УВО «Университет Управления «ТИСБИ»

Факультет информационных технологий

Кафедра информационных технологий

**Курсовая работа**

по дисциплине «Объектно-ориентированное программирование»

на тему: «Разработка объектной программы для задачи «Расписание киносеансов»

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

Проверил: ст. преп. Якунина Е.А.

Казань 2022

Оглавление

[Постановка задачи 3](#_Toc105284107)

[1. Описание используемых структур данных с алгоритмами выполнения основных операций 5](#_Toc105284108)

[1.1 Кольцевая очередь. 5](#_Toc105284109)

[1.2 Адресный стек 6](#_Toc105284110)

[2. Краткие сведения об объектном подходе 8](#_Toc105284111)

[3. Формализованное описание разработанных классов 14](#_Toc105284112)

[4. Описание демонстрационного модуля с характеристикой использованных стандартных компонентов и списком реализованных обработчиков 18](#_Toc105284113)

[5. Описание структуры проекта в соответствии с использованным инструментом разработки 23](#_Toc105284114)

[Список литературы 25](#_Toc105284115)

[Листинг программы 27](#_Toc105284116)

Постановка задачи

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

**Постановка задачи:** разработать объектную программу кинотеатр для составления расписания киносеансов организует список демонстрируемых кинофильмов с указанием названия и продолжительности показа (в минутах). Для каждого кинофильма создаётся список с указанием даты и времени показа.

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 кинофильмов используется структура данных в виде кольцевой очередь на основе обычного массива. Для объединения сеансов каждого кинофильма используется структура данных в виде адресного стека.

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 Кольцевая очередь.

Круговые очереди позволяют избежать пустых пространств, которые появляются в обычных очередях, реализованных с помощью массивов.

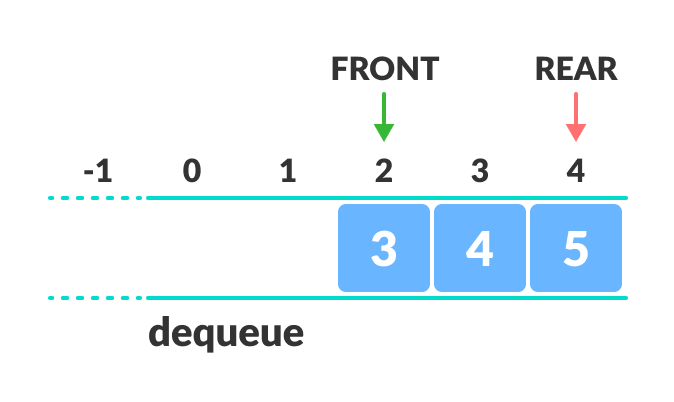


Рис. 1 Кольцевая очередь

После нескольких операций enqueue и dequeue образуются пустые пространства.

Места в очереди с индексом 0 и 1 использовать повторно можно только после сброса очередей в исходное состояние. То есть, после удаления всех элементов.

Круговая очередь работает по принципу циклического инкрементирования. Это значит, что, когда мы пытаемся увеличить значение указателя, который находится в конце очереди, он перемещается в начало этой очереди.

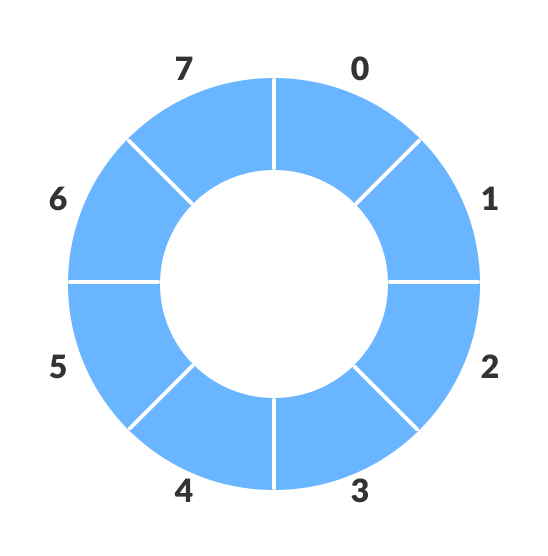


Рис. 2 Кольцевая очередь.

Принцип работы круговой очереди

Круговая очередь работает следующим образом:

Объявляем два указателя — REAR и FRONT.

FRONT указывает на первый элемент очереди.

REAR указывает на последний элемент очереди.

При инициализации круговой очереди присваиваем REAR и FRONT значение -1.

1. Операция enqueue

Проверяем, не заполнена ли очередь.

После добавления первого элемента присваиваем указателю FRONT значение 0.

Циклически инкрементируем значение REAR на 1. Когда указатель окажется в конце очереди, он должен переместится в ее начало.

Добавляем новый элемент на ту позицию, куда указывает REAR.

2. Операция dequeue

Проверяем, не пуста ли очередь.

Получаем значение, на которое указывает FRONT.

Циклически инкрементируем значение FRONT на 1.

