**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 85 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 5](#_Toc72928055)

[ПЕРВЫЙ РАЗДЕЛ. СОЗДАНИЕ ИЕРАРХИИ КЛАССОВ 6](#_Toc72928056)

[Цель 6](#_Toc72928057)

[Задание 6](#_Toc72928058)

[Теоретический аспект задачи 6](#_Toc72928059)

[Формализация задачи 7](#_Toc72928060)

[Спецификация программы 8](#_Toc72928061)

[Руководство пользователя 10](#_Toc72928062)

[Руководство программиста 11](#_Toc72928063)

[Контрольные примеры 11](#_Toc72928064)

[Листинг 12](#_Toc72928065)

[Выводы 13](#_Toc72928066)

[ВТОРОЙ РАЗДЕЛ. МАКСИМАЛЬНЫЙ ПОТОК В СЕТИ 15](#_Toc72928067)

[Цель 15](#_Toc72928068)

[Задание 15](#_Toc72928069)

[Теоретический аспект задачи 15](#_Toc72928070)

[Формализация задачи 18](#_Toc72928071)

[Руководство пользователя 18](#_Toc72928072)

[Руководство программиста 18](#_Toc72928073)

[Контрольный пример 19](#_Toc72928074)

[Листинг 20](#_Toc72928075)

[Вывод 20](#_Toc72928076)

[ТРЕТИЙ РАЗДЕЛ. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ 21](#_Toc72928077)

[Цель 21](#_Toc72928078)

[Задание 21](#_Toc72928079)

[Описание имитационной модели 22](#_Toc72928080)

[Формализация задачи 24](#_Toc72928081)

[Спецификация программы 26](#_Toc72928082)

[Руководство пользователя 28](#_Toc72928083)

[Руководство программиста 29](#_Toc72928084)

[Контрольный пример 30](#_Toc72928085)

[Листинг 31](#_Toc72928086)

[ЗАКЛЮЧЕНИЕ 37](#_Toc72928087)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38](#_Toc72928088)

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.

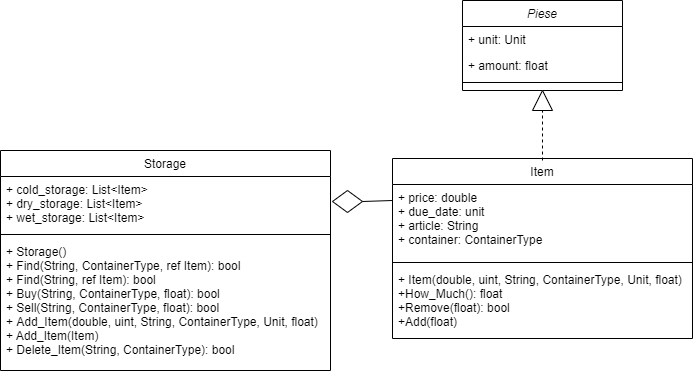


Рис. 1.1. Диаграмма классов.

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.

Спецификация программы

Описания методов и полей классов представлены в таблицах 1.1.а- 1.3.а.

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 |

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.

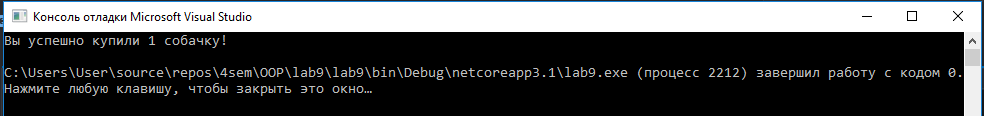


Рис. 1.2. Пример работы программы.

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.

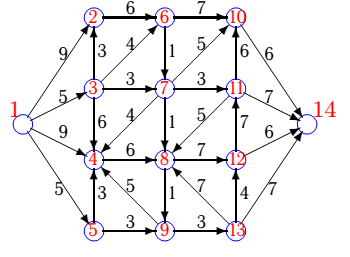
В качестве контрольного примера использовать задания по вариантам. 

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.

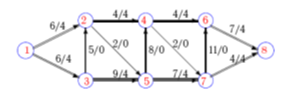


Рис. 2.2

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).

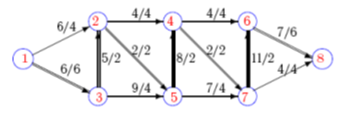


Рис. 2.3

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].

