Тема 10. Работа с локальными файлами в Windows Forms.

Учебные вопросы:

1. Понятие сериализации и десериализации.
2. Формат хранения данных JSON.
3. Работа с JSON через System.Text.Json.
4. Практика в WinForms. Сохранение данных из TextBox, DataGridView, ListBox.

1. Понятие сериализации и десериализации.

Сериализация (Serialization) — это процесс преобразования состояния объекта (его свойств и полей) в поток байтов или текстовую строку (например, в формат JSON или XML).

Десериализация (Deserialization) — это обратный процесс: восстановление объекта в оперативной памяти из сохраненного потока байтов или строки.

Основная мысль:

- Сериализация — это "упаковка" объектов в формат для хранения или передачи, а десериализация — "распаковка" обратно в объекты.

Простая аналогия: чемодан в аэропорту.

Вы летите в отпуск и сдаёте чемодан в багаж:

- У вас есть ВЕЩИ в комнате (объекты в программе)
- Вы УПАКОВЫВАЕТЕ их в чемодан (сериализация)
- Чемодан ЕДЕТ в самолёте (хранение/передача)
- Вы ПРИЕЗЖАЕТЕ и ДОСТАЁТЕ вещи (десериализация)
- У вас снова ВЕЩИ в номере отеля (объекты в программе)

Ключевой момент: Вещи не меняются, меняется только их форма хранения!

Что такое сериализация на пальцах?

Представьте: у вас есть объект "Студент" в программе

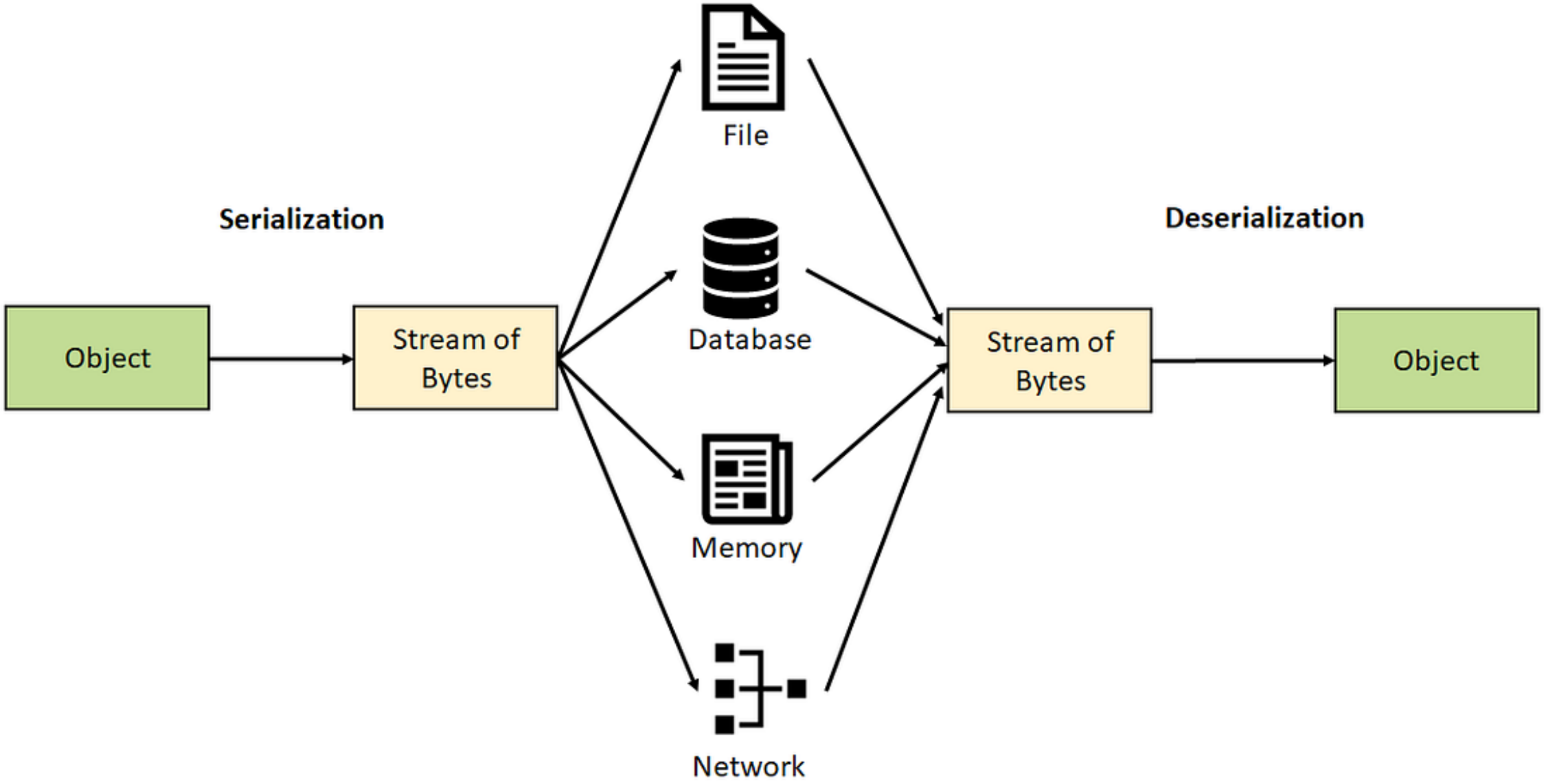
```
// В памяти компьютера это объект с полями:
Student student = new Student();
student.Name = "Иван";
student.Age = 20;
student.Grades = new List<int> { 5, 4, 5 };
```

Проблема: Как сохранить этот объект в файл?

- Файл — это просто текст или байты
- Объект в памяти — сложная структура со ссылками, методами

Решение — сериализация:

Объект в памяти → Преобразуем в текст → Сохраняем в файл


Serialization
Deserialization
Object
Stream of Bytes
File
Database
Memory
Network
Stream of Bytes
Object

Наглядный пример: без сериализации vs с сериализацией

❌ БЕЗ сериализации (проблемный способ):

```
// Пытаемся сохранить объект "напрямую"
Student student = GetStudent();


// В файл можно писать только строки или байты
// А у нас объект! Что делать?
File.WriteAllText("student.txt", student); // ОШИБКА!
```

✅ С сериализацией (правильный способ):

```
Student student = GetStudent();

// 1. Сериализуем (преобразуем объект в строку)
string json = JsonSerializer.Serialize(student);
// json = "{"Name":"Иван","Age":20,"Grades":[5,4,5]}"

// 2. Сохраняем строку в файл
File.WriteAllText("student.json", json); // УСПЕХ!
```

Зачем это нужно в Windows Forms?

- Сохранение состояния (Persistence): Пользователь закрыл программу, открыл её завтра — и все настройки, введенные тексты и списки остались на месте.
- Обмен данными: Если ваша программа должна отправить данные на сервер или в другое приложение, она не может отправить «кусок памяти». Она отправляет сериализованную строку (JSON).
- Клонирование объектов: Иногда проще сериализовать объект и тут же десериализовать его в новую переменную, чтобы получить точную копию.

Как это выглядит схематично:

- Объект в памяти: User { Name = "Ivan", Age = 20 } (Существует только пока запущена программа).
- Сериализация: Превращаем в строку {"Name":"Ivan", "Age":20}.
- Хранение: Записываем эту строку в файл user.json на диск.
- Десериализация: Читаем файл, создаем новый объект User и заполняем его данными из строки.

Важное замечание: Сериализуются только данные (свойства и поля). Методы (логика, функции) класса не сериализуются, так как они уже описаны в самом коде программы.

Что можно сериализовать:

- Простые типы
- Классы и структуры (с публичными свойствами)
- Списки и массивы
- Объекты

Основные форматы сериализации:

- JSON (JavaScript Object Notation) — самый популярный
- XML (eXtensible Markup Language)
- Бинарная сериализация. Не читаемый текст, а байты:
- Текстовые файлы (CSV, простой текст)

2. Формат хранения данных JSON.

JSON (JavaScript Object Notation) — это открытый стандарт текстового формата обмена данными.

Несмотря на название, он не зависит от языка программирования и может использоваться в C#, Java, Python и многих других.

На сегодняшний день это самый популярный формат в мире IT.