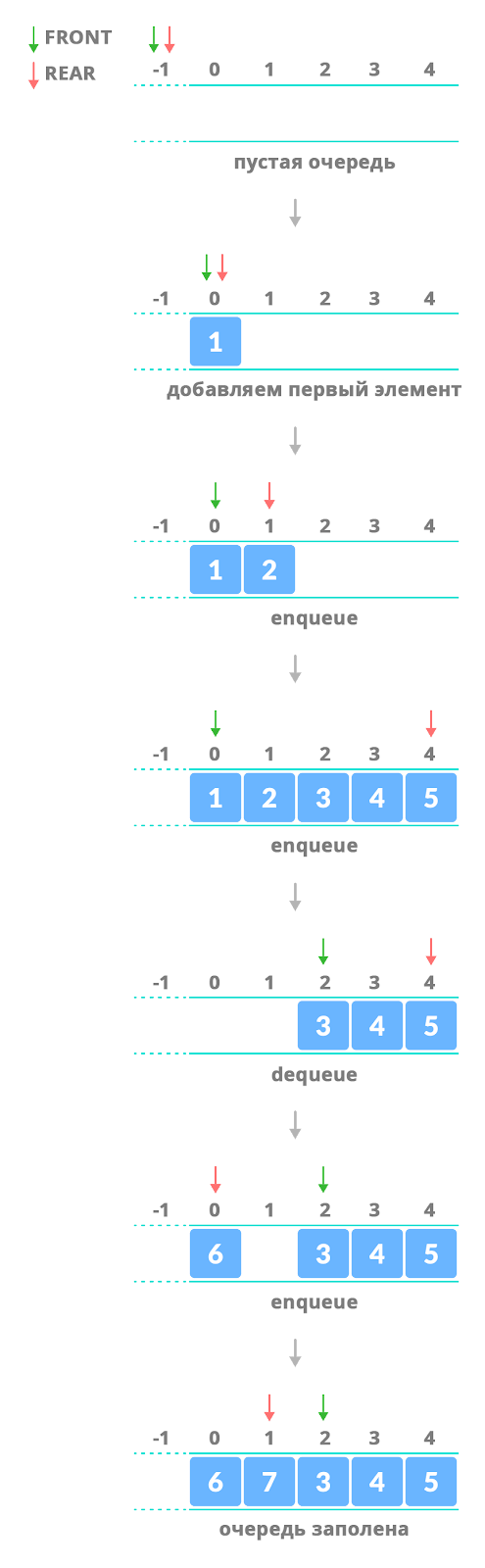
После добавления последнего элемента сбрасываем значения FRONT и REAR — присваиваем им -1.

Проверка на заполненность

Вариант 1: FRONT = 0 && REAR == SIZE - 1

Вариант 2: FRONT = REAR + 1

Рассмотрим второй вариант. Если указатель REAR перемещается в начало очереди и позже его значение становится на 1 меньше, чем FRONT, значит, очередь заполнена.

  
Рис. 3 Алгоритм кольцевой очереди

1.2 Адресный стек

Стек (англ. stack – стопка) – это структура данных, в которой новый элемент всегда записывается в ее начало (вершину) и очередной читаемый элемент также всегда выбирается из ее начала. В стеках используется метод доступа к элементам LIFO ( Last Input – First Output, "последним пришел – первым вышел"). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно сначала взять верхнюю.

Стек – это список, у которого доступен один элемент (одна позиция). Этот элемент называется вершиной стека. Взять элемент можно только из вершины стека, добавить элемент можно только в вершину стека. Например, если записаны в стек числа 1, 2, 3, то при последующем извлечении получим 3,2,1.

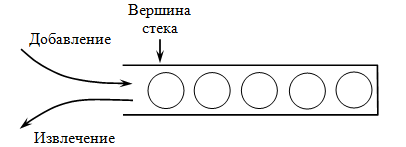


Рис. 4 Принцип стека.

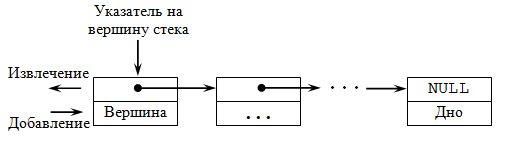


Рис. 5 Структура адресного стека.

Стек как динамическую структуру данных легко организовать на основе линейного списка. Поскольку работа всегда идет с заголовком стека, то есть не требуется осуществлять просмотр элементов, удаление и вставку элементов в середину или конец списка, то достаточно использовать экономичный по памяти линейный однонаправленный список. Для такого списка достаточно хранить указатель вершины стека, который указывает на первый элемент списка. Если стек пуст, то списка не существует, и указатель принимает значение NULL.

Возможны три операции со стеком: добавление элемента (иначе проталкивание, push), удаление элемента (pop) и чтение головного элемента (peek).

При проталкивании (push) добавляется новый элемент, указывающий на элемент, бывший до этого головой. Новый элемент теперь становится головным.

При удалении элемента (pop) убирается первый, а головным становится тот, на который был указатель у этого объекта (следующий элемент). При этом значение убранного элемента возвращается.

1. Краткие сведения об объектном подходе

Объектно-ориентированное программирование – одно из важнейших методологий разработки, которая основано на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Основные принципы ООП:

Инкапсуляция

Одним из самых важных факторов при проектировании компонентов приложения является сокрытие внутренних данных компонента и деталей его реализации от других компонентов приложения и предоставление набора методов для взаимодействия с ним (API). Этот принцип является одним из четырёх фундаментальных принципов ООП и называется инкапсуляцией.

Правильная инкапсуляция важна по многим причинам:

1. Она способствует переиспользованию компонентов: поскольку в этом случае компоненты взаимодействуют друг с другом только посредством их API и безразличны к изменениям внутренней структуры, они могут использоваться в более широком контексте.

2. Инкапсуляция ускоряет процесс разработки: слабо связанные друг с другом компоненты (то есть компоненты, чей код как можно меньше обращается или использует код других компонентов) могут разрабатываться, тестироваться и дополняться независимо.

3. Правильно инкапсулированные компоненты более легки для понимания и процесса отладки, что упрощает поддержку приложения.

В языке Java инкапсуляция реализована с помощью системы классов, которые позволяют собрать информацию об объекте в одном месте; пакетов, которые группируют классы по какому-либо критерию, и модификаторов доступа, которыми можно пометить весь класс или его поле или метод.

Всего модификаторов доступа четыре:

public – полный доступ к сущности (полю или методу класса) из любого пакета;

protected – доступ к сущности только для классов своего пакета и наследников класса;

неявный модификатор по умолчанию (при отсутствии трёх явных) – доступ к сущности только для классов своего пакета;

private – доступ только внутри класса, в котором объявлена сущность.

