Федеральное государственное бюджетное образовательное учреждение высшего образования
Национальный исследовательский университет «Московский энергетический институт»
Кафедра ВМСС

Лабораторная работа №5-6

«Паттерны проектирования, язык текстовых шаблонов T4»

Курс: Технологии разработки программного обеспечения

Группа: А-07м-23

Выполнил: Кретов Н.В.

Проверила: Раскатова М.В.

Москва 2024

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА 5

1. Цель

Главной задачей данной лабораторной работы является ознакомление и практическое применение паттернов проектирования при разработке программного обеспечения.

2. Вариант задания

Реализовать на языке C# приложение с графическим интерфейсом, с использованием паттерна, указанного в таблице 1 в соответствии со своим вариантом.

Таблица 1

Вариант задания к лабораторной работе №5

Вариант	Паттерн
6	Facade

В качестве второго применяемого в приложении паттерна был выбрал паттерн Prototype.

3. Описание предметной области

В качестве предметной области была выбрана система создания, редактирования, копирования и удаления персонажей для RPG-игры.

Пользователю должен быть доступен следующий функционал:

- Создание нового персонажа
- Копирование имеющегося персонажа
- Редактирование имеющегося персонажа
- Удаление имеющегося персонажа

Также должно быть реализовано сохранение данных (на момент закрытия программы) в файл формата scv и чтение данных из этого файла при запуске программы.

Логирования всех происходящих действий в программе (например, создание объектов классов, их удаление, вызов методов и пр.) должно происходить в текстовый файл.

4. Аргументация выбора паттернов

Одной из причин выбора паттерна проектирования Facade в написанном приложении, является необходимость предоставить пользователю простой интерфейс к довольно сложной системе взаимодействующих классов. При этом минимизируется зависимость работы системы от взаимодействия с пользователем (т.е. у пользователя нет необходимости в полном понимании внутренней структуры системы и ее компонентов, принципов ее работы, и в ручном управлении отдельными ее элементами). Еще одним преимуществом применения данного паттерна проектирования является возможность существенного изменения и расширения системы без необходимости существенного изменения в предоставленном пользователю интерфейса управления.

Так как клонирование объекта в разработанной системе более предпочтительный метод, чем создание нового объекта с помощью конструктора, то в качестве второго реализуемого паттерна проектирования был выбран Prototype. Другим важным фактором, повлиявшим на выбор данного паттерна проектирования, послужил тот факт, что класс порождаемого объекта не известен заранее, а определяется в момент выполнения.

5. Описание классов

В таблицах 2-5 представлено описание полей и методов классов CreateCharacterForm и MainForm, которые наследуются от стандартного класса Form (пространство имен System.Windows.Forms). Описание полей и методов вложенных в MainForm классов (Logger, Character, Knight, Archer, Wizard, Healer, Craftsman, Thief, PersoneRedactor) представлено далее в таблицах 6-11.

Таблица 2

Описание полей класса CreateCharacterForm

Имя	Тип	Модификаторы	Описание
_mainForm	MainForm	private	Поле, являющееся экземпляром класса MainForm и позволяющее взаимодействовать CreateCharacterForm с MainForm

Таблица 3

Описание методов класса CreateCharacterForm

Имя	Тип возвращаемого значения	Модификаторы	Входные параметры	Описание
CreateCharacterForm	-	public	**mainForm** – ссылка на MainForm **mode** – режим работы формы CreateCharacterForm	Конструктор класса CreateCharacterForm
SetUpdateMode	void	private	-	Метод, позволяющий установить режим работы «редактирование уже имеющегося персонажа» формы CreateCharacterForm
CheckData	bool	private		Метод для проверки заполненности всех необходимых полей для создания персонажа
CreateButton_Click	void	private	**sender** – содержит ссылку на элемент управления/объект, вызвавший событие **e** – содержит данные о событии	Метод, вызываемый при нажатии кнопки «Create» или «Update» и передающий введенные пользователем данные в структуру класса MainForm
AgeTrackBar_Scroll	void	private		Метод, вызываемый при каждом прокручивании AgeTrackBar, вызывающий метод CheckData
NameTextBox_TextChanged	void	private		Метод, вызываемый при каждом изменении текста в NameTextBox, вызывающий метод CheckData
SexComboBox_SelectedIndexChanged	void	private		Метод, вызываемый при каждом выборе элемента из выпадающего списка SexComboBox, вызывающий метод CheckData
ClassComboBox_SelectedIndexChanged	void	private		Метод, вызываемый при каждом выборе элемента из выпадающего списка ClassComboBox, вызывающий метод CheckData
CreateCharacterForm_FormClosing	void	private		Метод, вызываемый при закрытии формы CreateCharacterForm (при вызове управление передается форме MainForm)

Таблица 4

Описание полей класса MainForm

Имя	Тип	Модификаторы	Описание
_pathForRead	string	private static	Поле, хранящее путь к файлу для считывания
_pathForWrite	string	private static	Поле, хранящее путь к файлу для записи
_curentID	uint	private static	Поле-счетчик, для обеспечения уникальности id
_logger	Logger	private static	Поле, являющееся экземпляром класса Logger, для обращения к методам класса Logger из статических методов
_characters	List<Character>	private static	Поле, являющееся списком всех созданных персонажей
_charactersTable	DataGridView	private static	Поле, являющееся экземпляром класса DataGridView, для обращения к методам класса DataGridView из статических методов
_mainForm	MainForm	private static	Поле, являющееся экземпляром класса MainForm, для обращения к методам класса MainForm из статических методов
characterData	CharacterData	public static	Поле, являющееся экземпляром структуры CharacterData, для храния данных (имя, возраст, пол и класс персонажа), полученных от формы CreateCharacterForm

Таблица 5

Описание методов класса MainForm

Имя	Тип возвращаемого значения	Модификаторы	Входные параметры	Описание
MainForm	-	public	-	Конструктор класса MainForm
CreateButton_Click	void	private	**sender** – содержит ссылку на элемент управления/объект, вызвавший событие **e** – содержит данные о событии	Метод, вызываемый при нажатии кнопки «CreateCharacter» и вызывающий метод Createкласса PersoneRedactor
CopyButton_Click	void	private		Метод, вызываемый при нажатии кнопки «CopyCharacter» и вызывающий метод Copy класса PersoneRedactor
UpdateButton_Click	void	private		Метод, вызываемый при нажатии кнопки «UpdateCharacter» и вызывающий метод Update класса PersoneRedactor
DeleteButton_Click	void	private		Метод, вызываемый при нажатии кнопки «DeleteCharacter» и вызывающий метод Delete класса PersoneRedactor
OpenCreateForm	void	private static	**mode** – режим работы формы CreateCharacterForm	Метод, вызываемый с целью открытия «диалога» с формой CreateCharacterForm (форма MainForm становится не активной)

Таблица 5 Продолжение

CreateCharacter	Character	private static	**id** – уникальный id персонажа	Метод, который в зависимости от выбранного класса персонажа возвращает экземпляр соответствующего класса (Knight, Archer, Wizard, Healer, Craftsman, Thief)
ReadDataFromCSV	void	private static	-	Метод, вызываемый при запуске программы, для считывания данных о персонажах из файла формата csv
MainForm_FormClosed	void	private	**sender** – содержит ссылку на элемент управления/объект, вызвавший событие **e** - содержит данные о событии	Метод, вызываемый при закрытии формы CreateCharacterForm (при вызове имеющиеся данные о персонажах записываются в файл формата csv)

Таблица 6

Описание полей класса Logger

Имя	Тип	Модификаторы	Описание
_instance	Logger	private static	Поле, являющееся экземпляром класса Logger, для обращения к методам класса Logger из статических методов

Таблица 7

Описание методов класса Logger

Имя	Тип возвращаемого значения	Модификаторы	Входные параметры	Описание
Logger	-	private	-	Конструктор класса Logger
GetInstance	Logger	public static	-	Метод, вызываемый при запуске программы и возвращающий ссылку на единственный экземпляр класса Logger (реализация паттерна проектирования Singleton)
WriteToLogFile	void	public	**message** – сообщение, которое должно быть записано в текстовый файл	Метод, вызываемый для запили (логирования) событий в системе в текстовый файл

Таблица 8

Описание свойств абстрактного класса Character

Имя	Тип	Геттеры и сеттеры	Описание
_id	uint	{ get; protected set; }	Свойство, используемое для хранения уникального id персонажа
_characterClass	Character Class	{ get; protected set; }	Свойство, используемое для хранения класса персонажа
_ability	string	{ get; protected set; }	Свойство, используемое для хранения уникальной способности, присущей определенному классу персонажа
_name	string	{ get; private set; }	Свойство, используемое для хранения имени персонажа
_age	int	{ get; private set; }	Свойство, используемое для хранения возраста персонажа
_sex	Sex	{ get; private set; }	Свойство, используемое для хранения пола персонажа

Таблица 9

Описание методов класса Character

Имя	Тип возвращаемого значения	Модификаторы	Входные параметры	Описание
Character	-	public	**name** – имя персонажа **age** – возраст персонажа **sex** – пол персонажа	Конструктор класса Character
Clone	Character	public virtual	-	Метод, вызываемый при желании клонировать экземпляр класса Character

Таблица 10

Описание методов классов Knight, Archer, Wizard, Healer, Craftsman, Thief (наследники класса Character)

Имя	Тип возвращаемого значения	Модификаторы	Входные параметры	Описание
Knight	-	public	**id** – уникальный id персонажа **name** – имя персонажа **age** – возраст персонажа **sex** – пол персонажа	Конструктор класса Knight
Archer	-	public	**id** – уникальный id персонажа **name** – имя персонажа **age** – возраст персонажа **sex** – пол персонажа	Конструктор класса Archer
Wizard	-	public	**id** – уникальный id персонажа **name** – имя персонажа **age** – возраст персонажа **sex** – пол персонажа	Конструктор класса Wizard
Healer	-	public	**id** – уникальный id персонажа **name** – имя персонажа **age** – возраст персонажа **sex** – пол персонажа	Конструктор класса Healer

Таблица 10 Продолжение

Craftsman	-	public	**id** – уникальный id персонажа **name** – имя персонажа **age** – возраст персонажа **sex** – пол персонажа	Конструктор класса Craftsman
Thief	-	public	**id** – уникальный id персонажа **name** – имя персонажа **age** – возраст персонажа **sex** – пол персонажа	Конструктор класса Thief

Таблица 11

Описание методов класса PersoneRedactor

Имя	**Тип возвращаемого значения**	**Модификаторы**	**Входные параметры**	**Описание**
Create	void	public static	-	Метод, вызываемый из метода CreateButton_Click и отвечающий за весь процесс создания нового персонажа
Copy	void	public static	-	Метод, вызываемый из метода CreateButton_Click и отвечающий за весь процесс копирования уже существующего персонажа
Update	void	public static	-	Метод, вызываемый из метода CreateButton_Click и отвечающий за весь процесс редактирования уже существующего персонажа
View	void	public static	-	Метод, вызываемый из методов Create, Copy, Update и отвечающий а процесс вывода актуальной информации о персонажах на DataGridView

6. Диаграмма классов

На рис. 1 представлена диаграмма классов.

