

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

КУРСОВАЯ РАБОТА
по дисциплине «Объектно-ориентированное программирование»
Тема: Разработка приложений на основе принципов объектно-
ориентированного подхода

Студентка гр. 9302

Гелета А. И.

Преподаватель

Новакова Н.Е.

Санкт-Петербург

2021

ЗАДАНИЕ КУРСОВАЯ РАБОТА

Студентка Гелета А.И.

Группа 9302

Тема работы: разработка приложений на основе принципов объектно-ориентированного подхода

Исходные данные:

Поставленное задание:

Язык программирования: C#

Содержание пояснительной записки:

«Содержание», «Введение», «Цель работы», «Первый раздел», «Второй раздел», «Третий раздел», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее ? страниц.

Дата выдачи задания: 08.02.2021

Дата сдачи курсовой работы:

Дата защиты курсовой работы:

Студентка

Гелета Т.А.

Преподаватель

Новакова Н.Е.

АННОТАЦИЯ

Курсовая работа содержит в себе решения таких поставленных задач, как создание иерархии классов, реализация алгоритма нахождения количества остовов графа и минимального остовного дерева, разработка имитационной модели.

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

SUMMARY

Course work contains solutions to such tasks as creating a class hierarchy, realization of algorithm that finds amount of graph's spanning trees and algorithm of finding minimal spanning tree, developing an imitation model.

Based on these models, apps with various user interfaces were created. All results are presented inside the course work.

Оглавление

ВВЕДЕНИЕ	6
ЦЕЛЬ РАБОТЫ	6
1 ПЕРВЫЙ РАЗДЕЛ	7
Задание	7
Теоретический аспект	7
Формализация задачи	7
Спецификация	9
Руководство пользователя	10
Руководство программиста	11
Контрольный пример	11
Текст программы	13
2 ВТОРОЙ РАЗДЕЛ	19
Задание	19
Теоретический аспект	19
Формализация задачи	20
Спецификация	23
Руководство пользователя	25
Руководство программиста	28
Контрольный пример	29
Текст программы	31
3 ТРЕТИЙ РАЗДЕЛ	40
Задание	40
Теоретический аспект	40

Формализация задачи	41
Спецификация	46
Руководство пользователя	49
Руководство программиста	52
Контрольный пример	53
Текст программы.....	56
ЗАКЛЮЧЕНИЕ	64
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	65

ВВЕДЕНИЕ

Курсовая работа направлена на создание приложений на основе объектно-ориентированного подхода на языке С#. В ней рассматриваются иерархии классов и наследования, графы и алгоритмы, связанные с нахождением остовов и минимального остова графа, а также имитационные модели. Для первых двух задач реализуется консольное приложение для взаимодействия с пользователем, для третьей реализуется графический пользовательский интерфейс в Windows Forms.

ЦЕЛЬ РАБОТЫ

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

1 ПЕРВЫЙ РАЗДЕЛ

Задание

Вариант 6

Требуется: разработать программу для обеспечения работы библиотеки университета с использованием наследования и применением интерфейса.

Теоретический аспект

В данной работе под библиотекой понимается учреждение, которое собирает и хранит книги для общественного использования. Работа университетской библиотеки состоит из выдачи литературы (книг, газет, журналов) студентам и возврата этой же литературы.

Формализация задачи

Входные и выходные данные

Входные данные: запрос на выдачу или возврат книги/журнала/газеты, который включает в себя идентификационный номер нужной литературы.

Выходные данные: сообщение об успешном или неуспешном возврате литературы, либо данные о выдаче литературы (полное название, автор, количество страниц и т. д.)

Используемые классы и методы

В классе Test объявлен метод Main(), являющийся точкой входа в программу. В нем создается университетская библиотека и моделируется пример ее работы (какие-то книги добавляются, кто-то берет и возвращает книги).

Объявлен абстрактный класс Literature, от которого наследуются классы Book, Journal и Paper (виды литературы, существующие в библиотеке). Также объявлен интерфейс Library, от которого наследуется класс UniversityLibrary (в нем содержатся методы для добавления книги в

библиотеку, выдачи и возврата книг, а также для вывода информации о доступной в текущий момент литературе). Подробнее методы и поля данных классов представлены на рисунке 1.1

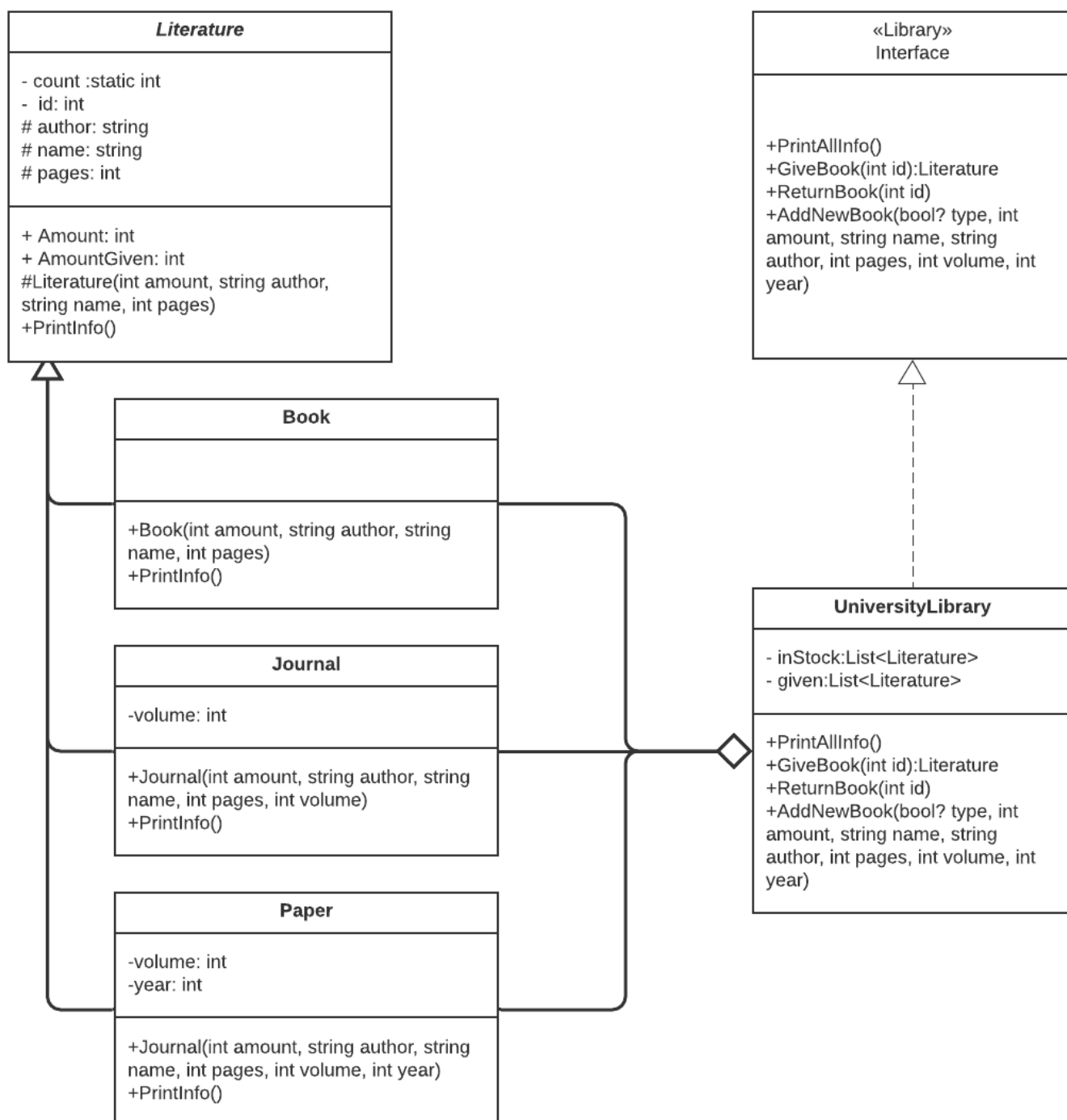


Рисунок 1.1 Диаграмма классов[2]

Спецификация

abstract class Literature

Поле	Назначение
count	Общее количество существующей литературы
id	Идентификационный номер
author	Имя автора
name	Имя произведения
pages	Количество страниц
Метод	Назначение
Amount	Возвращает количество литературы определенного вида
AmountGiven	Возвращает количество литературы, выданное
PrintInfo	Выводит информацию

class Book, наследует поля и методы от Literature, а также переопределяет:

Метод	Назначение
PrintInfo	Выводит информацию о книге

class Journal, наследует поля и методы от Literature, а также:

Поле	Назначение
volume	Номер выпуска
Метод	Назначение
PrintInfo	Выводит информацию о журнале

class Paper, наследует поля и методы от Literature, а также:

Поле	Назначение
volume	Номер выпуска
year	Год выпуска

Метод	Назначение
PrintInfo	Выводит информацию о газете

class UniversityLibrary

Поле	Назначение
inStock	Список литературы, который сейчас в наличии в библиотеке
given	Список литературы, отданный в пользование
Метод	Назначение
PrintAllInfo	Выводит информацию обо всей литературе
GiveBook	Выдача книги
ReturnBook	Возврат книги в библиотеку
AddNewBook	Добавление новой литературы в библиотеку

Руководство пользователя

Условие выполнения программы

Для запуска программы необходима установленная операционная система Windows. Приложение весит 40 Кб.

Выполнение программы

Программа разработана в виде консольного приложения. При его запуске показан список литературы, вместе с идентификационными номерами, по которым можно получить книгу (только по ним, дополнительный функционал с поиском по имени и автору разработан на данный момент не был).

Потом программа спрашивает, хочет ли пользователь взять книгу из библиотеки или вернуть (нужно ввести 1 или 0 и нажать «Enter»), после чего просит ввести идентификационный номер. После каждого действия уточняется, хочет ли пользователь продолжить работу или хочет выйти из программы.

При вводе неправильной цифры выводится сообщение о том, что был введен неверный номер.

Возможности добавить новые книги в консольном приложении нет, так как предполагается, что программа предоставляет интерфейс работы для пользователя (студента), который, может только получать и возвращать книги.

Руководство программиста

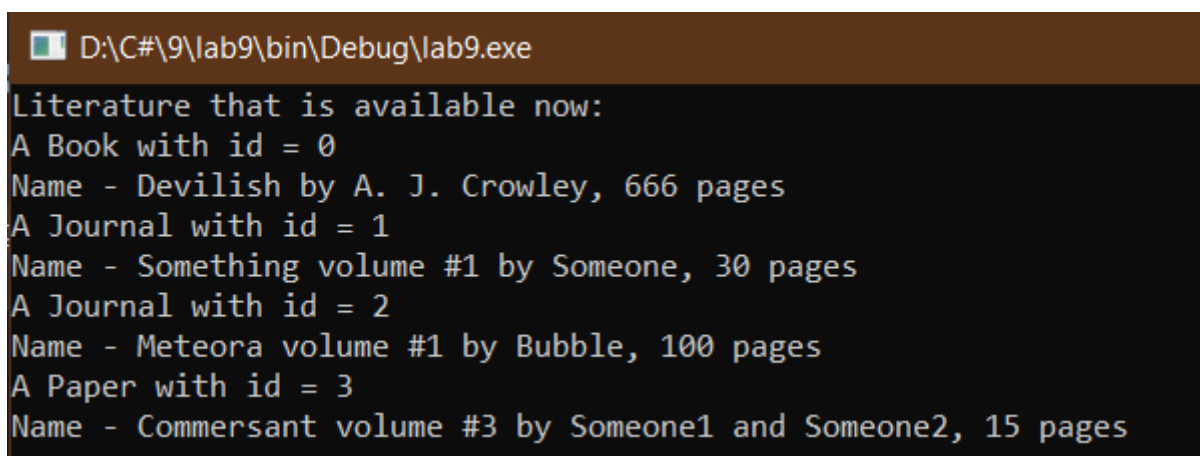
Запуск программы

Необходима операционная система Windows. При разработке использовалась Microsoft Visual Studio и платформа .NET Framework 4.7.2.

При запуске программа использует не более 7 Мб оперативной памяти. Приложение занимает в памяти 40 Кб. Весь проект Visual Studio занимает в памяти 168 Кб.

