УВО «Университет Управления «ТИСБИ»

Факультет информационных технологий

Кафедра информационных технологий

**Курсовая работа**

по дисциплине «Объектно-ориентированное программирование»

на тему: «Разработка объектной программы для задачи «Служба трудоустройства»

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

Проверил: ст. преп. Якунина Е.А.

Казань 2022

Оглавление

[Постановка задачи 3](#_Toc104845698)

[1. Описание используемых структур данных с алгоритмами выполнения основных операций 5](#_Toc104845699)

[1.1 Адресный замкнутый (кольцевой) упорядоченный однонаправленный список без заголовка. 5](#_Toc104845700)

[1.2 Кольцевая очередь на основе обычного массива 8](#_Toc104845701)

[2. Краткие сведения об объектном подходе 10](#_Toc104845702)

[3. Формализованное описание разработанных классов 16](#_Toc104845703)

[1. Описание демонстрационного модуля с характеристикой использованных стандартных компонентов и списком реализованных обработчиков 21](#_Toc104845704)

[2. Описание структуры проекта в соответствии с использованным инструментом разработки 36](#_Toc104845705)

[Листинг программы 40](#_Toc104845706)

Постановка задачи

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

**Постановка задачи:** Разработать объектную программу для хранения и обработки данных о вакансиях в службе трудоустройства. Программа должна поддерживать список организаций-работодателей с указанием уникального наименования организации. Для каждой организации создается свой список вакансий с указанием должности и оклада.

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 организаций используется структура данных в виде стека на основе обычного массива. Для объединения вакансий для каждой организации используется структура данных в виде адресного разомкнутого 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 Стек на массиве.

Стек представляет собой структуру данных, которая работает по принципу LIFO (Last In First Out - "последний пришел - первый вышел"). Графически стек можно представить в виде столбика или стопки объектов:

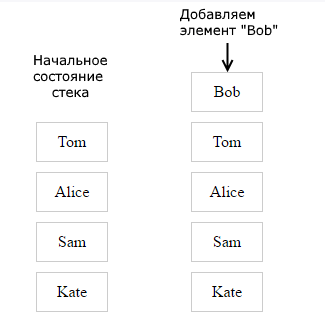


Рис. 1 Стек

Стек имеет вершину, который образует последний добавленный элемент. При добавлении новый элемент помещается поверх вершины стека и образует новую вершину. При удалении удаляется элемент из вершины стека, а предыдущий элемент образует новую вершину.

Так, на приведенном рисунке вначале вершиной стека является "Tom". После добавления нового элемента "Bob" этот элемент располагается поверх элемента "Tom" и становится новой вершиной.

В библиотеке классов .NET в принципе уже есть свой класс, который выполняет роль стека. Это класс - System.Collections.Generic.Stack. Но рассмотрим, как мы сами можем реализовать структуру в виде стека.

Структура стек вне зависимости от языка программирования обладает неким общим функционалом, который составляют метод добавления элемента (как правило, называется push()) и метод извлечения элемента из вершины стека (обычно называется pop()). Кроме того, нередко реализации стеков содержат метод получения элемента из вершины без его извлечения, метод определения размера стека и ряд других.

1.2 Адресный однонаправленный разомкнутый список.

Реальное многообразие структур данных базируется всего на двух основных способах получения адреса хранимого элемента: вычисление (массив) и хранение (указатель). До сих пор основной компонентой структуры данных являлся массив (обычный массив, массив указателей). Если же попытаться построить структуру данных, исходя только из указателей, то получается цепочка (последовательность) элементов, содержащих указатели друг на друга. В простейшем случае она может быть линейной (список), в более сложных случаях – ветвящейся (деревья, графы). Итак, список – линейная последовательность элементов, каждый из которых содержит указатели (ссылается) на своих соседей.

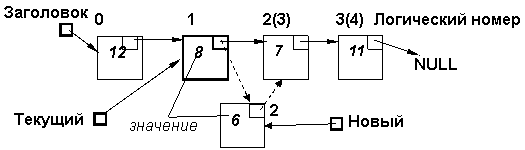


рис. 63-1. Списковая структура

Сразу же отметим основную особенность: физическое размещение в памяти элементов списка не имеет никакого значения, все определяется наличием ссылок на него в других элементах и извне. У массива всегда есть «начало». У списка по определению отсутствует фиксированная привязка к памяти. Перечислим основные релятивистские свойства списка:

элемент списка доступен в программе через указатель. «Смысл» этого указателя отражает функциональное назначение элемента списка в программе: первый, последний, текущий, предыдущий, новый и т.п.. Между указателем и элементом списка имеется такая же взаимосвязь, как между индексом в массиве и элементом массива;

в программе список задается посредством заголовка – указателя на первый элемент списка;

порядок следования элементов определяется последовательностью связей между элементами. Изменение порядка следования элементов (вставка, удаление) осуществляются изменением переустановкой указателей на соседние элементы.

логический (порядковый) номер элемента списка также задается его естественной нумерацией в цепочке элементов;

список является структурой данных с последовательным доступом. Для получения n-го по счету элемента необходимо последовательно пройти по цепочке от элемента, на который имеется указатель (например, от заголовка);

список удобен для использования именно как динамическая структура данных: элементы списка обычно создаются как динамические переменные, а связи между ними устанавливаются программно (динамически);

список обладает свойством локальности изменений: при вставке/удалении элемента изменения касаются только текущего и его соседей. Вспомним массив: при вставке/удалении его элементов происходит физическое перемещение (сдвиг) всех элементов от текущего до конца.