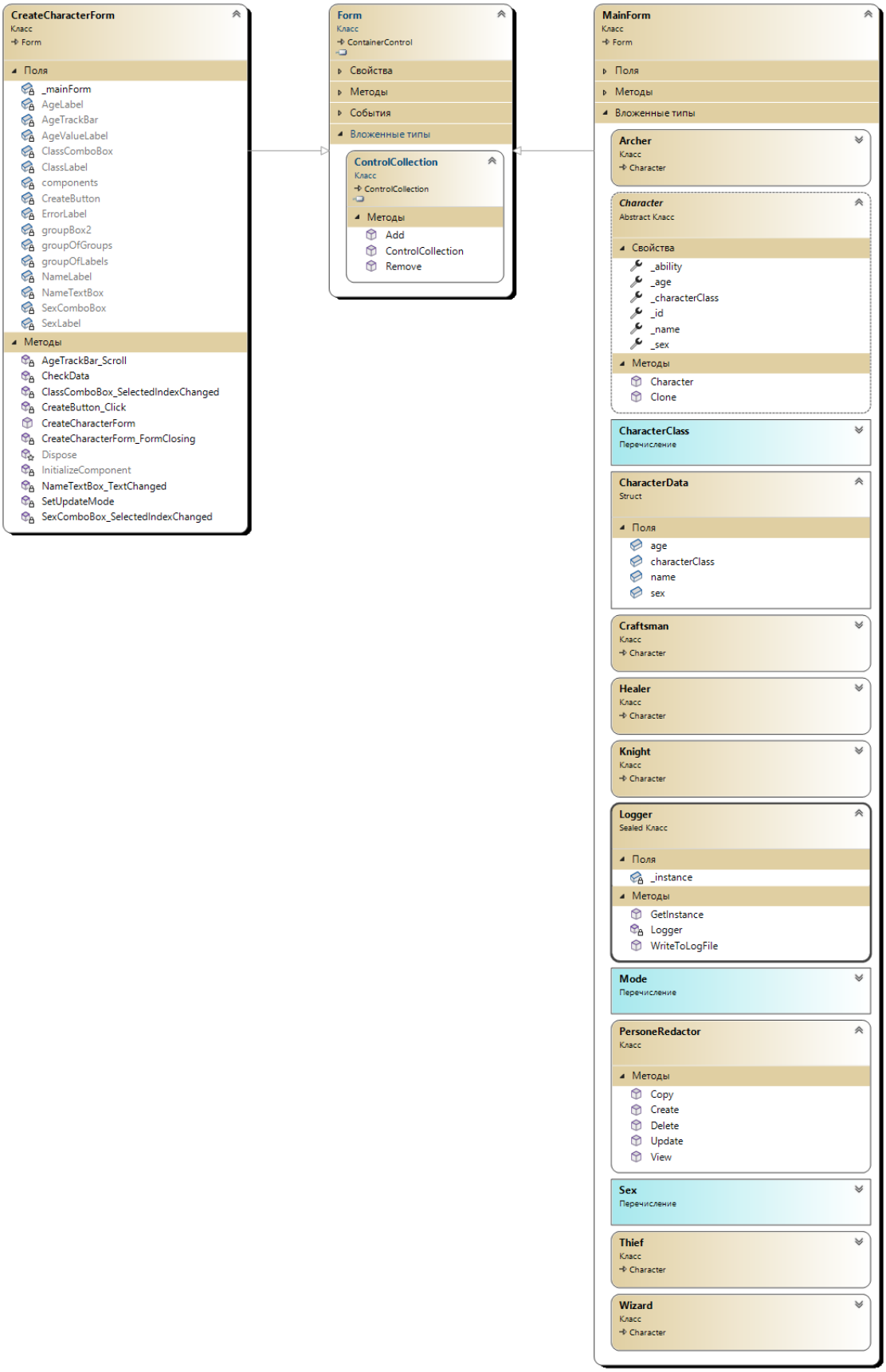

Рис. 1. Диаграмма классов

Как видно из диаграммы, программа представляет из себя две формы (главное окно программы и окно для внесения данных о персонаже), которые наследуются от стандартного класса Form (пространство имен System.Windows.Forms).

При этом в главной форме реализованы вложенные типы:

1. Структура (struct):
   a. CharacterData (данные о персонаже, передаваемые между формами);
2. Перечисления (enums):
   a. Sex (пол);
   b. Mode (режим работы окна CreateCharacterForm);
   c. CharacterClass (класс персонажа).
3. Классы (classes):
   a. Logger (класс без возможности наследования, реализующий паттерн проектирования Singleton для логирования событий в системе в текстовый файл);
   b. PersoneRedactor (реализация паттерна проектирования Facade для предоставление пользователю простого интерфейса для взаимодействия с системой);
   c. Character (абстрактный класс, реализующий паттерн проектирования Prototype);
   d. Knight, Archer, Healer, Wizard, Craftsman, Thief (наследники класса Character, являющиеся конкретными реализациями прототипа).

7. Результат работы программы

На рис. 2-14 представлены результаты работы программы.

