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

Студент гр. 9301 _____ Примакова Е.Е.

Преподаватель _____ Новакова Н.Е.

Санкт-Петербург

2021

ЗАДАНИЕ

КУРСОВАЯ РАБОТА

Студент Примакова Е.Е.

Группа 9301

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

Исходные данные:

Поставленное задание ()

Язык программирования: C# (.NET 4.6.1)

Содержание пояснительной записки:

«Содержание», «Введение», «Первый раздел. Создание иерархии классов», «Второй раздел. Поиск максимального потока в ориентированном графе», «Третий раздел. Имитационное моделирование», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 50 страниц.

Дата выдачи задания: 8 февраля 2021

Дата сдачи курсовой работы:

Дата защиты курсовой работы:

Студент _____ Примакова Е.Е.

Преподаватель _____ Новакова Н.Е.

АННОТАЦИЯ

Курсовой проект содержит в себе решения таких задач, как создание иерархии классов, поиск максимального потока в сети, разработка имитационной модели исследуемой системы. На основе полученных моделей были разработаны консольные приложения. Примеры и результаты работы приведены в качестве примеров.

SUMMARY

Course project contains solutions to such problems as creating a class hierarchy, finding the maximum flow in the network, and developing a simulation model of the system under study. Console applications were developed based on the models obtained. Examples and results of the work are given as examples.

Оглавление

ВВЕДЕНИЕ

Основная цель работы — освоение принципов объектно-ориентированного подхода в разработке приложений и применение их на практике. В работе ставятся и решаются такие задачи, как разработка имитационной модели исследуемой системы, исследование математической задачи и реализация алгоритма её решения.

Реализация каждой объектной модели проводится с помощью языка программирования C#.

ПЕРВЫЙ РАЗДЕЛ. СОЗДАНИЕ ИЕРАРХИИ КЛАССОВ

Цель

Целью данной части работы является ознакомление с иерархией классов, получение навыков работы с наследованием, закрепление навыков работы с классами, полученных за семестр.

Задание

Вариант 17

Разработать программу для представления склада продуктов

Упражнение 1 – Разработка структур данных для заданной предметной области

В этом упражнении необходимо для заданной предметной области (первый раздел курсовой работы) разработать несколько классов и интерфейсов. Желательно использовать абстрактный класс. Применить механизмы наследования. Результаты представить с помощью диаграммы классов и спецификации.

Упражнение 2 – Разработка программы на основе созданных в упр.1 структур данных

В этом упражнении необходимо для заданной предметной области разработать программу. Необходимо использовать механизмы наследования с применением интерфейсов и абстрактных классов.

Теоретический аспект задачи

Абстрактные классы — классы, реализованные с ключевым словом abstract. Ключевая особенность этих классов состоит в том, что создавать экземпляры этих классов нельзя. Данные классы используются для описания общих свойств некоторых классов объектов, после чего другие классы их наследуют. Таким образом, у классов наследуются поля и методы, а при необходимости что-то поменять меняется только абстрактный класс, а не все классы, наследовавшие его, по отдельности.

Наследование классов. Благодаря наследованию один класс может унаследовать функциональность другого класса. Все классы по умолчанию могут наследоваться. Однако здесь есть ряд ограничений: не поддерживается множественное наследование — класс может наследоваться только от одного класса; при создании производного класса надо учитывать тип доступа к базовому классу — тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть, если базовый класс у нас имеет тип доступа internal, то производный класс может иметь тип доступа internal или private, но не public, однако следует также учитывать, что если базовый и производный класс находятся в разных сборках (проектах), то в этом случае производный класс может наследовать только от класса, который имеет модификатор public; если класс объявлен с модификатором sealed, то от этого класса нельзя наследовать и создавать производные классы; нельзя наследовать от статического класса.

Интерфейс. Для определения интерфейса используется ключевое слово interface. Интерфейс представляет ссылочный тип, который может определять некоторый функционал - набор методов и свойств без реализации. Затем этот функционал реализуют классы и структуры, которые применяют данные интерфейсы. Методы и свойства интерфейса могут не иметь реализации, в этом они сближаются с абстрактными методами и свойствами абстрактных классов. По умолчанию спецификатор доступа public.

Формализация задачи

Диаграмма используемых в программе классов представлена на рисунке 1.1.

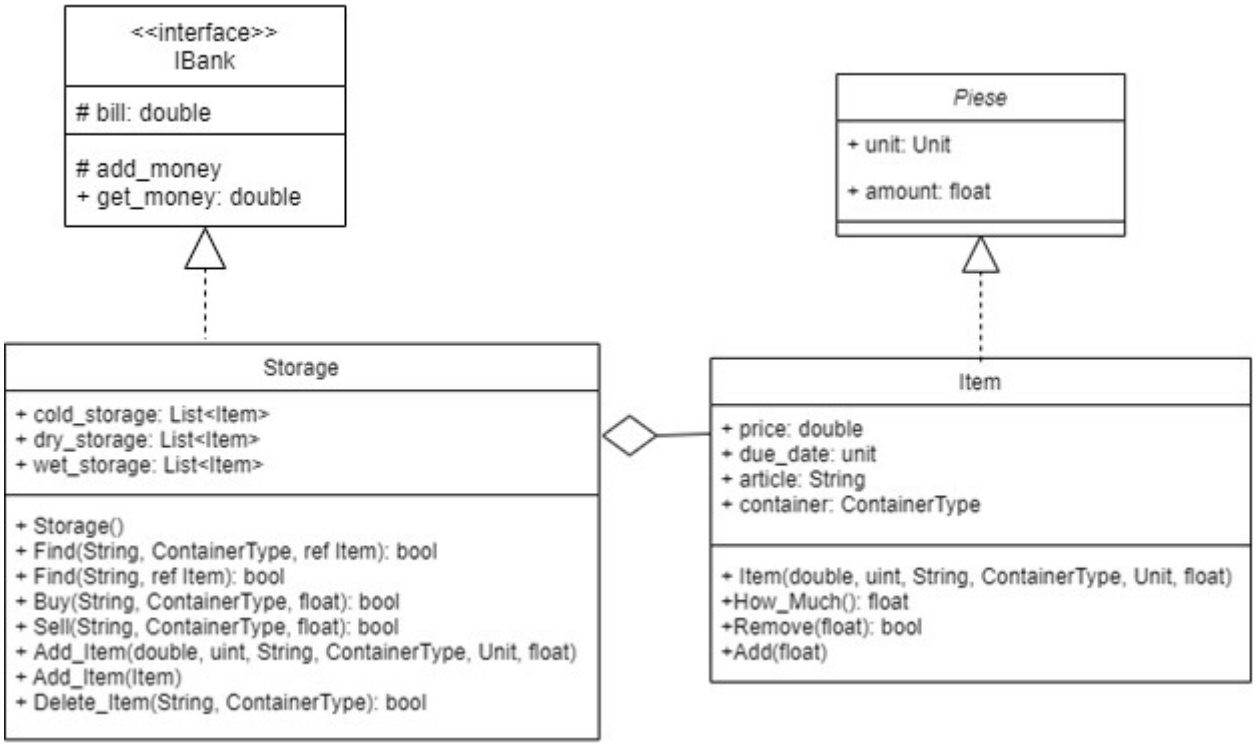


Рис. 1.1. Диаграмма классов.

Классы:

•*Piece* — абстрактный класс, описывающий единицу продукции. Тип Unit — пользовательский тип, перечисление единиц измерения.

•*Item* — класс, описывающий наименование. В этом классе есть такие поля как место хранения, артикул и количество на складе. Класс наследует от Piece его поля.

•*Storage* — класс, описывающий склад. В данном классе представлены списки товаров, хранящихся в разных местах.

•*IBank* — интерфейс, описывающий банковский счёт склада.

Спецификация программы

Описания методов и полей классов представлены в таблицах 1.1.а- 1.4.б.

