~ *Переписывание проекта 1, модуль №3* ~

Предисловие

В рамках этого проекта вы будете работать с реальными данными - данными игр «Cultist Simulator» и «Book Of Hours», разработанных «Weather Factory». Мы получили официальное разрешение на использование их интеллектуальной собственности в образовательных целях. Надеемся, что вам понравится этот проект.

Общие слова

Что делаем?

Проект предполагает самостоятельную домашнюю работу по разработке консольного приложения. Вам потребуется:

1. Изучить предложенные теоретические материалы самостоятельно.
2. Самостоятельно поработать с документацией по языку C#, в т.ч. осуществить информационный поиск.
3. Разработать программы, определённые основной задачей и индивидуальным вариантом.
4. Сдать в SmartLMS заархивированную папку с решением (Solution) Visual Studio, где содержится проект(ы) с решением вашей задачи.

Когда сдаем?

Определяется датами, назначенными в SmartLMS.

Что сдаём?

На проверку предоставляется заархивированная папка с решением Visual Studio, в которое включен(ы) проект(ы), через который(е) реализована программа. В названии решения и архива обязательно указать фамилию автора.

Изучить самостоятельно

В этом разделе собраны ссылки, с которыми необходимо ознакомится при 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 разработки. Для написания проекта потребуется использовать перенаправление стандартного потока ввода-вывода в текстовые файлы.

- Файловый и потоковый ввод-вывод
- Класс `Stream`
- Класс `FileStream`
- Перенаправление стандартного потока вывода консоли в файл
- Получение стандартного потока

Общие требования к работе

В индивидуальном варианте вы получите путь к JSON-файлам, изучив которые спроектируете нестатические классы, подходящие для представления данных в своей программе. Каждый класс следует размещать в отдельном файле с исходным кодом. Стоит отметить, что JSON файлы не отформатированы. Ваша программа **должна** уметь работать с форматированными и неформатированными JSON'ами. JSON'ы которые не считаются валидными по стандарту RFC 8259 читаться **не должны**. Если файл некорректен, то нужно сообщить об этом пользователю и запросить файл повторно.

Ваша программа представляет собой справочную систему, которая за счёт использования экранного меню и диалогов с пользователем, предоставляет возможность работы с данными, представленными в JSON-формате. Программа должна обеспечивать выполнение действий, определённых общими требованиями и индивидуальным вариантом работы.

