УВО «Университет Управления «ТИСБИ»

Факультет информационных технологий

Кафедра информационных технологий

**Курсовая работа**

по дисциплине «Объектно-ориентированное программирование»

на тему: «Разработка объектной программы для обработки данных о пациентах платной медицинской клиники»

Выполнил: студент гр. П-011 Сучёв Н.Е.

Проверил: ст. преп. Якунина Е.А.

Казань 2022

Оглавление

[Постановка задачи 3](#_Toc105256728)

[1. Описание используемых структур данных с алгоритмами выполнения основных операций 5](#_Toc105256729)

[1.1 Очередь на основе массива. 5](#_Toc105256730)

[1.2 Адресный замкнутый (кольцевой) однонаправленный список без заголовка. 6](#_Toc105256731)

[2. Краткие сведения об объектном подходе 10](#_Toc105256732)

[3. Формализованное описание разработанных классов 16](#_Toc105256733)

[4. Описание демонстрационного модуля с характеристикой использованных стандартных компонентов и списком реализованных обработчиков 22](#_Toc105256734)

[5. Описание структуры проекта в соответствии с использованным инструментом разработки 28](#_Toc105256735)

[Листинг программы 32](#_Toc105256736)

Постановка задачи

**Цели работы:** отработка навыков курсов «объектно-ориентированное программирование» и «структуры и алгоритмы обработки данных».

**Постановка задачи:** Медицинский центр ведёт учёт услуг с помощью списка своих пациентов с указанием уникальной фамилии пациента, числа посещений и суммарную стоимость оказанных услуг. Для каждого пациента создаётся список его посещений с указанием уникальной даты посещения, вида услуги и её стоимости.

Разработка программы включает в себя:

* определение необходимых объектов и способов их взаимодействия;
* формальное описание объектов в виде классов;
* программную реализацию всех необходимых методов, включая корректировку вкладов и подсчет текущего суммарного объема всех вкладов;
* всестороннее тестирование методов с помощью консольного (при разработке) и оконного (в окончательном варианте) приложения.

Для объединения пациентов используется структура данных в виде очереди на основе динамического массива со сдвигом элементов. Для объединения посещений каждого пациента используется структура данных в виде адресного замкнутого (кольцевого) неупорядоченного однонаправленного списка с заголовком.

Разработка выполняется с учетом следующих требований:

1. имена классов, свойств и методов должны носить содержательный смысл и соответствовать информационной задаче
2. обязательное соблюдение принципа инкапсуляции – использование в классах только закрытых свойств и реализация необходимого набора методов доступа
3. наличие двух методов для сохранения всей объектной структуры во внешнем файле с обратной загрузкой, при этом стандартные механизмы сериализации разрешается использовать только как дополнение к самостоятельно реализованным методам
4. тестовое оконное приложение должно обладать удобным пользовательским интерфейсом с контролем вводимых данных и отображением текущего состояния объектной структуры с помощью списковых или табличных компонентов
5. стандартные контейнеры/коллекции (включая обобщенные классы) разрешается использовать только как дополнение к самостоятельно разработанным классам
6. в качестве языка разработки разрешается использовать Java, С, C++, Object/Free Pascal и соответствующие инструменты быстрой разработки приложений.
7. **Описание используемых структур данных с алгоритмами выполнения основных операций**

1.1 Очередь на основе массива.

Очередь – структура данных типа «список», позволяющая добавлять элементы лишь в конец списка, и извлекать их из его начала. Она функционирует по принципу FIFO (First In, First Out — «первым пришёл — первым вышел»), для которого характерно, что все элементы a1, a2, …, an-1, an, добавленные раньше элемента an+1, должны быть удалены прежде, чем будет удален элемент an+1. Также очередь может быть определена как частный случай односвязного списка, который обслуживает элементы в порядке их поступления. Как и в «живой» очереди, здесь первым будет обслужен тот, кто пришел первым.

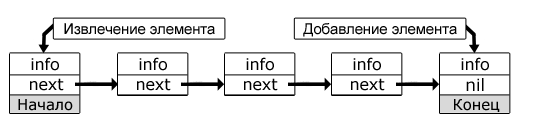


Рис. 1 Очередь.

Стандартный набор операций (часто у разных авторов он не идентичен), выполняемых над очередями, совпадает с тем, что используется при обработке стеков:

* добавление элемента;
* удаление элемента;
* чтение первого элемента.

Только, если в отношении стека в момент добавления или удаления элемента допустимо задействование лишь его вершины, то касательно очереди эти две операции должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала. Далее, при реализации интерфейса очереди, список стандартных операций будет расширен.

Выделяют два способа программной реализации очереди. Первый из них основан на базе массива, а второй на базе указателей (связного списка). Первый способ – статический, т. к. очередь представляется в виде простого статического массива, второй – динамический.

1.2 Адресный замкнутый (кольцевой) однонаправленный список без заголовка.

Каждый узел однонаправленного (односвязного) кольцевого списка содержит одно поле ссылочной переменной на следующий узел и поле с полезными данными. Односвязный циклический список можно представить следующим образом:



Рис. 2 Блок схема добавления элемента в таблицу.

Узел однонаправленного кольцевого списка можно представить в виде структуры, аналогичной односвязному линейному списку.

class Node**{**

public**:**

Operation data**;**

Node **\*** pNext**;**

Node**(**Operation data**,** Node**\*** start**){**

**this->**data **=** data**;**

**this->**pNext **=** start**;**

**}**

**};**

Основные действия, производимые над элементами ОЦС:

* Инициализация списка
* Добавление узла в список
* Удаление узла из списка
* Вывод элементов списка
* Взаимообмен двух узлов списка

Поскольку список является циклическим, реализация отдельной функции для удаления корня списка не требуется. Инициализация списка

Рис. 3 Узел списка

Инициализация списка предназначена для создания корневого узла списка, у которого поле указателя на следующий элемент содержит адрес самого корневого элемента.

// конструктор списка

ListOfOperation**::**ListOfOperation**(**QString name**){**

ListOfOperation**::**set\_name**(**name**);**

operations\_counter **=** 0**;**

start **=** **nullptr;**

last **=** **nullptr;**

**}**

**// первое добавление**

start **=** **new** Node**(**data**,** start**);**

last **=** start**;**

last**->**pNext **=** start**;**

operations\_counter**++;**

Добавление узла в ОЦС

Функция добавления узла в список принимает два аргумента:

* Указатель на элемент, после которого происходит добавление
* Данные для добавляемого элемента.

Процедуру добавления элемента можно отобразить следующей

схемой:



Рис. 4: Добавление в список

Добавление элемента в ОЦС включает в себя следующие этапы:

* создание добавляемого узла и заполнение его поля данных;
* переустановка указателя узла, предшествующего добавляемому, на добавляемый узел;
* установка указателя добавляемого узла на следующий узел (тот, на который указывал предшествующий узел).

Таким образом, функция добавления узла в ОЦС имеет вид, полностью аналогичный функции добавления узла в односвязный линейный список:

// добавляем в начало списка новые по дате транзакции

**}** **else** **if(**start**->**data**.**get\_date**()** **<** date**){**

temp **=** start**;**

start **=** **new** Node**(**data**,** temp**);**

last**->**pNext **=** start**;**

operations\_counter**++;**

// добавляем в конец списка старые по дате транзакции

**}** **else** **if** **(**last**->**data**.**get\_date**()** **>=** date**)** **{**

temp **=** last**;**

last **=** **new** Node**(**data**,** start**);**

temp**->**pNext **=** last**;**

operations\_counter**++;**

// добавление в середину перед датой растее

**}** **else{**

temp **=** start**;**

Node**\*** temp\_last**;** // [2] [8]

**while(**temp**->**data**.**get\_date**()** **>** date**){**

temp\_last **=** temp**;**

temp **=** temp**->**pNext**;**

**}**

Node**\*** create **=** **new** Node**(**data**,** temp**);**

temp\_last**->**pNext **=** create**;**

operations\_counter**++;**

**}**

Возвращаемым значением функции является адрес добавленного узла.

