**Лабораторная работа 1. 4 часть**

**Приложение «Телефонный справочник».**

**Сериализация / десериализация.**

**Задание**. Используя сериализацию, сохранить данные в бинарный файл, в xml-файл, в json-файл.

Чтобы сохранить значения для класса Note, прежде всего, необходимо отметить это класс атрибутом Serializable. Добавьте следующий атрибут непосредственно перед определением класса:

[Serializable]

public class Note

{

...

}

Атрибут Serializable сообщает компилятору, что все, находящееся в классе, может быть сохранено в файле. Сериализация применяется к свойствам и полям класса. Если какой-то элемент не подлежит сериализации, его помечают атрибутом NonSerialized.

Начнем с сериализации в бинарный файл. В папке Save создайте класс SerializeBin, реализующий интерфейс ISave. Далее речь пойдет о содержимом метода Save().

Для бинарной сериализации применяется класс BinaryFormatter, определенный в пространстве имен System.Runtime.Serialization.Formatters.Binary.

Создайте объект BinaryFormatter:

IFormatter formatter = new BinaryFormatter();

Далее нужен поток, в который будет сохраняться список. Данный поток представляет объект FileStream, который записывает список в файл с именем fileName (это файл с расширением.dat):

Stream stream = new FileStream(fileName, FileMode.Create, FileAccess.Write, FileShare.None);

formatter.Serialize(stream, notes);

stream.Close();

Сериализация одним методом formatter.Serialize() добавляет все данные об объекте notes в файл fileName.

Чтобы организовать открытие файла опишите в папке Open класс DeserializeBin, реализующий интерфейс IOpen.

В методе OpenFile() создайте список из Note – notes. Затем объект класса BinaryFormatter – formatter. Далее объект FileStream, который записывает считает нужный объект из файла с именем fileName.

При десериализации обязательно преобразовать объект, возвращаемый функцией Deserialize, к типу List<Note>:

using (FileStream fs = new FileStream(fileName, FileMode.OpenOrCreate))

{

notes = (List<Note>)formatter.Deserialize(fs);

}

Метод AddFromFile реализуйте самостоятельно.

Добавьте в форму дополнительные пункты меню для сохранения в бинарный файл. Задайте новые значения в фильтре классов диалогов.

**Запуск**. Запустите и протестируйте приложение. Убедитесь, что все работает правильно, изучите содержимое бинарного файла.

Сериализация в xml-файл. Так как список, который нужно сеарилизовать, имеет не простой тип, то его обязательно нужно пометить атрибутом XmlElement:

[XmlElement("Notes")]

private List<Note> PhoneNote;

Создайте классы SerializeXml, DeserializeXml, реализующие соответствующие интерфейсы в соответствующих папках. Начнем с сериализации. В методе Save запишите следующее:

XmlSerializer writer = new XmlSerializer(typeof(List<Note>));

FileStream file = File.Create(fileName);

writer.Serialize(file, notes);

file.Close();

Центральным классом XML-сериализации является класс XmlSerializer, а наиболее важные методы в нем – Serialize и Deserialize.

В реализации десериализации нужно изменить только две внутренние строки приведенного выше кода:

StreamReader file = new StreamReader(fileName);

notes= (List<Note>)reader.Deserialize(file);

Файл xml, генерируемый XmlSerializer, соответствует рекомендации World Wide Web Consortium (W3C) XML Schema (XSD) 1.0.

**Запуск**. Замените классы SaveXml и OpenXml только что созданными классами. Запустите и протестируйте приложение. Сначала сохраните, затем откройте сохраненные данные. Убедитесь, что все работает правильно, изучите содержимое xml-файла.

Сериализация в Json-файл.

Для сериализации в Json понадобится класс System.Text.Json. Эту библиотеку, скорее всего, придется устанавливать. Зайдите в меню Проект и выберите Управление пакетами NuGet (рис. 1)