Отсюда следует, что преимущества списков проявляются в таких структурах данных, где операции изменения порядка превалируют над операциями доступа и поиска.

1. Краткие сведения об объектном подходе

Объектно-ориентированное программирование (ООП) – это концепция программирования, которая работает на принципах абстракции, инкапсуляции, наследования и полиморфизма. Это позволяет пользователям создавать объекты, которые они хотят, а затем создавать методы для обработки этих объектов. Основная концепция ООП заключается в создании объектов, их повторном использовании в программе и манипулировании этими объектами для получения результатов.

Абстракция

Абстра́кция — в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.

Такой подход является основой объектно-ориентированного программирования. Это позволяет работать с объектами, не вдаваясь в особенности их реализации. В каждом конкретном случае применяется тот или иной подход: инкапсуляция, полиморфизм или наследование. Например, при необходимости обратиться к скрытым данным объекта, следует воспользоваться инкапсуляцией, создав, так называемую, функцию доступа или свойство.

Абстракция данных — популярная и в общем неверно определяемая техника программирования. Фундаментальная идея состоит в разделении несущественных деталей реализации подпрограммы и характеристик существенных для корректного ее использования. Такое разделение может быть выражено через специальный «интерфейс», сосредотачивающий описание всех возможных применений программы [1].

С точки зрения теории множеств, процесс представляет собой организацию для группы подмножеств своего множества. См. также Закон обратного отношения между содержанием и объемом понятия.

Инкапсуляция

Инкапсуляция — свойство программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента (что у него внутри?), а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные. При этом пользователю предоставляется только спецификация (интерфейс) объекта.

Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: public.

Пользователь не может использовать закрытые данные и методы. Реализуется с помощью ключевых слов: private, protected, internal.))

Инкапсуляция — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, полиморфизмом и наследованием).

Сокрытие реализации целесообразно применять в следующих случаях:

предельная локализация изменений при необходимости таких изменений,

прогнозируемость изменений (какие изменения в коде надо сделать для заданного изменения функциональности) и прогнозируемость последствий изменений.

Наследование

Наследование — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Простое наследование:

Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class).

В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Множественное наследование

При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++. Из других языков, предоставляющих эту возможность, можно отметить Python и Эйфель. Множественное наследование поддерживается в языке UML.

Множественное наследование — потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имен методов в предках. В языках, которые позиционируются как наследники C++ (Java, C# и др.), от множественного наследования было решено отказаться в пользу интерфейсов. Практически всегда можно обойтись без использования данного механизма. Однако, если такая необходимость все-таки возникла, то, для разрешения конфликтов использования наследованных методов с одинаковыми именами, возможно, например, применить операцию расширения видимости — «::» — для вызова конкретного метода конкретного родителя.

Попытка решения проблемы наличия одинаковых имен методов в предках была предпринята в языке Эйфель, в котором при описании нового класса необходимо явно указывать импортируемые члены каждого из наследуемых классов и их именование в дочернем классе.

Большинство современных объектно-ориентированных языков программирования (C#, Java, Delphi и др.) поддерживают возможность одновременно наследоваться от класса-предка и реализовать методы нескольких интерфейсов одним и тем же классом. Этот механизм позволяет во многом заменить множественное наследование — методы интерфейсов необходимо переопределять явно, что исключает ошибки при наследовании функциональности одинаковых методов различных классов-предков.

Полиморфизм

Полиморфи́зм — возможность объектов с одинаковой спецификацией иметь различную реализацию.

Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования[1].

Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Полиморфизм — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, инкапсуляцией и наследованием).

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс. Общность имеет внешнее и внутреннее выражение:

внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов и их количеством);

внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (перегрузкой функций, перегрузкой операций).

Формы полиморфизма

Используя Параметрический полиморфизм можно создавать универсальные базовые типы. В случае параметрического полиморфизма, функция реализуется для всех типов одинаково и таким образом функция реализована для произвольного типа. В параметрическом полиморфизме рассматриваются параметрические методы и типы.

Параметрические метод

Если полиморфизм включения влияет на наше восприятие объекта, то параметрический полиморфизм влияет на используемые методы, так как можно создавать методы родственных классов, откладывая объявление типов до времени выполнения. Для во избежание написания отдельного метода каждого типа применяется параметрический полиморфизм, при этом тип параметров будет являться таким же параметром, как и операнды...

Параметрические типы.

Вместо того, чтобы писать класс для каждого конкретного типа следует создать типы, которые будут реализованы во время выполнения программы то есть мы создаем параметрический тип.

3. Формализованное описание разработанных классов

Составляющим классом всех отальных реализованных классов явлется Vacantion который будут использовать остальные. Он содержит в себе информацию о небоходимом стаже работы, заплате и названии работы.