Контрольный пример

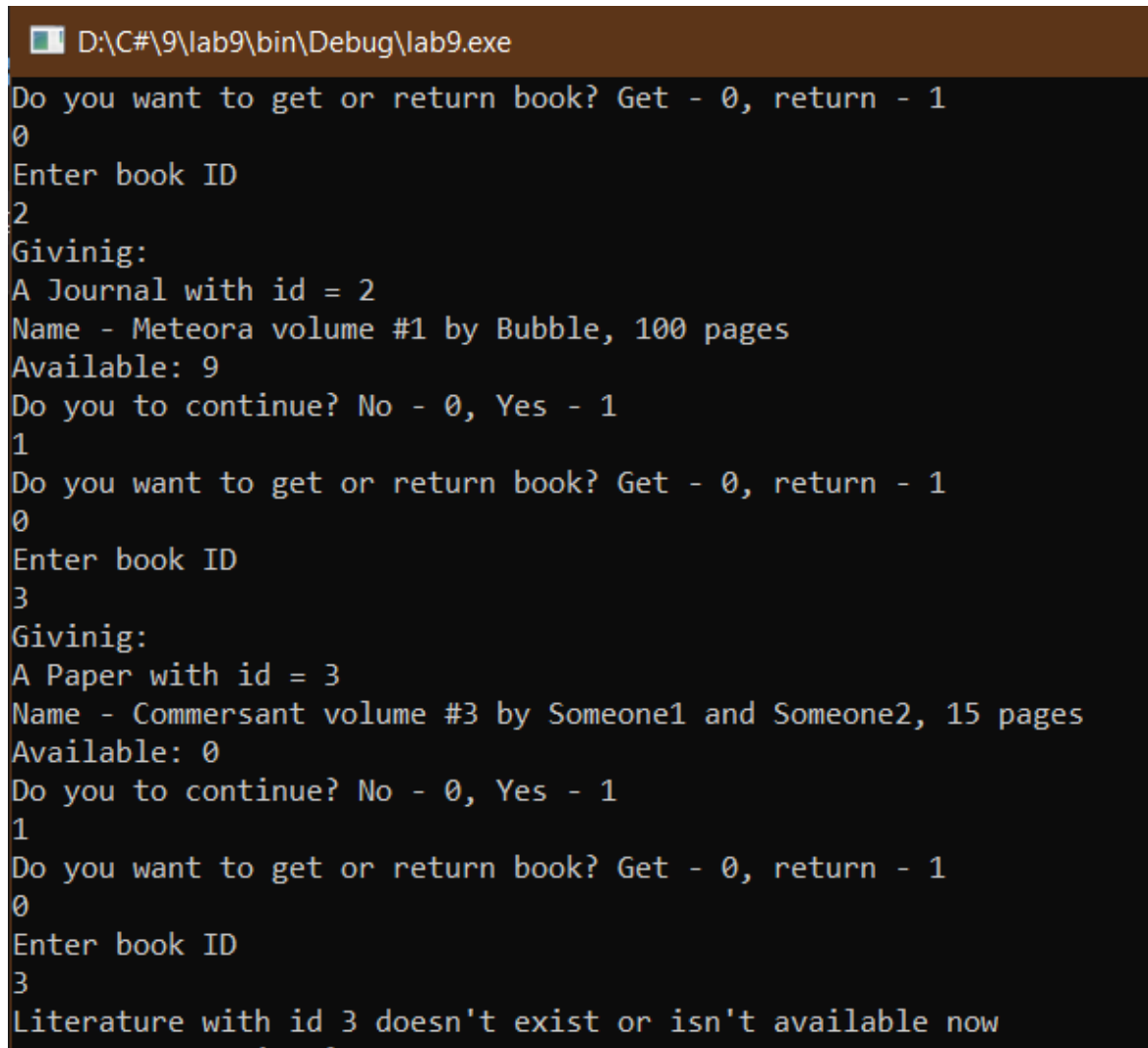
В контрольном примере изначально создается библиотека, в которой хранится 4 вида литературы: одна книга (с идентификационным номером 0) в 7 экземплярах, два журнала (id 1, 2) в 5 и 10 экземплярах соответственно и одна газета (id 3) в единственном экземпляре. Список всех существующих книг приводится в начале работы программы, в виде, показанном на рисунке 1.3:

A screenshot of a Windows command prompt window. The title bar shows the file path "D:\C#\9\lab9\bin\Debug\lab9.exe". The console text displays a list of available literature items. The text is as follows:

```
Literature that is available now:  
A Book with id = 0  
Name - Devilish by A. J. Crowley, 666 pages  
A Journal with id = 1  
Name - Something volume #1 by Someone, 30 pages  
A Journal with id = 2  
Name - Meteora volume #1 by Bubble, 100 pages  
A Paper with id = 3  
Name - Commersant volume #3 by Someone1 and Someone2, 15 pages
```

Рисунок 1.2. Список всех книг

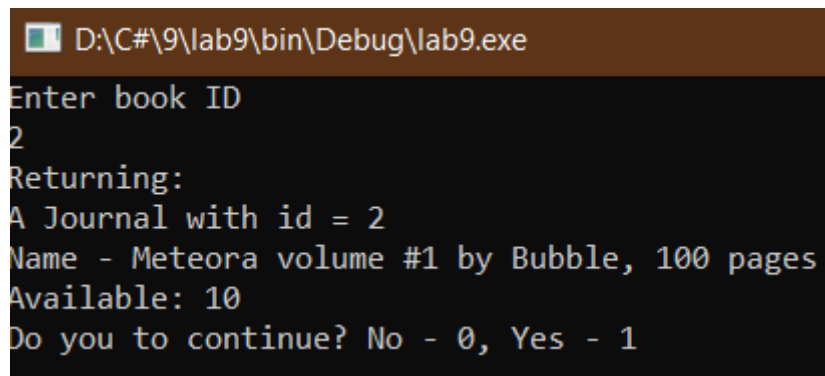
В контрольном примере на рисунке 1.3 показано моделирование работы, результаты которой наглядно приведены на рисунке 1.3: выдается журнал с id 2, газета с id 3, потом кто-то пытается взять газету с id 3, но так как ее единственный экземпляр уже был выдан, то выдается соответствующее предупреждение. Предупреждение выдается и при попытке взять книгу с неправильным id (несуществующим).



```
D:\C#\9\lab9\bin\Debug\lab9.exe
Do you want to get or return book? Get - 0, return - 1
0
Enter book ID
2
Givinig:
A Journal with id = 2
Name - Meteora volume #1 by Bubble, 100 pages
Available: 9
Do you to continue? No - 0, Yes - 1
1
Do you want to get or return book? Get - 0, return - 1
0
Enter book ID
3
Givinig:
A Paper with id = 3
Name - Commersant volume #3 by Someone1 and Someone2, 15 pages
Available: 0
Do you to continue? No - 0, Yes - 1
1
Do you want to get or return book? Get - 0, return - 1
0
Enter book ID
3
Literature with id 3 doesn't exist or isn't available now
```

Рисунок 1.3. Получение книг

Осуществлять возврат книг в библиотеку также возможно. Так, например, взятый журнал с номером 2 (был взят ранее, рисунок 1.3), можно успешно вернуть в библиотеку.



```
D:\C#\9\lab9\bin\Debug\lab9.exe
Enter book ID
2
Returning:
A Journal with id = 2
Name - Meteora volume #1 by Bubble, 100 pages
Available: 10
Do you to continue? No - 0, Yes - 1
```

Рисунок 1.4. Возврат книг

Если при вопросе о том, нужно ли продолжить работу, выбирается 0, то программа заканчивает свою работу.

Текст программы

Literature.cs

```
using System;
using System.Collections.Generic;

namespace lab9
{
    abstract class Literature
    {
        private static int count = 0;
        int id;
        protected string author; //can be publisher in case of journals
        protected string name;
        protected int pages;
        //how much is in stock
        public int Amount { get; set; }
        //how much is given to students
        public int AmountGiven { get; set; }
        public int Id
        {
            get { return id; }
        }
        protected Literature(int amount, string author, string name, int pages)
        {
            this.id = count;
            Amount = amount;
            this.author = author;
        }
    }
}
```

```

        this.name = name;
        this.pages = pages;
        AmountGiven = 0;
        count++;
    }
    public abstract void PrintInfo();
}

class Book : Literature
{
    public Book(int amount, string author, string name, int pages) :
base(amount, author, name, pages) { }
    public override void PrintInfo()
    {
        string type = this.GetType().ToString();
        Console.WriteLine("A " + type.Substring(type.IndexOf(".") + 1) + " with
id = " + Id);
        Console.WriteLine("Name - " + this.name + " by " + this.author + ", "
+ this.pages + " pages");
    }
}

class Journal : Literature
{
    int volume;
    public Journal(int amount, string author, string name, int pages, int
volume) : base(amount, author, name, pages) { this.volume = volume; }
    public override void PrintInfo()
    {
        string type = this.GetType().ToString();
        Console.WriteLine("A " + type.Substring(type.IndexOf(".") + 1) + "
with id = " + Id);
        Console.WriteLine("Name - " + this.name + " volume #" + this.volume +
" by " + this.author + ", " + this.pages + " pages");
    }
}

class Paper : Literature
{
    int volume;
    int year;
    public Paper(int amount, string author, string name, int pages, int
volume, int year) : base(amount, author, name, pages) { this.volume = volume;
this.year = year; }
}

```

```

        public override void PrintInfo()
        {
            string type = this.GetType().ToString();
            Console.WriteLine("A " + type.Substring(type.IndexOf(".") + 1) + "
with id = " + Id);
            Console.WriteLine("Name - " + this.name + " volume #" + this.volume +
" by " + this.author + ", " + this.pages + " pages");
        }
    }
}

```

Library.cs

```

using System;
using System.Collections.Generic;

namespace lab9
{
    interface Library
    {
        void AddNewBook(bool? type, int amount, string name, string author, int
pages, int volume = 0, int year = 0);
        void PrintAllInfo();
        Literature GiveBook(int id);
        void ReturnBook(int id);
    }

    class UniversityLibrary : Library
    {
        List<Literature> inStock = new List<Literature>(); //что есть в
библиотеке сейчас
        List<Literature> given = new List<Literature>(); //что выдано
        public void AddNewBook(bool? type, int amount, string name, string
author, int pages, int volume = 0, int year = 0)
        {
            //new book in the library
            //в библиотеку завезли новую книгу
            //null - book, true - journal, false - paper
            Literature lit;
            if (type == null) lit = new Book(amount, author, name, pages);
            else if (type == true) lit = new Journal(amount, author, name, pages,
volume);
            else lit = new Paper(amount, author, name, pages, volume, year);

```

```

        inStock.Add(lit);
    }
    public void PrintAllInfo()
    {
        Console.WriteLine("Literature that is available now: ");
        foreach (Literature lit in inStock)
            lit.PrintInfo();
    }
    public Literature GiveBook(int id)
    {
        //someone takes book from library
        //кто-то берет книгу из библиотеки
        foreach (Literature lit in inStock)
            if (lit.Id == id) {
                Console.WriteLine("Givinig: "); lit.PrintInfo();
                if (lit.Amount == 1) inStock.Remove(lit);
                lit.Amount--;
                if (lit.AmountGiven == 0) given.Add(lit);
                lit.AmountGiven++;
                Console.WriteLine("Available: " + lit.Amount);
                return lit;
            }
        Console.WriteLine("Literature with id " + id.ToString() + " doesn't
exist or isn't available now");
        return null;
    }

    public void ReturnBook(int id)
    {
        //someone returns book to the library
        //кто-то возвращает книгу в библиотеку
        foreach (Literature lit in given)
            if (lit.Id == id)
            {
                Console.WriteLine("Returning: "); lit.PrintInfo();
                if (lit.AmountGiven == 1) given.Remove(lit);
                lit.AmountGiven--;
                if (lit.Amount == 0) inStock.Add(lit);
                lit.Amount++;
                Console.WriteLine("Available: " + lit.Amount);
                return;
            }
    }

```



```

        Console.WriteLine("Literature with id " + id.ToString() + " doesn't
exist or weren't given to anyone");
    }
}
}
}
Program.cs
using System;

namespace lab9
{
    class Program
    {
        static void Main(string[] args)
        {
            //create library
            Library library = new UniversityLibrary();
            //adding some books
            library.AddNewBook(null, 7, "Devilish", "A. J. Crowley", 666);
            library.AddNewBook(true, 5, "Something", "Someone", 30, 1);
            library.AddNewBook(true, 10, "Meteora", "Bubble", 100, 1);
            library.AddNewBook(false, 1, "Commersant", "Someone1 and Someone2", 15, 3,
2000);

            library.PrintAllInfo();
            //giving out books (if book doesn't exist, nothing happens)
            Console.WriteLine();
            Interface(library);
        }

        public static void Interface(Library library)
        {
            Console.WriteLine("Do you want to get or return book? Get - 0, return - 1");
            string read = Console.ReadLine();
            if (read == "0")
            {
                Console.WriteLine("Enter book ID");
                int id = Int32.Parse(Console.ReadLine());
                library.GiveBook(id);
            }
            else if (read == "1")
            {
                Console.WriteLine("Enter book ID");
                int id = Int32.Parse(Console.ReadLine());
                library.ReturnBook(id);
            }
            else Console.WriteLine("Wrong number");

            if (Continue()) Interface(library);
        }

        public static bool Continue()
        {
            Console.WriteLine("Do you to continue? No - 0, Yes - 1");
            string read = Console.ReadLine();
            if (read == "0")
            {
                return false;
            }
            else if (read == "1")
            {

```

```
        return true;
    }
    else { Console.WriteLine("Wrong number"); return Continue(); }
}
}
```

2 ВТОРОЙ РАЗДЕЛ

Задание

Вариант Г-46-1-бс-1

Дан взвешенный неориентированный граф. Найти число остовов графа и остов наименьшего веса, используя алгоритм ближайшего соседа.

Теоретический аспект

Определение

Неориентированный граф – пара $G = (V, E)$, где V – множество вершин, а E – множество ребер: $\{\{v, u\}: v, u \in V\}$. Ребро – упорядоченная пара вершин[3]. Если каждому ребро соответствует какое-либо значение, то граф называется взвешенным.

Представление графа

Граф можно представить в виде списка смежности – списка ребер графа. Для взвешенного графа это набор из двух вершин и веса ребра.

Подсчет числа остовов

Остов графа – граф, не содержащий циклов и состоящий из некоторых ребер исходного графа и всех его вершин. Любой остов – дерево.

Для подсчета числа остовов графа сначала необходимо найти матрицу Кирхгофа, так как существует теорема, гласящая, что число остовов графа равно алгебраическому дополнению любого элемента матрицы Кирхгофа.