Для достижения правильной инкапсуляции также необходимо предоставить корректный API для работы с компонентом. Например, в сеттер для переменной можно включить логику по проверке передаваемых значений либо не предоставлять сеттеры в классе вовсе, если класс должен быть доступен только для чтения.

Пример корректной инкапсуляции класса:

В примере выше значение переменной name задаётся при создании объекта и не может быть изменено извне, так как сеттер для переменной отсутствует. В сеттере для переменной age реализована проверка на корректность передаваемого параметра и выброс исключения при неверном значении.

Наследование

Наследование является одним из важнейших принципов объектно-ориентированного программирования, поскольку оно позволяет создавать иерархические структуры объектов. Используя наследование, можно создать общий класс, который будет определять характеристики и поведение, свойственные какому-то набору связанных объектов. В дальнейшем этот класс может быть унаследован другими, более частными классами, каждый из которых будет добавлять уникальные, свойственные только ему характеристики и дополнять или изменять поведение базового класса. В терминах Java такой общий класс называется суперклассом (superclass), или базовым классом (base class), или классом-родителем (parent class), а класс, его наследующий, - подклассом (subclass), или дочерним классом (child class), или классом-потомком (derived class).

Наследование реализует отношение «является» (“is-a”) между суперклассом и подклассом. Пусть, например, классы Employee и Manager представляют собой абстракцию понятий «Сотрудник» и «Менеджер». Каждый менеджер также является сотрудником компании, в которой он работает, следовательно, класс Manager находится в отношении “is-a” с классом Employee. Таким образом, с точки зрения наследования, при выстраивании иерархии классов класс Employee будет являться суперклассом, а класс Manager – дочерним классом. При этом класс, который является наследником какого-либо класса, может быть суперклассом для одного или нескольких других классов. Также в отличие, например, от C++, в Java в строгом смысле отсутствует множественное наследование, то есть любой класс может иметь не более одного класса-родителя. А все классы, суперкласс у которых явно не указан, по умолчанию наследуются от класса Object.

Класс Employee в примере выше является суперклассом не потому, что он главнее класса Manager или содержит больше функциональности. На самом деле верно обратное: функциональность подклассов не ýже, а зачастую существенно шире, чем функциональность их классов-родителей. Приставки «супер-» и «под-» пришли в Java из математики: множество всех менеджеров содержится во множестве всех сотрудников и, таким образом, является подмножеством множества сотрудников.

Для того чтобы унаследовать какой-либо класс в Java, используется ключевое слово extends:

В примере выше класс Employee является базовым классом для класса Manager, а класс Manager – подклассом класса Employee. Класс Employee абстрагирует базовые характеристики для всех сотрудников компании – имя, фамилию, размер оклада и дату приёма на работу, а класс Manager дополняет эти характеристики процентом премии для менеджеров и меняет поведение метода getSalary() базового класса, используя полиморфизм.

Полиморфизм

При рассмотрении полиморфизма необходимо помнить, что этот принцип неразрывно связан с другим принципом ООП – наследованием, которое помогает реализовать полиморфизм.

Возьмем для примера абстрактный класс «Автомобиль», который наследуют два конкретных класса – «Спортивный автомобиль» и «Грузовой автомобиль».

И спортивные, и грузовые автомобили будут обладать общими характеристиками и будут иметь возможность выполнять общие для всех автомобилей действия, которые указаны в абстрактном классе-родителе, но конкретная реализация этих действий может быть разной.

Например, общее для всех автомобилей действие «завестись» у спортивного автомобиля может быть реализовано путем нажатия кнопки, а у грузового - с помощью ключа. Один результат – разные решения. В этом и состоит полиморфизм.

Более строго, полиморфизм - один из принципов ООП, позволяющий вызовом переопределённого метода через переменную класса-родителя получить поведение, которое будет соответствовать реальному классу-потомку, на который ссылается эта переменная.

Код выше представляет пример полиморфизма. Сначала переменной родительского класса Vehicle присваивается объект дочернего класса SportCar. При вызове метода start() на консоль будет выведено: "Starting my fancy sport car!"

При дальнейшем присвоении этой же переменной объекта дочернего класса Truck и вызове того же метода start() на консоль будет выведено: "Starting my heavy truck!"

Абстракция

Относительно недавно в качестве самостоятельного четвёртого принципа начали выделять абстракцию.

Одно из определений слова «абстракция», которые можно встретить в современных словарях:

Абстракция (от лат. abstractio — выделение, отвлечение или отделение) — теоретический прием исследования, позволяющий отвлечься от некоторых несущественных в определенном отношении свойств изучаемых явлений и выделить свойства существенные и определяющие.

Все языки программирования предоставляют их пользователю определённые абстракции. Так, языки семейства ассемблер являются в своём роде абстракцией соответствующих микропроцессоров, поскольку позволяют отвлечься от деталей их реализации и общаться с ними через определённый набор более высокоуровневых инструкций. Императивные языки программирования, последовавшие за ассемблером, например Basic, Fortran, C, являются более высоким уровнем абстракции над ассемблерными языками – они дают возможность использовать более привычные человеку синтаксические конструкции за счёт приближения синтаксиса к естественным языкам.

Объектно-ориентированные языки, такие как Java, выводят разработку на ещё более высокий уровень абстракции: объекты в ООП по своей сути представляют собой модели понятий окружающего мира, таких как Работник, Сервер, Запись в дневнике, и выделяют только те свойства этого понятия, которые необходимы в конкретном случае для решения конкретной проблемы.

Например, класс Student в приложении учёта студентов университета, кроме общих полей, таких как имя, фамилия, дата рождения и т.д., будет содержать поля, отражающие информацию о номере зачётной книжки, статусе студента (действующий, академический отпуск, отчислен), факультете, номере его группы, оценках за семестры и т.д. Но для того же класса Student в приложении учёта студентов в тренинг-центре EPAM такая информация будет неактуальна: класс будет содержать поля, отражающие учебный проект, на который был распределён студент, уровень его английского языка по результатам последнего тестирования, количество посещаемых мероприятий и т.д.