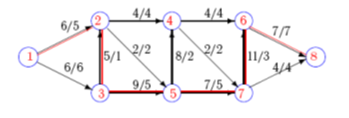


Рис. 2.4

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.

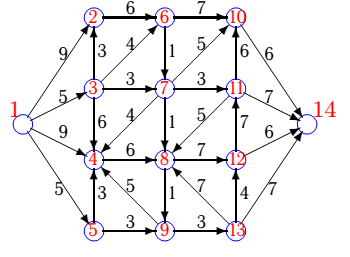


Рисунок 2.5. Граф, используемый для контрольного примера.

На рисунке 2.6 представлен вывод программы.

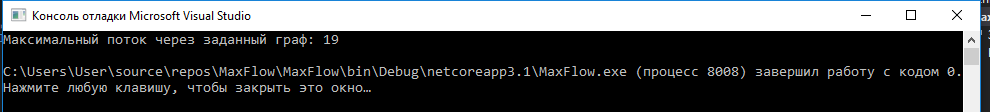


Рис. 2.6. Результат работы программы.

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 },

{ 0,0,0,0,0,6,0,0,0,0,0,0,0,0 },

{ 0,3,0,6,0,4,3,0,0,0,0,0,0,0 },

{ 0,0,0,0,0,0,0,6,0,0,0,0,0,0 },

{ 0,0,0,3,0,0,0,0,3,0,0,0,0,0 },

{ 0,0,0,0,0,0,1,0,0,7,0,0,0,0 },

{ 0,0,0,0,0,0,0,1,0,5,3,0,0,0 },

{ 0,0,0,0,0,0,0,0,1,0,0,7,0,0 },

{ 0,0,0,5,0,0,0,0,0,0,0,0,3,0 },

{ 0,0,0,0,0,0,0,0,0,0,0,0,0,6 },

{ 0,0,0,0,0,0,0,5,0,0,0,0,0,7 },

{ 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≤ K ≤20) продуктовых товаров, обслуживает М (3≤ М ≤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≤ N ≤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, М, 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, уценка товаров.

*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 (шаг периода моделирования задается пользователем). Таким образом, разработанная модель является дискретной (переменные изменяются дискретно в определённые моменты имитационного времени).

Формализация задачи

Диаграмма используемых классов представлена на рисунке 4.1.

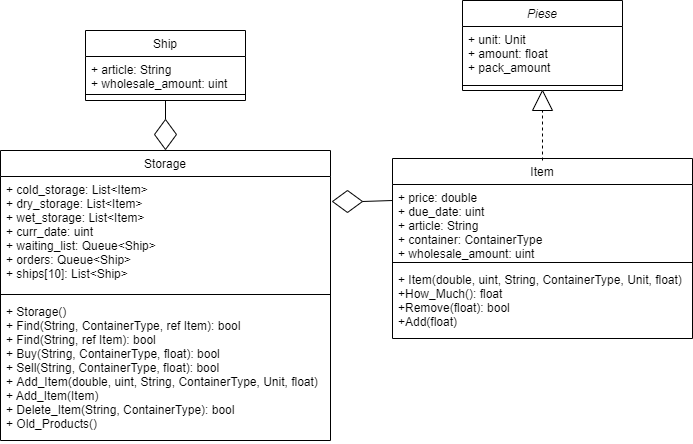


Рисунок 3.1 Диаграмма классов

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.

Спецификация программы

*Класс Act*

Таблица 4.1.а – описание методов класса *Act*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| GetDuration | Возвращает длительность услуги по номеру | int | public | Номер услуги |
| GetPrice | Возвращает стоимость услуги по номеру | int | public | Номер услуги |
| Act | Конструктор | - | public | Номер услуги, длительность услуги |

Таблица 4.1.б – описание полей класса *Act*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| number | int | private | Номер услуги |
| duration | int | private | Длительность улуги |

*Класс Gear*

Таблица 4.2.а – описание методов класса *Gear*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Gear | Конструктор | - | public | bool – занял ли мастер, int – прибыль по умолчанию |
| timeReduce | Множитель времени услуги в зависимости от количества мастеров | double | public | Список мастеров |

Таблица 4.2.б – описание полей класса *Gear*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| free | bool | private | Занят ли мастер |
| profit | int | private | Общая прибыль мастера |

