Министерство образования Российской Федерации Пензенский государственный университет

Кафедра «Математическое обеспечение и применение ЭВМ»

**ОТЧЕТ**

по лабораторной работе №8

по курсу «ООП» на тему «Паттерн MVP»

Выполнили:

студенты группы 23ВП1

Рейимов М.

Принял:

к.т.н. Афонин А.Ю.

Пенза 2025

### Название

Паттерн MVP.

### Цель работы

Научиться использовать паттерн MVP.

### Лабораторное задание

Реализовать паттерн программирования MVP.

Предметная область – фирма грузоперевозок.

### Листинг

### Файл FormView.cs:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Security.Policy;

using System.Text;

using System.Threading.Tasks;

using System.Text.RegularExpressions;

using System.Windows.Forms;

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

using Lab1;

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

using System.Diagnostics;

using System.Xml.Linq;

namespace Lab2

{

public partial class FormView : Form, IView

{

public event Add AddClicked;

public event Remove RemoveClicked;

public event GetAll GetAllClicked;

public event SaveChanges SaveChangesClicked;

public string NameCompany => nameInput.Text;

public int Price => (int)priceInput.Value;

public float Time => (float)avgtimeInput.Value;

public string Number => phonenumberInput.Text;

public int Year => (int)yearInput.Value;

public float Mass => (float)massInput.Value;

public float Rating => (float)ratingInput.Value;

public FormView()

{

InitializeComponent();

var presenter = new Presenter(this);

}

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

{

try

{

AddClicked.Invoke(NameCompany, Price, Time, Number, Year, Mass, Rating);

objCount.Text = TransportCompany.countObj.ToString();

ShowAll();

}

catch (MyException ex)

{

MessageBox.Show(ex.Message, "Ошибка");

}

}

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

{

ShowAll();

TransportCompany.countObj--;

MessageBox.Show(RemoveClicked.Invoke());

}

private void save\_button\_Click(object sender, EventArgs e)

{

for (int i = 0; i < dataGridView1.RowCount; ++i)

{

string selectedStrategy = dataGridView1.Rows[i].Cells[8].Value.ToString();

string selectedMethod = dataGridView1.Rows[i].Cells[7].Value.ToString();

SaveChangesClicked.Invoke(i, selectedStrategy, selectedMethod);

}

ShowAll();

MessageBox.Show("Изменения успешно сохранены", "Сохранить");

}

private void ShowAll()

{

dataGridView1.Rows.Clear();

foreach (var company in GetAllClicked.Invoke().Reverse())

{

int rowIndex = dataGridView1.Rows.Add();

dataGridView1.Rows[rowIndex].Cells[0].Value = company.name;

dataGridView1.Rows[rowIndex].Cells[1].Value = company.pricePerKilometer.ToString();

dataGridView1.Rows[rowIndex].Cells[2].Value = company.averageDeliveryTime.ToString();

dataGridView1.Rows[rowIndex].Cells[3].Value = company.yearFounded.ToString();

dataGridView1.Rows[rowIndex].Cells[4].Value = company.transportedMass.ToString();

dataGridView1.Rows[rowIndex].Cells[5].Value = company.rating.ToString();

dataGridView1.Rows[rowIndex].Cells[6].Value = company.phoneNumber;

dataGridView1.Rows[rowIndex].Cells[7].Value = company.deliveryType.GetDeliveryType();

dataGridView1.Rows[rowIndex].Cells[8].Value = company.efficiencyStrategy.Description();

dataGridView1.Rows[rowIndex].Cells[9].Value = company.efficiency.ToString();

}

}

}

}

Файл ConsoleView.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

class ConsoleView : IView

{

public event Add AddClicked;

public event Remove RemoveClicked;

public event GetAll GetAllClicked;

public event SaveChanges SaveChangesClicked;

private Presenter controller;

public string NameCompany { get; private set; }

public int Price { get; private set; }

public float Time { get; private set; }

public string Number { get; private set; }

public int Year { get; private set; }

public float Mass { get; private set; }

public float Rating { get; private set; }

public void Run()

{

controller = new Presenter(this);

while (true)

{

Console.WriteLine("\nМеню:");