В этом и состоит абстракция: фокусировка разработчика на конкретных свойствах объекта зависит от тех задач, которые призван решать объект. Следствием такого подхода является то, что, если в императивных языках разработчику необходимо думать в терминах компьютерной логики, в объектно-ориентированных языках разработчик думает в терминах проблемной области, в которой он разрабатывает приложения.

3. Формализованное описание разработанных классов

У каждого класса есть сеанс поэтому первым делом реализуем класс сеанс с датой сеанса и продолжительностью показа фильма.

**using** System**;**

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

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

**namespace** Third

**{**

class Seans

**{**

**private** string date**;**//дата показа фильма

**private** int time**;**//время показа фильма

**public** Seans**(**string \_date**,** int \_time**)**

**{**

**this.**date **=** \_date**;**

**this.**time **=** \_time**;**

**}**

**public** string getDate**()**

**{**

**return** date**;**

**}**

**public** void setDate**(**string \_date**)**

**{**

**this.**date **=** \_date**;**

**}**

**public** int getTime**()**

**{**

**return** time**;**

**}**

**public** void setTime**(**int \_time**)**

**{**

**this.**time **=** \_time**;**

**}**

**}**

**}**

Класс фильмы содержит в себе: сеансы, название фильма. Внутри класса идёт работа с такой структурой данных как адресный стек.

**using** System**;**

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

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

**namespace** Third

**{**

class Film

**{**

**private** Stack firstSeans**;**//первый сеанс

**private** string nameFilm**;**//название фильма

**private** int durationFilm**;**//длина фильма

**private** Stack SP**;**

**private** int count**;**

**public** Film**(**string \_nameFilm**,** int \_durationFilm**)**

**{**

**this.**nameFilm **=** \_nameFilm**;**

**this.**durationFilm **=** \_durationFilm**;**

SP **=** **null;**

**}**

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

**public** string Name **{** **get** **{** **return** nameFilm**;** **}** **set** **{** **this.**nameFilm **=** **value;** **}** **}**

**public** int Durat **{** **get** **{** **return** durationFilm**;** **}** **set** **{** **this.**durationFilm **=** **value;** **}** **}**

**public** Stack getSPEl **{** **get{** **return** SP**;** **}** **}**

**public** void addSeans**(**string \_date**,** int \_time**)**

**{**

Seans newSea **=** **new** Seans**(**\_date**,**\_time**);**

Stack pTemp **=** **new** Stack**(**newSea**,**SP**);**

**if** **(**SP **==** **null)** **{** firstSeans **=** pTemp**;}**

SP **=** pTemp**;**

count**++;**

**}**

**public** string getNameFilm**()** **{** **return** nameFilm**;** **}**

**public** bool delSeans**()**

**{**

**if** **(**SP **!=** **null)**

**{**

SP **=** SP**.**getPred**();**

count**--;**

**return** **true;**

**}**

**else** **return** **false;**

**}**

**public** int getSumStack**()**

**{**

**return** durationFilm**\***count**;**

**}**

**public** String getInfoStack**()**

**{**

String info **=** Convert**.**ToString**(**Count**);**

Stack pTemp **=** SP**;**

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

**{**

info **+=** "\n" **+** pTemp**.**\_aDate**.**getDate**();**

info **+=** "\n" **+** pTemp**.**\_aDate**.**getTime**();**

pTemp **=** pTemp**.**getPred**();**

**}**

**return** info**;**

**}**

**public** Seans FindSeans**(**string \_nameSeans**)**

**{**

Stack vspom = SP;

while(vspom != null)

{

if(vspom.\_aDate.getDate().Equals(\_nameSeans))

{

return vspom.\_aDate;

}

else { vspom = vspom.getPred(); }

}

return null;

}

}

}

Класс стек реализует схему работы со стеком который будет хранить сеансы.

**using** System**;**

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

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

**namespace** Third

**{**

class Stack

**{**

**public** Seans \_aDate**;**

**private** Stack pred**;**

**public** Stack**(**Seans \_date**,** Stack \_SP**)**

**{**

\_aDate **=** \_date**;**

pred **=** \_SP**;**

**}**

**public** Stack getPred**()**

**{**

**return** pred**;**

**}**

**}**

**}**

Класс кинотеатр реализует контейнер для хранения фильмов. Методы класса предоставляют доступ к методам других классов.