Класс *Storage*

Таблица 1.1.а – описание методов класса Storage

Метод	Назначение	Возвращаемый тип	Модификатор доступа	Входные параметры
Find	Поиск элемента	bool	public	Article, container, Item
Find	Поиск элемента	bool	public	Item
Buy	Купить элемент	bool	public	Article, container, amount
Sell	Продать элемент	bool	public	Article, container, amount
Add_Item	Добавить новый элемент	void	public	Article, container, amount, price, due_date
Add_Item	Добавить новый элемент	void	public	Item
Delete_Item	Удалить элемент	bool	public	Article, container

Таблица 1.1.б – описание полей класса Storage

Имя	Тип	Модификатор доступа	Назначение
cold_storage	List<Item>	public	Список элементов, хранящихся на складе
dry_storage	List<Item>	public	Список элементов, хранящихся на складе
wet_storage	List<Item>	public	Список элементов, хранящихся на складе

Класс *Item*

Таблица 1.2.а – Описание методов класса Item

Метод	Назначение	Возвращаемый тип	Модификатор доступа	Входные параметры
How_Much	Возвращает количество предметов на складе	float	public	-
Remove	Уменьшает количество	bool	public	amount
Add	Увеличивает количество	void	public	amount

Таблица 1.2.б – Описание полей класса Item

Имя	Тип	Модификатор доступа	Назначение
price	double	public	Цена за единицу товара
due_date	uint	public	Срок годности
article	String	public	Артикул
container	ContainerType	public	Место хранения

Класс *Piece*

Таблица 1.3.а – Описание полей класса Piece

Имя	Тип	Модификатор доступа	Назначение
unit	Unit	public	Единица измерения
amount	float	public	количество

Класс IBank

Таблица 1.4.а – описание методов класса *IBank*

Метод	Назначение	Возвращаемый тип	Модификатор доступа	Входные параметры
add_money	Добавление баланса	void	protected	Amount
get_money	Получение значения баланса	double	public	-

Таблица 1.4.б – описание полей класса *IBank*

Имя	Тип	Модификатор доступа	Назначение
bill	double	protected static	Баланс счёта

Руководство пользователя

Назначение программы

Программа реализует систему склада с возможностью покупать на склад и продавать со склада элементы.

Условие выполнения программы

Для запуска программы необходима установленная операционная система Windows версии не ниже XP и .NET Framework версии не ниже 4.6.1.

Выполнение программы

В функции main класса lab9 пропишите необходимые манипуляции, используя методы классов. Запустите программу, нажав F5 или выбрав запуск без отладки.

Руководство программиста

Назначение и условие выполнения программы

Программа реализует систему хранения склада. В функции main класса lab9 пропишите необходимые манипуляции, используя методы классов.

Характеристика программы

Программа разработана в среде Microsoft Visual Studio 2017, использовалась версия платформы .NET 4.6.1. Язык разработки — C#. Программа отвечает требованиям, описанным в задании.

Выполнение программы

Для запуска программы нажмите f5 или выберите режим «запуск без отладки»

Входные и выходные данные

Входные данные в программе не требуются: пользователь рассматривает уже заполненный каталог.

В качестве выходные данных выступают отображение характеристик и изображения выбранного транспортного средства.

Контрольные примеры

В функции main создаётся объект класса Storage — склад, добавляются 2 наименования с помощью метода Add_Item, производится покупка. На рисунке 1 представлен результат выполнения программы.

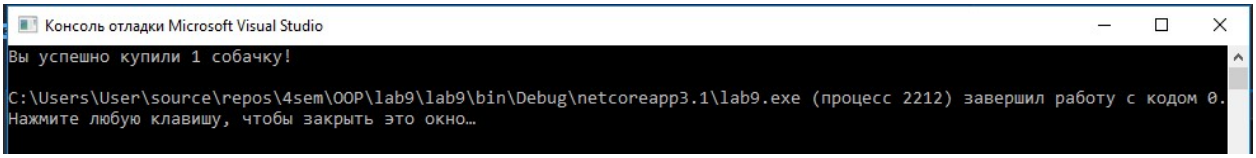


Рис. 1.2. Пример работы программы.

Листинг

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

namespace lab9
{
    enum ContainerType // тип хранения
    {
        COLD, // холодильник
        WET, // сухой склад
        DRY, // влажный склад
    }

    enum Unit // единица измерения
    {
        piece, // штуки
        kg, // килограммы
        gram, // граммы
        pack, // упаковки
        liter, // литры
        ml, // милилитры

    }

    abstract class Piece // единица
    {
        public Unit unit; // единица измерения -- в них количество товара и за 1 единицу этого стоимость
        public float amount; // количество тоара в единицах unit -- добавить ограничения на >=0
    }

    class Item : Piece // наименование
    {
        public double price; // цена за единицу товара
        public uint due_date; // [1;366] срок годности
        public String article; // артикул товара (строка потому что артикул может содержать буквы и знаки препинания)
```

```
        public ContainerType container; // место хранения --
сухой/влажный склад или холодильник

        public Item(double price = 0.0, uint due_date = 0, String
article = "", ContainerType container = 0, Unit unit = 0, float
amount = 0)
        {
            this.price = price; // инициализация поля цена
            this.due_date = due_date; // инициализация поля срок
годности
            this.article = article; // инициализация поля артикул
            this.container = container; // инициализация поля место
хранения
            this.unit = unit; // инициализация поля единица
измерения
            this.amount = amount; // инициализация поля количество
        }
        public float How_Much() // возвращает значение поля
количество
        {
            return amount;
        }
        public bool Remove(float amount) // уменьшает количество на
заданное значение, если это возможно
        {
            if (this.amount < amount)
            {
                return false;
            }
            this.amount -= amount;
            return true;
        }

        public void Add(float amount) // увеличивает количество на
заданное значение
        {
            this.amount += amount;
        }

    }

    class Storage // склад
    {
        public List<Item> cold_storage; // список товаров,
хранящихся в холодильнике
        public List<Item> wet_storage; // список товаров на влажном
складе
        public List<Item> dry_storage; // список товаров в сухом
складе

        public Storage()
        {
            cold_storage = new List<Item>();
```

```
            wet_storage = new List<Item>();
            dry_storage = new List<Item>();
        }

        public bool Find(String article, ContainerType container,
ref Item item) // поиск элемента на складе типа container
        {
            switch (container) // в зависимости от типа контейнера
            {
                // просматриваем каждый элемент списка хранилища,
пока не найдём тот элемент, который нам нужен
                case ContainerType.COLD:
                    {
                        foreach (Item a in cold_storage)
                        {
                            if (a.article == article)
                            {
                                item = a;
                                return true;
                            }
                        }
                        break;
                    }
                case ContainerType.WET:
                    {
                        foreach (Item a in wet_storage)
                        {
                            if (a.article == article)
                            {
                                item = a;
                                return true;
                            }
                        }
                        break;
                    }
                case ContainerType.DRY:
                    {
                        foreach (Item a in dry_storage)
                        {
                            if (a.article == article)
                            {
                                item = a;
                                return true;
                            }
                        }
                        break;
                    }
            }
            return false; // если элемент не был найден
        }

        public bool Find(String article, ref Item item) // поиск
элемента на складе всех типов
```

```
        {
            // проходим по всем хранилищам, пока не найдём элемент
            foreach (Item a in cold_storage)
            {
                if (a.article == article)
                {
                    item = a;
                    return true;
                }
            }
            foreach (Item a in wet_storage)
            {
                if (a.article == article)
                {
                    item = a;
                    return true;
                }
            }
            foreach (Item a in dry_storage)
            {
                if (a.article == article)
                {
                    item = a;
                    return true;
                }
            }
            return false; // если элемент не был найден
        }

        public bool Buy(String article, ContainerType container,
float amount)
        {
            Item item = new Item(); // переменная для хранения
наименования склада
            if(Find(article, container, ref item)) // ищем элемент
            {
                return item.Remove(amount); // если вохможно,
убираем нужное количество, если нет -- вернём false
            }
            return false; // если элемент не был найден
        }

        public bool Sell(String article, ContainerType container,
float amount)
        {
            Item item = new Item();
            if (Find(article, container, ref item))
            {
                item.Add(amount); // добавляем элементы
                return true;// если элемент был найден
            }
            return false; // если элемент не был найден
        }
```

```
        public void Add_Item(double price = 0.0, uint due_date = 0,
String article = "",
            ContainerType container = 0, Unit unit = 0, float amount
= 0) // добавление элемента на склад
        {
            // создаём элемент и добавляем его на свой склад
            Item item = new Item(price, due_date, article,
container, unit, amount);
            switch (container)
            {
                case ContainerType.COLD:
                    {
                        cold_storage.Add(item);
                        break;
                    }
                case ContainerType.WET:
                    {
                        wet_storage.Add(item);
                        break;
                    }
                case ContainerType.DRY:
                    {
                        dry_storage.Add(item);
                        break;
                    }
            }
        }

        public void Add_Item(Item item) // добавление элемента на
склад
        {
            // добавляем элемент на свой склад
            switch (item.container)
            {
                case ContainerType.COLD:
                    {
                        cold_storage.Add(item);
                        return;
                    }
                case ContainerType.DRY:
                    {
                        dry_storage.Add(item);
                        return;
                    }
                case ContainerType.WET:
                    {
                        wet_storage.Add(item);
                        return;
                    }
            }
        }
```