Console.WriteLine("1. Добавить компанию");

Console.WriteLine("2. Удалить последнюю компанию");

Console.WriteLine("3. Показать все компании");

Console.WriteLine("4. Изменить стратегию расчета эффективности и метод доставки");

Console.WriteLine("0. Выход");

Console.Write("Выбор: ");

string input = Console.ReadLine();

switch (input)

{

case "1":

try

{

ReadCompanyInput();

AddClicked.Invoke(NameCompany, Price, Time, Number, Year, Mass, Rating);

Console.WriteLine("Компания добавлена.");

break;

}

catch (MyException ex)

{

Console.WriteLine(ex.Message);

break;

}

case "2":

Console.WriteLine(controller.DeleteCompany());

break;

case "3":

var companies = GetAllClicked.Invoke();

PrintCompanies(companies);

break;

case "4":

Console.Write("Введите индекс компании (0 — первая): ");

int index = int.Parse(Console.ReadLine());

Console.Write("Выберите стратегию рассчета эффективности (По массе и времени / По возрасту и цене / Баланс): ");

string strategy = Console.ReadLine();

Console.Write("Выберите метод доставки (По земле / По воде / По воздуху): ");

string methodInput = Console.ReadLine();

string method = null;

switch (methodInput)

{

case "По земле":

method = new RoadDelivery().GetDeliveryType();

break;

case "По воде":

method = new SeaDelivery().GetDeliveryType();

break;

case "По воздуху":

method = new AirDelivery().GetDeliveryType();

break;

default:

Console.WriteLine("Неверный ввод");

break;

}

SaveChangesClicked.Invoke(index, strategy, method);

Console.WriteLine("Изменения сохранены.");

break;

case "0":

return;

default:

Console.WriteLine("Неверный ввод.");

break;

}

}

}

private void ReadCompanyInput()

{

Console.Write("Название компании: ");

NameCompany = Console.ReadLine();

if (string.IsNullOrWhiteSpace(NameCompany))

throw new MyException("Название компании не может быть пустым.");

Console.Write("Цена за километр: ");

if (!int.TryParse(Console.ReadLine(), out int price) || price < 0)

throw new MyException("Некоррект. цена.");

Price = price;

Console.Write("Ср. время доставки: ");

if (!float.TryParse(Console.ReadLine(), out float time) || time < 0)

throw new MyException("Некорректн. время");

Time = time;

Console.Write("Год основания: ");

if (!int.TryParse(Console.ReadLine(), out int year) || year < 2000 || year > 2025)

throw new MyException("Ограничения на год основания - с 2000 по 2025");

Year = year;

Console.Write("Масса перевезенного груза: ");

if (!int.TryParse(Console.ReadLine(), out int mass) || mass < 0)

throw new MyException("Некорректн. масса");

Mass = mass;

Console.Write("Номер компании:");

Number = Console.ReadLine();

if (string.IsNullOrWhiteSpace(Number) || Number.Length != 11 || !Number.All(char.IsDigit))

throw new MyException("Некорректн. номер телефона.");

Console.Write("Рейтинг:");

if (!float.TryParse(Console.ReadLine(), out float rating) || rating < 0 || rating > 10)

throw new MyException("Некрректн. рейтинг");

Rating = rating;

}

private void PrintCompanies(Stack<TransportCompany> companies)

{

if (companies == null || companies.Count == 0)

{

Console.WriteLine("\nСписок компаний пуст.");

return;

}

Console.WriteLine("\nСписок компаний:");

var list = companies.Reverse().ToList();

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

{

var c = list[i];

Console.WriteLine($"[{i}] {c.name} | Цена: {c.pricePerKilometer} | Ср.время доставки: {c.yearFounded} | Год основания: {c.yearFounded} | Масса перевезенного груза: {c.transportedMass} | Рейтинг: {c.rating} | Номер телефона: {c.phoneNumber} | Доставка: {c.GetDeliveryInfo()} | Стратегия эффективности: {c.efficiencyStrategy.Description()}| Эффективность:{c.efficiency}|");

}

}

}

}

Файл Model.cs:

using System;

using System.Collections.Generic;

using System.Drawing.Printing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

public interface IDeliveryType

{

string GetDeliveryType();