*Класс Model*

Таблица 4.3.а – описание методов класса *Model*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Model | Конструктор | - | public | Количество мастеров в каждом боксе, время между появлением заявок |
| Y | Зависимость появления заявки от времен и дня | double | private | Текущее время |
| Play | Запуск модели | void | public | Текущее время, шаг периода моделирования |
| outBox | Вывод очередей боксов | void | public | Бокс, список |
| outCurrent | Вывод текущей заявки | void | public | Бокс, текстбокс |

Таблица 4.3.б – описание полей класса *Model*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| box1 | Queue | public | Бокс1 |
| box2 | Queue | public | Бокс2 |
| box3 | Queue | public | Бокс3 |
| box4 | Queue | public | Бокс4 |
| profit | int | public | Общая прибыль |
| totalRequest | int | public | Всего заявок обслужено |
| appearTime | int | private | Время между заявками |
| rand | Random | public | Рандомайзер |

*Класс Queue*

Таблица 4.4.а – описание методов класса *Queue*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Queue | Конструктор | - | public | Номер бокса, количество мастеров |
| Add | Добавление заявки в бокс | void | public | Заявка, услуга |
| Action | Обработка заявки | void | public | Шаг периода моделирования |

Таблица 4.4.б – описание полей класса *Queue*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| list | List<Act> | private | Список услуг |
| gear | List<Gear> | private | Список мастеров |
| requestList | List<Request> | private | Список заявок |
| requestActList | List<Act> | private | Список услуг, соответствующих заявкам |
| currentRequest | Request | private | Текущая заявка |
| currentAct | Act | private | Текущая услуга |

*Класс Request*

Таблица 4.5.а – описание методов класса *Request*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Назначение | Возвращаемый тип | Модификатор доступа | Входные параметры |
| Request | конструктор по умолчанию | - | public | - |
| Request | Конструктор | - | public | время появления заявки |
| GenerateOrder | Создание заявки | List<Act> | private | - |
| HasFree | Проверка, есть ли в очереди свободные заявки | int | public | Список - очередь |

Таблица 4.5.б – описание полей класса *Request*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя | Тип | Модификатор  доступа | Назначение |
| name | string | private | Имя заявки |
| free | bool | private | Заявка свободна |
| list | List<Act> | private | Список услуг |

Описания методов и полей классов представлены в таблицах 4.1.а – 4.5.б.

Руководство пользователя

*Назначение программы*

Программа реализует имитацию работы автосервиса с четырьмя боксам обслуживание, каждый из которых оказывает определенный виды услуг.

*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*

После запуска программы пользователю необходимо заполнить поля количества мастеров в каждом боксе, шаг периода моделирования, количество отображаемых шагов в секунду и среднее время между появлением клиентов, после этого можно запустить имитационную модель с заданными параметрами нажатием кнопки Старт. Результаты моделирования можно сохранить в текстовый документ с помощью кнопки сохранить.

*Сообщение пользователю*

Интерфейс включает в себя поля для ввода исходных данных, поля отображения текущей ситуации в каждом боксе, текущее время модели, а также поля общей статистики

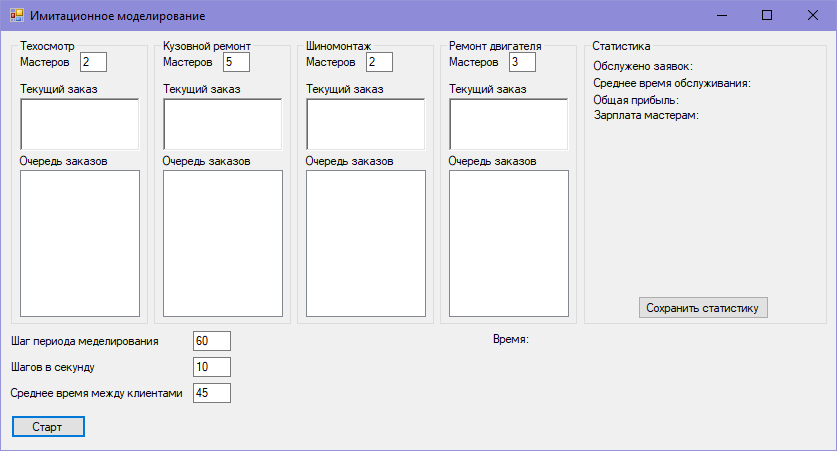


Рисунок 4.3. Интерфейс приложения

Интерфейс разработанного приложения представлен на рисунке 4.3.