**using** System**;**

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

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

**namespace** Third

**{**

class Cinema

**{**

**private** Film**[]** Queue**;**//реализация очереди на основе массива

**private** static int size **=** 10**;**

**private** int countFilm **=** 0**;**

**private** int first**;**//ссылка на первый элемент в очереди

**private** int last**;**//ссылка на последний элемент в очереди

**public** Cinema**()**

**{**

first **=** 0**;**

last **=** 0**;**

Queue **=** **new** Film**[**size**];**

**}**

**public** string addFilm**(**string \_nameFilm**,** int \_durationFilm**)**

**{**

**if** **(**size **>=** countFilm**)**

**{**

**if** **(**last **>** size **&&** first **>** 0**)** **{**last **=** 0**;}**

Film newFilm **=** **new** Film**(**\_nameFilm**,** \_durationFilm**);**

Queue**[**last**]** **=** newFilm**;**

last**++;**

countFilm**++;**

**return** \_nameFilm**;**

**}**

**return** ""**;**

**}**

**public** bool delFilm**()**

**{**

**if** **(** countFilm **!=** 0**)**

**{**

**if** **(**first **>** size**)** **{** first **=** 0**;}**

Queue**[**first**]** **=** **null;**

countFilm**--;**

first**++;**

**return** **true;**

**}**

**return** **false;**

**}**

**public** int COUNT **{** **get** **{** **return** countFilm**;** **}** **}**

**public** string getinfo**()**

**{**

string inform **=** ""**;**

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

**{**

inform **+=** "\n" **+** Queue**[**i**].**getInfoStack**();**

**}**

**return** inform**;**

**}**

**public** string getinfoName**()**

**{**

string inform **=** Convert**.**ToString**(**COUNT**);**

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

**{**

inform **+=** "\n" **+** Queue**[**i**].**Name**;**

inform **+=** "\n" **+** Queue**[**i**].**Durat**;**

**}**

**return** inform**;**

**}**

**public** Film getFirst**()** **{** **return** Queue**[**first**];** **}**

**public** Film FindFilm**(**string \_nameFilm**)**

**{**

Film pTemp**;**

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

**{**

pTemp **=** Queue**[**i**];**

**if** **(**pTemp **!=** **null** **&&** pTemp**.**getNameFilm**().**Equals**(**\_nameFilm**))**

**{**

**return** pTemp**;**

**}**

**}**

**return** **null;**

**}**

**public** bool getSeansToEdit**(**string \_date**,** int \_time**,** String \_nameFilm**)**

**{**

int i **=** 0**;**

Film pTemp **=** Queue**[**i**];**

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

**{**

**if** **(**pTemp **!=** **null** **&&** pTemp**.**getNameFilm**().**Equals**(**\_nameFilm**))**

**{**

pTemp**.**addSeans**(**\_date**,** \_time**);**

**return** **true;**

**}**

**else**

**{**

i**++;**

pTemp **=** Queue**[**i**];**

**}**

**}**

**return** **false;**

**}**

**}**

**}**

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

При запуске программы создаётся экземпляр класса Кинотеатр с названием компании необходимое для создания файлов сохранения данных в файл.

**using** System**;**

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

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

**namespace** Third

**{**

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

**{**

Cinema news **=** **new** Cinema**();**

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

**{**

InitializeComponent**();**

**}**

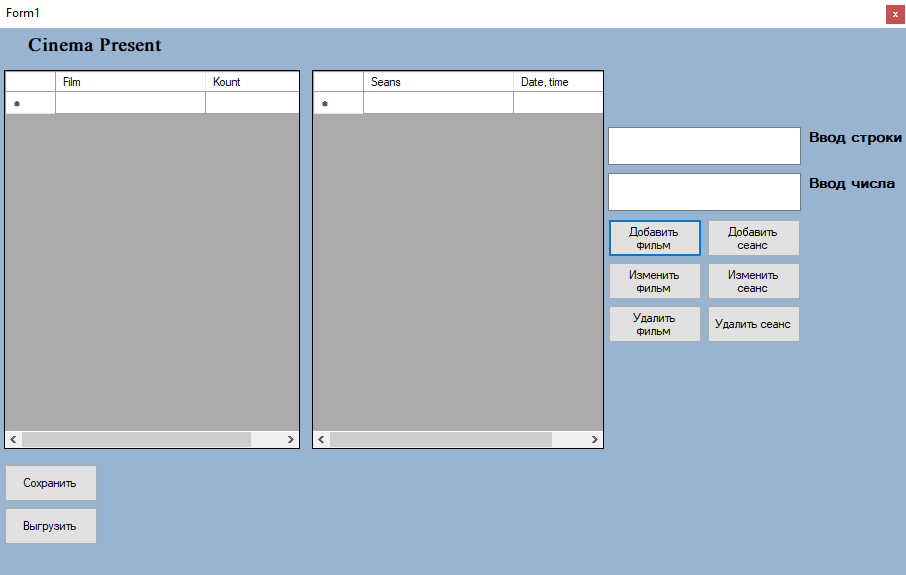


Рис. Запуск программы.

Обрабатываем удаление сеанса. Удаляем из стека данные и очищаем таблицу.

**private** void button4\_Click**(object** sender**,** EventArgs e**)**

**{**

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

Film vspom **=** news**.**FindFilm**(**text**);**

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

**{**

vspom**.**delSeans**();**

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

**}**

**else** **{** MessageBox**.**Show**(**"Выберите фильм, сеанс которого хотите удалить"**);** **}**

**}**

Кнопка добавления фильма. Читаем данные с TextBox и записываем данные в таблицу и структуру фильмов.

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

**{**

news**.**addFilm**(**textBox**.**Text**,** Convert**.**ToInt32**(**textBox1**.**Text**));**

dataGridView1**.**Rows**.**Add**(**textBox**.**Text**,** textBox1**.**Text**);**

textBox**.**Clear**();**

textBox1**.**Clear**();**

**}**

**private** void listBox2\_SelectedIndexChanged**(object** sender**,** EventArgs e**){}**

Отрабатываем добавление сеанса. Обновляем таблицу сеансов и добавляем у структуру экземпляр от класса сеанс.

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

**{**

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

news**.**getSeansToEdit**(**textBox**.**Text**,** Convert**.**ToInt32**(**textBox1**.**Text**),** text**);**

dataGridView2**.**Rows**.**Add**(**textBox**.**Text**,** textBox1**.**Text**);**

textBox**.**Clear**();**

textBox1**.**Clear**();**

**}**

Отрабатываем событие выбора фильма. При выборе фильма обновляем вторую таблицу с сеансами и отображаем соответствующие данные.

**private** void dataGridView2\_CellContentClick**(object** sender**,** DataGridViewCellEventArgs e**){}**

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

**{**

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

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

Film vspom **=** news**.**FindFilm**(**text**);**

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

**{**

Stack pTemp **=** vspom**.**getSPEl**;**

int chet **=** news**.**FindFilm**(**text**).**Count**;**

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

**{**

dataGridView2**.**Rows**.**Add**(**pTemp**.**\_aDate**.**getDate**(),** Convert**.**ToString**(**pTemp**.**\_aDate**.**getTime**()));**

pTemp **=** pTemp**.**getPred**();**

**}**

**}**

**}**

Отрабатываем нажатие кнопки изменения данных фильма. Читаем данные с TextBox и изменяем выбранную ячейку в таблице и изменяем данные с очереди фильмов.