```
        public bool Delete_Item(String article, ContainerType
container) // удаление наименования из списка хранения
        {
            Item item = new Item();
            if(Find(article, container, ref item)) // если элемент
найден
            { // удаляем его из своего контейнера
                switch (container)
                {
                    case ContainerType.COLD:
                        {
                            cold_storage.Remove(item);
                            return true;
                        }
                    case ContainerType.WET:
                        {
                            wet_storage.Remove(item);
                            return true;
                        }
                    case ContainerType.DRY:
                        {
                            dry_storage.Remove(item);
                            return true;
                        }
                }
            }
            return false; // если элемент не был найден
        }
    }
}
```

Функция main:

```
class lab9
    {
        static void Main(string[] args)
        {
            Storage myStorage = new Storage();
            myStorage.Add_Item(10.0, 256, "мяу", ContainerType.DRY,
Unit.piece, 100); // Добавляем на сухой склад 100 котиков по цене 10
            myStorage.Add_Item(100.0, 304, "гав", ContainerType.WET,
Unit.pack, 100); // Добавляем на влажный склад 100 собачек по цене
100
            if(myStorage.Buy("гав", ContainerType.WET, 1))
            {
                Console.WriteLine("Вы успешно купили 1 собачку!");
            }
        }
    }
```

Выводы

В ходе работы была реализована структура классов, описывающая склад для хранения. Были реализованы методы добавления и удаления элементов со склада, покупки и продажи существующих.

Также в ходе выполнения данной лабораторной работы были изучены принципы наследования и закреплены навыки работы с классами в C#, на их основе была реализована модель, описывающая склад продуктов.

ВТОРОЙ РАЗДЕЛ. МАКСИМАЛЬНЫЙ ПОТОК В СЕТИ

Цель

Целью данной части работы является разработка и реализация программы для работы с графами.

Задание

Вариант Г-42-4

Для заданного орграфа найти кратчайший путь между вершинами 1 и 12. Задачу решить в общем виде. На рисунке рядом с вершинами указан ее вес.

В качестве контрольного примера использовать задания по вариантам.

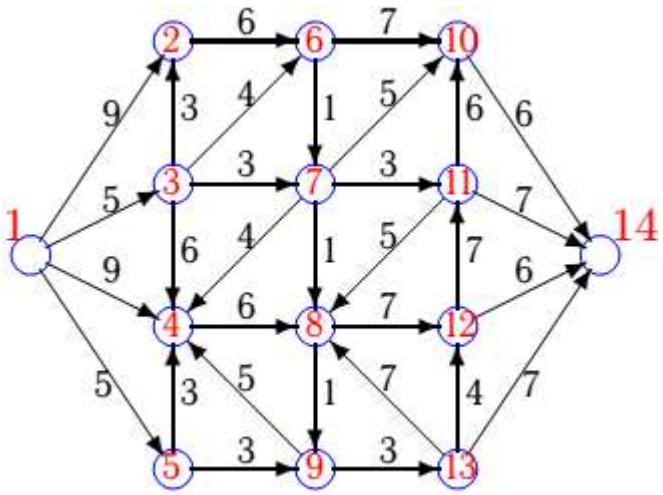


Рисунок 2.1. Граф для контрольного примера

Теоретический аспект задачи

Сетью называют взвешенный граф с двумя выделенными вершинами: истоком и стоком. Исток имеет нулевую полустепень захода, а сток – нулевую полустепень исхода. Вес дуги означает ее пропускную способность. Поток – еще одно число, приписанное дуге. Поток дуги не больше ее пропускной способности и может меняться. Поток выходит из истока и без потерь, в том же объеме заходит в сток. Условие равновесия (по объему входа и выхода) выполняется для каждой вершины сети.

Задача о наибольшем потоке в сети – не единственная, но, вероятно, основная задача для потоков в сети. Очевидна возможность практического применения этой

задачи для решения транспортных проблем (пробки на дорогах можно условно связать с насыщением сети или отдельной ее дуги), проблем транспортировки нефтепродуктов или электроэнергии.

Алгоритм нахождения максимального потока в сети Форда – Фалкерсона состоит из двух частей – насыщения потока и его перераспределения. Поток называется насыщенным, если любая цепь из истока в сток и все дуги цепи насыщаются одинаковым возможно большим потоком, определяемым пропускной способностью наиболее «тонкой» дуги или наименьшей разностью между пропускной способностью и потоком в дуге. Различные цепи могут иметь общие дуги. Полученный поток согласован с условием сохранения в узлах (вершинах). Поток, входящий в вершину, равен потоку, выходящему из нее. Поток в сети проходит по цепям из истока в сток, т.е. недопустим многократный проход по отдельной дуге. Первая часть задачи считается решенной, если нет ненасыщенных цепей из истока в сток. Первая часть задачи не имеет единственного решения.

Во второй части перераспределение потока выполняется исходя из условия достижения общего по сети максимума потока. Для этого в основании графа (т.е. в графе, в котором снята ориентация дуг) разыскиваются маршруты из истока в сток, состоящие из ребер, соответствующих ненасыщенным дугам, направленным вперед, и непустым дугам, направленным назад. Потоки в дугах прямого направления увеличиваются на величину, на которую уменьшаются потоки в обратных дугах выбранного маршрута. При этом, очевидно, нельзя превышать пропускную способность дуг, направленных вперед, и допускать отрицательных потоков в обратных дугах. В некоторых случаях при удачном выборе цепей в первой части алгоритма перераспределения потока не требуется.

1. Насыщение потока. Рассмотрим путь 1-2-4-6-8. Пропустим через этот путь поток, равный 4. При этом дуги [2, 4] и [4, 6] будут насыщенными. Аналогично, путь 1-3-5-7-8 насытим потоком 4. Распределение потока отметим на графе (рис. 3.2). В числителе ставим пропускную способность, в знаменателе — поток. Числитель всегда больше знаменателя, а знаменатель может быть и нулем.

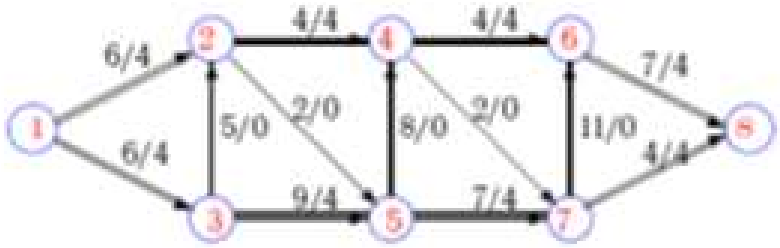


Рис. 2.2

Заметим, что из 1 в 8 есть еще ненасыщенный путь, 1-3-2-5-4-7- 6-8, поток в котором можно увеличить на 2. При этом насытятся дуги [1, 3]. [2, 5] и [4, 7] (рис. 3.3).

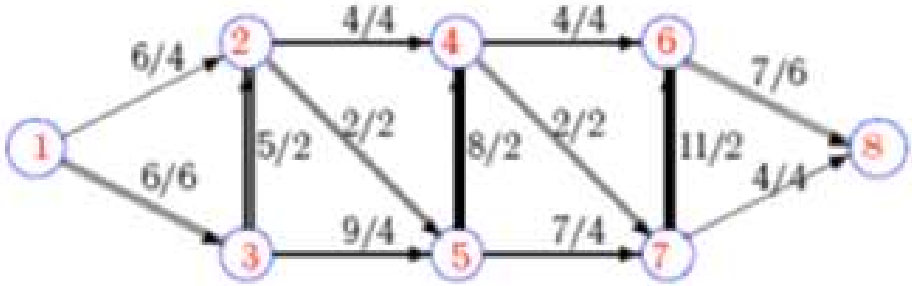


Рис. 2.3

Из 1 в 8 больше нет ненасыщенных путей. По дуге [1,3] двигаться нельзя (она уже насыщена), а движение по дуге [1,2] заканчивается в вершине 2, так как обе выходящие из нее дуги насыщены.

2. Перераспределение потока. Найдем последовательность вершин от 1 к 8, такую, что дуги, соединяющие соседние вершины, направленные из 1 в 8. не насыщены, а дуги, направленные в обратном направлении, не пусты. Имеем единственную последовательность: 1- 2-3-5-7-6-8. Перераспределяем поток. Поток в дугах прямого направления увеличиваем на 1. а поток в дугах обратного направления уменьшаем на 1. Процесс продолжаем до тех пор, пока одна из прямых дуг не будет насыщена или какая-нибудь обратная дуга не будет пуста [3].

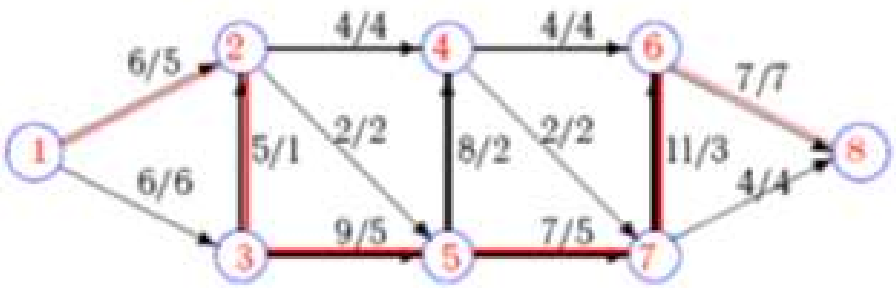


Рис. 2.4

