Контрольное домашнее задание 2, 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 заархивированное решение, включающее в себя код 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

Задание

Требования к библиотеке классов

Библиотека классов должна содержать:

1) Классы **MyType (имя MyType – является заглушкой и должно быть заменено вами на более подходящее имя для каждого класса, на основе понимания данных из файла индивидуального варианта)** представляют объекты, описанные в JSON файле индивидуального варианта. Вложенные и связанные объекты описываются отдельными классами. В таблице с индивидуальными вариантами приведено описание типов отношений между связанными объектами. Поля каждого класса должны быть доступны для чтения, но

закрыты для записи. Классы должны содержать конструктор для инициализации своих полей. Помимо полей, описанных JSON-файлом, классы должны реализовывать метод **ToJSON**, предоставляющий строковое представление текущего объекта в JSON формате. В индивидуальном варианте также могут быть описаны дополнительные методы и требования для реализации паттерна «Наблюдатель» (Издатель-Подписчик). Выберите самостоятельно идентификатор (имя) для класса, таким образом, чтобы он логично описывал объект и удовлетворял правилам нейминга Microsoft.

2) Все классы **MyType** реализуют событие **EventHandler<EventArgs> Updated**, которое отвечает за оповещение подписчиков об изменении объекта. Реализуйте класс наследник **EventArgs**, хранящий в себе дату и время изменений.
3) Класс **AutoSaver**, подписанный на события **Updated** объектов из JSON файла. При получении двух событий с разницей не более 15 секунд должен записывать текущее состояние коллекции объектов в JSON новый файл **<original_json_file_name>_tmp.json**
4) Реализации классов не должны нарушать инкапсуляцию данных и принцип единственной ответственности (Single Responsibility Principle).
5) 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)
6) Реализации классов не должны нарушать инкапсуляцию и отношения, заданные между типами, например, предоставлять внешние ссылки на поля или изменять состояние объекта без проверок.
7) Классы библиотеки должны быть доступны за пределами сборки.
8) Каждый нестатический класс (при наличии) обязательно должен содержать, в числе прочих, конструктор без параметров или эквивалентные описания, допускающие его прямой или неявный вызов.
9) Запрещено изменять набор данных для классов, которые строятся на основе JSON-представлений из индивидуальных вариантов (например, добавлять поля, не содержащиеся в JSON-представлении).
10) Допускается расширение открытого поведения или добавление закрытых функциональных членов класса.
11) Допускается использование собственных (самописных) иерархий классов в дополнение к предложенным в индивидуальном варианте, также с соблюдением ООП принципов.

Требования к консольному приложению

Консольное приложение использует описанную выше библиотеку классов и, при помощи стандартной библиотеки System.Text.Json, получает данные для формирования объектов типов **MyType (не забываем о необходимости выбрать имя каждого типа, исходя из описываемых им объектов)**. Другие варианты JSON-сериализации в данном задании запрещены. Данные для объектов, как и тип связи между ними, извлекаются из JSON-представления файла индивидуального варианта. Тип коллекции для объектов необходимо выбрать самостоятельно. Ознакомиться со стандартной Json сериализацией вы можете в документации Microsoft и в материалах лекций:

- Пример сериализации (https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to)
- Пример десериализации (https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/deserialization)

Приложение должно реализовывать меню, которое позволяет пользователю:

1. Передать путь к файлу для считывания и записи данных.
2. Отсортировать коллекцию объектов по одному из полей (не включая вложенные объект).

3. Выбрать объект и отредактировать в нем любое поле, кроме идентификаторов (поля, в названии которых есть «Id») и полей, которые изменяются в результате активации и обработки событий. При этом ввод пользователя должен корректно обрабатываться, и, в случае некорректного ввода, запрашиваться повторно.

В таблице с индивидуальными вариантами вы найдете дополнительные события, которые необходимо реализовать. В колонке «События» описано при каких условиях должно активироваться событие, а в колонке «Обработка событий», как должны действовать подписчики при возникновении события.

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 программой JSON-файлы при сохранении имеют такую же структуру, как и файл с примером и должны без проблем обрабатываться программой 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.

				красных и желтых карточек больше 7, то выводится сообщение о дисквалификации команды.
7	***7V.json***	**Агрегация.** Внешние объекты агрегируют вложенные	Изменение доходов фильма.	При изменении доходов фильма, для всех актеров, участвовавших в этом фильме, их доход пересчитывается согласно фиксированному проценту, который они получают с дохода фильма.
8	***8V.json***	**Ассоциация.** Внешние объекты ассоциированы с вложенными и наоборот	Изменение рейтинга актера	При возникновении события пересчитываются рейтинги всех фильмов с участием актера (рейтинг фильма равен среднему рейтингу актеров, которые принимают участие)
9	***9V.json***	**Композиция.** Внешние объекты содержат вложенные	Изменение расходов на актеров (**earnings** * **actorsPercent**)	При возникновении события пересчитывается доход актеров фильма (считаем, что все актеры получают равные суммы за участие)
10	***10V.json***	**Агрегация.** Внешние объекты агрегируют вложенные	Изменение рейтинга рецензии	При возникновении события пересчитывается рейтинг книги (среднее от рейтинга рецензий)
11	***11V.json***	**Композиция.** Внешние объекты содержат вложенные	Изменение доступности книги.	При изменении доступности книги, все участники, находящиеся в листе ожидания, получают уведомление о возможности взять книгу.
12	***12V.json***	**Ассоциация.** Внешние объекты ассоциированы с вложенными и наоборот	Изменение доходов книги	При возникновении события пересчитывается доход каждого автора (считаем, что все авторы получают равные суммы за написание книги)
13	***13V.json***	**Агрегация.** Внешние объекты агрегируют вложенные	Изменение количества очков за достижение	При возникновении события пересчитывается количество очков **game_score** пользователя по формуле **level * sum(points)**
14	***14V.json***	**Ассоциация.** Внешние объекты ассоциированы с вложенными и наоборот	Добавление достижения игроку.	Каждый игрок при получении достижения получает дополнительно к своему **game_score** количество очков, равное **points достижения / количество человек получивших это достижение**. При возникновении события добавления достижения новому игроку, очки всех игроков, получивших это достижение, пересчитываются согласно новому количеству игроков имеющих это достижение.