}

public class SeaDelivery : IDeliveryType

{

public string GetDeliveryType()

{

return "По воде";

}

}

public class RoadDelivery : IDeliveryType

{

public string GetDeliveryType()

{

return "По земле";

}

}

public class AirDelivery : IDeliveryType

{

public string GetDeliveryType()

{

return "По воздуху";

}

}

public interface IEfficiencyStrategy

{

float CalculateEfficiency(TransportCompany company);

string Description();

}

public class MassTimeEfficiency : IEfficiencyStrategy

{

public float CalculateEfficiency(TransportCompany company)

{

if (company.averageDeliveryTime == 0)

return 0;

return (float)Math.Round(company.transportedMass / company.averageDeliveryTime, 2);

}

public string Description()

{

return "По массе и времени";

}

}

public class AgePriceEfficiency : IEfficiencyStrategy

{

public float CalculateEfficiency(TransportCompany company)

{

int currentYear = DateTime.Now.Year;

int age = currentYear - company.yearFounded;

if (company.pricePerKilometer == 0)

return 0;

return (float)Math.Round(age / (float)company.pricePerKilometer, 2);

}

public string Description()

{

return "По возрасту и цене";

}

}

public class BalancedEfficiency : IEfficiencyStrategy

{

public float CalculateEfficiency(TransportCompany company)

{

float massEff = company.transportedMass / (company.averageDeliveryTime + 1);

float priceEff = 100f / (company.pricePerKilometer + 1);

return (float)Math.Round(0.6f \* massEff + 0.4f \* priceEff, 2);

}

public string Description()

{

return "Сбалансированная";

}

}

public abstract class TransportCompany

{

public string name { get; set; }

public int pricePerKilometer { get; set; }

public float averageDeliveryTime { get; set; }

public int yearFounded { get; set; }

public float transportedMass { get; set; }

public float rating { get; set; }

public string phoneNumber { get; set; }

public float efficiency { get; set; }

public IDeliveryType deliveryType { get; set; }

public IEfficiencyStrategy efficiencyStrategy { get; set; }

public static int countObj = 0;

public TransportCompany()

{

name = "";

pricePerKilometer = 0;

averageDeliveryTime = 0;

yearFounded = 2000;

transportedMass = 0;

rating = 0;

phoneNumber = "";

countObj++;

deliveryType = null;

efficiencyStrategy = null;

}

public TransportCompany(string name) : this()

{

this.name = name;

}

public TransportCompany(string name, int pricePerKilometer) : this(name)

{

this.pricePerKilometer = pricePerKilometer;

}

public TransportCompany(string name, int pricePerKilometer, float averageDeliveryTime,

int yearFounded, float transportedMass, float rating, string phoneNumber, IDeliveryType deliveryType) : this(name, pricePerKilometer)

{

this.averageDeliveryTime = averageDeliveryTime;

this.yearFounded = yearFounded;

this.transportedMass = transportedMass;

this.rating = rating;

this.phoneNumber = phoneNumber;

this.deliveryType = deliveryType;

this.efficiencyStrategy = new BalancedEfficiency();

}

public virtual string GetDeliveryInfo()

{

return deliveryType.GetDeliveryType();

}

public virtual void RecalculateEfficiency()

{

efficiency = efficiencyStrategy.CalculateEfficiency(this);

}

}

public class ExpressCompany : TransportCompany

{

public ExpressCompany(string name,

int pricePerKilometer,

float averageDeliveryTime,

int yearFounded,

float transportedMass,

float rating,

string phoneNumber,

IDeliveryType deliveryType) : base(name, pricePerKilometer, averageDeliveryTime, yearFounded, transportedMass, rating, phoneNumber, deliveryType) { }

public override string GetDeliveryInfo()

{

return base.GetDeliveryInfo();

}

}

public class StackTransportCompany

{

private Stack<TransportCompany> transportCompanies;

public delegate void AddToStack(TransportCompany company);

public delegate void RemoveFromStack();

public event AddToStack StackAdded;

public event RemoveFromStack StackRemoved;

public StackTransportCompany()

{

this.transportCompanies = new Stack<TransportCompany>();