**using** System**;**

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

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

**namespace** Third

**{**

class Vacantion

**{**

**private** string work**;**//стаж работы

**private** int salary**;**//зарплата

**private** string name**;**//название вакансии

**public** Vacantion**(**string \_work**,** int \_salary**,**string \_name**)**//конструктор

**{**

**this.**work **=** \_work**;**

**this.**salary **=** \_salary**;**

**this.**name **=** \_name**;**

**}**

**public** string GetWork**()**//методы доступа для переменной work

**{**

**return** work**;**

**}**

**public** void SetWork**(**string newWork**)**

**{**

**this.**work **=** newWork**;**

**}**

**public** int GetSalary**()**////методы доступа для переменной salary

**{**

**return** salary**;**

**}**

**public** void SetSalary**(** int newSalary**)**

**{**

**this.**salary **=** newSalary**;**

**}**

**public** void SetName**(**string \_name**)**////методы доступа для переменной name

**{**

**this.**name **=** \_name**;**

**}**

**public** string GetName**()**

**{**

**return** name**;**

}

}

}

Класс огранизации позоляет содавать организацию с вакансиями содержащими ся в адресном списке. Храним помимо списка ещё название организации.

**using** System**;**

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

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

**namespace** Third

**{**

class Organizations

**{**

**private** string nameOrganization**;**//название организации

**private** int count**;**//количество вакансий

**private** SList Header**,**pTemp**;**//первый элемент

**public** Organizations**(**string \_nameOrganization**)**//конструктор

**{**

nameOrganization **=** \_nameOrganization**;**

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

Header**.**FIRST **=** **null;**

Header**.**LAST **=** **null;**

pTemp **=** **null;**

**}**

**public** string NameOrganization

**{**

**get** **{** **return** nameOrganization**;** **}**

**set** **{** **this.**nameOrganization **=** **value;** **}**

**}**

**public** SList SHeader **{** **get** **{** **return** Header**.**FIRST**;** **}** **}**

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

**public** Vacantion search**(**string \_name**)**

**{**

SList vspom **=** Header**.**FIRST**;**

**while(**vspom**!=** **null)**

**{**

**if(**vspom**.**data**.**GetName**().**Equals**(**\_name**))**

**{**

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

**}**

**else** **{** vspom **=** vspom**.**NEXT**;** **}**

**}**

**return** **null;**

**}**

**public** string getList**()**

**{**

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

SList pTemp **=** Header**.**FIRST**;**

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

**{**

**if** **(**pTemp **!=** **null)**

**{**

data **+=** "\n" **+** pTemp**.**data**.**GetName**();**

data **+=** "\n" **+** pTemp**.**data**.**GetWork**();**

data **+=** "\n" **+** pTemp**.**data**.**GetSalary**();**

pTemp **=** pTemp**.**NEXT**;**

**}**

**}**

**return** data**;**

**}**

**public** void AddVacantion**(**string \_work**,** int \_salary**,** string \_name**)**//Метод добавдения вакансии

**{**

Vacantion news **=** **new** Vacantion**(**\_work**,** \_salary**,** \_name**);**

SList dob **=** **new** SList**(**news**);**

count**++;**

**if(**Header**.**FIRST **==** **null** **&&** Header**.**LAST **==** **null)**

**{**

Header**.**FIRST **=** dob**;**

Header**.**LAST **=** dob**;**

**}**

**else**

**{**

pTemp **=** Header**.**LAST**;**

Header**.**LAST **=** dob**;**

pTemp**.**NEXT **=** Header**.**LAST**;**

**}**

**}**

**public** bool DeleteVacantion**(**string \_work**)**//метод удаления

**{**

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

**{**

pTemp **=** Header**.**FIRST**;**

**while** **(**pTemp **!=** **null)**

**{**

**if** **(**pTemp **==** Header**.**FIRST**)**

**{**

**if** **(**pTemp**.**data**.**GetName**().**Equals**(**\_work**))**

**{**

count**--;**

Header**.**FIRST **=** Header**.**FIRST**.**NEXT**;**

**return** **true;**

**}**

**else** pTemp **=** pTemp**.**NEXT**;**

**}**

**else**

**{**

SList vspom **=** Header**.**FIRST**;**

**if** **(**pTemp**.**data**.**GetName**().**Equals**(**\_work**))**

**{**

count**--;**

vspom**.**NEXT **=** pTemp**.**NEXT**;**

**return** **true;**

**}**

**else** **{** pTemp **=** pTemp**.**NEXT**;** vspom **=** vspom**.**NEXT**;** **}**

**}**

**}**

**}**

**return** **false;**

**}**

**public** int SumVacantion**()**//суммарное кол-во вакансий

**{**

**return** count**;**

**}**

**public** string chek**()** //вывод данных

**{**

string info **=** "\n------" **+** nameOrganization **+** " present"**;**

pTemp **=** Header**.**FIRST**;**

**while** **(**pTemp **!=** **null)**

**{**

info **+=** "\nVacantion - " **+** pTemp**.**data**.**GetName**()** **+** ". Required lenght of service - " **+** pTemp**.**data**.**GetWork**();**

info **+=** "\nSalary - " **+** pTemp**.**data**.**GetSalary**();**

pTemp **=** pTemp**.**NEXT**;**

**}**

info **+=** "\nSumm count vacantion - " **+** SumVacantion**()** **+** "\n\n"**;**

**return** info**;**

**}**

**}**

**}**

Класс SList реализует стуктуру спиок необходимую для хранениявакансий в классе организации.