Удаление узла ОЦС

В качестве аргументов функции удаления узла ОЦС передается указатель на удаляемый узел. Поскольку список циклический, нет необходимости передавать указатель на корень списка. Функция возвращает указатель на узел, следующий за удаляемым элементом.

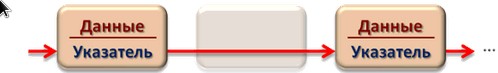
Удаление узла может быть представлено следующей схемой:

Рис. 5: Удаление узла

Удаление узла ОЦС включает в себя следующие этапы:

* установка указателя предыдущего узла на узел, следующий за удаляемым;
* освобождение памяти удаляемого узла.

**if(**operations\_counter **==** 1**){**

**delete** start**;**

operations\_counter **=** 0**;**

start **=** **nullptr;**

last **=** **nullptr;**

**return;**

**}**

temp **=** start**;**

last**->**pNext **=** start**->**pNext**;**

start **=** start**->**pNext**;**

**delete** temp**;**

operations\_counter**--;**

1. Краткие сведения об объектном подходе

Класс (class) - элемент программы, который описывает какой-то тип данных. Класс описывает шаблон для создания объектов, как правило, указывает переменные этого объекта и действия, которые можно выполнять применимо к объекту.

Экземпляр класса (instance) - объект, который является представителем класса.

Метод (method) - функция, которая определена внутри класса и описывает какое-то действие, которое поддерживает класс

Переменная экземпляра (instance variable, а иногда и instance attribute) - данные, которые относятся к объекту

Переменная класса (class variable) - данные, которые относятся к классу и разделяются всеми экземплярами класса

Атрибут экземпляра (instance attribute) - переменные и методы, которые относятся к объектам (экземплярам) созданным на основании класса. У каждого объекта есть своя копия атрибутов.

Пример из реальной жизни в стиле ООП:

Проект дома - это класс

Конкретный дом, который был построен по проекту - экземпляр класса

Такие особенности как цвет дома, количество окон - переменные экземпляра, то есть конкретного дома

Дом можно продать, перекрасить, отремонтировать - это методы

Все основанные на объектах языки (C#, Java, С++, Smalltalk, Visual Basic и т.п.) должны отвечать трем основным принципам объектно-ориентированного программирования (ООП), которые перечислены ниже:

Инкапсуляция

Как данный язык скрывает детали внутренней реализации объектов и предохраняет целостность данных?

Наследование

Как данный язык стимулирует многократное использование кода?

Полиморфизм

Как данный язык позволяет трактовать связанные объекты сходным образом?

Прежде чем погрузиться в синтаксические детали реализации каждого принципа, важно понять базовую роль каждого из них.

Роль инкапсуляции

Инкапсуляция — это механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных. В объектно-ориентированном языке данные и код могут быть объединены в совершенно автономный черный ящик. Внутри такого ящика находятся все необходимые данные и код. Когда код и данные связываются вместе подобным образом, создается объект. Иными словами, объект — это элемент, поддерживающий инкапсуляцию.

Т.е. инкапсуляция представляет собой способности языка скрывать излишние детали реализации от пользователя объекта. Например, предположим, что используется класс по имени DatabaseReader, который имеет два главных метода: Open() и Close().

Фиктивный класс DatabaseReader инкапсулирует внутренние детали нахождения, загрузки, манипуляций и закрытия файла данных. Программистам нравится инкапсуляция, поскольку этот принцип ООП упрощает кодирование. Нет необходимости беспокоиться о многочисленных строках кода, которые работают "за кулисами", чтобы реализовать функционирование класса DatabaseReader. Все, что потребуется — это создать экземпляр и отправлять ему соответствующие сообщения (например, "открыть файл по имени AutoLot.mdf, расположенный на диске С:").

С идеей инкапсуляции программной логики тесно связана идея защиты данных. В идеале данные состояния объекта должны быть специфицированы с использованием ключевого слова private (или, возможно, protected). Таким образом, внешний мир должен вежливо попросить, если захочет изменить или получить лежащее в основе значение. Это хороший принцип, поскольку общедоступные элементы данных можно легко повредить (даже нечаянно, а не преднамеренно).

Основной единицей инкапсуляции в C# является класс, который определяет форму объекта. Он описывает данные, а также код, который будет ими оперировать. В C# описание класса служит для построения объектов, которые являются экземплярами класса. Следовательно, класс, по существу, представляет собой ряд схематических описаний способа построения объекта.

Код и данные, составляющие вместе класс, называют членами. Данные, определяемые классом, называют полями, или переменными экземпляра. А код, оперирующий данными, содержится в функциях-членах, самым типичным представителем которых является метод. В C# метод служит в качестве аналога подпрограммы. (К числу других функций-членов относятся свойства, события и конструкторы.) Таким образом, методы класса содержат код, воздействующий на поля, определяемые этим классом.

Роль наследования

Следующий принцип ООП — наследование — касается способности языка позволять строить новые определения классов на основе определений существующих классов. По сути, наследование позволяет расширять поведение базового (или родительского) класса, наследуя основную функциональность в производном подклассе (также именуемом дочерним классом):

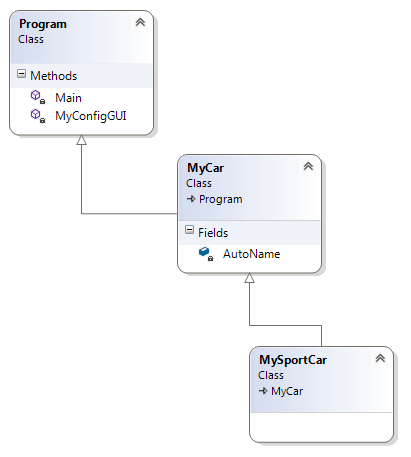


Рис. 6 Принцип наследования.

Принцип наследования

Т.е. наследование представляет собой процесс, в ходе которого один объект приобретает свойства другого объекта. Это очень важный процесс, поскольку он обеспечивает принцип иерархической классификации. Если вдуматься, то большая часть знаний поддается систематизации благодаря иерархической классификации по нисходящей.

Если не пользоваться иерархиями, то для каждого объекта пришлось бы явно определять все его свойства. А если воспользоваться наследованием, то достаточно определить лишь те свойства, которые делают объект особенным в его классе. Он может также наследовать общие свойства своего родителя. Следовательно, благодаря механизму наследования один объект становится отдельным экземпляром более общего класса.

Роль полиморфизма

Последний принцип ООП — полиморфизм. Он обозначает способность языка трактовать связанные объекты в сходной манере. В частности, этот принцип ООП позволяет базовому классу определять набор членов (формально называемый полиморфным интерфейсом), которые доступны всем наследникам. Полиморфный интерфейс класса конструируется с использованием любого количества виртуальных или абстрактных членов.

По сути, виртуальный член — это член базового класса, определяющий реализацию по умолчанию, которая может быть изменена (или, говоря более формально, переопределена) в производном классе. В отличие от него, абстрактный метод — это член базового класса, который не предусматривает реализации по умолчанию, а предлагает только сигнатуру. Когда класс наследуется от базового класса, определяющего абстрактный метод, этот метод обязательно должен быть переопределен в производном классе. В любом случае, когда производные классы переопределяют члены, определенные в базовом классе, они по существу переопределяют свою реакцию на один и тот же запрос.

Рассмотрим для примера стек, т.е. область памяти, функционирующую по принципу "последним пришел — первым обслужен". Допустим, что в программе требуются три разных типа стеков: один — для целых значений, другой — для значений с плавающей точкой, третий — для символьных значений. В данном примере алгоритм, реализующий все эти стеки, остается неизменным, несмотря на то что в них сохраняются разнотипные данные. В языке, не являющемся объектно-ориентированным, для этой цели пришлось бы создать три разных набора стековых подпрограмм с разными именами. Но благодаря полиморфизму для реализации всех трех типов стеков в C# достаточно создать лишь один общий набор подпрограмм. Зная, как пользоваться одним стеком, вы сумеете воспользоваться и остальными.