}

public Stack<TransportCompany> GetTransportCompanies()

{

return transportCompanies;

}

public void AddCompany(TransportCompany company)

{

transportCompanies.Push(company);

StackAdded?.Invoke(company);

}

public void DeleteCompany()

{

if (transportCompanies.Count == 0)

throw new MyException("Стек пуст");

TransportCompany top = transportCompanies.Pop();

StackRemoved?.Invoke();

}

}

}

Файл Presenter.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

namespace Lab1

{

public class Presenter

{

private StackTransportCompany companies;

IView view;

public Presenter(IView view)

{

companies = new StackTransportCompany();

this.view = view;

this.view.AddClicked += CreateCompany;

this.view.SaveChangesClicked += SaveChanges;

this.view.RemoveClicked += DeleteCompany;

this.view.GetAllClicked += GetAll;

}

public void CreateCompany(string name, int pricePerKilometer, float avrgDeliveryTime, string phoneNumber,

int yearFounded, float transportedMass, float rating)

{

if (string.IsNullOrWhiteSpace(name))

throw new MyException("Фирма должна иметь название");

if (!Regex.IsMatch(phoneNumber.Trim(), @"^\d{11}$"))

throw new MyException("Номер должен состоять из 11 цифр");

TransportCompany firm = new ExpressCompany(

name,

pricePerKilometer,

avrgDeliveryTime,

yearFounded,

transportedMass,

rating,

phoneNumber,

new RoadDelivery()

);

firm.RecalculateEfficiency();

companies.AddCompany(firm);

}

public string DeleteCompany()

{

try

{

companies.DeleteCompany();

return "Компания успешно удалена";

}

catch (MyException ex)

{

return ex.Message;

}

}

public Stack<TransportCompany> GetAll()

{

return companies.GetTransportCompanies();

}

public void SaveChanges(int index, string selectedStrategy, string selectedMethod)

{

var transportCompanies = companies.GetTransportCompanies().Reverse().ToList();

TransportCompany company = transportCompanies[index];

if (selectedStrategy == "По массе и времени")

company.efficiencyStrategy = new MassTimeEfficiency();

else if (selectedStrategy == "По возрасту и цене")

company.efficiencyStrategy = new AgePriceEfficiency();

else if (selectedStrategy == "Сбалансированная")

company.efficiencyStrategy = new BalancedEfficiency();

company.RecalculateEfficiency();

if (selectedMethod == "По воздуху")

company.deliveryType = new AirDelivery();

else if (selectedMethod == "По воде")

company.deliveryType = new SeaDelivery();

else if (selectedMethod == "По земле")

company.deliveryType = new RoadDelivery();

}

}

}

### Пояснительный текст к программе

В данной программе реализован паттерн **MVP** (Model-View-Presenter), который помогает разделить логику приложения на три части: модель, представление и представителя.

* **Model.cs** – это часть модели, которая отвечает за данные и бизнес-логику. В этом файле описаны классы для транспортных компаний, включая их свойства (например, название, цена, масса груза, рейтинг и т. д.), а также методы для расчёта рейтинга и работы с различными способами доставки (с помощью паттернов "Мост" и "Стратегия"). Классы, такие как TransportCompany, LogisticCompany и другие, содержат логику для создания и работы с объектами транспортных компаний, а также для вычисления и хранения их данных.
* **FormView.cs** – это представление (форма), которое отвечает за отображение данных пользователю и взаимодействие с ним. В этом файле находятся элементы формы Windows (например, DataGridView, кнопки и текстовые поля), которые отображают информацию о транспортных компаниях. Также здесь находятся обработчики событий, которые реагируют на действия пользователя, такие как создание, удаление и сохранение данных.
* **ConsoleView.cs –** представление (консоль), отвечающее за отображение данных пользователю и взаимодействие с ним.
* **Presenter.cs**. Представитель – это связующее звено между моделью и представлением. Он содержит бизнес-логику и управляет взаимодействием между View и Model.  
  В Presenter.cs реализована подписка на события представления (AddClicked, RemoveClicked, SaveChangesClicked и т. д.), и в ответ на них вызываются методы модели. Интерфейс IView представляет собой абстракцию пользовательского интерфейса и необходим для реализации паттерна MVP. Он обеспечивает связь между Presenter и конкретным представлением (View1.cs, View2.cs), позволяя Presenter взаимодействовать с представлением через четко определённые события и свойства. Это делает код более гибким, модульным и удобным для тестирования. В интерфейсе определены события (AddClicked, RemoveClicked, ShowAllClicked и др.) и свойства, необходимые для получения данных от пользователя.