**using** System**;**

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

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

**namespace** Third

**{**

class SList

**{**

**public** Vacantion data**;** //данные о организации

**private** SList next**;** //ссылка на потомка

**private** SList last**,** first**;**//заголовок списка

**public** SList**(**Vacantion \_data**)** //конструктор

**{**

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

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

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

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

**}**

**public** SList NEXT//методы доступа к переменной next

**{**

**get** **{** **return** next**;** **}**

**set** **{** **this.**next **=** **value;** **}**

**}**

**public** SList FIRST//методы доступа к переменной first

**{**

**get** **{** **return** first**;** **}**

**set** **{** **this.**first **=** **value;** **}**

**}**

**public** SList LAST//методы доступа к переменной last

**{**

**get** **{** **return** last**;** **}**

**set** **{** **this.**last **=** **value;** **}**

**}**

**}**

**}**

Класс реализующий контейнер стек на основе массива. Для хранения и работе с оранизациями и их вакансиями.

**using** System**;**

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

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

**namespace** Third

**{**

class EmploymentService

**{**

**private** Organizations**[]** stack**;**//массив типа Patient

**private** int SP**;**//вспомогательная переменная для добавления/удаления

**private** int startSize**;**//

**public** EmploymentService**(**int \_startSize**)**//конструктор

**{**

SP **=** 0**;**

startSize **=** \_startSize**;**

stack **=** **new** Organizations**[**startSize**];**

**}**

**public** int getSP **{** **get** **{** **return** SP**;** **}** **}**

**public** void addComp**(**string \_nameOrganization**)**//добавление организаций

**{**

**if(**SP **<=** startSize**)**

**{**

Organizations newsdob **=** **new** Organizations**(**\_nameOrganization**);**

stack**[**SP**]** **=** newsdob**;**

SP**++;**

**}**

**}**

**public** Organizations FindComp**(**string \_nameOrganization**)**//поиск организаций

**{**

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

**{**

**if** **(**stack**[**i**].**NameOrganization**.**Equals**(**\_nameOrganization**))** **{** **return** stack**[**i**];** **}**

**}**

**return** **null;**

**}**

**public** string getName**()**

**{**

string info **=** Convert**.**ToString**(**SP**);**

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

**{**

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

**{**

info **+=** "\n" **+** stack**[**i**].**NameOrganization**;**

**}**

**}**

**return** info**;**

**}**

**public** string getQueueFor**()**

**{**

string info **=** ""**;**

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

**{**

info **+=** "\n" **+** stack**[**i**].**getList**();**

**}**

**return** info**;**

**}**

**public** void delComp**()**//удаление организаций

**{**

**if** **(** SP **>** 0**)**

**{**

stack**[**SP **-** 1**]** **=** **null;**

SP**--;**

**}**

**}**

**public** string getQueue**()**//вывод информации по всей службе трудоустройства

**{**

string information **=** ""**;**

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

**{**

information **+=** stack**[**i**].**chek**();**

**}**

information **+=** "summ Vacantion = " **+** getSumQueue**();**

**return** information**;**

**}**

**public** int getSumQueue**()**//количество вакансий по всей службе

**{**

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

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

**{**

sum **+=** stack**[**i**].**SumVacantion**();**

**}**

**return** sum**;**

**}**

**}**

**}**

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

При запуске программы создаётся экземплар класса EmploymentService с передаваемым параметром интового заначения для указания вместимости стека.

**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** Third

**{**

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

**{**

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

**{**

InitializeComponent**();**

**}**

EmploymentService mains **=** **new** EmploymentService**(**10**);**

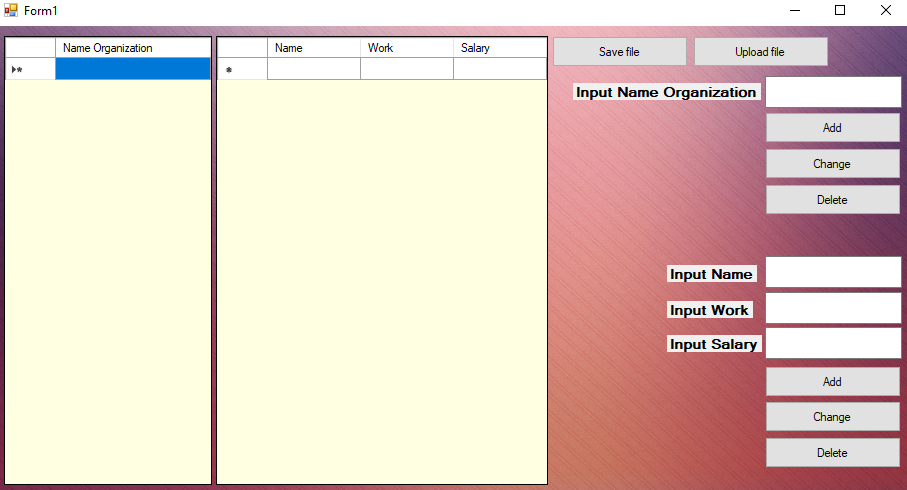


Рис Запуск программы.

**private** void backgroundWorker1\_DoWork**(object** sender**,** DoWorkEventArgs e**)**

**{**

**}**

**private** void Form1\_Load**(object** sender**,** EventArgs e**)**

**{**

**}**

Метод отвечаюхий за создание орнанизации с указанными польхователм параметрами.