Руководство программиста

*Назначение и условие выполнения программы*

Программа реализует имитацию работы автосервиса с четырьмя боксам обслуживание, каждый из которых оказывает определенный виды услуг. 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 входных данных выступают количество мастеров в каждом из боксов, шаг периода моделирования, скорость обновления текущей ситуации и среднее время между появлением клиентов.

В качестве выходные данных выступает протокол выполнения программы, выведенный на экран, а также сохраненная статистика моделирования.

*Сообщения*

После выполнения программы доступна статистика выполненной программы.

Контрольный пример

Результаты прогонов имитационной модели с различными параметрами представлены в таблице 4.6.

Таблица 4.6. Результаты прогонов имитационной модели

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Шаг периода моделирования | Среднее время между клиентами | Количество мастеров | | | | Обслужено заявок | Среднее время обслуживания | Общая прибыль | Зарплата мастерам |
| Техосмотр | Кузовной ремонт | Шиномонтаж | Ремонт двигателя |
| 60 | 60 | 2 | 2 | 2 | 2 | 63 | 54 | 82980 | 56000 |
| 60 | 60 | 7 | 7 | 7 | 7 | 94 | 60 | 138635 | 196000 |
| 30 | 30 | 2 | 5 | 2 | 3 | 86 | 66 | 90580 | 84000 |
| 60 | 45 | 2 | 4 | 2 | 2 | 81 | 66 | 97705 | 70000 |
| 60 | 45 | 2 | 4 | 2 | 2 | 67 | 54 | 146390 | 70000 |
| 60 | 45 | 2 | 4 | 2 | 2 | 79 | 60 | 138720 | 70000 |
| 60 | 45 | 2 | 4 | 2 | 2 | 73 | 60 | 99285 | 70000 |

Многократные прогоны позволили определить оптимальное количество мастеров в каждом боксе – в боксе техосмотра – 2 мастера, в боксе кузовного ремонта – 4-5 мастеров, в боксе шиномонтажа – 2 мастера, в боксе ремонта двигателя 2-3 мастера. С данными параметрами зарплата мастерам не превышает общую прибыль и количество обслуженных заявок не слишком мало.

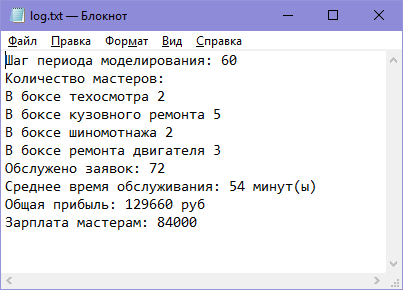
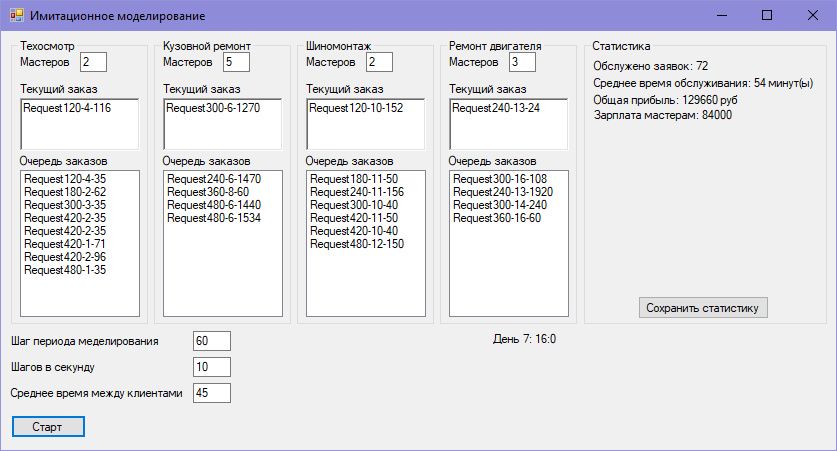


Рисунок 4.5 Пример работы программы

Листинг

**public class Request**

{

private string name;

private bool free;

private List<Act> list;

public string Name { get => name; set => name = value; }

public bool Free { get => free; set => free = value; }

public List<Act> List { get => list; set => list = value; }

public Request(int n)

{

Name = "Request" + n;

List = GenerateOrder();

Free = true;

}

public Request()

{

}

List<Act> GenerateOrder()

{

int count = Model.rand.Next(1, 6);

List<Act> order = new List<Act>(count);

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

{

int num = Model.rand.Next(1, 17);

order.Add(new Act(num, Act.GetDuration(num)));

if (Model.rand.NextDouble() > 0.5)

order[i].Duration += Model.rand.Next(120);

