**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«ЮЖНО-РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (НПИ) имени М.И. Платова»**

**ФАКУЛЬТЕТ** **КАФЕДРА** **НАПРАВЛЕНИЕ**

**ОТЧЕТ**

**о \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_ практике**

**вид практики**

**на \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**наименование предприятия, организации**

**студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**курса, группы**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**фамилия, имя, отчество**

**Руководитель практики Оценка \_\_\_\_ \_\_\_\_\_\_\_\_\_ М. П.**

**от предприятия**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**

**должность, звание, степень подпись, дата**

**Руководитель практики Оценка \_\_\_\_ \_\_\_\_\_\_\_\_\_**

**от ЮРГПУ(НПИ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**должность, звание, степень**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**фамилия, имя, отчество подпись, дата**

**Новочеркасск 202 г.**

Содержание

[ВВЕДЕНИЕ 3](#_Toc172194846)

[1 ЦЕЛИ ПРОЕКТА И ЕГО ЗАДАЧИ 5](#_Toc172194847)

[2 КРАТКОЕ ОПИСАНИЕ АРХИТЕКТУР И ТЕХНОЛОГИЙ 6](#_Toc172194848)

[3 ДЕМОНСТРАЦИЯ ОСНОВНЫХ ФУНКЦИЙ ПРИЛОЖЕНИЯ 16](#_Toc172194849)

[3.1 ОБЗОР ФУНКЦИЙ ПРОГРАММЫ 16](#_Toc172194850)

[3.2 ДЕМОНСТРАЦИЯ ОСНОВНЫХ ФУНКЦИЙ ПРИЛОЖЕНИЯ 21](#_Toc172194851)

[ЗАКЛЮЧЕНИЕ И ПЛАНЫ ПО ДАЛЬНЕЙШЕМУ РАЗВИТИЮ ПРОЕКТА 23](#_Toc172194852)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 24](#_Toc172194853)

[ЛИСТИНГ КОДА 26](#_Toc172194854)

# ВВЕДЕНИЕ

Учебная практика проводилась в срок с 09.07.2024 г по 22.07.2024 г на кафедре Информационные и измерительные системы и технологии ЮРГПУ НПИ с ассистентом кафедры ИИСТ Филоновой Марьяной Андреевной. Практика направлена на формирование следующих компетенций:

1. ОПК-3-способен решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникативных технологий с учетом основных требований информационной безопасности.
2. ОПК-3.1-знать: принципы, методы и средства решения стандартных задач профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникационных технологий с учетом основных требований информационной безопасности.
3. ОПК-3-2-уметь: решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с применением информационно-коммуникативных технологий с учетом основных требований информационной безопасности.
4. ОПК-3-3 иметь навыки: подготовки обзоров, аннотаций, составления рефератов, научных докладов, публикаций, и библиографии по научно-исследовательской работе с учетом требований информационной безопасности.
5. ОПК-5-способен инсталлировать программное и аппаратное обеспечение для информационных и автоматизированных систем.
6. ОПК-5-1-знать: основы системного администрирования, администрирование СУБД, современные стандарты информационного взаимодействия систем.
7. ОПК-5-2-уметь: выполнять параметрическую настройку информационных и автоматизированных систем
8. ОПК-5-3-иметь навыки: инсталляции программного и аппаратного обеспечения информационных и автоматизированных систем.

# 1 ЦЕЛИ ПРОЕКТА И ЕГО ЗАДАЧИ

Тема проекта «Разработка приложения на C# для управления автопарком, используя принципы инкапсуляции и наследования»

К целям проекта можно отнести следующие пункты:

1. создание удобного и интуитивно понятного приложения на C# для управления списком;
2. предоставление пользователям возможности изменять значения, подстраивать приложение под себя.

