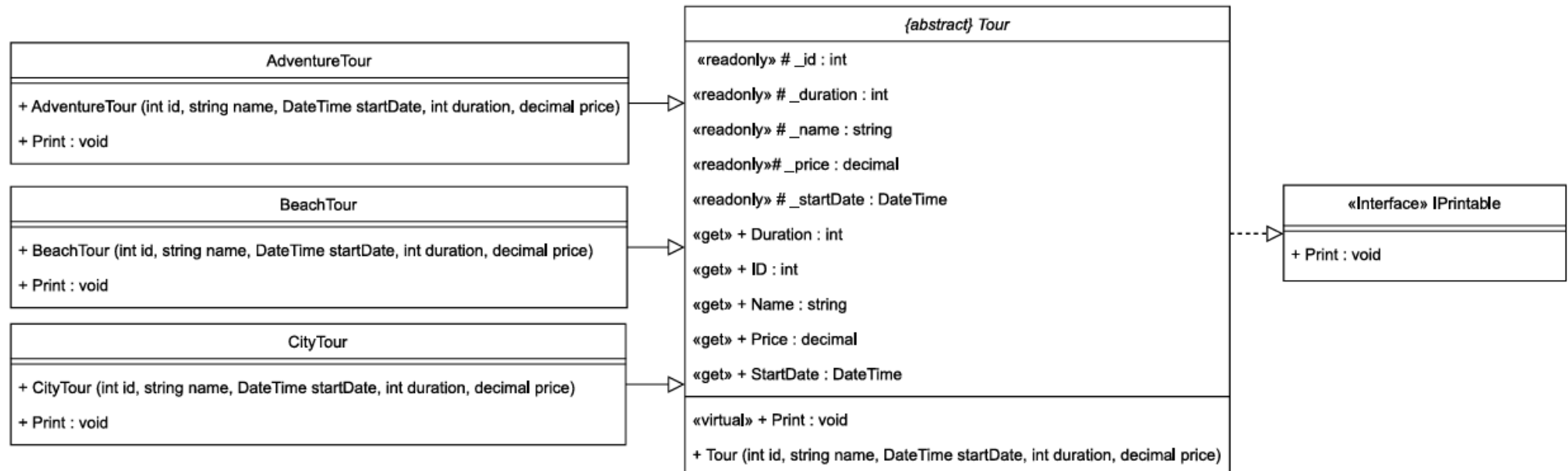


Рисунок 1 – Диаграмма классов для первого раздела



Основой программы является абстрактный класс «Tour», который описывает общую модель тура. Поля этого класса включают в себя основные свойства тура (идентификационный номер, название, дата начала, продолжительность и цена). Конструктор этого класса инициализирует эти поля значениями, и они могут быть доступны только для чтения.

Кроме того, в классе «Tour» есть метод «Print», который реализует интерфейс «IPrintable» и выводит информацию о туре в презентабельном виде. Также реализована логика для определения типа тура через методы классов-наследников.

Для конкретных типов туров (пляжный тур, городской тур, приключенческий тур) создаются производные классы «BeachTour», «CityTour» и «AdventureTour». Эти классы содержат статическое поле «Price», которое задает стоимость тура для каждого типа. Конструктор каждого из этих классов передает параметры в конструктор базового класса, поскольку все эти параметры одинаковы для всех типов туров. Метод «Print» в каждом из классов перегружается, чтобы дополнительно выводить цену тура, после вызова метода «Print» базового класса.

Программа выводит информацию о турах, включая их цену, и позволяет пользователю легко различать типы туров, которые предлагаются для продажи.

#### 1.4. Спецификация классов

Рассмотрим спецификацию классов данного проекта.

Имя	Тип	Модификатор доступа	Назначение
_id	int	protected	Идентификационный номер тура
_name	string	protected	Название тура
_startDate	DateTime	protected	Дата начала тура
_duration	int	protected	Продолжительность тура (в днях)
_price	decimal	protected	Цена тура

Таблица 1 – Описание полей класса Tour

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
Tour	-	public	int id, string name, DateTime startDate, int duration, decimal price	-	Конструктор класса для инициализации полей
Print	void	public	-	-	Вывод информации о туре

Таблица 2 – Описание методов класса Tour

Имя	Тип	Модификатор доступа	Назначение
_price	decimal	private	Цена пляжного тура

Таблица 3 – Описание полей класса BeachTour

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
BeachTour	-	public	int id, string name, DateTime startDate, int duration, decimal price	-	Конструктор для инициализации полей класса
print	void	public	-	-	Переопределение метода для вывода информации о пляжном туре

Таблица 4 – Описание методов класса BeachTour

Имя	Тип	Модификатор доступа	Назначение
_price	decimal	private	Цена городского тура

Таблица 5 – Описание полей класса CityTour

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
-------	------------------	---------------------	-------------------	--------------------	------------

CityTour	-	public	int id, string name, DateTime startDate, int duration, decimal price	-	Конструктор для инициализации полей класса
Print	void	public	-	-	Переопределение метода для вывода информации о городском туре

Таблица 6 – Описание методов класса CityTour

Имя	Тип	Модификатор доступа	Назначение
_price	decimal	private	Цена приключенческого тура

Таблица 7 – Описание полей класса AdventureTour

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
AdventureTour	-	public	int id, string name, DateTime startDate, int duration, decimal price	-	Конструктор для инициализации полей класса
Print	void	public	-	-	Переопределение метода для вывода информации о приключенческом туре

Таблица 8 – Описание методов класса AdventureTour

### 1.5. Руководство оператора

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

## 1.6. Руководство программиста

Все логически отдельные структуры находятся в отдельных 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 Tour является абстрактным, поэтому создание экземпляра класса Tour является недопустимым.

## 1.7. Контрольный пример

Далее представлены результаты работы программы (Рисунок 3).



```
Консоль отладки Microsoft V × +
Tour ID: 1
Name: Maldives Getaway
Start Date: 01.12.2024
Duration: 7 days
Price: 49999,99 P
Type: Beach Tour

Tour ID: 2
Name: Paris Explorer
Start Date: 15.05.2024
Duration: 5 days
Price: 34999,99 P
Type: City Tour

Tour ID: 3
Name: Amazon Jungle Trek
Start Date: 20.08.2024
Duration: 10 days
Price: 59999,99 P
Type: Adventure Tour
```

Рисунок 2 – Контрольный пример работы программы

Пояснение: в методе «Main» создаются три различных тура («BeachTour», «CityTour», «AdventureTour»), и через конструктора им присваиваются свои данные. Затем для каждого объекта по очереди вызывается метод Print, который выводит на экран пользователя информацию о созданных турах. Информация включает идентификационный номер тура, название, дату начала, продолжительность тура, цену и тип тура (например, «Beach Tour», «City Tour», «Adventure Tour»).

## 1.8. Листинг программы

### Program.cs

```
namespace TourSales
{
    internal class Program
    {
        static void Main(string[] args)
        {
            BeachTour beachTour = new BeachTour(1, "Maldives Getaway", new
DateTime(2024, 12, 1), 7, 49999.99M);
            CityTour cityTour = new CityTour(2, "Paris Explorer", new
DateTime(2024, 5, 15), 5, 34999.99M);
            AdventureTour adventureTour = new AdventureTour(3, "Amazon Jungle
Trek", new DateTime(2024, 8, 20), 10, 59999.99M);

            beachTour.Print();
            Console.WriteLine();
            cityTour.Print();
            Console.WriteLine();
            adventureTour.Print();
        }
    }
}
```

### IPrintable.cs

```
namespace TourSales
{
    internal interface IPrintable
    {
        void Print();
    }
}
```

```
}  
}
```

## Tour.cs

```
namespace TourSales  
{  
    internal abstract class Tour : IPrintable  
    {  
        protected readonly int _id; // ID тура  
        protected readonly string _name; // Название тура  
        protected readonly DateTime _startDate; // Дата начала тура  
        protected readonly int _duration; // Продолжительность тура в днях  
        protected readonly decimal _price; // Цена тура  
  
        public int ID { get { return _id; } }  
        public string Name { get { return _name; } }  
        public DateTime StartDate { get { return _startDate; } }  
        public int Duration { get { return _duration; } }  
        public decimal Price { get { return _price; } }  
  
        public Tour(int id, string name, DateTime startDate, int duration,  
decimal price)  
        {  
            _id = id;  
            _name = name;  
            _startDate = startDate;  
            _duration = duration;  
            _price = price;  
        }  
  
        public virtual void Print()  
        {  
            Console.WriteLine("Tour ID: {0}\nName: {1}\nStart Date:  
{2}\nDuration: {3} days\nPrice: {4} P",  
ID, Name, StartDate.ToShortDateString(), Duration, Price);  
        }  
    }  
}
```

## AdventureTour.cs

```
namespace TourSales  
{
```

```

        internal class AdventureTour : Tour
        {
            public AdventureTour(int id, string name, DateTime startDate, int
duration, decimal price)
                : base(id, name, startDate, duration, price) { }

            public override void Print()
            {
                base.Print();
                Console.WriteLine("Type: Adventure Tour");
            }
        }
    }
}

```

### CityTour.cs

```

namespace TourSales
{
    internal class CityTour : Tour
    {
        public CityTour(int id, string name, DateTime startDate, int duration,
decimal price)
            : base(id, name, startDate, duration, price) { }

        public override void Print()
        {
            base.Print();
            Console.WriteLine("Type: City Tour");
        }
    }
}

```

### BeachTour.cs

```

namespace TourSales
{
    internal class BeachTour : Tour
    {
        public BeachTour(int id, string name, DateTime startDate, int duration,
decimal price)
            : base(id, name, startDate, duration, price) { }

        public override void Print()
        {

```

```
        base.Print();  
        Console.WriteLine("Type: Beach Tour");  
    }  
}  
}
```



## 2. ВТОРОЙ РАЗДЕЛ

### 2.1. Формулировка задания

Вариант Г-44-3

Для заданного двудольного графа найти число совершенных паросочетаний  $P$  и одно из наибольших паросочетаний. Задачу решить в общем виде. В качестве контрольного примера использовать задание вашего варианта.

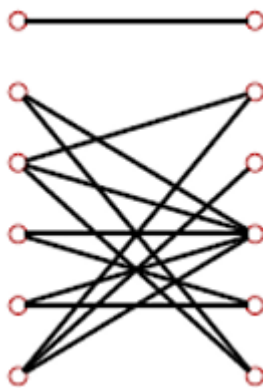


Рисунок 3 – Задание в качестве контрольного варианта

### 2.2. Теоретический аспект задачи

Для решения поставленной задачи требуется изучить принципы нахождения всех паросочетаний в двудольном графе, а также определить число совершенных паросочетаний и одно из наибольших паросочетаний, используя методы теории графов.

Двудольный граф — это граф, в котором множество вершин делится на две непересекающиеся части, так что все рёбра соединяют вершины из разных частей. Для решения задачи необходимо рассматривать все возможные рёбра между вершинами двух частей и искать максимальные паросочетания, то есть такие подмножества рёбер, где каждому элементу из одной части графа соответствует ровно одно ребро из другой части. Важно, что совершенные паросочетания существуют только в графах с четным числом вершин, поскольку для этого требуется, чтобы все вершины были задействованы в рёбрах.

Для нахождения числа совершенных паросочетаний и вывода одного из наибольших паросочетаний, применяется метод поиска всех паросочетаний в графе с последующей сортировкой и фильтрацией по размеру. В этом контексте также учитываются дополнительные параметры, такие как сортировка по количеству пар в паросочетаниях и нахождение максимальных по размеру паросочетаний.

1. Таким образом, решение задачи включает в себя реализацию методов для:
2. Ввода вершин и рёбер;
3. Поиска всех возможных паросочетаний;
4. Сортировки паросочетаний по размеру и выборки одного из наибольших;
5. Вычисления числа совершенных паросочетаний в графе.

Эти условия были учтены при разработке, что позволило эффективно решать задачу для различных графов.

## 2.3. Формализация задачи

### 2.3.1. Описание классов.

Далее представлена диаграмма классов (Рисунок 4).

BipartiteGraphs
- adjacencyList : Dictionary<string, List<string>>
+ AddEdge(string vertex1, string vertex2) : void + AddVertex(string vertex) : void + BipartiteGraphs() + FindAllMatchings() : List<List<Tuple<string, string>>> + FindAndPrintAllMatchings() : void - FindMatchings(List<Tuple<string, string>> currentMatching, List<List<Tuple<string, string>>> allMatchings, HashSet<string> usedVertices, List<Tuple<string, string>> allEdges) : void + InputAllEdge() : void + IsPerfectMatching() : int + PrintAllMatching(List<List<Tuple<string, string>>> allMatchings) : void + PrintMaxMatching() : void + PrintSortedMatchingsBySizeAndInside() : void + PrintSortedMatchingsBySizeDescending() : void + RemoveEdge(string vertex1, string vertex2) : void + RemoveVertex(string vertex) : void + SortMatchingsBySizeAndInside(List<List<Tuple<string, string>>> allMatchings) : List<List<Tuple<string, string>>> + SortMatchingsBySizeDescending(List<List<Tuple<string, string>>> allMatchings) : List<List<Tuple<string, string>>>

Рисунок 4 – Диаграмма классов для второго раздела

Класс BipartiteGraphs является основной частью реализации задачи и описывает двудольный граф, его вершины и рёбра, а также алгоритмы для поиска паросочетаний и проверки на совершенность паросочетаний. В нем используется представление графа через список смежности, который хранится в поле adjacencyList – это словарь, где ключами являются вершины, а значениями – списки смежных вершин.

Класс предоставляет несколько методов для работы с графом. Метод `AddVertex` добавляет новую вершину в граф, при этом, если вершина уже существует, она не добавляется повторно. Метод `RemoveVertex` удаляет вершину и все рёбра, связанные с этой вершиной, из списка смежности других вершин. Метод `AddEdge` добавляет рёбра между двумя существующими вершинами, а метод `RemoveEdge` удаляет ребро между двумя вершинами, если оно существует. Для ввода данных, метод `InputAllVertex` позволяет пользователю вводить вершины, пока не будет введена команда "END", а метод `InputAllEdge` позволяет вводить рёбра между уже добавленными вершинами до команды "END". Метод `PrintAdjacencyList` выводит список смежности графа, показывая для каждой вершины все вершины, с которыми она соединена рёбрами.

Основные алгоритмы нахождения паросочетаний реализованы через методы `FindMatchings`, `FindAllMatchings` и другие. Метод `FindMatchings` рекурсивно находит все возможные паросочетания в графе, перебирая рёбра и добавляя их в текущее паросочетание, если обе вершины еще не использованы. После добавления ребра метод продолжает искать остальные паросочетания, пока не переберет все возможные рёбра. Каждый найденный набор рёбер, представляющий паросочетание, добавляется в список `allMatchings`. Метод `FindAllMatchings` собирает все рёбра графа и передает их в `FindMatchings` для поиска всех возможных паросочетаний. Метод `PrintAllMatchings` выводит все найденные паросочетания в формате списков рёбер (`vertex1, vertex2`). Метод `PrintMaxMatching` находит и выводит одно из наибольших паросочетаний (по числу рёбер), сортируя все паросочетания по количеству рёбер и выводя одно из самых больших. Метод `IsPerfectMatching` проверяет, является ли граф совершенным паросочетанием, то есть паросочетанием, в котором количество рёбер равно половине общего числа вершин в графе. Граф должен иметь чётное количество вершин для существования совершенного паросочетания, и метод ищет такие паросочетания и возвращает их количество или 0, если их нет.