В более общем смысле понятие полиморфизма нередко выражается следующим образом: "один интерфейс — множество методов". Это означает, что для группы взаимосвязанных действий можно разработать общий интерфейс. Полиморфизм помогает упростить программу, позволяя использовать один и тот же интерфейс для описания общего класса действий. Выбрать конкретное действие (т.е. метод) в каждом отдельном случае — это задача компилятора. Программисту не нужно делать это самому. Ему достаточно запомнить и правильно использовать общий интерфейс.

3. Формализованное описание разработанных классов

Базовым классом в иерархии наследования является класс Visiting. Данный класс содержит в себе информацию о посещения пациентом клиники. Структура данных список в классе Patient будет хранить в себе все экземпляры данного класса.

// Visiting

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Tretie

**{**

class Visiting

**{**

**private** string date**,** service**;**//создание необходим переменных

**private** int cost**;**

**public** Visiting**(**string \_date**,** string \_service**,** int \_cost**)**

**{**

date **=** \_date**;**

service **=** \_service**;**

cost **=** \_cost**;**

**}**

**public** string Date//получение даты

**{**

**get{** **return** date**;}**

**set** **{** date **=** **value;** **}**

**}**

**public** string Service//получение оказанной услуги

**{**

**get** **{** **return** service**;** **}**

**set** **{** service **=** **value;** **}**

**}**

**public** int Cost//получение стоимости услуги

**{**

**get** **{** **return** cost**;** **}**

**set** **{** cost **=** **value;** **}**

**}**

**}**

**}**

Класс Patient содержит в себе всю информацию о пациентах клиники, а именно: фамилию пациента, количество посещений, сумму потраченных денег за всё время посещений.

// Patient

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Tretie

**{**

class Patient

**{**

**private** string name**;**//фамилия пациента

**private** int count**;**//количество посещений

**private** int sum**;**//сумма потраченных денег

**private** List Header**,**Temp**;**//ссылка на первый элемент

**public** Patient**(**string \_name**)**//создание конструктора

**{**

name **=** \_name**;**

Header **=** **new** List**(null);**

Header**.**First **=** **null;**

Header**.**Last **=** **null;**

**}**

**public** string Name **{** **get** **{** **return** name**;** **}** **set** **{** name **=** **value;** **}** **}**

**public** int Count **{** **get** **{** **return** count**;** **}** **}**

**public** List GHeader **{** **get** **{** **return** Header**.**First**;** **}** **}**

**public** void addVisit**(**string \_date**,** string \_service**,** int \_cost**)**//добавление посещения

**{**

Visiting adds **=** **new** Visiting**(**\_date**,** \_service**,** \_cost**);**

List dob **=** **new** List**(**adds**);**

count**++;**

**if(** Header**.**First **==** **null** **)**

**{**

Header**.**First **=** dob**;**

Header**.**Last **=** dob**;**

**}**

**else**

**{**

Temp **=** Header**.**Last**;**

Header**.**Last **=** dob**;**

Temp**.**Next **=** Header**.**Last**;**

Header**.**Last**.**Next **=** Header**.**First**;**

**}**

**}**

**public** bool delVisit**(**string \_date**)** //удаление посещения

**{**

**if** **(**Header **!=** **null)**

**{**

List vspom **=** Header**.**First**;**

Temp **=** Header**.**First**;**

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

**{**

**if** **(**Temp **==** Header**.**First**)**

**{**

**if** **(**Temp**.**data**.**Date**.**Equals**(**\_date**))**

**{**

count**--;**

Header**.**First **=** Header**.**First**.**Next**;**

**return** **true;**

**}**

**else** Temp **=** Temp**.**Next**;**

**}**

**else**

**{**

**if** **(**Temp**.**data**.**Date**.**Equals**(**\_date**))**

**{**

count**--;**

vspom**.**Next **=** Temp**.**Next**;**

**return** **true;**

**}**

**else** **{** Temp **=** Temp**.**Next**;** vspom **=** vspom**.**Next**;** **}**

**}**

**}**

**}**

**return** **false;**

**}**

**public** int Summa**()** //сумма потраченных денег

**{**

sum **=** 0**;**

Temp **=** Header**.**First**;**

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

**{**

sum **+=** Temp**.**data**.**Cost**;**

Temp **=** Temp**.**Next**;**

**}**

**return** sum**;**

**}**

**public** string infoForSave**()**

**{**

string infoSave **=** Convert**.**ToString**(**Count**);**

Temp **=** Header**.**First**;**

**for(**int i **=** 0**;** i **<** Count**;**i**++)**

**{**

infoSave **+=** "\n" **+** Temp**.**data**.**Date**;**

infoSave **+=** "\n" **+** Temp**.**data**.**Service**;**

infoSave **+=** "\n" **+** Temp**.**data**.**Cost**;**

**}**

**return** infoSave**;**

**}**

**public** string Print**()** //вывод информации о пациенте

**{**

string print **=** "\n\nName patient - " **+** name**;**

Temp **=** Header**.**First**;**

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

**{**

print **+=** "\nDate visit: " **+** Temp**.**data**.**Date**;**

print **+=** "\nService: " **+** Temp**.**data**.**Service**;**

print **+=** "\nCost: " **+** Temp**.**data**.**Cost**;**

Temp **=** Temp**.**Next**;**

**}**

print **+=** "\nTotal cost of patient services " **+** name **+** ": " **+** Summa**();**

**return** print**;**

**}**

**public** Visiting search**(**string \_date**,** string \_service**)**

**{**

List vspom **=** Header**.**First**;**

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

**{**

**if(**vspom**.**data**.**Date**.**Equals**(**\_date**)** **&&** vspom**.**data**.**Service**.**Equals**(**\_service**))**

**{**

**return** vspom**.**data**;**

**}**

**}**

**return** **null;**

**}**

**}**

**}**

В процессе работы класса Patient создаётся экземпляр ещё одного созданного класса List. Который реализует адресный замкнутый (кольцевой) однонаправленный список без заголовка.

// List

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Tretie

**{**

class List

**{**

**public** Visiting data**;** //данные посещения

**private** List next**,** first**,** last**;** //вспомогательная переменная-ссылка на потомка

**public** List**(**Visiting \_data**)** //конструктор

**{**

data **=** \_data**;**

next **=** **null;**

first **=** **null;**

last **=** **null;**

**}**

**public** List Next **{** **get** **=>** next**;** **set** **=>** next **=** **value;** **}**

**public** List First **{** **get** **=>** first**;** **set** **=>** first **=** **value;** **}**

**public** List Last **{** **get** **=>** last**;** **set** **=>** last **=** **value;** **}**

**}**

**}**

В конце иерархии наследования находиться класс Clinic в котором реализуется работа с классом Patient реализуя очередь на основе массива. В классе клиника реализуется: добавление новых пациентов, изменение данных пациента, добавление и изменение посещений.