Матрица квадратная, а ее размерность – количество вершин на количество вершин. Если вершины i и j смежные (между ними проходит ребро), то соответствующий элемент матрицы $b_{ij} = -1$, иначе он равен 0, а элементы на диагонали равны степени вершины (т.е. количеству ребер, которые из нее выходят).

Далее берется минор любого элемента составленной матрицы и считается его определитель, например, по формуле 2.1

$$\Delta = \sum_{i=0}^N (-1)^{i+j} a_{ij} M_{ij} \quad (2.1)$$

Где j – любой столбец/строка (формула фиксирует определенное значение по столбцу или строке и проходит по всем ее элементам), N – размерность матрицы, M – минор соответствующего элемента.

Нахождение остова минимального веса с помощью алгоритма ближайшего соседа

Также алгоритм известен под названием «алгоритм Прима». Он выглядит следующим образом:

1. Отмечаем произвольную вершину графа, с которой начнется построение. Строим ребро наименьшего веса, инцидентное этой вершине.
2. Ищем ребро минимального веса, инцидентное одной из двух полученных вершин. В множество поиска не входит построенное ребро.
3. Продолжаем далее, разыскивая каждый раз ребро наименьшего веса, инцидентное построенным вершинам, не включая в круг поиска все ребра, их соединяющие.

Формализация задачи

Хранение графа удобнее всего для данной задачи осуществлять в виде списка ребер. Для определения ребра в программе имеется класс Edge, в котором определяются две вершины (первая вершина меньше по своему порядковому номеру, чем вторая, для удобства работы с классом) и вес ребра. Класс ребра никаких методов, кроме конструктора, не реализовывает.

В классе Graph хранится вся информация о графе: количество вершин, список ребер (характеризующий граф), матрица Кирхгофа и минимальное остовное дерево, найденное с помощью алгоритма ближайшего соседа.

Метод FindKirchhoff находит матрицу Кирхгофа, если поле матрицы пустое, и ничего не делает, если матрица уже была найдена ранее. Ее

можно легко составить с помощью списка ребер, так как в классе Edge как раз хранятся номера вершин, поэтому по ним мы сможем присваивать матрице -1 и увеличивать значение степени этих вершин на 1 на каждой итерации.

В классе Matrix представлен метод FindMinor, который находит минор матрицы по переданным значениям индекса элемента (ряд и колонка). Метод Determinant рекурсивно ищет определитель матрицы по формуле (2.1). Если передается матрица 2 на 2, то возвращается определитель, посчитанный по формуле $\Delta = a_{11} * a_{22} - a_{12} * a_{21}$

С помощью этих двух методов метод OstovNum из класса Graph может посчитать количество остовов. Для этого берется любой из миноров матрицы Кирхгофа, а потом считается его определитель.

Реализация метода PrimaAlg (алгоритма Прима для нахождения остова минимального веса) представлена в виде блок-схемы на рисунке 2.1. Перед началом работы задаются списки использованных вершин (пустой), неиспользованных вершин (все вершины), неиспользованных ребер (все ребра).

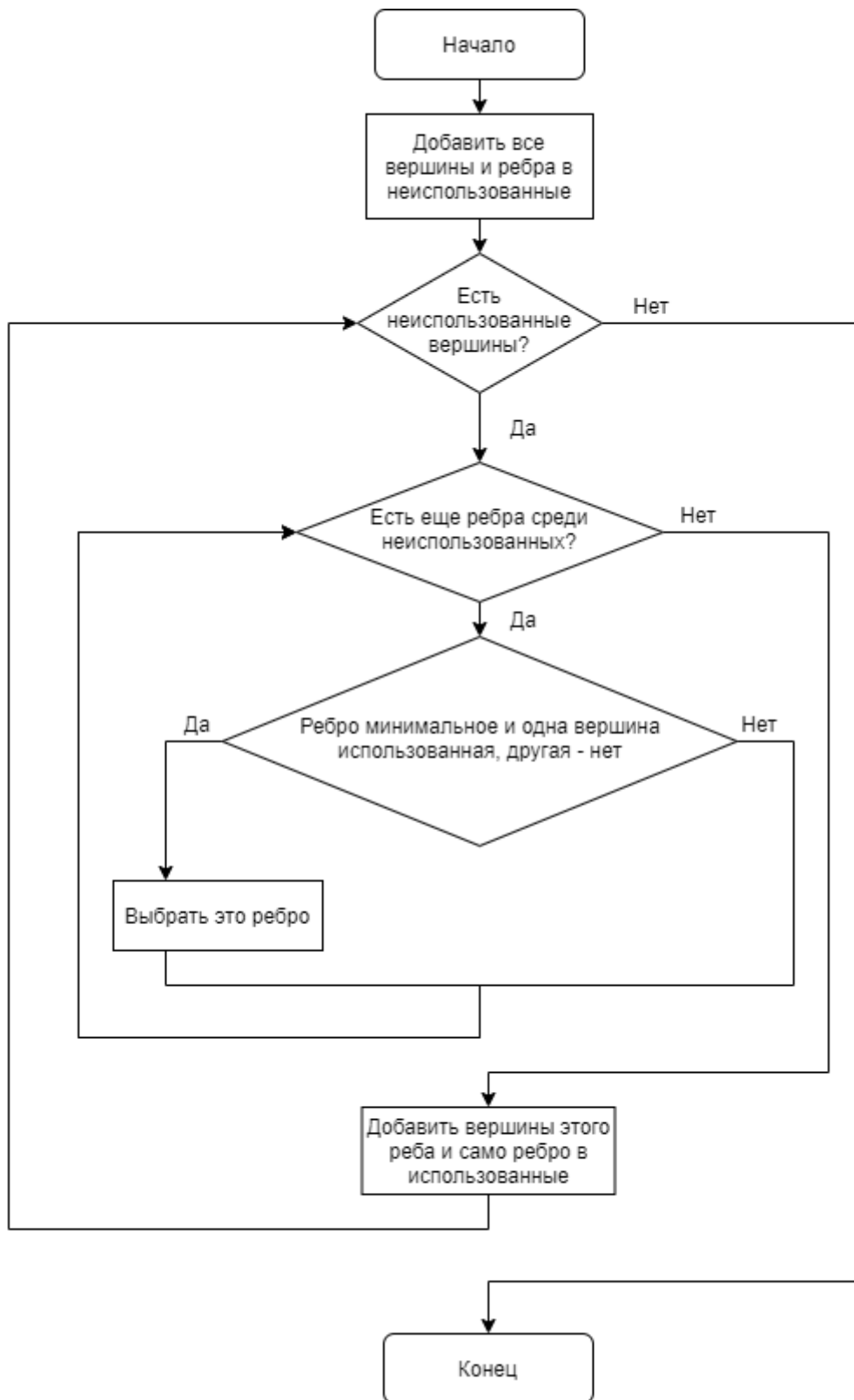


Рисунок 2.1. Алгоритм Прима

Также в классах `ConsolePrint` и `FilePrint` присутствуют такие методы, как `PrintGraph`, `PrintMatrix`, которые выводят граф и матрицу Кирхгофа в

консоль и в файл соответственно. Структура программы представлена на рисунке 2.2.

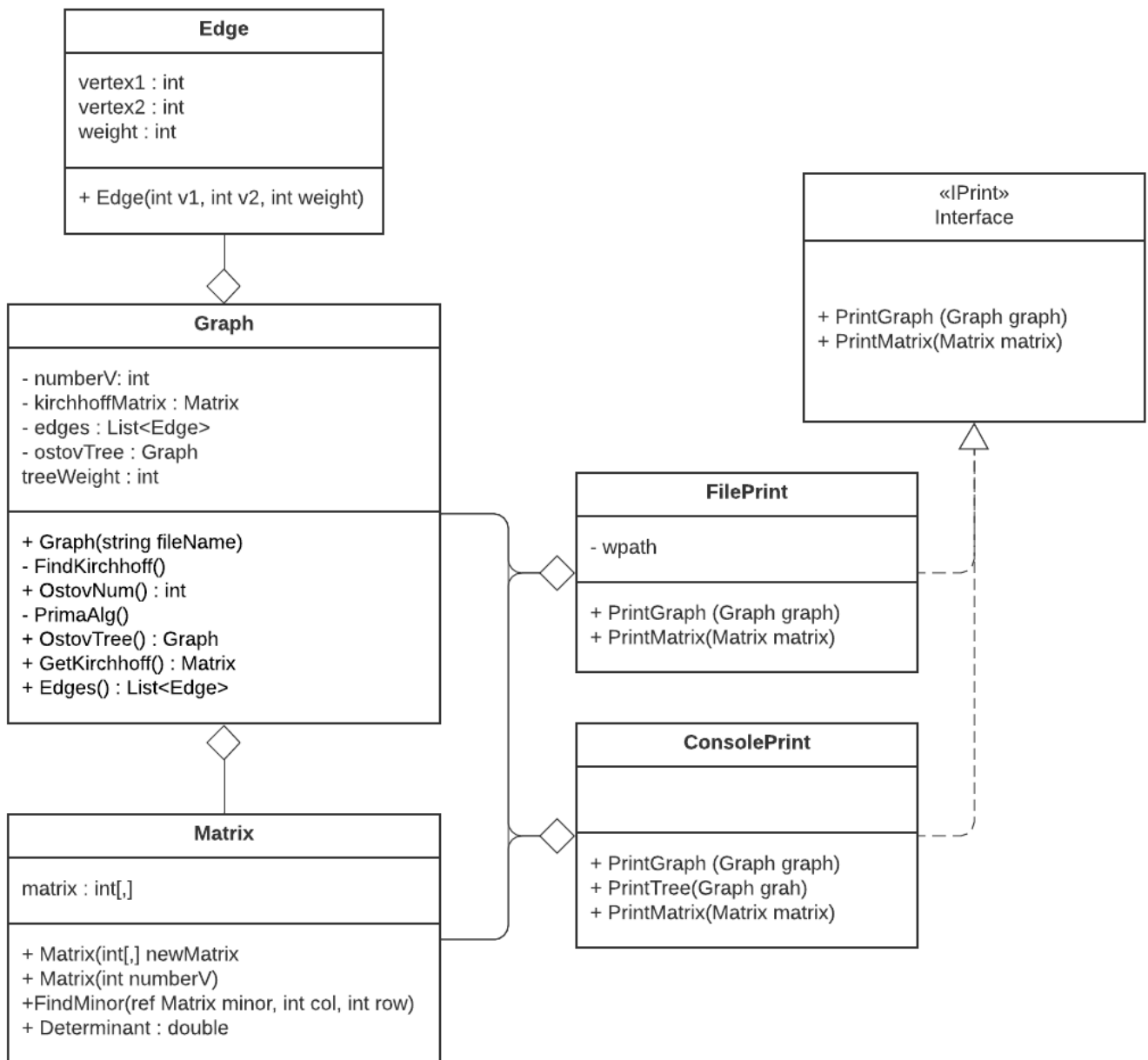


Рисунок 2.2. Диаграмма классов

Спецификация

class Edge

Поле	Назначение
vertex1	Первая вершина
vertex2	Вторая вершина

Поле	Назначение
weight	Вес ребра
Метод	Назначение
Edge	Конструктор, создает экземпляр класса (ребро)

Class Matrix

Поле	Назначение
matrix	Хранит значения матрицы
Метод	Назначение
Matrix	Создает матрицу
FindMinor	Находит минор матрицы
Determinant	Находит определитель матрицы

class Graph

Поле	Назначение
numberV	Количество вершин
kirchhoffMatrix	Матрица Кирхгофа
edges	Список ребер
ostovTree	Минимальное остовное дерево
treeWeight	Вес минимального остова
Метод	Назначение
Graph	Конструктор
FindKirchhoff	Ищет матрицу Кирхгофа для данного графа
OstovNum	Ищет количество остовов
PrimaAlg	Ищет минимальное остовное дерева
OstovTree	Возвращает остовное дерево
GetKirchhoff	Возвращает матрицу Кирхгофа
Edges	Возвращает список ребер графа

class ConsolePrinter

Метод	Назначение
PrintGraph	Выводит граф в консоль
PrintTree	Выводит остовное дерево в консоль
PrintMatrix	Выводит матрицу в консоль

class FilePrinter

Поле	Назначение
wpath	Путь для записи
Метод	Назначение
PrintGraph	Выводит минимальный остов
PrintMatrix	Выводит матрицу

Руководство пользователя

Условие выполнения программы

Для запуска программы необходима установленная операционная система Windows. Приложение весит 46 Кб.

Работа программы

Программа считывает данные об одном графе из файла в текущей директории (название пользователь задает при запуске программы, в консольном приложении), после чего выполняет обработку и выводит информацию в консоль и в файл в текущей директории (имя файла также задает пользователь в консольном приложении).

Вид входного файла

В качестве примера возьмем граф на рисунке 2.3. Для корректной работы программы необходимо, чтобы вершины были пронумерованы с 0, далее в любом порядке.

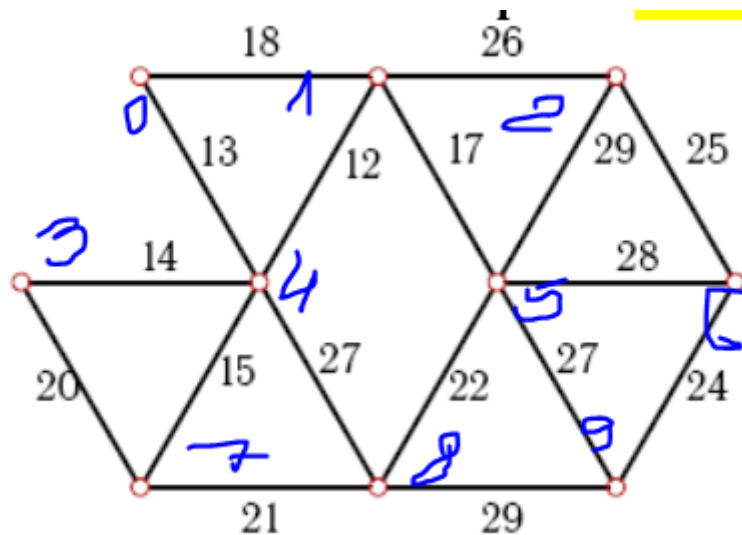


Рисунок 2.3. Нумерация вершин

Программа обрабатывает файлы только с определенным форматом записи данных. Так, правильно записанный в файл граф с рисунка 2.3 представлен на рисунке 2.4.