Задачи проекты представляют собой следующие пункты:

1. разработать интерфейс приложения, который будет понятен и

удобен для пользователей;

1. реализовать функционал добавления, удаления и редактирования

списка;

1. провести тестирование приложения на соответствие функциональным

требованиям и исправить выявленные ошибки.

# 

# 2 КРАТКОЕ ОПИСАНИЕ АРХИТЕКТУР И ТЕХНОЛОГИЙ

Код следует принципам объектно-ориентированного программирования (ООП) и использует наследование для создания иерархии классов, представляющих различные типы транспортных средств. Класс Vehicle является базовым классом, который определяет общие свойства и методы для всех транспортных средств, а класс Car  наследуются от Vehicle и добавляет дополнительные свойства и методы, специфичные для каждого типа транспортного средства.

Приложение было создано с помощью Windows Forms.

Windows Forms (WinForms) — это платформа пользовательского интерфейса для создания классических приложений Windows. Это оболочка .NET поверх библиотек пользовательского интерфейса Windows, таких как User32 и GDI+. Он также предлагает элементы управления и другие функции, уникальные для Windows Forms.

Windows Forms также предоставляет один из самых продуктивных способов создания классических приложений на основе визуального конструктора, предоставляемого в Visual Studio. Он позволяет перетаскивать визуальные элементы управления и другие подобные функции, которые упрощают создание настольных приложений.

WinForms — это технология, которая была впервые представлена как часть .NET Framework 1.0 13 февраля 2002 года. Его основной целью было и остается быть средство быстрого приложения для приложений на базе Windows, и это принципиальное мнение не изменилось за эти годы. В то время WinForms удовлетворяла запросы разработчиков на

1. Фреймворк для стабильной, монолитной линейки бизнес-приложений,

даже с чрезвычайно сложными и запутанными рабочими процессами,

специфичными для предметной области

1. Возможность легко предоставлять многофункциональные и

совместимые с возможностями пользовательские интерфейсы

1. Безопасный и — по сравнению с первыми 3 версиями .NET Framework

все более производительный способ взаимодействия через границы процессов через различные службы коммуникации Windows или доступа к локальным базам данных через поставщиков ADO.NET.

1. Очень простой в использовании, визуальный дизайнер по принципу

«что видишь, то и получаешь», который не требует большого времени на разработку, и в первую очередь ориентирован на поддержку стратегий перетаскивания, основанных на разрешении 96 DPI.

1. Гибкая модель расширения конструктора на основе отражения .NET,

использующая компонентную модель .NET.

1. Визуальные элементы управления и компоненты, которые

предоставляют

собственные функциональные возможности во время разработки с помощью конструкторов элементов управления

Со временем, а также с растущей потребностью в решении рабочих сценариев с несколькими мониторами, мониторами с высоким разрешением, значительно более мощным оборудованием и многим другим, WinForms продолжает модернизироваться.

Приложения Windows Forms создаются с использованием языка программирования C# и платформы .NET. Они создаются путем перетаскивания в форму элементов управления, таких как кнопки, текстовые поля, метки и другие элементы пользовательского интерфейса.

1. Платформа Windows Forms предоставляет широкий набор элементов управления, которые разработчики могут использовать для создания приложений. Эти элементы управления предназначены для обеспечения согласованного и знакомого пользовательского интерфейса для пользователей Windows. Разработчики могут настраивать внешний вид и поведение этих элементов управления, устанавливая различные свойства и обрабатывая события.
2. Для создания приложения Windows Forms на C# можно использовать Microsoft Visual Studio, которая представляет собой интегрированную среду разработки (IDE), предоставляющую визуальный конструктор для создания и компоновки элементов пользовательского интерфейса. Визуальный конструктор — это интерфейс перетаскивания для создания пользовательского интерфейса, и вы можете легко настроить свойства каждого элемента управления с помощью удобного интерфейса.
3. В дополнение к визуальному конструктору Visual Studio также предоставляет редактор кода, который позволяет разработчикам писать код C# для логики приложения. Разработчики могут обрабатывать события и выполнять такие задачи, как проверка данных, обработка данных и реализация бизнес-логики.
4. Приложения Windows Forms универсальны и могут использоваться для создания различных типов приложений, таких как приложения для ввода данных, управления и отчетности, а также игр и мультимедийных приложений.