Формализация задачи

Найти максимальный поток в сети с помощью алгоритма Форда-Фалкерсона. Задачу решить в общем виде. В качестве контрольного примера использовать задание соответствующего варианта.

Руководство пользователя

Назначение программы

Программа реализует нахождение максимального потока графа с помощью алгоритма Форда-Фалкерсона.

Условие выполнения программы

Для запуска программы необходима установленная операционная система Windows версии не ниже XP.

Выполнение программы

После запуска программы на экран пользователю выводится максимальный поток представленного графа (алгоритм использует матрицу смежности).

Сообщение пользователю

После работы алгоритма пользователь получает сообщение о максимальном потоке заданной сети.

Руководство программиста

Назначение и условие выполнения программы

Программа реализует нахождение максимального потока графа с помощью алгоритма Форда-Фалкерсона. Для запуска программы необходима установленная операционная система Windows версии не ниже XP.

Характеристика программы

Программа разработана в среде Microsoft Visual Studio 2017. Язык разработки — C#. Программа отвечает требованиям, описанным в задании.

Выполнение программы

Выполнение программы описано в разделе «Руководство пользователя». Матрица смежности сети вводится в функции main.

Входные и выходные данные

В качестве выходных данных выступает результат работы алгоритма

Контрольный пример

В качестве контрольного примера использован граф, представленный на рисунке 3.7.

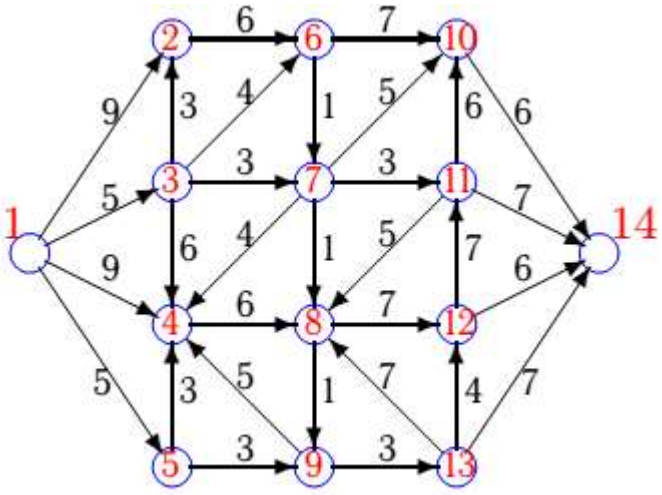


Рисунок 2.5. Граф, используемый для контрольного примера.

На рисунке 2.6 представлен вывод программы.

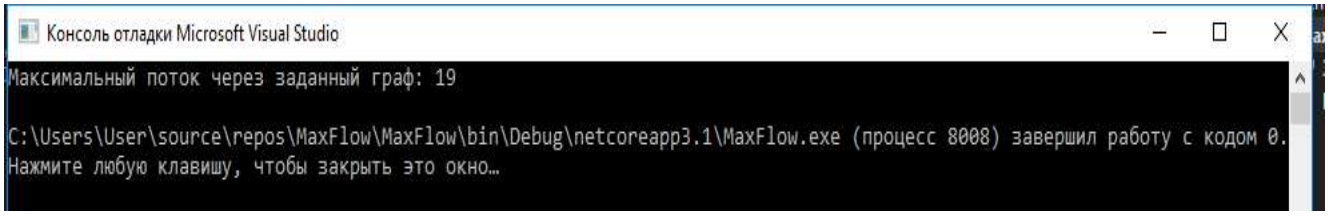


Рис. 2.6. Результат работы программы.

Листинг

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

public class MaxFlow
{
    static readonly int V = 14; // кол-во вершин

    // возвращает true, если есть путь от
    // S до T в графе. заполняет parent[] для
    // хранения пути
    bool bfs(int[,] rGraph, int s, int t, int[] parent)
    {
        // создаем массив посещений и
        // помечаем все вершины как не посещенные
        bool[] visited = new bool[V];
```

```
        for (int i = 0; i < V; ++i)
            visited[i] = false;

        // создаем очередь и добавляем исходную
        // вершину в очередь, отмечаем вершину как посещенную
        List<int> queue = new List<int>();
        queue.Add(s);
        visited[s] = true;
        parent[s] = -1;

        while (queue.Count != 0)
        {
            int u = queue[0];
            queue.RemoveAt(0);

            for (int v = 0; v < V; v++)
            {
                if (visited[v] == false
                    && rGraph[u, v] > 0)
                {
                    // если находим соединение стока с истоком в
остаточном графе,
                    // то прекращаем алгоритм BFS. устанавливаем
                    // родительский узел и возвращаем true
                    if (v == t)
                    {
                        parent[v] = u;
                        return true;
                    }
                    queue.Add(v);
                    parent[v] = u;
                    visited[v] = true;
                }
            }
        }

        // мы не нашли путь в BFS, поэтому возвращаем false
        return false;
    }

    // возвращает максимальный поток из S в T в графе
    int fordFulkerson(int[,] graph, int s, int t)
    {
        int u, v;

        // создаем остаточный граф и заполняем его
        // заданными пропускными способностями.
        // остаточный граф rGrapg[i,j] показывает
        // остаточную пропускную способность ребра
        // от i до j.
        int[,] rGraph = new int[V, V];

        for (u = 0; u < V; u++)
```

```
            for (v = 0; v < V; v++)
                rGraph[u, v] = graph[u, v];

        // массив зяполняется BFS и хранит путь
        int[] parent = new int[V];

        int max_flow = 0; // изначально максимальный поток равен 0

        // пока существует путь от истока до стока в остаточном
графе
        // увеличиваем максимальный поток
        while (bfs(rGraph, s, t, parent))
        {
            // найдем максимальный поток на найденном пути
            int path_flow = int.MaxValue;
            for (v = t; v != s; v = parent[v])
            {
                u = parent[v];
                path_flow
                    = Math.Min(path_flow, rGraph[u, v]);
            }

            // обновляем очтаточные пропускные способности
            // на найденном пути
            for (v = t; v != s; v = parent[v])
            {
                u = parent[v];
                rGraph[u, v] -= path_flow;
                rGraph[v, u] += path_flow;
            }

            // добавляем максимальный поток пути к общему
максимальному потоку
            max_flow += path_flow;
        }

        return max_flow;
    }

    public static void Main()
    {
        int[,] graph = new int[,] {
            { 0,9,5,9,5,0,0,0,0,0,0,0,0,0,0 },
            { 0,0,0,0,0,6,0,0,0,0,0,0,0,0,0 },
            { 0,3,0,6,0,4,3,0,0,0,0,0,0,0,0 },
            { 0,0,0,0,0,0,0,6,0,0,0,0,0,0,0 },
            { 0,0,0,3,0,0,0,0,3,0,0,0,0,0,0 },
            { 0,0,0,0,0,0,1,0,0,7,0,0,0,0,0 },
            { 0,0,0,0,0,0,0,1,0,5,3,0,0,0,0 },
            { 0,0,0,0,0,0,0,0,1,0,0,7,0,0,0 },
            { 0,0,0,5,0,0,0,0,0,0,0,0,3,0,0 },
            { 0,0,0,0,0,0,0,0,0,0,0,0,0,0,6 },
            { 0,0,0,0,0,0,0,5,0,0,0,0,0,7,0 },
```

```
            { 0,0,0,0,0,0,3,0,0,0,7,0,0,6 },
            { 0,0,0,0,0,0,0,7,0,0,0,4,0,7 },
            { 0,0,0,0,0,0,0,0,0,0,0,0,0,0 }
        };
        MaxFlow m = new MaxFlow();

        Console.WriteLine("максимальный поток через заданный граф: "
                          + m.fordFulkerson(graph, 0, 13));
    }
}
```

Вывод

В ходе работы была разработана программа, реализующая нахождение максимального потока в сети с помощью алгоритма Форда-Фалкерсона. Алгоритмы на графах

ТРЕТИЙ РАЗДЕЛ. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

Цель

Целью данной части работы является разработка и реализация имитационной модели, соответствующей варианту задания, углубление знаний об объектно-ориентированном подходе к программированию.