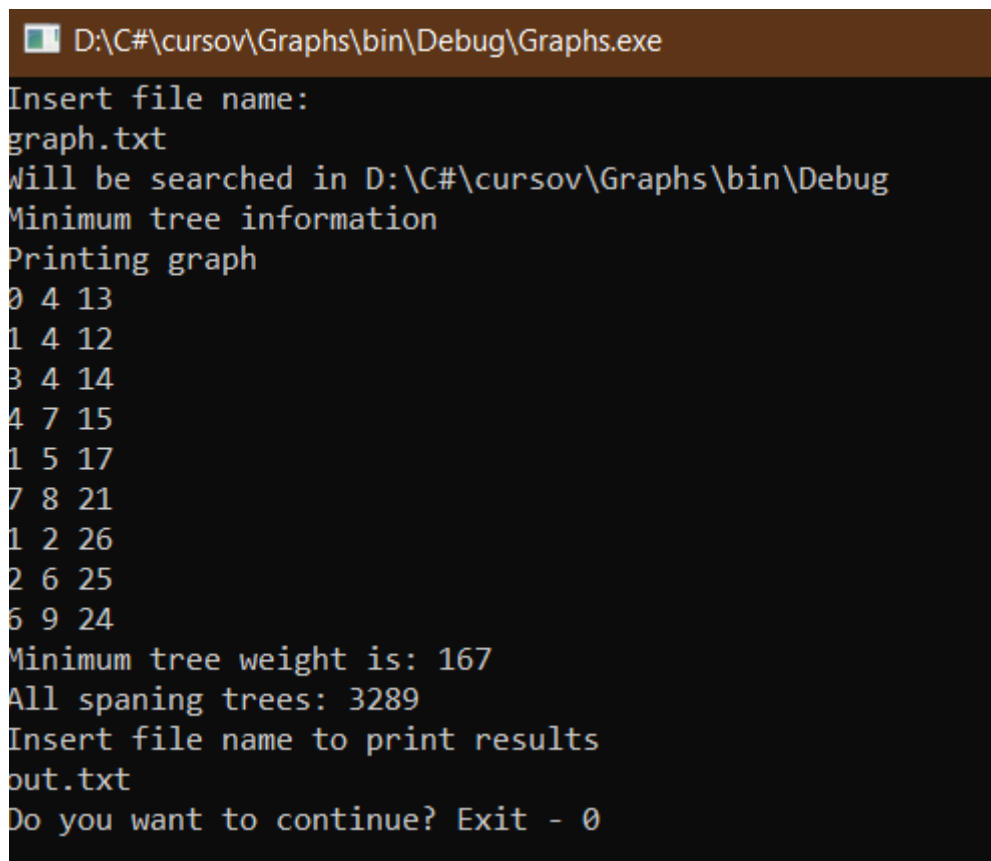
```
graph.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
10
0 1 18
0 4 13
1 2 26
1 4 12
1 5 17
2 5 29
2 6 25
3 4 14
3 7 20
4 7 15
4 8 27
5 6 28
5 8 22
5 9 27
6 9 24
7 8 21
8 9 29
```

Рисунок 2.4. Представление в файле

Как видно, в самом начале должно идти число – количество вершин графа. После чего следует список ребер, для каждого из которого должна быть выделена отдельная строка, на которой последовательно записано три числа: номера вершин (как было пронумеровано заранее) и вес ребра.

Запуск программы

При запуске программа просит ввести имя файла, из которого будет читаться граф. После ввода, если файл был найден, консоль отображает минимальное остовное дерево графа, его вес и общее количество остовов графа (рисунок 2.5). Далее программа просит ввести имя файла, в который выведутся результаты (файл не обязательно должен существовать).



```
D:\C#\cursov\Graphs\bin\Debug\Graphs.exe
Insert file name:
graph.txt
Will be searched in D:\C#\cursov\Graphs\bin\Debug
Minimum tree information
Printing graph
0 4 13
1 4 12
3 4 14
4 7 15
1 5 17
7 8 21
1 2 26
2 6 25
6 9 24
Minimum tree weight is: 167
All spanning trees: 3289
Insert file name to print results
out.txt
Do you want to continue? Exit - 0
```

Рисунок 2.5. Пользовательский интерфейс

Если файл не был найден в нужной директории, то программа выводит сообщение об этом (рисунок 2.6):

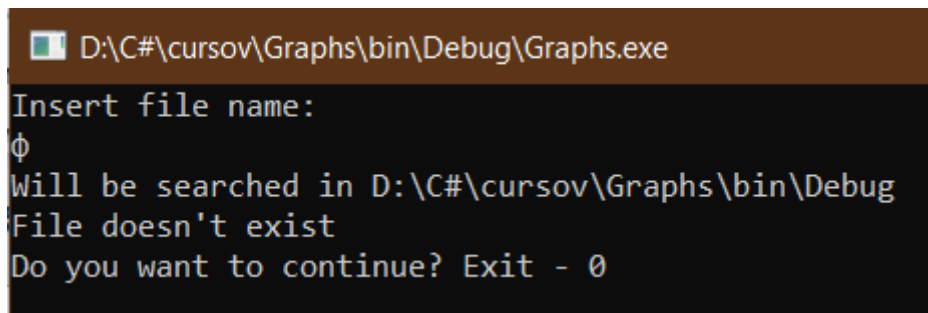


Рисунок 2.6. Ошибка нахождения файла

В конце работы программа спрашивает, хочет ли пользователь продолжить. Если был нажат 0, то программа заканчивает свою работу.

Руководство программиста

Запуск программы

Необходима операционная система Windows. При разработке использовалась Microsoft Visual Studio и платформа .NET Framework 4.7.2.

При запуске программа использует не более 10 Мб оперативной памяти. Приложение занимает в памяти 46 Кб. Весь проект Visual Studio занимает в памяти 280 Кб.

Модульные тесты

Данный вариант программы был протестирован с помощью модульных тестов C#[4]. На рисунке 2.7 приведен список тестов, как видно, они все выполнились верно.

✔ UnitTest1 (8)	79 мс
✔ EdgeCreate	4 мс
✔ GraphOstovNum	75 мс
✔ GraphPrimaAlg	< 1 мс
✔ GraphReadError1	< 1 мс
✔ GraphReadError2	< 1 мс
✔ GraphReadError3	< 1 мс
✔ MatrixDeterminantTest	< 1 мс
✔ MatrixMinorTest	< 1 мс

Рисунок 2.7. Модульные тесты

Описание тестов:

EdgeCreate – тестирует правильность добавления ребра (так, что в списке вершин меньшая по номеру идет раньше).

GraphOstovNum – тестирует правильность подсчета количества остовов графа

GraphPrimaAlg – тестирует правильность нахождения минимального остовного дерева.

GraphReadError1 – тестирует ошибку, выдаваемую, если заданный файл с графом не был найден.

GraphReadError2– тестирует ошибку, выдаваемую, если в файле не заданно количество вершин/заданно в неверном формате.

GraphReadError3– тестирует ошибку, выдаваемую, если в файле ошибка в списке ребер (неверный формат, то есть не три числа в строчке, лишние символы и т.д.)

MatrixDeterminantTest – тестирует правильность нахождения определителя классом матрицы.

MatrixMinorTest – тестирует правильность нахождения минора классом матрицы.

Контрольный пример

Для проверки работы программы возьмем тот же граф, что и в разделе «Руководство пользователя» – рисунок 2.3.

Начальной вершиной выбираем нулевую (по алгоритму это может быть любая, но для удобства программа всегда берет нулевую). Тогда наименьшее ребро, исходящее из вершины – 04 с весом 13. Так, теперь мы можем рассматривать все ребра, исходящих из вершин 0 и 4. По алгоритму выбираются ребра 14, 34, 47, 15. Теперь, хотя минимальное ребро – 01, мы его взять не можем, так как и вершина 0, и 1 уже были задействованы, значит нужно задействовать какое-либо другое. Единственное ребро, удовлетворяющее условию (то, что одна из вершин задействована, другая

– нет) – это ребро 78. Продолжая идти по алгоритму до тех пор, пока дерево не использует все вершины, после чего алгоритм закончится. В результате получим дерево, приведенное на рисунке 2.

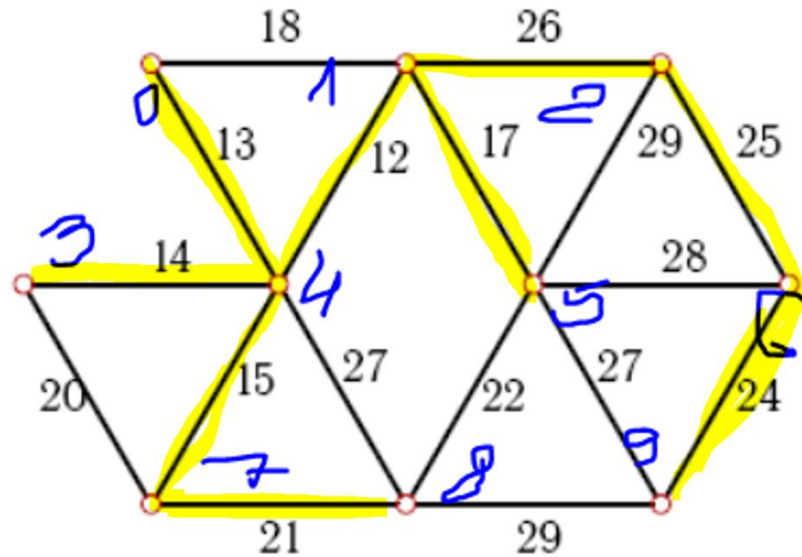


Рисунок 2.8. Минимальное остовное дерево

На рисунке представлен выходной файл для этого случая. В начале выводится матрица Кирхгофа и количество остонов, после чего выводится минимальный остов и его вес. При сравнении дерева на рисунке 2.8 и рисунке 2.9 видно, что программа работает корректно.

```
out.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
Kirchhoff matrix
2 -1 0 0 -1 0 0 0 0 0
-1 4 -1 0 -1 -1 0 0 0 0
0 -1 3 0 0 -1 -1 0 0 0
0 0 0 2 -1 0 0 -1 0 0
-1 -1 0 -1 5 0 0 -1 -1 0
0 -1 -1 0 0 5 -1 0 -1 -1
0 0 -1 0 0 -1 3 0 0 -1
0 0 0 -1 -1 0 0 3 -1 0
0 0 0 0 -1 -1 0 -1 4 -1
0 0 0 0 0 -1 -1 0 -1 3
All spanning trees: 3289

Minimum tree information
0 4 13
1 4 12
3 4 14
4 7 15
1 5 17
7 8 21
1 2 26
2 6 25
6 9 24
Minimum tree weight is: 167
```

Рисунок 2.9. Вывод в файл

Пример консольного вывода и отображения данных в приложении был приведен в «Руководстве пользователя» на рисунках 2.5 и 2.6.