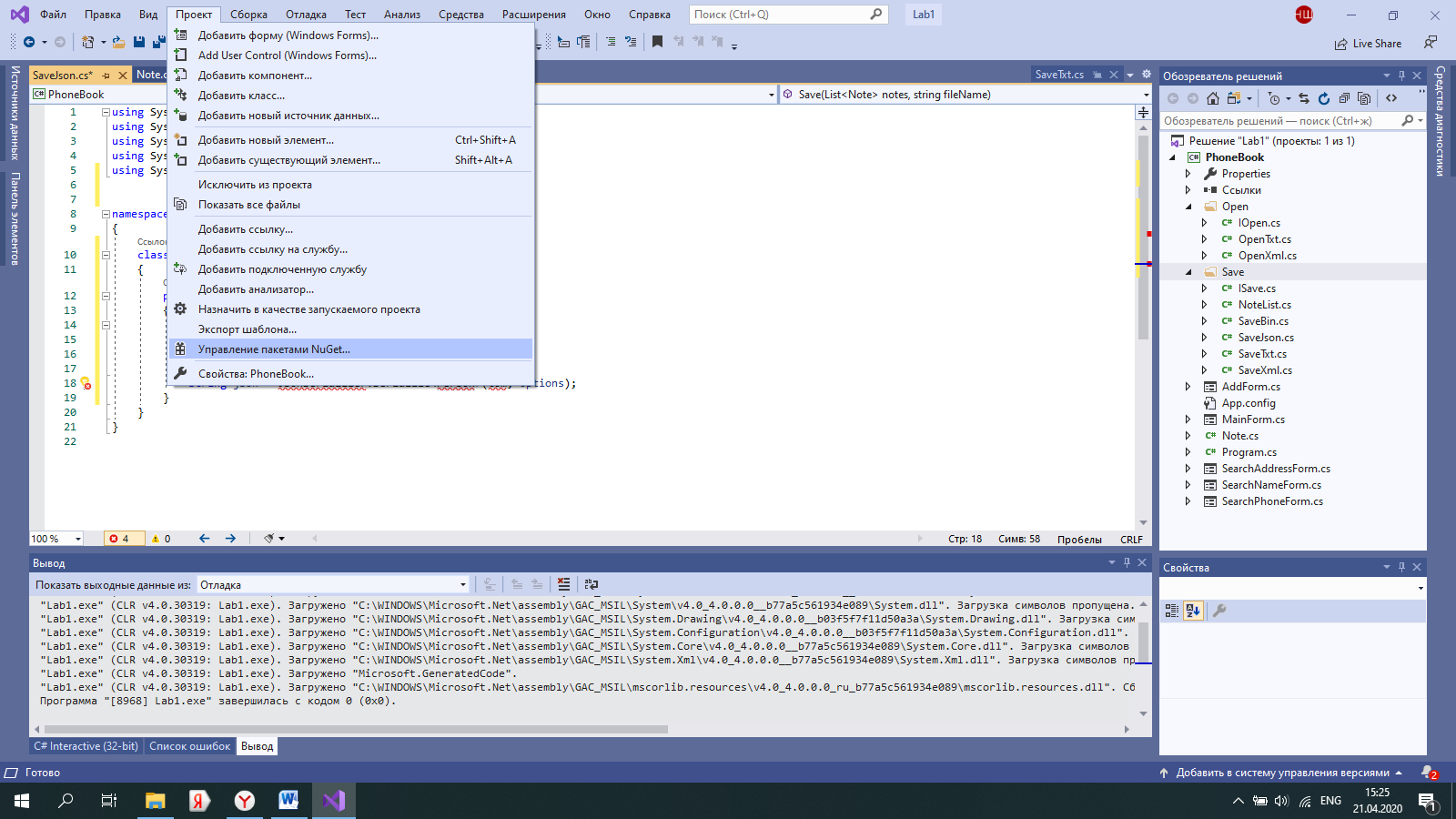


Рисунок 1. Открытие меню NuGet.

Введите в строку поиска – System.Text.Json (рис. 2). Установите найденный пакет.