// Clinic

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Tretie

**{**

class Clinic

**{**

**private** string nameComp**;**//название клиники

**private** int size**;**//длина массива

**private** Patient**[]** pati**;**//массив типа Patient

**private** int first**,** count**;**//первый элемент в очереди

**private** int last**,** summa**;**//последний элемент в очереди

**public** Clinic**(**string \_nameComp**,** int \_size**)**

**{**

nameComp **=** \_nameComp**;**

size **=** \_size**;**

pati **=** **new** Patient**[**size**];**

first **=** 0**;** last **=** 0**;** count **=** 0**;**

**}**

**public** int Count **{** **get** **{** **return** count**;** **}** **}**

**public** void addPatient**(**string \_name**)** //добавление пациента

**{**

Patient help **=** **new** Patient**(**\_name**);**

pati**[**last**]** **=** help**;**

last**++;** count**++;**

**if** **(**count **>** **(**0.8 **\*** size**))** **{** IzmArr**();** **}**

**}**

**public** Patient search**(**string \_name**)** //поиск пациента

**{**

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

**{**

**if** **(**pati**[**i**]** **!=** **null)**

**{**

**if** **(**pati**[**i**].**Name**.**Equals**(**\_name**))**

**{**

**return** pati**[**i**];**

**}**

**}**

**}**

**return** **null;**

**}**

**public** void delPatient**()** //удаление пациента

**{**

**if** **(**count **>** 0**)**

**{**

pati**[**first**]** **=** **null;**

count**--;**

**for** **(**int i **=** **(**first **+** 1**);** i **<** last**;** i**++)**

**{**

pati**[**i **-** 1**]** **=** pati**[**i**];**

**}**

pati**[**last**]** **=** **null;**

last**--;**

**}**

**if** **(**count **<** **(**0.3 **\*** size**))** **{** IzmArr**();** **}**

**}**

**public** int summPatient**()** **{** **return** count**;** **}**//количество пациентов

**public** int sumCost**()** //сумма потраченных денег на услуги

**{**

summa **=** 0**;**

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

**{**

summa **+=** pati**[**i**].**Summa**();**

**}**

**return** summa**;**

**}**

**public** string InfoSave**()**

**{**

string infoSave **=** Convert**.**ToString**(**count**);**

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

**{**

infoSave **+=** "\n" **+** pati**[**i**].**Name**;**

**}**

infoSave **+=** "\n"**;**

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

**{**

infoSave **+=**"\n" **+** pati**[**i**].**infoForSave**();**

**}**

**return** infoSave**;**

**}**

**public** string Info**()** //получение информации о всех клиентах

**{**

string info **=** "Name company - " **+** nameComp**;**

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

**{**

info **+=** pati**[**i**].**Print**();**

**}**

info **+=** "\nSumm all patients: " **+** sumCost**();**

info **+=** "\nCount client: " **+** summPatient**();**

**return** info**;**

**}**

**private** void IzmArr**()** //увеличить или уменьшить размер массива

**{**

**if** **(**count **>** 0.8 **\*** size**)**

**{**

Patient**[]** vspom **=** **new** Patient**[**size **\*** 2**];**

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

**{**

vspom**[**i**]** **=** pati**[**i**];**

**}**

size **\*=** 2**;**

pati **=** **new** Patient**[**size**];**

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

**{**

pati**[**i**]** **=** vspom**[**i**];**

**}**

**}**

**else**

**{**

Patient**[]** vspom **=** **new** Patient**[**size **/** 2**];**

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

**{**

vspom**[**i**]** **=** pati**[**i**];**

**}**

size **/=** 2**;**

pati **=** **new** Patient**[**size**];**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

**{**

pati**[**i**]** **=** vspom**[**i**];**

**}**

**}**

**}**

**}**

**}**

4. Описание демонстрационного модуля с характеристикой использованных стандартных компонентов и списком реализованных обработчиков

При первом запуске программы создаётся экземпляр класса Clinic c названием "Клиника Доктора Бубновского" и задаётся количество возможных пациентов – 5. Далее идёт создание окна приложения.

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**ComponentModel**;**

**using** System**.**Data**;**

**using** System**.**Drawing**;**

**using** System**.**IO**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**Windows**.**Forms**;**

**namespace** Tretie

**{**

**public** **partial** class Form1 **:** Form

**{**

Clinic doctor **=** **new** Clinic**(**"Клиника Доктора Бубновского"**,** 5**);**

**public** Form1**()**

**{**

InitializeComponent**();**

**}**

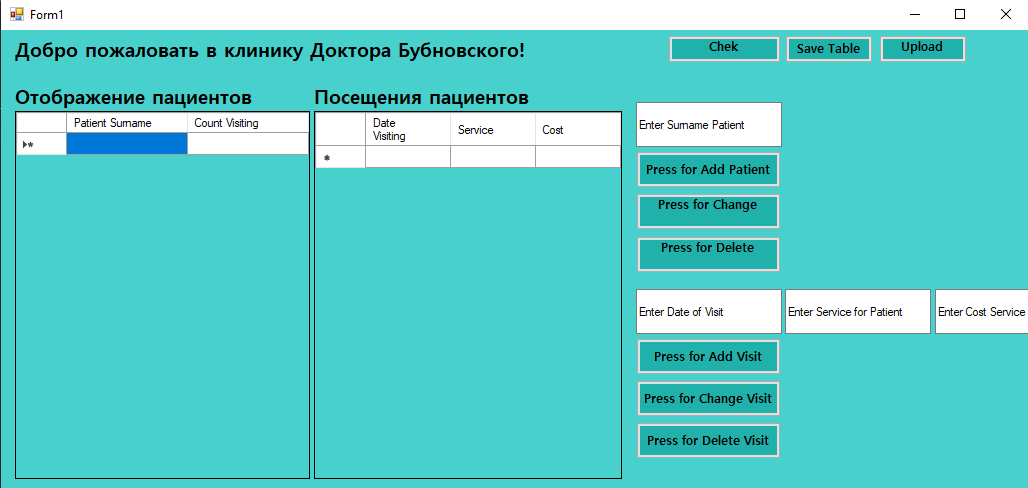


Рис. 7 Старт программы.

Прописываем события для существующих TextBox, что при начале ввода, старая информация стирается.

**private** void textCost\_Click**(object** sender**,** EventArgs e**)**

**{**

textCost**.**Clear**();**

**}**

**private** void textDate\_Click**(object** sender**,** EventArgs e**)**

**{**

textDate**.**Clear**();**

**}**

**private** void textPat\_Click**(object** sender**,** EventArgs e**)**

**{**

textPat**.**Clear**();**

**}**

**private** void textServ\_Click**(object** sender**,** EventArgs e**)**

**{**

textServ**.**Clear**();**

**}**

**private** void textCost\_TextChanged**(object** sender**,** EventArgs e**)**

**{**

textCost**.**MaxLength **=** 7**;**

**}**

Прописывается валидатор на textbox стоимости посещения, для ввода только цифр.

**private** void textCost\_KeyPress**(object** sender**,** KeyPressEventArgs e**)**

**{**

// ввод в texBox только цифр и кнопки Backspace

char ch **=** e**.**KeyChar**;**

**if** **(!**Char**.**IsDigit**(**ch**)** **&&** ch **!=** 8**)**

**{**

e**.**Handled **=** **true;**

**}**

**}**

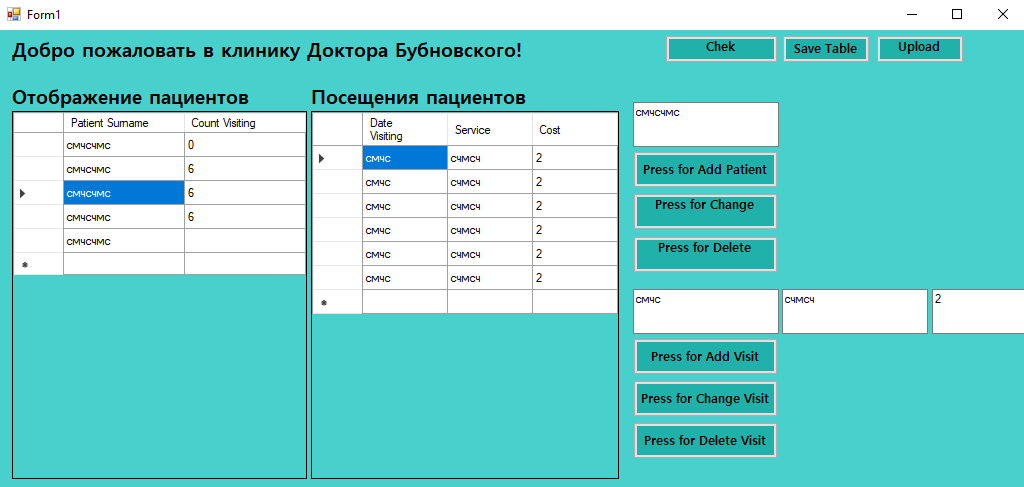
****

Рис. Процесс работы программы.