Текст программы

Edges.cs

```
namespace Graphs
{
    public class Edge
    {
        //edge that is between vertex1 and vertex2
        public int vertex1 { get; }
        public int vertex2 { get; }
        public int weight { get; }
        public Edge(int v1, int v2, int weight)
```

```

    {
        //vertex1 is smaller then vertex2
        //for simplier use
        if (v1 < v2)
        {
            this.vertex1 = v1;
            this.vertex2 = v2;
        }
        else
        {
            this.vertex1 = v2;
            this.vertex2 = v1;
        }
        this.weight = weight;
    }
}
}

```

Matrix.cs

```

using System;

namespace Graphs
{
    public class Matrix
    {
        public int[,] matrix { get; }
        public Matrix(int[,] newMatrix)
        {
            matrix = newMatrix;
        }
        //creates empty
        public Matrix(int numberV)
        {
            matrix = new int[numberV, numberV];
        }

        //finds minor A[row][col]
        public void FindMinor(ref Matrix minor, int col, int row)
        {
            //col - column
            int n = (int)Math.Sqrt(minor.matrix.Length);
            int di = 0, dj;
            for (int ki = 0; ki < n; ki++)
            {
                //if we came to the row that needs to be 'deleted'
                //we pretend it doesn't exist and jump over it with di koeff
                if (ki == row) di = 1;
                dj = 0;

                for (int kj = 0; kj < n; kj++)
                {
                    //same for colum as for row
                    if (kj == col) dj = 1;
                    minor.matrix[ki, kj] = this.matrix[ki + di, kj + dj];
                }
            }
        }

        public double Determinant() // n - размерность матрицы A
        {
            //only for square matrix
            int n = (int)Math.Sqrt(matrix.Length);
            int k = 1; // minor koeff (+-1)

```



```

        double d = 0; // stores final determinant
        Matrix B = new Matrix(new int[n - 1, n - 1]);

        if (n < 1) return 0;
        if (n == 1) return matrix[0, 0];
        if (n == 2)
        {
            //a11*a22 - a12*a21 for 2x2 matrix
            return (matrix[0, 0] * matrix[1, 1] - matrix[0, 1] * matrix[1, 0]);
        }

        for (int i = 0; i < n; i++)
        {
            this.FindMinor(ref B, i, 0); //Find minors for first row
            d += k * matrix[0, i] * B.Determinant(); //count with a formula
            k *= -1;
        }
        return d;
    }
}
}

```

Graph.cs

```

using System;
using System.IO;
using System.Collections.Generic;

namespace Graphs
{
    public class Graph
    {
        int numberV; //number of vertices
        Matrix kirchhoffMatrix = null;
        List<Edge> edges = new List<Edge>();
        Graph ostovTree; //minimum ostov tree
        public int treeWeight { get; private set; }

        public Graph(string fileName)
        {
            //creating empty graph
            ostovTree = new Graph(0);
            //read from file
            string wpath = Directory.GetCurrentDirectory() + "\\\" + fileName;
            if (!File.Exists(wpath)) throw new Exception("File doesn't exist");

            FileStream stream = new FileStream(wpath, FileMode.Open);
            StreamReader reader = new StreamReader(stream);
            //file should start with number of vertices
            bool correct = int.TryParse(reader.ReadLine(), out numberV);
            if (!correct || numberV < 1) throw new Exception("Incorrect number of
vertices");

            while (!reader.EndOfStream)
            {
                //each line - new edge
                string newEdge = reader.ReadLine();
                newEdge = newEdge.Trim();
                //contains 3 parametrs splited with a space
                //two vertices and weight
                string[] bits = newEdge.Split(' ');
                int[] edgeParam = new int[3];
                //each parameter is an integer (should be)
            }
        }
    }
}

```

```

        for (int i = 0; i < 3; i++)
            if (!int.TryParse(bits[i], out edgeParam[i]))
                throw new Exception("Edge is incorrect");
        //successfully create a new edge
        edges.Add(new Edge(edgeParam[0], edgeParam[1], edgeParam[2]));
    }
    reader.Close();
    //write graph in kirchhoff matrix
    FindKirchhoff();
    //find tree with a minimum weight
    PrimaAlg();
}

public int OstovNum()
{
    //just in case
    if (kirchhoffMatrix == null) FindKirchhoff();
    if (numberV == 0) return 0;
    //find minor of kirchhoff matrix
    Matrix minor = new Matrix(new int[numberV - 1, numberV - 1]);
    kirchhoffMatrix.FindMinor(ref minor, numberV - 1, numberV - 1);
    //find determinant = number of ostov trees
    return (int)minor.Determinant();
}

public Graph OstovTree() { return ostovTree; } //to test
public Matrix GetKirchhoff() { return kirchhoffMatrix; }
public List<Edge> Edges() { return edges; }

void FindKirchhoff()
{
    int[,] kMatrix = new int[numberV, numberV];
    foreach (Edge edge in edges)
    {
        //on a diagonal a number of edges that go out from vertex with this number
        kMatrix[edge.vertex1, edge.vertex1]++;
        kMatrix[edge.vertex2, edge.vertex2]++;
        //adjacent vertices are marked -1 in matrix
        //matrix is symmetric
        kMatrix[edge.vertex1, edge.vertex2] = -1;
        kMatrix[edge.vertex2, edge.vertex1] = -1;
    }
    kirchhoffMatrix = new Matrix(kMatrix);
}
//to create empty graph
Graph(int numberV)
{
    this.numberV = numberV;
}
//to be able to change graph, but only inside
void AddEdge(Edge edge) { edges.Add(edge); }
//finds minimum tree
void PrimaAlg()
{
    treeWeight = 0;
    ostovTree = new Graph(this.numberV);
    //unused edges
    List<Edge> notUsedEdges = new List<Edge>(edges);
    //used vertices

```

```

List<int> usedV = new List<int>();
usedV.Add(0);
//unused vertices
List<int> notUsedV = new List<int>();
for (int i = 1; i < numberV; i++)
    notUsedV.Add(i);

while (notUsedV.Count > 0)
{
    int minE = -1; //number of a smallest edge
    //search for edge with min eright
    for (int i = 0; i < notUsedEdges.Count; i++)
    {
        if (((usedV.IndexOf(notUsedEdges[i].vertex1) != -1) &&
(notUsedV.IndexOf(notUsedEdges[i].vertex2) != -1)) ||
            ((usedV.IndexOf(notUsedEdges[i].vertex2) != -1) &&
(notUsedV.IndexOf(notUsedEdges[i].vertex1) != -1)))
        {
            if (minE != -1)
            {
                if (notUsedEdges[i].weight < notUsedEdges[minE].weight)
                    minE = i;
            }
            else
                minE = i;
        }
    }
    Edge minEdge = notUsedEdges[minE];
    //put new vertex in unsed and delete from unused
    if (usedV.IndexOf(minEdge.vertex1) != -1)
    {
        usedV.Add(minEdge.vertex2);
        notUsedV.Remove(minEdge.vertex2);
    }
    else
    {
        usedV.Add(minEdge.vertex1);
        notUsedV.Remove(minEdge.vertex1);
    }
    //put neew edge in tree and delete from unused
    treeWeight += minEdge.weight;
    ostovTree.AddEdge(minEdge);
    notUsedEdges.RemoveAt(minE);
}
}
}
}

```

Printer.cs

```

using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Graphs
{
    interface IPrinter
    {
        void PrintGraph(Graph graph);
    }
}

```

```

        void PrintMatrix(Matrix matrix);
    }
    class ConsolePrinter : IPrinter
    {
        public void PrintGraph(Graph graph)
        {
            //print initial graph on console
            Console.WriteLine("Printing graph");
            foreach (Edge edge in graph.Edges())
                Console.WriteLine(edge.vertex1.ToString() + " " + edge.vertex2.ToString() +
" " + edge.weight.ToString());
        }

        public void PrintTree(Graph graph)
        {
            //print tree in console
            Console.WriteLine("Minimum tree information");
            //minimum tree weight
            PrintGraph(graph.OstovTree());
            Console.WriteLine("Minimum tree weight is: " + graph.treeWeight);
        }

        public void PrintMatrix(Matrix mToPrint)
        {
            Console.WriteLine();
            int n = (int)Math.Sqrt(mToPrint.matrix.Length);
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                    Console.Write(mToPrint.matrix[i, j].ToString() + " ");
                Console.WriteLine();
            }
        }
    }
    class FilePrinter : IPrinter
    {
        string wpath;
        public FilePrinter(string fileName)
        {
            wpath = Directory.GetCurrentDirectory() + "\\\" + fileName;
        }

        public void PrintGraph(Graph graph)
        {
            PrintMatrix(graph.GetKirchhoff());
            using (StreamWriter sw = new StreamWriter(wpath, true,
System.Text.Encoding.Default))
            {
                //printing matrix + all spanning trees num
                sw.WriteLine("All spaning trees: " + graph.OstovNum());
                sw.WriteLine();
                //printing tree + weight
                sw.WriteLine("Minimum tree information");
                int treeWeight = 0;
                foreach (Edge edge in graph.OstovTree().Edges())
                {
                    sw.WriteLine(edge.vertex1.ToString() + " " + edge.vertex2.ToString() +
" " + edge.weight.ToString());
                    treeWeight += edge.weight;
                }
                sw.WriteLine("Minimum tree weight is: " + treeWeight.ToString());
            }
        }
    }

```

```

    }

    public void PrintMatrix(Matrix mToPrint)
    {
        int n = (int)Math.Sqrt(mToPrint.matrix.Length);
        using (StreamWriter sw = new StreamWriter(wpath, true,
System.Text.Encoding.Default))
        {
            sw.WriteLine("Kirchhoff matrix");
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    sw.Write(mToPrint.matrix[i, j].ToString() + " ");
                }
                sw.WriteLine();
            }
        }
    }
}

```

Program.cs

```

using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Graphs
{
    class Program
    {
        static void Main(string[] args)
        {
            string flag = "1";
            while (flag != "0")
            {
                Console.WriteLine("Insert file name: ");
                string fileName = Console.ReadLine();
                //fileName = "graph.txt";

                Console.WriteLine("Will be searched in " + Directory.GetCurrentDirectory());
                Graph smth;
                try
                {
                    smth = new Graph(fileName);
                    ConsolePrinter cPrint = new ConsolePrinter();
                    cPrint.PrintTree(smth);
                    Console.WriteLine("All spanning trees: " + smth.OstovNum());
                    Console.WriteLine("Insert file name to print results");
                    fileName = Console.ReadLine();
                    FilePrinter fPrint = new FilePrinter(fileName);
                    fPrint.PrintGraph(smth);
                }
                catch (Exception ex) { Console.WriteLine(ex.Message); }

                Console.WriteLine("Do you want to continue? Exit - 0");
                flag = Console.ReadLine();
            }
        }
    }
}

```

```

    }
}

```

UnitTest1.cs

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Graphs;
using System;

namespace TestGraph
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void EdgeCreate()
        {
            Edge edge = new Edge(2, 1, 13); //create edge between vertex 1 and 2
            Assert.AreEqual(edge.vertex1, 1); //smaller go first - true
            Assert.AreEqual(edge.vertex2, 2);
            Assert.AreEqual(edge.weight, 13);
        }

        [TestMethod]
        public void MatrixMinorTest()
        {
            int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
            Matrix testM = new Matrix(arr);
            Matrix minorA33 = new Matrix(2); ; testM.FindMinor(ref minorA33, 2, 2);
            Assert.AreEqual(minorA33.matrix[0, 0], 1);
            Assert.AreEqual(minorA33.matrix[0, 1], 2);
            Assert.AreEqual(minorA33.matrix[1, 0], 4);
            Assert.AreEqual(minorA33.matrix[1, 1], 5);
        }

        [TestMethod]
        public void MatrixDeterminantTest()
        {
            int[,] arr = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
            Matrix testM = new Matrix(arr);
            Assert.AreEqual(testM.Determinant(), 0);
        }

        [TestMethod]
        public void GraphReadError1()
        {
            Graph testG;
            try
            {
                testG = new Graph("a");
            }
            catch (Exception ex) { Assert.AreEqual(ex.Message, "File doesn't exist"); }
        }

        [TestMethod]
        public void GraphReadError2()
        {
            Graph testG;
            try
            {
                testG = new Graph("graphError.txt");
            }
        }
    }
}

```

```

        catch (Exception ex) { Assert.AreEqual(ex.Message, "Incorrect number of
vertices"); }
    }

    [TestMethod]
    public void GraphReadError3()
    {
        Graph testG;
        try
        {
            testG = new Graph("graphError3.txt");
        }
        catch (Exception ex) { Assert.AreEqual(ex.Message, "Edge is incorrect"); }
    }

    [TestMethod]
    public void GraphOstovNum()
    {
        Graph testG;
        testG = new Graph("graph.txt");
        Assert.AreEqual(testG.OstovNum(), 3289);
    }