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

**{**

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

int index **=** dataGridView1**.**CurrentRow**.**Index**;**

news**.**FindFilm**(**text**).**Name **=** textBox**.**Text**;**

news**.**FindFilm**(**textBox**.**Text**).**Durat **=** Convert**.**ToInt32**(**textBox1**.**Text**);**

dataGridView1**.**Rows**[**index**].**Cells**[**0**].**Value **=** textBox**.**Text**;**

dataGridView1**.**Rows**[**index**].**Cells**[**1**].**Value **=** textBox1**.**Text**;**

**}**

Событие изменение данных сеанса.

**private** void button2\_Click**(object** sender**,** EventArgs e**)**

**{**

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

int index **=** dataGridView2**.**CurrentRow**.**Index**;**

news**.**FindFilm**(**text**).**FindSeans**(**dataGridView2**.**CurrentCell**.**Value**.**ToString**()).**setDate**(**textBox**.**Text**);**

news**.**FindFilm**(**text**).**FindSeans**(**textBox**.**Text**).**setTime**(**Convert**.**ToInt32**(**textBox1**.**Text**));**

dataGridView2**.**Rows**[**index**].**Cells**[**0**].**Value **=** textBox**.**Text**;**

dataGridView2**.**Rows**[**index**].**Cells**[**1**].**Value **=** textBox1**.**Text**;**

**}**

Удаление фильма из таблицы и из очереди.

**private** void button1\_Click**(object** sender**,** EventArgs e**)**

**{**

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

**if** **(**news**.**getFirst**().**Name**.**Equals**(**text**))**

**{**

dataGridView1**.**Rows**.**RemoveAt**(**dataGridView1**.**CurrentRow**.**Index**);**

news**.**delFilm**();**

**}**

**else** **{** MessageBox**.**Show**(**"Выберите первый в списке фильм для удаления"**);** **}**

**}**

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

**{**

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

int Count**,** LCount**,** gName**;**

**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 **<=** news**.**COUNT**;** i**++)**

**{**

news**.**delFilm**();**

**}**

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

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

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

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

**{**

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

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

news**.**addFilm**(**nameLH**,** seats**);**

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

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

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

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

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

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

news**.**FindFilm**(**name**).**addSeans**(**inf**,**gName**);**

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

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

**else**

**{**

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

**}**

stream3**.**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 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 **+=** news**.**getinfoName**()** **+** "\n"**;**

**for(**int i **=** 0**;** i **<** news**.**COUNT**;** i**++)**

**{**

fileText **+=** news**.**getinfo**();**

**}**

//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**();**

**}**

**}**

**}**

**}**

**}**

**}**

1. Описание структуры проекта в соответствии с использованным инструментом разработки

Microsoft Visual Studio — линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, UWP а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET, MAUI, Xbox, Windows Phone .NET Compact Framework и Silverlight. После покупки компании Xamarin корпорацией Microsoft появилась возможность разработки IOS и Android программ.

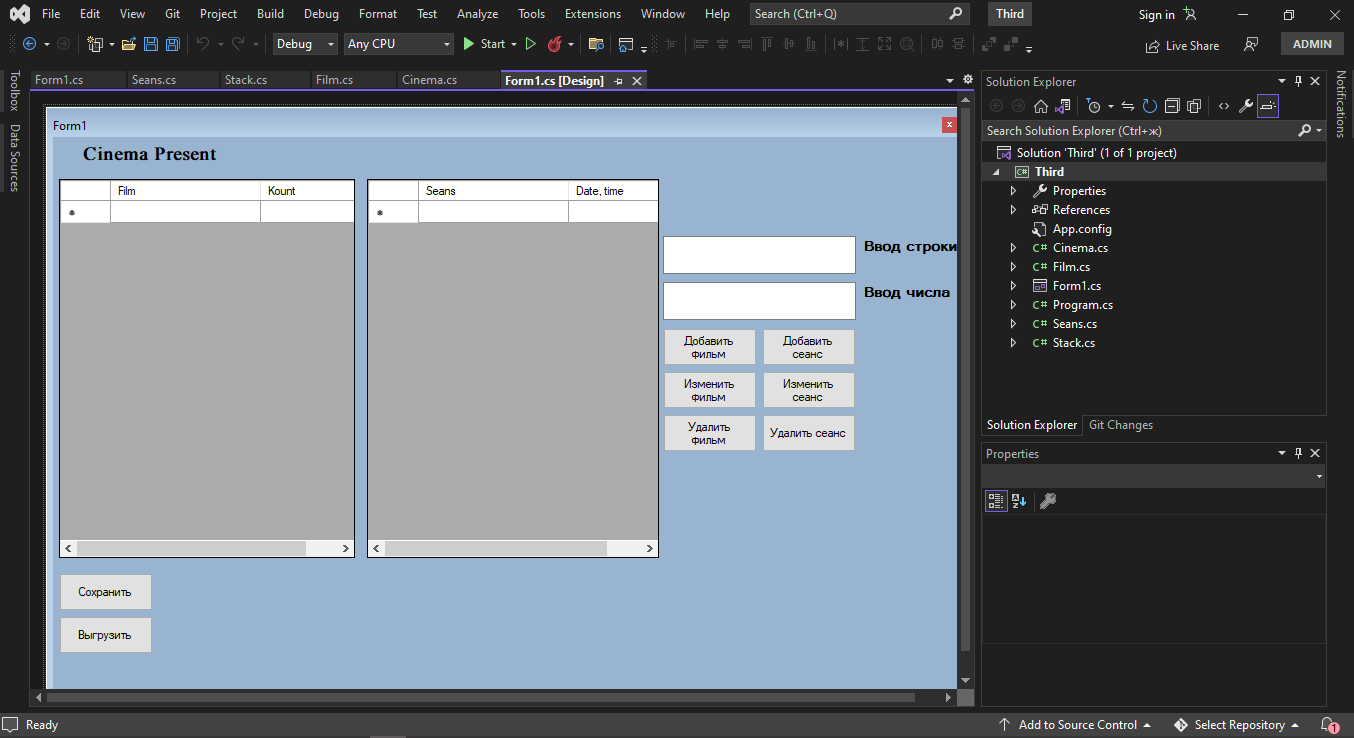


Рис Visual Studio

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.