Основные правила структуры JSON:

- Данные хранятся в парах «ключ : значение».
- Объекты заключаются в фигурные скобки { }.
- Массивы (списки) заключаются в квадратные скобки [ ].
- Ключ всегда пишется в двойных кавычках: "Name".
- Значения могут быть:
  - Строкой (в кавычках): "Иван"
  - Числом (без кавычек): 25
  - Логическим значением: true / false
  - Пустым значением: null
  - Другим объектом или массивом.

Пример:

```
{
  "студент": {
    "имя": "Иван Петров",
    "возраст": 20,
    "студент": true,
    "средний_балл": 4.5,
    "посещаемость": null,
    "курсы": ["C#", "Базы данных", "Алгоритмы"],
    "контакты": {
      "email": "ivan@mail.ru",
      "телефон": "+7 999 123-45-67"
    }
  }
}
```

Важные правила:

- Все ключи в двойных кавычках — "ключ"
- Строки в двойных кавычках — "значение"
- Числа, true, false, null без кавычек
- Запятые между элементами, но не после последнего
- Фигурные скобки для объектов {}
- Квадратные скобки для массивов []

Сравнение: Объект C# vs JSON

Представим, что у нас есть класс в C#:

```
public class Student
{
    Ссылок: 1
    public string Name { get; set; } = "Алексей";
    Ссылок: 1
    public int Age { get; set; } = 20;
    Ссылок: 0
    public bool IsActive { get; set; } = true;
}
```

В формате JSON этот же объект будет выглядеть так:

```
{
    "Name": "Алексей",
    "Age": 20,
    "IsActive": true
}
```

Итог:

- JSON — это текстовый формат для хранения и передачи данных
- Основа — пары "ключ": "значение"
- Ключи всегда в двойных кавычках
- Значения бывают: строка, число, true/false, null, массив, объект
- Используется ВСЕМИ современными языками программирования
- Идеально подходит для: настроек, конфигураций, обмена данными

3. Работа с JSON через System.Text.Json.

System.Text.Json — это встроенная в .NET библиотека, которая:

- Преобразует объекты C# в JSON (сериализация)
- Преобразует JSON обратно в объекты C# (десериализация)
- Работает быстро и не требует установки доп. библиотек

Пример:

Нужно обавить пространство имён. В начале файла (под using System;)

using System.Text.Json;

```
// Простой класс для примера
Ссылок: 2
public class Person
{
    Ссылок: 1
    public string Name { get; set; }
    Ссылок: 1
    public int Age { get; set; }
    Ссылок: 1
    public bool IsStudent { get; set; }
}
```

Сериализация:

```
Person person = new Person
{
    Name = "Nikita",
    Age = 20,
    IsStudent = true
};
// Сериализуем в JSON
string personJson = JsonSerializer.Serialize(person);

Console.WriteLine(personJson);
                // Результат:
// {"Name":"Nikita","Age":20,"IsStudent":true}
```

Десериализация:

```
// Есть JSON строка
string jsonString = "{\"Name\":\"Мария\",\"Age\":22,\"IsStudent\":false}";

// Десериализуем в объект
Person loadedPerson = JsonSerializer.Deserialize<Person>(jsonString);

// Теперь можем использовать:
Console.WriteLine(loadedPerson.Name);     // Мария
Console.WriteLine(loadedPerson.Age);      // 22
```

Пояснение:

- JsonSerializer — это класс, который предоставляет методы для сериализации и десериализации объектов в JSON и наоборот.
- Deserialize<Person>(jsonString) — метод десериализует строку JSON обратно в объект типа Person.
- loadedPerson — это переменная, которая будет содержать результат десериализации. В этот объект будут помещены значения полей, соответствующие структуре Person.
- Для успешной десериализации строка JSON должна соответствовать структуре класса Person.

По умолчанию JSON получается «сжатым» (в одну строку) и не всегда корректно отображает кириллицу. Для исправления этого используются настройки:

```
var options = new JsonSerializerOptions
{
    WriteIndented = true, // Красивые отступы (лесенкой)
    // Правильное отображение кирилицы
    Encoder = System.Text.Encodings.Web.JavaScriptEncoder.UnsafeRelaxedJsonEscaping
};
// Сериализуем в JSON
string personJson = JsonSerializer.Serialize(person, options);
```

```
{
  "Name": "Руслан",
  "Age": 20,
  "IsStudent": true
}
```

Работа со списками

Библиотека легко справляется с массивами и списками. Это именно то, что нужно для сохранения данных из DataGridView или ListBox.

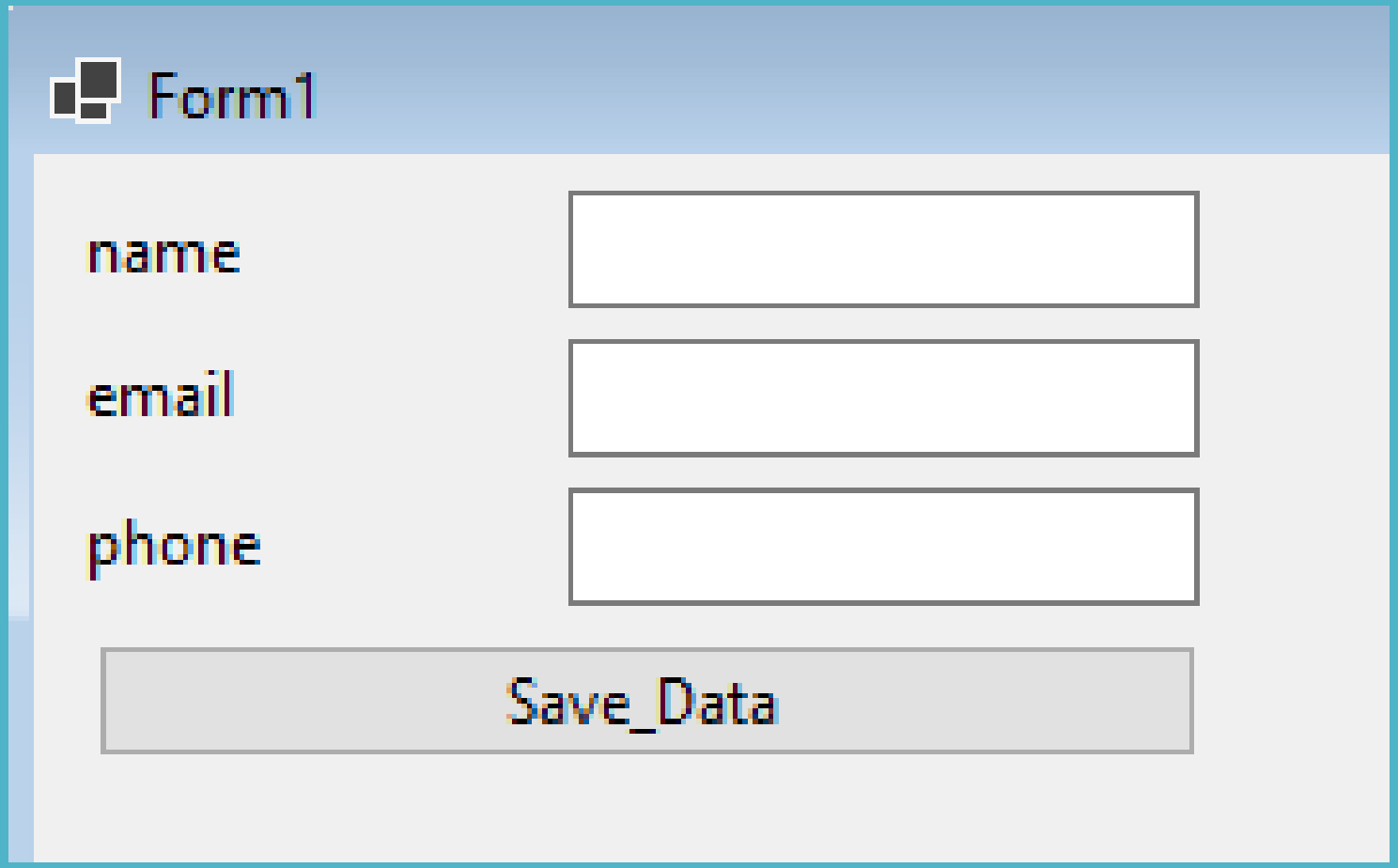
```
List<Person> people = new List<Person>();
// ... добавление данных в список ...
people.Add(person1);
people.Add(person2);

// Сериализуется весь список сразу в массив JSON [{}, {}]
string jsonList = JsonSerializer.Serialize(people, options);
```

4. Практика в WinForms. Сохранение данных из TextBox, DataGridView, ListBox.