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
    }
}
```

При работе с данными:

1. Программа должна сохранять работоспособность при вводе некорректных адресов и имён файлов, с учётом различных платформ запуска файлов. При получении на вход некорректного адреса либо имени файла пользователю выводится сообщение об ошибке и, затем, экранное меню.
2. Некорректно структурированный файл вашей программой не обрабатывается, пользователю выводится сообщение об ошибке и, затем, экранное меню.
3. Программа обязательно должна корректно открывать созданные ею файлы и позволять выполнять над ними все операции в соответствии с пунктами меню.
4. Программа автоматически определяет кодировку файла и выводит данные на экран и в файлы в человекочитаемом виде.
5. Если у объектов будут отсутствовать некоторые из ожидаемых ключей, то **не** прерывайте работу программы и **не** выводите предупреждение. Заполните эти поля значениями по-умолчанию (выберите самостоятельно).

Критерии качества исходного кода:

1. В этой работе мы опираемся на создание нестатических классов и принципы объектно-ориентированного программирования (ООП).
2. Разделите программный код по отдельным файлам, т.е. каждый тип (класс, структура и т.п.) должен быть реализован в отдельном файле с исходным кодом.

Ограничения

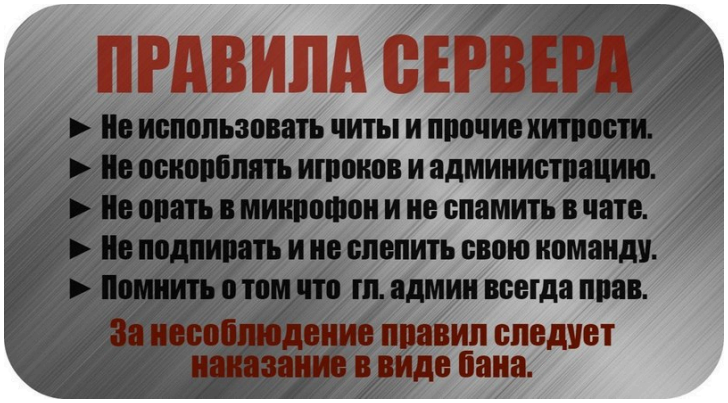

Рис. 1. Мем про правила

В основной и дополнительной задачах **требуется** (да, это блокирующие критерии):

- весь программный код написан на языке программирования C# с учётом использования .net 8.0;
- представленная к проверке библиотека классов решает все поставленные задачи и успешно компилируется.
- в приложении реализован цикл повторения решения, позволяющий повторить работу на других данных без завершения сеанса работы;
- имена классов соответствуют соглашениям об именовании;
- иерархии типов не нарушают принципа подстановки Лисков (Liskov Substitution Principle) и быть спроектированы, исходя из соблюдения принципа инверсии зависимостей (Dependency Inversion Principle);
- спроектированные классы соответствуют принципу единственной ответственности (Single Responsibility Principle) и не нарушают инкапсуляцию;
- поскольку мы активно используем интерфейсы, то важно соблюдение принципа разделения интерфейсов (Interface Segregation Principle);
- каждый нестатический класс (при наличии) содержит, в числе прочего, конструктор без параметров, либо эквивалентные описания, допускающие его явный или неявный вызов;
- не использованы:
  - NuGet-пакеты, использование которых не разрешено условием;
  - `System.Text.Json`;
  - прочий публично доступный код / библиотеки, не относящиеся к стандартным.

В основной и дополнительной задачах **требуется**:

- при перемещении папки проекта библиотеки (копировании / переносе на другое устройство) файлы должны открываться программой также успешно, как и на компьютере создателя (т.е. вашем), т.е. по относительному пути;
- текстовые данные, включая данные на русском языке, успешно декодируются при представлении пользователю и человекочитаемы;
- неуправляемые ресурсы, выделяемые при работе с файлами, обязательно освобождаются программой;
- 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;
- программа не допускает пользователя до решения задач, пока с клавиатуры не будут введены корректные данные;
- консольное приложение обрабатывает исключительные ситуации, связанные:
  - со вводом и преобразованием / приведением данных, как с клавиатуры, так и из файла;
  - с созданием, инициализацией, обращением к элементам массивов и строк;
  - с вызовом методов библиотеки;
- исходный код должен содержать комментарии, объясняющие не очевидные фрагменты и решения, резюме кода, описание целей кода.

Описание задачи

Требования к библиотеке классов

Интерфейс `IJSONObject`

Реализации этого интерфейса должны предоставлять следующие методы:

- `IEnumerable<string> GetAllFields`: возвращает коллекцию строк, представляющую имена всех полей объекта JSON
- `string GetField(string fieldName)`: возвращает значение поля с указанным именем `fieldName` в формате строки. В случае отсутствия поля с заданным именем, возвращает `null`.
- `void SetField(string fieldName, string value)`: присваивает полю с именем fieldName значение value. Если поле с таким именем отсутствует, то должно генерироваться исключение `KeyNotFoundException`.

Классы представляющие объекты

Классы представляют объекты, описанные в JSON файле индивидуального варианта. Вложенные и связанные объекты описываются отдельными классами. В индивидуальном варианте приведено описание типов отношений между связанными объектами. Поля каждого класса должны быть доступны для чтения, но закрыты для записи. Выберите самостоятельно идентификатор класса, который будет логично описывать объект и удовлетворять правилам именования классов. Все такие классы должны реализовывать интерфейс `IJSONObject`.

Класс `JsonParser`

Статический класс `JsonParser` содержит два статических метода: `WriteJson` и `ReadJson`. Эти методы обязательно используют 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
- регулярных выражений
  - Пространство имён `System.Text.RegularExpressins`
  - Класс `Regex`
  - Краткий справочник элементов языка регулярных выражений

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

Важное примечание

Мы рекомендуем вам реализовывать работу `JsonParser` так, чтобы ему было все равно какой класс или структура «прячется» под интерфейсом. Используйте только методы, описанные в интерфейсе. Такая реализация будет проще и правильнее.

Пример реализации конечного автомата

В примере решим с помощью конечного автомата задачу подсчета количества строк и комментариев в 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 случае 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}");
    }
}
```

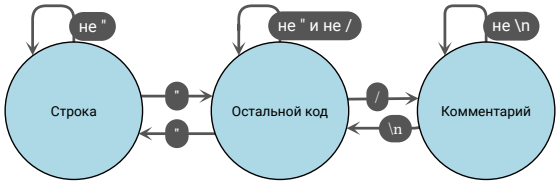

Рис. 2. Состояния конечного автомата и переходы между ними

Требования к консольному приложению

Консольное приложение предоставляет пользователю интерактивный интерфейс для работы с коллекцией объектов, описанных в индивидуальном варианте. Приложение использует библиотеку классов, описанную выше, для работы с JSON.