    [TestMethod]
    public void GraphPrimaAlg()
    {
        Graph testG;
        testG = new Graph("graph.txt");
        Graph testTree = testG.OstovTree();
        Assert.AreEqual(9, testTree.Edges().Count);
    }
}

```

3 ТРЕТИЙ РАЗДЕЛ

Задание

Вариант 6. Имитация работы автоматизированного участка цепи

В цехе производится обработка деталей на 3 различных станках с числовым программным управлением (каждый станок настроен на выполнение определённых автоматических операций). Часть деталей обрабатывается только на 1 станке, часть на 2 станках (последовательно), часть на всех 3 станках. Порядок обработки деталей на 1 или на 2 станках не имеет значения, но на третьем станке деталь должна быть обработана только после того, как будет обработана на первых двух станках. Время поступления заготовки через каждые $A \pm B$ минут, время обработки на каждом станке $C_k \pm D_k$ минут, где k — номер станка. Задать времена поступления заготовок и обработки и на соответствующих станках, а также доли деталей, которые обрабатываются только на первом станке, на 2-х станках и на 3-х станках. Промоделировать работу автоматизированного участка цеха в течение 8 часов. Исследовать процесс образования очередей при перемещении заготовки между станками.

Теоретический аспект

Имитационное моделирование — метод, позволяющий строить модели, описывающие процессы так, как они проходили бы в действительности. Такую модель можно «проиграть» во времени как для одного испытания, так и заданного их множества. При этом результаты будут определяться случайным характером процессов [5].

В данном случае применяется дискретно-событийная модель моделирования, то есть функционирование системы представляется как последовательность событий. Событие происходит в определенный момент и означает изменение состояния системы [6].

Основными компонентами дискретно-событийной модели моделирования являются:

- Часы: синхронизируют события. В данном случае это внутреннее время модели, пропорциональное реальному времени.
- Список событий. В задаче – обработка деталей.
- Генератор случайных чисел. В данном задании случайно генерируется время поступления деталей, время обработки детали на станке (в определенном промежутке, установленном пользователем).
- Статистика.
- Условие завершения. Достижение времени работы, установленного пользователем (8 часов по заданию).

Формализация задачи

Структура программы

В структуру программы (рисунок 3.1) входят несколько классов: абстрактный Detail – непосредственно сама деталь, Machine – станок, который обрабатывает различные детали и Factory – класс, в котором контролируется длительность рабочего дня и сами процессы обработки. В классе ControlQueue происходит контроль очередей (отдельный класс, чтобы была возможность применять разные способы образования очередей в будущем, если такая потребность будет), а класс Statistic происходит сбор статистики о текущей работе фабрики в любой момент. На фабрике создаются три машины (три экземпляра класса Machine), каждая из которых отвечает за отдельную стадию обработки. Единственное ограничение – третья машина не сможет работать, если деталь не прошла обработку на первых двух, данный факт учитывается при контроле очередей, то есть необработанная на первых двух станках деталь не сможет попасть на обработку на третий станок. Также на фабрике имеются параметры $A \pm B$ – это время, через которое поступает заготовка. Метод Work запускает имитацию одной итерации работы, до поступления заготовки.

Перечисление enum задает три типа деталей: A, B, C. В классе Deatil хранится массив states – количество обработок, которое прошла деталь. В зависимости от типа детали массив может быть длины 1, 2 и 3 (определяется при создании детали). Метод ProcessState обрабатывает определенное состояние, меняя соответствующее значение массива на true, а метод IfReady проверяет, готова ли деталь.

В классе Machine имеются поля $C \pm D$ – время работы станка, type – какие детали обрабатывает станок, detailsToProcess – очередь на обработку. Метод Process непосредственно обрабатывает деталь, AddDetail добавляет деталь в очередь.

На фабрике предусмотрены переработки, но на время, не большее, чем ошибка времени при поступлении новых деталей (параметр B на фабрике), который находится в диапазоне от 0 до 10 минут. На каждой итерации программа проверяет, больше ли текущее время работы, чем планируемое время работы плюс минимальное время следующей итерации (нижняя граница, параметр A - B). Таким образом, фабрика не заканчивает работу слишком рано, но и не перерабатывает чрезмерно.

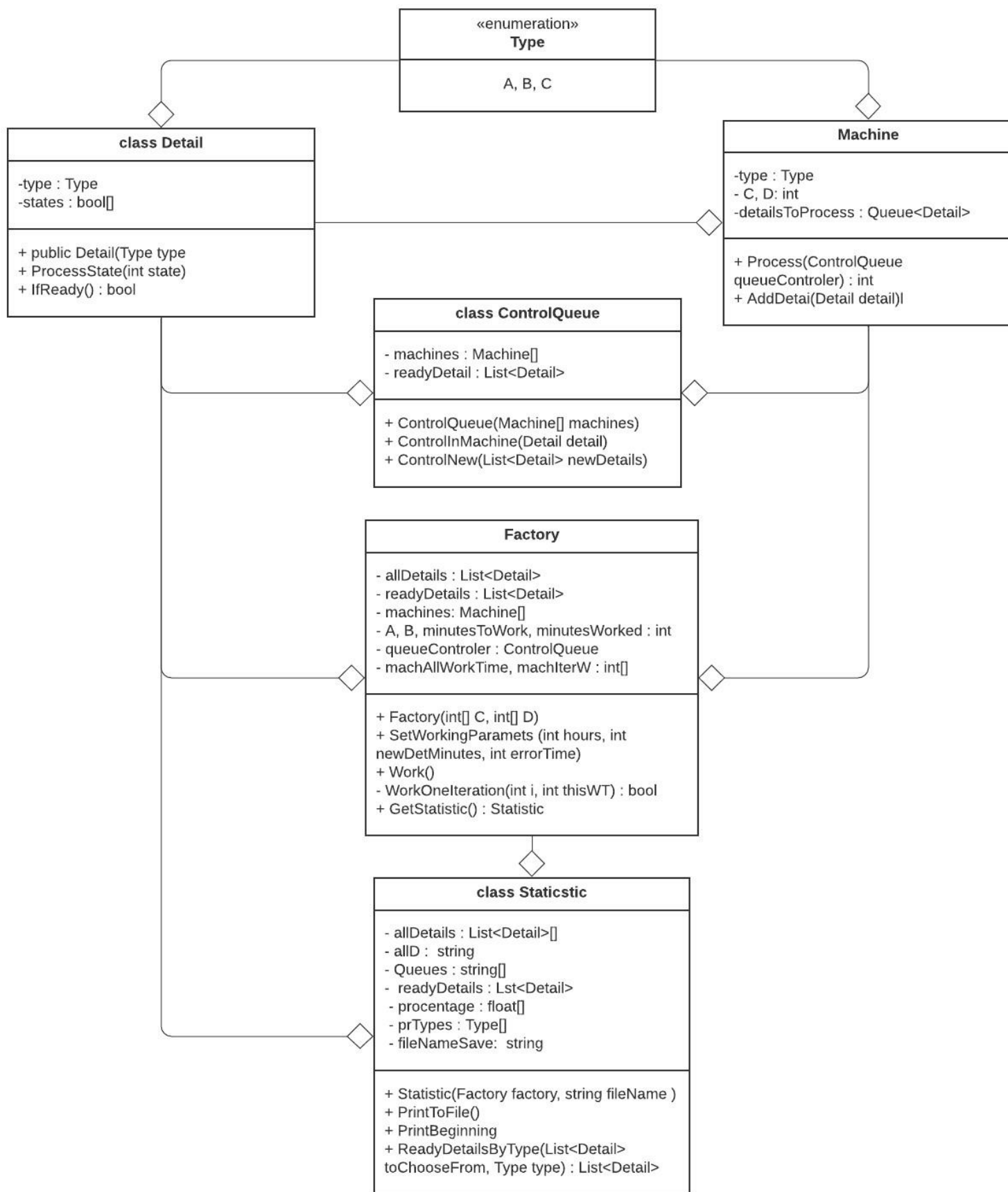


Рисунок 3.1. Диаграмма классов

Идея моделирования самой работы фабрики заключается в том, что каждую итерацию машина проверяет, есть ли у нее время, чтобы обработать еще одну деталь (до конца итерации). Если после обработки время превысило то время, которое должно быть достигнуто в конце одной итерации, то машина прекращает работу. Если время еще есть, то обрабатывается очередная деталь в очереди. В конце каждой итерации выводится статистика. Прохождение одной итерации представлено на рисунке 3.2:

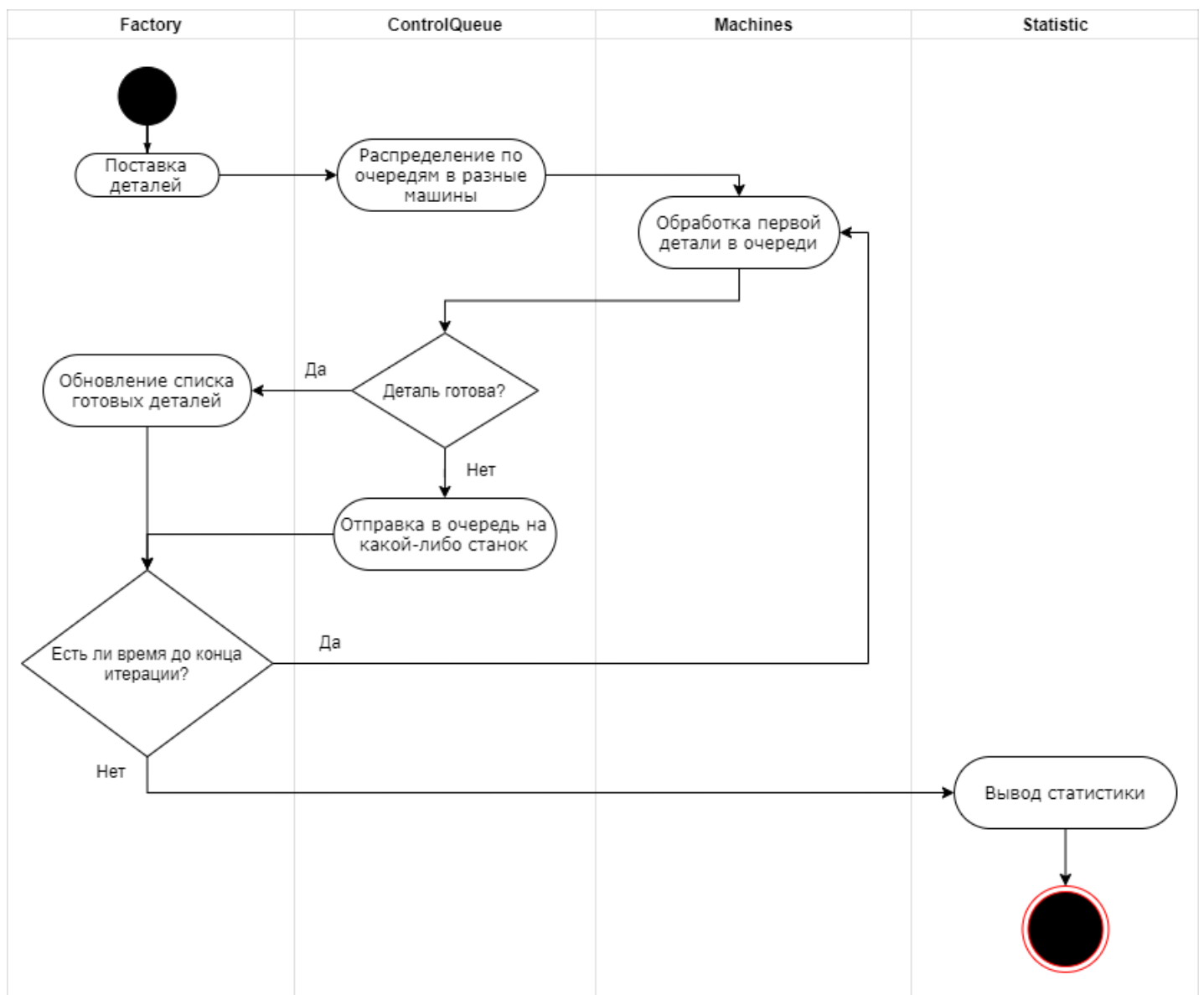


Рисунок 3.2. Диаграмма состояний

Образование очередей

За данный процесс отвечает класс `ControlQueue`. Деталь типа А обрабатывается только с помощью первой машины (если на фабрику попадает заготовка для детали А, то она просто отправляется в очередь на обработку первым станком, так как других вариантов нет). Детали В и С отправляются в зависимости от занятости либо на первый станок, либо на второй. Если очередь на первый больше (считается по среднему времени работы машины, умноженное на количество элементов в очереди), то деталь отправляется на второй, и потом на первый. Деталь С после двух обработок отправляется на третий станок, где заканчивает обработку. Порядок обработки, который может пройти деталь, показан на рисунке 3.3. Цвета стрелок на рисунке формальны, показывающие, что в начале обработки есть только два варианта направления:

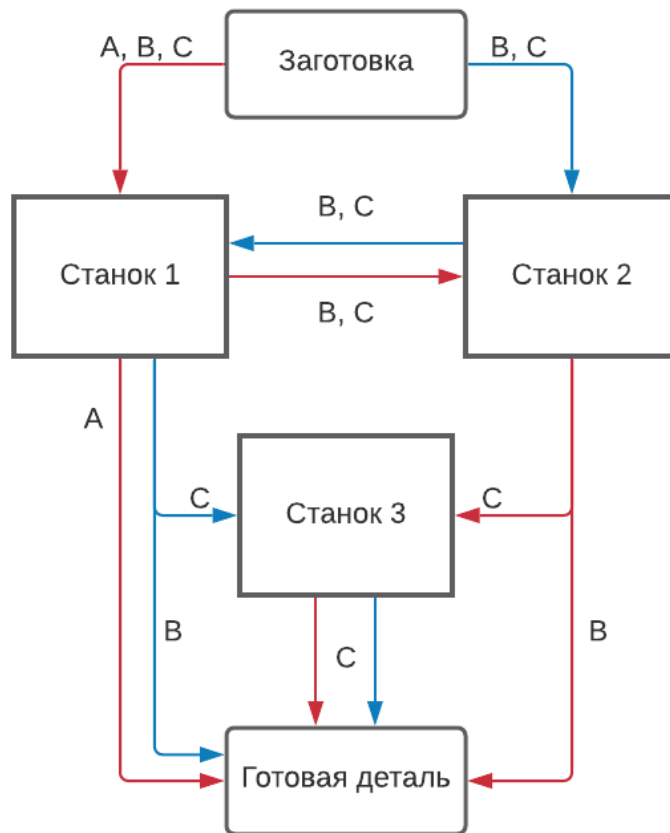


Рисунок 3.3. Порядок обработки деталей

При получении заготовки метод ControlNew определяет, на какой из станков отправится та или иная деталь, а после обработки на любом из станков вызывается метод ControlInMachine, который проверяет готова ли деталь, и если нет, то отправляет на один из станков.

Сбор статистики

В конце работы одной итерации на фабрике экземпляр класса Statistic с помощью LINQ [7] (исходя из данных о готовых деталях) подсчитывает статистику и записывает в файл. Класс статистики включает в себя список всех деталей, список готовых деталей и массив очередей в каждой машине, а также имя файла, в который производится вывод информации. Для всех деталей подсчитывается их количество и сортируется по деталям, для готовых деталей производится тоже самое, но считается процент деталей от всех готовых (для реализации разных способов отображения информации).

Принципы объектно-ориентированного программирования

В программе применен принцип единственной ответственности [8], так как каждый из классов выполняет только одну задачу: Machine обрабатывает, ControlQueue контролирует очередь (они контролируются не с помощью Factory, а отдельным классом), Factory только имитирует работу (не выводит ничего в файл и не сохраняет никаких данных), а Statistic выводит информацию в файл.

Спецификация

class Detail

Поле	Назначение
type	Тип детали
states	Состояние детали, которые нужно обработать
Метод	Назначение
Detail	Создает экземпляр класса

Метод	Назначение
IfReady	Проверяет, готова ли деталь
ProcessState	Обрабатывает заданное состояние детали

class Machine

Поле	Назначение
type	Тип машины
C	Время, за которое производится обработка
D	Погрешность времени
detailsToProcess	Очередь деталей на обработку
Метод	Назначение
Machine	Создает экземпляр класса (машину заданного типа)
AddDetail	Добавляет деталь в очередь
Process	Обрабатывает деталь

class ControlQueue

Поле	Назначение
machines	Список машин, между которыми происходит контроль очередей
readyDetail	Список готовых деталей
Метод	Назначение
ControlQueue	Создает экземпляр класса
ControlInMachine	Контролирует очередь в машинах, распределяет детали в новую очередь после обработки на одной из машин.
ControlNew	Контролирует очередь новых деталей, список деталей распределяет по очередям

class Factory

Поле	Назначение
A	Время, которое работает фабрика
B	Погрешность времени
minutesToWork	Время, которое нужно проработать в минутах
minutesWorked	Проработанное время
machAllWorkTime	Полное время работы каждой из машин
machIterWT	Время работы каждой из машин на текущей итерации
queueController	Контролер очередей
machines	Список машин
allDetails	Список всех деталей
readyDetails	Список готовых деталей
Метод	Назначение
Factory	Создает экземпляр класса (фабрику с заданными параметрами)
AddToReady	Добавляет деталь в список готовых
SetWorkingParamets	Позволяет устанавливать новые параметры для работы фабрики
Work	Работа одной итерации (до следующего поступления заготовок)
WorkOneIteration	Вызывается из метода Work, работа на итерации для одной из машин (любой)

class Statistic

Поле	Назначение
allDetails	Массив списков всех деталей, отсортированных по типу
allID	Все детали в порядке поступления заготовок

Поле	Назначение
Queues	Все детали в очередях каждой машины
readyDetails	Список всех готовых деталей
procentage	Процент всех деталей по типу
prTypes	Соответствие проценту типу
fileNameSave	Имя файла, в которые записываются данные на каждой итерации
Метод	Назначение
Statitic	Создает экземпляр класса статистики
PrintToFile	Печатает текущую информацию в файл
PrintBeginnig	Печатает индикатор начала работы в файл
ReadyDetailsByType	Возвращает список готовых деталей определенного типа

Руководство пользователя

Запуск программы

Необходима операционная система Windows. Приложение занимает в памяти 96 Кб.

Выполняемая задача

Приложение имитирует работу фабрики в течение установленного пользователем времени и установленными пользователем параметрами, подробнее о который написано в следующем пункте. Принцип работы фабрики описан в пунктах «Здание» и «Формализация задачи».

Ввод данных

Программа представлена в виде оконного приложения, интерфейс которого показан на рисунке 3.4. В левой части представлены поля ввода, в правой – поля вывода. Параметр «Time to work, hours» определяет, сколько часов будет идти имитация (минимум – 1, максимум – 15, с шагом в 1 час).

Параметр «Receiving blank details, minutes» определяет, через сколько минут подаются новые детали, плюс минус погрешность (Первое можно изменять от 30 до 60 минут с шагом 5, второй – от 0 до 9 минут с шагом 1). Иначе этот параметр можно назвать переменным шагом моделирования. Каждый шаг обновляются данные в окошках справа.

Далее нужно установить время работы для машин («Machines Process time, minutes»), последовательно для первой, второй и третьей. На каждой машине первый параметр меняется от 10 до 60 минут с погрешностью (второй параметр) от 0 до 9 минут.

Благодаря установленным ограничениям программа не сможет выдать отрицательное время и будет работать корректно.

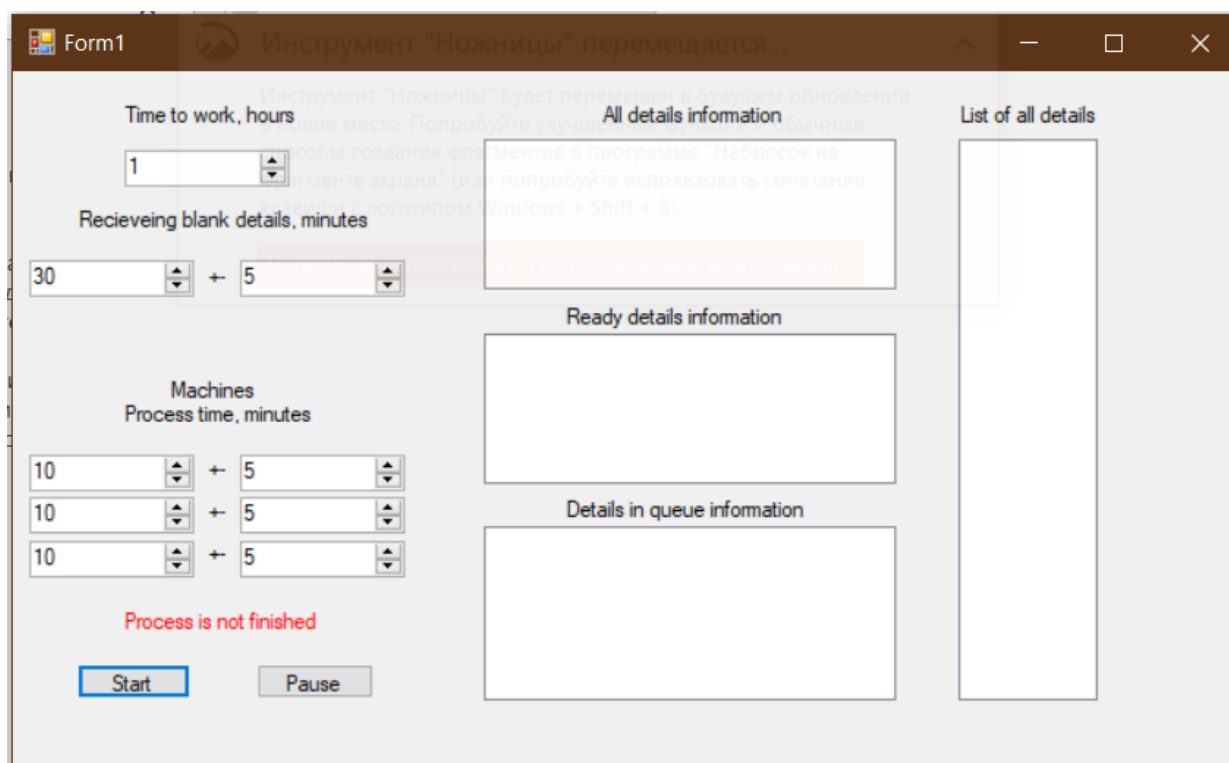


Рисунок 3.4. Пользовательский интерфейс

Начало работы

Для начала работы, после заполнения полей, нужно нажать кнопку «Start». Если параметры не установлены пользователем, то программа запускается на автоматических параметрах (они отображаются на экране).

Кнопка «Pause» до начала работы неактивна. Если нажать кнопку «Start» во время работы фабрики, то работа начнется заново с новыми параметрами (если они были изменены).

Отображение данных

Пример отображения данных в середине работы (после двух итераций или после 54 минут работы) при нажатии паузы представлен на рисунке 3.5. В окне «All details information» отображается информация обо всех деталях, поступивших на фабрику за время работы, а в окне «List of all details» представлен список этих деталей в порядке поступления (от первой до последней). В окне «Ready Details information» показано текущее время работы, количество готовых деталей всего и количество готовых деталей в процентах. «Details in queue information» предоставляет актуальную информацию об очередях в каждой машине (первое в очереди, первое в списке). Красный индикатор над кнопками старт и стоп показывает, закончено ли моделирование процесса или нет.

Рисунок 3.5. Отображения статистики

Вывод в файл

Также информация о каждой итерации выводится в файл log.txt в папке bin/Debug.

Руководство программиста

Запуск программы

Необходима операционная система Windows. При разработке программы использовалась Microsoft Visual Studio и платформа .NET Framework 4.7.2, а также конструктор Windows Forms.

При запуске программа использует не более 17 Мб оперативной памяти. Приложение занимает в памяти 96 Кб. Весь проект Visual Studio занимает в памяти 388 Кб.

Использование классов отдельно от Windows Forms

Так как требовалось отображать данные на каждой итерации, метод Work в классе Factory должен вызываться несколько раз, до тех пор, пока не будет возвращено значение -1, обозначающее конец работы. В остальных случаях метод вернет время, которое заняла данная итерация, в минутах.

При создании экземпляра класса Machine программа может выдать ошибку, если произойдет попытка установить время работы меньше, чем 0. В данном примере благодаря тому, что пользователь в интерфейсе Windows Forms может устанавливать только заданные значения, это не произойдет.

Будущая доработка

В данный момент программа может реализовывать только случайную генерацию поставляемых деталей, однако предполагается, что при доработке программы список заготовок будет приниматься из какого-либо файла. Также пользователь не может ввести из графического интерфейса имя файла для вывода данных, хотя данная возможность предусмотрена классом Statistic, хотя она и не обрабатывает исключения.

Контрольный пример

На рисунках 3.6 – 3.8 представлен пример процесса имитации работы фабрики в течение 8-ми часов, с временем подачи заготовок 30 ± 5 минут, и с одинаковым временем обработки для каждой машины: 10 ± 5 минут.

The screenshot shows a software window titled "Form1" with a light gray background and a dark brown title bar. The window contains several sections for simulation parameters and real-time data:

- Time to work, hours:** A numeric input field with the value "8".
- Recieveing blank details, minutes:** Two numeric input fields with values "30" and "5", separated by a "+" sign.
- Machines Process time, minutes:** Three identical pairs of numeric input fields, each with values "10" and "5" separated by a "+" sign.
- Status:** A red text label "Process is not finished" and two buttons labeled "Start" and "Continue".
- All details information:** A text box displaying:
All details: 10
Details of type A : 5
Details of type B : 2
Details of type C : 3
- Ready details information:** A text box displaying:
Time: 85
Ready details: 7
57,14286% of type A
14,28571% of type B
28,57143% of type C
- Details in queue information:** A text box displaying:
Machine 1
Queue: A C B
Machine 2
Queue:
Machine 3
Queue:
- List of all details:** A vertical text box displaying the sequence:
A B A A
C A C C
B A

Рисунок 3.6. Первый пример

Form1

Time to work, hours
8

Recieveing blank details, minutes
30 + 5

Machines
Process time, minutes
10 + 5
10 + 5
10 + 5

Process is not finished

Start Continue

All details information

All details: 17
Details of type A : 8
Details of type B : 5
Details of type C : 4

Ready details information

Time: 183
Ready details: 16
50% of type A
31,25% of type B
18,75% of type C

Details in queue information

Machine 1
Queue: C
Machine 2
Queue:
Machine 3
Queue:

List of all details

A B A A
C A C C
B A B B
B A C A
A

Рисунок 3.7. Второй пример

Form1

Time to work, hours
8

Recieveing blank details, minutes
30 + 5

Machines
Process time, minutes
10 + 5
10 + 5
10 + 5

Process is not finished

Start Continue

All details information

All details: 27
Details of type A : 11
Details of type B : 9
Details of type C : 7

Ready details information

Time: 358
Ready details: 24
45,83333% of type A
29,16667% of type B
25% of type C

Details in queue information

Machine 1
Queue: B B
Machine 2
Queue: C
Machine 3
Queue:

List of all details

A B A A
C A C C
B A B B
B A C A
A A B C
B C A A
B C B

Рисунок 3.8. Третий пример

На рисунке 3.9 представлен итоговый вывод после завершения работы (красный индикатор показывает «Process is finished»):

Рисунок 3.9. Результат

На рисунке 3.10 показан пример вывода одной из итераций в файл. Данные в файле такие же, как и в приложении (рисунок 3.7):