Данный метод отрабатывает событие, когда пользователь выбирает клиента в первой таблице клиентов. Данный метод необходим для дальнейшего добавления посещений конкретному пациенту.

**private** void dataGridView1\_CellContentClick**(object** sender**,** DataGridViewCellEventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null)**

**{**

dataGridView2**.**Rows**.**Clear**();**

Patient vspom **=** doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**());**

**if** **(**vspom **!=** **null)**

**{**

List pTemp **=** vspom**.**GHeader**;**

int chet **=** doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**Count**;**

**for** **(**int i **=** 0**;** i **<** chet**;** i**++)**

**{**

dataGridView2**.**Rows**.**Add**(**pTemp**.**data**.**Date**,** pTemp**.**data**.**Service**,** pTemp**.**data**.**Cost**);**

pTemp **=** pTemp**.**Next**;**

**}**

dataGridView1**.**Rows**[**dataGridView1**.**CurrentRow**.**Index**].**Cells**[**1**].**Value **=** chet**;**

**}**

**}**

**else** MessageBox**.**Show**(**"Выбрана пустая строчка"**);**

**}**

Метод кнопки добавления клиента. Читаются данные с TextBox и передаются в качестве параметра в таблицу и в экземпляр класса Clinic.

**private** void butAddPat\_Click**(object** sender**,** EventArgs e**)**

**{**

doctor**.**addPatient**(**textPat**.**Text**);**

dataGridView1**.**Rows**.**Add**(**textPat**.**Text**);**

**}**

Метод кнопки изменения данных о клиенте. Идёт чтение данных из TextBox и запись в таблицу и вызова метода изменения данных в экземпляре класса Clinic.

**private** void butChaPat\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null** **&&** textPat**.**Text **!=** **null)**

**{**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**Name **=** textPat**.**Text**;**

dataGridView1**.**Rows**[**dataGridView1**.**CurrentRow**.**Index**].**Cells**[**0**].**Value **=** textPat**.**Text**;**

dataGridView1**.**Rows**[**dataGridView1**.**CurrentRow**.**Index**].**Cells**[**1**].**Value **=** doctor**.**search**(**textPat**.**Text**).**Count**;**

**}**

**else** MessageBox**.**Show**(**"Выберите пациента, которого хотите изменить, или напишите новое имя для пациента."**);**

**}**

Кнопка удаления клиента. Удаляется клиент из массива и очищается из таблицы.

**private** void butDelPat\_Click**(object** sender**,** EventArgs e**)**

**{**

doctor**.**delPatient**();**

dataGridView1**.**Rows**.**RemoveAt**(**0**);**

dataGridView2**.**Rows**.**Clear**();**

**}**

**private** void butAddVis\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null** **&&** textCost**.**Text **!=** **null** **&&** textDate**.**Text **!=** **null** **&&** textServ**.**Text **!=** **null)**

**{**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**addVisit**(**textDate**.**Text**,** textServ**.**Text**,** int**.**Parse**(**textCost**.**Text**));**

dataGridView2**.**Rows**.**Add**(**textDate**.**Text**,** textServ**.**Text**,** textCost**.**Text**);**

**}**

**else** MessageBox**.**Show**(**"Выберите пациента, или впишите недостающие параметры"**);**

**}**

Метод для вывода информации о всей клинике в новом всплывающем окне.

**private** void butChek\_Click**(object** sender**,** EventArgs e**)**

**{**

MessageBox**.**Show**(**doctor**.**Info**());**

**}**

Метод для изменения данных о пациенте, проверяется, что TextBox не пустой, в таблице выбрано посещение, которое нужно изменить. После этого вызывается метод у экземпляра Clinic, для изменения данных и обновляются данные в таблице.

**private** void butChVis\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null** **&&** textCost**.**Text **!=** **null** **&&** textDate**.**Text **!=** **null** **&&** textServ**.**Text **!=** **null)**

**{**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**search**(**textDate**.**Text**,** textServ**.**Text**).**Date **=** textDate**.**Text**;**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**search**(**textDate**.**Text**,** textServ**.**Text**).**Service **=** textServ**.**Text**;**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**search**(**textDate**.**Text**,** textServ**.**Text**).**Cost **=** int**.**Parse**(**textCost**.**Text**);**

dataGridView2**.**Rows**[**dataGridView2**.**CurrentRow**.**Index**].**Cells**[**0**].**Value **=** textDate**.**Text**;**

dataGridView2**.**Rows**[**dataGridView2**.**CurrentRow**.**Index**].**Cells**[**1**].**Value **=** textServ**.**Text**;**

dataGridView2**.**Rows**[**dataGridView2**.**CurrentRow**.**Index**].**Cells**[**2**].**Value **=** textCost**.**Text**;**

**}**

**else** MessageBox**.**Show**(**"Выберите пациента, или впишите недостающие параметры"**);**

**}**

Метод удаления посещения клиентом поликлиники. Удаление происходит из таблицы и из экземпляра Clinic.

**private** void butDelVis\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null** **&&** dataGridView2**.**CurrentCell**.**Value **!=** **null)**

**{**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**delVisit**(**dataGridView2**.**CurrentCell**.**Value**.**ToString**());**

dataGridView2**.**Rows**.**RemoveAt**(**dataGridView2**.**CurrentRow**.**Index**);**

**}**

**else** MessageBox**.**Show**(**"Выберите пациента, или его посещения в клинику"**);**

**}**

Метод выгрузки данных в файл формата txt. Для дальнейшего чтения в начале файла пишем "Клиника Доктора Бубновского\n"**, чтобы** можно было идентифицировать файл программы от других txt.

**private** void butSave\_Click**(object** sender**,** EventArgs e**)**

**{**

string path**;**

string textInFile **=** "Клиника Доктора Бубновского\n"**;**

**using** **(**SaveFileDialog saveFile **=** **new** SaveFileDialog**()** **{** Filter **=** "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" **})**

**{**

**if** **(**saveFile**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

path **=** saveFile**.**FileName**;**

textInFile **+=** doctor**.**InfoSave**();**

**using** **(**FileStream file **=** **new** FileStream**(**path**,** FileMode**.**OpenOrCreate**))**

**using** **(**StreamWriter stream **=** **new** StreamWriter**(**file**))**

**{**

stream**.**Write**(**textInFile**);**

stream**.**Close**();**

file**.**Close**();**

**}**

**}**

**}**

**}**

Метод загрузки из файла. Открывает диалоговое окно для отображения txt файлов. Если файл удалось открыть происходит проверка на корректные данные, т.е. первая сточка в файле должна быть "Клиника Доктора Бубновского" после этого происходит дальнейшее чтение файла, заполнение структуры и таблицы данными из файла.

**private** void butUpload\_Click**(object** sender**,** EventArgs e**)**

**{**

string path **=** ""**,** doIt **=** ""**;**

int Count**,** CusCount**,** \_cost**;**

**using** **(**OpenFileDialog openFile **=** **new** OpenFileDialog**()** **{** Filter **=** "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" **})**

**{**

**if** **(**openFile**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

path **=** openFile**.**FileName**;**

**}**

**}**

**using** **(**StreamReader stream3 **=** **new** StreamReader**(**path**))**

**{**

doIt **=** stream3**.**ReadLine**();**

**if** **(**doIt **==** "Клиника Доктора Бубновского"**)**

**{**

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

**{**

doctor**.**delPatient**();**

dataGridView1**.**Rows**.**Clear**();**

dataGridView2**.**Rows**.**Clear**();**

**}**

Count **=** int**.**Parse**(**stream3**.**ReadLine**());**

**for** **(**int i **=** 1**;** i **<=** Count**;)**

**{**

string SurnamePat **=** stream3**.**ReadLine**();**

doctor**.**addPatient**(**SurnamePat**);**

dataGridView1**.**Rows**.**Add**();**

dataGridView1**.**Rows**[**i **-** 1**].**Cells**[**0**].**Value **=** SurnamePat**;**

dataGridView1**.**Rows**[**i **-** 1**].**Cells**[**1**].**Value **=** doctor**.**search**(**SurnamePat**).**Count**;**