Взаимодействие с данными

- **Чтение данных:**
  - Приложение должно уметь читать данные из стандартного потока ввода `System.Console` с помощью `Console.ReadLine()`.
  - Учтите, что вставленный в терминал файл может быть многострочным
  - Приложение должно предоставлять возможность чтения данных из файла, указанного пользователем. Для этого необходимо перенаправить стандартный поток ввода с помощью метода `Console.SetIn()`.
- **Запись данных:**
  - Приложение должно уметь выводить данные в стандартный поток вывода `System.Console` с помощью `Console.WriteLine()`.
  - Приложение должно предоставлять возможность записи данных в файл, указанный пользователем. Для этого необходимо перенаправить стандартный поток вывода с помощью метода `Console.SetOut()`. Пользователь должен иметь возможность перезаписать исходный файл или указать новый путь для сохранения.

Работа с коллекцией объектов

Приложение должно предоставлять пользователю следующие возможности через экранное меню:

- **Ввод данных:** Выбор источника данных (`System.Console` или файл).
- **Фильтрация:**
  - Выбор поля для фильтрации из списка всех доступных полей объекта;
  - Отображение выбранного поля для фильтрации в интерфейсе приложения;
  - Фильтрация объектов коллекции по заданному критерию, связанному с выбранным полем;
  - Пользователь должен вписать массив объектов, которые должны остаться в этом столбце.
- **Сортировка:**
  - Выбор поля для сортировки из списка всех доступных полей объекта и направления сортировки;
  - Отображение выбранного поля для сортировки в интерфейсе приложения;
  - Сортировка объектов коллекции по значениям выбранного поля.
- **Вывод данных:** Выбор места вывода данных (`System.Console` или файл).

Сортировку и фильтрацию нужно реализовывать **только** для полей со строками, целыми и вещественными числами. Сортировка и фильтрация массивов **не** предполагается. Вам будет чем заняться...

Важно

- При фильтрации и сортировке необходимо отображать все поля объекта, а не только поле, используемое для фильтрации/сортировки.
- Для работы с JSON приложение использует статический класс `JsonParser` и интерфейс `IJSONObject`.
- Для возврата к стандартным потокам ввода/вывода после работы с файлами необходимо использовать методы `Console.OpenStandardInput()` и `Console.OpenStandardOutput()`.

Пример структуры меню

```
1. Ввести данные (консоль/файл)
2. Отфильтровать данные
3. Отсортировать данные
4. <основная задача индивидуального варианта>
5. <дополнительная задача индивидуального варианта>
6. Вывести данные (консоль/файл)
7. Выход
```

~ *Индивидуальные варианты* ~

~ *Вариант №1* ~

CS: Ритуалы

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с ритуалами в Cultist Simulator. Эти данные представлены в формате JSON и хранятся в файлах типа `./CS/<locale>/elements/rituals.json`

Вам необходимо создать **структуру** «Ритуал» для представления этих данных в приложении.

Основная задача

Разработайте приложение, которое позволит пользователю выбирать ритуал по его ID и отображать подробную информацию о нём. Формат вывода:

```
Ритуал: Rite of the Sea's Feasting (ritetoolconsumeingredient)

Описание: This rite calls on the Witch-and-Sister to close the gap between what is and what might
be. The adept pours, or hurls, an offering into water: preferably, but not essentially, the sea.

Слоты:
  - Invocation
  - Instrument
  - Offering
  - Desire
```

Дополнительная задача

Реализуйте TUI-версию приложения с использованием библиотеки `Terminal.Gui` из NuGet. Функциональность должна быть расширена:

- Меню выбора файла:
  - Отображение всех файлов в текущей директории.
  - Возможность навигации по директориям (переход в дочернюю/родительскую директорию).
- Индикатор выполнения:
  - Отображение панели загрузки во время выполнения длительных операций (чтение/запись файлов, слияние данных).
- Отображение слотов в виде «карточек» в интерфейсе

Требования к реализации.

~ *Вариант №2* ~

BoH: Рецепты

Общая информация

ВВ рамках проекта вам предстоит разработать приложение для работы с рецептами в Book of Hours. Эти данные представлены в формате JSON и хранятся в файлах типа `./CS/<locale>/elements/rituals.json`. Данные о рецептах хранятся в JSON файлах, в директории `./BoH/<locale>/recipes/`.

Вам необходимо создать структуру «Рецепт» для представления этих данных.