Чтобы сохранить данные из интерфейса, мы должны сначала «собрать» их в объект или список объектов (List), а затем сохранить этот список.

Мы никогда не сериализуем сами визуальные элементы (контролы), мы сериализуем данные, которые в них лежат.

Сохранение данных из TextBox



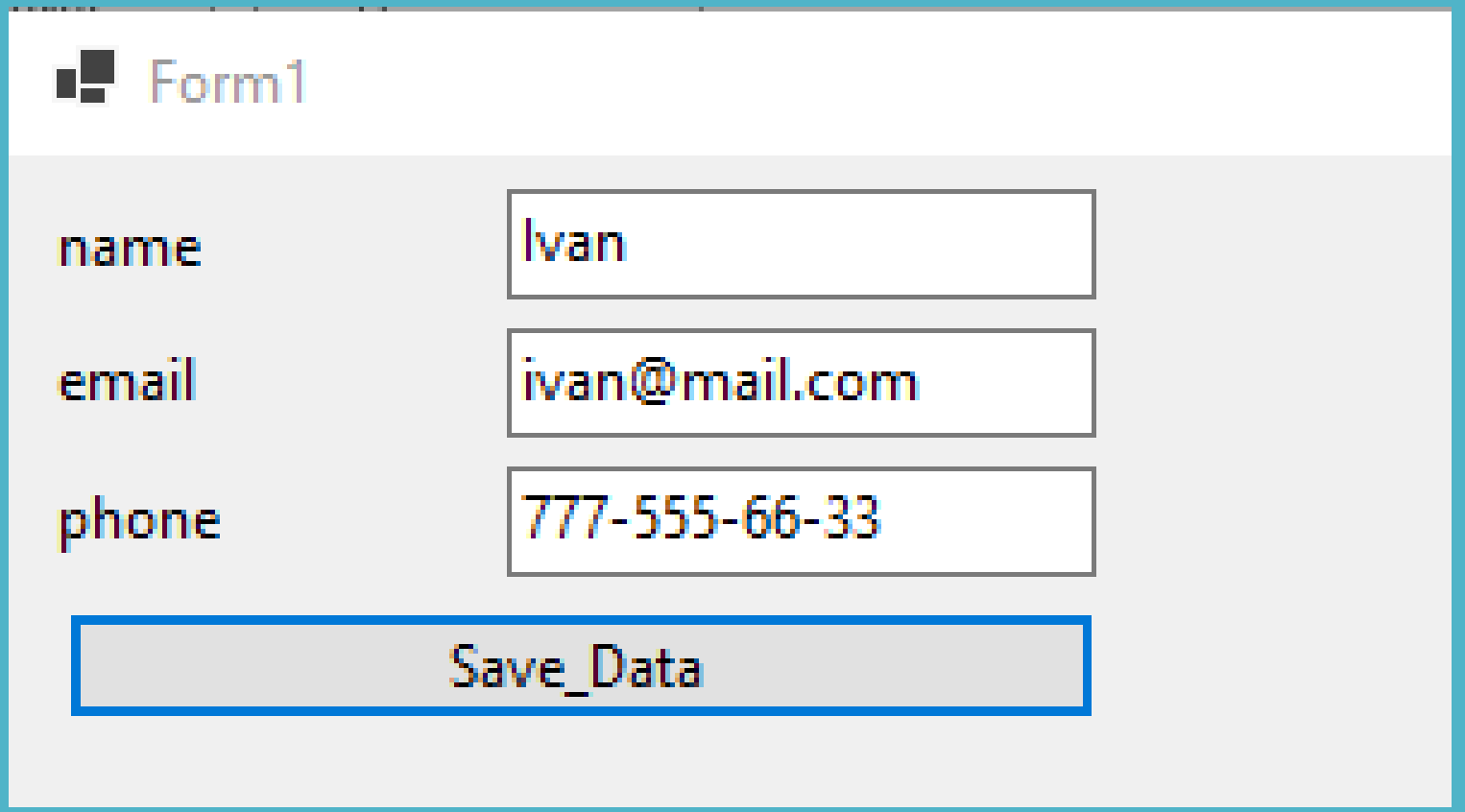
Класс для хранения данных:

```
// Класс для хранения данных
Ссылок: 0
public class UserProfile
{
    Ссылок: 0
    public string Name { get; set; }
    Ссылок: 0
    public string Email { get; set; }
    Ссылок: 0
    public string Phone { get; set; }
}
```

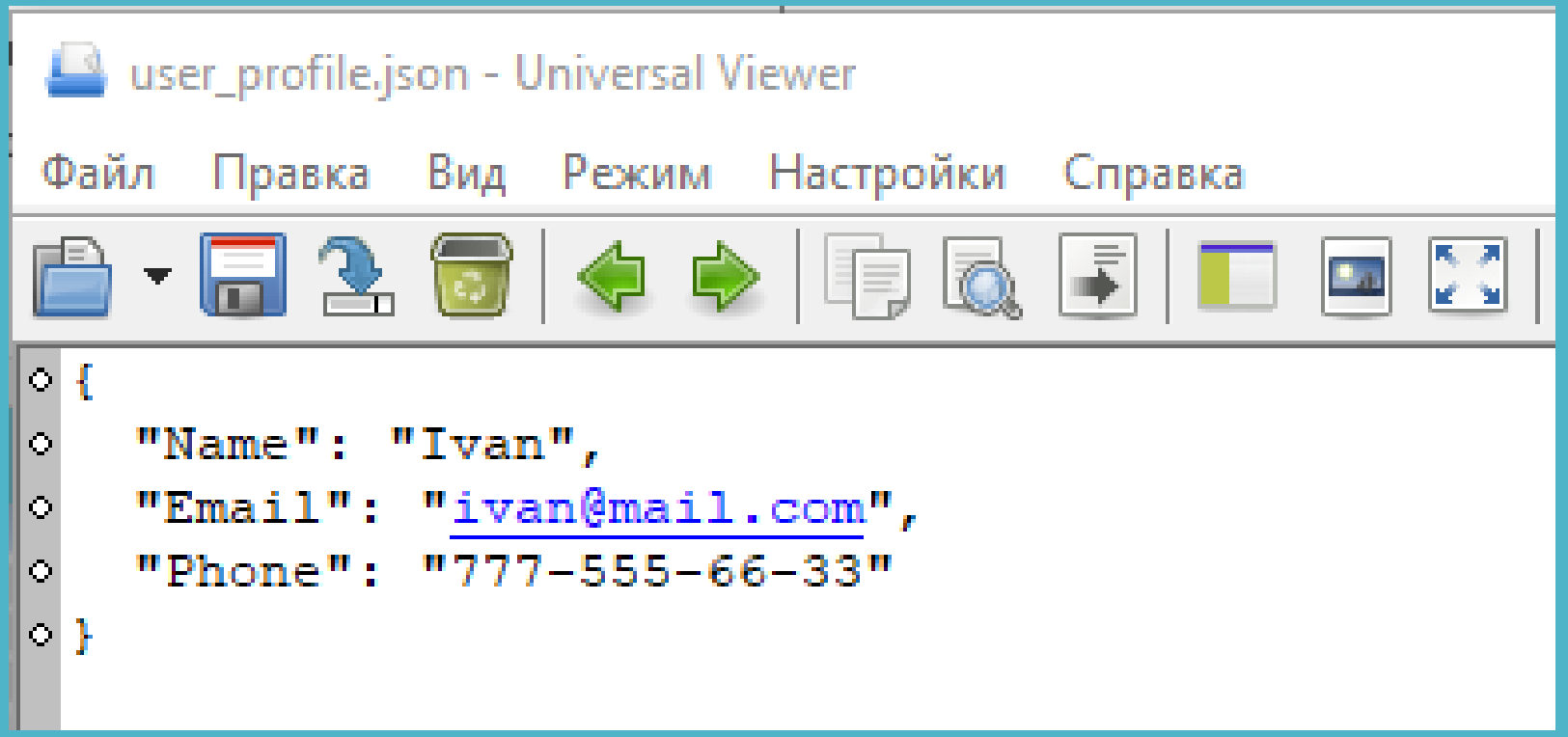
Обработчик кнопки:

```
private void btnSaveData_Click(object sender, EventArgs e)
{
    // 1. Создаём объект из TextBox'ов
    var profile = new UserProfile
    {
        Name = textBoxName.Text,
        Email = textBoxEmail.Text,
        Phone = textBoxPhone.Text,
    };

    // 2. Сериализуем в JSON
    string json = JsonSerializer.Serialize(profile,
        new JsonSerializerOptions { WriteIndented = true });

    // 3. Сохраняем в файл
    File.WriteAllText("user_profile.json", json);

    MessageBox.Show("Данные сохранены!");
}
```