**Принцип работы:**

* **Модель** (Model.cs) управляет данными (например, транспортными компаниями) и бизнес-логикой (расчёт рейтинга, выбор метода доставки и т. д.).
* **Представление** (FormView.cs и ConsoleView) отображает эти данные на экране и получает ввод от пользователя (например, имя компании или её цену).
* **Представитель** (Presenter.cs) слушает события представления, обрабатывает их, взаимодействует с моделью и обновляет представление в ответ на изменения.

### Результаты работы программы

Результаты работы программы показаны на рисунках 1-3.

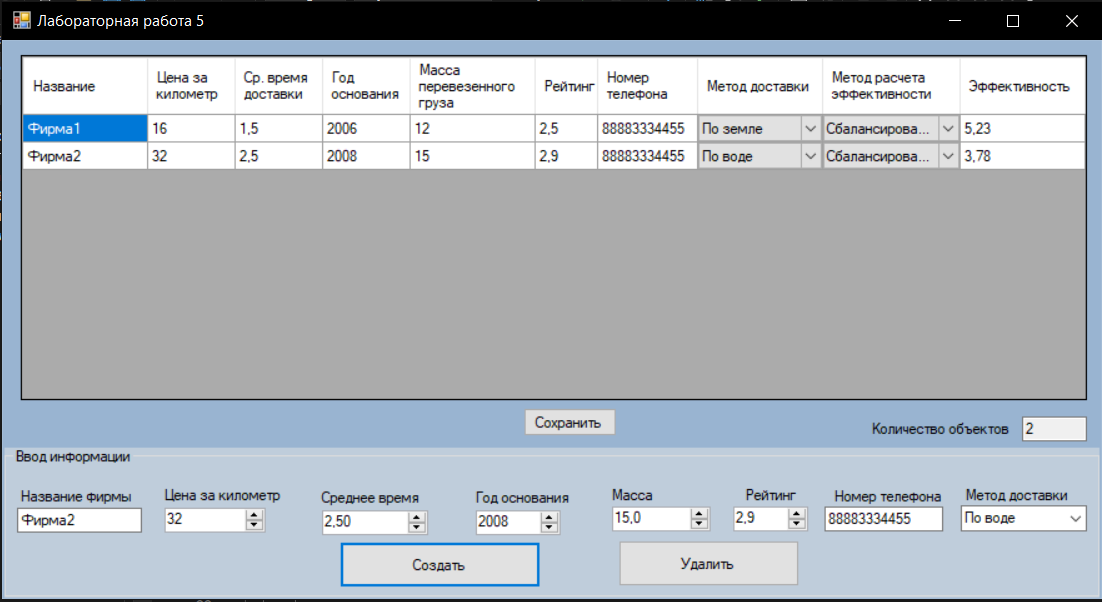


Рисунок 1 – Создание объекта, добавление в стек и вывод информации (FormView)