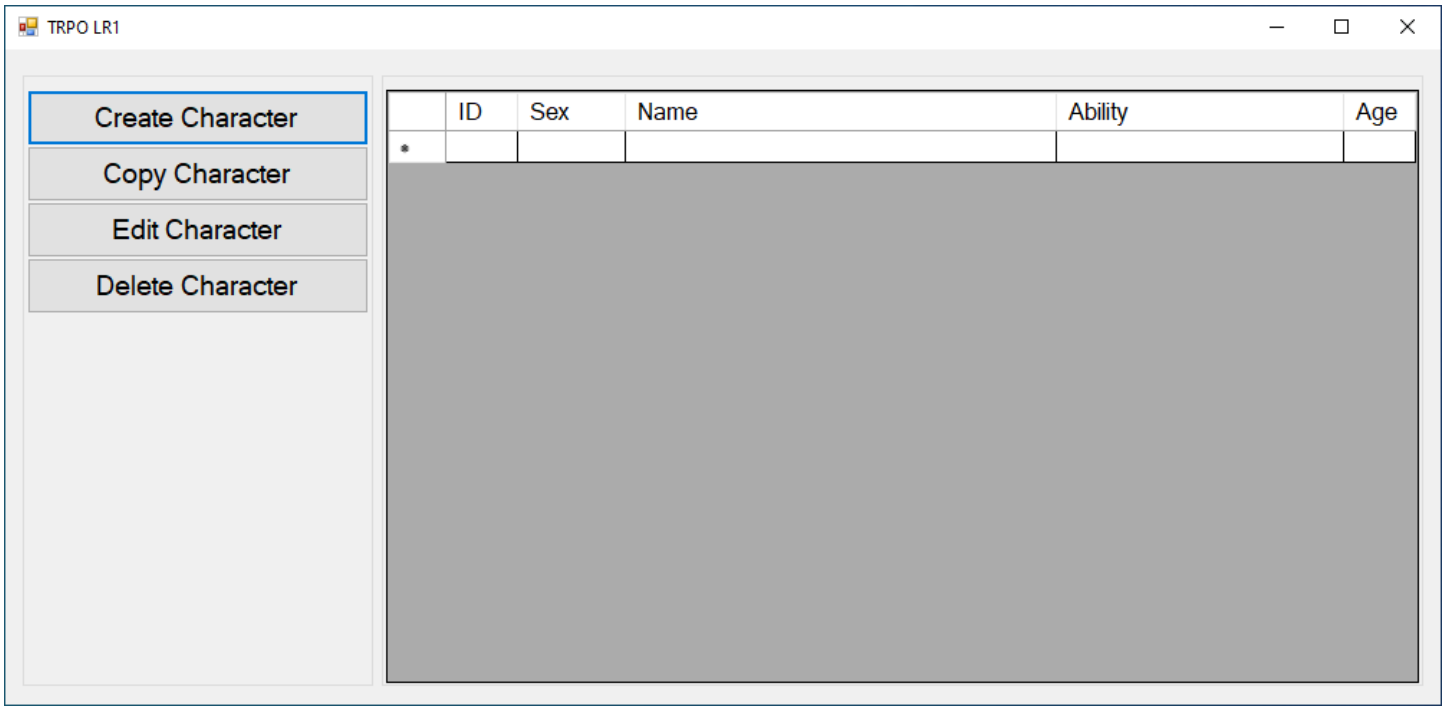

Рис. 2. Первый запуск программы

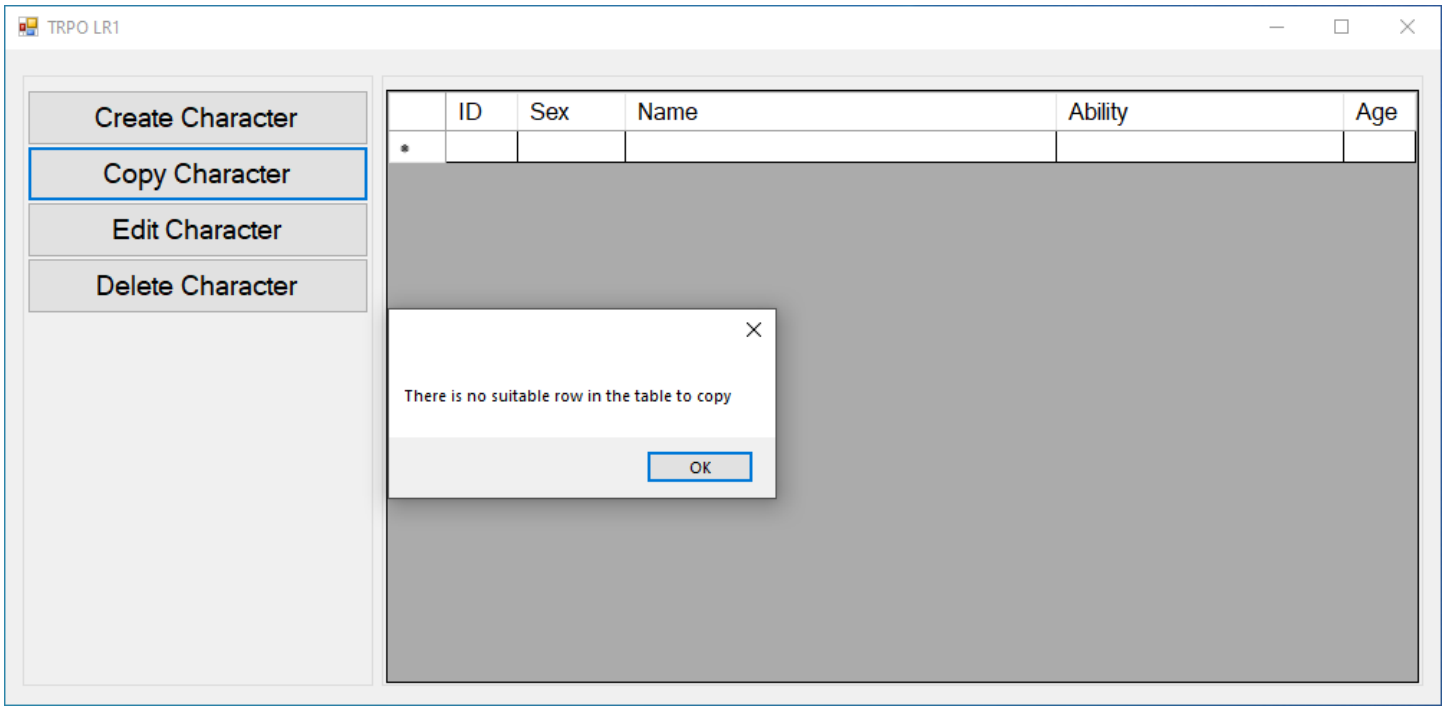

Рис. 3. Попытка скопировать запись

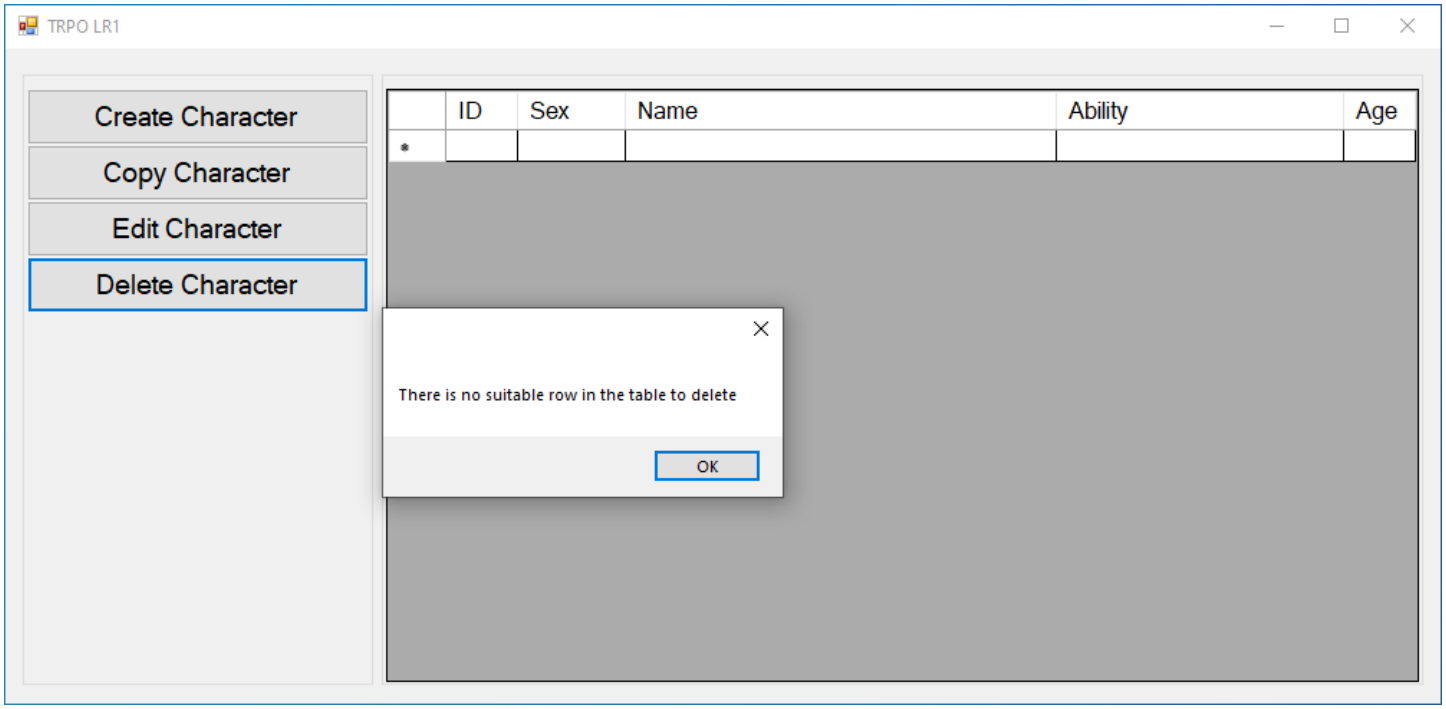

Рис. 4. Попытка удалить запись

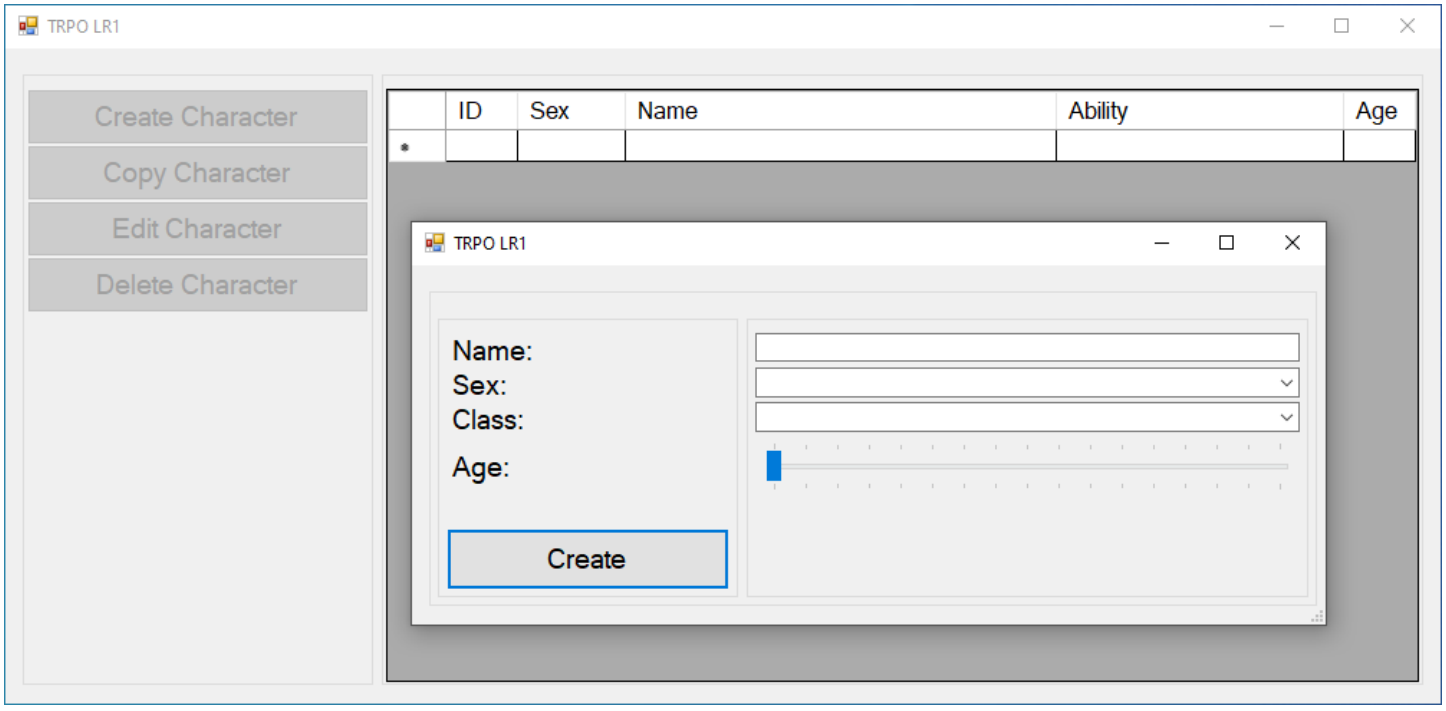