Проверяем:


Form1
name
Ivan
email
ivan@mail.com
phone
777-555-66-33
Save_Data

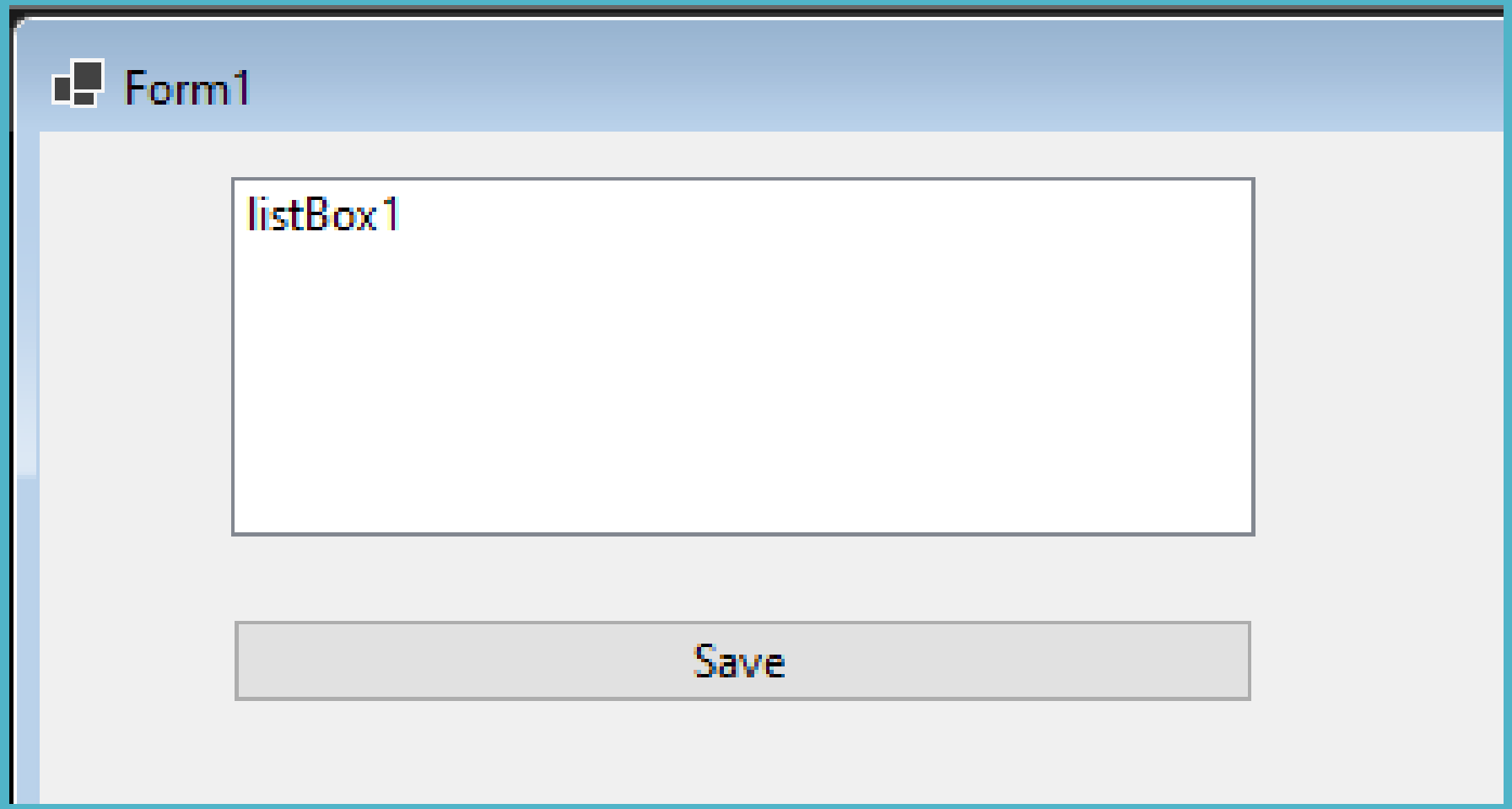
Проверяем:


user_profile.json - Universal Viewer
Файл Правка Вид Режим Настройки Справка
{
"Name": "Ivan",
"Email": "ivan@mail.com",
"Phone": "777-555-66-33"
}

Обработчик для кнопки «Загрузить»:

```
private void btnLoad_Click(object sender, EventArgs e)
{
    if (!File.Exists("user_profile.json"))
    {
        MessageBox.Show("Файл с данными не найден");
        return;
    }

    // 1. Читаем JSON из файла
    string json = File.ReadAllText("user_profile.json");

    // 2. Десериализуем в объект
    var profile = JsonSerializer.Deserialize<UserProfile>(json);

    // 3. Заполняем TextBox'ы
    textBoxName.Text = profile.Name;
    textBoxEmail.Text = profile.Email;
    textBoxPhone.Text = profile.Phone;

    MessageBox.Show("Данные загружены!");
}
```

Сохранение данных из ListBox.
Пример. Добавим элементы на форму:



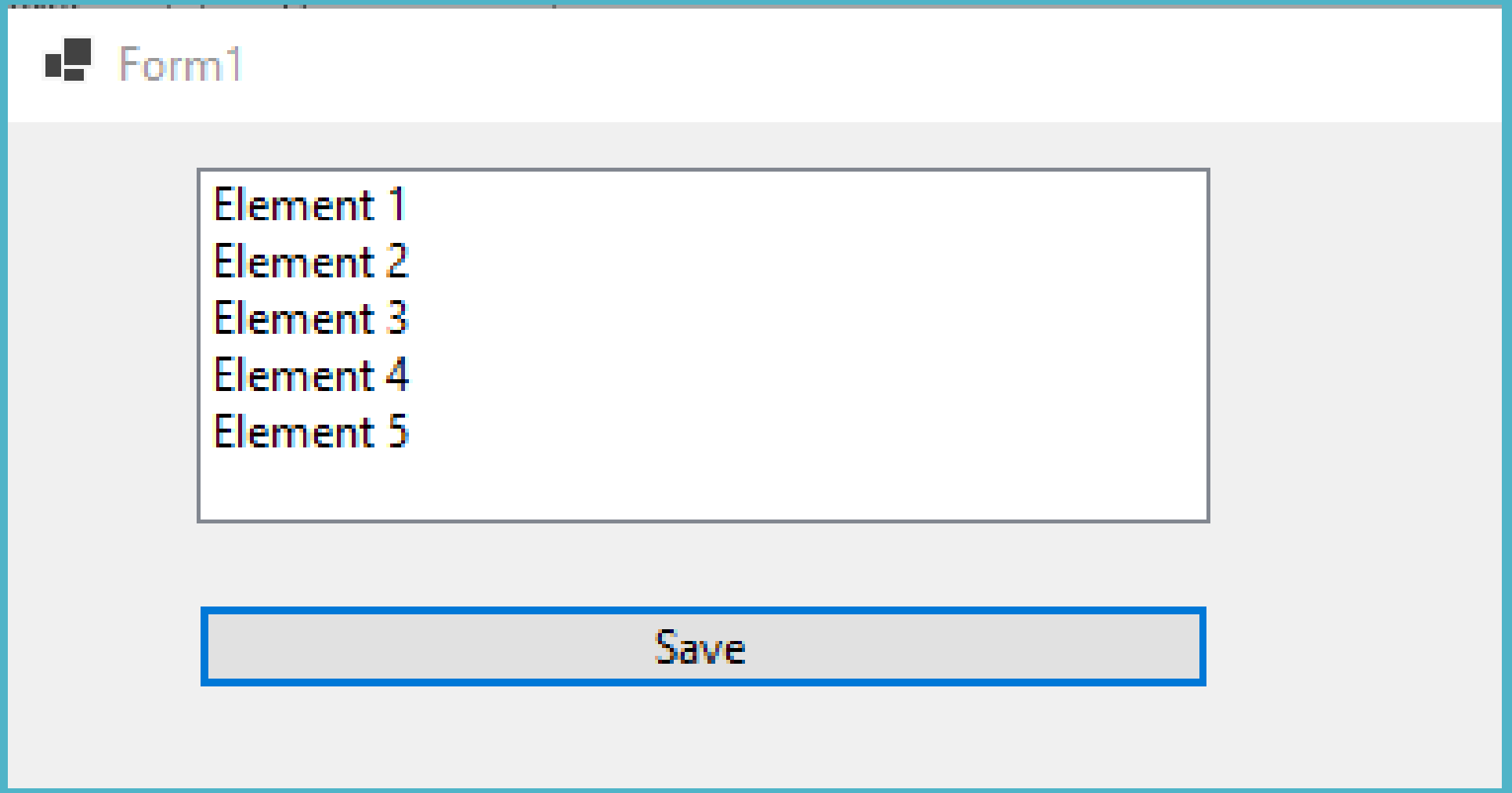
Добавим элементы в ListBox:

```
public Form1()
{
    InitializeComponent();
    // Добавление элементов в ListBox
    listBox1.Items.Add("Element 1");
    listBox1.Items.Add("Element 2");
    listBox1.Items.Add("Element 3");
    listBox1.Items.Add("Element 4");
    listBox1.Items.Add("Element 5");
}
```