```

Debuglog.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
All details: 27
Details of type A : 11
Details of type B : 9
Details of type C : 7
Machine 1
    Queue: B B
Machine 2
    Queue: C
Machine 3
    Queue:
Ready details: 24
45,83333% of type A
29,16667% of type B
25% of type C
-----

```

Рисунок 3.10. Вывод в файл

Если при создании класса `Statistic` не устанавливается собственное имя файла, то новый результат дозаписывается к данным в файле `log.txt`.

Текст программы

Detail.cs

```
using System;

namespace Modeling
{
    enum Type { A = 0, B = 1, C = 2}
    //detail types
    //A is processed only on 1st machine
    //B is processed on 1st and 2nd
    //C is processed on 1st, 2nd and 3rd
    class Detail
    {
        public Type type { get; }
        public bool[] states { get; } //states that can be processed

        public Detail(Type type)
        {
            this.type = type;
            states = new bool[(int)type + 1];
        }
        public bool IfReady() //checks if detail is ready
        {
            foreach (bool st in states)
                if (st == false) return false;
            return true;
        }

        public void ProcessState(Type stateNum)
        {
            //stateNum - what state should be processed
            if (stateNum == Type.C && (states[0] == false || states[1] == false)) throw new
Exception("The detail haven't been processed in prev steps");
            states[(int)stateNum] = true;
        }
    }
}
```

Machine.cs

```
using System;
using System.Collections.Generic;

namespace Modeling
{
    class Machine
    {
        Type type;
        //time to process is C+-D
        public int C { get; }
        int D; //minutes
        public Queue<Detail> detailsToProcess { get; }

        public Machine(Type typeM, int C, int D)
        {
            if (C < 0) throw new Exception("Cannot create machine");
        }
    }
}
```



```

        this.type = typeM;
        this.C = C;
        this.D = D;
        detailsToProcess = new Queue<Detail>();
    }
    public void AddDetail(Detail detail) //adding smth to queue
    {
        detailsToProcess.Enqueue(detail);
    }
    public int Process(ControlQueue queueController)//returns time that took to process
    {
        if (detailsToProcess.Count == 0) return 0;