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

**{**

mains**.**addComp**(**textOrg**.**Text**);**

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

**}**

При нажатии на кнопку данные оранизации обновляються.

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

**{**

mains**.**FindComp**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**NameOrganization **=** textOrg**.**Text**;**

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

**}**

При удалении оранизации очищается таблици и стирается экземпляр класса оранизации.

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

**{**

**if** **(**mains**.**getSP **!=** 0**)**

**{**

mains**.**delComp**();**

dataGridView1**.**Rows**.**RemoveAt**(**mains**.**getSP**);**

**}**

**else** MessageBox**.**Show**(**"Нечего удалять"**);**

**}**

Событие добавляющее ваканию в указанную оранизацию. Данные берутся из TextBox.

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

**{**

mains**.**FindComp**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**AddVacantion**(**textWork**.**Text**,** Convert**.**ToInt32**(**textSal**.**Text**),** textName**.**Text**);**

dataGridView2**.**Rows**.**Add**(**textName**.**Text**,** textWork**.**Text**,** textSal**.**Text**);**

**}**

При изменение данных меняются данные в таблице и в экземпляре класса.

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

**{**

string data1 **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

string data2 **=** dataGridView2**.**CurrentCell**.**Value**.**ToString**();**

Vacantion vspom **=** mains**.**FindComp**(**data1**).**search**(**data2**);**

vspom**.**SetName**(**textName**.**Text**);**

vspom**.**SetSalary**(**Convert**.**ToInt32**(**textSal**.**Text**));**

vspom**.**SetWork**(**textWork**.**Text**);**

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

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

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

**}**

При удалении удаляются даные как из таблици так и и з алрксного списка.

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

**{**

string data1 **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

string data2 **=** dataGridView2**.**CurrentCell**.**Value**.**ToString**();**

mains**.**FindComp**(**data1**).**DeleteVacantion**(**data2**);**

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

**}**

Метод выгрузки данных в файл формата txt. Для дальнейшего чтения в начале файла пишем "Файл, чтобы можно было идентифицировать файл программы от других txt.

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

**{**

string path**;**

string fileText **=** "Файл\n"**;**

//Создание экземпляра диалогового окна (Для того чтобы не занимать поток) + фильтр на создание текстовых файлов

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

**{**

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

**{**

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

fileText **+=** mains**.**getName**()** **+** "\n"**;**

fileText **+=** mains**.**getQueueFor**();**

//Та же работа с экземплярами файла и записи файла

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

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

**{**

stream**.**Write**(**fileText**);**

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 "Файл"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**,** LCount**,** salary**;**

**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**();**

//Проверка на пригодность файла для работы с программой

**if** **(**doIt **==** "Файл"**)**

**{**

**for** **(**int i **=** 0**;** i **<=** mains**.**getSP**;** i**++)**

**{**

mains**.**delComp**();**

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

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

**}**

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

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

**{**

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

mains**.**addComp**(**name**);**

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

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

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

i**++;**

**else**

**break;**

**}**

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

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

**{**

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

**if** **(**LCount **!=** 0**)**

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

**{**

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

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

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

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

mains**.**FindComp**(**name**).**AddVacantion**(**\_work**,** salary**,**\_name**);**

**if** **(**i **!=** LCount**)**

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

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

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

**}**

**}**

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

**{**

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

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

Organizations vspom **=** mains**.**FindComp**(**text**);**

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

**{**

SList copy **=** vspom**.**SHeader**;**

int chet **=** mains**.**FindComp**(**text**).**Count**;**

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

**{**

dataGridView2**.**Rows**.**Add**(**copy**.**data**.**GetName**(),** copy**.**data**.**GetWork**(),**copy**.**data**.**GetSalary**());**

**}**

**}**

**}**

**}**

**}**

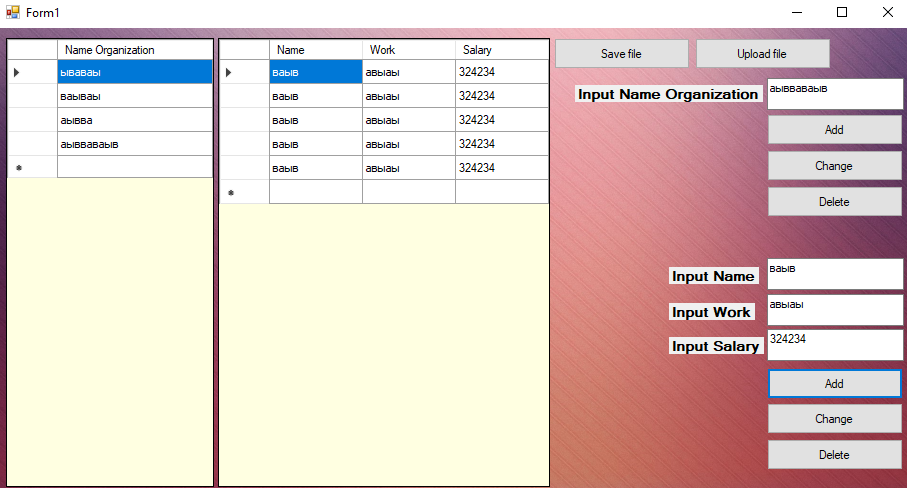


Рис Работа Программы.