Рис. 5. Вызов окна для создания записи

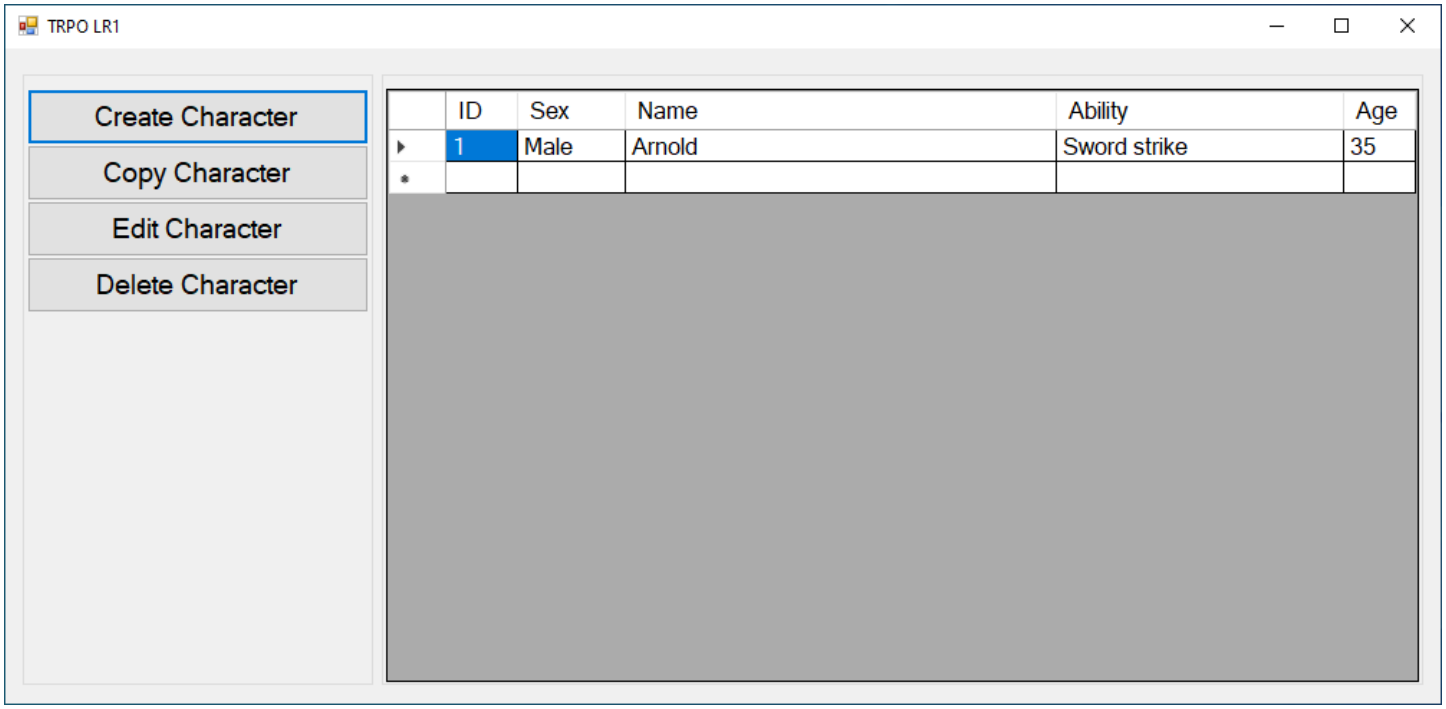

Рис. 6. Результат создания записи

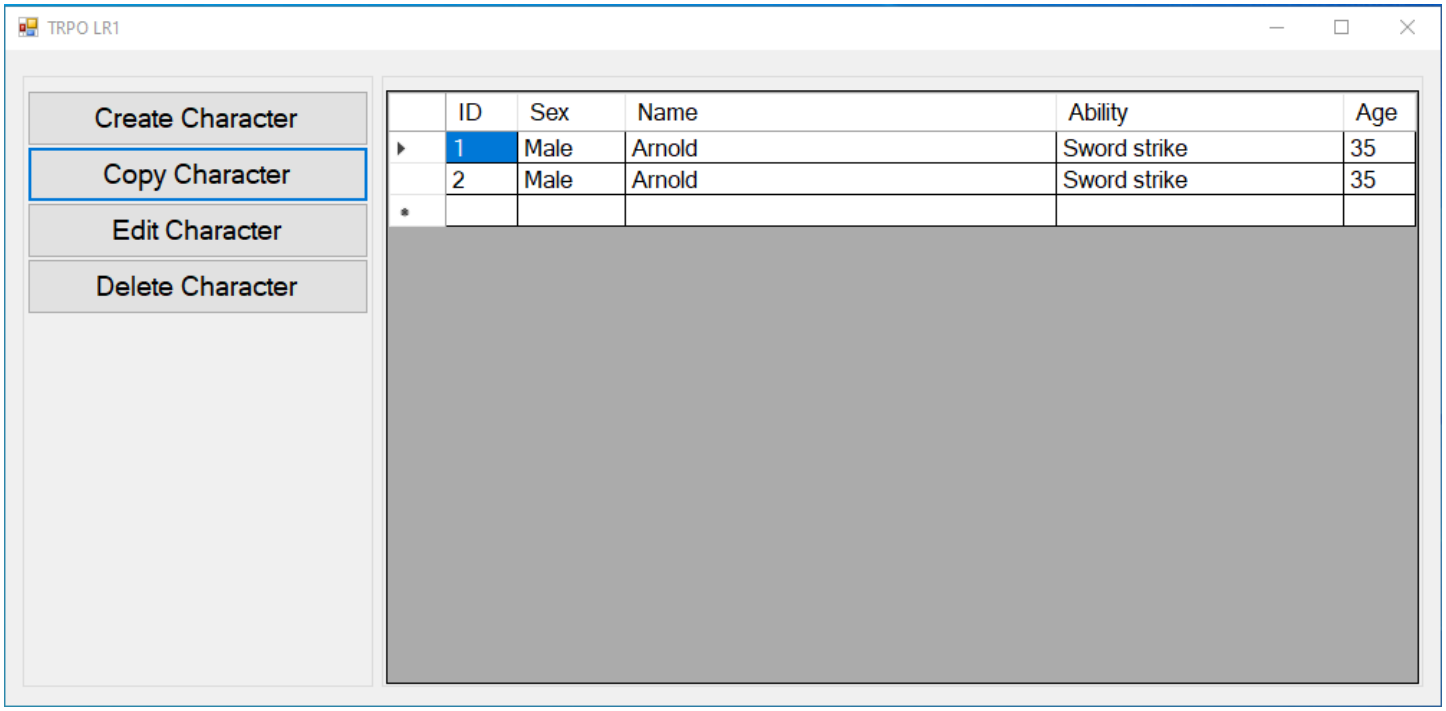

Рис. 7. Результат копирования записи

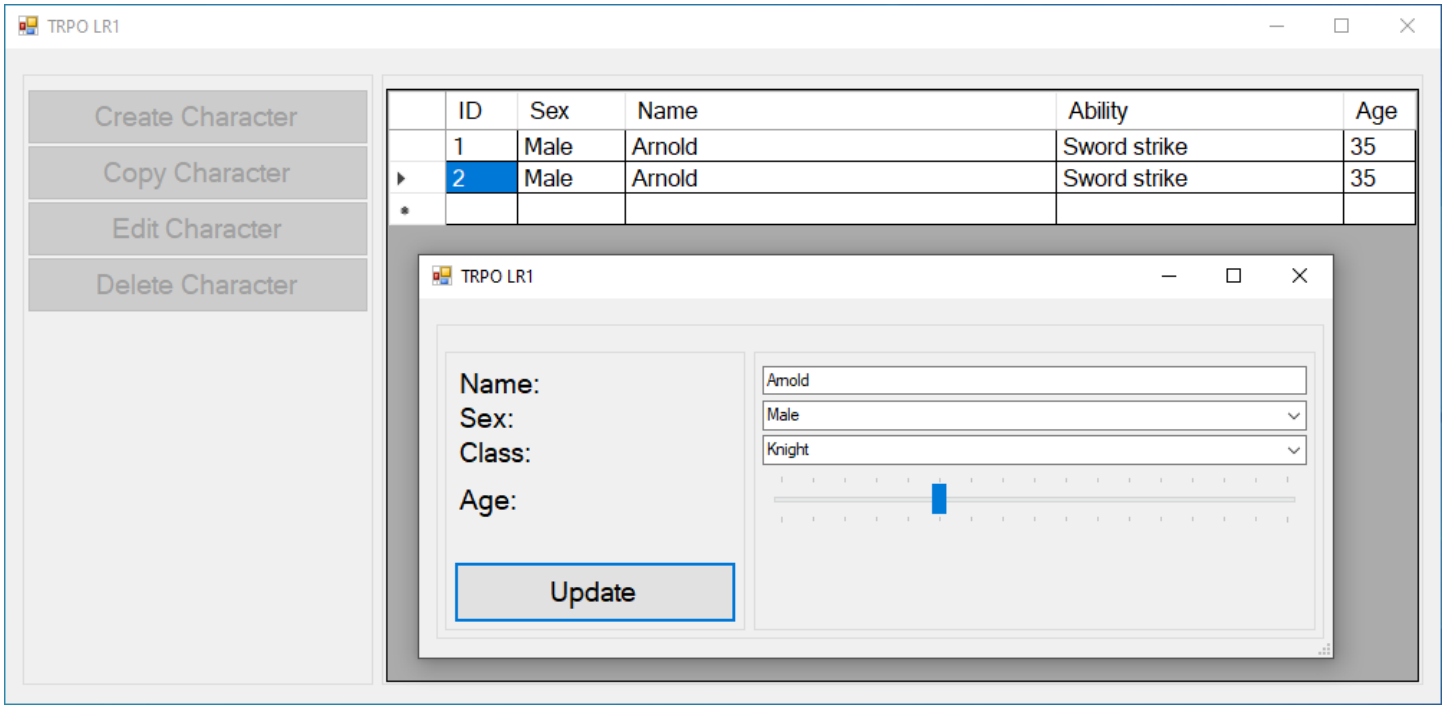

Рис. 8. Вызов окна для редактирования записи