Кроме того, существует эволюция Windows: когда в новых версиях Windows появляются новые или изменяются существующие API или технологии, WinForms должна идти в ногу со временем и соответствующим образом корректировать свои API.

И именно это по-прежнему является основной мотивацией для модернизации и инноваций, а также планкой для достижения потенциальных областей инноваций, которые нам либо необходимы, либо мы хотим рассмотреть:

1. Области, в которых, например, по соображениям безопасности команде

Windows пришлось вывести зависимую область из строя, и мы видим экстремальное снижение производительности в приложениях WinForms, работающих под новым пакетом обновления или новой версией Windows

1. Новые функции для соответствия обновленным отраслевым стандартам

доступности.

1. Сценарии с высоким разрешением и для каждого монитора V2.
2. Использование измененных или расширенных функций управления

Win32, чтобы элементы управления в WinForms работали так, как этого хочет команда Windows.

1. Решение проблем с производительностью и безопасностью
2. Представляем способы поддержки асинхронных вызовов на

интерактивном уровне, чтобы приложения могли выбирать пути миграции через проекцию Windows API/Windows Desktop Bridge, включить сценарии для асинхронных вызовов WebAPI, SignalR, Azure Function и т. д., чтобы серверные части WinForms можно было модернизировать и даже перенести в облако.

C# — это объектно-ориентированный язык программирования. Четыре основных принципа объектно-ориентированного программирования следующие.

1. Абстракция. Моделирование требуемых атрибутов и взаимодействий

сущностей в виде классов для определения абстрактного представления системы.

1. Инкапсуляция. Скрытие внутреннего состояния и функций объекта и

предоставление доступа только через открытый набор функций.

1. Наследование. Возможность создания новых абстракций на основе

существующих.

1. Полиморфизм. Возможность реализации наследуемых свойств или

методов отличающимися способами в рамках множества абстракций.

При работе с инкапсуляцией всегда следует принимать во внимание то, какие аспекты типа видимы различным частям приложения. В частности, типы (классы, интерфейсы, структуры, перечисления и делегаты), а также их члены (свойства, методы, конструкторы и поля) определяются с использованием определенного ключевого слова, управляющего "видимостью" элемента другим частям приложения. Хотя в С# определены многочисленные ключевые слова для управления доступом, их значение может отличаться в зависимости от места применения (к типу или члену). В табл. 1 описаны роли и применение модификаторов доступа.

Таблица 1

Модификаторы доступа С#

|  |  |
| --- | --- |
| Модификатор доступа (к чему применяется) | Назначение |
| public (типы или члены типов) | Общедоступные (public) элементы не имеют ограничений доступа. Общедоступный член может быть доступен как из объекта, так и из любого производного класса. Общедоступный тип может быть доступен из других внешних сборок |
| private (члены типов или вложенные типы) | Приватные (private) элементы могут быть доступны только в классе (или структуре), в котором они определены |
| protected (члены типов или вложенные типы) | Защищенные (protected) элементы могут использоваться классом, который определил их, и любым дочерним классом. Однако защищенные элементы не доступны внешнему миру через операцию точки (.) |

|  |  |
| --- | --- |
| internal (типы или члены типов) | Внутренние (internal) элементы доступны только в пределах текущей сборки. Таким образом, если в библиотеке классов .NET определен набор внутренних типов, то другие сборки не смогут ими пользоваться |
| protected internal (члены типов или вложенные типы) | Когда ключевые слова protected и internal комбинируются в объявлении элемента, такой элемент доступен внутри определяющей его сборки, определяющего класса и всех его наследников |

Концепция инкапсуляции вращается вокруг принципа, гласящего, что внутренние данные объекта не должны быть напрямую доступны через экземпляр объекта. Вместо этого, если вызывающий код желает изменить состояние объекта, то должен делать это через методы доступа (accessor, или метод get) и изменения (mutator, или метод set). В С# инкапсуляция обеспечивается на синтаксическом уровне с использованием ключевых слов public, private, internal и protected.

