Контрольное домашнее задание 1, модуль 3

Контрольное домашнее задание предполагает самостоятельную домашнюю работу. Вам потребуется:

1. Изучить предложенные теоретические материалы самостоятельно.
2. Самостоятельно поработать с документацией по языку C#, в т.ч. осуществлять информационный поиск.
3. Разработать программы, определённые основной задачей и индивидуальным вариантом.
4. Вовремя сдать в SmartLMS заархивированный проект с кодом проекта консольного приложения и библиотеки классов, определённые заданием и вариантом.

Формат сдачи работы

Для проверки предоставляется решение, содержащие два проекта: консольное приложение и библиотеку классов. Решение должно быть заархивировано и приложено в качестве ответа на задание в SmartLMS.

Срок выполнения и загрузки работы

Две недели (фактический дедлайн смотреть по SmartLMS)

Опоздания и штрафы

Дедлайн является мягким и еще на протяжении суток работу можно будет отправить на проверку, с учётом штрафов.

Опоздание в часах	**Максимальная оценка, которую можно получить**
1	8
2-3	7
4-5	6
6-7	5
8-9	4
9 и более	1

Изучить самостоятельно

В этом разделе собраны ссылки, с которыми необходимо познакомится для выполнения работы, обратите внимание, что отдельные дополнительные материалы размещены в разделах с требованиями к библиотеке классов и консольному приложению.

Самостоятельно ознакомиться с текстовым форматом данных JSON и материалами о потоковом вводе и выводе данных в файлы языка C#. Для решения задачи КДЗ потребуется использовать перенаправление стандартного потока ввода-вывода в текстовые файлы.

- Файловый и потоковый ввод-вывод (http://learn.microsoft.com/ru-ru/dotnet/standard/io/#streams)

- Класс **Stream** (http://learn.microsoft.com/ru-ru/dotnet/api/system.io.stream?view=net-6.0)
- Класс **FileStream** (http://learn.microsoft.com/ru-ru/dotnet/api/system.io.filestream?view=net-6.0)
- Перенаправление стандартного потока вывода консоли в файл (http://learn.microsoft.com/ru-ru/dotnet/api/system.console.setout?view=net-6.0)
- Получение стандартного потока (https://learn.microsoft.com/ru-ru/dotnet/api/system.console.openstandardinput?view=net-6.0)

Пример перенаправления стандартных потоков в файл

```
using System;
using System.IO;
class Program
{
    static void Main()
    {
        using StreamWriter log = new StreamWriter(@"system_log.txt");
        Console.SetOut(log);

        DateTime dt = DateTime.Now;

        Console.WriteLine("Начало системного журнала.");
        Console.WriteLine($"{dt}; {dt.Millisecond} {Milliseconds}");

        for (int k = 0; k < 100000; k++)
            if (k % 10000 == 0)
                Console.WriteLine(k);

        Console.WriteLine("Конец системного журнала.");
        dt = DateTime.Now;
        Console.WriteLine($"{dt}; {dt.Millisecond}{ Milliseconds}");
        log.Dispose(); // вызов Dispose не обязателен при наличии using выше
    }
}
```

JSON (*JavaScript Object Notation*) - простой текстовый формат обмена данными, основанный на JavaScript, но считается независимым от языка и поддерживается на всех платформах.

- Introducing JSON (http://json.org)
- СтандартECMA-404, 2013 год (http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf)

Данные в формате JSON:

- JavaScript-объекты **{... } (множество пар "имя" : значение);**
- массивы **[...];**
- значения одного из типов:
  - строки в двойных кавычках
  - число (целое или вещественное)
  - логическое значение (**true** или **false**)
  - **null**

Примеры JSON-представления объектов:

Пустой объект:
{}
Объект с одной парой «имя»: «значение»:
{ "lastName" : "Ivanov" }
Объект с несколькими парами «имя»: «значение»:
{ "firstName" : "Ivan", "lastName" : "Ivanov" }
Правила представления массива:

- Начинается с символа **[**
- Заканчивается символом **]**
- Содержит набор значений, разделенных запятыми

Задание

Требования к библиотеке классов

Библиотека классов должна содержать два класса:

1) Класс **MyType (MyType – является заглушкой и должно быть заменено вами на основе понимания данных из файла индивидуального варианта на более подходящее имя для класса)** представляющий объекты, описанные в JSON файле индивидуального варианта. Пример JSON-представления объекта приведён на рисунке 1. Поля класса должны быть доступны для чтения, но закрыты для записи. Класс должен содержать конструктор для инициализации своих полей. Выберите самостоятельно идентификатор класса, который будет логично описывать объект и удовлетворять правилам нейминга Microsoft.