Методы `SortMatchingsBySizeAndInside` и `SortMatchingsBySizeDescending` предоставляют функционал сортировки паросочетаний. Первый сортирует

паросочетания по количеству рёбер в каждом паросочетании, а внутри каждого паросочетания – по возрастанию рёбер. Второй метод делает аналогичную сортировку, но в порядке убывания числа рёбер. Для вывода отсортированных паросочетаний используются методы `PrintSortedMatchingsBySizeAndInside` и `PrintSortedMatchingsBySizeDescending`, которые выводят паросочетания в порядке возрастания и убывания количества рёбер соответственно.

### **2.3.2. Описание алгоритмов.**

В классе `BipartiteGraphs` реализуется решение задачи поиска паросочетаний в двудольном графе. Алгоритм начинается с создания и инициализации графа, в котором вершины и рёбра добавляются с помощью соответствующих методов. Каждый граф представляется в виде списка смежности, где для каждой вершины хранится список её соседей.

После того как граф был создан, алгоритм приступает к поиску паросочетаний. В классе есть рекурсивный метод `FindMatchings()`, который находит все возможные паросочетания. Он перебирает рёбра графа, добавляя их в текущее паросочетание, если вершины ещё не были использованы. После добавления рёбер в паросочетание, метод рекурсивно продолжает искать другие возможные рёбра, избегая пересечения вершин. Полученные паросочетания сохраняются в список.

Далее, для нахождения наибольших паросочетаний, алгоритм сортирует найденные паросочетания по количеству рёбер. Одно из наибольших паросочетаний выводится с помощью метода `PrintMaxMatching()`.

Для нахождения числа совершенных паросочетаний используется метод `IsPerfectMatching()`, который проверяет, является ли паросочетание совершенным. Совершенным считается паросочетание, количество рёбер в котором равно половине числа вершин в графе (если число вершин чётное). Метод подсчитывает и выводит количество таких паросочетаний.

Ключевые моменты алгоритма:

1. представление графа: граф хранится как список смежности;

2. поиск всех паросочетаний: рекурсивный перебор рёбер с добавлением в паросочетание;

3. поиск наибольшего паросочетания: сортировка паросочетаний по количеству рёбер и вывод одного из наибольших;

4. число совершенных паросочетаний: поиск совершенных паросочетаний, где количество рёбер равно половине числа вершин.

Программа эффективно решает задачу нахождения всех паросочетаний, одно из наибольших и вычисления числа совершенных паросочетаний для заданного двудольного графа.

## 2.4. Спецификация классов

Имя	Тип	Модификатор доступа	Назначение
adjacencyList	Dictionary<string, List<string>>	private	Список смежности, хранящий вершины и рёбра графа

Таблица 9 – Описание полей класса BipartiteGraphs

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
BipartiteGraphs	-	public	-	-	Конструктор класса, инициализирует список смежности
AddVertex	void	public	string vertex	-	Добавляет вершину в граф, если она ещё не существует
RemoveVertex	void	public	string vertex	-	Удаляет вершину из графа, а также все рёбра, связанные с ней
AddEdge	void	public	string vertex1, string vertex2	-	Добавляет ребро между двумя вершинами,

					если они существуют
RemoveEdge	void	public	string vertex1, string vertex2	-	Удаляет ребро между двумя вершинами, если оно существует
InputAllVertex	void	public	-	-	Вводит все вершины для графа, завершение ввода по команде "END"
InputAllEdge	void	public	-	-	Вводит рёбра графа в формате пар вершин, завершение ввода по команде "END"
PrintAdjacencyList	void	public	-	-	Выводит список смежности для всех вершин графа
FindMatchings	void	private	List<Tuple<string, string>> currentMatching, List<List<Tuple<string, string>>> allMatchings, HashSet<string> usedVertices, List<Tuple<string, string>> allEdges	-	Рекурсивный метод для нахождения всех паросочетаний в графе
FindAllMatchings	List<List<Tuple<string, string>>>	public	-	-	Находит все возможные паросочетания в графе
PrintAllMatchings	void	public	List<List<Tuple<string, string>>> allMatchings	-	Выводит все найденные паросочетания в графе
FindAndPrintAllMatchings	void	public	-	-	Находит и выводит все паросочетания в графе
SortMatchingsBySizeAndInside	List<List<Tuple<string, string>>>	public	List<List<Tuple<string, string>>> allMatchings	-	Сортирует паросочетания по количеству пар и сортирует пары

					внутри каждого паросочетания
PrintSortedMatchingsBySizeAndInside	void	public	-	-	Выводит отсортированные по размеру и внутреннему порядку паросочетания
PrintMaxMatching	void	public	-	-	Выводит одно из наибольших паросочетаний (по количеству пар)
SortMatchingsBySizeDescending	List<List<Tuple<string, string>>>	public	List<List<Tuple<string, string>>> allMatchings	-	Сортирует паросочетания по убыванию числа пар и внутри паросочетаний
PrintSortedMatchingsBySizeDescending	void	public	-	-	Выводит отсортированные паросочетания по убыванию числа пар

Таблица 10 – Описание методов класса BipartiteGraphs

## 2.5. Руководство оператора

После запуска программа предложит пользователю ввести вершины и рёбра графа.

1. Ввод вершин: вершины вводятся по одной, каждая на новой строке, через клавишу «Enter». Для завершения ввода вершин необходимо ввести команду «END»;

2. Ввод рёбер: далее программа предложит ввести рёбра. Для этого нужно указать две вершины, которые соединяет данное ребро, разделяя их пробелом;

а. Если одна из вершин не существует в графе, программа предложит добавить её. Для этого нужно подтвердить добавление, введя “Y”, или отказаться, введя “N”;

б. Если пользователь согласится добавить вершину, она будет добавлена в граф, и программа попросит повторно ввести ребро с уже добавленной вершиной;



3. Автоматическое построение графа: после завершения ввода вершин и рёбер, программа автоматически построит граф и выведет на экран список смежности, представляющий введённую структуру графа;

4. Поиск паросочетаний: программа также выполнит поиск всех паросочетаний в графе, выведет одно из максимальных паросочетаний и рассчитает количество совершенных паросочетаний, если таковые имеются.

Таким образом, процесс взаимодействия с программой включает в себя последовательный ввод данных, подтверждения на добавление вершин и рёбер, и автоматическую обработку для построения и анализа графа.

```
Вводите вершины (для окончания ввода введите "END"):  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
END  
  
Вводите по две ранее добавленные вершины через пробел, разделяя пары "Enter",  
между которыми находятся ребра (для окончания ввода введите "END"):  
1 7  
2 10  
2 12  
Vertex2 – не существует, для начала добавьте вершину  
Хотите добавить вершину "12"? (Y/N)  
Y  
Вершина добавлена, можете добавлять ребро к этой вершине  
2 12  
3 8  
3 10  
3 12  
4 10  
4 11  
5 10  
5 11  
6 8  
6 9  
6 10  
END  
Ввод завершён.
```

Рисунок 5.1 – Интерфейс ввода вершин и ребер

```
Ваш введенный граф:  
1: 7  
2: 10, 12  
3: 8, 10, 12  
4: 10, 11  
5: 10, 11  
6: 8, 9, 10  
7: 1  
8: 3, 6  
9: 6  
10: 2, 3, 4, 5, 6  
11: 4, 5  
12: 2, 3
```

Рисунок 5.2 – Вывод графа

```
Одно из наибольших паросочетаний: 6 пар  
(1, 7), (10, 5), (11, 4), (12, 2), (3, 8), (6, 9)  
Число совершенных паросочетаний: 2
```

Рисунок 5.3 – Итоговый вывод

На предоставленных рисунках видно, что все действия, требуемые от пользователя, подробно описываются в консоли. Пользователю необходимо поочерёдно ввести вершины графа, а затем рёбра, соединяющие эти вершины. Если при вводе ребра будет указана вершина, которая отсутствует в графе, программа автоматически уведомит об этом пользователя и предложит добавить новую вершину, предоставляя возможность подтвердить или отказаться от этого действия.

## **2.6. Руководство программиста**

### **2.6.1. Введение**

Проект «BipartiteGraphs» представляет собой консольное приложение на языке C#, предназначенное для работы с двудольными графами. Программа позволяет пользователю вводить вершины и рёбра, строить граф, выводить список смежности, находить паросочетания (включая наибольшее) и определять число совершенных паросочетаний.

### **2.6.2. Структура проекта**

- Class1.cs: Основной функционал для работы с графами, включая создание, редактирование и анализ графов;
- Program.cs: Главная программа, управляющая взаимодействием с пользователем.

### **2.6.3. Основные классы**

Класс BipartiteGraphs

Этот класс предоставляет функционал для работы с двудольными графами.

Конструкторы:

- BipartiteGraphs() – инициализирует объект, создавая пустой граф (в виде словаря смежности).

Основные методы:

#### **1. Работа с вершинами**

- AddVertex(string vertex)

Добавляет вершину в граф. Если вершина уже существует, ничего не происходит.

- RemoveVertex(string vertex)

Удаляет вершину и все рёбра, связанные с ней.

#### **2. Работа с рёбрами**

- AddEdge(string vertex1, string vertex2)

Добавляет ребро между двумя вершинами. Если хотя бы одна из вершин не существует, пользователю предлагается её создать.

- `RemoveEdge(string vertex1, string vertex2)`

Удаляет ребро между двумя указанными вершинами, если оно существует.

### 3. Ввод данных

- `InputAllVertex()`

Позволяет пользователю вводить вершины графа. Для завершения ввода используется команда END.

- `InputAllEdge()`

Позволяет пользователю вводить рёбра графа. Для завершения ввода используется команда END.

### 4. Вывод данных

- `PrintAdjacencyList()`

Выводит список смежности графа.

### 5. Работа с паросочетаниями

- `FindAllMatchings()`

Находит все паросочетания графа.

- `PrintAllMatchings(List<List<Tuple<string, string>>> allMatchings)`

Выводит найденные паросочетания.

- `FindAndPrintAllMatchings()`

Находит и выводит все паросочетания.

- `SortMatchingsBySizeAndInside(List<List<Tuple<string, string>>> allMatchings)`

Сортирует паросочетания по количеству пар и внутреннему порядку.

- `PrintSortedMatchingsBySizeAndInside()`

Выводит отсортированные паросочетания.

- `PrintMaxMatching()`

Выводит одно из максимальных паросочетаний.

- `IsPerfectMatching()`

Определяет число совершенных паросочетаний в графе. Если граф не подходит для поиска совершенных паросочетаний, метод уведомляет пользователя.

#### **2.6.4. Класс Program**

Главный класс программы. Реализует взаимодействие с пользователем.

Точка входа:

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

Алгоритм работы программы:

1. Инициализация графа: `BipartiteGraphs graphs = new BipartiteGraphs();`
2. Ввод вершин: `graphs.InputAllVertex();`
3. Ввод рёбер: `graphs.InputAllEdge();`
4. Вывод списка смежности графа: `graphs.PrintAdjacencyList();`
5. Вывод максимального паросочетания: `graphs.PrintMaxMatching();`
6. Вывод числа совершенных паросочетаний:  
`graphs.IsPerfectMatching().`

#### **2.6.5. Особенности и ограничения**

1. Обработка ошибок:

Программа контролирует ввод и уведомляет о неверных данных (например, если вершина или ребро не существует).

2. Двудольные графы:

Методы ориентированы на работу с двудольными графами, но программа может работать с произвольными графами в случае небольшой доработки.

3. Сложность:

Алгоритмы нахождения паросочетаний могут быть ресурсозатратными для больших графов.

### 2.6.6. Рекомендации

- Проверяйте корректность вводимых данных (например, избегайте дублирования рёбер).
- Используйте уникальные имена для вершин.
- Для работы с крупными графами может потребоваться оптимизация алгоритмов.

### 2.7. Контрольный пример

Далее представлены результаты работы программы (Рисунки 6.1–6.3).

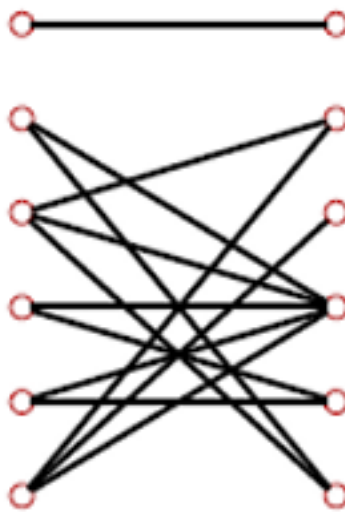


Рисунок 6.1 - Контрольный пример для второго раздела

Исходные данные:

```
Вводите вершины (для окончания ввода введите "END"):  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
END  
  
Вводите по две ранее добавленные вершины через пробел, разделяя пары "Enter",  
между которыми находятся ребра (для окончания ввода введите "END"):  
1 7  
2 10  
2 12  
3 8  
3 10  
3 12  
4 10  
4 11  
5 10  
5 11  
6 8  
6 9  
6 10  
END  
Ввод завершён.
```

Рисунок 6.2– Вводимые данные для контрольного примера

```
Одно из наибольших паросочетаний: 6 пар  
(1, 7), (10, 5), (11, 4), (12, 2), (3, 8), (6, 9)  
Число совершенных паросочетаний: 2
```

Рисунок 6.3 – Результат работы программы для контрольного примера

Пояснение: была запущена программа с начальными данными, где вершины именовались от 1 до 12 начиная с левого столбца → вниз левого столбца, верха правого столбца в его низ.