Проблема с общедоступными данными состоит в том, что сами по себе эти данные не имеют возможности распознать, является ли присваиваемое значение допустимым в рамках существующих бизнес-правил системы.

Если в текущей системе установлено правило, гласящее, что книга должна иметь от 1 до 1000 страниц, его придется обеспечить программно. По этой причине общедоступным полям обычно нет места в определении класса производственного уровня.

Инкапсуляция предоставляет способ предохранения целостности данных о состоянии объекта. Вместо определения общедоступных полей (которые легко приводят к повреждению данных), необходимо выработать привычку определения приватных данных, управление которыми осуществляется опосредованно, с применением одной из двух техник:

* 1. определение пары методов доступа и изменения;
  2. определение свойства .NET.

Свойство — способ доступа к внутреннему состоянию объекта, имитирующий переменную некоторого типа. Обращение к свойству объекта реализовано через вызов функции. При попытке задать значение данного свойства вызывается один метод, а при попытке получить значение данного свойства — другой. Свойства в C# — поля с логическим блоком, в котором есть ключевые слова get и set.

Пример класса со свойством:

public class Date{private int month = 7;public int Month{get{return month;}set{if ((value > 0) && (value < 13)){month = value;}}}}

Наследование - это фундаментальная концепция объектно-ориентированного программирования, которая позволяет нам определять новый класс на основе существующего класса. Новый класс наследует свойства и методы существующего класса, а также может добавлять свои собственные свойства и методы. Наследование способствует повторному использованию кода, упрощает обслуживание кода и улучшает организацию кода.

Главное преимущество наследования заключается в следующем: как только будет создан базовый класс, в котором определены общие для множества объектов атрибуты, он может быть использован для создания любого числа более конкретных производных классов. А в каждом производном классе может быть точно выстроена своя собственная классификация.

Члены класса зачастую объявляются закрытыми, чтобы исключить их несанкционированное или незаконное использование. И наследование класса не отменяет ограничения, накладываемые на доступ к закрытым членам класса. Поэтому если в производный класс и входят все члены его базового класса, в нем все равно оказываются недоступными те члены базового класса, которые являются закрытыми.

Для преодоления ограничения на доступ к частным членам базового класса из производного класса в С# предусмотрены разные способы:

Первый из них состоит в применении открытых свойств для доступа к закрытым данным.

Свойство позволяет управлять доступом к переменной экземпляра. Например, с помощью свойства можно ввести ограничения на доступ к значению переменной или же сделать ее доступной только для чтения. Так, если сделать свойство открытым, но объявить его базовую переменную закрытой, то этим свойством можно будет воспользоваться в производном классе, но нельзя будет получить непосредственный доступ к его базовой закрытой переменной.

Второй же состоит в использовании защищенных (protected) членов класса.

Защищенный член является открытым в пределах иерархии классов, но закрытым за пределами этой иерархии. Защищенный член создается с помощью модификатора доступа protected. Если член класса объявляется как protected, он становится закрытым, но за исключением одного случая, когда защищенный член наследуется. В этом случае защищенный член базового класса становится защищенным членом производного класса, а значит, доступным для производного класса. Таким образом, используя модификатор доступа protected, можно создать члены класса, являющиеся закрытыми для своего класса, но все же наследуемыми и доступными для производного класса.

Модификатор доступа protected следует применять в том случае, если требуется создать член класса, доступный для всей иерархии классов, но для остального кода он должен быть закрытым. А для управления доступом к значению члена класса лучше воспользоваться свойством.