Задание

Вариант 19

Система управления оптовым складом

Оптовый склад, на котором хранятся K видов ($12 \leq K \leq 20$) продуктовых товаров, обслуживает M ($3 \leq M \leq 9$) близлежащих торговых точек (мелких магазинов и палаток). Вместимость склада ограничена: каждого вида товара может храниться не более определенного количества оптовых упаковок. Все продукты имеют срок годности, и если этот срок истекает через несколько дней, то товар уценивается, чтобы продать его быстрее. После истечения срока годности продуктовый товар списывается и вывозится со склада.

Система управления складом хранит данные о наличии и количестве каждого продукта в текущий момент (например, 12 оптовых упаковок гречи, в каждой 20 пачек по 1 кг), а также о сроке годности продукта и его стоимости. Система фиксирует поступающие в течение рабочего дня заказы на доставку товаров со склада в торговые точки, последовательно их обрабатывает и формирует список соответствующих перевозок на следующий рабочий день.

Каждый день от любой торговой точки на склад поступает не более одного заказа, который включает перечень заказываемых продуктов и их количество.

Поскольку любой товар отпускается в оптовых упаковках, при выполнении заказа на определенный продукт система выделяет такое количество оптовых упаковок, которое дает чуть большее или чуть меньшее количество товара (килограмм, пачек или др.) по сравнению с заказанным. Если заказанного товара нет в достаточном количестве, то он отпускается частично.

Система управления складом отслеживает убыль товаров, и если какого-то вида товара становится меньше определенного количества, то составляется заявка в фирму-поставщик на доставку на склад нужного количества этого продукта. Сформированный системой список перевозок заказанных товаров, список заявок в фирмы-поставщики и список продуктов, подлежащих уценке, может просматривать и утверждать пользователь системы (заведующим складом), при этом он может принимать решения о проценте уценки продукта.

Для тестирования заложенных в систему процедур автоматизации обработки заказов и составления заявок требуется смоделировать поток заказов, поступающих от торговых точек, а также поставку продуктов на склад фирмами-поставщиками. Период моделирования – N дней ($10 \leq N \leq 30$), шаг моделирования – один день.

Поток поступающих заказов на продукты следует моделировать статистически: случайными величинами, изменяющимися в некоторых диапазонах, являются все составляющие каждого заказа, причем вероятность заказа уцененных продуктов выше, чем неуцененных, и зависит от процента уценки. Случайной величиной (от 1 до 5 дней) является также время выполняемой по заявке поставки продуктов на склад фирмой-поставщиком. В параметры моделирования следует включить величины N, M, K, начальный набор продуктов на складе, а также диапазоны изменения вышеописанных случайных величин. В ходе моделирования должна быть доступна основная информация о работе склада: о наличии товаров, о заказах за текущий день и перевозках на следующий, о вывозе просроченных продуктов и денежных потерях склада за счет уценки продуктов и их списания др. По окончании моделирования целесообразно вывести некоторые статистические данные о работе склада за весь период моделирования, например, общий объем и стоимость проданных продуктов.

Описание имитационной модели

Предметная область системы

Продуктовый склад.

Описание объекта моделирования

Функционирование склада на протяжении нескольких дней — поступающие заказы, списки товаров на отправку.

Цель моделирования

Цель моделирования работы продуктового склада — оптимизация алгоритмов работы склада, отладка и совершенствование его функционирования.

Характер реализации модели

Выбран машинный характер реализации модели: модель представляет собой программу, позволяющую с помощью последовательных вычислений имитировать процесс функционирования объекта.

Управление модельным временем

В качестве способа изменения времени в модели выбран метод фиксированного шага, т.е. отсчёт системного времени идёт через интервалы времени определённой длины. Таким образом, разработанная модель является дискретной (переменные изменяются дискретно в определённые моменты имитационного времени).

Формализация задачи

Диаграмма используемых классов представлена на рисунке 3.1.

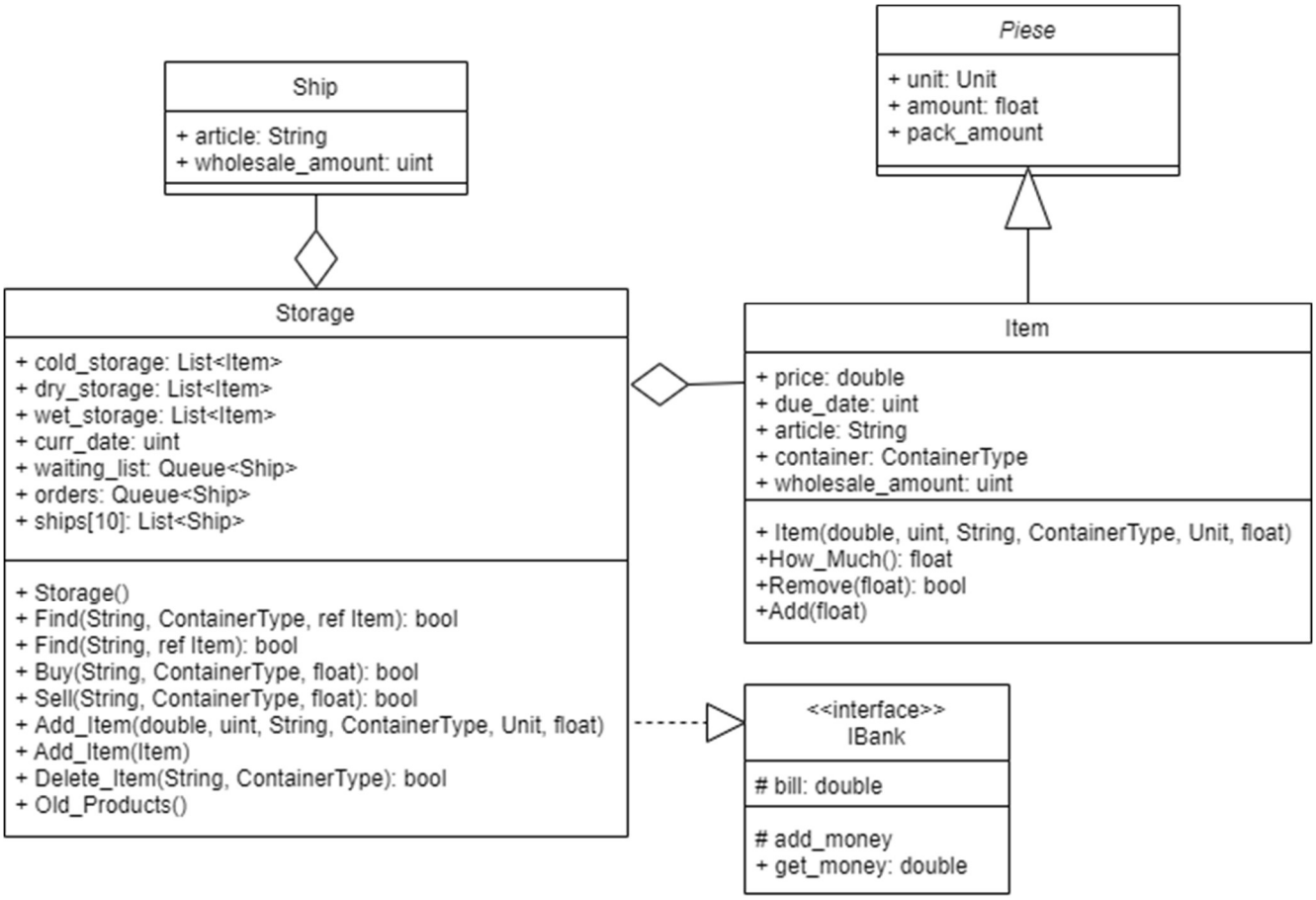


Рисунок 3.1 Диаграмма классов

Классы:

- *Ship* – класс, реализующий заявку.
- *Piece* — абстрактный класс, описывающий оптовую пачку для единицы продукции.
- *Item* — класс, описывающий наименование. В этом классе есть такие поля как место хранения, артикул и количество на складе. Класс наследует от Piece его поля.
- *Storage* — класс, описывающий склад. В данном классе представлены списки товаров, хранящихся в разных местах, а также списки заказов торговых точек и очередь на заказанные продукты.

- *IBank* — интерфейс, описывающий банковский счёт склада.

Спецификация программы

Описания методов и полей классов представлены в таблицах 3.1.а- 3.5.б.