## 2.8. Листинг кода

### Program.cs

```
namespace BipartiteGraphs;

class Program
{
    static void Main(string[] args)
    {
        BipartiteGraphs graphs = new BipartiteGraphs();
        graphs.InputAllVertex();
        graphs.InputAllEdge();
        Console.WriteLine("Ваш введенный граф:\n");
        graphs.PrintAdjacencyList();
        //graphs.PrintSortedMatchingsBySizeDescending();
        graphs.PrintMaxMatching();
        Console.WriteLine($"Число совершенных паросочетаний:
{graphs.IsPerfectMatching()}");
    }
}
```

### FunctionBipartitegrapg.cs

```
namespace BipartiteGraphs
{
    internal class BipartiteGraphs
    {
        private Dictionary<string, List<string>> adjacencyList;
        public BipartiteGraphs()
        {
            adjacencyList = new Dictionary<string, List<string>>();
        }
        public void AddVertex(string vertex)
        {
            if (!adjacencyList.ContainsKey(vertex))
            {
                adjacencyList[vertex] = new List<string>();
            }
        }
        public void RemoveVertex(string vertex)
        {
            if (adjacencyList.ContainsKey(vertex))
            {
                foreach (var key in adjacencyList.Keys.ToList())
                {
                    adjacencyList[key].Remove(vertex);
                }
                adjacencyList.Remove(vertex);
            }
        }

        public void AddEdge(string vertex1, string vertex2)
        {
            if (!adjacencyList.ContainsKey(vertex1))
            {
                Console.WriteLine("Vertex1 - не существует, для начала добавьте
вершину");
                Console.WriteLine($"Хотите добавить вершину \"{vertex1}\"?
(Y/N)");
                string input;
```



```

do
{
    input = Console.ReadLine();
    if (input == "Y")
    {
        this.AddVertex(vertex1);
        Console.WriteLine("Вершина добавлена, можете добавлять
ребро к этой вершине");
        return;
    }
    else if (input == "N") return;
    else Console.WriteLine("Введите \"Y\" или \"N\"");
} while (true);
return;
}
if (!adjacencyList.ContainsKey(vertex2))
{
    Console.WriteLine("Vertex2 - не существует, для начала добавьте
вершину");
    Console.WriteLine($"Хотите добавить вершину \"{vertex2}\"?
(Y/N)");

    string input;
    do
    {
        input = Console.ReadLine();
        if (input == "Y")
        {
            this.AddVertex(vertex2);
            Console.WriteLine("Вершина добавлена, можете добавлять
ребро к этой вершине");
            return;
        }
        else if (input == "N") return;
        else Console.WriteLine("Введите \"Y\" или \"N\"");
    } while (true);
    return;
}
if (!adjacencyList[vertex1].Contains(vertex2)){
    adjacencyList[vertex1].Add(vertex2);
    adjacencyList[vertex2].Add(vertex1);
}
}

public void RemoveEdge(string vertex1, string vertex2)
{
    if (!adjacencyList.ContainsKey(vertex1))
    {
        Console.WriteLine($"Вершина {vertex1} не существует.");
        return;
    }

    if (!adjacencyList.ContainsKey(vertex2))
    {
        Console.WriteLine($"Вершина {vertex2} не существует.");
        return;
    }

    if (adjacencyList[vertex1].Contains(vertex2))
    {
        adjacencyList[vertex1].Remove(vertex2); // Удаляем ребро из
vertex1
        Console.WriteLine($"Ребро между {vertex1} и {vertex2}
удалено.");
    }
    else

```

```

        {
            Console.WriteLine($"Ребро между {vertex1} и {vertex2} не
существует.");
        }

        if (adjacencyList[vertex2].Contains(vertex1))
        {
            adjacencyList[vertex2].Remove(vertex1); // Удаляем ребро из
vertex2
        }
        else
        {
            Console.WriteLine($"Ребро между {vertex2} и {vertex1} не
существует.");
        }
    }

    public void InputAllVertex()
    {
        Console.WriteLine("Вводите вершины (для окончания ввода введите
\\\"END\\\":");
        string vertex;
        do
        {
            vertex = Console.ReadLine();
            if (vertex != "END") this.AddVertex(vertex);
            else
            {
                Console.WriteLine(' ');
                break;
            }
        } while (true);
    }

    public void InputAllEdge()
    {
        Console.WriteLine("Вводите по две ранее добавленные вершины через
пробел, разделяя пары \\\"Enter\\\",\\\"между которыми находятся ребра (для окончания ввода
введите \\\"END\\\":");

        string input;
        do
        {
            input = Console.ReadLine();
            if (input == "END")
            {
                Console.WriteLine("Ввод завершён.\\n");
                break;
            }

            try
            {
                // Разбиваем ввод на части
                var parts = input.Split(' ');
                if (parts.Length != 2)
                {
                    throw new FormatException("Нужно ввести ровно две
вершины через пробел.");
                }

                string vertex1 = parts[0];
                string vertex2 = parts[1];

                // Добавляем ребро

```

```

        this.AddEdge(vertex1, vertex2);
    }
    catch (FormatException ex)
    {
        Console.WriteLine($"Ошибка: {ex.Message}");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Неожиданная ошибка: {ex.Message}");
    }
} while (true);
}

public void PrintAdjacencyList()
{
    foreach (var vertex in adjacencyList)
    {
        Console.WriteLine($"{vertex.Key}: {string.Join(", ",
vertex.Value)}");
        Console.WriteLine();
    }
}

private void FindMatchings(List<Tuple<string, string>> currentMatching,
List<List<Tuple<string, string>>> allMatchings, HashSet<string> usedVertices,
List<Tuple<string, string>> allEdges)
{
    // Если текущее паросочетание не пусто, добавляем его в список
    if (currentMatching.Count > 0)
    {
        // Чтобы избежать дублирования, проверяем, если такое
        паросочетание уже существует
        var sortedMatching = currentMatching.OrderBy(m =>
m.Item1).ThenBy(m => m.Item2).ToList();
        if (!allMatchings.Any(matching =>
matching.SequenceEqual(sortedMatching)))
        {
            allMatchings.Add(sortedMatching);
        }
    }

    // Перебираем все рёбра
    foreach (var edge in allEdges)
    {
        var vertex1 = edge.Item1;
        var vertex2 = edge.Item2;

        // Проверяем, были ли уже использованы эти вершины
        if (!usedVertices.Contains(vertex1) &&
!usedVertices.Contains(vertex2))
        {
            // Добавляем текущее ребро в паросочетание
            currentMatching.Add(edge);
            usedVertices.Add(vertex1);
            usedVertices.Add(vertex2);

            // Рекурсивно ищем другие паросочетания
            FindMatchings(currentMatching, allMatchings, usedVertices,
allEdges);

            // Возвращаемся в предыдущее состояние
            currentMatching.RemoveAt(currentMatching.Count - 1);
            usedVertices.Remove(vertex1);
            usedVertices.Remove(vertex2);
        }
    }
}

```

```

    }
    }
}

public List<List<Tuple<string, string>>> FindAllMatchings()
{
    List<List<Tuple<string, string>>> allMatchings = new
List<List<Tuple<string, string>>>();
    List<Tuple<string, string>> allEdges = new List<Tuple<string,
string>>();

    // Собираем все рёбра
    foreach (var vertex in adjacencyList.Keys)
    {
        foreach (var neighbor in adjacencyList[vertex])
        {
            if (vertex.CompareTo(neighbor) < 0) // Чтобы избежать
симметричных рёбер
            {
                allEdges.Add(Tuple.Create(vertex, neighbor));
            }
        }
    }

    FindMatchings(new List<Tuple<string, string>>(), allMatchings, new
HashSet<string>(), allEdges);

    return allMatchings;
}

// Метод для вывода всех паросочетаний
public void PrintAllMatchings(List<List<Tuple<string, string>>>
allMatchings)
{
    if (allMatchings.Count == 0)
    {
        Console.WriteLine("Паросочетаний в данном случае нет, введены
неверные данные");
        return;
    }

    Console.WriteLine("Найденные паросочетания:");
    foreach (var matching in allMatchings)
    {
        Console.WriteLine(string.Join(", ", matching.Select(m =>
$"({m.Item1}, {m.Item2})"))));
    }
}

// Метод для нахождения и вывода всех паросочетаний
public void FindAndPrintAllMatchings()
{
    List<List<Tuple<string, string>>> allMatchings = FindAllMatchings();
    // Находим все паросочетания
    PrintAllMatchings(allMatchings); // Выводим все паросочетания
}

// Функция для сортировки паросочетаний по количеству пар
public List<List<Tuple<string, string>>>
SortMatchingsBySizeAndInside(List<List<Tuple<string, string>>> allMatchings)
{
    // Сортируем паросочетания по количеству пар
    var sortedMatchings = allMatchings.OrderBy(matching =>
matching.Count).ToList();
}

```

```

        // Для каждого паросочетания сортируем пары внутри по возрастанию
        foreach (var matching in sortedMatchings)
        {
            matching.Sort((m1, m2) =>
            {
                int compareFirst = m1.Item1.CompareTo(m2.Item1);
                if (compareFirst != 0)
                    return compareFirst;
                return m1.Item2.CompareTo(m2.Item2);
            });
        }

        return sortedMatchings;
    }

    // Функция для вывода отсортированных паросочетаний по количеству пар и
    сортировке внутри
    public void PrintSortedMatchingsBySizeAndInside()
    {
        List<List<Tuple<string, string>>> allMatchings = FindAllMatchings();
        // Находим все паросочетания

        // Сортируем паросочетания по количеству пар и внутри каждого
        паросочетания
        List<List<Tuple<string, string>>> sortedMatchings =
        SortMatchingsBySizeAndInside(allMatchings);

        // Выводим отсортированные паросочетания
        PrintAllMatchings(sortedMatchings);
    }

    //Метод для вывода одного из больших паросочетаний
    public void PrintMaxMatching()
    {
        List<List<Tuple<string, string>>> allMatchings = FindAllMatchings();
        // Находим все паросочетания

        // Сортируем паросочетания по количеству пар и внутри каждого
        паросочетания
        List<List<Tuple<string, string>>> sortedMatchings =
        SortMatchingsBySizeAndInside(allMatchings);
        // Проверим, есть ли хотя бы одно паросочетание
        if (sortedMatchings.Count > 0)
        {
            // Выводим только последнюю строку (последнее паросочетание)
            var lastMatching = sortedMatchings.Last();
            Console.WriteLine($"Одно из наибольших паросочетаний:
{lastMatching.Count} пар");
            Console.WriteLine(string.Join(", ", lastMatching.Select(m =>
                $"({m.Item1}, {m.Item2})")));
        }
        else
        {
            Console.WriteLine("Паросочетаний не найдено.");
        }
    }

    public List<List<Tuple<string, string>>>
    SortMatchingsBySizeDescending(List<List<Tuple<string, string>>> allMatchings)
    {
        // Сортируем паросочетания по количеству пар по возрастанию
        var sortedMatchings = SortMatchingsBySizeAndInside(allMatchings);

        // Инвертируем порядок для сортировки по убыванию

```

```

        sortedMatchings.Reverse();

        return sortedMatchings;
    }

    public void PrintSortedMatchingsBySizeDescending()
    {
        List<List<Tuple<string, string>>> allMatchings = FindAllMatchings();
// Находим все паросочетания

        // Сортируем паросочетания по убыванию
        List<List<Tuple<string, string>>> sortedMatchings =
SortMatchingsBySizeDescending(allMatchings);

        // Выводим отсортированные паросочетания
        PrintAllMatchings(sortedMatchings);
    }

//Метод для нахождения числа совершенных паросочетаний
    public int IsPerfectMatching()
    {
        // Находим все паросочетания
        List<List<Tuple<string, string>>> allMatchings = FindAllMatchings();

        // Если нет паросочетаний, возвращаем 0
        if (allMatchings.Count == 0)
        {
            return 0;
        }

        // Находим максимальную длину паросочетания
        int maxLength = allMatchings.Max(matching => matching.Count);

        if (maxLength * 2 != adjacencyList.Count)
        {
            Console.WriteLine("Совершенные паросочетания могут быть только у
графов с четным количеством вершин");
            return 0;
        }

        // Подсчитываем количество паросочетаний с максимальной длиной
        int countMaxLengthMatchings = allMatchings.Count(matching =>
matching.Count == maxLength);

        return countMaxLengthMatchings;
    }
}
}

```

### 3. ТРЕТИЙ РАЗДЕЛ

#### 3.1. Формулировка задания

Вариант 4

Моделирование работы морского порта.

Требуется создать компьютерную модель обслуживания потока заявок на разгрузку, поступающих от грузовых судов (сухогрузов и танкеров), прибывающих в морской порт. Грузовые суда прибывают в порт согласно расписанию, но возможны опоздания и досрочные прибытия. Расписание включает день и время прибытия, название судна, вид груза и его вес, а также планируемый срок стоянки в порту для разгрузки.

Для разгрузки судов в порту используются три вида разгрузочных кранов, соответствующих трем видам грузов: сыпучим и жидким грузам, контейнерам. Число разгрузочных кранов каждого вида ограничено, так что поступающие заявки на разгрузку одного вида груза образуют очередь. Длительность разгрузки судна зависит от вида и веса его груза, а также некоторых других факторов, например, погодных условий. Любой дополнительный (сверх запланированного срока) день стояния судна в порту (из-за ожидания разгрузки в очереди или из-за задержки самой разгрузки) влечет за собой выплату штрафа (например, 2 тыс. условных единиц за каждый дополнительный день простоя судна).

При моделировании прибытия судов отклонение их от расписания рассматривается как случайная величина с равномерным распределением в некотором интервале (например, от -2 до 9 дней). Еще одной случайной величиной, изменяющейся в фиксированном диапазоне (например, от 0 до 12 дней), является время задержки окончания разгрузки судна по сравнению с обычным (зависящим только от вида груза и его веса).

Цель моделирования работы морского порта – определение для заданного расписания прибытия судов минимально достаточного числа кранов в порту, позволяющего уменьшить штрафные суммы. Период моделирования – месяц,

шаг моделирования – 1 день. В параметры моделирования следует включить расписание прибытия судов, количество кранов каждого вида, диапазоны разброса случайных величин (отклонения от расписания прибытия и отклонения от обычного времени разгрузки), а также шаг моделирования.

Визуализация моделируемого процесса должна предусматривать показ очередей у разгрузочных кранов, приход судов в порт и их отход после разгрузки. Должен быть показан также список произведенных разгрузок, в котором указывается название разгруженного судна, время его прихода в порт и время ожидания в очереди на разгрузку, время начала разгрузки и ее продолжительность.

По окончании моделирования должна быть выведена итоговая статистика: число разгруженных судов, средняя длина очереди на разгрузку, среднее время ожидания в очереди, максимальная и средняя задержка разгрузки, общая сумма выплаченного штрафа. Результат сбора статистики должен быть выведен в текстовый файл.

### **3.2. Теоретический аспект задачи**

Для решения задачи важно изучить ключевые аспекты моделирования обслуживания потоков заявок, а также особенности работы морского порта. Сначала требуется понять, как формируются и обслуживаются потоки заявок. В данном случае поток заявок представлен судами, прибывающими в порт для разгрузки. Каждое судно характеризуется временем прибытия, типом и весом груза, а также планируемым временем стоянки. Для организации обслуживания заявок используются модели теории массового обслуживания, где заявки распределяются между ограниченным числом обслуживающих устройств, в данном случае – разгрузочных кранов.

Работа морского порта предполагает разграничение типов грузов: сыпучие, жидкие и контейнеры, для каждого из которых выделены специализированные краны. Ограниченное количество кранов порождает очереди, если количество заявок на разгрузку превышает их пропускную



способность. Время разгрузки зависит от типа и веса груза, а также от внешних факторов, таких как погодные условия. Эти факторы могут вызывать задержки, которые учитываются в модели как случайные величины.

Моделирование отклонений прибытия судов от расписания также осуществляется через случайные величины с заданным распределением. Эти отклонения и задержки разгрузки влияют на эффективность работы порта. Для оценки этой эффективности необходимо проанализировать длину очередей, время ожидания судов и штрафы за дополнительные дни стоянки, которые возникают из-за задержек.

Цель моделирования состоит в определении минимального числа кранов, необходимого для сокращения штрафов и оптимизации работы порта. Для этого требуется использовать заданное расписание прибытия судов, учитывать ограничения на число кранов, диапазоны случайных отклонений и шаг моделирования. Визуализация процессов должна включать движение судов, формирование очередей и их разгрузку, а итоговая статистика должна содержать данные о количестве разгруженных судов, средней длине очереди, времени ожидания, задержках и сумме штрафов. Эта информация фиксируется в текстовых отчетах, что позволяет проводить детальный анализ эффективности работы порта.

### 3.3. Формализация задачи

#### Описание классов.

Далее представлена диаграмма классов (Рисунок 7.1).