В иерархии классов допускается, чтобы у базовых и производных классов были свои собственные конструкторы. Когда конструктор определен только в производном классе, то конструируется объект производного класса, а базовая часть объекта автоматически конструируется его конструктором, используемым по умолчанию.

Когда конструкторы определяются как в базовом, так и в производном классе, процесс построения объекта несколько усложняется, поскольку должны выполняться конструкторы обоих классов. В данном случае приходится обращаться к еще одному ключевому слову языка С#: base, которое находит двоякое применение: во-первых, для вызова конструктора базового класса; и во-вторых, для доступа к члену базового класса, скрывающегося за членом производного класса.

В C # существует 4 типа наследования:

1. Одиночное наследование: производный класс, который наследуется

только от одного базового класса.

1. Многоуровневое наследование: производный класс, который наследуется

от базового класса, а сам производный класс становится базовым классом для другого производного класса.

1. Иерархическое наследование: базовый класс, который служит

родительским классом для двух или более производных классов.

1. Множественное наследование: производный класс, который наследуется

от двух или более базовых классов.

Преимущества наследования:

1. Возможность повторного использования кода: наследование позволяет

нам повторно использовать существующий код путем наследования свойств и методов из существующего класса.

1. Обслуживание кода: Наследование упрощает обслуживание кода,

позволяя нам изменять базовый класс и автоматически отражать изменения в производных классах.

1. Организация кода: Наследование улучшает организацию кода, группируя

связанные классы в иерархическую структуру.

Недостатки наследования:

1. Тесная связь: наследование создает тесную связь между базовым классом

и производным классом, что может усложнить поддержку кода.

1. Сложность: Наследование может увеличить сложность кода за счет

введения дополнительных уровней абстракции.

1. Хрупкость: наследование может сделать код более хрупким, создавая зависимости между базовым классом и производным классом.

Преимущества использования этой архитектуры и технологий:  
  Модульность: Разделение кода на классы делает код более модульным и легко поддерживаемым.

Повторное использование кода: Наследование позволяет нам повторно использовать общий код, что уменьшает дублирование кода и повышает его поддерживаемость.  
  Расширяемость: Иерархия классов позволяет нам легко добавлять новые типы транспортных средств, наследуя от базового класса `Vehicle`.  
 Удобство для пользователя: Использование DataGridView упрощает отображение данных об автомобилях в удобном для чтения табличном формате.

# 3 ДЕМОНСТРАЦИЯ ОСНОВНЫХ ФУНКЦИЙ ПРИЛОЖЕНИЯ

3.1 ОБЗОР ФУНКЦИЙ ПРОГРАММЫ

Проект представляет собой приложение, которое состоит из двух частей: первая часть - это визуальное составляющая приложения, созданная с помощью .NET Framework. Приложение созданное с помощью него представляет собой отдельное окно для дальнейшей работы в нём (см.рисунок 1).

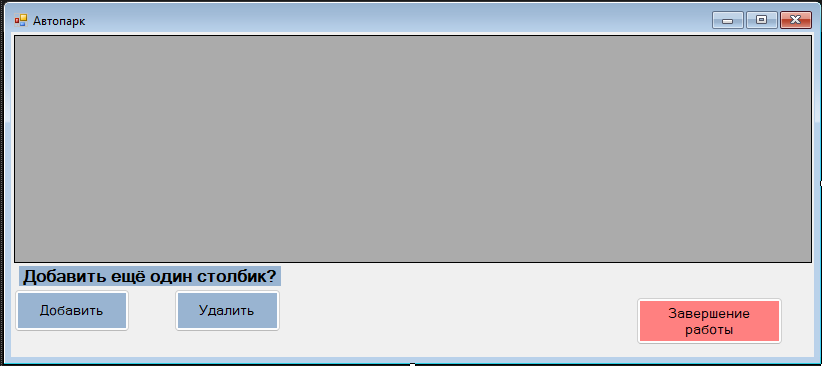


Рисунок 1 – Внешний вид приложения

К его основным функциям относятся следующие:

1. Form — это представление любого окна, отображаемого в приложении.

Класс Form можно использовать для создания стандартных, инструментальных, безграничных и плавающих окон, а также для создания модальных окон, таких как диалоговое окно.