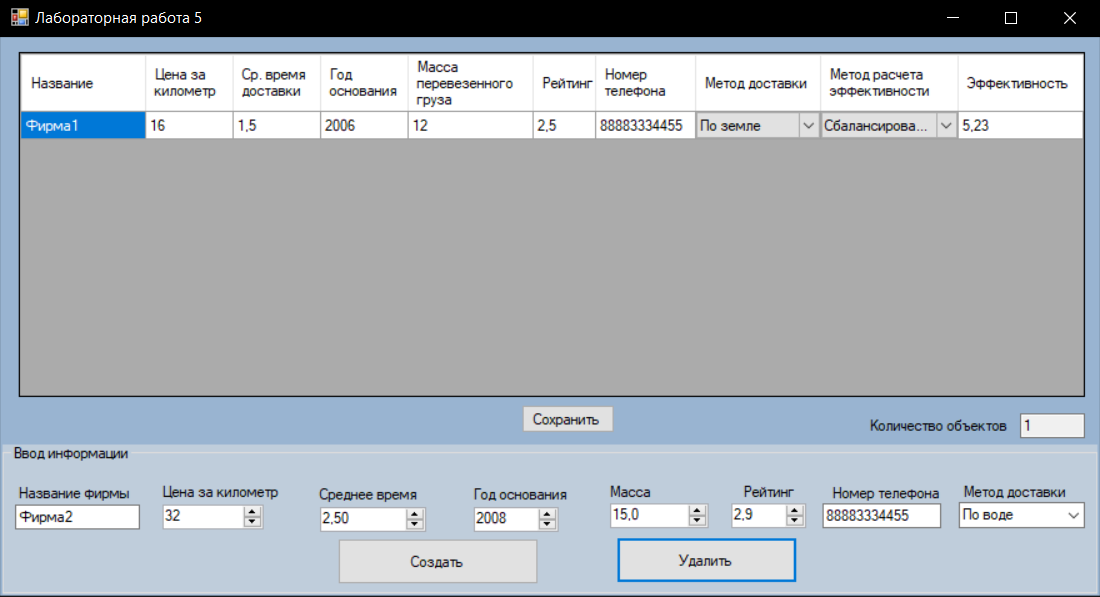


Рисунок 2 – Удаление фирмы из стека и вывод информации (FormView)

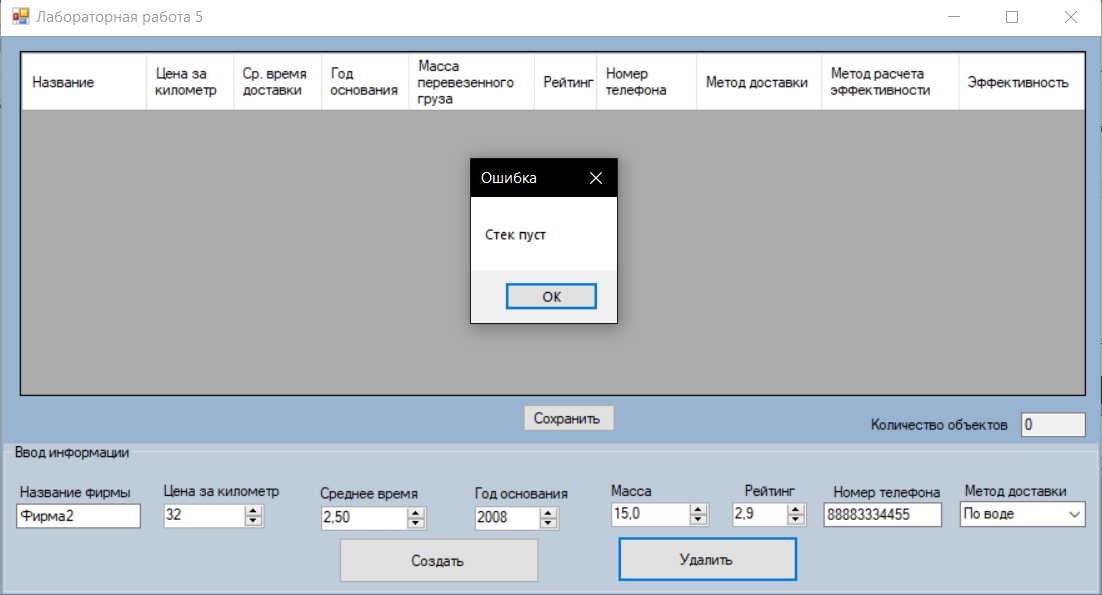


Рисунок 3 – Попытка удаления элемента из пустого стека (FormView)