Класс *Storage*

Таблица 3.1.а – описание методов класса Storage

Метод	Назначение	Возвращаемый тип	Модификатор доступа	Входные параметры
Find	Поиск элемента	bool	public	Article, container, Item
Find	Поиск элемента	bool	public	Item
Buy	Купить элемент	bool	public	Article, container, amount
Sell	Продать элемент	bool	public	Article, container, amount
Add_Item	Добавить новый элемент	void	public	Article, container, amount, price, due_date
Add_Item	Добавить новый элемент	void	public	Item
Delete_Item	Удалить элемент	bool	public	Article, container

Таблица 3.1.б – описание полей класса Storage

Имя	Тип	Модификатор доступа	Назначение
cold_storage	List<Item>	public	Список элементов, хранящихся на складе
dry_storage	List<Item>	public	Список элементов, хранящихся на складе
wet_storage	List<Item>	public	Список элементов, хранящихся на складе

Класс *Item*

Таблица 3.2.а – Описание методов класса Item

Метод	Назначение	Возвращаемый тип	Модификатор доступа	Входные параметры
How_Much	Возвращает количество предметов на складе	float	public	-
Remove	Уменьшает количество	bool	public	amount
Add	Увеличивает количество	void	public	amount

Таблица 3.2.б – Описание полей класса Item

Имя	Тип	Модификатор доступа	Назначение
price	double	public	Цена за единицу товара
due_date	uint	public	Срок годности
article	String	public	Артикул
container	ContainerType	public	Место хранения

Класс *Piece*

Таблица 3.3.а – Описание полей класса Piece

Имя	Тип	Модификатор доступа	Назначение
unit	Unit	public	Единица измерения
amount	float	public	количество

Класс Ship

Таблица 3.4.а – описание полей класса *Ship*

Имя	Тип	Модификатор доступа	Назначение
article	String	public	Артикул товара
wholesale_amount	uint	public	Число оптовых упаковок
where	uint	public	Точка для отправки

Класс IBank

Таблица 3.5.а – описание методов класса *IBank*

Метод	Назначение	Возвращаемый тип	Модификатор доступа	Входные параметры
add_money	Добавление баланса	void	protected	Amount
get_money	Получение значения баланса	double	public	-

Таблица 3.5.б – описание полей класса *IBank*

Имя	Тип	Модификатор доступа	Назначение
bill	double	protected static	Баланс счёта

Руководство пользователя

Назначение программы

Программа реализует имитацию работы склада продуктов и доставки продуктов в места реализации.

Условие выполнения программы

Для запуска программы необходима установленная операционная система Windows версии не ниже XP.

Выполнение программы

После запуска программы с интервалом в 3 секунды пользователь будет получать информацию о том, сколько заказов поступило на склад и было обработано, сколько товаров добавлено в список ожидания, сколько заказов нужно будет отвезти завтра.

Руководство программиста

Назначение и условие выполнения программы

Программа реализует имитацию работы склада. Для запуска программы необходима установленная операционная система Windows версии не ниже XP и .NET Framework версии не ниже 4.6.1.

Характеристика программы

Программа разработана в среде Microsoft Visual Studio 2017, использовалась версия платформы .NET 4.6.1. Язык разработки – C#. Программа отвечает требованиям, описанным в задании, и имеет простой и понятный интерфейс, реализованный с помощью платформы Windows Forms.

Выполнение программы

Выполнение программы описано в разделе «Руководство пользователя».

Входные и выходные данные

В качестве выходные данных выступает протокол выполнения программы, выведенный на экран.

Сообщения

После выполнения программы доступна статистика выполненной программы.

Контрольный пример

Результаты прогонов имитационной модели с различными параметрами представлены в листинге и на рисунке 3.2.

Листинг контрольного примера:

Точек: 5, наименований: 15, доставок 8

День 1 из 20.

Очередь заказов из 40 товаров для 5 точек

Доставок на завтра: 39, в листе ожидания: 1

День 2 из 20.

Очередь заказов из 10 товаров для 5 точек

Доставок на завтра: 11, в листе ожидания: 0

День 3 из 20.

Очередь заказов из 35 товаров для 5 точек

Доставок на завтра: 35, в листе ожидания: 0

День 4 из 20.

Очередь заказов из 40 товаров для 5 точек

Доставок на завтра: 40, в листе ожидания: 0

День 5 из 20.

Очередь заказов из 25 товаров для 5 точек

Доставок на завтра: 25, в листе ожидания: 0

День 6 из 20.

Очередь заказов из 30 товаров для 5 точек

Доставок на завтра: 30, в листе ожидания: 0

День 7 из 20.

Очередь заказов из 30 товаров для 5 точек

Доставок на завтра: 30, в листе ожидания: 0

День 8 из 20.

Очередь заказов из 30 товаров для 5 точек

Доставок на завтра: 30, в листе ожидания: 0

День 9 из 20.

Очередь заказов из 15 товаров для 5 точек

Доставок на завтра: 15, в листе ожидания: 0

День 10 из 20.

Очередь заказов из 20 товаров для 5 точек

Доставок на завтра: 20, в листе ожидания: 0

День 11 из 20.

Очередь заказов из 5 товаров для 5 точек

Доставок на завтра: 5, в листе ожидания: 0

День 12 из 20.

Очередь заказов из 40 товаров для 5 точек

Доставок на завтра: 40, в листе ожидания: 0

День 13 из 20.

Очередь заказов из 30 товаров для 5 точек

Доставок на завтра: 30, в листе ожидания: 0

День 14 из 20.

Очередь заказов из 10 товаров для 5 точек

Доставок на завтра: 10, в листе ожидания: 0

День 15 из 20.

Очередь заказов из 45 товаров для 5 точек

Доставок на завтра: 45, в листе ожидания: 0

День 16 из 20.

Очередь заказов из 40 товаров для 5 точек

Доставок на завтра: 40, в листе ожидания: 0

День 17 из 20.

Очередь заказов из 0 товаров для 5 точек

Доставок на завтра: 0, в листе ожидания: 0

День 18 из 20.

Очередь заказов из 35 товаров для 5 точек

Доставок на завтра: 35, в листе ожидания: 0

День 19 из 20.

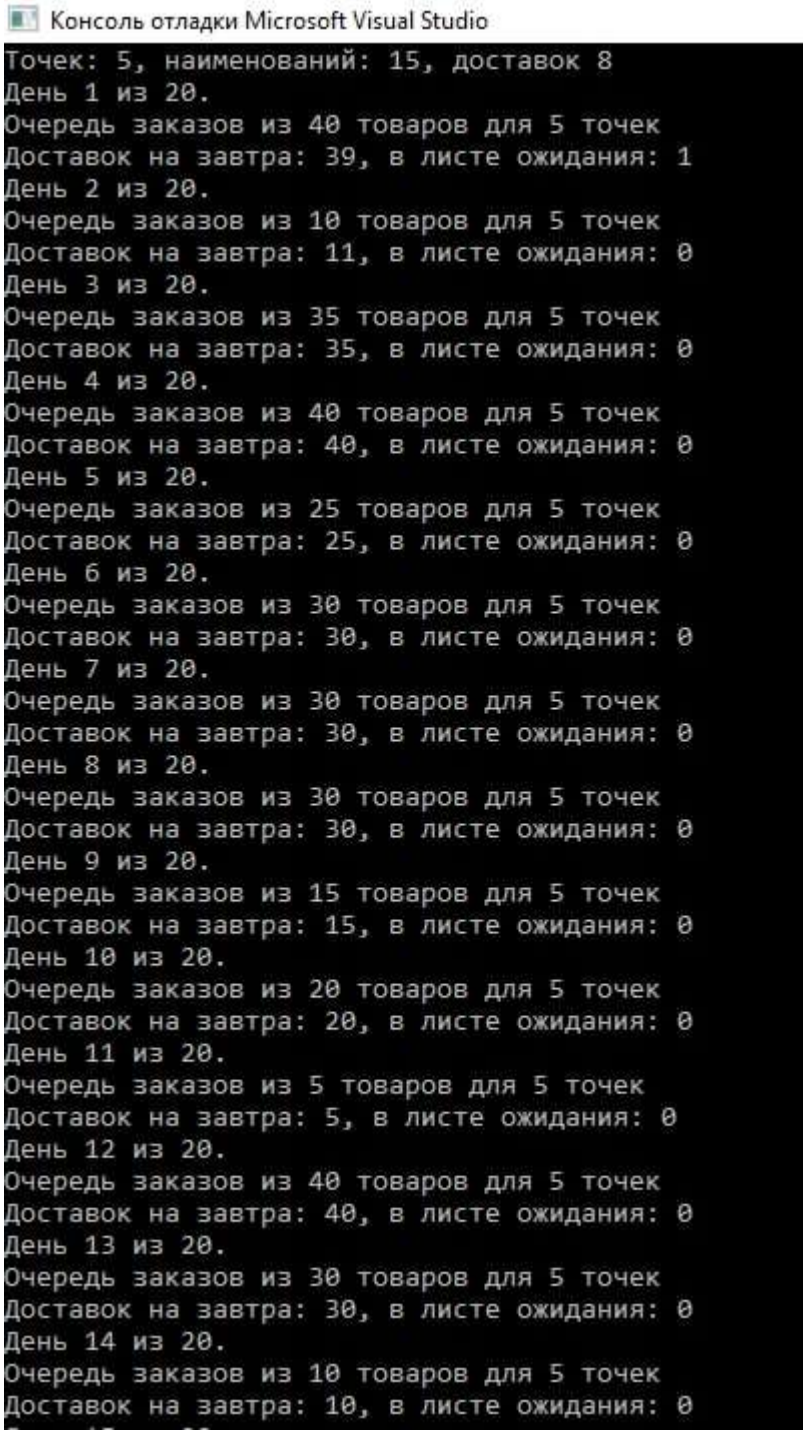
Очередь заказов из 0 товаров для 5 точек

Доставок на завтра: 0, в листе ожидания: 0

День 20 из 20.