**if** **(**i **!=** Count**)**

i**++;**

**else**

**break;**

**}**

stream3**.**ReadLine**();** int hall **=** 0**;**

**while** **(**stream3**.**Peek**()** **!=** **-**1**)**

**{**

CusCount **=** int**.**Parse**(**stream3**.**ReadLine**());**

**if** **(**CusCount **!=** 0**)**

**for** **(**int i **=** 1**;** i **<=** CusCount**;)**

**{**

string \_date **=** stream3**.**ReadLine**();**

string \_service **=** stream3**.**ReadLine**();**

\_cost **=** int**.**Parse**(**stream3**.**ReadLine**());**

string name **=** Convert**.**ToString**(**dataGridView1**.**Rows**[**hall**].**Cells**[**0**].**Value**);**

doctor**.**search**(**name**).**addVisit**(**\_date**,**\_service**,**\_cost**);**

**if** **(**i **!=** CusCount**)**

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

**else** **{** MessageBox**.**Show**(**"Данный файл не подходит для работы с программой"**);** **}**

stream3**.**Close**();**

**}**

**}**

**}**

**}**

**5. Описание структуры проекта в соответствии с использованным инструментом разработки**

Программа была разработана в среде разработки Visual Studio 2022. Это универсальная среда разработке позволяющая разрабатывать программы на языке программирования C# , а так же С++ и Python. IDE позволяет созывать комплексные программы, а интуитивно понятный интерфейс позволяет легко разобраться в программе и концентрироваться только на разработке.

Помимо удобного интерфейса программа позволяет тестировать программу на ресурсоёмкость, показывая программисту потребление программой ресурсов оперативной памяти и процессора.



Рис. 8 Заставка Visual Studio 2022

В главной папке проекта находятся разработанные классы \*.cs , так же в App.config находятся настройки для среды разработки Visual Studio находятся настройки конфигурации для запуска приложения. Файл \*.resx содержит в себе дизайн оконного приложения, составленный на языке разметки XML.

В папке Propertis находятся все подключенные в разработанных классах стандартные библиотеки для установления зависимостей в проекте.

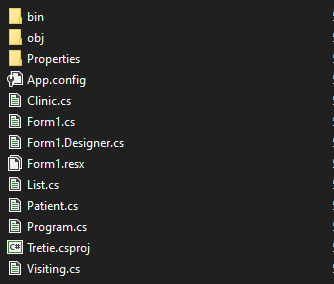


Рис. 9 Содержимое папки проекта.

Папка obj содержит в себе данные последнего Debug запуска проекта с соответствующими файлами конфигурации и логами.

В парке bin находиться главный исполняемый файл программы с последней сборки проекта, а также папка Relies с исполняемым файлом готовым к установке программы со всеми зависимостями на любой Windows операционной системе.

**Список литературы**

1. Алгоритмы и структуры данных: учебное пособие / О.Б. Фофанов; Томский политехнический университет. – Томск: Изд-во Томского политех нического университета, 2019.
2. Васильев А. Н. Java. Объектно-ориентированное программирование. Учебное пособие. Стандарт третьего поколения. – "Издательский дом Питер 2021.
3. Кувшинов Д. Р., Осипов С. И. Основы программирования: язык C++: учебное пособие. – 2021.
4. Ляпин А. А., Быкова Ю. А. РАЗРАБОТКА БАЛЛИСТИЧЕСКОЙ ФУНКЦИИ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ C/C++ //ПОЛИ- ХОТОМИЧЕСКИЕ КЛАССИФИКАЦИИ СПЕЦИАЛЬНЫХ ТЕРМИ- НОВ В НАУЧНОЙ И УЧЕБНОЙ ЛИТЕРАТУРЕ. – 2021. – С. 308-311.
5. Объектно-ориентированное программирование. Учебно-методическое пособие/ Козин.А.Н., Якунина Е.А. – Казань: УВО «Университет управления «ТИСБИ», 2020.
6. Структуры и алгоритмы обработки данных. Часть 2. Усложнённые структуры данных. Теория и методика обучения. Учебно-методическое пособие / А.Н. Козин, Л.Б. Таренко. - Казань: УВО "Университет управления "ТИСБИ 2018
7. Павловская Т. А., Щупак Ю. А. C/C++. Структурное и объектно- ориентированное программирование. Практикум. – "Издательский дом Питер 2021.
8. Структуры и алгоритмы обработки данных. Часть 3. Поиск и сортировка данных. Теория и методика обучения. Учебно-методическое пособие/ А.Н. Козин, Л.Б. Таренко. - Казань: УВО "Университет управления "ТИСБИ 2020
9. Puuronen M. Implementing Horizontal Layout for the Qt Design Studio’s Component Library. – 2021.
10. Sherriff N. Learn Qt 5: Build modern, responsive cross-platform desktop applications with Qt, C++, and QML. – Packt Publishing Ltd, 2018.