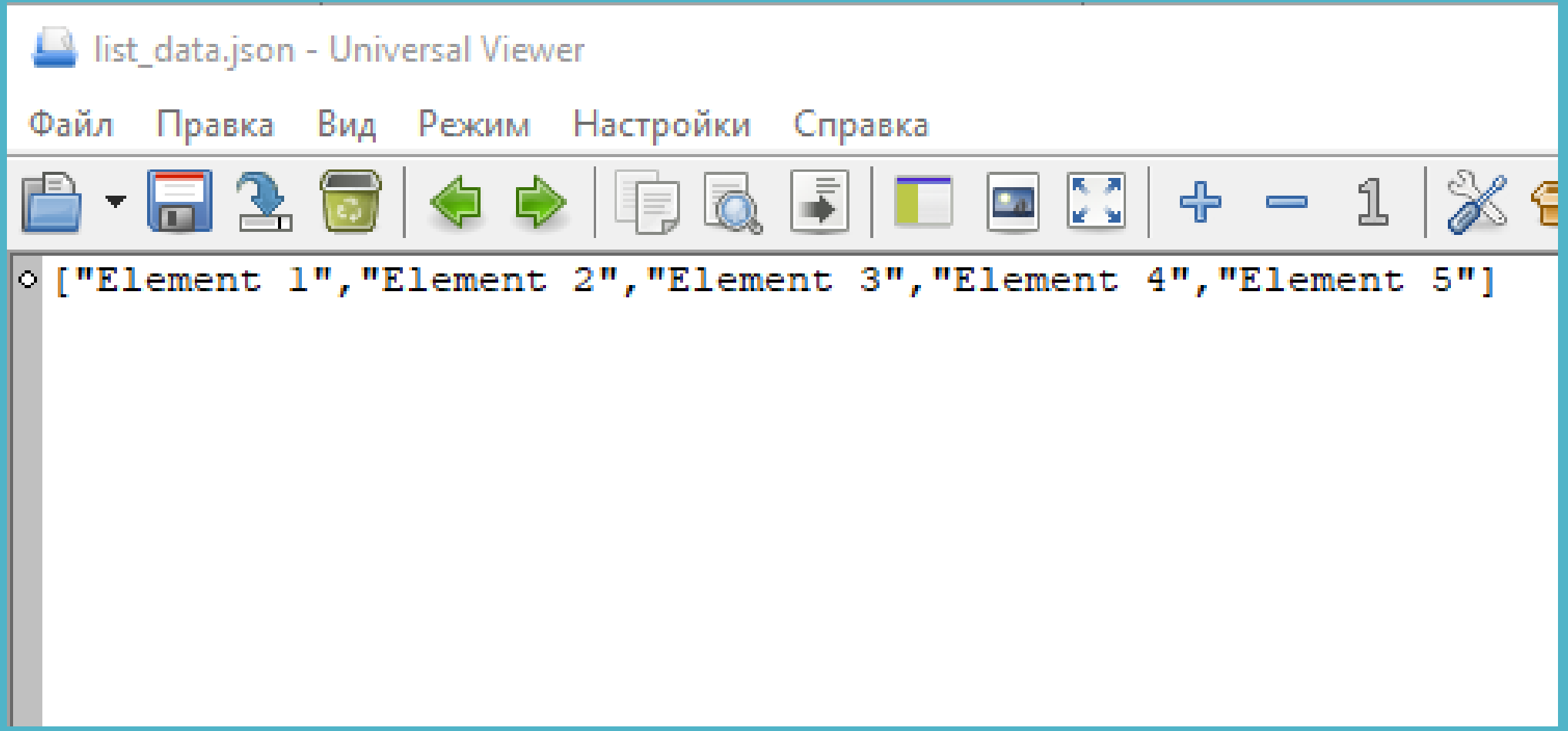
Обработчик для кнопки:

```
private void btnSave_Click(object sender, EventArgs e)
{
    List<string> items = new List<string>();

    // Собираем данные из ListBox в список
    foreach (var item in listBox1.Items)
    {
        items.Add(item.ToString());
    }

    // Сериализуем и сохраняем
    string json = JsonSerializer.Serialize(items);
    File.WriteAllText("list_data.json", json);
}
```

Проверяем:

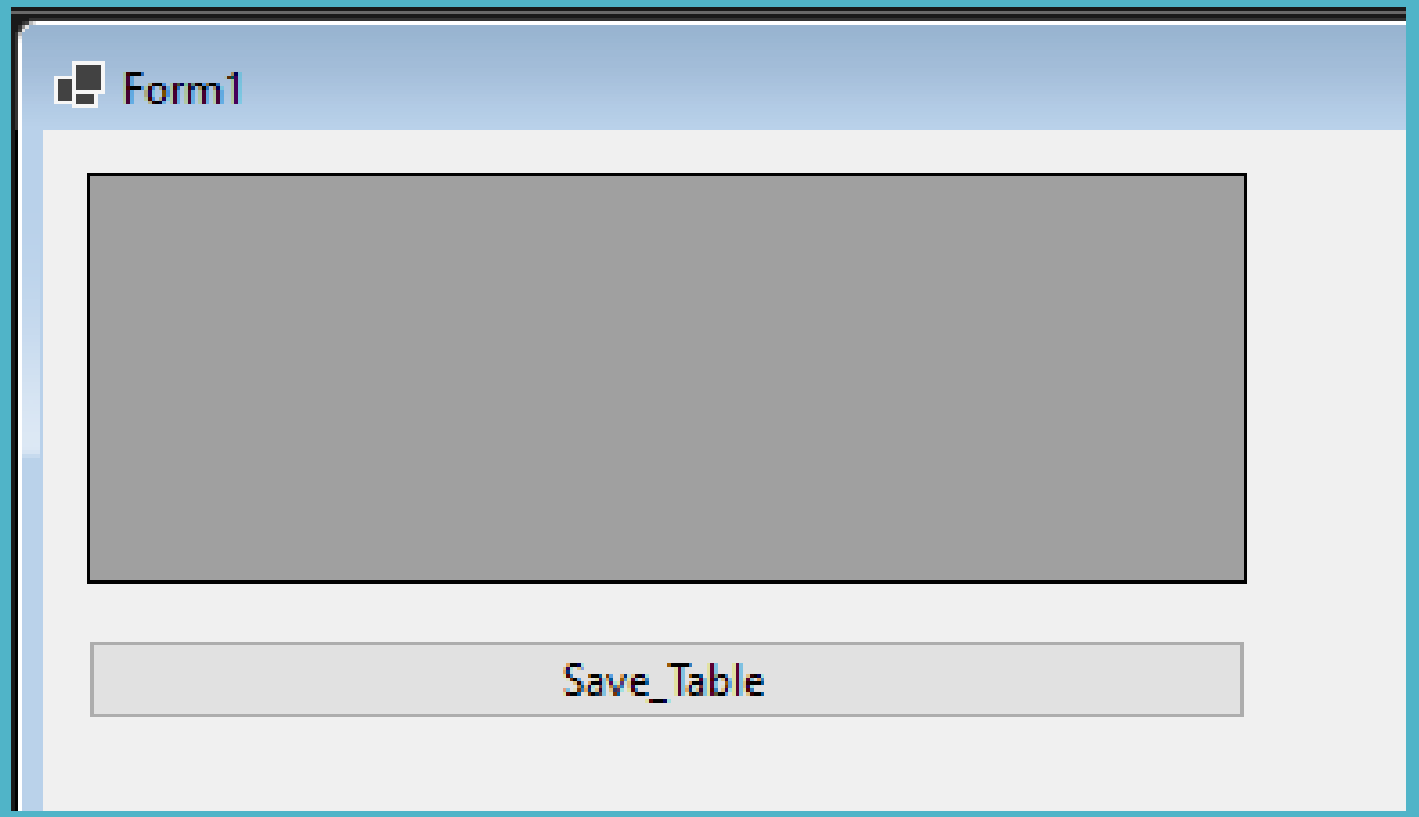


Список сохранен в файл:



Сохранение данных из DataGridView (Самый частый случай)

Добавим элементы на форму:



Создаем список и привязываем его к таблице:

```
public partial class Form1 : Form
{
    // Объявляем список здесь, чтобы он был доступен везде в этой форме
    BindingList<Student> students = new BindingList<Student>();

    Ссылок: 1
    public Form1()
    {
        InitializeComponent();
        LoadData();
```

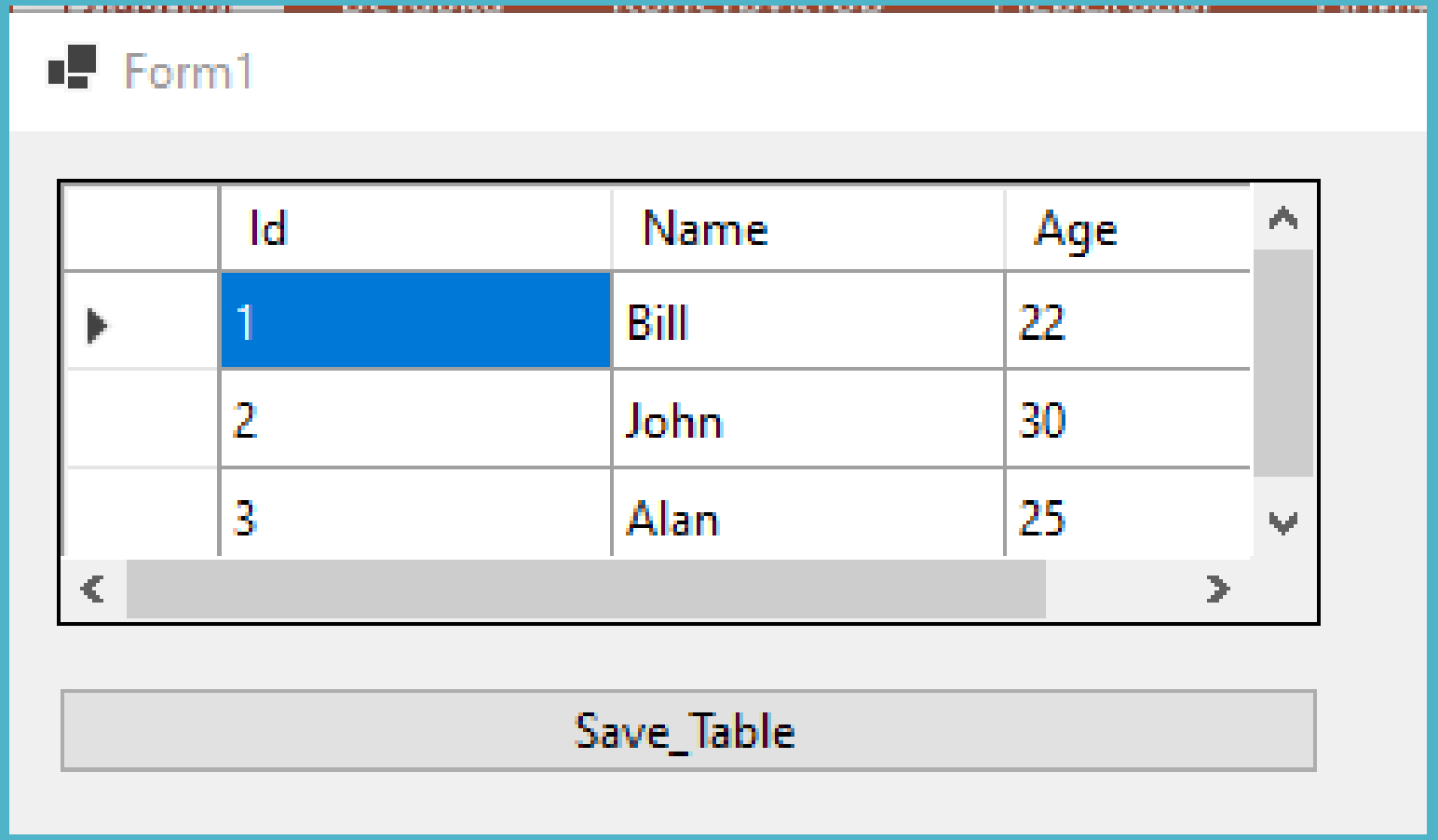
P.S: Вместо обычного List<Student> используйте BindingList<Student>. Этот класс специально создан для WinForms: как только вы добавите или удалите элемент из такого списка, таблица тут же перерисуется сама.

```
private void LoadData()
{
    // добавим объекты в список
    students.Add(new Student { Id = 1, Name = "Bill", Age = 22 });
    students.Add(new Student { Id = 2, Name = "John", Age = 30 });
    students.Add(new Student { Id = 3, Name = "Alan", Age = 25 });

    // Привязка списка к DataGridView
    dataGridView1.DataSource = students;
}
```