Очередь заказов из 40 товаров для 5 точек

Доставок на завтра: 40, в листе ожидания: 0


Консоль отладки Microsoft Visual Studio
Точек: 5, наименований: 15, доставок 8
День 1 из 20.
Очередь заказов из 40 товаров для 5 точек
Доставок на завтра: 39, в листе ожидания: 1
День 2 из 20.
Очередь заказов из 10 товаров для 5 точек
Доставок на завтра: 11, в листе ожидания: 0
День 3 из 20.
Очередь заказов из 35 товаров для 5 точек
Доставок на завтра: 35, в листе ожидания: 0
День 4 из 20.
Очередь заказов из 40 товаров для 5 точек
Доставок на завтра: 40, в листе ожидания: 0
День 5 из 20.
Очередь заказов из 25 товаров для 5 точек
Доставок на завтра: 25, в листе ожидания: 0
День 6 из 20.
Очередь заказов из 30 товаров для 5 точек
Доставок на завтра: 30, в листе ожидания: 0
День 7 из 20.
Очередь заказов из 30 товаров для 5 точек
Доставок на завтра: 30, в листе ожидания: 0
День 8 из 20.
Очередь заказов из 30 товаров для 5 точек
Доставок на завтра: 30, в листе ожидания: 0
День 9 из 20.
Очередь заказов из 15 товаров для 5 точек
Доставок на завтра: 15, в листе ожидания: 0
День 10 из 20.
Очередь заказов из 20 товаров для 5 точек
Доставок на завтра: 20, в листе ожидания: 0
День 11 из 20.
Очередь заказов из 5 товаров для 5 точек
Доставок на завтра: 5, в листе ожидания: 0
День 12 из 20.
Очередь заказов из 40 товаров для 5 точек
Доставок на завтра: 40, в листе ожидания: 0
День 13 из 20.
Очередь заказов из 30 товаров для 5 точек
Доставок на завтра: 30, в листе ожидания: 0
День 14 из 20.
Очередь заказов из 10 товаров для 5 точек
Доставок на завтра: 10, в листе ожидания: 0

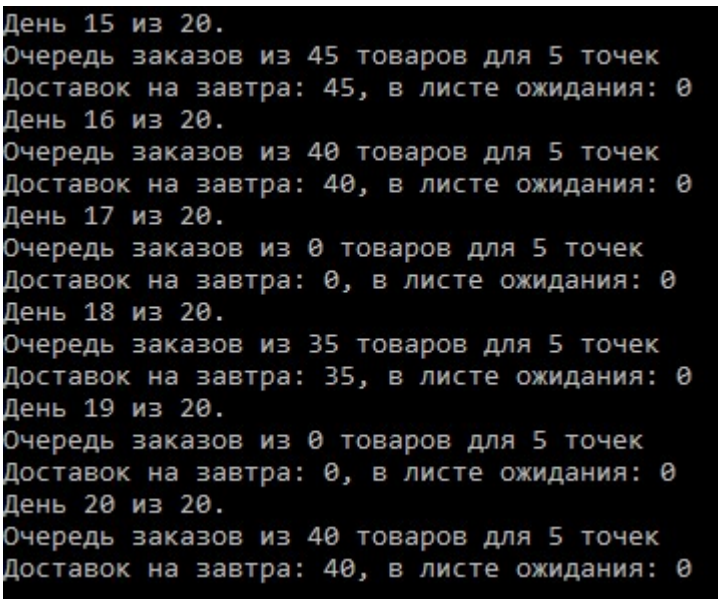


Рисунок 3.2. Контрольный пример работы программы.

Листинг

Storage.cs

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

namespace imitation
{
    class Storage : IBill // склад
    {
        public List<Item> cold_storage; // список товаров, 
хранящихся в холодильнике
        public List<Item> wet_storage; // список товаров на влажном 
складе
        public List<Item> dry_storage; // список товаров в сухом 
складе
        public uint curr_date; // текущая дата
        public List<Ship> waiting_list; // лист ожидания
        public List<Ship> order; // заказы, поступившие сегодня
        public List<Ship> ships; // заказы, которые нужно сегодня 
отправить

        public Storage(uint date = 0) // конструктор класса
        {
            cold_storage = new List<Item>();
            wet_storage = new List<Item>();
            dry_storage = new List<Item>();
            curr_date = date;
            waiting_list = new List<Ship>();
            order = new List<Ship>();
            ships = new List<Ship>();
            IBill.bill = 0;
```

```
        }

        public bool Find(String article, ContainerType container,
ref Item item) // поиск элемента на складе типа container
        {
            switch (container) // в зависимости от типа контейнера
            {
                // просматриваем каждый элемент списка хранилища,
пока не найдет тот элемент, который нам нужен
                case ContainerType.COLD:
                    {
                        foreach (Item a in cold_storage)
                        {
                            if (a.article == article)
                            {
                                item = a;
                                return true;
                            }
                        }
                        break;
                    }
                case ContainerType.WET:
                    {
                        foreach (Item a in wet_storage)
                        {
                            if (a.article == article)
                            {
                                item = a;
                                return true;
                            }
                        }
                        break;
                    }
                case ContainerType.DRY:
                    {
                        foreach (Item a in dry_storage)
                        {
                            if (a.article == article)
                            {
                                item = a;
                                return true;
                            }
                        }
                        break;
                    }
            }
            return false; // если элемент не был найден
        }

        public bool Find(String article, ref Item item) // поиск
элемента на складе всех типов
        {
            // проходим по всем хранилищам, пока не найдем элемент
```

```
            foreach (Item a in cold_storage)
            {
                if (a.article == article)
                {
                    item = a;
                    return true;
                }
            }
            foreach (Item a in wet_storage)
            {
                if (a.article == article)
                {
                    item = a;
                    return true;
                }
            }
            foreach (Item a in dry_storage)
            {
                if (a.article == article)
                {
                    item = a;
                    return true;
                }
            }
            return false; // если элемент не был найден
        }

        public bool Sell(String article, uint amount, uint where)
        {
            bool r = false;  // если элемент не был найден
            Item item = new Item(); // переменная для хранения
наименования склада
            if (Find(article, ref item)) // ищем элемент
            {
                r = item.Retail_remove(amount); // если возможно,
убираем нужное количество, если нет -- вернем false
                Ship ship = new Ship(article, amount, where);
                if (r)
                {
                    ships.Add(ship);
                    IBill.add_money(item.price * amount);
                }
                else
                {
                    waiting_list.Add(ship);
                }
            }
            return r;
        }

        public bool WS_Sell(Ship s)
        {
            bool r = false;  // если элемент не был найден
```

```
            Item item = new Item(); // переменная для хранения
наименования склада
            if (Find(s.article, ref item)) // ищем элемент
            {
                r = item.Wholesale_remove(s.wholesale_amount); //
если возможно, убираем нужное количество, если нет -- вернем false
                if (r)
                {
                    ships.Add(s);
                }
                else
                {
                    waiting_list.Add(s);
                }
            }
            return r;
        }

        public bool wholesale_sell(String article, ContainerType
container, uint amount, uint where)
        {
            bool r = false;  // если элемент не был найден
            Item item = new Item(); // переменная для хранения
наименования склада
            if (Find(article, container, ref item)) // ищем элемент
            {
                r = item.Wholesale_remove(amount); // если возможно,
убираем нужное количество, если нет -- верн'м false
                if (r)
                {
                    Ship ship = new Ship(article, amount, where);
                    ships.Add(ship);
                }
                else
                {
                    Ship ship = new Ship(article,
amount*item.pack_amount, where);
                    waiting_list.Add(ship);
                }
            }
            return r;
        }

        public void Add_Item(float s = 0, uint pa = 0, uint m = 0,
double price = 0.0,
            uint due_date = 0, String article = "", ContainerType
container = 0,
            Unit unit = 0, uint amount = 0) // добавление элемента
на склад
        {
            // созда'м элемент и добавляем его на свой склад
            Item item = new Item(s, pa, m, price, due_date,article,
container, unit, amount);
```