Листинг программы

// Visiting

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Tretie

**{**

class Visiting

**{**

**private** string date**,** service**;**//создание необходим переменных

**private** int cost**;**

**public** Visiting**(**string \_date**,** string \_service**,** int \_cost**)**

**{**

date **=** \_date**;**

service **=** \_service**;**

cost **=** \_cost**;**

**}**

**public** string Date//получение даты

**{**

**get{** **return** date**;}**

**set** **{** date **=** **value;** **}**

**}**

**public** string Service//получение оказанной услуги

**{**

**get** **{** **return** service**;** **}**

**set** **{** service **=** **value;** **}**

**}**

**public** int Cost//получение стоимости услуги

**{**

**get** **{** **return** cost**;** **}**

**set** **{** cost **=** **value;** **}**

**}**

**}**

**}**

// Patient

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Tretie

**{**

class Patient

**{**

**private** string name**;**//фамилия пациента

**private** int count**;**//количество посещений

**private** int sum**;**//сумма потраченных денег

**private** List Header**,**Temp**;**//ссылка на первый элемент

**public** Patient**(**string \_name**)**//создание конструктора

**{**

name **=** \_name**;**

Header **=** **new** List**(null);**

Header**.**First **=** **null;**

Header**.**Last **=** **null;**

**}**

**public** string Name **{** **get** **{** **return** name**;** **}** **set** **{** name **=** **value;** **}** **}**

**public** int Count **{** **get** **{** **return** count**;** **}** **}**

**public** List GHeader **{** **get** **{** **return** Header**.**First**;** **}** **}**

**public** void addVisit**(**string \_date**,** string \_service**,** int \_cost**)**//добавление посещения

**{**

Visiting adds **=** **new** Visiting**(**\_date**,** \_service**,** \_cost**);**

List dob **=** **new** List**(**adds**);**

count**++;**

**if(** Header**.**First **==** **null** **)**

**{**

Header**.**First **=** dob**;**

Header**.**Last **=** dob**;**

**}**

**else**

**{**

Temp **=** Header**.**Last**;**

Header**.**Last **=** dob**;**

Temp**.**Next **=** Header**.**Last**;**

Header**.**Last**.**Next **=** Header**.**First**;**

**}**

**}**

**public** bool delVisit**(**string \_date**)** //удаление посещения

**{**

**if** **(**Header **!=** **null)**

**{**

List vspom **=** Header**.**First**;**

Temp **=** Header**.**First**;**

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

**{**

**if** **(**Temp **==** Header**.**First**)**

**{**

**if** **(**Temp**.**data**.**Date**.**Equals**(**\_date**))**

**{**

count**--;**

Header**.**First **=** Header**.**First**.**Next**;**

**return** **true;**

**}**

**else** Temp **=** Temp**.**Next**;**

**}**

**else**

**{**

**if** **(**Temp**.**data**.**Date**.**Equals**(**\_date**))**

**{**

count**--;**

vspom**.**Next **=** Temp**.**Next**;**

**return** **true;**

**}**

**else** **{** Temp **=** Temp**.**Next**;** vspom **=** vspom**.**Next**;** **}**

**}**

**}**

**}**

**return** **false;**

**}**

**public** int Summa**()** //сумма потраченных денег

**{**

sum **=** 0**;**

Temp **=** Header**.**First**;**

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

**{**

sum **+=** Temp**.**data**.**Cost**;**

Temp **=** Temp**.**Next**;**

**}**

**return** sum**;**

**}**

**public** string infoForSave**()**

**{**

string infoSave **=** Convert**.**ToString**(**Count**);**

Temp **=** Header**.**First**;**

**for(**int i **=** 0**;** i **<** Count**;**i**++)**

**{**

infoSave **+=** "\n" **+** Temp**.**data**.**Date**;**

infoSave **+=** "\n" **+** Temp**.**data**.**Service**;**

infoSave **+=** "\n" **+** Temp**.**data**.**Cost**;**

**}**

**return** infoSave**;**

**}**

**public** string Print**()** //вывод информации о пациенте

**{**

string print **=** "\n\nName patient - " **+** name**;**

Temp **=** Header**.**First**;**

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

**{**

print **+=** "\nDate visit: " **+** Temp**.**data**.**Date**;**

print **+=** "\nService: " **+** Temp**.**data**.**Service**;**

print **+=** "\nCost: " **+** Temp**.**data**.**Cost**;**

Temp **=** Temp**.**Next**;**

**}**

print **+=** "\nTotal cost of patient services " **+** name **+** ": " **+** Summa**();**

**return** print**;**

**}**

**public** Visiting search**(**string \_date**,** string \_service**)**

**{**

List vspom **=** Header**.**First**;**

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

**{**

**if(**vspom**.**data**.**Date**.**Equals**(**\_date**)** **&&** vspom**.**data**.**Service**.**Equals**(**\_service**))**

**{**

**return** vspom**.**data**;**

**}**

**}**

**return** **null;**

**}**

**}**

**}**

// List

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Tretie

**{**

class List

**{**

**public** Visiting data**;** //данные посещения

**private** List next**,** first**,** last**;** //вспомогательная переменная-ссылка на потомка

**public** List**(**Visiting \_data**)** //конструктор

**{**

data **=** \_data**;**

next **=** **null;**

first **=** **null;**

last **=** **null;**

**}**

**public** List Next **{** **get** **=>** next**;** **set** **=>** next **=** **value;** **}**

**public** List First **{** **get** **=>** first**;** **set** **=>** first **=** **value;** **}**

**public** List Last **{** **get** **=>** last**;** **set** **=>** last **=** **value;** **}**

**}**

**}**

// Clinic

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**Text**;**

**namespace** Tretie

**{**

class Clinic

**{**

**private** string nameComp**;**//название клиники

**private** int size**;**//длина массива

**private** Patient**[]** pati**;**//массив типа Patient

**private** int first**,** count**;**//первый элемент в очереди

**private** int last**,** summa**;**//последний элемент в очереди

**public** Clinic**(**string \_nameComp**,** int \_size**)**

**{**

nameComp **=** \_nameComp**;**

size **=** \_size**;**

pati **=** **new** Patient**[**size**];**

first **=** 0**;** last **=** 0**;** count **=** 0**;**

**}**

**public** int Count **{** **get** **{** **return** count**;** **}** **}**

**public** void addPatient**(**string \_name**)** //добавление пациента

**{**

Patient help **=** **new** Patient**(**\_name**);**

pati**[**last**]** **=** help**;**

last**++;** count**++;**

**if** **(**count **>** **(**0.8 **\*** size**))** **{** IzmArr**();** **}**

**}**

**public** Patient search**(**string \_name**)** //поиск пациента

**{**

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

**{**

**if** **(**pati**[**i**]** **!=** **null)**

**{**

**if** **(**pati**[**i**].**Name**.**Equals**(**\_name**))**

**{**

**return** pati**[**i**];**

**}**

**}**

**}**

**return** **null;**

**}**

**public** void delPatient**()** //удаление пациента

**{**

**if** **(**count **>** 0**)**

**{**

pati**[**first**]** **=** **null;**

count**--;**

**for** **(**int i **=** **(**first **+** 1**);** i **<** last**;** i**++)**

**{**

pati**[**i **-** 1**]** **=** pati**[**i**];**

**}**

pati**[**last**]** **=** **null;**

last**--;**

**}**

**if** **(**count **<** **(**0.3 **\*** size**))** **{** IzmArr**();** **}**

**}**

**public** int summPatient**()** **{** **return** count**;** **}**//количество пациентов

**public** int sumCost**()** //сумма потраченных денег на услуги

**{**

summa **=** 0**;**

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

**{**

summa **+=** pati**[**i**].**Summa**();**

**}**

**return** summa**;**

**}**

**public** string InfoSave**()**

**{**

string infoSave **=** Convert**.**ToString**(**count**);**

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

**{**

infoSave **+=** "\n" **+** pati**[**i**].**Name**;**

**}**

infoSave **+=** "\n"**;**

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

**{**

infoSave **+=**"\n" **+** pati**[**i**].**infoForSave**();**

**}**

**return** infoSave**;**

**}**

**public** string Info**()** //получение информации о всех клиентах

**{**

string info **=** "Name company - " **+** nameComp**;**

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

**{**

info **+=** pati**[**i**].**Print**();**

**}**

info **+=** "\nSumm all patients: " **+** sumCost**();**

info **+=** "\nCount client: " **+** summPatient**();**

**return** info**;**

**}**

**private** void IzmArr**()** //увеличить или уменьшить размер массива

**{**

**if** **(**count **>** 0.8 **\*** size**)**

**{**

Patient**[]** vspom **=** **new** Patient**[**size **\*** 2**];**

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

**{**

vspom**[**i**]** **=** pati**[**i**];**

**}**

size **\*=** 2**;**

pati **=** **new** Patient**[**size**];**

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

**{**

pati**[**i**]** **=** vspom**[**i**];**

**}**

**}**

**else**

**{**

Patient**[]** vspom **=** **new** Patient**[**size **/** 2**];**

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

**{**

vspom**[**i**]** **=** pati**[**i**];**

**}**

size **/=** 2**;**

pati **=** **new** Patient**[**size**];**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

**{**

pati**[**i**]** **=** vspom**[**i**];**

**}**

**}**

**}**

**}**

**}**

**using** System**;**

**using** System**.**Collections**.**Generic**;**

**using** System**.**ComponentModel**;**

**using** System**.**Data**;**

**using** System**.**Drawing**;**

**using** System**.**IO**;**

**using** System**.**Linq**;**

**using** System**.**Text**;**

**using** System**.**Threading**.**Tasks**;**

**using** System**.**Windows**.**Forms**;**

**namespace** Tretie

**{**

**public** **partial** class Form1 **:** Form

**{**

Clinic doctor **=** **new** Clinic**(**"Клиника Доктора Бубновского"**,** 5**);**

**public** Form1**()**

**{**

InitializeComponent**();**

**}**

**private** void textCost\_Click**(object** sender**,** EventArgs e**)**

**{**

textCost**.**Clear**();**

**}**

**private** void textDate\_Click**(object** sender**,** EventArgs e**)**

**{**

textDate**.**Clear**();**

**}**

**private** void textPat\_Click**(object** sender**,** EventArgs e**)**

**{**

textPat**.**Clear**();**

**}**

**private** void textServ\_Click**(object** sender**,** EventArgs e**)**

**{**

textServ**.**Clear**();**

**}**

**private** void textCost\_TextChanged**(object** sender**,** EventArgs e**)**

**{**

textCost**.**MaxLength **=** 7**;**

**}**

**private** void textCost\_KeyPress**(object** sender**,** KeyPressEventArgs e**)**

**{**

// ввод в texBox только цифр и кнопки Backspace

char ch **=** e**.**KeyChar**;**

**if** **(!**Char**.**IsDigit**(**ch**)** **&&** ch **!=** 8**)**

**{**

e**.**Handled **=** **true;**

**}**

**}**

**private** void dataGridView1\_CellContentClick**(object** sender**,** DataGridViewCellEventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null)**