}

return order;

}

public static int HasFree(List<Request> list)

{

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

if (list[i].Free)

return i;

return -1;

}

}

**class Model**

{

public static Queue box1;

public static Queue box2;

public static Queue box3;

public static Queue box4;

public static int profit;

public static int totalRequest;

int appearTime;

public static Random rand = new Random();

public Model(int \_gear1, int \_gear2, int \_gear3, int \_gear4, int \_appearTime)

{

box1 = new Queue(1, \_gear1);

box2 = new Queue(2, \_gear2);

box3 = new Queue(3, \_gear3);

box4 = new Queue(4, \_gear4);

profit = 0;

totalRequest = 0;

appearTime = \_appearTime;

appearTime = 0;

}

static double Y(int x)

{

double result;

result = -(1.0 / 129600.0);

result \*= Math.Pow((x - 360.0), 2);

result += 1.0;

return result;

}

public void Play(int \_time, int \_step)

{

int tempTime;

int lastTime = 0;

if (rand.Next(1) == 1)

tempTime = appearTime - rand.Next(1, 16);

else

tempTime = appearTime + rand.Next(1, 16);

if (rand.NextDouble() < Y(\_time) && (lastTime + tempTime) < \_time)

{

Request request = new Request(\_time);

lastTime = \_time;

for (int i = 0; i < request.List.Capacity; i++)

{

if (request.List[i].Number <= 4 && request.List[i].Number >= 1 && box1.RequestList.Count <= 10)

box1.Add(ref request, request.List[i]);

else if (request.List[i].Number <= 8 && request.List[i].Number >= 5 && box2.RequestList.Count <= 3)

box2.Add(ref request, request.List[i]);

else if (request.List[i].Number <= 12 && request.List[i].Number >= 9 && box3.RequestList.Count <= 10)

box3.Add(ref request, request.List[i]);

else if (request.List[i].Number <= 16 && request.List[i].Number >= 13 && box4.RequestList.Count <= 3)

box4.Add(ref request, request.List[i]);

}

}

box1.Action(\_step);

box2.Action(\_step);

box3.Action(\_step);

box4.Action(\_step);

}

public static void outBox(Queue box, ListBox list)

{

list.Items.Clear();

string obj;

for (int i = 0; i < box.RequestList.Count; i++)

{

obj = box.RequestList[i].Name + "-" + box.RequestActList[i].Number + "-" + box.RequestActList[i].Duration;

list.Items.Add(obj);

}

}

public static void outCurrent(Queue box, RichTextBox textbox)

{

if (!box.CurrentRequest.Free)

{

textbox.Clear();

textbox.Text = box.CurrentRequest.Name + "-" + box.CurrentAct.Number + "-" + box.CurrentAct.Duration;

}

else

textbox.Clear();

}

}

**public class Queue**

{

List<Act> list;

List<Gear> gear;

private List<Request> requestList;

private List<Act> requestActList;

private Request currentRequest;

private Act currentAct;

public List<Request> RequestList { get => requestList; set => requestList = value; }

public List<Act> RequestActList { get => requestActList; set => requestActList = value; }

public Request CurrentRequest { get => currentRequest; set => currentRequest = value; }

public Act CurrentAct { get => currentAct; set => currentAct = value; }

public Queue(int n, int gearCount)

{

CurrentRequest = new Request { Free = true };

gear = new List<Gear>(gearCount);

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

gear.Add(new Gear(true, 0));

list = new List<Act>();

RequestList = new List<Request>();

RequestActList = new List<Act>();

switch (n)

{

case 1:

{

list.Add(new Act(1, 35));

list.Add(new Act(2, 35));

list.Add(new Act(3, 35));

list.Add(new Act(4, 35));

break;

}

case 2:

{

list.Add(new Act(5, 300));

list.Add(new Act(4, 1440));

list.Add(new Act(7, 2400));

list.Add(new Act(8, 60));

break;

}

case 3:

{

list.Add(new Act(9, 100));

list.Add(new Act(10, 40));

list.Add(new Act(11, 50));

list.Add(new Act(12, 150));

break;

}

case 4:

{

list.Add(new Act(9, 1920));

list.Add(new Act(10, 240));

list.Add(new Act(11, 60));

list.Add(new Act(12, 60));

break;

}

}

}

public void Add(ref Request request, Act act)

{

RequestList.Add(request);