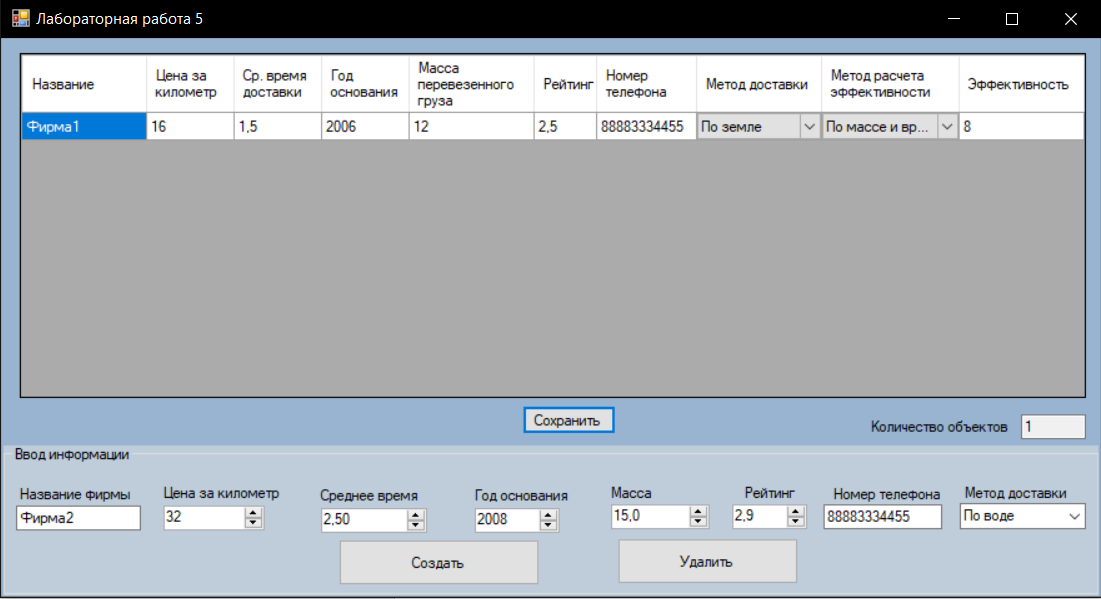


Рисунок 4 – Попытка изменения стратегии вычисления рейтинга и сохранения изменений (FormView)

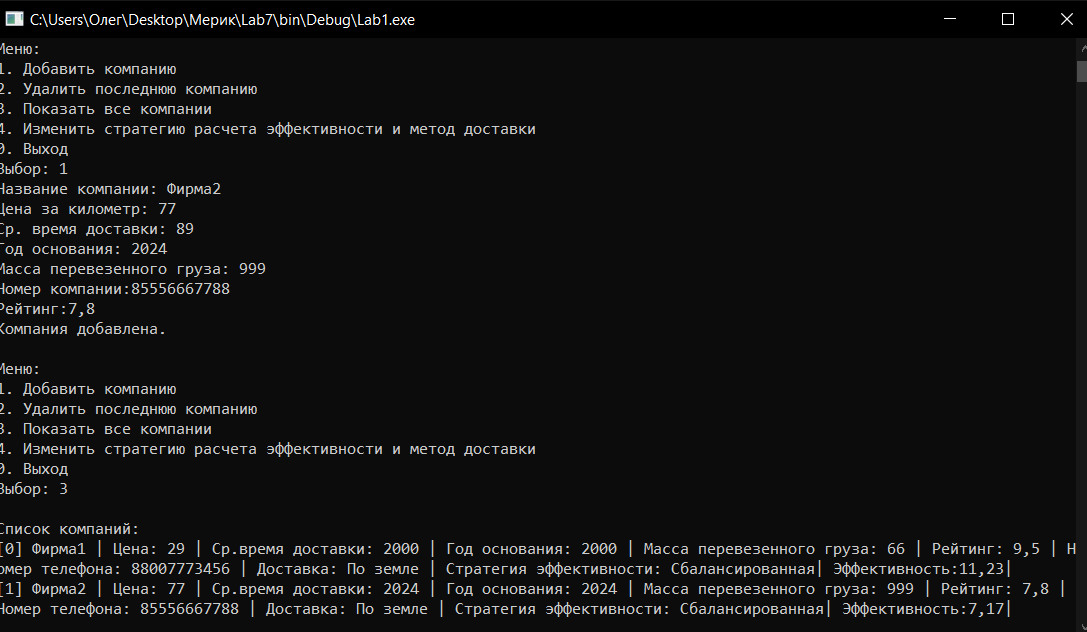


Рисунок 5 – Добавление компании в стек (ConsoleView)