Рисунок 1. Пример описания объекта в формате JSON.

2) Статический класс **JsonParser**, содержащий два статических метода: **WriteJson** и **ReadJson**. Данные методы должны использовать потоки данных, определённые в **System.Console** для чтения и записи данных. Данные подаются в формате JSON, для их парсинга запрещено использовать любые специализированные библиотеки. Вы можете воспользоваться подходом на основе конечных автоматов или регулярных выражений:
    a. Пространство имён System.Text.RegularExpressins (http://learn.microsoft.com/ru-ru/dotnet/api/system.text.regularexpressions?view=net-6.0)
    b. Класс Regex отвечает за объекты, представляющие регулярные выражения (http://learn.microsoft.com/ru-ru/dotnet/api/system.text.regularexpressions.regex?view=net-6.0)
    c. Элементы языка регулярных выражений описаны в кратком справочнике (http://learn.microsoft.com/ru-ru/dotnet/standard/base-types/regular-expression-language-quick-reference)

    Решение задачи не предполагает использования атрибутов типов, рефлексии и контрактов данных.

3) Реализации классов не должны нарушать инкапсуляцию данных и принцип единственной ответственности (Single Responsibility Principle).
4) Иерархии не должны нарушать принципа подстановки Лисков (Liskov Substitution Principle) и проектируются, исходя из соблюдения принципа инверсии зависимостей (Dependency Inversion Principle).
    a. Архитектурные принципы (https://learn.microsoft.com/ru-ru/dotnet/architecture/modern-web-apps-azure/architectural-principles)
5) Реализации классов не должны нарушать доступа к данным, например, предоставлять внешние ссылки на поля или изменять состояние объекта без проверок.
6) Классы библиотеки должны быть доступны за пределами сборки.

7) Каждый нестатический класс (при наличии) обязательно должен содержать, в числе прочих, конструктор без параметров или эквивалентные описания, допускающие его прямой вызов или неявный вызов.
8) Запрещено изменять набор данных для классов, которые строятся на основе JSON-представлений из индивидуальных вариантов.
9) Допускается расширение открытого поведения или добавление закрытых функциональных членов класса.
10) Допускается использование собственных (самописных) иерархий классов в дополнение к предложенным в индивидуальном варианте, но требования по ООП принципам к ним сохраняется.

Пример реализации конечного автомата

В примере решим с помощью конечного автомата задачу подсчета количества строк и комментариев исходном коде, переданном нам на вход. Для упрощения задачи будем считать, что комментарий начинается с символа ‘/’ и заканчивается концом строки, а строка начинается и заканчивается символом ‘"’. В таком случаи наш автомат будет иметь следующие три состояния: (1) Комментарий, (2) Строка, (3) Остальной код программы (см. рисунок 2).

```
public class Program
{

    public enum State
    {
        String,
        Comment,
        Program
    }

    public static void Main()
    {
        Console.WriteLine("Введите код программы:");
        string? code = Console.ReadLine();

        State state = State.Program;
        int commentCount = 0;
        int stringCount = 0;

        foreach (var symbol in code ?? "")
        {
            switch (state)
            {
                case State.Program when symbol == '/':
                    commentCount++;
                    state = State.Comment;
                    break;
                case State.Program when symbol == '"':
                    stringCount++;
                    state = State.String;
                    break;
                case State.Comment when symbol == '\n':
                    state = State.Program;
                    break;
                case State.String when symbol == '"':
                    state = State.Program;
                    break;
            }
        }

        Console.WriteLine($"Количество строк: {stringCount}");
```

```
        Console.WriteLine($"Количество комментариев: {commentCount}");
    }
}
```

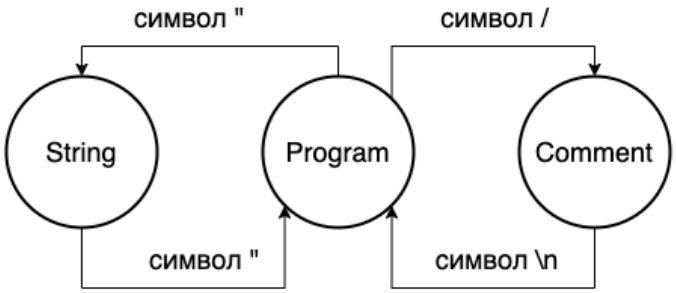

Рисунок 2. Состояния конечного автомата и переходы между ними

Требования к консольному приложению

Консольное приложение использует описанную выше библиотеку классов, при помощи методов статического класса **JsonParser** получает данные для коллекции объектов типа **MyType (не забываем о необходимости выбрать имя типа, исходя из описываемых им объектов)**, данные для объектов получены из JSON-представлений файла индивидуального варианта. Тип коллекции для объектов выбрать самостоятельно.