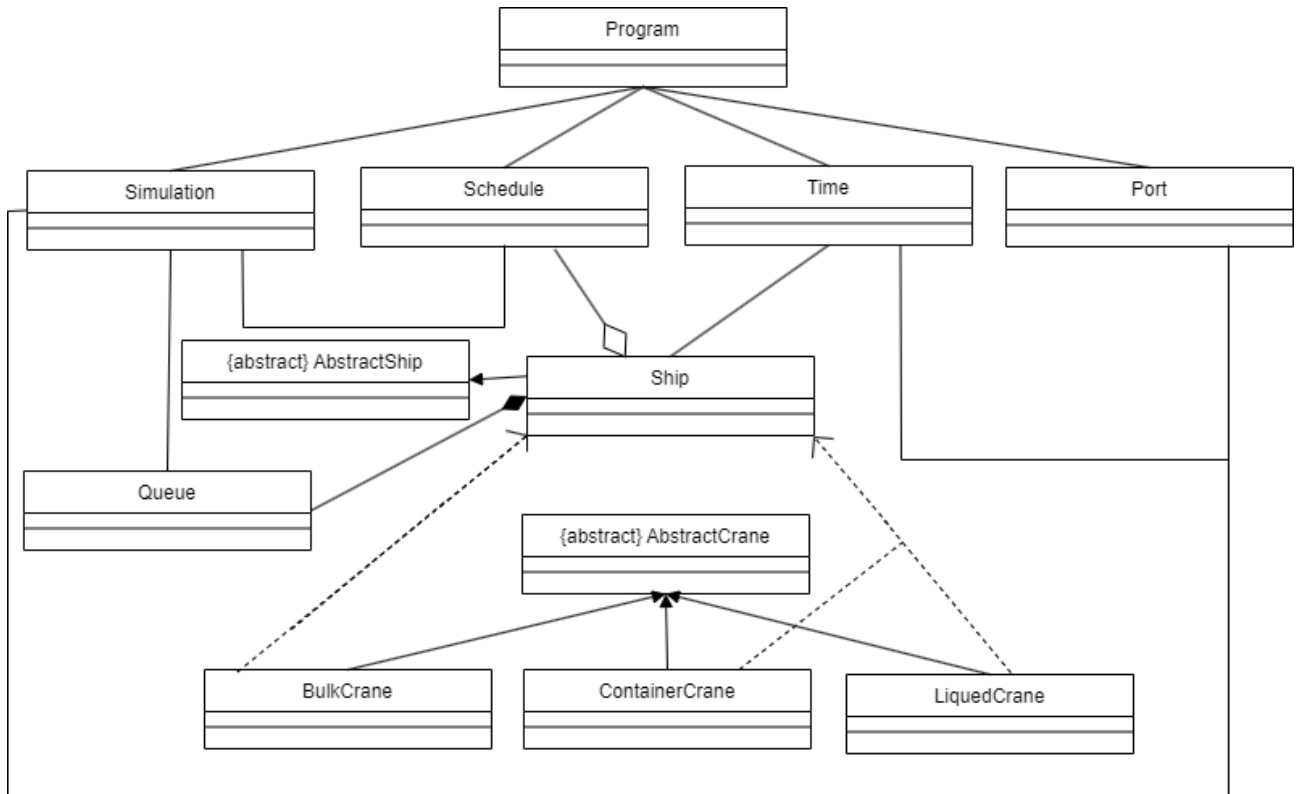


Рисунок 7.1 – Диаграмма классов для третьего раздела

На рисунках 7.2-7.12 представлены диаграммы отдельных классов.

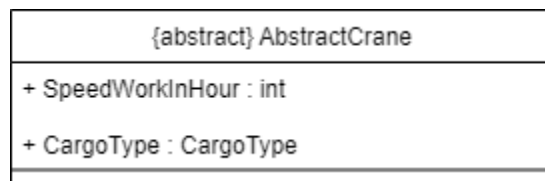


Рисунок 7.2 – Диаграмма абстрактного класса AbstractCrane

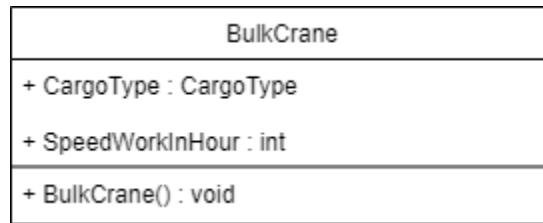


Рисунок 7.3 – Диаграмма класса BulkCrane

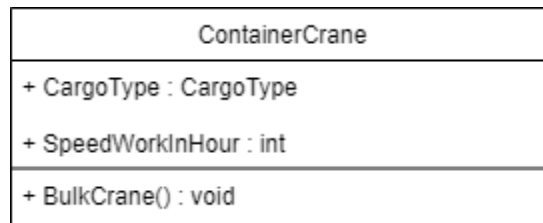


Рисунок 7.4 – Диаграмма класса ContainerCrane

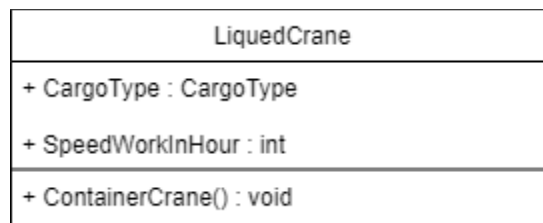


Рисунок 7.5 – Диаграмма класса LiquedCrane

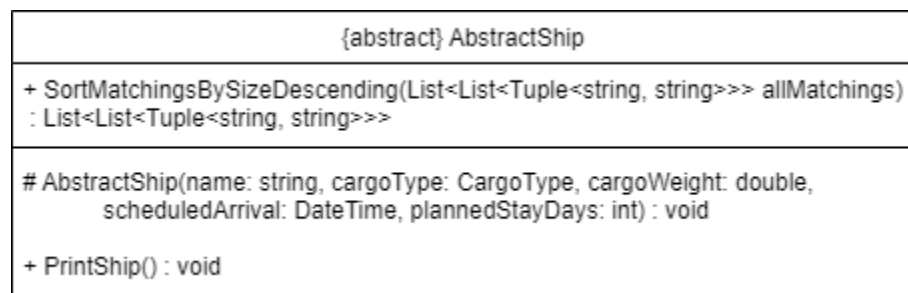


Рисунок 7.6 – Диаграмма абстрактного класса AbstractShip

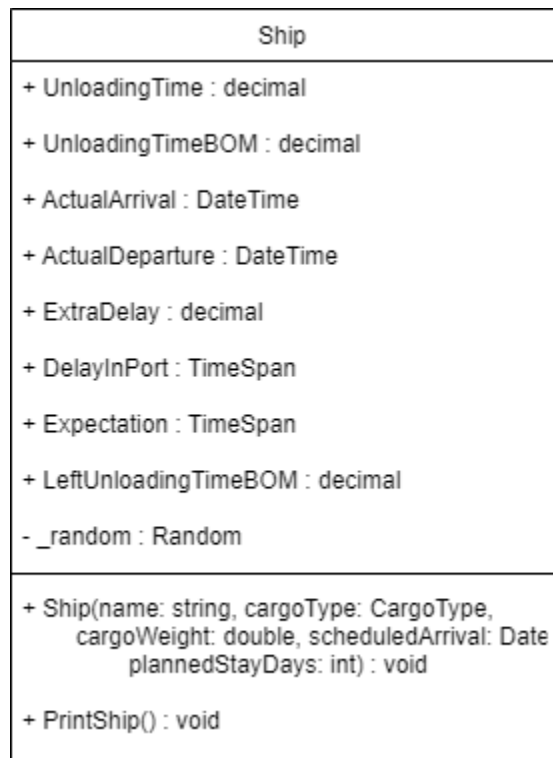


Рисунок 7.7 – Диаграмма класса Ship

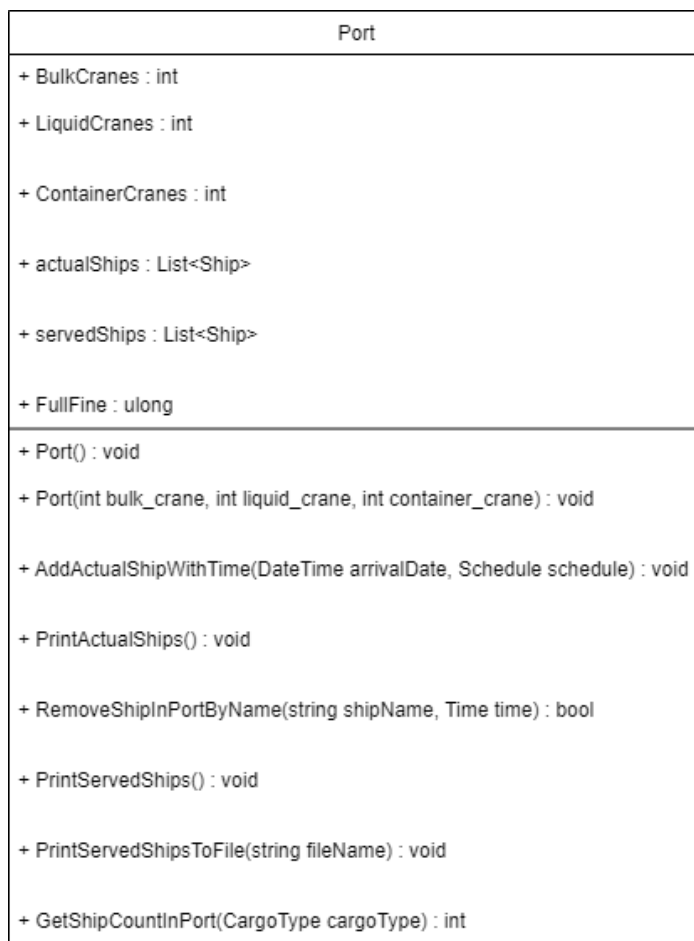


Рисунок 7.8 – Диаграмма класса Port

Queue
+ queue_bulk : List<Ship> + queue_liquid : List<Ship> + queue_container : List<Ship>
+ Queue() : void + AddQueue(Ship ship, Time time) : void + RemoveQueue(Ship ship) : void + IsShipInBulkQueue(Ship ship) : bool + IsShipInLiquidQueue(Ship ship) : bool + IsShipInContainerQueue(Ship ship) : bool + IsShipInQueue(Ship ship) : bool + PrintAllShipsInQueue() : void + ProcessQueue(List<Ship> queue, Port port, Time time, CargoType cargoType) : void + IsAnyShipInQueue() : bool + GetShipCountInQueueReflection(CargoType cargoType) : ulong

Рисунок 7.9 – Диаграмма класса Queue

Schedule
- scheduleShip : List<Ship>
+ GenerateScheduleShip(int numberOfShips, DateTime startDate) : void + SortSchedule(Func<Ship, object> keySelector) : List<Ship> + SortByScheduledArrival() : List<Ship> + SortByActualArrival() : List<Ship> + PrintSchedule(List<Ship> ships) : void + PrintScheduleToFile(string fileName, List<Ship> ships) : void + PrintScheduleReally(List<Ship> ships) : void + PrintFullSchedule() : void

Рисунок 7.10 – Диаграмма класса Schedule

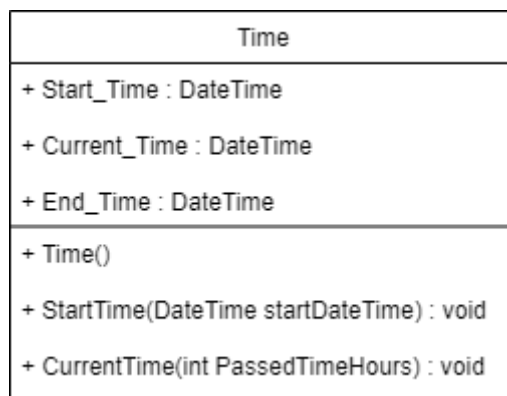


Рисунок 7.11 – Диаграмма класса Time

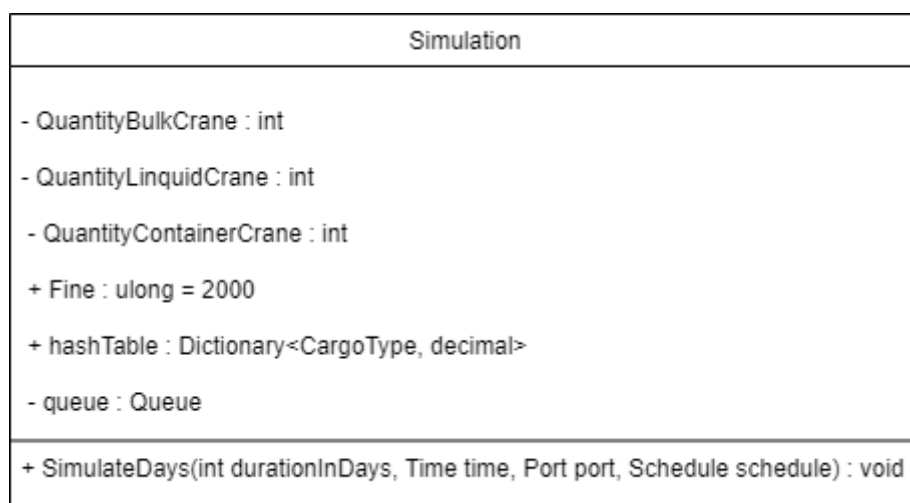


Рисунок 7.12 – Диаграмма класса Simulation

Система симуляции порта представляет собой композицию базовых классов. Класс *AbstractShip* является фундаментальным и описывает модель корабля. Полями этого класса являются ключевые свойства корабля, такие как название, тип груза, вес груза, запланированная дата прибытия и срок стоянки. *AbstractShip* выступает частью композиции класса *Schedule*, который моделирует расписание судов. Полями класса *Schedule* является список кораблей, представляющий собой расписание их прибытия, и методы для сортировки и вывода этого расписания.

Класс *AbstractCrane* является базовым для описания модели разгрузочных кранов. Он определяет свойства, такие как скорость работы крана и тип

обрабатываемого груза. От него наследуются классы BulkCrane, LiquidCrane и ContainerCrane, которые описывают краны для обработки сыпучих грузов, жидкостей и контейнеров соответственно.

Классы Queue и Port являются частями композиции класса Simulation, который представляет собой основную логику симуляции. Класс Queue управляет очередями судов в зависимости от типа груза и наличия свободных кранов. Полями Queue являются списки очередей для каждого типа груза и методы для добавления, удаления судов и обработки времени ожидания.

Класс Port моделирует порт, где происходит разгрузка судов. Его полями являются количество кранов каждого типа, список текущих судов в порту, список разгруженных судов и накопленная сумма штрафов. Он также содержит методы для управления судами в порту и сбора статистики.

Класс Simulation является центральным элементом системы. Он отвечает за моделирование процесса разгрузки, управление портом, очередями и обработку отклонений в расписании. Полями Simulation являются параметры симуляции, такие как продолжительность моделируемого периода, таблица средней очереди для каждого типа груза, и методы для выполнения всех операций.

Эта архитектура обеспечивает модульность и гибкость системы, позволяя легко адаптировать или расширять функциональность для новых требований.

### **Описание алгоритмов.**

В процессе моделирования работы порта ключевые этапы обработки ситуаций реализованы в классе Simulation, который управляет основным алгоритмом симуляции. Начальная подготовка включает генерацию расписания судов с учетом случайных отклонений от планируемых дат прибытия и расчета времени разгрузки каждого судна в зависимости от типа груза и веса. Также определяется начальная конфигурация портовой инфраструктуры: количество кранов каждого типа и стартовые временные параметры.