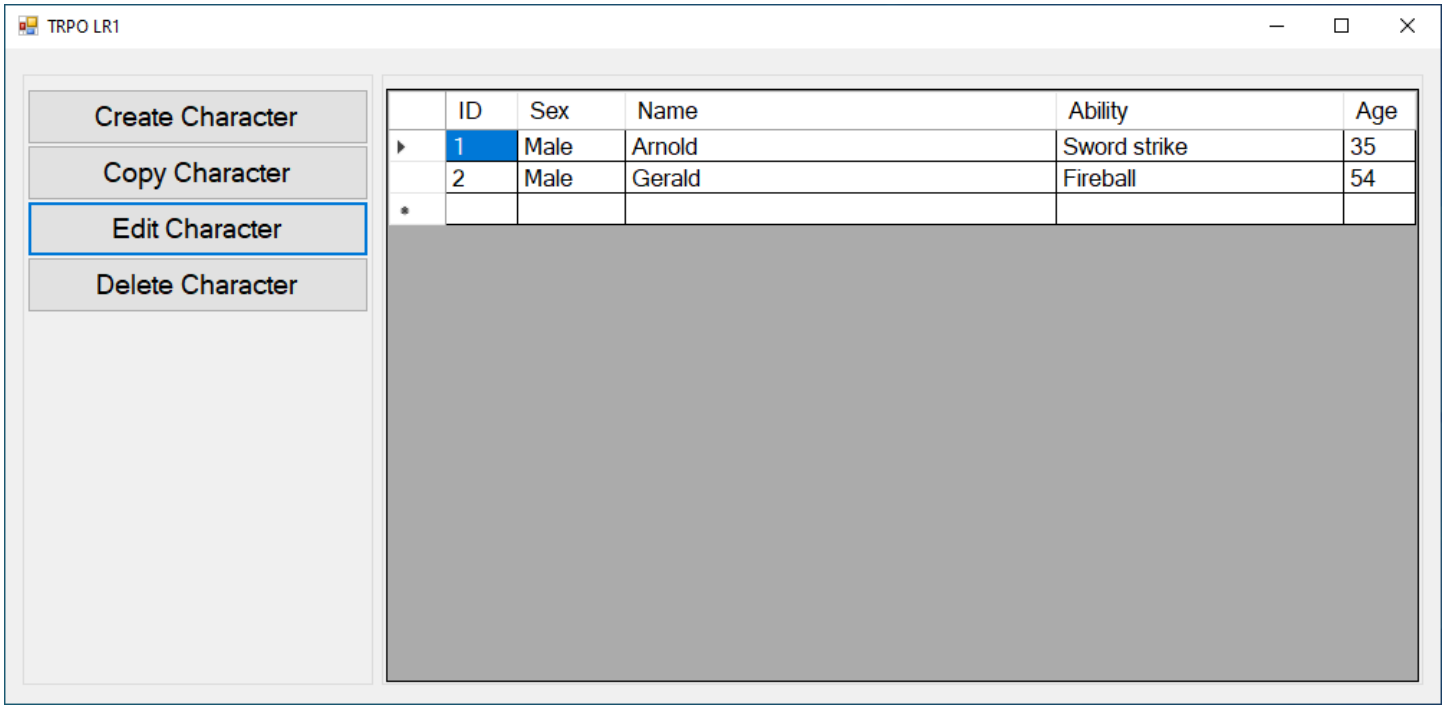

Рис. 9. Результат редактирования записи

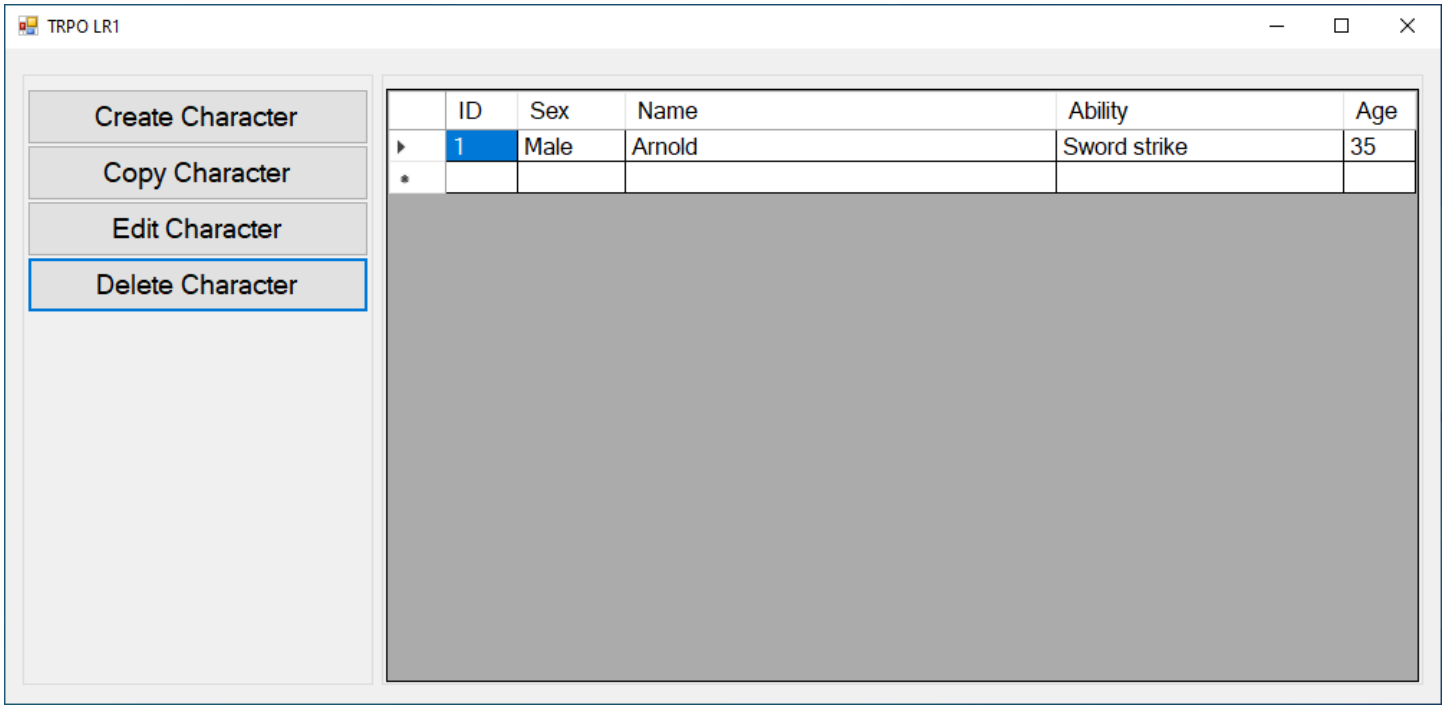

Рис. 10. Результат удаления записи

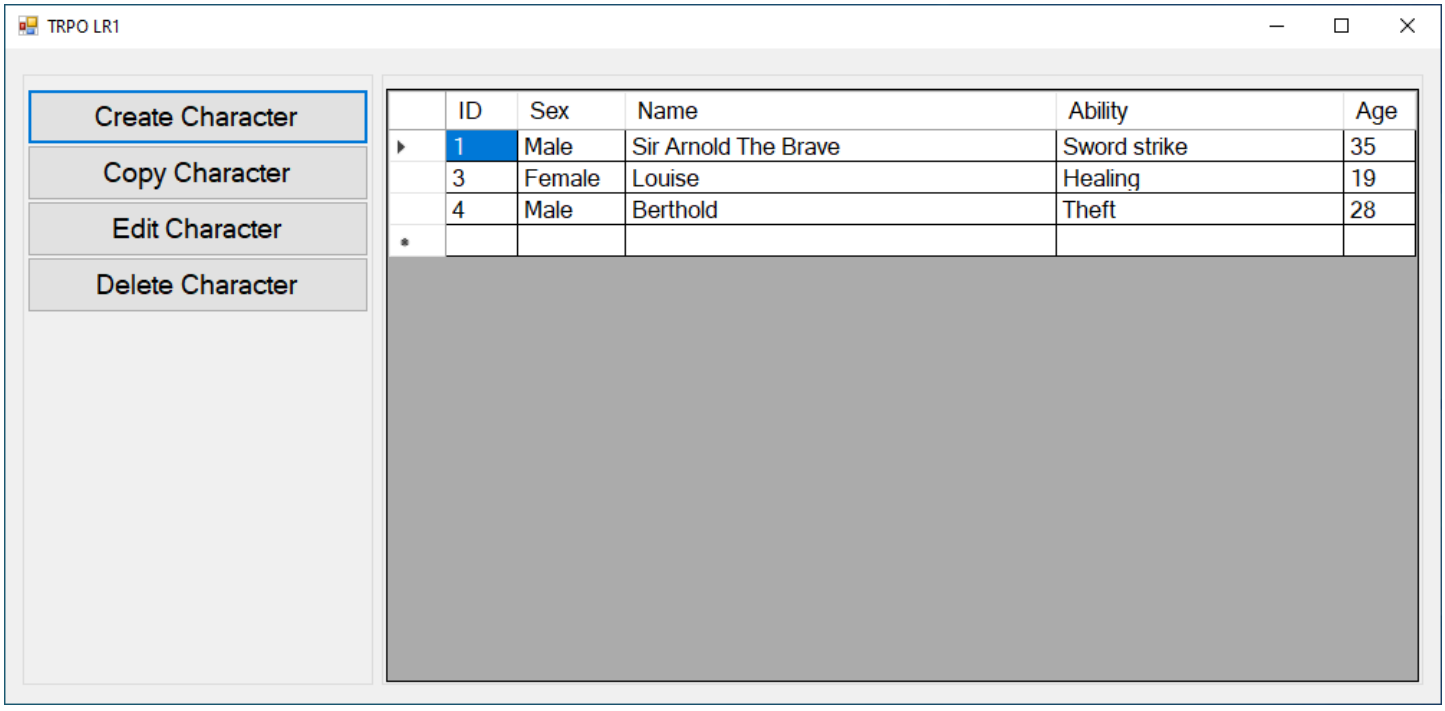

Рис. 11. Добавление еще нескольких записей

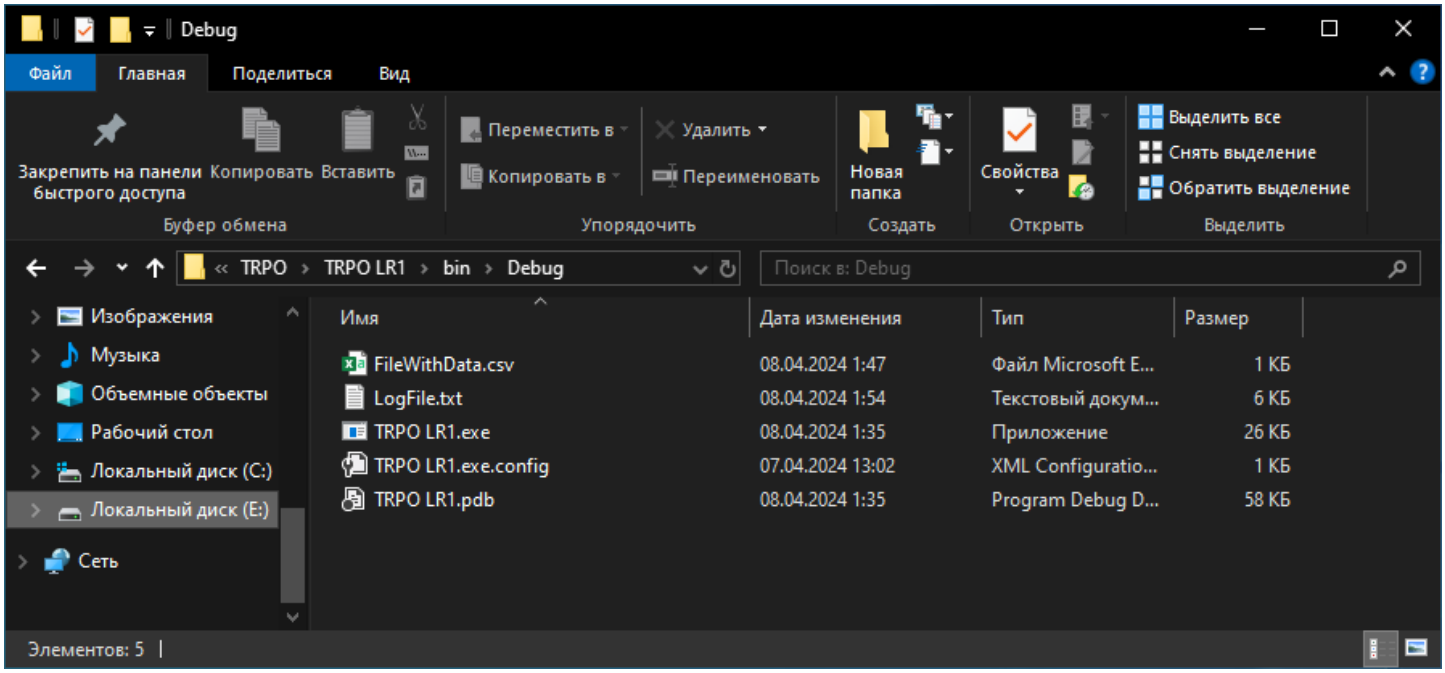

Рис. 12. Репозиторий проекта после завершения программы

Как видно из рис. 12, в репозитории проекта после завершения программы появились файлы FileWithData.csv (файл, содержащий записи, оставшиеся на момент завершения программы) и LogFile.txt (файл с логами).