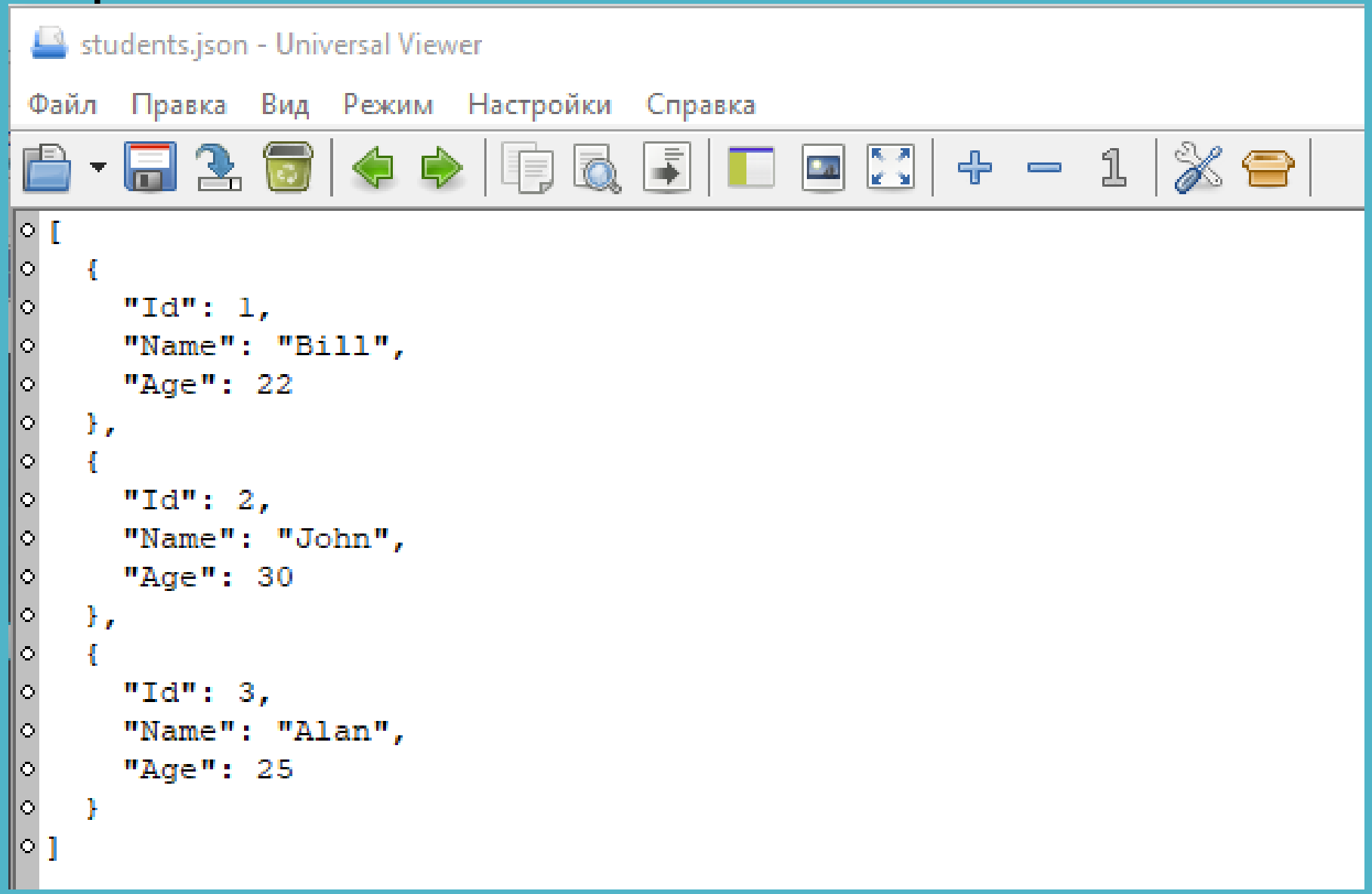
Обработчик нажатия кнопки:

```
Ссылок: 1
private void btnSaveTable_Click(object sender, EventArgs e)
{
    var options = new JsonSerializerOptions { WriteIndented = true };

    // Сериализуем весь список, который привязан к таблице
    string json = JsonSerializer.Serialize(students, options);
    File.WriteAllText("students.json", json);

    MessageBox.Show("Данные таблицы сохранены!");
}
```

Проверяем:

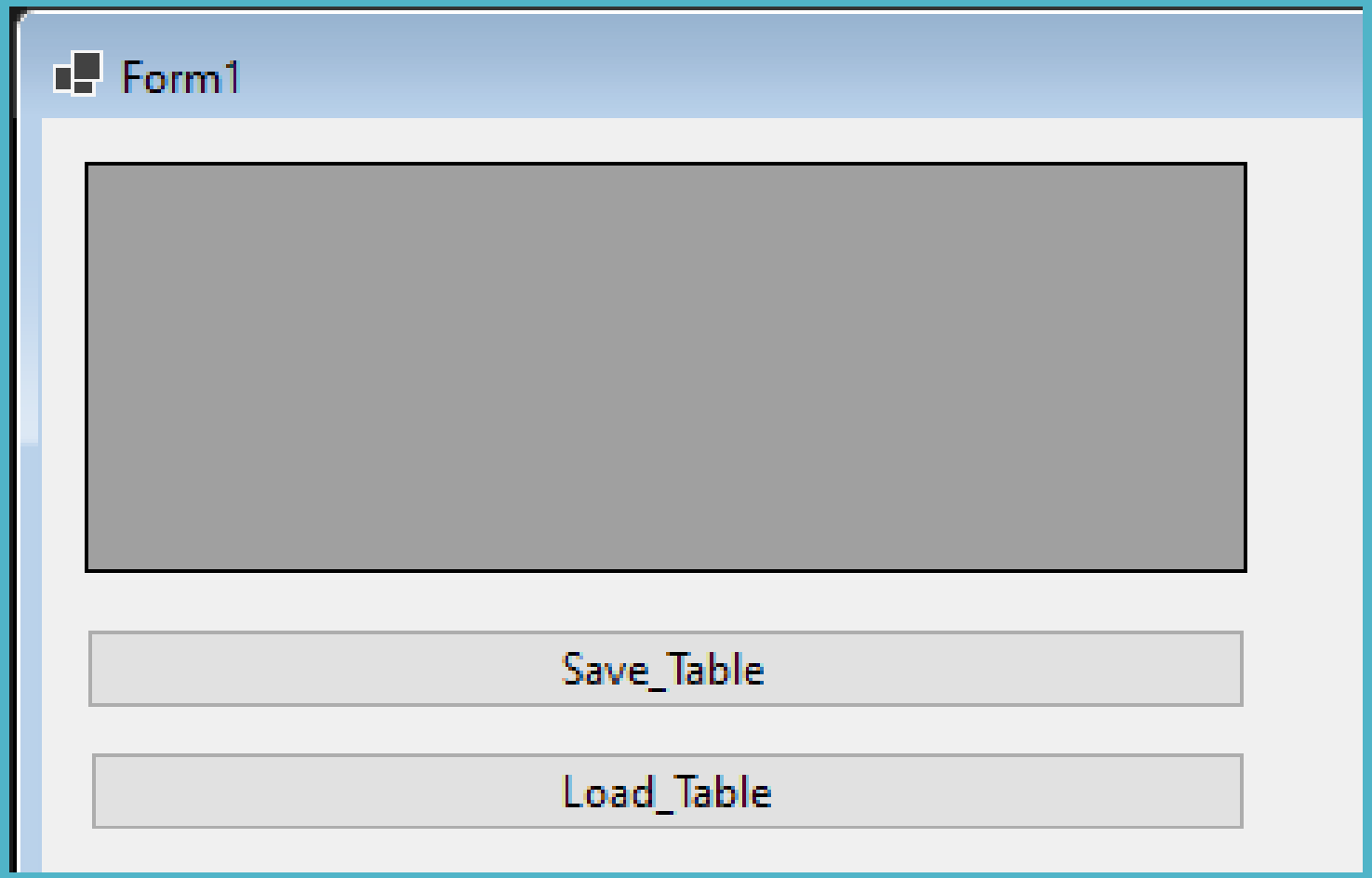


Проверяем:


students.json - Universal Viewer
Файл Правка Вид Режим Настройки Справка
[
{
"Id": 1,
"Name": "Bill",
"Age": 22
},
{
"Id": 2,
"Name": "John",
"Age": 30
},
{
"Id": 3,
"Name": "Alan",
"Age": 25
}
]