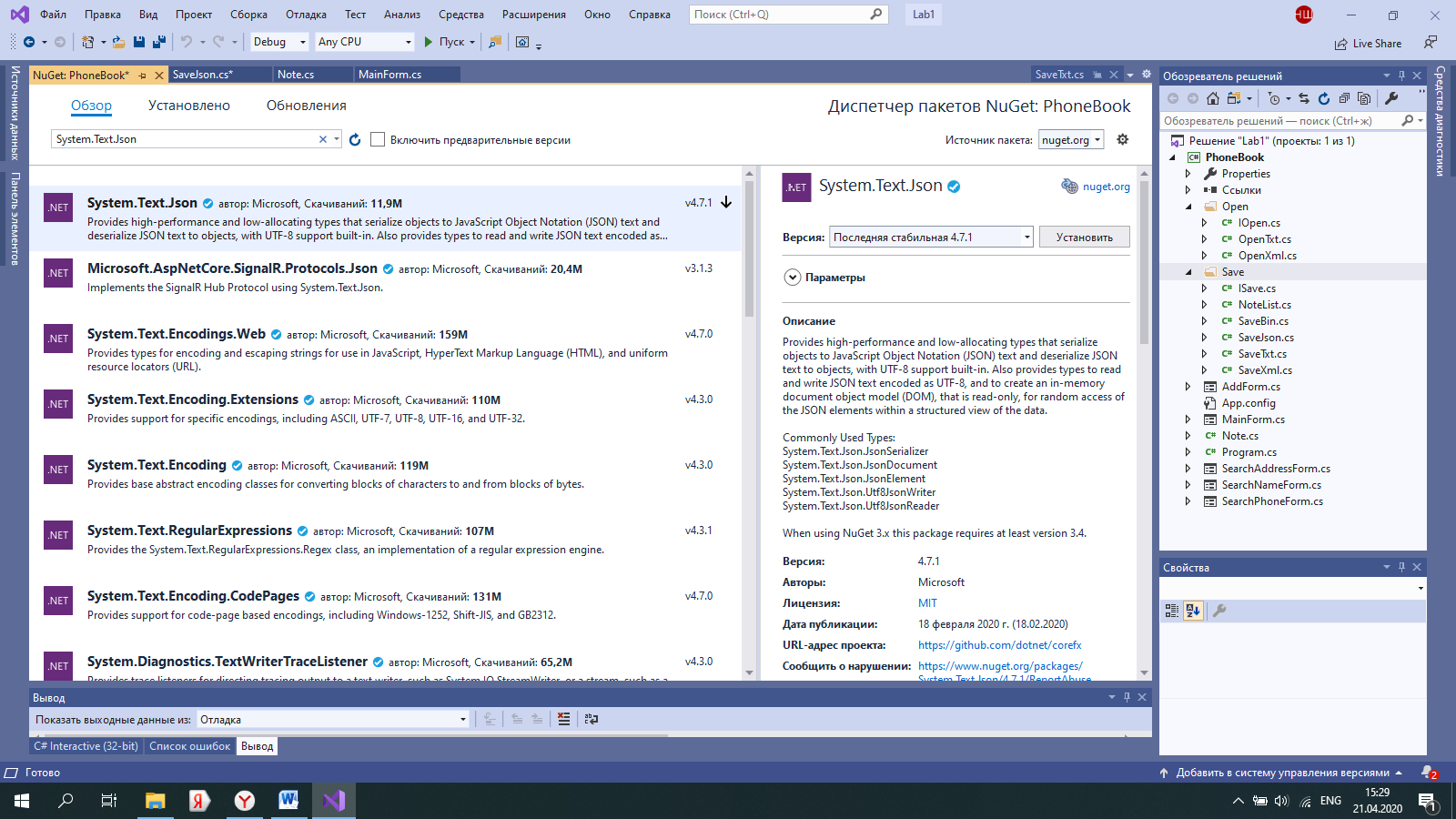


Рисунок 2. Окно NuGet.

Согласитесь с внесением изменений.

Создайте классы SerializeJson, DeserializeJson, реализующие соответствующие интерфейсы в соответствующих папках.

Чтобы сериализовать в методе Save запишите следующие строки:

var options = new JsonSerializerOptions

{

WriteIndented = true

};

string json = JsonSerializer.Serialize(notes, options);

File.WriteAllText(fileName, json);

Параметр options определяет отступы в json-файле. Объект сериализуется в строку, а затем строка записывается в файл.

Обратные действия при открытии файла. Сначала нужно считать содержимое файла в строку s, а затем строку десериализовать в список:

notes = JsonSerializer.Deserialize<List<Note>>(s);

Допишите недостающий код.

**Запуск**. Запустите и протестируйте приложение. Сначала сохраните, затем откройте сохраненные данные. Убедитесь, что все работает правильно, изучите содержимое json-файла. Некоторые символы могут сохраниться не в том виде, в котором были записаны. Почему?

**Простая фабрика**

Делегируем создание класса для открытия файла специальному классу. В папке Open создайте класс с именем OpenCreater. В нем единственный метод:

public static IOpen Create(string fileName)

Этот метод должен содержать всю логику по выбору расширения файла и созданию необходимого класса.

В форме при открытии файла нужно будет написать:

openBehavior = OpenCreater.Create(openFileDialog1.FileName);

Создайте такой же класс для сохранения данных в файл.

Таким образом, вся логика по открытию и сохранению данных выделена из общего кода.

**Запуск**. Запустите и протестируйте приложение.