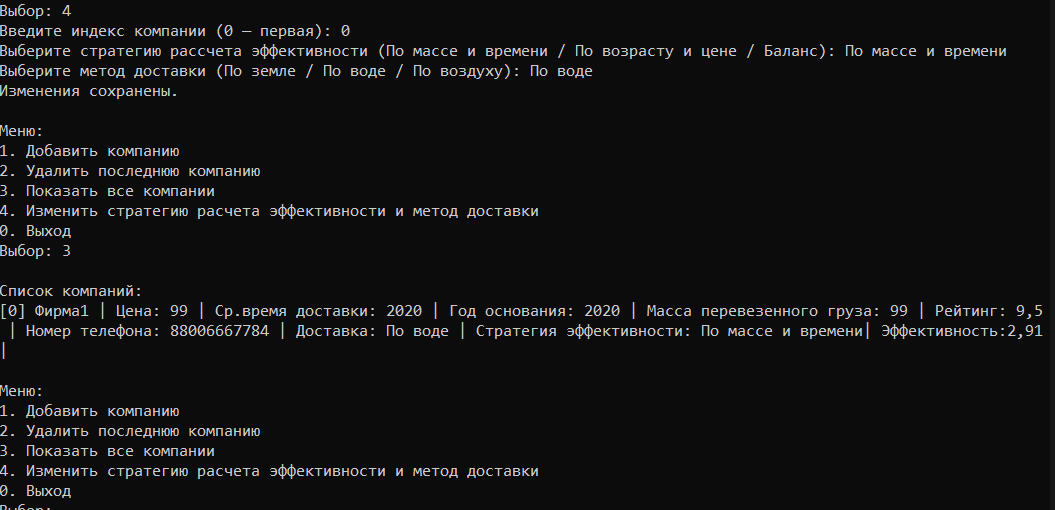


Рисунок 6 – Изменение метода доставки и расчета рейтинга (ConsoleView)

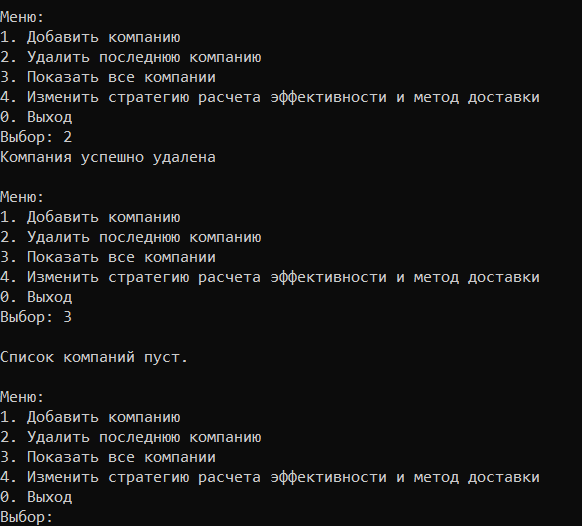


Рисунок 7 – Удаление компании из стека (ConsoleView)

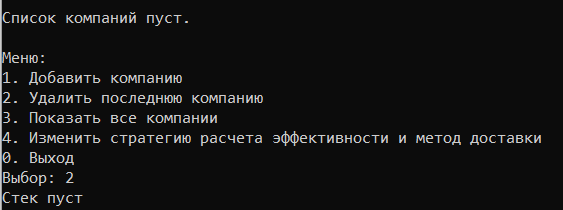


Рисунок 8 ­– Удаление компании из пустого стека (ConsoleView)

### Диаграмма вариантов использования

### C:\Users\Влад\Downloads\2025-03-31_19-50-47.png

Рисунок 5 - Диаграмма вариантов использования

### Выводы

В ходе лабораторной работы была реализована архитектура паттерна MVP (Model-View-Presenter), что позволило чётко разделить логику приложения на три компонента: модель, представление и посредник между ними — представитель. Это повысило модульность и читаемость кода, а также упростило сопровождение и масштабирование проекта. Благодаря использованию интерфейса IView удалось абстрагироваться от конкретной реализации интерфейса пользователя, что улучшает расширяемость и делает возможным лёгкое тестирование логики представителя независимо от формы.