После подготовки запускается основной цикл симуляции, который разбит на временные шаги (дни). На каждом шаге проверяется текущее время, добавляются корабли, прибывшие в порт в данный день, и обновляются очереди на разгрузку в зависимости от доступности кранов. В случае наличия ожидающих в очереди судов и свободных кранов начинается разгрузка. Время разгрузки рассчитывается индивидуально для каждого судна с учетом его характеристик и текущих условий.

Алгоритм разгрузки включает проверку возможности обработки судна в текущий день. Если оставшееся время работы крана превышает время, необходимое для полной разгрузки, судно считается разгруженным, и его данные переносятся в список обслуженных судов. Если времени недостаточно, незавершенная разгрузка переносится на следующий день. Дополнительно для каждого судна рассчитывается время ожидания в очереди, а также задержка относительно планируемой даты отбытия, что влияет на итоговую сумму штрафов.

Разгрузка судов организована через методы класса Queue, которые управляют добавлением судов в очередь и их распределением по типам грузов. Для каждого типа груза используется отдельная очередь, и судно может быть добавлено в очередь только если его тип груза соответствует свободному крану.

Методы класса Port контролируют состояние порта: добавление судов в активный список, перемещение разгруженных судов в архив, и расчет штрафов за простои. В конце симуляции с помощью класса Schedule создаются отчетные файлы, содержащие данные о реальных и планируемых расписаниях, обслуженных судах и итоговой статистике.

Таким образом, алгоритмы симуляции организованы с учетом модульного подхода, где каждый компонент системы отвечает за свою часть обработки, что позволяет эффективно моделировать работу порта и оптимизировать его инфраструктуру.



### 3.4. Спецификация классов

Описание полей класса AbstractShip представлено в таблице 11.

**Таблица 11**

Имя	Тип	Модификатор доступа	Назначение
Name	string	public	Название судна
CargoType	CargoType	public	Тип груза
CargoWeight	double	public	Вес груза
ScheduledArrival	DateTime	public	Планируемая дата прибытия
PlannedStayDays	int	public	Планируемое количество дней стоянки
DepartureDate	DateTime	public	Планируемая дата отбытия, рассчитанная на основе времени прибытия и срока стоянки

Описание методов класса AbstractShip представлено в таблице 12.

**Таблица 12**

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
AbstractShip	-	protected	string name, CargoType cargoType, double cargoWeight, DateTime scheduledArrival, int plannedStayDays	-	Конструктор базового класса для инициализации судна
PrintShip	void	public	-	-	Вывод информации о судне

Описание полей класса Port представлено в таблице 13.

**Таблица 13**

Имя	Тип	Модификатор доступа	Назначение
BulkCranes	int	public	Количество кранов для сыпучих грузов
LiquidCranes	int	public	Количество кранов для жидких грузов
ContainerCranes	int	public	Количество кранов для контейнеров
actualShips	List<Ship>	public	Список судов, находящихся в порту

servedShips	List<Ship>	public	Список судов, которые успешно разгружены
FullFine	ulong	public	Общая сумма штрафов за задержки судов

Описание методов класса Port представлено в таблице 14.

**Таблица 14**

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
Port	-	public	int bulk_crane, int liquid_crane, int container_crane	-	Конструктор класса для инициализации портовой инфраструктуры
AddActualShipWithTime	void	public	DateTime arrivalDate, Schedule schedule	-	Добавление судна в порт на основе времени фактического прибытия
PrintActualShips	void	public	-	-	Вывод текущих судов, находящихся в порту
PrintServedShips	void	public	-	-	Вывод списка разгруженных судов
RemoveShipInPortByName	bool	public	string shipName, Time time	bool	Удаление судна из порта по имени, возвращает успех удаления
PrintServedShipsToFile	void	public	string fileName	-	Запись списка обслуженных судов в файл
GetShipCountInPort	int	public	CargoType cargoType	int	Получение количества судов в порту для

					заданного типа груза
--	--	--	--	--	-------------------------

Описание полей класса Queue представлено в таблице 15.

**Таблица 15**

Имя	Тип	Модификатор доступа	Назначение
queue_bulk	List<Ship>	public	Очередь судов с сыпучими грузами
queue_liquid	List<Ship>	public	Очередь судов с жидкими грузами
queue_container	List<Ship>	public	Очередь судов с контейнерами

Описание методов класса Queue представлено в таблице 16.

**Таблица 16**

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
AddQueue	void	public	Ship ship, Time time	-	Добавление судна в очередь соответствующего типа
RemoveQueue	void	public	Ship ship	-	Удаление судна из очереди
PrintAllShipsInQueue	void	public	-	-	Вывод информации обо всех очередях
ProcessQueue	void	public	List<Ship> queue, Port port, Time time, CargoType cargoType	-	Обработка очереди с учетом времени работы и разгрузки
IsShipInBulkQueue	bool	public	Ship ship	bool	Проверяет, находится ли корабль в очереди с сыпучими грузами
IsShipInLiquidQueue	bool	public	Ship ship	bool	Проверяет, находится ли корабль в очереди с жидкими грузами
IsShipInContainerQueue	bool	public	Ship ship	bool	Проверяет, находится ли корабль

					в очереди с контейнерами
IsShipInQueue	bool	public	Ship ship	bool	Проверяет, находится ли корабль в какой-либо очереди
IsAnyShipInQueue	bool	public	-	bool	Проверяет, есть ли хотя бы одно судно в одной из очередей
GetShipCountInQueueReflection	ulong	public	CargoType cargoType	ulong	Получение количества судов в очереди для заданного типа груза

Описание полей класса Simulation представлено в таблице 17.

**Таблица 17**

Имя	Тип	Модификатор доступа	Назначение
QuantityBulkCrane	int	private	Количество кранов для разгрузки судов с сыпучими грузами
QuantityLiquidCrane	int	private	Количество кранов для разгрузки судов с жидкими грузами
QuantityContainerCrane	int	private	Количество кранов для разгрузки судов с контейнерами
Fine	ulong	public	Штраф за задержку судна в часах
hashTable	Dictionary<CargoType, decimal>	public	Хранит среднее количество судов каждого типа в порту в течение симуляции
queue	Queue	private	Содержит очереди судов

			для каждого типа груза
--	--	--	---------------------------

Описание методов класса Simulation представлено в таблице 18.

**Таблица 18**

Метод	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
SimulateDays	void	public	int durationInDays, Time time, Port port, Schedule schedule	-	Моделирование работы порта в течение указанного периода времени

Описание полей класса Ship представлено в таблице 19.

**Таблица 19**

Имя	Тип	Модификатор доступа	Назначение
UnloadingTime	decimal	public (только чтение)	Время разгрузки судна без дополнительных задержек
UnloadingTimeBOM	decimal	public	Время разгрузки судна с учетом возможных происшествий
ActualArrival	DateTime	public (только чтение)	Реальное время прибытия судна
ActualDeparture	DateTime	private	Реальное время отъезда судна
ExtraDelay	decimal	private	Дополнительная задержка, случайным образом сгенерированная
DelayInPort	TimeSpan	public	Задержка судна в порту
Expectation	TimeSpan	public	Время ожидания до начала разгрузки
LeftUnloadingTimeBOM	decimal	public	Остаток времени на разгрузку судна с учетом происшествий

Описание методов класса Ship представлено в таблице 20.

**Таблица 20**

Имя	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
Ship	Конструктор	public	string name, CargoType cargoType, double cargoWeight, DateTime scheduledArrival , int plannedStayDays	-	Инициализация судна, расчет времени разгрузки и вероятных задержек
PrintShip	void	public	-	-	Вывод информации о судне, включая время прибытия, выгрузки и дополнительные задержки

Описание полей класса AbstractCrane представлено в таблице 21.

**Таблица 21**

Имя	Тип	Модификатор доступа	Назначение
SpeedWorkInHour	int	protected	Скорость работы крана (грузов в час)
CargoType	CargoType	protected	Тип груза, который может обрабатывать кран

Описание полей класса BulkCrane представлено в таблице 22.

**Таблица 22**

Имя	Тип	Модификатор доступа	Назначение
SpeedWorkInHour	int	protected	Скорость работы крана (грузов в час)
CargoType	CargoType	protected	Тип груза, который может обрабатывать кран

Описание методов класса BulkCrane представлено в таблице 23.

**Таблица 23**

Имя	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
BulkCrane (конструктор)	-	public	-	-	Инициализация крана для обработки сыпучих

					грузов. Устанавливает тип груза (Bulk) и скорость работы (1500 кг/час).
--	--	--	--	--	---

Описание полей класса ContainerCrane представлено в таблице 24.

**Таблица 24**

Имя	Тип	Модификатор доступа	Назначение
SpeedWorkInHour	int	protected	Скорость работы крана (контейнеров в час)
CargoType	CargoType	protected	Тип груза, который может обрабатывать кран

Описание методов класса ContainerCrane представлено в таблице 25.

**Таблица 25**

Имя	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
ContainerCrane (конструктор)	-	public	-	-	Инициализация крана для обработки контейнеров. Устанавливает тип груза (Container) и скорость работы (100 контейнеров/час)

Описание полей класса LiquefiedCrane представлено в таблице 26.

**Таблица 26**

Имя	Тип	Модификатор доступа	Назначение
SpeedWorkInHour	int	protected	Скорость работы крана (грузов в час)
CargoType	CargoType	protected	Тип груза, который может обрабатывать кран

Описание методов класса LiquefiedCrane представлено в таблице 27.

**Таблица**

Имя	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
LiquedCrane (конструктор)	-	public	-	-	Инициализация крана для обработки жидких грузов. Устанавливает тип груза (Liquid) и скорость работы (3000 кг/час).

Описание полей класса Schedule представлено в таблице 28.

**Таблица 28**

Имя	Тип	Модификатор доступа	Назначение
scheduleShip	List<Ship>	private	Список судов, запланированных на месяц

Описание методов класса Schedule представлено в таблице 29.

**Таблица 29**

Имя	Возвращаемый тип	Модификатор доступа	Входные параметры	Выходные параметры	Назначение
GenerateScheduleShip	void	public	int numberOfShips, DateTime startDate	-	Генерация случайного расписания судов на месяц.
SortSchedule	List<Ship>	public	Func<Ship, object> keySelector	List<Ship>	Общая функция сортировки по произвольному ключу.
SortByScheduledArrival	List<Ship>	public	-	List<Ship>	Сортировка расписания по запланированному времени прибытия.
SortByActualArrival	List<Ship>	public	-	List<Ship>	Сортировка расписания по реальному времени прибытия судов.
PrintSchedule	void	public	List<Ship> ships	-	Вывод списка судов с их



					детальями в консоль.
PrintScheduleToFile	void	public	string fileName, List<Ship> ships	-	Запись расписания судов в файл.
PrintScheduleReally	void	public	List<Ship> ships	-	Альтернативный вывод расписания судов с дополнительной информацией о прибытии и отъезде.
PrintFullSchedule	void	public	-	-	Вывод полного расписания судов.

Описание полей класса Time представлено в таблице 30.

**Таблица 30**

Имя	Тип	Модификатор доступа	Назначение
Start_Time	DateTime	public	Начальное время симуляции.
Current_Time	DateTime	public	Текущее время симуляции.
End_Time	DateTime	public	Конечное время симуляции.

Описание методов класса Time представлено в таблице 31.

**Таблица 31**

Имя	Возвращаемый тип	Модификатор доступа	Входные параметры
Time()	void	public	-
StartTime()	void	public	DateTime startDateTime
CurrentTime()	void	public	int PassedTimeHours

### 3.5. Руководство оператора

При запуске симуляции порт отображает консольный интерфейс, предоставляющий пользователю информацию о текущем состоянии порта, разгружаемые корабли, прибывшие и убывшие корабли, а также статистику после завершения моделирования. Взаимодействие с системой осуществляется автоматически, но пользователь может настроить исходные параметры, такие

как количество кранов, число генерируемых судов и диапазоны отклонений времени прибытия и разгрузки.

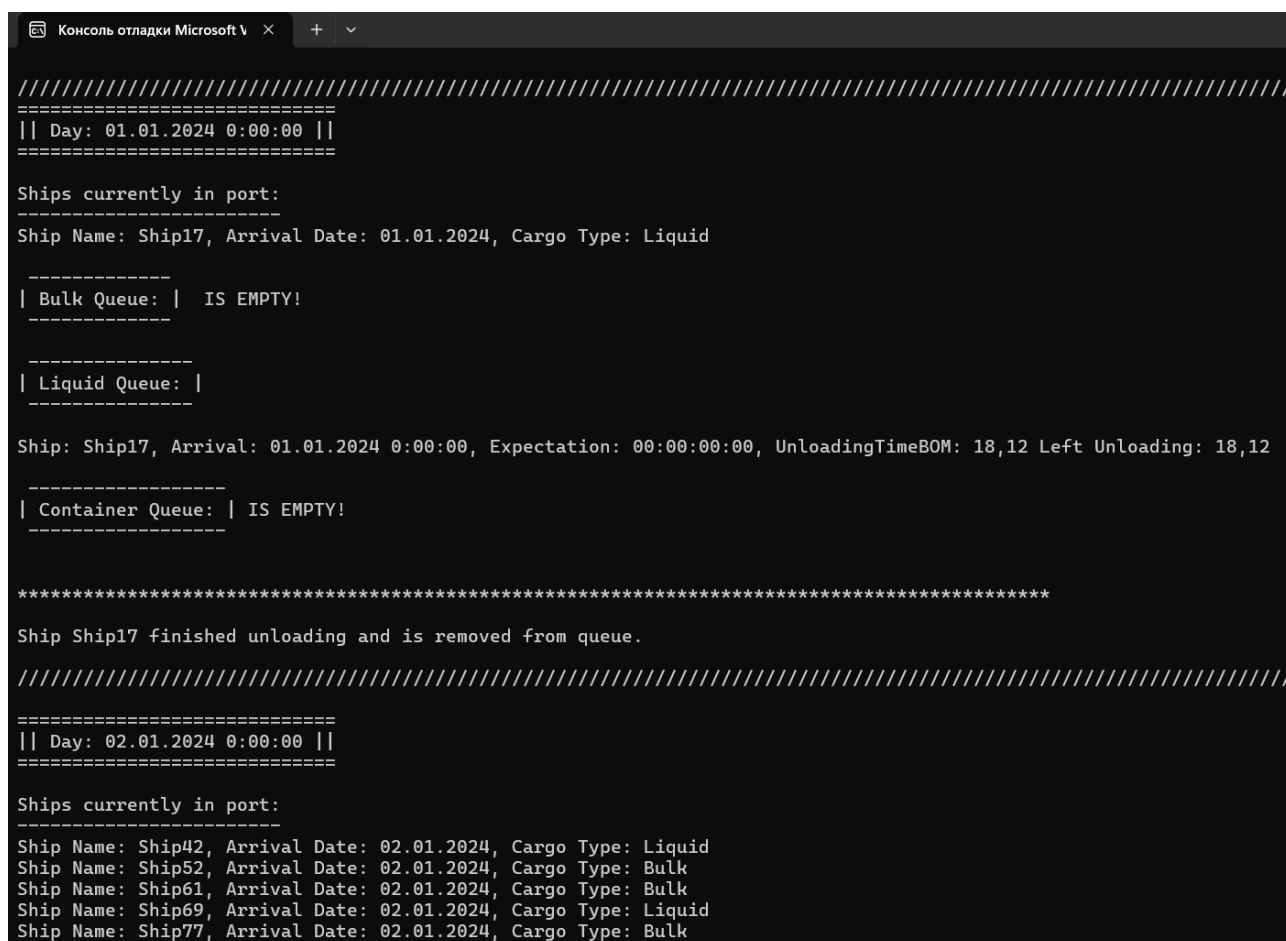
Основные этапы работы интерфейса:

### 1. Запуск симуляции

Пользователь запускает программу, и симуляция автоматически иницирует процесс моделирования. На экране отображается начальная информация:

- Дата начала симуляции.
- Генерация расписания судов (в файл).