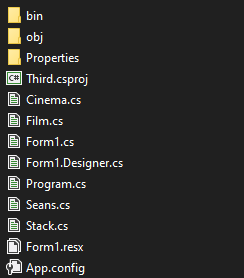


Рис. 10 Содержимое папки проекта.

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

**using** System**;**

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

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

**namespace** Third

**{**

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

**{**

Cinema news **=** **new** Cinema**();**

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

**{**

InitializeComponent**();**

**}**

**private** void button4\_Click**(object** sender**,** EventArgs e**)**

**{**

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

Film vspom **=** news**.**FindFilm**(**text**);**

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

**{**

vspom**.**delSeans**();**

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

**}**

**else** **{** MessageBox**.**Show**(**"Выберите фильм, сеанс которого хотите удалить"**);** **}**

**}**

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

**{**

news**.**addFilm**(**textBox**.**Text**,** Convert**.**ToInt32**(**textBox1**.**Text**));**

dataGridView1**.**Rows**.**Add**(**textBox**.**Text**,** textBox1**.**Text**);**

textBox**.**Clear**();**

textBox1**.**Clear**();**

**}**

**private** void listBox2\_SelectedIndexChanged**(object** sender**,** EventArgs e**){}**

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

**{**

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

news**.**getSeansToEdit**(**textBox**.**Text**,** Convert**.**ToInt32**(**textBox1**.**Text**),** text**);**

dataGridView2**.**Rows**.**Add**(**textBox**.**Text**,** textBox1**.**Text**);**

textBox**.**Clear**();**

textBox1**.**Clear**();**

**}**

**private** void dataGridView2\_CellContentClick**(object** sender**,** DataGridViewCellEventArgs e**){}**

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

**{**

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

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

Film vspom **=** news**.**FindFilm**(**text**);**

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

**{**

Stack pTemp **=** vspom**.**getSPEl**;**

int chet **=** news**.**FindFilm**(**text**).**Count**;**

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

**{**

dataGridView2**.**Rows**.**Add**(**pTemp**.**\_aDate**.**getDate**(),** Convert**.**ToString**(**pTemp**.**\_aDate**.**getTime**()));**

pTemp **=** pTemp**.**getPred**();**

**}**

**}**

**}**

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

**{**

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

int index **=** dataGridView1**.**CurrentRow**.**Index**;**

news**.**FindFilm**(**text**).**Name **=** textBox**.**Text**;**

news**.**FindFilm**(**textBox**.**Text**).**Durat **=** Convert**.**ToInt32**(**textBox1**.**Text**);**

dataGridView1**.**Rows**[**index**].**Cells**[**0**].**Value **=** textBox**.**Text**;**

dataGridView1**.**Rows**[**index**].**Cells**[**1**].**Value **=** textBox1**.**Text**;**

**}**

**private** void button2\_Click**(object** sender**,** EventArgs e**)**

**{**

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

int index **=** dataGridView2**.**CurrentRow**.**Index**;**

news**.**FindFilm**(**text**).**FindSeans**(**dataGridView2**.**CurrentCell**.**Value**.**ToString**()).**setDate**(**textBox**.**Text**);**

news**.**FindFilm**(**text**).**FindSeans**(**textBox**.**Text**).**setTime**(**Convert**.**ToInt32**(**textBox1**.**Text**));**

dataGridView2**.**Rows**[**index**].**Cells**[**0**].**Value **=** textBox**.**Text**;**

dataGridView2**.**Rows**[**index**].**Cells**[**1**].**Value **=** textBox1**.**Text**;**

**}**

**private** void button1\_Click**(object** sender**,** EventArgs e**)**

**{**

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

**if** **(**news**.**getFirst**().**Name**.**Equals**(**text**))**

**{**

dataGridView1**.**Rows**.**RemoveAt**(**dataGridView1**.**CurrentRow**.**Index**);**

news**.**delFilm**();**

**}**

**else** **{** MessageBox**.**Show**(**"Выберите первый в списке фильм для удаления"**);** **}**

**}**

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

**{**

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

int Count**,** LCount**,** gName**;**

**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 **<=** news**.**COUNT**;** i**++)**

**{**

news**.**delFilm**();**

**}**

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

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

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

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

**{**

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

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

news**.**addFilm**(**nameLH**,** seats**);**

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

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

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

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

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

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

news**.**FindFilm**(**name**).**addSeans**(**inf**,**gName**);**

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

i**++;**

**else**

**break;**

**}**

hall**++;**

**}**

**}**

**else**

**{**

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

**}**

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

**}**

**}**

**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 **+=** news**.**getinfoName**()** **+** "\n"**;**

**for(**int i **=** 0**;** i **<** news**.**COUNT**;** i**++)**

**{**

fileText **+=** news**.**getinfo**();**

**}**

//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**();**

**}**

**}**

**}**

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** Third

**{**

class Film

**{**

**private** Stack firstSeans**;**//первый сеанс

**private** string nameFilm**;**//название фильма

**private** int durationFilm**;**//длина фильма

**private** Stack SP**;**

**private** int count**;**

**public** Film**(**string \_nameFilm**,** int \_durationFilm**)**

**{**

**this.**nameFilm **=** \_nameFilm**;**

**this.**durationFilm **=** \_durationFilm**;**

SP **=** **null;**

**}**

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

**public** string Name **{** **get** **{** **return** nameFilm**;** **}** **set** **{** **this.**nameFilm **=** **value;** **}** **}**