```
            switch (container)
            {
                case ContainerType.COLD:
                    {
                        cold_storage.Add(item);
                        break;
                    }
                case ContainerType.WET:
                    {
                        wet_storage.Add(item);
                        break;
                    }
                case ContainerType.DRY:
                    {
                        dry_storage.Add(item);
                        break;
                    }
            }
        }

        public void Add_Item(Item item) // добавление элемента на
склад
        {
            // добавляем элемент на свой склад
            switch (item.container)
            {
                case ContainerType.COLD:
                    {
                        cold_storage.Add(item);
                        return;
                    }
                case ContainerType.DRY:
                    {
                        dry_storage.Add(item);
                        return;
                    }
                case ContainerType.WET:
                    {
                        wet_storage.Add(item);
                        return;
                    }
            }
        }

        public bool Delete_Item(String article, ContainerType
container) // удаление наименования из списка хранения
        {
            Item item = new Item();
            if (Find(article, container, ref item)) // если элемент
найден
            { // удаляем его из своего контейнера
                switch (container)
                {
```

```
                    case ContainerType.COLD:
                        {
                            cold_storage.Remove(item);
                            return true;
                        }
                    case ContainerType.WET:
                        {
                            wet_storage.Remove(item);
                            return true;
                        }
                    case ContainerType.DRY:
                        {
                            dry_storage.Remove(item);
                            return true;
                        }
                }
            }
            return false; // если элемент не был найден
        }

        public void Sell_all()
        {
            int shit = waiting_list.Count;
            for (int i = 0; i < shit; i++)
            {
                WS_Sell(waiting_list[i]);
            }
            for (int i = 0; i < order.Count; i++)
            {
                WS_Sell(order[i]);
            }
            Add_all();
        }

        public void Add_all()
        {
            Random smol = new Random();
            foreach(Item i in cold_storage)
            {
                i.wholesale_amount += (uint)smol.Next(100);
            }
            foreach (Item i in wet_storage)
            {
                i.wholesale_amount += (uint)smol.Next(100);
            }
            foreach (Item i in dry_storage)
            {
                i.wholesale_amount += (uint)smol.Next(100);
            }
        }
    }
}
```

Sourse.cs

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

namespace imitation
{
    enum ContainerType // тип хранения
    {
        COLD, // холодильник
        WET, // сухой склад
        DRY, // влажный склад
    }

    enum Unit // единица измерения
    {
        piece, // штуки
        kg, // килограммы
        gram, // граммы
        pack, // упаковки
        liter, // литры
        ml, // милилитры

    }

    abstract class Piece // единица -- оптовая упаковка
    {
        public Unit unit; // единица измерения -- в них количество
товара в 1 розничной упаковке
        public float size; // количество тоара в единицах unit в 1
розничной упаковке
        public uint pack_amount; // количество розничных упаковок
    }

    class Item : Piece // наименование
    {
        public uint wholesale_amount; // количество оптовых упаковок
        public uint max_ws_amount; // максимальное число упаковок на
складе
        public double price; // цена за единицу товара
        public uint due_date; // [1;366] срок годности
        public String article; // артикул товара (строка потому что
артикул может содержать буквы и знаки препинания)
        public ContainerType container; // место хранения --
сухой/влажный склад или холодильник

        public Item(float s = 0, uint pa = 0, uint m = 0, double
price = 0.0, uint due_date = 0, String article = "",
            ContainerType container = 0, Unit unit = 0, uint amount
= 0)
        {
```

```
            this.unit = unit; // инициализация поля единица
измерения
            this.size = s;
            this.pack_amount = pa;
            this.wholesale_amount = amount; // инициализация поля
количество
            this.max_ws_amount = m;
            this.price = price; // инициализация поля цена
            this.due_date = due_date; // инициализация поля срок
годности
            this.article = article; // инициализация поля артикул
            this.container = container; // инициализация поля место
хранения

        }

        public bool Retail_remove(uint a) // уменьшает количество на
заданное значение, если это возможно
        {
            if (this.wholesale_amount*pack_amount < a)
            {
                return false;
            }
            if(a%pack_amount == 0)
            {
                this.wholesale_amount -= (a/pack_amount);
            }
            else if (a % pack_amount < pack_amount/2)
            {
                this.wholesale_amount -= (a / pack_amount);
            }
            else
            {
                this.wholesale_amount -= (a / pack_amount + 1);
            }
            return true;
        }

        public bool Wholesale_remove(uint a)
        {
            if(a > wholesale_amount)
            {
                return false;
            }
            wholesale_amount -= a;
            return true;
        }

        public void Add(uint a) // увеличивает количество на
заданное значение
        {
            this.wholesale_amount += a;
        }
```

```
    }

    class Ship
    {
        public String article;
        public uint wholesale_amount;
        public uint where;

        public Ship(string s = "", uint a = 0, uint w = 0)
        {
            this.article = s;
            this.wholesale_amount = a;
            this.where = w;
        }
    }

    interface IBill
    {
        protected static double bill;

        protected static void add_money(double a)
        {
            bill += a;
        }
        public static double get_money()
        {
            return bill;
        }
    }
}
```

Program.cs

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

namespace imitation
{
   class imitation
    {
        static void Main(string[] args)
        {
            Random smol = new Random();
            int l = smol.Next(10, 30);
            Storage myStorage = new Storage();
            int goods_amount = smol.Next(12, 20), // число товаров
на складе, от 12 до 20
                points_amount = smol.Next(3, 9), // число торговых
точек от 3 до 9
                ships_amount = smol.Next(10);
            Console.WriteLine("Точек: {0}, наименований: {1},
доставок {2}", points_amount, goods_amount, ships_amount);
```

```
            for (int i = 0; i < goods_amount; i++)
            { // заполнение склада случайными элементами
                myStorage.Add_Item(smol.Next(), (uint)smol.Next(),
(uint)smol.Next(), (uint)smol.Next(),
                (uint)smol.Next(), Convert.ToString(i),
(ContainerType)(smol.Next() % 3),
                (Unit)(smol.Next() % 6), (uint)smol.Next(10, 200));
// Создание элемента со случайными значениями полей
            }
            for (int k= 1; k<=l; k++)
            {
                for (int i = 0; i < points_amount; i++) // для
каждой точки
                {
                    for (int j = 0; j < ships_amount; j++) // свое
число заказов
                    {
                        Ship s = new
Ship(Convert.ToString(smol.Next(0, goods_amount)),
                            (uint)(smol.Next(10)),
(uint)(smol.Next(points_amount)));
                        myStorage.order.Add(s); // заполняем очередь
заказов
                    }
                }
                Console.WriteLine("день {0} из {1}.", k, l);
                Console.WriteLine("очередь заказов из {0} товаров
для {1} точек", myStorage.order.Count, points_amount);
                myStorage.Sell_all();
                Console.WriteLine("доставок на завтра: {0}, в листе
ожидания: {1}", myStorage.ships.Count,
myStorage.waiting_list.Count);
                ships_amount = smol.Next(10);
                Thread.Sleep(3000);
            }
        }
    }
}
```

ЗАКЛЮЧЕНИЕ

Поставленные задачи были выполнены, разработаны три приложения. Приложения работают корректно и соответствуют поставленным задачам и корректно выполняют назначение.

Таким образом, были закреплены на практике навыки разработки приложений на основе объектно-ориентированного подхода, а также раскрыты возможности языка C#.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кирсанов М. Н. Графы в MAPLE Задачи, алгоритмы, программы / М.: ФИЗМАТЛИТ, 2007

2. Иванов Б.Н.Дискретная математика / М.: ЛБЗ, 2002

3. C# и платформа .NET: монография / Э. Троелсен; Пер. с англ. Р. Михеева. - СПб.: Питер, 2002. - 795 с.: ил., табл. - (Б-ка программиста).

4. http://msdn.microsoft.com/ru-ru/

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

6. Унифицированный процесс разработки программного обеспечения: монография / А. Якобсон, Г. Буч, Дж. Рамбо; пер. с англ. В. Горбунков. - СПб.: Питер, 2002. - 492 с.