Основная задача

Разработайте приложение, которое позволяет пользователю ввести ID рецепта и отображать информацию о процессе его выполнения и результатах. Формат вывода может быть следующим:

```
Рецепт: Craft: Labhitic Tincture (craft.edictsmartial_tincture.labhitic_edge)

Описание: As it clarifies, the Tincture vibrates judderingly. The technical term for this is
'growling'.

Необходимые аспекты:
    ability: 1
    s.edictsmartial: 1
    edge: 5

Эффекты:
    tincture.labhitic: 1

Время выполнения: 60 секунд
```

Дополнительная задача

Напишите функцию конвертации загруженных из JSON данных в Excel-таблицу (*.xlsx) и обратно.

Требования к реализации.

~ *Вариант №3* ~

CS: Культы

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с культами в Cultist Simulator. Эти данные представлены в формате JSON и хранятся в файлах типа `./CS/<locale>/elements/cults.json`

Вам необходимо создать **структуру** «Культ» для представления этих данных в приложении.

Основные задачи

Реализуйте следующие функции:

- **Дозагрузка данных**
  Функция должна обеспечивать добавление новых данных к существующим без удаления старых. При обработке новых данных, необходимо проверять наличие культа с таким же id в коллекции. Если культ с таким id уже существует, старый объект заменяется новым. Если аспект с таким id отсутствует, новый культ добавляется в коллекцию.
- **Редактирование данных**
  Функция должна позволять пользователю редактировать существующий культ по его id. Если культ с указанным id не найдено, должен быть создан новый объект с этим id. Способ реализации редактирования не регламентируется.
- **Удаление данных**
  Пользователь вводит `id` культа, который он хочет удалить. Этот культ удаляется.
- **Слияние данных**
  Функция должна объединять данные из двух файлов, выбранных пользователем. Если культ присутствует только в одном из файлов, он добавляется в итоговый файл. Если культ присутствует в обоих файлах, приложение должно запросить у пользователя, какую версию культа (из какого файла) следует использовать в итоговом файле.

Дополнительная задача

Реализуйте TUI-версию приложения с использованием библиотеки `Terminal.Gui` из NuGet. Функциональность приложения должна быть расширена:

- Меню выбора файла:
  - Отображение всех файлов в текущей директории.
  - Возможность навигации по директориям (переход в дочернюю/родительскую директорию).
- Индикатор выполнения:
  - Отображение панели загрузки во время выполнения длительных операций (чтение/запись файлов, слияние данных).
- Инструмент слияния (Merge tool):
  - Разделение экрана на две части для отображения информации о «спорных» культах из каждого файла.

Требования к реализации.

~ *Вариант №4* ~

BoH: Посетители

Общая информация

В рамках проекта вам предстоит разработать приложение для работы с посетителями в Book of Hours. Эти данные представлены в формате JSON: `./BoH/<locale>/elements/visitors*.json`

Вам необходимо создать **структуру** «Посетитель» для представления этих данных в приложении (название типа данных не должно нарушать соглашений об именовании).

Основная задача

Разработайте приложение, которое позволяет пользователю выбрать персонажа по его ID и отображать ключевую информацию о нём. Формат вывода может быть следующим:

```
Персонаж:  Изидора Базилиска (isidora_basilica)

Описание:  Загадочная и эрудированная смотрительница библиотеки. Хранительница древних знаний.

Аспекты:
    lore: 7
    lantern: 5
    grail: 3
```

Дополнительная задача

Напишите функцию конвертации загруженных из JSON данных в Excel-таблицу (*.xlsx) и обратно.

Требования к реализации.

~ *Приложение: Требования к реализации TUI* ~

В интерфейсе должен быть доступен весь функционал, что и в консольном приложении, то есть:

- импортирование/экспортирование файла с данными
- фильтрация данных
- сортировка данных
- основная задача из индивидуального варианта

Также требуется добавить кнопку «О программе» в `MenuBar`. При нажатии появляется информация о том, кто эту программу делал и номер индивидуального варианта.

Требуется сдать **две** версии программы: с классическим терминальным интерфейсом и с TUI.

При невыполнении любого из требований за дополнительную задачу будет поставлено 0 баллов.

~ *Приложение: Требования к реализации конвертации в Excel документ* ~

- Запрещается хранить словари или массивы в ячейках
- Первая строка каждой страницы таблицы должна иметь акцентный цвет и содержать названия столбцов
- При конвертации в табличный и из табличного вида информация не должна теряться. Порядок объектов должен быть сохранен.

При невыполнении любого из требований за дополнительную задачу будет поставлено 0 баллов.