1. Описание структуры проекта в соответствии с использованным инструментом разработки

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.

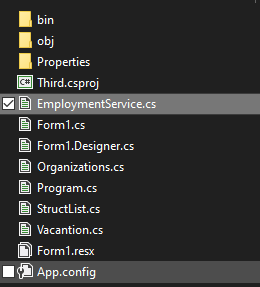


Рис. 7 Содержимое папки проекта.

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.

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.

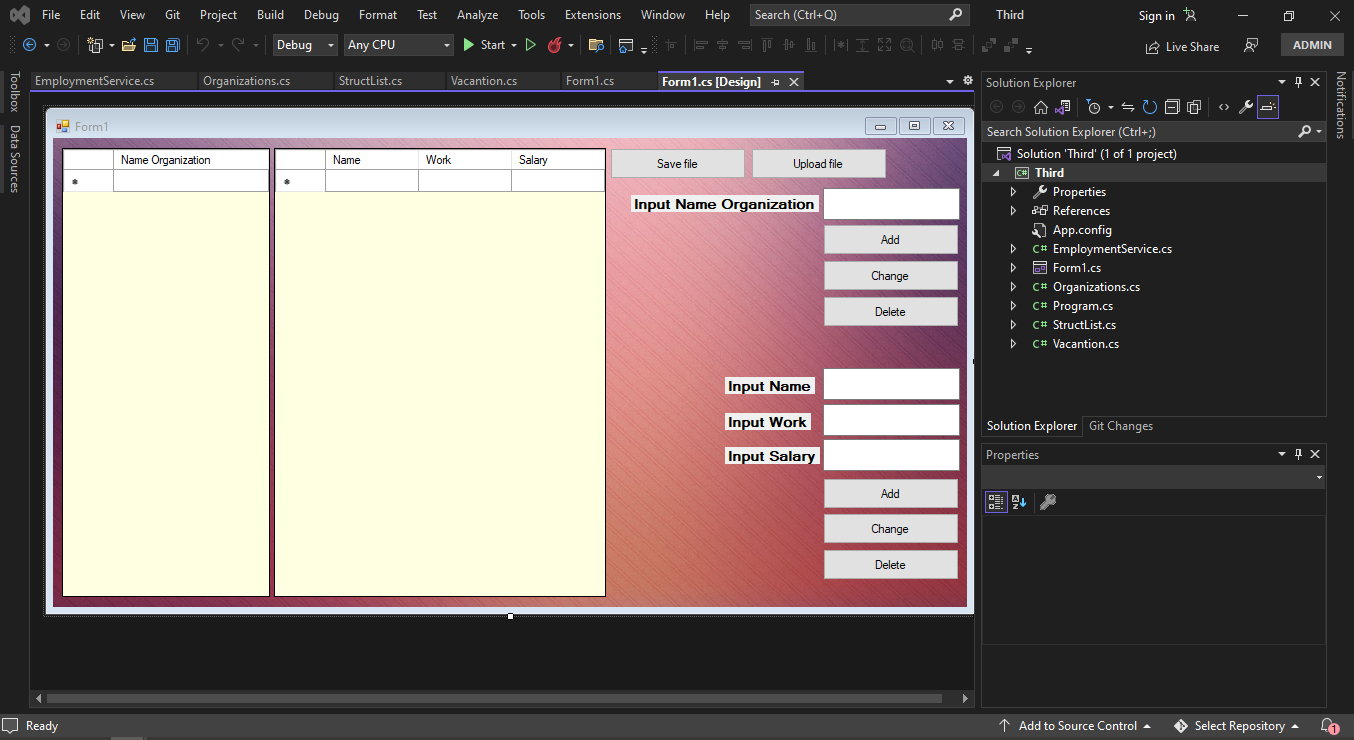


Рис. 8 Работа Visual Studio 2022

# **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

**using** System**;**

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

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

**namespace** Third

**{**

class EmploymentService

**{**

**private** Organizations**[]** stack**;**//массив типа Patient

**private** int SP**;**//вспомогательная переменная для добавления/удаления

**private** int startSize**;**//

**public** EmploymentService**(**int \_startSize**)**//конструктор

**{**

SP **=** 0**;**

startSize **=** \_startSize**;**

stack **=** **new** Organizations**[**startSize**];**

**}**

**public** int getSP **{** **get** **{** **return** SP**;** **}** **}**

**public** void addComp**(**string \_nameOrganization**)**//добавление организаций

**{**

**if(**SP **<=** startSize**)**

**{**

Organizations newsdob **=** **new** Organizations**(**\_nameOrganization**);**

stack**[**SP**]** **=** newsdob**;**

SP**++;**

**}**

**}**

**public** Organizations FindComp**(**string \_nameOrganization**)**//поиск организаций

**{**

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

**{**

**if** **(**stack**[**i**].**NameOrganization**.**Equals**(**\_nameOrganization**))** **{** **return** stack**[**i**];** **}**