Пример интерфейса начального этапа представлен на Рисунке 8.1.



```
#####  
|| Day: 01.01.2024 0:00:00 ||  
#####  
  
Ships currently in port:  
-----  
Ship Name: Ship17, Arrival Date: 01.01.2024, Cargo Type: Liquid  
  
-----  
| Bulk Queue: | IS EMPTY!  
-----  
  
-----  
| Liquid Queue: |  
-----  
  
Ship: Ship17, Arrival: 01.01.2024 0:00:00, Expectation: 00:00:00:00, UnloadingTimeBOM: 18,12 Left Unloading: 18,12  
  
-----  
| Container Queue: | IS EMPTY!  
-----  
  
*****  
  
Ship Ship17 finished unloading and is removed from queue.  
  
#####  
  
#####  
|| Day: 02.01.2024 0:00:00 ||  
#####  
  
Ships currently in port:  
-----  
Ship Name: Ship42, Arrival Date: 02.01.2024, Cargo Type: Liquid  
Ship Name: Ship52, Arrival Date: 02.01.2024, Cargo Type: Bulk  
Ship Name: Ship61, Arrival Date: 02.01.2024, Cargo Type: Bulk  
Ship Name: Ship69, Arrival Date: 02.01.2024, Cargo Type: Liquid  
Ship Name: Ship77, Arrival Date: 02.01.2024, Cargo Type: Bulk
```

Рисунок 8.1 – Интерфейс запуска симуляции.

### 2. Дневные события симуляции

Во время симуляции система выводит информацию о текущем дне, включая:

- Список судов, прибывших в порт за день.
- Текущее состояние очередей на разгрузку (для каждого типа грузов).
- Количество разгруженных судов.

Если судно было успешно разгружено, это также отображается в консоли с указанием его характеристик и времени простоя. Пример интерфейса для ежедневного состояния порта показан на Рисунке 8.2.

```
*****
Ship Ship23 finished unloading and is removed from queue.
Adding new ship Ship45 to the queue.
Ship: Ship45, Arrival: 14.01.2024 0:00:00, Expectation: 15:20:20:43, UnloadingTimeBOM: 61,04
Ship Ship8 finished unloading and is removed from queue.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Рисунок 8.2 – Интерфейс состояния порта в конце дня.

### 3. Вывод итоговой статистики

После завершения симуляции (по окончании заданного периода) пользователю предоставляется сводка результатов:

- Общая сумма штрафов за задержки судов.
- Среднее время ожидания судов в очереди.
- Максимальная и средняя задержка разгрузки.
- Список разгруженных судов с указанием их характеристик.

Итоговые данные записываются в файлы, включая:

- PlannedSchedule.txt — расписание судов.
- ActualSchedule.txt — реальное расписание с учетом отклонений.
- ServedShips.txt — список обслуженных судов.
- Statistic.txt — сводная статистика.

### Поведение при ошибках или недопустимых действиях

Система реализует автоматическую проверку корректности обработки данных. Если судно невозможно разгрузить из-за отсутствия кранов, оно остается в очереди до следующего шага симуляции. Непредусмотренные ситуации, такие как отсутствие данных в расписании, сопровождаются сообщениями в консоли.

Пользователь взаимодействует с программой через конфигурацию параметров в коде перед запуском симуляции, что делает интерфейс понятным и доступным для оператора.

## **3.6. Руководство программиста**

Структура программы для симуляции работы порта организована в соответствии с принципами модульного программирования. Все логически связанные классы и структуры разделены по отдельным файлам, что упрощает навигацию, поддержку и расширение системы.

### Организация файлов

#### 1. Классы базовой логики:

- AbstractShip.cs – базовый класс для описания судов.
- AbstractCrane.cs – базовый класс для описания кранов.
- CargoType.cs – перечисление типов грузов.

#### 2. Классы портовой инфраструктуры:

- Port.cs – управление портом, текущими и разгруженными судами.
- Queue.cs – обработка очередей судов в зависимости от типа груза.

#### 3. Классы, связанные с симуляцией:

- Schedule.cs – генерация и управление расписанием судов.
- Simulation.cs – управление основным циклом симуляции.

- Time.cs – модель времени для учета текущей даты и длительности симуляции.

#### 4. Классы реализации кранов:

- BulkCrane.cs – краны для сыпучих грузов.
- LiquidCrane.cs – краны для жидких грузов.
- ContainerCrane.cs – краны для контейнеров.

#### Комментарии в коде

Код содержит подробные комментарии, поясняющие логику работы каждого метода и их назначение. Например:

- Комментарии к классам описывают их назначение и основные поля.
- Методы снабжены комментариями, описывающими входные параметры, возвращаемые значения и общую логику работы.
- В местах обработки сложных алгоритмов добавлены пояснения, уточняющие расчет времени разгрузки, ожидания и штрафов.

### 3.7. Пример работы программы

Далее представлены результаты работы программы на примере моделирования работы порта в течение одного месяца. Пример демонстрирует процесс выполнения симуляции: от генерации расписания судов до подсчета итоговой статистики.

#### Пояснение:

Для контрольного примера была запущена симуляция с параметрами:

- Количество сыпучих кранов: 1.
- Количество жидких кранов: 1.
- Количество контейнерных кранов: 1.
- Количество генерируемых судов: 100.
- Диапазон отклонений прибытия: от -2 до 4 дней.
- Диапазон задержек разгрузки: от 1 до 6 часов.

- Период симуляции: 31 день.

На рисунке показан процесс моделирования одного дня симуляции. В начале дня выводится дата, затем отображается список судов, прибывших в порт, разгружаемые корабли для каждого типа грузов, а также завершенные операции разгрузки.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
=====
|| Day: 01.01.2024 0:00:00 ||
=====

Ships currently in port:
-----
Ship Name: Ship1, Arrival Date: 01.01.2024, Cargo Type: Container
Ship Name: Ship31, Arrival Date: 01.01.2024, Cargo Type: Bulk

-----
| Bulk Queue: |
-----

Ship: Ship31, Arrival: 01.01.2024 0:00:00, Expectation: 00:00:00:00, UnloadingTimeBOM: 46,49 Left Unloading: 46,49

-----
| Liquid Queue: | IS EMPTY!
-----

| Container Queue: |
-----

Ship: Ship1, Arrival: 01.01.2024 0:00:00, Expectation: 00:00:00:00, UnloadingTimeBOM: 956,31, Left Unloading: 956,31
*****
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

Рисунок 9.1 – Запуск симуляции одного дня

На рисунке показано запланированное расписание на начало месяца. Оно включает в себя, название судна, тип груза, планируемую дату прибытия, количество дней в порту, планируемую дату выезда.

PlannedSchedule.txt

Файл Изменить Просмотр

```

+++++
+ Schedule of Ships: +
+++++

```

Name	CargoType	CargoWeight (kg)	Scheduled Arrival	Planned Stay (days)	Departure Date
Ship31	Bulk	62229	01-01-2024, -15	6	07-01-2024, -15
Ship90	Container	38150	01-01-2024, -15	11	12-01-2024, -15
Ship26	Container	28374	02-01-2024, -15	9	11-01-2024, -15
Ship27	Bulk	79752	02-01-2024, -15	8	10-01-2024, -15
Ship1	Container	95531	03-01-2024, -15	11	14-01-2024, -15
Ship15	Liquid	97281	03-01-2024, -15	7	10-01-2024, -15
Ship33	Bulk	26147	03-01-2024, -15	13	16-01-2024, -15
Ship41	Container	88886	03-01-2024, -15	13	16-01-2024, -15
Ship82	Liquid	56460	03-01-2024, -15	12	15-01-2024, -15

Рисунок 9.2 – Запланированное расписание

На рисунке изображено актуальное расписание, которое включает в себя, название судна, тип груза, вес груза, реальная дата прибытия, запланированное количество дней в порту, дата отбытия исходя из начального расписания.

ActualSchedule.txt

Файл Изменить Просмотр

```

+++++
+ Schedule of Ships: +
+++++

```

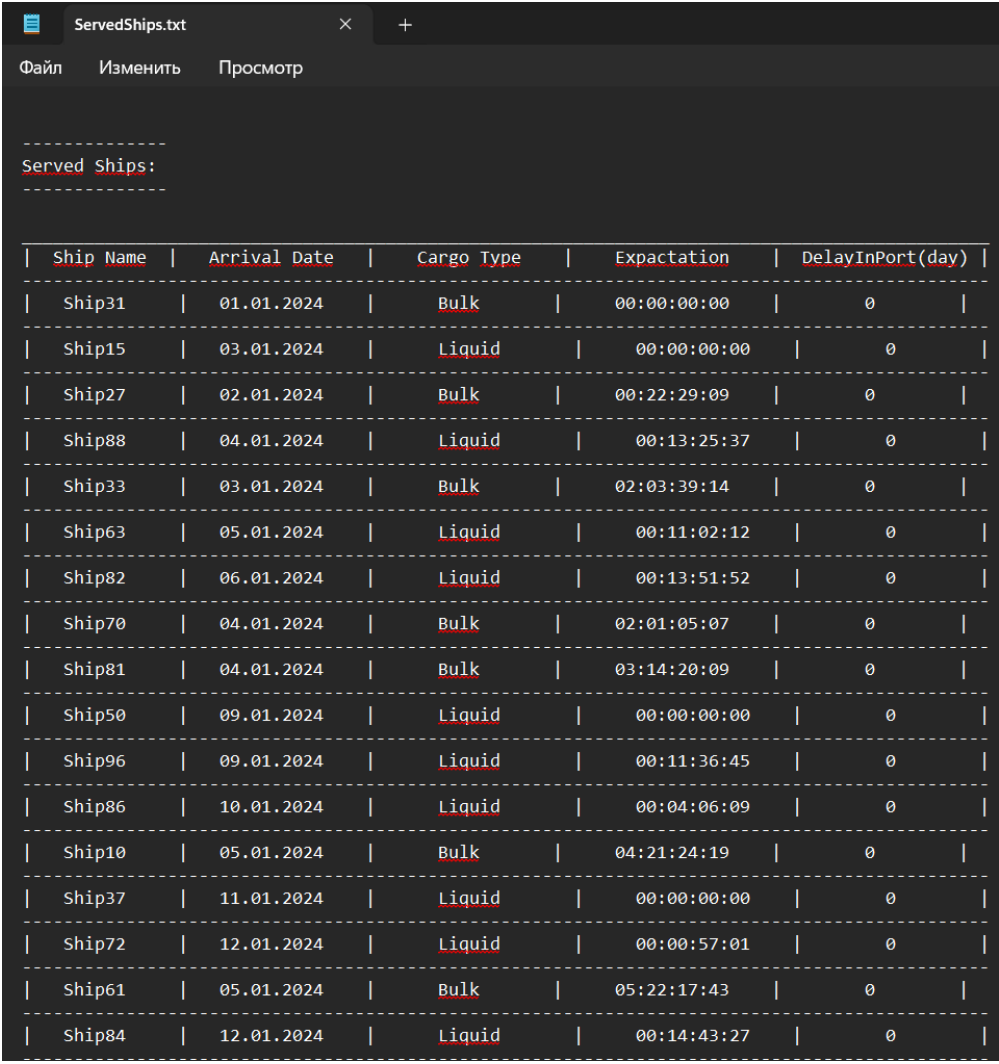
Name	CargoType	CargoWeight (kg)	Scheduled Arrival	Planned Stay (days)	Departure Date
Ship90	Container	38150	01-01-2024, -15	11	12-01-2024, -15
Ship1	Container	95531	03-01-2024, -15	11	14-01-2024, -15
Ship31	Bulk	62229	01-01-2024, -15	6	07-01-2024, -15
Ship26	Container	28374	02-01-2024, -15	9	11-01-2024, -15
Ship27	Bulk	79752	02-01-2024, -15	8	10-01-2024, -15
Ship15	Liquid	97281	03-01-2024, -15	7	10-01-2024, -15
Ship33	Bulk	26147	03-01-2024, -15	13	16-01-2024, -15
Ship41	Container	88886	03-01-2024, -15	13	16-01-2024, -15
Ship12	Container	76054	04-01-2024, -15	13	17-01-2024, -15
Ship17	Container	74847	04-01-2024, -15	7	11-01-2024, -15
Ship34	Container	80431	05-01-2024, -15	12	17-01-2024, -15
Ship70	Bulk	91876	06-01-2024, -15	9	15-01-2024, -15
Ship81	Bulk	43604	04-01-2024, -15	9	13-01-2024, -15
Ship88	Liquid	61829	04-01-2024, -15	12	16-01-2024, -15
Ship10	Bulk	73335	05-01-2024, -15	8	13-01-2024, -15

Рисунок 9.3 – Реальное расписание прибытия судов





Далее продемонстрирован итоговый список всех кораблей, которые были разгружены в течение месяца.



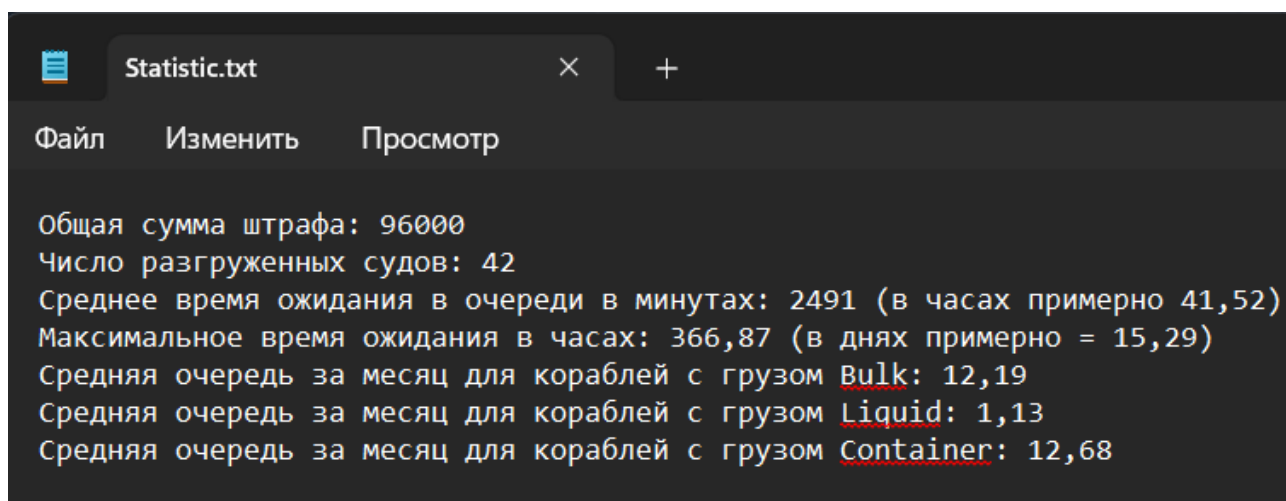
The screenshot shows a text editor window with the title 'ServedShips.txt'. The window has a menu bar with 'Файл', 'Изменить', and 'Просмотр'. The content of the file is a table of ship arrivals, preceded by a header line 'Served Ships:'. The table has five columns: 'Ship Name', 'Arrival Date', 'Cargo Type', 'Expectation', and 'DelayInPort(day)'. The data is as follows:

Ship Name	Arrival Date	Cargo Type	Expectation	DelayInPort(day)
Ship31	01.01.2024	Bulk	00:00:00:00	0
Ship15	03.01.2024	Liquid	00:00:00:00	0
Ship27	02.01.2024	Bulk	00:22:29:09	0
Ship88	04.01.2024	Liquid	00:13:25:37	0
Ship33	03.01.2024	Bulk	02:03:39:14	0
Ship63	05.01.2024	Liquid	00:11:02:12	0
Ship82	06.01.2024	Liquid	00:13:51:52	0
Ship70	04.01.2024	Bulk	02:01:05:07	0
Ship81	04.01.2024	Bulk	03:14:20:09	0
Ship50	09.01.2024	Liquid	00:00:00:00	0
Ship96	09.01.2024	Liquid	00:11:36:45	0
Ship86	10.01.2024	Liquid	00:04:06:09	0
Ship10	05.01.2024	Bulk	04:21:24:19	0
Ship37	11.01.2024	Liquid	00:00:00:00	0
Ship72	12.01.2024	Liquid	00:00:57:01	0
Ship61	05.01.2024	Bulk	05:22:17:43	0
Ship84	12.01.2024	Liquid	00:14:43:27	0

Рисунок 9.4 – Список разгруженных кораблей за месяц

На рисунке 9.5 также представлен итоговый протокол симуляции, записанный в файл Statistic.txt. Этот файл включает общую сумму штрафов, число разгруженных судов, среднее время ожидания в очереди, максимальная задержку разгрузки, среднюю длину очереди для каждого типа груза.

Пример записи из файла Statistic.txt:



```
Общая сумма штрафа: 96000
Число разгруженных судов: 42
Среднее время ожидания в очереди в минутах: 2491 (в часах примерно 41,52)
Максимальное время ожидания в часах: 366,87 (в днях примерно = 15,29)
Средняя очередь за месяц для кораблей с грузом Bulk: 12,19
Средняя очередь за месяц для кораблей с грузом Liquid: 1,13
Средняя очередь за месяц для кораблей с грузом Container: 12,68
```

Рисунок 9.5 – Статистика за месяц работы порта

### 3.8. Листинг программы

#### Program.cs

```
namespace SeaPort
{
    class Program
    {
        static void Main(string[] args)
        {
            ulong QueneTime;
            string filePathStatistic = "Statistic.txt";
            string filePathPlannedSchedule = "PlannedSchedule.txt";
            string filePathActualSchedule = "ActualSchedule.txt";
            string filePathServedShips = "ServedShips.txt";
            DateTime dateTime = new DateTime(2024, 1, 1);
            Simulation simulation = new Simulation();
            Time time = new Time();
            time.StartTime(dateTime);
            Schedule schedule = new Schedule();
            Port port = new Port(1, 1, 1);

            // Генерация расписания
            schedule.GenerateScheduleShip(100, time.Start_Time);

            //schedule.Otladka();

            // Сортировка по дате прибытия идеального расписания
            var sortedByPlannedArrival = schedule.SortByScheduledArrival();
            // Вывод отсортированного идеального расписания
            ////!!schedule.PrintSchedule(sortedByPlannedArrival);

            //Сортировка по дате прибытия реального расписания
            var sortedByActualArrival = schedule.SortByActualArrival();
            // Вывод отсортированного реального расписания
            ////!!schedule.PrintScheduleReally(sortedByActualArrival);
            ////!!Console.WriteLine();

            // Запуск симуляции
            simulation.SimulateDays(31, time, port, schedule);

            //Выводим разгруженные корабли
            ////!!port.PrintServedShips();
            ////!!Console.WriteLine();
            /*
            //Выводим статистику
            Console.WriteLine($"Общая сумма штрафа: {port.FullFine}");
            Console.WriteLine($"Число разгруженных судов:
            {port.servedShips.Count}");
            QueneTime = 0;
            foreach (var ship in port.actualShips.Concat(port.servedShips))
            {
                QueneTime += (ulong)ship.Expectation.TotalMinutes;
            }
            QueneTime /= (ulong)(port.actualShips.Count +
            port.servedShips.Count);
            Console.WriteLine($"Среднее время ожидания в очереди в минутах:
            {QueneTime} (в часах пирмерно {(decimal)QueneTime / 60 :F2})");
            var maxExpectation =
            port.actualShips.Concat(port.servedShips).Max(ship => ship.Expectation.TotalHours);
            Console.WriteLine($"Максимальное время ожидания в часах:
            {maxExpectation:F2} (в днях примерно = {maxExpectation/24:F2})");
            foreach (var pair in simulation.hashTable)
            {
```



```

        public double CargoWeight { get; protected set; } //Вес груза
        public DateTime ScheduledArrival { get; protected set; } //Дата прибытия
        public int PlannedStayDays { get; protected set; } //Планируемый срок
        public DateTime DepartureDate { get; protected set; } //Планируемая дата

        protected AbstractShip(string name, CargoType cargoType, double
cargoWeight, DateTime scheduledArrival, int plannedStayDays)
        {
            Name = name;
            CargoType = cargoType;
            CargoWeight = cargoWeight;
            ScheduledArrival = scheduledArrival;
            PlannedStayDays = plannedStayDays;
            DepartureDate = ScheduledArrival.AddDays(PlannedStayDays);
        }
        public virtual void PrintShip()
        {
            Console.WriteLine("Ship Information:");
            Console.WriteLine($"Name: {Name}");
            Console.WriteLine($"Cargo Type: {CargoType}");
            Console.WriteLine($"Cargo Weight (kg) or Quantity (things):
{CargoWeight}");
            Console.WriteLine($"Scheduled Arrival: {ScheduledArrival}");
            Console.WriteLine($"Planned Stay (Days): {PlannedStayDays}");
            Console.WriteLine($"Departur Date: {DepartureDate}");
        }
    }
}

```

### ContainerCrane.cs

```

using SeaPort;

namespace SeaPort.Crane
{
    internal class ContainerCrane : AbstractCrane
    {
        public ContainerCrane()
        {
            CargoType = CargoType.Container;
            SpeedWorkInHour = 100; //100 контейнеров в час
        }
    }
}

```

### Container.cs

```

using SeaPort;

namespace SeaPort.Crane
{
    internal class ContainerCrane : AbstractCrane
    {
        public ContainerCrane()
        {
            CargoType = CargoType.Container;
            SpeedWorkInHour = 100; //100 контейнеров в час
        }
    }
}

```

## Ship.cs

```
using SeaPort.Crane;

namespace SeaPort
{
    internal class Ship : AbstractShip
    {
        public decimal UnloadingTime { get; private set; } //Время выгрузки без
        происшествий
        public decimal UnloadingTimeBOM { get; set; } //Время выгрузки с
        происшествиями

        private Random _random = new Random();
        public DateTime ActualArrival { get; private set; } // Реальное время
        прибытия
        public DateTime ActualDeparture { get; private set; } // Реальное время
        отъезда
        public decimal ExtraDelay { get; private set; } // Дополнительная
        задержка

        public TimeSpan DelayInPort { get; set; }
        public TimeSpan Expectation { get; set; } //Время ожидания до разгрузки

        public decimal LeftUnloadingTimeBOM { get; set; }

        public Ship(string name, CargoType cargoType, double cargoWeight,
        DateTime scheduledArrival, int plannedStayDays)
        : base(name, cargoType, cargoWeight, scheduledArrival,
        plannedStayDays)
        {
            DelayInPort = TimeSpan.Zero;
            Expectation = TimeSpan.Zero;
            // Рассчитываем случайные отклонения и задержки
            int arrivalDeviationDays; // Отклонение от расписания прибытия (-2
            до +9 дней)
            int delayDeviationDays; // Задержка выгрузки (0-5 часов)

            int probability_arrivals = _random.Next(0, 2); //Вероятность
            задержки прибытия
            int probability_BOM = _random.Next(0, 2); //Вероятность неожиданной
            задержки выгрузки

            if (probability_arrivals == 0) arrivalDeviationDays = _random.Next(-
            2, 4);
            else arrivalDeviationDays = 0;

            if (probability_BOM == 0) delayDeviationDays = _random.Next(1, 6);
            else delayDeviationDays = 0;

            ActualArrival = scheduledArrival.AddDays(arrivalDeviationDays);
            ExtraDelay = delayDeviationDays;

            switch (cargoType)
            {
                case CargoType.Bulk:
                    BulkCrane bulkCrane = new BulkCrane();
                    UnloadingTime = (decimal)cargoWeight /
                    bulkCrane.SpeedWorkInHour;
                    break;
                case CargoType.Liquid:
```

```

        LiquefiedCrane liquefiedCrane = new LiquefiedCrane();
        UnloadingTime = (decimal) cargoWeight /
liquefiedCrane.SpeedWorkInHour;
        break;
        case CargoType.Container:
            ContainerCrane containerCrane = new ContainerCrane();
            UnloadingTime = (decimal) cargoWeight /
containerCrane.SpeedWorkInHour;
            break;
        default:
            throw new Exception("Unknown cargo type");
    }
    UnloadingTimeBOM = ExtraDelay + UnloadingTime;
    LeftUnloadingTimeBOM = UnloadingTimeBOM;
    //Console.WriteLine(UnloadingTime);
}
// Переопределяем PrintShip, чтобы добавить информацию о времени
выгрузки
public override void PrintShip()
{
    // Выводим общую информацию о судне с помощью базового метода
    base.PrintShip();

    Console.WriteLine($"ActualArrival: {ActualArrival}");
    // Выводим информацию о времени выгрузки

    Console.WriteLine($"Unloading Time: {UnloadingTime:F2} hours");

    Console.WriteLine($"ExtraDelay: {ExtraDelay} hours");

    Console.WriteLine();
}
}
}

```

### LiquidCrane.cs

```

using SeaPort;

namespace SeaPort.Crane
{
    internal class LiquefiedCrane : AbstractCrane
    {
        public LiquefiedCrane()
        {
            CargoType = CargoType.Liquid;
            SpeedWorkInHour = 3000; //3000 кг в час
        }
    }
}

```

### BulkCrane.cs

```

using SeaPort;

namespace SeaPort.Crane
{
    internal class BulkCrane : AbstractCrane
    {
        public BulkCrane()
        {
            CargoType = CargoType.Bulk;
            SpeedWorkInHour = 1500; //1500 кг в час
        }
    }
}

```

```

    }
}

```

## StartTime.cs

```

namespace SeaPort
{
    internal class Time
    {
        public DateTime Start_Time { get; set; }
        public DateTime Current_Time { get; set; }
        public DateTime End_Time { get; set; }
        public Time()
        {
            Start_Time = new DateTime(2024, 1, 1);
            Current_Time = Start_Time;
            End_Time = Start_Time.AddDays(31);
        }
        public void StartTime(DateTime startDateTime)
        {
            Start_Time = startDateTime;
            Current_Time = startDateTime;
            End_Time = Start_Time.AddDays(31);
        }
        public void CurrentTime(int PassedTimeHours)
        {
            Current_Time.AddHours(PassedTimeHours);
        }
    }
}

```

## Schedule.cs

```

using System.Xml.Linq;

namespace SeaPort
{
    internal class Schedule
    {
        private List<Ship> scheduleShip = new List<Ship>(); // Расписание всех
кораблей на месяц

        // Генерация расписания судов
        public void GenerateScheduleShip(int numberOfShips, DateTime startDate)
        {
            int cargoTypeCount = Enum.GetValues(typeof(CargoType)).Length; //
Количество типов груза

            Random random = new Random();
            for (int i = 0; i < numberOfShips; i++)
            {
                string name = "Ship" + (i + 1); // Имя судна
                CargoType cargoType = (CargoType)(random.Next(0,
cargoTypeCount)); // Тип груза
                double cargoWeight = random.Next(20000, 100010); // Случайный
вес груза
                DateTime arrivalDate = startDate.AddDays(random.Next(0, 31)); //
Прибытие в течение месяца
                int plannedStayDays = random.Next(6, 14); // Планируемое время
стоянки

                // Создание судна
            }
        }
    }
}

```



```

        Ship ship = new Ship(name, cargoType, cargoWeight, arrivalDate,
plannedStayDays);

        // Добавление судна в расписание
        scheduleShip.Add(ship);
    }

    // Общая функция сортировки по произвольному ключу
    public List<Ship> SortSchedule(Func<Ship, object> keySelector)
    {
        return scheduleShip.OrderBy(keySelector).ToList();
    }

    // Функция сортировки по ScheduledArrival
    public List<Ship> SortByScheduledArrival()
    {
        return SortSchedule(ship => ship.ScheduledArrival);
    }
    public List<Ship> SortByActualArrival()
    {
        return SortSchedule(ship => ship.ActualArrival);
    }

    // Функция для вывода списка судов
    public void PrintSchedule(List<Ship> ships)
    {
        if (ships == null || ships.Count == 0)
        {
            Console.WriteLine("No ships to display.");
            return;
        }
        Console.WriteLine("\n+++++");
        Console.WriteLine("+ Schedule of Ships: +");
        Console.WriteLine("+++++\n");

        Console.WriteLine("\n-----");
        Console.WriteLine("
        | Name | CargoType | CargoWeight (kg) |
Scheduled Arrival | Planned Stay (days) | Departure Date |");
        Console.WriteLine("-----");

        foreach (var ship in ships)
        {
            Console.WriteLine($"
            | {ship.Name} | {ship.CargoType} |
{ship.CargoWeight} | {ship.ScheduledArrival} | {ship.PlannedStayDays}
            | {ship.DepartureDate} |");
            Console.WriteLine("-----");

            /*
            Console.WriteLine($"Name: {ship.Name}, Cargo Type:
{ship.CargoType}, Cargo Weight: {ship.CargoWeight} tons, " +
                $"Scheduled Arrival:
{ship.ScheduledArrival:dd-MM-yyyy}, Planned Stay: {ship.PlannedStayDays} days, " +
                $"Departure Date: {ship.DepartureDate:dd-MM-
yyyy}");*/
        }
    }
    public void PrintScheduleToFile(string fileName, List<Ship> ships)
    {
        // Проверяем, есть ли корабли
        if (ships == null || ships.Count == 0)
        {
            Console.WriteLine("No ships to display.");
            return;
        }
    }

```

```

    }

    // Открываем файл для записи
    using (StreamWriter writer = new StreamWriter(fileName, false))
    {
        writer.WriteLine("\n+++++++");
        writer.WriteLine("+ Schedule of Ships: +");
        writer.WriteLine("++++++\n");

        writer.WriteLine("\n-----");
        writer.WriteLine("Name | CargoType | CargoWeight (kg) |");
        writer.WriteLine("Scheduled Arrival | Planned Stay (days) | Departure Date |");
        writer.WriteLine("-----");

        foreach (var ship in ships)
        {
            writer.WriteLine($"| {ship.Name,-8} | {ship.CargoType,-8} | {ship.CargoWeight,-14} | {ship.ScheduledArrival:dd-MM-yyyy,-15} | {ship.PlannedStayDays,-10} | {ship.DepartureDate:dd-MM-yyyy,-15} |");
            writer.WriteLine("-----");
        }