С помощью свойств, доступных в классе Form, можно определить внешний вид, размер, цвет и функции управления окном создаваемого окна или диалогового окна.

1. DataGridView — это элемент управления, который позволяет вывести

таблицу с данными в программе. Элемент управления DataGridView предоставляет настраиваемую таблицу для отображения данных. Класс DataGridView позволяет настраивать ячейки, строки, столбцы и границы с помощью таких свойств, как DefaultCellStyle, ColumnHeadersDefaultCellStyle, CellBorderStyleи GridColor.

1. Button — это обычная кнопка, клик по которой вызывает срабатывание

события. Click по которому приложение может выполнять какие-либо действия. RepeatButton позволяет повторять событие. Click с определенным интервалом пока зажата левая кнопка мыши на кнопке.

1. Компонент Label – это простая текстовая метка, предназначенная для

вывода текста. Свойство Caption типа AnsiString позволяет принимать текст и выводить его в приложении. Задавать значение свойству можно как через «Инспектор объектов», так и программно.

Вторая часть проекта-это программа написанная на языке C#. Она служит для того, чтобы все элементы приложеня, имели логику и могли функционировать.

В ней будут прописаны различные методы:

1. Метод которой вызывается при нажатии кнопки «Добавить».

Он будет служить для того, чтобы при нажатии на эту кнопку в таблице появлялись новые столбцы, в которые пользователь сможет записать свою информацию(см.рисунок 2 ). Реализация метода (см.рисунок 3 )

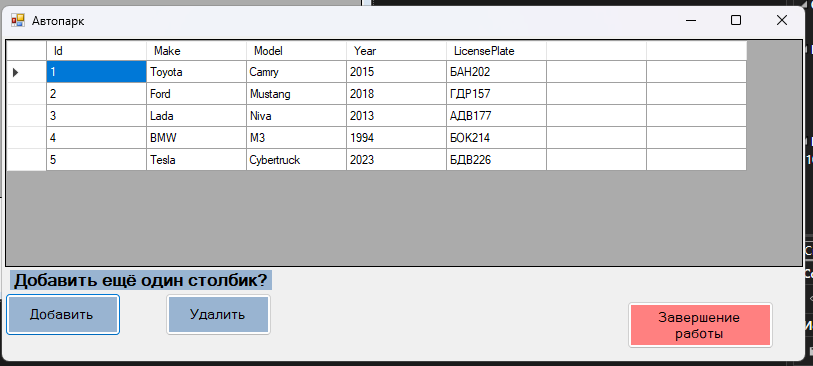


Рисунок 2 – Реализация кнопки «Добавить»

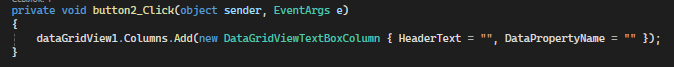


Рисунок 3 – Код для кнопки «Добавить»

1. Метод, который вызывается при нажатии кнопки «Удалить».

Он служит для того, чтобы при нажатии на эту кнопку удалялся ненужный столбец(см.рисунок 4 ). Реализация метода (см.рисунок 5 )

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 4 – Реализация кнопки «Удалить»

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 5 – Код для кнопки «Удалить»

1. Метод, который вызывается при нажатии кнопки «Завершение работы». Он служит для того, чтобы при нажатии на эту кнопку приложение завершало свою работу(см.рисунок 6 ).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 6 – Код для кнопки «Завершение работы»

1. Элемент управления - dataGridView1. Служит для того, чтобы выводить таблицу на экран для дальнейших работ с ней (см.рисунок 7).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 – Информация для таблицы

1. Для того чтобы код работал и выводил информацию надо указать переменные. Для этого создаём основной класс «Vehicle» (см.рисунок 8).