**}**

**return** **null;**

**}**

**public** string getName**()**

**{**

string info **=** Convert**.**ToString**(**SP**);**

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

**{**

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

**{**

info **+=** "\n" **+** stack**[**i**].**NameOrganization**;**

**}**

**}**

**return** info**;**

**}**

**public** string getQueueFor**()**

**{**

string info **=** ""**;**

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

**{**

info **+=** "\n" **+** stack**[**i**].**getList**();**

**}**

**return** info**;**

**}**

**public** void delComp**()**//удаление организаций

**{**

**if** **(** SP **>** 0**)**

**{**

stack**[**SP **-** 1**]** **=** **null;**

SP**--;**

**}**

**}**

**public** string getQueue**()**//вывод информации по всей службе трудоустройства

**{**

string information **=** ""**;**

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

**{**

information **+=** stack**[**i**].**chek**();**

**}**

information **+=** "summ Vacantion = " **+** getSumQueue**();**

**return** information**;**

**}**

**public** int getSumQueue**()**//количество вакансий по всей службе

**{**

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

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

**{**

sum **+=** stack**[**i**].**SumVacantion**();**

**}**

**return** sum**;**

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** Third

**{**

class SList

**{**

**public** Vacantion data**;** //данные о организации

**private** SList next**;** //ссылка на потомка

**private** SList last**,** first**;**//заголовок списка

**public** SList**(**Vacantion \_data**)** //конструктор

**{**

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

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

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

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

**}**

**public** SList NEXT//методы доступа к переменной next

**{**

**get** **{** **return** next**;** **}**

**set** **{** **this.**next **=** **value;** **}**

**}**

**public** SList FIRST//методы доступа к переменной first

**{**

**get** **{** **return** first**;** **}**

**set** **{** **this.**first **=** **value;** **}**

**}**

**public** SList LAST//методы доступа к переменной last

**{**

**get** **{** **return** last**;** **}**

**set** **{** **this.**last **=** **value;** **}**

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** Third

**{**

class Organizations

**{**

**private** string nameOrganization**;**//название организации

**private** int count**;**//количество вакансий

**private** SList Header**,**pTemp**;**//первый элемент

**public** Organizations**(**string \_nameOrganization**)**//конструктор

**{**

nameOrganization **=** \_nameOrganization**;**

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

Header**.**FIRST **=** **null;**

Header**.**LAST **=** **null;**

pTemp **=** **null;**

**}**

**public** string NameOrganization

**{**

**get** **{** **return** nameOrganization**;** **}**

**set** **{** **this.**nameOrganization **=** **value;** **}**

**}**

**public** SList SHeader **{** **get** **{** **return** Header**.**FIRST**;** **}** **}**

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

**public** Vacantion search**(**string \_name**)**

**{**

SList vspom **=** Header**.**FIRST**;**

**while(**vspom**!=** **null)**

**{**

**if(**vspom**.**data**.**GetName**().**Equals**(**\_name**))**

**{**

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

**}**

**else** **{** vspom **=** vspom**.**NEXT**;** **}**

**}**

**return** **null;**

**}**

**public** string getList**()**

**{**

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

SList pTemp **=** Header**.**FIRST**;**

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

**{**

**if** **(**pTemp **!=** **null)**

**{**

data **+=** "\n" **+** pTemp**.**data**.**GetName**();**

data **+=** "\n" **+** pTemp**.**data**.**GetWork**();**

data **+=** "\n" **+** pTemp**.**data**.**GetSalary**();**

pTemp **=** pTemp**.**NEXT**;**

**}**

**}**

**return** data**;**

**}**

**public** void AddVacantion**(**string \_work**,** int \_salary**,** string \_name**)**//Метод добавдения вакансии

**{**

Vacantion news **=** **new** Vacantion**(**\_work**,** \_salary**,** \_name**);**

SList dob **=** **new** SList**(**news**);**

count**++;**

**if(**Header**.**FIRST **==** **null** **&&** Header**.**LAST **==** **null)**

**{**

Header**.**FIRST **=** dob**;**

Header**.**LAST **=** dob**;**

**}**

**else**

**{**

pTemp **=** Header**.**LAST**;**

Header**.**LAST **=** dob**;**

pTemp**.**NEXT **=** Header**.**LAST**;**

**}**

**}**

**public** bool DeleteVacantion**(**string \_work**)**//метод удаления

**{**

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

**{**

pTemp **=** Header**.**FIRST**;**

**while** **(**pTemp **!=** **null)**

**{**

**if** **(**pTemp **==** Header**.**FIRST**)**

**{**

**if** **(**pTemp**.**data**.**GetName**().**Equals**(**\_work**))**

**{**

count**--;**

Header**.**FIRST **=** Header**.**FIRST**.**NEXT**;**

**return** **true;**

**}**

**else** pTemp **=** pTemp**.**NEXT**;**

**}**

**else**

**{**

SList vspom **=** Header**.**FIRST**;**

**if** **(**pTemp**.**data**.**GetName**().**Equals**(**\_work**))**

**{**

count**--;**

vspom**.**NEXT **=** pTemp**.**NEXT**;**

**return** **true;**

**}**

**else** **{** pTemp **=** pTemp**.**NEXT**;** vspom **=** vspom**.**NEXT**;** **}**

**}**

**}**

**}**

**return** **false;**

**}**

**public** int SumVacantion**()**//суммарное кол-во вакансий

**{**

**return** count**;**

**}**

**public** string chek**()** //вывод данных

**{**

string info **=** "\n------" **+** nameOrganization **+** " present"**;**

pTemp **=** Header**.**FIRST**;**

**while** **(**pTemp **!=** **null)**

**{**

info **+=** "\nVacantion - " **+** pTemp**.**data**.**GetName**()** **+** ". Required lenght of service - " **+** pTemp**.**data**.**GetWork**();**

info **+=** "\nSalary - " **+** pTemp**.**data**.**GetSalary**();**

pTemp **=** pTemp**.**NEXT**;**

**}**

info **+=** "\nSumm count vacantion - " **+** SumVacantion**()** **+** "\n\n"**;**