        //processing
        Detail detailToProcess = detailsToProcess.Dequeue();
        detailToProcess.ProcessState(type);
        //controlling queue
        queueController.ControlInMachine(detailToProcess);

        Random rand = new Random();
        return rand.Next(C - D, C + D); //random work time
    }
}
}

```

ControlQueue.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Modeling
{
    class ControlQueue
    {
        //for specific working pattern (queues creations)
        public Machine[] machines { get; private set; }
        public List<Detail> readyDetail { get; private set; }

        public ControlQueue(Machine[] machines)
        {
            readyDetail = new List<Detail>();
            this.machines = machines;
        }
        //if details comes from machine
        public void ControlInMachine(Detail detail)
        {
            //if ready (all A details will go here)
            if (detail.IfReady()) { readyDetail.Add(detail); }
            //if it's type C and it was processed on 1st and 2nd machines
            else if (detail.type == Type.C && detail.states[0] && detail.states[1])
                machines[2].AddDetail(detail);
            //if detail B was processed on 1st, then we send it on 2nd
            else if (detail.states[0]) machines[1].AddDetail(detail);
            else machines[0].AddDetail(detail);
        }
        //if details have just arrived
        public void ControlNew(List<Detail> newDetails)
        {
            foreach (Detail detail in newDetails)
            {

```

```

        if (detail.type == Type.A) machines[0].AddDetail(detail);
        else
        {
            //check where queue is shorter (in minutes)
            if (machines[0].detailsToProcess.Count * machines[0].C >
machines[1].detailsToProcess.Count * machines[1].C)
                machines[1].AddDetail(detail);
            else machines[0].AddDetail(detail);
        }
    }
}
}
}

```

Factory.cs

```

using System;
using System.Collections.Generic;

namespace Modeling
{
    class Factory
    {
        int A, B; //minutes
        //each A +- B minutes factory gets new details
        int minutesToWork = 8 * 60; //standart to work - 8 hours
        public int minutesWorked { get; private set; } //worked currently
        int[] machAllWorkTime = new int[3]; //current combined work time for each machine
        int[] machIterWT = new int[3]; //work time on each iteration
        ControlQueue queueController;
        public Machine[] machines { get; } //three machines

        //lists for further statistic
        public List<Detail> allDetails { get; private set; }
        public List<Detail> readyDetail { get; private set; }
        public void AddToReady(Detail value) { readyDetail.Add(value); }
        public Factory(int[] C, int[] D)
        {
            //ControlQueue.CleanInfo();
            allDetails = new List<Detail>();
            minutesWorked = 0;
            //create 3 machines
            machines = new Machine[3];
            for (int i = 0; i < 3; i++)
            {
                Type types = (Type)Enum.GetValues(typeof(Type)).GetValue(i);
                machines[i] = new Machine(types, C[i], D[i]);
            }
            //creating a controller
            queueController = new ControlQueue(machines);
        }

        public void SetWorkingParamets(int hoursToWork = 8, int newDetTimeMinutes = 15, int
errorTime = 2)
        {
            machAllWorkTime = new int[3];
            machIterWT = new int[3];
            minutesToWork = hoursToWork * 60;
            minutesWorked = 0;
            this.A = newDetTimeMinutes;
            this.B = errorTime;
            allDetails = new List<Detail>();
            queueController = new ControlQueue(machines);
        }
    }
}

```

```

    }

    public int Work() //one iteration
    {
        //generating random number of random details
        Random rand = new Random();
        List<Detail> newDetails = new List<Detail>();
        int n = rand.Next(1, 4);
        for (int i = 0; i < n; i++)
        {
            Detail newD = new
Detail((Type)Enum.GetValues(typeof(Type)).GetValue(rand.Next(0, 3)));
            newDetails.Add(newD);
            allDetails.Add(newD);
        }

        //putting them in a queue
        queueController.ControlNew(newDetails);

        //this iter working time
        //(aka next step will be in)
        int thisWT = rand.Next(A - B, A + B);
        //process all that can be processed in this time
        bool NeedAnotherTry = WorkOneIteration(0, thisWT);
        WorkOneIteration(1, thisWT);
        if (NeedAnotherTry) WorkOneIteration(0, thisWT);
        WorkOneIteration(2, thisWT);

        minutesWorked += thisWT;
        readyDetail = queueController.readyDetail;
        //finishing work condition
        if ((minutesWorked+A-B) > minutesToWork) return -1;
        return thisWT;
    }

    //additional to better read the code
    bool WorkOneIteration(int i, int thisWT)
    {
        //returns true if machine is wmpy at some piont of iteration
        //(there are no more details in queue
        int tryP = 0;
        while (true)
        {
            if (machIterWT[i] < (thisWT))
            {
                tryP = machines[i].Process(queueController);
                if (tryP == 0) { machAllWorkTime[i] += (thisWT); machIterWT[i] = 0;
return true; } //if there are no more details in queue
                else if ((machIterWT[i] + tryP) >= thisWT) { machIterWT[i] =
(machIterWT[i] + tryP) - thisWT; machAllWorkTime[i] += thisWT; break; } //if the iteration
ends
                else { machIterWT[i] += tryP; /* machAllWorkTime[i] += thisWT; */ }
            }
            else
            {
                machAllWorkTime[i] += thisWT;
                machIterWT[i] -= thisWT; break;
            }
        }
        return false;
    }
}

```

```
}
```

Statistic.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Text;
using System.Threading.Tasks;

namespace Modeling
{
    class Statistic
    {
        public List<Detail>[] allDetails { get; } //sorted in array by type
        public string allD { get; private set; } //same but in string format
        public string[] Queues { get; } //for each machine
        public List<Detail> readyDetails { get; } //just all details
        public float[] procentage { get; private set; } //of ready details
        public Type[] prTypes { get; private set; }
        string fileNameSave;

        public Statistic(Factory factory, string fileName = "log.txt")
        {
            fileNameSave = fileName;
            int i = 0;
            readyDetails = factory.readyDetail;
            procentage = new float[3];
            prTypes = new Type[3];
            Queues = new string[3];
            allDetails = new List<Detail>[3];
            List<Detail>[] detailsInQueue = new List<Detail>[3];

            //getting info about ready details
            var readyD = readyDetails.GroupBy(x => x.type).OrderBy(group =>
group.Key).Select(y => new { count = y.Count(), detType = y.Key });
            foreach (var elem in readyD)
            {
                if (readyDetails.Count != 0) procentage[i] = (float)elem.count /
readyDetails.Count * 100;
                else procentage[i] = 0;
                prTypes[i] = elem.detType; i++;
            }
            //getting info about details in queue (for each machine)
            Machine[] m = factory.machines;
            for (i = 0; i < 3; i++)
            {
                detailsInQueue[i] = m[i].detailsToProcess.ToList();
                foreach (Detail det in detailsInQueue[i])
                    Queues[i] += det.type.ToString() + " ";
            }
            //getting overall info (ab every detail)
            foreach (Detail det in factory.allDetails)
                allD += det.type + " ";
            for (i = 0; i < 3; i++)
                allDetails[i] = ReadyDetailsByType(factory.allDetails,
(Type)Enum.GetValues(typeof(Type)).GetValue(i));
        }

        //to print in log file
        public void PrintToFile()
        {

```

```

        string writePath = Directory.GetCurrentDirectory() + @"\" + fileNameSave;
        try
        {
            using (StreamWriter sw = new StreamWriter(writePath, true,
Encoding.Default))
            {
                //sw.WriteLine("Time: " + factory.minutesWorked.ToString());
                //all details info
                sw.WriteLine("All details: " + allD.Length / 2);
                for (int i = 0; i < 3; i++)
                    sw.WriteLine("Details of type " +
Enum.GetValues(typeof(Type)).GetValue(i).ToString() + " : " + allDetails[i].Count);
                //queues in machines current
                for (int i = 0; i < 3; i++)
                {
                    sw.WriteLine("Machine " + (i + 1));
                    sw.WriteLine("    Queue: " + Queues[i]);
                }

                sw.WriteLine("Ready details: " + readyDetails.Count);
                for (int i = 0; i < 3; i++)
                    sw.WriteLine(procentage[i].ToString() + "% of type " +
prTypes[i].ToString());
                sw.WriteLine("-----");
            }
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
        }
    }

    public void PrintBeginning()
    {
        string writePath = Directory.GetCurrentDirectory() + @"\" + fileNameSave;
        using (StreamWriter sw = new StreamWriter(writePath, true, Encoding.Default))
        {
            sw.WriteLine("*****");
            sw.WriteLine("Starting to work...");
        }
    }

    public List<Detail> ReadyDetailsByType(List<Detail> toChooseFrom, Type type)
    {
        List<Detail> statD = toChooseFrom.Where(x => (x.type == type)).ToList();
        return statD;
    }
}

```

Form1.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Modeling

```

```

{
    public partial class Form1 : Form
    {
        Factory factory;
        int ticks = 0, time = -1;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
        }

        private void StartButton_Click(object sender, EventArgs e)
        {
            int[] C = new int[3];
            int[] D = new int[3];
            C[0] = (int)numericUpDown2.Value;
            C[1] = (int)numericUpDown3.Value;
            C[2] = (int)numericUpDown4.Value;
            D[0] = (int)numericUpDown5.Value;
            D[1] = (int)numericUpDown6.Value;
            D[2] = (int)numericUpDown7.Value;
            factory = new Factory(C, D);
            factory.SetWorkingParams((int)numericUpDown1.Value, (int)numericUpDown9.Value,
(int)numericUpDown8.Value);
            timer2.Start();
            label10.Text = "Process is not finished";
            ticks = 0;
        }

        //for each new iteration
        private void timer2_Tick(object sender, EventArgs e)
        {
            time = factory.Work();
            //check if processing is finished
            if (time == -1) {
                timer2.Stop(); ShowStats();
                label10.Text = "Process is finished";
            }
            else timer2.Interval = time * 50;

            //get statistic and put it into listbox
            ShowStats();

            ticks++;
        }

        private void ShowStats()
        {
            listBox1.Items.Clear();
            listBox2.Items.Clear();
            listView1.Clear();
            listBox3.Items.Clear();
            Statistic curSats = factory.GetStatistic();
            if (ticks == 0) curSats.PrintBeginning();
            //current time and ready details
            listBox1.Items.Add("Time: " + factory.minutesWorked.ToString());
            listBox1.Items.Add("Ready details: " + curSats.readyDetails.Count);
            for (int i = 0; i < 3; i++)

```

```

        if (curSats.procentage[i] != 0)
listBox1.Items.Add(curSats.procentage[i].ToString() + "% of type " +
curSats.prTypes[i].ToString());
        //queues in machines current
        for (int i = 0; i < 3; i++)
        {
            listBox2.Items.Add("Machine " + (i+1));
            listBox2.Items.Add("    Queue: " + curSats.Queues[i]);
        }
        //all details info
listBox3.Items.Add("All details: " + curSats.allD.Length/2);
        for (int i = 0; i < 3; i++)
            listBox3.Items.Add("Details of type " +
Enum.GetValues(typeof(Type)).GetValue(i).ToString() + " : " + curSats.allDetails[i].Count);
        //list of all details
        ListViewItem lvi = new ListViewItem();
        lvi.Text = curSats.allD;
        listView1.Items.Add(lvi);
        curSats.PrintToFile();
    }

    private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
    {

    }

    private void label11_Click(object sender, EventArgs e)
    {

    }

    private void listBox2_SelectedIndexChanged(object sender, EventArgs e)
    {
        int curItem = listBox2.SelectedIndex;

    }

    private void button2_Click(object sender, EventArgs e)
    {
        if (ticks != 0 && time != -1)
        {
            if (button2.Text == "Pause")
            {
                timer2.Stop();
                button2.Text = "Continue";
            }
            else
            {
                timer2.Start();
                button2.Text = "Pause";
            }
        }
    }
}
}
}
}

```

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были закреплены навыки работы с языком объектно-ориентированного программирования C#, освоен процесс создания программ с использованием Windows Forms в среде Microsoft Visual Studio. В первой части курсовой работы успешно применяется наследование и создается иерархия классов. Во второй части корректно работает алгоритм поиска количества остовов и минимального остова в связном взвешенном графе. В третьей части была разработана имитационная модель и создан пользовательский графический интерфейс.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Горячев А.В., Новакова Н.Е. Методические указания к лабораторным работам по дисциплине объектно-ориентированное программирование, СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2014.
2. Приложение для создания диаграмм. // Lucidchart. URL: <https://www.lucidchart.com/pages/> (даты обращения: 25.05.2021, 01.06.2021, 02.06.2021)
3. Основные определения теории графов. // Викиконспекты. URL: https://neerc.ifmo.ru/wiki/index.php?title=Основные_определения_теории_графов
4. Руководство по использованию модульных тестов. // Официальный сайт Microsoft. URL: <https://docs.microsoft.com/ru-ru/visualstudio/test/walkthrough-creating-and-running-unit-tests-for-managed-code?view=vs-2019>
5. Строгалева В. П., Толкачева И. О. Имитационное моделирование. — МГТУ им. Баумана, 2008.
6. Эльберг М. С., Цыганков Н. С. Имитационное моделирование. — Красноярск, СФУ, 2017 – С.40 – 43.
7. Основы LINQ. // Metanit.com, сайт о программировании. URL: <https://metanit.com/sharp/tutorial/15.1.php>
8. Принципы SOLID. // WebDev. URL: <https://medium.com/webbdev/solid-4ffc018077da>