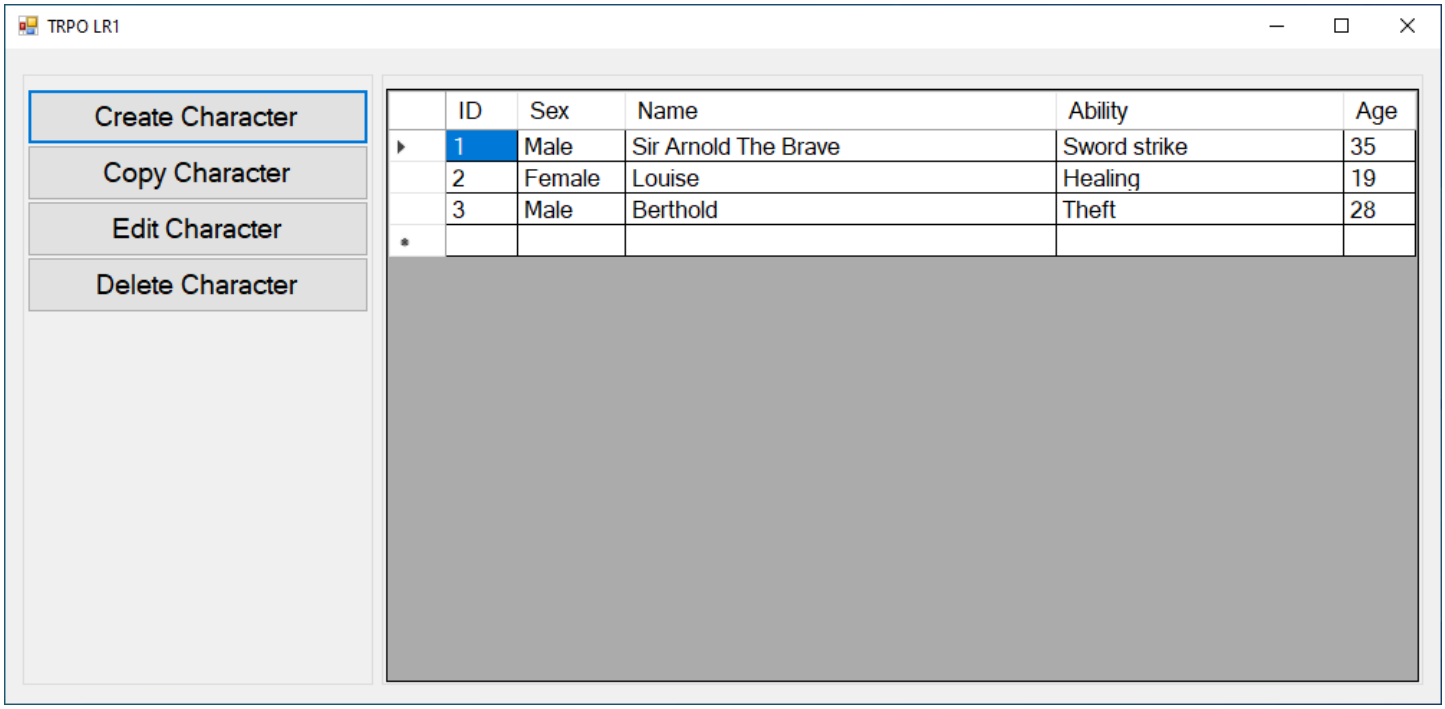

Рис. 13. Второй запуск программы

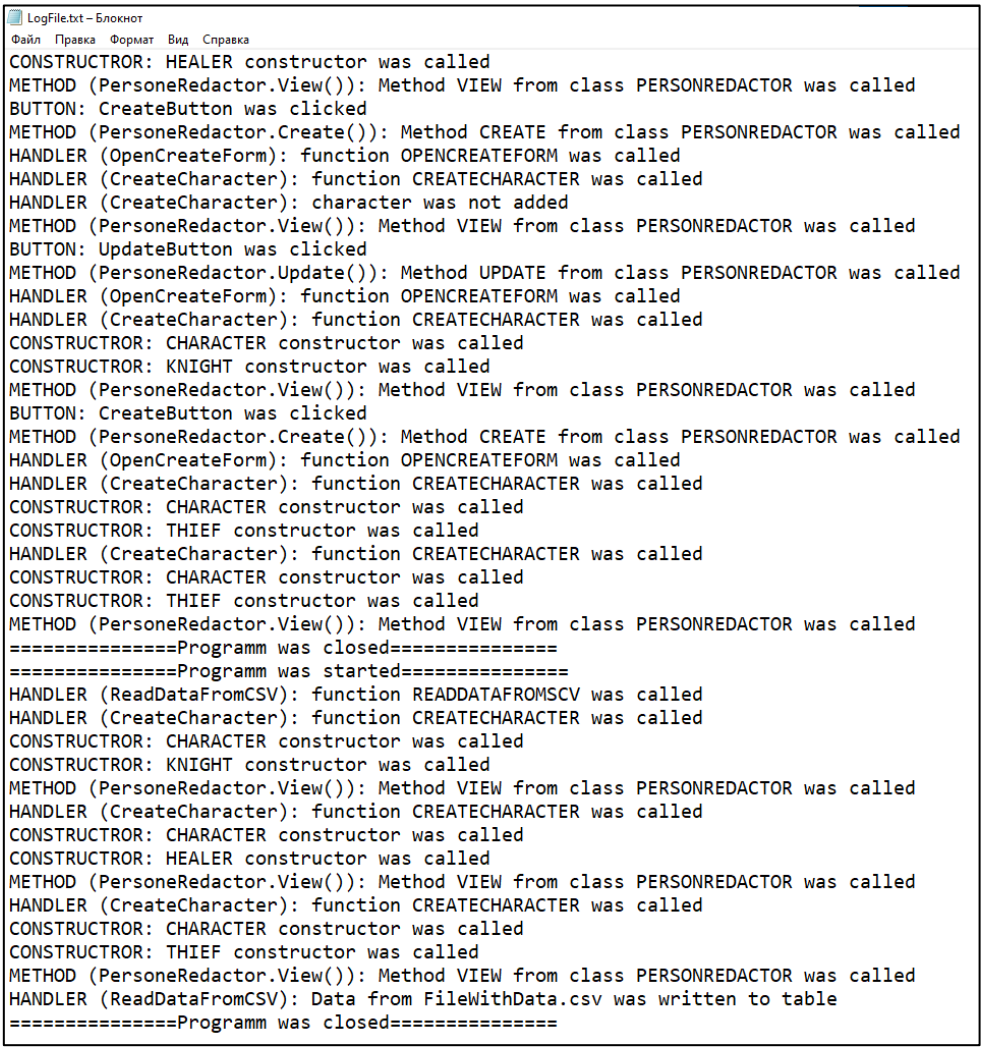

Рис. 14. Пример логирования

С полным листингом программы можно ознакомиться в приложении А.

ЛАБОРАТОРНАЯ РАБОТА 6

1. Цель

Главной задачей данной лабораторной работы является ознакомление и практическое применение языка текстовых шаблонов T4.

2. Вариант задания

Тип задания: генератор SQL-запросов к БД SomeDB.

Необходимо получить выборку (select) определённых полей Field_1, Field_2, …, Field_M из таблиц Table_1, Table_2, … Table_N (в каждой из N таблиц содержатся M полей).

Какие именно поля выбирать, а также сколько таблиц N и полей M – указано в варианте (см. таблицу 12). Результат – набор корректных select-ов из таблиц.

Таблица 12

Вариант задания к лабораторной работе №6

Вариант	Данные
4	Генератор SQL-запросов, N = 5, M = 10, из первой таблицы вывести поля 1 и 2, из второй таблицы вывести поля 3 и 4, …, из таблицы N вывести поля M-1 и M. Разрешено использовать любые циклы и блоки. Обязательно создание собственного класса в блоке расширения и использования его в коде шаблона.

3. Результат работы программы

В результате работы написанного текстового шаблона LR2.tt (см. рис. 15) был сгенерирован текстовый файл LR2.txt, содержание которого представлено на рис. 16.

```
<#@ template debug="false" hostspecific="false" language="C#" #>
<#@ output extension=".txt" #>

<#
        string[] selectors = SelectorGenerator.GenerateArrayOfSelectors();

        for (int i = 0; i < SelectorGenerator.N; i++)
        {
                Write(selectors[i]);
        }
#>

<#+
        class SelectorGenerator
                {
                        public const int N = 5;         //кол-во таблиц

                        public static string[] GenerateArrayOfSelectors()
                        {
                                string[] selectors = new String[N];

                                for (int i = 1; i <= N; i++)
                                {
                                        int j = 2 * i - 1;
                                        selectors[i-1] = "SELECT Field_" + j + ", Field_" + ++j + " FROM Table_" + i + "\n";
                                }

                                return selectors;
                        }
                }
#>
```

Рис.15. Содержание текстового шаблона LR2.tt

```
SELECT Field_1, Field_2 FROM Table_1
SELECT Field_3, Field_4 FROM Table_2
SELECT Field_5, Field_6 FROM Table_3
SELECT Field_7, Field_8 FROM Table_4
SELECT Field_9, Field_10 FROM Table_5
```

Рис. 16. Содержание сгенерированного текстового файла LR2.txt

ВЫВОДЫ

В результаты выполнения лабораторной работы №5 была разработана система, позволяющая создавать, редактировать, копировать и удалять персонажей для RPG-игры, и которая может быть в дальнейшем дополнена и/или применена в рамках более сложной системы. При этом в разработке были применены паттерны проектирования Facade, Prototype и Singleton.

Было реализовано сохранение данных в файл формата scv и чтение данных из него. Решена задача логирования всех происходящих действий в программе в текстовый файл.

В ходе выполнения лабораторной работы №6 было произведено знакомство с языком текстовых шаблонов T4 и написан пример его применения для поставленной задачи (генерация SQL-запросов).

ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ

MainForm.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Windows.Forms;

namespace TRPO_LR1
{
    public partial class MainForm : Form
    {
        #region Variables
        private static string _pathForRead;
        private static string _pathForWrite;

        private static uint _curentID;
        private static Logger _logger;
        private static List<Character> _characters = new List<Character>();
        private static DataGridView _charactersTable;
        private static MainForm _mainForm;
        public static CharacterData characterData;
        #endregion

        #region Enums_and_Structures
        public enum Mode
        {
            Create = 1,
            Update = 2
        }

        public enum Sex
        {
            Male = 1,
            Female = 2
        }
        public enum CharacterClass
        {
            Knight = 1,
            Archer = 2,
            Wizard = 3,
            Healer = 4,
            Craftsman = 5,
            Thief = 6
        }
        public struct CharacterData
        {
            public string name;
            public int age;
            public Sex sex;
            public CharacterClass characterClass;
        }
        #endregion

        #region Constructors
        public MainForm()
        {
            _curentID = 1;
            _pathForRead = "FileWithData.csv";
            _pathForWrite = "LogFile.txt";
            characterData = new CharacterData();
```

```
            _logger = Logger.GetInstance();
            _logger.WriteToLogFile("===============Programm was started===============");

            InitializeComponent();

            _charactersTable = CharactersTable;
            _mainForm = this;

            ReadDataFromCSV();
        }
        #endregion

        #region Handler
        private static void OpenCreateForm(Mode mode)
        {
            _logger.WriteToLogFile("HANDLER (OpenCreateForm): function OPENCREATEFORM was
called");
            _mainForm.Enabled = false;
            CreateCharacterForm createCharacterForm = new CreateCharacterForm(_mainForm, mode);
            createCharacterForm.ShowDialog();
        }

        private static Character CreateCharacter(uint id)
        {
            _logger.WriteToLogFile("HANDLER (CreateCharacter): function CREATECHARACTER was
called");
            switch (characterData.characterClass)
            {
                case CharacterClass.Knight:
                    return new Knight(id, characterData.name, characterData.age,
characterData.sex);
                case CharacterClass.Archer:
                    return new Archer(id, characterData.name, characterData.age,
characterData.sex);
                case CharacterClass.Wizard:
                    return new Wizard(id, characterData.name, characterData.age,
characterData.sex);
                case CharacterClass.Healer:
                    return new Healer(id, characterData.name, characterData.age,
characterData.sex);
                case CharacterClass.Craftsman:
                    return new Craftsman(id, characterData.name, characterData.age,
characterData.sex);
                case CharacterClass.Thief:
                    return new Thief(id, characterData.name, characterData.age,
characterData.sex);
                default:
                    _curentID--;
                    _logger.WriteToLogFile("HANDLER (CreateCharacter): character was not added");
                    return null;
            }
        }

        private static void ReadDataFromCSV()
        {
            _logger.WriteToLogFile("HANDLER (ReadDataFromCSV): function READDATAFROMSCV was
called");
            try
            {
                using (StreamReader reader = new StreamReader(_pathForRead, Encoding.UTF8))
                {
```

```
                    string record;

                    while (!reader.EndOfStream)
                    {
                        record = reader.ReadLine();

                        string[] data = record.Split(new char[] { ',', '.' },
StringSplitOptions.RemoveEmptyEntries);

                        if (!int.TryParse(data[1], out int age) || !int.TryParse(data[2], out int
sex) || !int.TryParse(data[3], out int characterClass))
                        {
                            continue;
                        }

                        characterData.name = data[0];
                        characterData.age = age;
                        characterData.sex = (Sex)sex;
                        characterData.characterClass = (CharacterClass)characterClass;

                        _characters.Add(CreateCharacter(_curentID));
                        _curentID++;
                        PersoneRedactor.View();
                    }
                    _logger.WriteToLogFile("HANDLER (ReadDataFromCSV): Data from FileWithData.csv
was written to table");
                }
            }
            catch
            {
                _logger.WriteToLogFile("HANDLER (ReadDataFromCSV): FileWithData.csv was not read
or was empty");
                return;
            }
        }
        #endregion
        #region XxxxButton_Click
        private void CreateButton_Click(object sender, EventArgs e)
        {
            _logger.WriteToLogFile("BUTTON: CreateButton was clicked");
            PersoneRedactor.Create();
        }

        private void CopyButton_Click(object sender, EventArgs e)
        {
            _logger.WriteToLogFile("BUTTON: CopyButton was clicked");
            PersoneRedactor.Copy();
        }

        private void UpdateButton_Click(object sender, EventArgs e)
        {
            _logger.WriteToLogFile("BUTTON: UpdateButton was clicked");
            PersoneRedactor.Update();
        }

        private void DeleteButton_Click(object sender, EventArgs e)
        {
            _logger.WriteToLogFile("BUTTON: DeleteButton was clicked");
            PersoneRedactor.Delete();
        }
        #endregion
```