        Console.WriteLine($"Schedule successfully written to {fileName}");
    }

    public void PrintScheduleReally(List<Ship> ships)
    {
        if (ships == null || ships.Count == 0)
        {
            Console.WriteLine("No ships to display.");
            return;
        }

        Console.WriteLine("Schedule of Ships:");
        foreach (var ship in ships)
        {
            Console.WriteLine($"Name: {ship.Name}, Cargo Type: {ship.CargoType}, Cargo Weight: {ship.CargoWeight} tons, " +
                $"Scheduled Actual Arrival: {ship.ActualArrival:dd-MM-yyyy}, Planned Stay: {ship.PlannedStayDays} days, " +
                $"Departure Date: {ship.DepartureDate:dd-MM-yyyy}");
        }
    }

    // Пример: вывод полного расписания
    public void PrintFullSchedule()
    {
        PrintSchedule(scheduleShip);
    }
}

```

## Simulation.cs

```

using System.Collections;

namespace SeaPort
{
    internal class Simulation

```





```

        break;
    case CargoType.Liquid:
        queue_liquid.Add(ship);
        ship.Expectation = time.Current_Time - ship.ActualArrival;
        break;
    case CargoType.Container:
        queue_container.Add(ship);
        ship.Expectation = time.Current_Time - ship.ActualArrival;
        break;
    default:
        Console.WriteLine("Unknown Cargotype (AddQueue)");
        return;
    }
}

public void RemoveQueue(Ship ship)
{
    switch (ship.CargoType)
    {
        case CargoType.Bulk:
            queue_bulk.Remove(ship);
            break;
        case CargoType.Liquid:
            queue_liquid.Remove(ship);
            break;
        case CargoType.Container:
            queue_container.Remove(ship);
            break;
        default:
            Console.WriteLine("Unknown CargoType (RemoveQueue)");
            return;
    }
}

// Проверяет, находится ли корабль в списке bulk
public bool IsShipInBulkQueue(Ship ship)
{
    return queue_bulk.Contains(ship);
}

// Проверяет, находится ли корабль в списке liquid
public bool IsShipInLiquidQueue(Ship ship)
{
    return queue_liquid.Contains(ship);
}

// Проверяет, находится ли корабль в списке container
public bool IsShipInContainerQueue(Ship ship)
{
    return queue_container.Contains(ship);
}

//Проверяет, находится ли корабль в очереди
public bool IsShipInQueue(Ship ship)
{
    switch (ship.CargoType)
    {
        case (CargoType.Bulk):
            return queue_bulk.Contains(ship);
        case CargoType.Liquid:
            return queue_liquid.Contains(ship);
        case CargoType.Container:
            return queue_container.Contains(ship);
        default:
            Console.WriteLine("Error IsShipInQueue");
            return false;
    }
}

```

```

}
// Вывод всех кораблей в очередях
public void PrintAllShipsInQueue()
{
    if (queue_bulk.Count > 0)
    {
        Console.WriteLine("\n -----");
        Console.WriteLine("| Bulk Queue: |");
        Console.WriteLine(" ----- \n");
        foreach (var ship in queue_bulk)
        {
            Console.WriteLine($"Ship: {ship.Name}, Arrival:
{ship.ActualArrival}, Expectation: {ship.Expectation:dd\\:hh\\:mm\\:ss},
UnloadingTimeBOM: {ship.UnloadingTimeBOM:F2} " +
                $"Left Unloading: {ship.LeftUnloadingTimeBOM:F2}");
        }
        Console.WriteLine();
    }
    else
    {
        Console.WriteLine("\n -----");
        Console.WriteLine("| Bulk Queue: | IS EMPTY!");
        Console.WriteLine(" -----");
    }
    if (queue_liquid.Count > 0)
    {
        Console.WriteLine("\n -----");
        Console.WriteLine("| Liquid Queue: |");
        Console.WriteLine(" ----- \n");
        foreach (var ship in queue_liquid)
        {
            Console.WriteLine($"Ship: {ship.Name}, Arrival:
{ship.ActualArrival}, Expectation: {ship.Expectation:dd\\:hh\\:mm\\:ss},
UnloadingTimeBOM: {ship.UnloadingTimeBOM:F2} " +
                $"Left Unloading: {ship.LeftUnloadingTimeBOM:F2}");
        }
        Console.WriteLine();
    }
    else
    {
        Console.WriteLine(" -----");
        Console.WriteLine("| Liquid Queue: | IS EMPTY!");
        Console.WriteLine(" -----");
    }
    if (queue_container.Count > 0)
    {
        Console.WriteLine(" -----");
        Console.WriteLine("| Container Queue: |");
        Console.WriteLine(" ----- \n");
        foreach (var ship in queue_container)
        {
            Console.WriteLine($"Ship: {ship.Name}, Arrival:
{ship.ActualArrival}, Expectation: {ship.Expectation:dd\\:hh\\:mm\\:ss},
UnloadingTimeBOM: {ship.UnloadingTimeBOM:F2}, " +
                $"Left Unloading: {ship.LeftUnloadingTimeBOM:F2}");
        }
    }
    else
    {
        Console.WriteLine(" -----");
        Console.WriteLine("| Container Queue: | IS EMPTY!");
        Console.WriteLine(" -----");
        Console.WriteLine();
    }
}

```

```

    }

    // Метод для обработки очереди
    public void ProcessQueue(List<Ship> queue, Port port, Time time,
CargoType cargoType)
    {
        decimal remainingTime = 24; // Изначально у нас 24 часа
        //decimal tmp; //Переменная для переноса времени
        foreach (var ship in queue.ToList()) // Преобразуем коллекцию в
'ToList()' для безопасной итерации
        {
            if (ship.LeftUnloadingTimeBOM <= remainingTime)
            {
                remainingTime -= ship.LeftUnloadingTimeBOM;
                ship.LeftUnloadingTimeBOM = 0; // Корабль разгрузился

                Console.WriteLine($"Ship {ship.Name} finished unloading and
is removed from queue.");
                port.actualShips.Remove(ship);
                queue.Remove(ship);

                if (time.Current_Time - ship.ActualArrival -
TimeSpan.FromDays(ship.PlannedStayDays) > TimeSpan.Zero)
                {
                    ship.DelayInPort = time.Current_Time -
ship.ActualArrival - TimeSpan.FromDays(ship.PlannedStayDays);
                }
                else { ship.DelayInPort = TimeSpan.Zero; }
                port.servedShips.Add(ship); //Разгруженные корабли

                do
                {
                    var nextShip = port.actualShips
.FirstOrDefault(s => s.CargoType == cargoType &&
!IsShipInQueue(s));

                    if (nextShip != null)
                    {
                        Console.WriteLine($"Adding new ship {nextShip.Name}
to the queue.");
                        queue.Add(nextShip);
                        TimeSpan remainingTimes =
TimeSpan.FromHours((double)remainingTime);
                        nextShip.Expectation = time.Current_Time -
ship.ActualArrival - remainingTimes;

                        Console.WriteLine($"Ship: {nextShip.Name}, Arrival:
{nextShip.ActualArrival}, Expectation: {nextShip.Expectation:dd\\:hh\\:mm\\:ss},
UnloadingTimeBOM: {nextShip.UnloadingTimeBOM:F2}");
                        if (nextShip.LeftUnloadingTimeBOM - remainingTime >
0)
                        {
                            nextShip.LeftUnloadingTimeBOM -= remainingTime;
                            break;
                        }
                        else
                        {
                            remainingTime -= nextShip.LeftUnloadingTimeBOM;
                            nextShip.LeftUnloadingTimeBOM = 0;
                            Console.WriteLine($"Ship {ship.Name} finished
unloading and is removed from queue.");
                            port.actualShips.Remove(nextShip);
                            queue.Remove(nextShip);
                        }
                    }
                }
            }
        }
    }

```

```

        if(time.Current_Time - nextShip.ActualArrival -
TimeSpan.FromDays(nextShip.PlannedStayDays) > TimeSpan.Zero)
        {
            nextShip.DelayInPort = time.Current_Time -
nextShip.ActualArrival - TimeSpan.FromDays(nextShip.PlannedStayDays);
        }
        else { nextShip.DelayInPort = TimeSpan.Zero; }

        port.servedShips.Add(nextShip);
    }
}
else break;
} while (remainingTime > 0);
}
else
{
    // Если времени не хватает на текущий корабль, переносим
остаток времени на следующий день
    ship.LeftUnloadingTimeBOM -= remainingTime;
    break; // Дальше уже нечего разгружать в этот день
}
}
}

public bool IsAnyShipInQueue()
{
    return queue_bulk.Count > 0 || queue_liquid.Count > 0 ||
queue_container.Count > 0;
}
public ulong GetShipCountInQueueReflection(CargoType cargoType)
{
    string fieldName = $"queue_{cargoType.ToString().ToLower()}";
    var field = GetType().GetField(fieldName);

    if (field != null && field.GetValue(this) is List<Ship> queue)
    {
        return (ulong)queue.Count;
    }
    else
    {
        throw new ArgumentException("Unknown CargoType or queue not
found");
    }
}
}
}
}

```

## Port.cs

```

using SeaPort.Crane;

namespace SeaPort
{
    internal class Port
    {
        public int BulkCranes { get; set; } //Количество "сыпучих" кранов
        public int LiquidCranes { get; set; } //Количество "жидких" кранов
        public int ContainerCranes { get; set; } //Количество "контейнерных"
кранов

        public List<Ship> actualShips = new List<Ship>(); //Корабли находящиеся
в порту на данный момент

        public List<Ship> servedShips = new List<Ship>();
    }
}

```



```

public ulong FullFine { get; set; }

public Port()
{
    FullFine = 0;
    BulkCranes = 0;
    LiquidCranes = 0;
    ContainerCranes = 0;
    actualShips = new List<Ship>();
}

public Port(int bulk_crane, int liquid_crane, int container_crane)
{
    FullFine = 0;
    BulkCranes = bulk_crane;
    LiquidCranes = liquid_crane;
    ContainerCranes = container_crane;
    actualShips = new List<Ship>();
}

// Функция для добавления кораблей с проверкой на совпадение даты
ActualArrival
schedule)
прибытия
public void AddActualShipWithTime(DateTime arrivalDate, Schedule
{
    // Получаем отсортированный список судов по фактическому времени
    var sortedByActualArrival = schedule.SortByActualArrival();

    foreach (var ship in sortedByActualArrival)
    {
        // Если дата прибытия судна совпадает с переданной датой
        if (ship.ActualArrival.Date == arrivalDate.Date)
        {
            // Добавляем судно в список
            actualShips.Add(ship);
        }
    }
}

// Функция для вывода списка судов, которые находятся в порту
public void PrintActualShips()
{
    if (actualShips.Count == 0)
    {
        Console.WriteLine("~ Ships currently in port: No ships in port!");
        Console.WriteLine("-----");
    }
    else
    {
        Console.WriteLine("Ships currently in port:");
        Console.WriteLine("-----");
        foreach (var ship in actualShips)
        {
            Console.WriteLine($"Ship Name: {ship.Name}, Arrival Date:
{ship.ActualArrival.ToShortDateString()}, Cargo Type: {ship.CargoType}");
        }
    }
}

// Функция для удаления корабля из списка по имени
public bool RemoveShipInPortByName(string shipName, Time time)
{
    // Найти корабль по имени

```

```

        var shipToRemove = actualShips.FirstOrDefault(ship =>
ship.Name.Equals(shipName, StringComparison.OrdinalIgnoreCase));

        if (shipToRemove != null)
        {
            actualShips.Remove(shipToRemove);
            Console.WriteLine($"Ship '{shipName}' has been removed from the
port.");

            return true; // Успешно удален
        }
        else
        {
            Console.WriteLine($"Ship '{shipName}' not found in the port.");
            return false; // Не найден
        }
    }
    // Функция для вывода списка обслуженных кораблей
    public void PrintServedShips()
    {
        if (servedShips.Count == 0)
        {
            Console.WriteLine("\n-----");
            Console.WriteLine("~ No served ships yet! ~");
            Console.WriteLine("-----");
        }
        else
        {
            Console.WriteLine("\n-----");
            Console.WriteLine("Served Ships:");
            Console.WriteLine("-----");

            Console.WriteLine("\n-----");
            Console.WriteLine($"| Ship Name | Arrival Date | Cargo
Type | Expectation | DelayInPort(day) |");
            Console.WriteLine("-----");

            foreach (var ship in servedShips)
            {
                Console.WriteLine($"| {ship.Name} |
{ship.ActualArrival.ToShortDateString()} | {ship.CargoType} |
{ship.Expectation:dd\\:hh\\:mm\\:ss} | {ship.DelayInPort.Days} |");
                Console.WriteLine("-----");
            }
        }
    }

    public void PrintServedShipsToFile(string fileName)
    {
        // Проверка на наличие разгруженных судов
        if (servedShips.Count == 0)
        {
            using (StreamWriter writer = new StreamWriter(fileName, false))
            {
                writer.WriteLine("\n-----");
                writer.WriteLine("~ No served ships yet! ~");
                writer.WriteLine("-----");
            }
            Console.WriteLine("No served ships yet. Information written to
file.");
        }
        else
    }

```

```

        {
            using (StreamWriter writer = new StreamWriter(fileName, false))
            {
                writer.WriteLine("\n-----");
                writer.WriteLine("Served Ships:");
                writer.WriteLine("-----");

                writer.WriteLine("\n-----");
                writer.WriteLine("-----");
                writer.WriteLine($"| Ship Name | Arrival Date |");
                writer.WriteLine("Cargo Type | Expectation | DelayInPort(day) |");
                writer.WriteLine("-----");
                writer.WriteLine("-----");

                foreach (var ship in servedShips)
                {
                    writer.WriteLine($"| {ship.Name} |");
                    {ship.ActualArrival.ToShortDateString()} | {ship.CargoType} |
                    {ship.Expectation:dd\\:hh\\:mm\\:ss} | {ship.DelayInPort.Days} |");
                    writer.WriteLine("-----");
                }
            }
            Console.WriteLine($"Served ships information successfully
written to {fileName}");
        }
    }
    public int GetShipCountInPort(CargoType cargoType)
    {
        return actualShips.Count(ship => ship.CargoType == cargoType);
    }
}

```

### CargoType.cs

```

namespace SeaPort
{
    public enum CargoType
    {
        Bulk, //Сыпучие
        Liquid, //Жидкие
        Container //Контейнеры
    }
}

```

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы были разработаны три программы, построенные на основе принципов объектно-ориентированного программирования (ООП). Программы демонстрируют практическое применение таких ключевых аспектов ООП, как наследование, полиморфизм, инкапсуляция и работа с абстрактными классами и интерфейсами.

В ходе работы были закреплены навыки работы с файлами, включая чтение, запись и структурирование данных в текстовых файлах. Особое внимание уделялось проектированию архитектуры приложений, включая использование абстрактных классов для определения базовых моделей и интерфейсов для реализации общих методов, что позволило создать модульный и легко расширяемый код.

Программы продемонстрировали способность моделировать сложные и динамические процессы, такие как взаимодействие объектов с учетом случайных факторов и управление очередями. Реализация симуляций потребовала применения алгоритмов для обработки событий, что способствовало развитию навыков в построении имитационных моделей.

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Microsoft Learn – сеть разработчиков Microsoft. URL: <https://learn.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 08.12.2024)
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 C# 12 и платформе .NET 8. URL: <https://metanit.com/sharp/tutorial/> (дата обращения: 01.12.2024)