Также консольное приложение предоставляет пользователю интерфейс для взаимодействия с данными объектов коллекции. Пользователь должен иметь возможность подать данные на вход с помощью стандартного потока ввода и вывода **System.Console** или с помощью файлового потокового ввода-вывода. Заметьте, что библиотека классов использует данные именно из **System.Console**, поэтому для работы с файлом необходимо перенаправить стандартный поток. Стандартный поток **System.Console** перенаправляется с помощью методов **Console.SetIn** и **Console.SetOut**. Чтобы снова использовать стандартную консоль, нужно вернуть перенаправленный поток с помощью методов **Console.OpenStandardInput** и **Console.OpenStandardOutput**.

Пользователю должно предоставляться экранное меню со следующими возможностями:

1. Ввести данные через **System.Console** или предоставить путь к файлу для чтения данных.
2. Отфильтровать данные по одному из полей. Поля для фильтрации выбирает пользователь. При работе с программой должно быть понятно, какое поле было выбрано для фильтрации.
3. Отсортировать данные по одному из полей. Поле для сортировки выбирает пользователь. При работе с программой должно быть понятно, по какому полю проведена сортировка.
4. Вывести (сохранить) данные в **System.Console** или файл. Пользователь должен иметь возможность перезаписать файл, из которого были считаны данные, или указать новый путь к файлу для записи.

При сортировке и фильтрации по определенному полю не забываем, что выводить нужно не только значения этого поля.

Общие требования к КДЗ

1. Цикл повторения решения и проверки корректности получаемых данных **обязательны**.
2. Соблюдение определённых программой учебной дисциплины требований к программной реализации работ – **обязательно**.
3. Соблюдение соглашений о качестве кода – **обязательно**.

4. Весь программный код должен быть написан на языке программирования C# с учётом использования .net 6.0.
5. Для реализации КДЗ запрещено использовать сторонние библиотеки, Nuget-пакеты и сервисы.
6. Исходный код должен содержать комментарии, объясняющие неочевидные фрагменты и решения, резюме кода, описание целей кода (см. материалы лекции 1, модуль 1).
7. При перемещении папки проекта библиотеки (копировании / переносе на другое устройство) файлы должны открываться программой также успешно, как и на компьютере создателя, т.е. по относительному пути.
8. Текстовые данные, включая данные на русском языке, успешно декодируются при представлении пользователю и человекочитаемы.
9. Ресурсы, выделяемые при работе с файлами, должны освобождаться программой;
10. Все созданные при сохранении программой JSON-файлы имеют такую же структуру, как и файл с примером и должны без проблем читаться в качестве входных данных;
11. Программа **не** допускает пользователя до решения задач, пока с клавиатуры не будут введены корректные данные.
12. Консольное приложение обрабатывает исключительные ситуации, связанные (1) со вводом и преобразованием / приведением данных, как с клавиатуры, так и из файла; (2) с созданием, инициализацией, обращением к элементам массивов и строк; (3) с вызовом методов библиотеки.
13. Представленная к проверке библиотека классов должна решать все поставленные задачи и успешно компилироваться.
14. Поскольку в описаниях классов присутствует «простор» для принятия решений, то каждое такое решение должно быть описано в комментариях к коду программы. Например, если выбран тип исключения, то должно быть письменно обоснованно, почему вы считаете его наиболее подходящим в рамках данной задачи.

Индивидуальные варианты

Номер варианта	**Имя файла**
1	data_1V.json
2	data_2V.json
3	data_3V.json
4	data_4V.json
5	data_5V.json
6	data_6V.json
7	data_7V.json
8	data_8V.json
9	data_9V.json
10	data_10V.json
11	data_11V.json
12	data_12V.json
13	data_13V.json
14	data_14V.json
15	data_15V.json
16	data_16V.json
17	data_17V.json
18	data_18V.json
19	data_19V.json
20	data_20V.json