Лабораторная работа № 10

Тема: Сериализация объектов в JSON и работа с файлами

Задание:

Вариант №1: Приложение «Учет библиотечного фонда»

Задание 1: Модель данных
Создайте класс Book со свойствами:
Id (целое число)
Title (название, строка)
Author (автор, строка)
Year (год издания, целое число)

Обязательно используйте автоматические свойства { get; set; }.

Задание 2: Работа с таблицей (DataGridView)
Реализуйте форму, где пользователь может вводить данные о книге в TextBox и добавлять их в таблицу. Используйте BindingList<Book> для автоматического обновления DataGridView.

Задание 3: Сохранение и кодировка
Реализуйте кнопку «Сохранить библиотеку».
Данные должны сохраняться в файл library.json.
Используйте JsonSerializerOptions, чтобы JSON был «красивым» (отступы) и корректно отображал русские буквы.

Задание 4: Автозагрузка и проверка
Реализуйте чтение данных из файла при нажатии кнопки «Загрузить».
Добавьте проверку: если файла не существует, выведите сообщение «Файл данных не найден», не допуская вылета программы.

Вариант №2: Приложение «Менеджер задач (To-Do List)»

Задание 1: Модель данных
Создайте класс TaskItem со свойствами:
Description (описание задачи, строка)
Deadline (дата, DateTime)
IsCompleted (выполнено, логическое)
Priority (приоритет от 1 до 5, целое число)

Задание 2: Работа со списками
Реализуйте форму с DataGridView для отображения списка задач. Добавьте возможность удаления выбранной строки из таблицы (и, соответственно, из списка объектов).

Задание 3: Сериализация списка
Реализуйте кнопку «Экспорт в JSON».
Программа должна превращать весь список задач в строку и записывать в файл tasks.json.
Настройте сериализатор так, чтобы он игнорировал регистр имен при последующем чтении.

Задание 4: Восстановление данных
Сделайте так, чтобы при запуске формы (событие Load) программа автоматически проверяла наличие файла tasks.json.
Если файл есть — загрузить задачи в таблицу.
Если файл поврежден (некорректный JSON) — обработать ошибку через try-catch и сообщить пользователю.

Отчет должен содержать (см. образец):
- номер и тему лабораторной работы;
- фамилию, номер группы студента и вариант задания;
- скриншоты окна Visual Studio с исходным кодом программы и комментариями;
- скриншоты с результатами выполнения программ;
- пояснения, если необходимо;
- выводы.

Отчеты в формате **pdf** отправлять на email: **colledge20education23@gmail.com**

Шпаргалка на тему " Сериализация объектов в JSON и работа с файлами "

1. Подключение библиотек

Для работы обязательно добавьте эти строки в самый верх файла:

```
using System.Text.Json;         // Сама магия JSON
using System.IO;                // Работа с файлами
using System.ComponentModel;    // Для BindingList
using System.Text.Encodings.Web; // Чтобы русский язык не превращался в коды
```

2. Настройка "красивого" JSON (с кириллицей)

По умолчанию библиотека заменяет русские буквы на коды вида `\u0410`. Чтобы этого не было, создайте объект настроек:

```
JsonSerializerOptions options = new JsonSerializerOptions
{
    WriteIndented = true, // Делает структуру "лесенкой"
    Encoder = JavaScriptEncoder.UnsafeRelaxedJsonEscaping, // Разрешает
кириллицу
    PropertyNameCaseInsensitive = true // Игнорирует регистр (Name или name)
};
```

3. Правильное объявление класса

DataGridView "видит" только свойства с `get; set;`.

```
public class Item
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

4. Работа с BindingList (Авто-обновление таблицы)

Вместо `List<T>` используйте `BindingList<T>`, чтобы не перепривязывать `DataSource` каждый раз.

```
// 1. Создаем список в классе формы
BindingList<Item> myData = new BindingList<Item>();

// 2. В конструкторе формы связываем его с таблицей
public Form1()
{
    InitializeComponent();
    dataGridView1.DataSource = myData;
}
```

```
// 3. Добавление данных (таблица обновится сама)
myData.Add(new Item { Id = 1, Name = "Тест" });
```

5. Сохранение и Загрузка (Минимальный шаблон)

Сохранение:

```
string json = JsonSerializer.Serialize(myData, options);
File.WriteAllText("data.json", json);
```

Загрузка (с проверками):

```
if (File.Exists("data.json"))
{
    try
    {
        string json = File.ReadAllText("data.json");
        // Десериализуем сначала в обычный List
        var temp = JsonSerializer.Deserialize<List<Item>>(json, options);

        myData.Clear();
        foreach (var item in temp) myData.Add(item);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Ошибка при чтении файла: " + ex.Message);
    }
}
```

пример (Вариант №1: Книги)

Этот код можно использовать как каркас для выполнения работы.

```
// Кнопка добавления книги из TextBox-ов
private void btnAdd_Click(object sender, EventArgs e)
{
    // Собираем объект из полей ввода
    Book newBook = new Book {
        Id = int.Parse(txtId.Text),
        Title = txtTitle.Text,
        Author = txtAuthor.Text,
        Year = (int)numYear.Value
    };

    // Добавляем в список (таблица увидит сама)
    books.Add(newBook);
}

// Кнопка удаления выбранной книги
private void btnDelete_Click(object sender, EventArgs e)
{
    if (dataGridView1.CurrentRow != null)
    {
        // Удаляем объект, соответствующий выделенной строке
        var book = (Book)dataGridView1.CurrentRow.DataBoundItem;
        books.Remove(book);
    }
}
```

Типичные ошибки:

1. **Отсутствие `get; set;`**: Класс создан, данные в коде есть, но в JSON и в таблицу они не попадают.
2. **Забытый `using System.Text.Json`**: Visual Studio подчеркивает `JsonSerializer` красным.
3. **Попытка сохранить `dataGridView1.DataSource` напрямую**: Нужно сначала привести его к типу `BindingList<Book>` или `List<Book>`.
4. **Кодировка**: Если не настроить `Encoder`, в файле JSON вместо «Книга» будет `\u041A\u043D...`