**return** info**;**

**}**

**}**

**}**

**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** Third

**{**

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

**{**

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

**{**

InitializeComponent**();**

**}**

EmploymentService mains **=** **new** EmploymentService**(**10**);**

**private** void backgroundWorker1\_DoWork**(object** sender**,** DoWorkEventArgs e**)**

**{**

**}**

**private** void Form1\_Load**(object** sender**,** EventArgs e**)**

**{**

**}**

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

**{**

mains**.**addComp**(**textOrg**.**Text**);**

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

**}**

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

**{**

mains**.**FindComp**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**NameOrganization **=** textOrg**.**Text**;**

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

**}**

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

**{**

**if** **(**mains**.**getSP **!=** 0**)**

**{**

mains**.**delComp**();**

dataGridView1**.**Rows**.**RemoveAt**(**mains**.**getSP**);**

**}**

**else** MessageBox**.**Show**(**"Нечего удалять"**);**

**}**

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

**{**

mains**.**FindComp**(**dataGridView1**.**CurrentCell**.**Value**.**ToString**()).**AddVacantion**(**textWork**.**Text**,** Convert**.**ToInt32**(**textSal**.**Text**),** textName**.**Text**);**

dataGridView2**.**Rows**.**Add**(**textName**.**Text**,** textWork**.**Text**,** textSal**.**Text**);**

**}**

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

**{**

string data1 **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

string data2 **=** dataGridView2**.**CurrentCell**.**Value**.**ToString**();**

Vacantion vspom **=** mains**.**FindComp**(**data1**).**search**(**data2**);**

vspom**.**SetName**(**textName**.**Text**);**

vspom**.**SetSalary**(**Convert**.**ToInt32**(**textSal**.**Text**));**

vspom**.**SetWork**(**textWork**.**Text**);**

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

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

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

**}**

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

**{**

string data1 **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

string data2 **=** dataGridView2**.**CurrentCell**.**Value**.**ToString**();**

mains**.**FindComp**(**data1**).**DeleteVacantion**(**data2**);**

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

**}**

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

**{**

string path**;**

string fileText **=** "Файл\n"**;**

//Создание экземпляра диалогового окна (Для того чтобы не занимать поток) + фильтр на создание текстовых файлов

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

**{**

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

**{**

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

fileText **+=** mains**.**getName**()** **+** "\n"**;**

fileText **+=** mains**.**getQueueFor**();**

//Та же работа с экземплярами файла и записи файла

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

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

**{**

stream**.**Write**(**fileText**);**

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

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

**}**

**}**

**}**

**}**

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

**{**

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

int Count**,** LCount**,** salary**;**

**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**();**

//Проверка на пригодность файла для работы с программой

**if** **(**doIt **==** "Файл"**)**

**{**

**for** **(**int i **=** 0**;** i **<=** mains**.**getSP**;** i**++)**

**{**

mains**.**delComp**();**

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

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

**}**

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

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

**{**

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

mains**.**addComp**(**name**);**

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

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

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

i**++;**

**else**

**break;**

**}**

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

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

**{**

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

**if** **(**LCount **!=** 0**)**

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

**{**

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

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

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

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

mains**.**FindComp**(**name**).**AddVacantion**(**\_work**,** salary**,**\_name**);**

**if** **(**i **!=** LCount**)**

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

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

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

**}**

**}**

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

**{**

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

string text **=** dataGridView1**.**CurrentCell**.**Value**.**ToString**();**

Organizations vspom **=** mains**.**FindComp**(**text**);**

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

**{**

SList copy **=** vspom**.**SHeader**;**

int chet **=** mains**.**FindComp**(**text**).**Count**;**

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

**{**

dataGridView2**.**Rows**.**Add**(**copy**.**data**.**GetName**(),** copy**.**data**.**GetWork**(),**copy**.**data**.**GetSalary**());**

**}**

**}**

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** Third

**{**

class Vacantion

**{**

**private** string work**;**//стаж работы

**private** int salary**;**//зарплата

**private** string name**;**//название вакансии

**public** Vacantion**(**string \_work**,** int \_salary**,**string \_name**)**//конструктор

**{**

**this.**work **=** \_work**;**

**this.**salary **=** \_salary**;**

**this.**name **=** \_name**;**

**}**

**public** string GetWork**()**//методы доступа для переменной work

**{**

**return** work**;**

**}**

**public** void SetWork**(**string newWork**)**

**{**

**this.**work **=** newWork**;**

**}**

**public** int GetSalary**()**////методы доступа для переменной salary

**{**

**return** salary**;**

**}**

**public** void SetSalary**(** int newSalary**)**

**{**

**this.**salary **=** newSalary**;**

**}**

**public** void SetName**(**string \_name**)**////методы доступа для переменной name

**{**

**this.**name **=** \_name**;**

**}**

**public** string GetName**()**

**{**

**return** name**;**

**}**

**}**

**}**