**public** int Durat **{** **get** **{** **return** durationFilm**;** **}** **set** **{** **this.**durationFilm **=** **value;** **}** **}**

**public** Stack getSPEl **{** **get{** **return** SP**;** **}** **}**

**public** void addSeans**(**string \_date**,** int \_time**)**

**{**

Seans newSea **=** **new** Seans**(**\_date**,**\_time**);**

Stack pTemp **=** **new** Stack**(**newSea**,**SP**);**

**if** **(**SP **==** **null)** **{** firstSeans **=** pTemp**;}**

SP **=** pTemp**;**

count**++;**

**}**

**public** string getNameFilm**()** **{** **return** nameFilm**;** **}**

**public** bool delSeans**()**

**{**

**if** **(**SP **!=** **null)**

**{**

SP **=** SP**.**getPred**();**

count**--;**

**return** **true;**

**}**

**else** **return** **false;**

**}**

**public** int getSumStack**()**

**{**

**return** durationFilm**\***count**;**

**}**

**public** String getInfoStack**()**

**{**

String info **=** Convert**.**ToString**(**Count**);**

Stack pTemp **=** SP**;**

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

**{**

info **+=** "\n" **+** pTemp**.**\_aDate**.**getDate**();**

info **+=** "\n" **+** pTemp**.**\_aDate**.**getTime**();**

pTemp **=** pTemp**.**getPred**();**

**}**

**return** info**;**

**}**

**public** Seans FindSeans**(**string \_nameSeans**)**

**{**

Stack vspom = SP;

while(vspom != null)

{

if(vspom.\_aDate.getDate().Equals(\_nameSeans))

{

return vspom.\_aDate;

}

else { vspom = vspom.getPred(); }

}

return null;

}

}

}

**using** System**;**

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

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

**namespace** Third

**{**

class Seans

**{**

**private** string date**;**//дата показа фильма

**private** int time**;**//время показа фильма

**public** Seans**(**string \_date**,** int \_time**)**

**{**

**this.**date **=** \_date**;**

**this.**time **=** \_time**;**

**}**

**public** string getDate**()**

**{**

**return** date**;**

**}**

**public** void setDate**(**string \_date**)**

**{**

**this.**date **=** \_date**;**

**}**

**public** int getTime**()**

**{**

**return** time**;**

**}**

**public** void setTime**(**int \_time**)**

**{**

**this.**time **=** \_time**;**

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** Third

**{**

class Stack

**{**

**public** Seans \_aDate**;**

**private** Stack pred**;**

**public** Stack**(**Seans \_date**,** Stack \_SP**)**

**{**

\_aDate **=** \_date**;**

pred **=** \_SP**;**

**}**

**public** Stack getPred**()**

**{**

**return** pred**;**

**}**

**}**

**}**

**using** System**;**

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

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

**namespace** Third

**{**

class Cinema

**{**

**private** Film**[]** Queue**;**//реализация очереди на основе массива

**private** static int size **=** 10**;**

**private** int countFilm **=** 0**;**

**private** int first**;**//ссылка на первый элемент в очереди

**private** int last**;**//ссылка на последний элемент в очереди

**public** Cinema**()**

**{**

first **=** 0**;**

last **=** 0**;**

Queue **=** **new** Film**[**size**];**

**}**

**public** string addFilm**(**string \_nameFilm**,** int \_durationFilm**)**

**{**

**if** **(**size **>=** countFilm**)**

**{**

**if** **(**last **>** size **&&** first **>** 0**)** **{**last **=** 0**;}**

Film newFilm **=** **new** Film**(**\_nameFilm**,** \_durationFilm**);**

Queue**[**last**]** **=** newFilm**;**

last**++;**

countFilm**++;**

**return** \_nameFilm**;**

**}**

**return** ""**;**

**}**

**public** bool delFilm**()**

**{**

**if** **(** countFilm **!=** 0**)**

**{**

**if** **(**first **>** size**)** **{** first **=** 0**;}**

Queue**[**first**]** **=** **null;**

countFilm**--;**

first**++;**

**return** **true;**

**}**

**return** **false;**

**}**

**public** int COUNT **{** **get** **{** **return** countFilm**;** **}** **}**

**public** string getinfo**()**

**{**

string inform **=** ""**;**

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

**{**

inform **+=** "\n" **+** Queue**[**i**].**getInfoStack**();**

**}**

**return** inform**;**

**}**

**public** string getinfoName**()**

**{**

string inform **=** Convert**.**ToString**(**COUNT**);**

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

**{**

inform **+=** "\n" **+** Queue**[**i**].**Name**;**

inform **+=** "\n" **+** Queue**[**i**].**Durat**;**

**}**

**return** inform**;**

**}**

**public** Film getFirst**()** **{** **return** Queue**[**first**];** **}**

**public** Film FindFilm**(**string \_nameFilm**)**

**{**

Film pTemp**;**

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

**{**

pTemp **=** Queue**[**i**];**

**if** **(**pTemp **!=** **null** **&&** pTemp**.**getNameFilm**().**Equals**(**\_nameFilm**))**

**{**

**return** pTemp**;**

**}**

**}**

**return** **null;**

**}**

**public** bool getSeansToEdit**(**string \_date**,** int \_time**,** String \_nameFilm**)**

**{**

int i **=** 0**;**

Film pTemp **=** Queue**[**i**];**

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

**{**

**if** **(**pTemp **!=** **null** **&&** pTemp**.**getNameFilm**().**Equals**(**\_nameFilm**))**

**{**

pTemp**.**addSeans**(**\_date**,** \_time**);**

**return** **true;**

**}**

**else**

**{**

i**++;**

pTemp **=** Queue**[**i**];**

**}**

**}**

**return** **false;**

**}**

**}**

**}**