RequestActList.Add(act);

}

public void Action(int step)

{

int i;

if (RequestList.Capacity != 0 && CurrentRequest.Free == true && (i = Request.HasFree(RequestList)) >= 0)

{

RequestList[i].Free = false;

CurrentRequest = RequestList[i];

CurrentAct = RequestActList[i];

if (Model.rand.NextDouble() > 0.3)

CurrentAct.Duration += currentAct.Duration/10\*2;

CurrentAct.Duration \*= Convert.ToInt32( Gear.timeReduce(this.gear));

RequestList.Remove(CurrentRequest);

RequestActList.Remove(CurrentAct);

}

else if (!CurrentRequest.Free)

{

CurrentAct.Duration -= step;

if (CurrentAct.Duration <= 0)

{

Model.totalRequest++;

CurrentRequest.Free = true;

Model.profit += Act.GetPrice(CurrentAct.Number);

this.Action(step);

}

}

}

}

**class Gear**

{

bool free;

int profit;

public Gear(bool fr, int prof)

{

free = fr;

profit = prof;

}

public static double timeReduce(List<Gear> gear)

{

double k = 1;

for (int i = 2; i<gear.Count;i++)

{

k -= 0.06;

}

return k;

}

}

**public class Act**

{

private int number;

private int duration;

public int Number { get => number; set => number = value; }

public int Duration { get => duration; set => duration = value; }

public static int GetDuration(int n)

{

switch (n)

{

case 1: return 35;

case 2: return 35;

case 3: return 35;

case 4: return 35;

case 5: return 300;

case 6: return 1440;

case 7: return 2400;

case 8: return 60;

case 9: return 100;

case 10: return 40;

case 11: return 50;

case 12: return 150;

case 13: return 1920;

case 14: return 240;

case 15: return 60;

case 16: return 60;

default: return -1;

}

}

public static int GetPrice(int n)

{

switch (n)

{

case 1: return 800;

case 2: return 940;

case 3: return 290;

case 4: return 500;

case 5: return 715;

case 6: return 7300;

case 7: return 51000;

case 8: return 600;

case 9: return 2000;

case 10: return 200;

case 11: return 300;

case 12: return 800;

case 13: return 21000;

case 14: return 1500;

case 15: return 4000;

case 16: return 800;

default: return -1;

}

}

public Act(int \_num, int \_dur)

{

number = \_num;

duration = \_dur;

}

}

ЗАКЛЮЧЕНИЕ

Поставленные задачи были выполнены, разработаны пять приложений с графическим пользовательским интерфейсом. Приложения работают корректно и соответствуют поставленным задачам и корректному графическому отображению в окне программы при помощи соответствующих инструментов.

Таким образом, были закреплены на практике навыки разработки приложений на основе объектно-ориентированного подхода и создания графического пользовательского интерфейса с помощью платформы Windows Forms, а также раскрыты возможности языка C#.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Транспортное средство [Электронный ресурс] // Википедия: свободная энцикл. – URL: https://ru.wikipedia.org/wiki/Транспортное\_средство (дата обращения: 09.11.2017).
2. Кирсанов М. Н. Графы в MAPLE Задачи, алгоритмы, программы / М.: ФИЗМАТЛИТ, 2007
3. Иванов Б.Н.Дискретная математика / М.: ЛБЗ, 2002
4. Структура имитационной модели автосервиса [Электронный ресурс] // Современные проблемы транспортного комплекса России – URL: https://transcience.ru/index.php/MPRTC/article/view/2222-9396-2016-6-2-46-50# (дата обращения: 13.11.2017)
5. Вапник В.Н. Теория распознавания образов / В.Н. Вапник, А.Я. Червоненкис - М.: Изд. Наука, 1974. – 416 с.
6. C# и платформа .NET: монография / Э. Троелсен; Пер. с англ. Р. Михеева. - СПб.: Питер, 2002. - 795 с.: ил., табл. - (Б-ка программиста).
7. http://msdn.microsoft.com/ru-ru/
8. Горячев А.В., Кравчук Д.К., Новакова Н.Е. Объектно-ориентированное моделирование. Учеб. Пособие. СПб.: Изд-во СПбГЭТУ “ЛЭТИ”, 2010.
9. Унифицированный процесс разработки программного обеспечения: монография / А. Якобсон, Г. Буч, Дж. Рамбо; пер. с англ. В. Горбунков. - СПб.: Питер, 2002. - 492 с.