```
        #region Singleton
        private sealed class Logger
        {
            private static Logger _instance;

            private Logger()
            {
            }

            public static Logger GetInstance()
            {
                if (_instance == null)
                {
                    _instance = new Logger();
                }
                return _instance;
            }

            public void WriteToLogFile(string message)
            {
                using (StreamWriter writer = new StreamWriter(_pathForWrite, true,
Encoding.ASCII))
                {
                    writer.WriteLine(message);
                }
            }
        }
        #endregion
        #region Prototype
        private abstract class Character
        {
            public uint _id { get; protected set; }
            public CharacterClass _characterClass { get; protected set; }
            public string _ability { get; protected set; }
            public string _name { get; private set; }
            public int _age { get; private set; }
            public Sex _sex { get; private set; }

            public Character(string name, int age, Sex sex)
            {
                _name = name;
                _age = age;
                _sex = sex;
                _logger.WriteToLogFile("CONSTRUCTROR: CHARACTER constructor was called");
            }

            public virtual Character Clone()
            {
                Character tempCharacter = (Character)this.MemberwiseClone();
                tempCharacter._id = _curentID;

                return tempCharacter;
            }
        }

        private class Knight : Character
        {
            public Knight(uint id, string name, int age, Sex sex) : base(name, age, sex)
            {
                _id = id;
                _ability = "Sword strike";
                _characterClass = CharacterClass.Knight;
```

```
            _logger.WriteToLogFile("CONSTRUCTROR: KNIGHT constructor was called");
        }
    }

    private class Archer : Character
    {
        public Archer(uint id, string name, int age, Sex sex) : base(name, age, sex)
        {
            _id = id;
            _ability = "Archery";
            _characterClass = CharacterClass.Archer;
            _logger.WriteToLogFile("CONSTRUCTROR: ARCHER constructor was called");
        }
    }

    private class Wizard : Character
    {
        public Wizard(uint id, string name, int age, Sex sex) : base(name, age, sex)
        {
            _id = id;
            _ability = "Fireball";
            _characterClass = CharacterClass.Wizard;
            _logger.WriteToLogFile("CONSTRUCTROR: WIZARD constructor was called");
        }
    }

    private class Healer : Character
    {
        public Healer(uint id, string name, int age, Sex sex) : base(name, age, sex)
        {
            _id = id;
            _ability = "Healing";
            _characterClass = CharacterClass.Healer;
            _logger.WriteToLogFile("CONSTRUCTROR: HEALER constructor was called");
        }
    }

    private class Craftsman : Character
    {
        public Craftsman(uint id, string name, int age, Sex sex) : base(name, age, sex)
        {
            _id = id;
            _ability = "Repair";
            _characterClass = CharacterClass.Craftsman;
            _logger.WriteToLogFile("CONSTRUCTROR: CRAFTSMAN constructor was called");
        }
    }

    private class Thief : Character
    {
        public Thief(uint id, string name, int age, Sex sex) : base(name, age, sex)
        {
            _id = id;
            _ability = "Theft";
            _characterClass = CharacterClass.Thief;
            _logger.WriteToLogFile("CONSTRUCTROR: THIEF constructor was called");
        }
    }
    #endregion

    #region Facade
    private class PersoneRedactor
```

```
        {
            public static void Create()
            {
                characterData.characterClass = 0;
                _logger.WriteToLogFile("METHOD (PersoneRedactor.Create()): Method CREATE from
class PERSONREDACTOR was called");

                OpenCreateForm(Mode.Create);

                Character tempCharacter = CreateCharacter(_curentID);
                if (tempCharacter != null)
                {
                    _characters.Add(CreateCharacter(_curentID));
                }

                _curentID++;

                View();
            }

            public static void Copy()
            {
                _logger.WriteToLogFile("METHOD (PersoneRedactor.Copy()): Method COPY from class
PERSONREDACTOR was called");

                int index;

                try
                {
                    if (_charactersTable.CurrentCell == null)
                    {
                        _logger.WriteToLogFile("METHOD (PersoneRedactor.Copy()): There was no row
to copy in the table");
                        throw new Exception("There is no suitable row in the table to copy");
                    }
                    else
                    {
                        index = _charactersTable.CurrentCell.RowIndex;
                    }

                    if (index != -1 && _characters.Count > index)
                    {
                        _characters.Add(_characters[index].Clone());
                    }
                    else
                    {
                        _logger.WriteToLogFile("METHOD (PersoneRedactor.Copy()): Line to copy was
not selected");
                        MessageBox.Show("Select one of the filled lines to copy it");
                    }
                }
                catch (Exception e)
                {
                    MessageBox.Show(e.Message);
                }

                _curentID++;

                View();
            }

            public static void Update()
```

```
            {
                _logger.WriteToLogFile("METHOD (PersoneRedactor.Update()): Method UPDATE from
class PERSONREDACTOR was called");

                int index;

                try
                {
                    if (_charactersTable.CurrentCell == null)
                    {
                        _logger.WriteToLogFile("METHOD (PersoneRedactor.Update()): There was no
row to update in the table");
                        throw new Exception("There is no suitable row in the table to update");
                    }
                    else
                    {
                        index = _charactersTable.CurrentCell.RowIndex;
                    }

                    if (index != -1 && _characters.Count > index)
                    {
                        characterData.name = _characters[index]._name;
                        characterData.age = _characters[index]._age;
                        characterData.sex = _characters[index]._sex;
                        characterData.characterClass = _characters[index]._characterClass;

                        OpenCreateForm(Mode.Update);
                        _characters[index] = CreateCharacter(_characters[index]._id);
                    }
                    else
                    {
                        _logger.WriteToLogFile("METHOD (PersoneRedactor.Update()): Line to update
was not selected");
                        MessageBox.Show("Select one of the filled lines to update it");
                    }
                }
                catch (Exception e)
                {
                    MessageBox.Show(e.Message);
                }

                View();
            }

            public static void View()
            {
                _logger.WriteToLogFile("METHOD (PersoneRedactor.View()): Method VIEW from class
PERSONREDACTOR was called");

                _charactersTable.Rows.Clear();

                foreach (Character character in _characters)
                {
                    _charactersTable.Rows.Add(character._id, character._sex, character._name,
character._ability, character._age);
                }

            }
            public static void Delete()
            {
                _logger.WriteToLogFile("METHOD (PersoneRedactor.Delete()): Method DELETE from
class PERSONREDACTOR was called");
```

```
                int index;

                try
                {
                    if (_charactersTable.CurrentCell == null)
                    {
                        _logger.WriteToLogFile("METHOD (PersoneRedactor.Delete()): There was no
row to delete in the table");
                        throw new Exception("There is no suitable row in the table to delete");
                    }
                    else
                    {
                        index = _charactersTable.CurrentCell.RowIndex;
                    }

                    if (index != -1 && _characters.Count > index)
                    {
                        _characters.RemoveAt(index);
                    }
                    else
                    {
                        _logger.WriteToLogFile("METHOD (PersoneRedactor.Delete()): Line to delete
was not selected");
                        MessageBox.Show("Select one of the filled lines to delete it");
                    }
                }
                catch (Exception e)
                {
                    MessageBox.Show(e.Message);
                }

                View();
            }
        }

        #endregion
        private void MainForm_FormClosed(object sender, FormClosedEventArgs e)
        {
            _logger.WriteToLogFile("===============Programm was closed===============");

            string record;

            using (StreamWriter writer = new StreamWriter(_pathForRead, false, Encoding.UTF8))
            {
                foreach (Character character in _characters)
                {
                    record = character._name.ToString() + ", " + character._age.ToString()
                        + ", " + ((int)character._sex).ToString() + ", " +
((int)character._characterClass).ToString() + ".";

                    writer.WriteLine(record);
                }
            }
        }
    }
}
```

CreateCharacterForm.cs

```
using System;
using System.Windows.Forms;
using static TRPO_LR1.MainForm;

namespace TRPO_LR1
{
    public partial class CreateCharacterForm : Form
    {
        #region Variables
        private MainForm _mainForm;
        #endregion

        #region Constructors
        public CreateCharacterForm(MainForm mainForm, Mode mode)
        {
            _mainForm = mainForm;
            InitializeComponent();

            SexComboBox.Items.AddRange(new string[] { "Male", "Female" });
            ClassComboBox.Items.AddRange(new string[] { "Knight", "Archer", "Wizard", "Healer",
"Craftsman", "Thief" });

            if (mode == Mode.Update)
            {
                SetUpdateMode();
            }
        }
        #endregion

        #region Handler
        private void SetUpdateMode()
        {
            CreateButton.Text = "Update";
            NameTextBox.Text = characterData.name;
            SexComboBox.SelectedIndex = (int)(characterData.sex - 1);
            ClassComboBox.SelectedIndex = (int)(characterData.characterClass - 1);
            AgeTrackBar.Value = characterData.age;
        }

        private bool CheckData()
        {
            return NameTextBox.Text != null && NameTextBox.Text != "" &&
SexComboBox.SelectedIndex != -1 && ClassComboBox.SelectedIndex != -1;
        }
        #endregion

        #region XxxxButton_Click
        private void CreateButton_Click(object sender, EventArgs e)
        {
            characterData.name = NameTextBox.Text;
            characterData.sex = (Sex)(SexComboBox.SelectedIndex + 1);
            characterData.characterClass = (CharacterClass)(ClassComboBox.SelectedIndex + 1);
            characterData.age = AgeTrackBar.Value;
            _mainForm.Enabled = true;
            Close();
        }
        private void AgeTrackBar_Scroll(object sender, EventArgs e)
        {
            AgeValueLabel.Text = AgeTrackBar.Value.ToString();
        }
```

```
        private void NameTextBox_TextChanged(object sender, EventArgs e)
        {
            CreateButton.Enabled = CheckData();
        }

        private void SexComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            CreateButton.Enabled = CheckData();
        }

        private void ClassComboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            CreateButton.Enabled = CheckData();
        }
        #endregion

        private void CreateCharacterForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            _mainForm.Enabled = true;
        }
    }
}
```