**{**

dataGridView2**.**Rows**.**Clear**();**

Patient vspom **=** doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**());**

**if** **(**vspom **!=** **null)**

**{**

List pTemp **=** vspom**.**GHeader**;**

int chet **=** doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**Count**;**

**for** **(**int i **=** 0**;** i **<** chet**;** i**++)**

**{**

dataGridView2**.**Rows**.**Add**(**pTemp**.**data**.**Date**,** pTemp**.**data**.**Service**,** pTemp**.**data**.**Cost**);**

pTemp **=** pTemp**.**Next**;**

**}**

dataGridView1**.**Rows**[**dataGridView1**.**CurrentRow**.**Index**].**Cells**[**1**].**Value **=** chet**;**

**}**

**}**

**else** MessageBox**.**Show**(**"Выбрана пустая строчка"**);**

**}**

**private** void butAddPat\_Click**(object** sender**,** EventArgs e**)**

**{**

doctor**.**addPatient**(**textPat**.**Text**);**

dataGridView1**.**Rows**.**Add**(**textPat**.**Text**);**

**}**

**private** void butChaPat\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null** **&&** textPat**.**Text **!=** **null)**

**{**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**Name **=** textPat**.**Text**;**

dataGridView1**.**Rows**[**dataGridView1**.**CurrentRow**.**Index**].**Cells**[**0**].**Value **=** textPat**.**Text**;**

dataGridView1**.**Rows**[**dataGridView1**.**CurrentRow**.**Index**].**Cells**[**1**].**Value **=** doctor**.**search**(**textPat**.**Text**).**Count**;**

**}**

**else** MessageBox**.**Show**(**"Выберите пациента, которого хотите изменить, или напишите новое имя для пациента."**);**

**}**

**private** void butDelPat\_Click**(object** sender**,** EventArgs e**)**

**{**

doctor**.**delPatient**();**

dataGridView1**.**Rows**.**RemoveAt**(**0**);**

dataGridView2**.**Rows**.**Clear**();**

**}**

**private** void butAddVis\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null** **&&** textCost**.**Text **!=** **null** **&&** textDate**.**Text **!=** **null** **&&** textServ**.**Text **!=** **null)**

**{**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**addVisit**(**textDate**.**Text**,** textServ**.**Text**,** int**.**Parse**(**textCost**.**Text**));**

dataGridView2**.**Rows**.**Add**(**textDate**.**Text**,** textServ**.**Text**,** textCost**.**Text**);**

**}**

**else** MessageBox**.**Show**(**"Выберите пациента, или впишите недостающие параметры"**);**

**}**

**private** void butChek\_Click**(object** sender**,** EventArgs e**)**

**{**

MessageBox**.**Show**(**doctor**.**Info**());**

**}**

**private** void butChVis\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null** **&&** textCost**.**Text **!=** **null** **&&** textDate**.**Text **!=** **null** **&&** textServ**.**Text **!=** **null)**

**{**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**search**(**textDate**.**Text**,** textServ**.**Text**).**Date **=** textDate**.**Text**;**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**search**(**textDate**.**Text**,** textServ**.**Text**).**Service **=** textServ**.**Text**;**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**search**(**textDate**.**Text**,** textServ**.**Text**).**Cost **=** int**.**Parse**(**textCost**.**Text**);**

dataGridView2**.**Rows**[**dataGridView2**.**CurrentRow**.**Index**].**Cells**[**0**].**Value **=** textDate**.**Text**;**

dataGridView2**.**Rows**[**dataGridView2**.**CurrentRow**.**Index**].**Cells**[**1**].**Value **=** textServ**.**Text**;**

dataGridView2**.**Rows**[**dataGridView2**.**CurrentRow**.**Index**].**Cells**[**2**].**Value **=** textCost**.**Text**;**

**}**

**else** MessageBox**.**Show**(**"Выберите пациента, или впишите недостающие параметры"**);**

**}**

**private** void butDelVis\_Click**(object** sender**,** EventArgs e**)**

**{**

**if** **(**dataGridView1**.**CurrentCell**.**Value **!=** **null** **&&** dataGridView2**.**CurrentCell**.**Value **!=** **null)**

**{**

doctor**.**search**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**delVisit**(**dataGridView2**.**CurrentCell**.**Value**.**ToString**());**

dataGridView2**.**Rows**.**RemoveAt**(**dataGridView2**.**CurrentRow**.**Index**);**

**}**

**else** MessageBox**.**Show**(**"Выберите пациента, или его посещения в клинику"**);**

**}**

**private** void butSave\_Click**(object** sender**,** EventArgs e**)**

**{**

string path**;**

string textInFile **=** "Клиника Доктора Бубновского\n"**;**

**using** **(**SaveFileDialog saveFile **=** **new** SaveFileDialog**()** **{** Filter **=** "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" **})**

**{**

**if** **(**saveFile**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

path **=** saveFile**.**FileName**;**

textInFile **+=** doctor**.**InfoSave**();**

**using** **(**FileStream file **=** **new** FileStream**(**path**,** FileMode**.**OpenOrCreate**))**

**using** **(**StreamWriter stream **=** **new** StreamWriter**(**file**))**

**{**

stream**.**Write**(**textInFile**);**

stream**.**Close**();**

file**.**Close**();**

**}**

**}**

**}**

**}**

**private** void butUpload\_Click**(object** sender**,** EventArgs e**)**

**{**

string path **=** ""**,** doIt **=** ""**;**

int Count**,** CusCount**,** \_cost**;**

**using** **(**OpenFileDialog openFile **=** **new** OpenFileDialog**()** **{** Filter **=** "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" **})**

**{**

**if** **(**openFile**.**ShowDialog**()** **==** DialogResult**.**OK**)**

**{**

path **=** openFile**.**FileName**;**

**}**

**}**

**using** **(**StreamReader stream3 **=** **new** StreamReader**(**path**))**

**{**

doIt **=** stream3**.**ReadLine**();**

**if** **(**doIt **==** "Клиника Доктора Бубновского"**)**

**{**

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

**{**

doctor**.**delPatient**();**

dataGridView1**.**Rows**.**Clear**();**

dataGridView2**.**Rows**.**Clear**();**

**}**

Count **=** int**.**Parse**(**stream3**.**ReadLine**());**

**for** **(**int i **=** 1**;** i **<=** Count**;)**

**{**

string SurnamePat **=** stream3**.**ReadLine**();**

doctor**.**addPatient**(**SurnamePat**);**

dataGridView1**.**Rows**.**Add**();**

dataGridView1**.**Rows**[**i **-** 1**].**Cells**[**0**].**Value **=** SurnamePat**;**

dataGridView1**.**Rows**[**i **-** 1**].**Cells**[**1**].**Value **=** doctor**.**search**(**SurnamePat**).**Count**;**

**if** **(**i **!=** Count**)**

i**++;**

**else**

**break;**

**}**

stream3**.**ReadLine**();** int hall **=** 0**;**

**while** **(**stream3**.**Peek**()** **!=** **-**1**)**

**{**

CusCount **=** int**.**Parse**(**stream3**.**ReadLine**());**

**if** **(**CusCount **!=** 0**)**

**for** **(**int i **=** 1**;** i **<=** CusCount**;)**

**{**

string \_date **=** stream3**.**ReadLine**();**

string \_service **=** stream3**.**ReadLine**();**

\_cost **=** int**.**Parse**(**stream3**.**ReadLine**());**

string name **=** Convert**.**ToString**(**dataGridView1**.**Rows**[**hall**].**Cells**[**0**].**Value**);**

doctor**.**search**(**name**).**addVisit**(**\_date**,**\_service**,**\_cost**);**

**if** **(**i **!=** CusCount**)**

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

**else** **{** MessageBox**.**Show**(**"Данный файл не подходит для работы с программой"**);** **}**

stream3**.**Close**();**

**}**

**}**

**}**

**}**