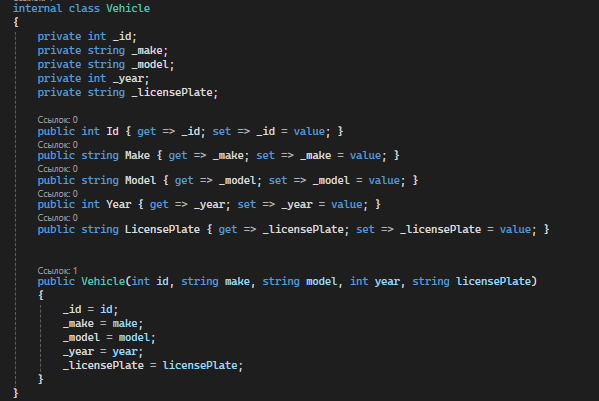


Рисунок 8 – Основной класс «Vehicle»

1. Для того чтобы можно было добавлять различную дополнительную

информацию создаём ещё один класс под названием «Car» (см.рисунок 9).

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Шрифт

Автоматически созданное описание

Рисунок 9 – Дополнительный класс «Car»

3.2 ДЕМОНСТРАЦИЯ ОСНОВНЫХ ФУНКЦИЙ ПРИЛОЖЕНИЯ

1. Замена определённой информации. Замена производится в любое соответствующие поле. (см.рисунок 10)

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 10 – Замена определённой информации

1. Добавление информации о машинах в свободные столбцы. (см.рисунок 11)

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Рисунок 11 – Добавление информации

1. Выделение одной или нескольких строк (см.рисунок 12).

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 12 – Выделение строк

1. Использование кнопок «Завершение работы», «Добавить», «Удалить» рассматривались ранее.

# ЗАКЛЮЧЕНИЕ И ПЛАНЫ ПО ДАЛЬНЕЙШЕМУ РАЗВИТИЮ ПРОЕКТА

В ходе разработки приложения были успешно реализованы основные функции. Пользователь может удобно создавать, редактировать и удалять информацию. Интерфейс приложения интуитивно понятен и удобен для использования.

В дальнейшем проект можно развить, добавив туда такие функции как

- Сохранения изменённой таблицы.

- Добавление возможности фильтрации по различным критериям (по марке, номеру, модели и т. д.) и сортировки по выбранным параметрам.

- Возможность добавления автомобилей из другого списка.

- Добавление возможности создания резервных копий списка автомобилей и восстановления данных в случае их потери или повреждения.

- Сохранение данных в облаке для доступа с других устройств и возможность использования данных из других приложений.

Ссылка на репозиторий: https://github.com/Alex-Kraken777/praktika.git

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Интернет ресурсы:

1. Winforms/README.md at main dotnet/winforms GitHub [Электронный ресурс]. Режим доступа: <https://github.com/dotnet/winforms/blob/main/README.md?ysclid=m0q0wj234u741044290> (дата обращения 9.07.2024)
2. Основы Windows Forms в C# .NET [Электронный ресурс]. Режим доступа: <https://esate.ru/uroki/OpenGL/uroki-OpenGL-c-sharp/osnovi-windows-forms/?ysclid=m0pulhk1bu567536316> (дата обращения 10.07.2024)
3. How to Create a C# Windows Forms Application [Электронный ресурс]. Режим доступа: <https://net-informations.com/csharp/gui/cs_forms.htm> (дата обращения 10.07.2024)
4. Принципы ООП в примерах для начинающих / Хабр [Электронный ресурс]. Режим доступа: <https://habr.com/ru/companies/otus/articles/764266/> (дата обращения 13.07.2024)
5. Объектно-ориентированное программирование (C#) [Электронный ресурс]. Режим доступа: https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/tutorials/oop (дата обращения 11.07.2024)
6. [Классы и объекты C#: инкапсуляция | Платформа .NET и C# от А до Я](https://csharp.webdelphi.ru/klassy-i-obekty-c-inkapsulyaciya/) [Электронный ресурс]. Режим доступа: <https://csharp.webdelphi.ru/klassy-i-obekty-c-inkapsulyaciya/> (дата обращения 12.07.2024)
7. Инкапсуляция в c# [Электронный ресурс]. Режим доступа: https://studfile.net/preview/16562554/page:5/ (дата обращения 12.07.2024)
8. Инкапсуляция в C#: 2 поясняющих примера [Электронный ресурс]. Режим доступа: <https://ci-sharp.ru/obuchenie/osnovy/inkapsulyacziya-v-c-2-poyasnyayushhih-primera/?ysclid=m0tmecp08j663642790> (дата обращения 12.07.2024)
9. C# и .NET | Наследование [Электронный ресурс]. Режим доступа: https://metanit.com/sharp/tutorial/3.7.php?ysclid=m0ptxou3d6448360155 (дата обращения 13.07.2024)
10. C # | Наследование - GeeksforGeeks [Электронный ресурс]. Режим доступа: https://www.geeksforgeeks.org/c-sharp-inheritance/ (дата обращения 13.07.2024)

# ЛИСТИНГ КОДА

Листинг 1.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace praktika2.\_0

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

// Создание объектов типа Car

Car car1 = new Car(1, "Toyota", "Camry", 2015, "БАН202", 9, "Sedan");

Car car2 = new Car(2, "Ford", "Mustang", 2018, "ГДР157", 6, "Coupe");

Car car3 = new Car(3, "Lada", "Niva", 2013, "АДВ177", 2, "Universal");

Car car4 = new Car(4, "BMW", "M3", 1994, "БОК214", 30, "Sedan" );

Car car5 = new Car(5, "Tesla", "Cybertruck", 2023, "БДВ226", 1, "Pickup");

// Добавление автомобилей в DataGridView

dataGridView1.DataSource = new List<Vehicle> { car1, car2, car3, car4, car5 };

}

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

{

}

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

{

Application.Exit();

}

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

{

dataGridView1.Columns.Add(new DataGridViewTextBoxColumn { HeaderText = "", DataPropertyName = "" });

}

private void Remove\_Click(object sender, EventArgs e)

{

dataGridView1.Columns.RemoveAt(dataGridView1.Columns.Count - 1);

}

}

Листинг 2.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Remoting.Lifetime;

using System.Text;

using System.Threading.Tasks;

namespace praktika2.\_0

{

internal class Vehicle

{

private int \_id;

private string \_make;

private string \_model;

private int \_year;

private string \_licensePlate;

public int Id { get => \_id; set => \_id = value; }

public string Make { get => \_make; set => \_make = value; }

public string Model { get => \_model; set => \_model = value; }

public int Year { get => \_year; set => \_year = value; }

public string LicensePlate { get => \_licensePlate; set => \_licensePlate = value; }

public Vehicle(int id, string make, string model, int year, string licensePlate)

{

\_id = id;

\_make = make;

\_model = model;

\_year = year;

\_licensePlate = licensePlate;

}

}

}

Листинг 3.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Remoting.Lifetime;

using System.Text;

using System.Threading.Tasks;

namespace praktika2.\_0

{

internal class Car : Vehicle

{

private int \_numSeats;

private string \_bodyType;

public int NumSeats { get => \_numSeats; set => \_numSeats = value; }

public string BodyType { get => \_bodyType; set => \_bodyType = value; }

public Car(int id, string make, string model, int year, string licensePlate, int numSeats, string bodyType)

: base(id, make, model, year, licensePlate)

{

\_numSeats = numSeats;

\_bodyType = bodyType;

}

}

}