Загрузка данных в таблицу

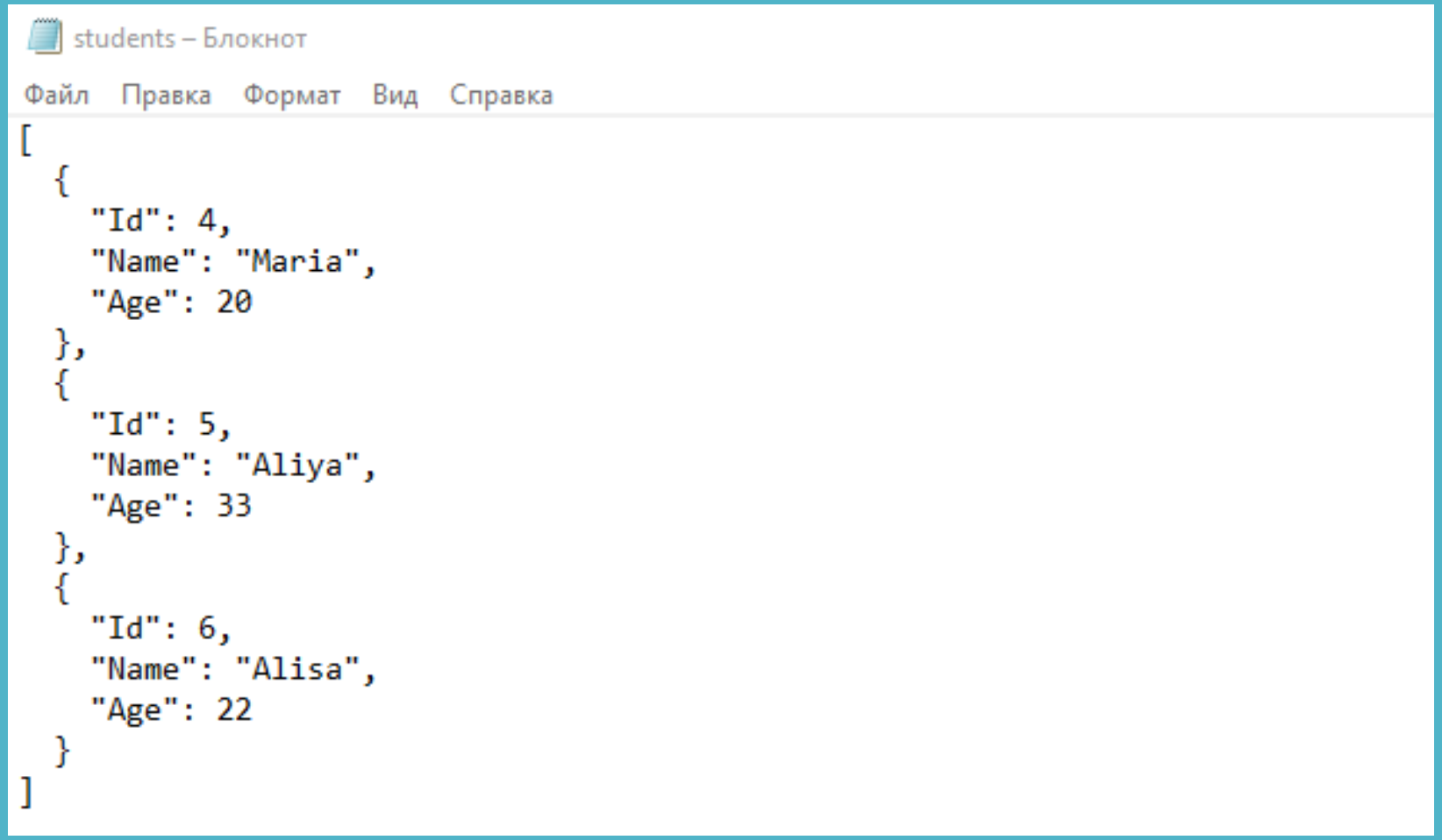
Добавим еще одну кнопку, для записи данных в DataGridView из файла:



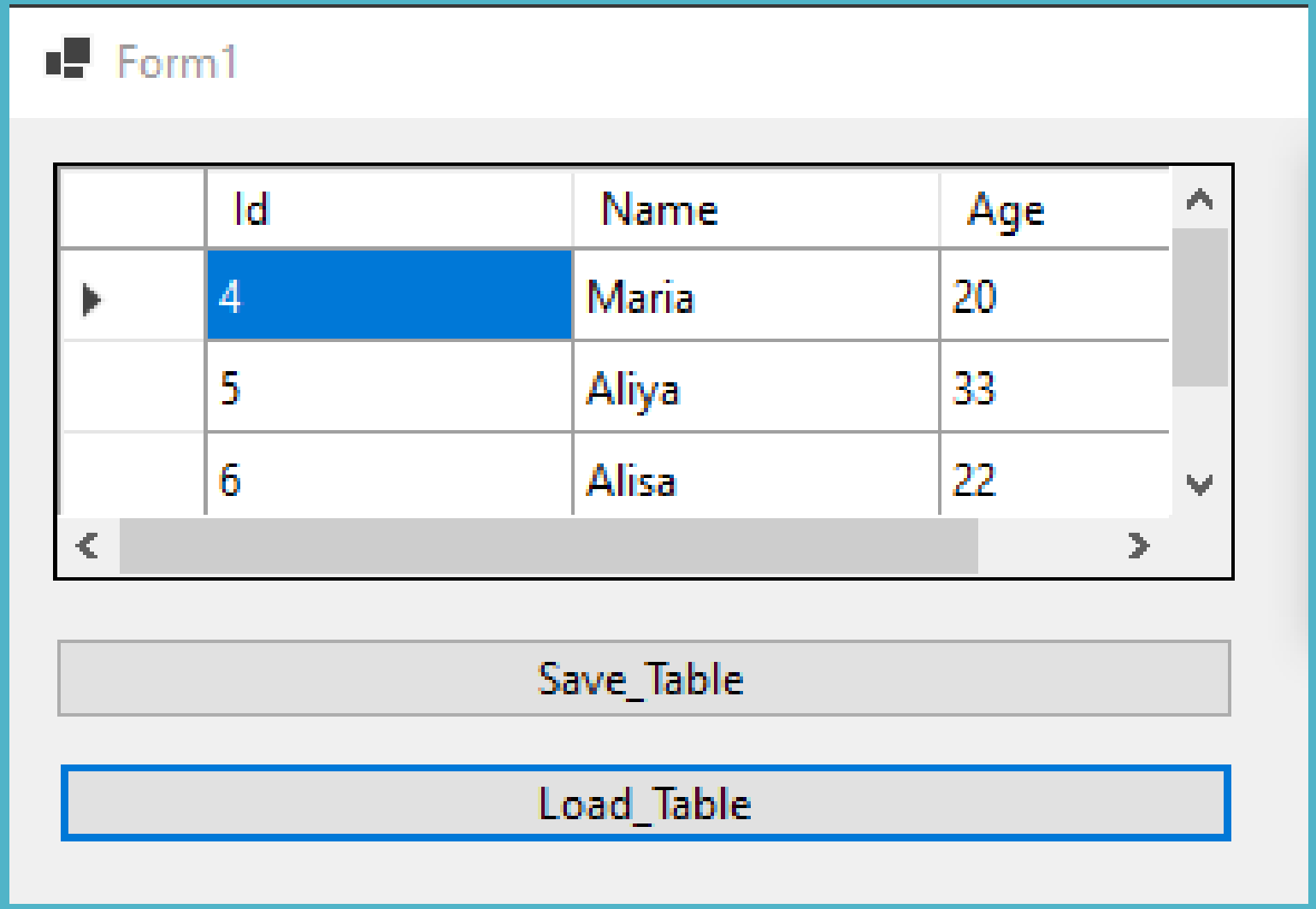
Обработчик:

```
private void btnLoadTable_Click(object sender, EventArgs e)
{
    if (File.Exists("students.json"))
    {
        string json = File.ReadAllText("students.json");

        // Десериализуем в обычный список
        var loadedList = JsonSerializer.Deserialize<List<Student>>(json);

        // Очищаем старые данные и добавляем новые
        students.Clear();
        foreach (var s in loadedList) students.Add(s);
        MessageBox.Show("Данные таблицы изменены!");
    }
}
```

Изменим json-файл и запишем новые данные:



Проверяем:


Form1
Id
Name
Age
4
Maria
20
5
Aliya
33
6
Alisa
22
Save_Table
Load_Table

Итоги лекции:

- Суть процесса: Сериализация — это перевод объекта в текст (JSON), десериализация — восстановление объекта из текста.
- Правило класса: Сохраняются только свойства с { get; set; }. Обычные поля (без get/set) библиотека System.Text.Json проигнорирует.
- Связь с UI: Мы сохраняем не ListBox или DataGridView, а связанные с ними списки объектов (List<T>).
- BindingList — лучший друг: Чтобы DataGridView сам обновлялся после загрузки данных из файла, вместо List<T> используйте BindingList<T>.
- Опции (Options): Для красивого вида (отступов) и поддержки русского языка используйте JsonSerializerOptions.
- Безопасность: Перед чтением файла всегда проверяй File.Exists(), иначе программа упадет при первом же запуске без данных.

Материалы лекций:

https://github.com/ShViktor72/Education2025