Контрольное домашнее задание № 2, модуль 2

Контрольное домашнее задание предполагает самостоятельную домашнюю работу. Вам потребуется:

1. Изучить некоторые 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. Сдать в SmartLMS вовремя заархивированный проект с кодом проекта консольного приложения и библиотеки классов, определённые заданием и вариантом.

Время выполнения

На выполнение работы отводится одна неделя, точные даты выполнения устанавливаются в SmartLMS.

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.

Общее задание

В одном решении разместить проект библиотеки классов и проект консольного приложения. Подробные описание библиотеки и приложения см. в индивидуальном варианте.

Требования к библиотеке классов

1. Реализации классов не должны нарушать инкапсуляцию данных и принцип единственной ответственности (Single Responsibility Principle).
2. Реализации классов должны содержать регламентированный доступ к данным.
3. Классы библиотеки должны быть доступны за пределами сборки.
4. Каждый нестатический класс обязательно должен содержать, в числе прочих, конструктор без параметров или эквивалентные описания, допускающие его прямой вызов или неявный вызов.
5. Запрещено изменять набор данных (удалять / дополнять), хранящихся в классах или не использовать указанные в задании открытые варианты поведения.
6. Допускается расширение открытого поведения или добавление закрытых функциональных членов класса.
7. Допускается использование абстрактных типов данных, таких как List, ArrayList, Set, Stack, их обобщённых реализаций и проч., но не в качестве замены определённых вариантом структур данных.

8. 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.

Требования к консольному приложению

1. Предусмотреть проверку корректности для каждого ввода данных, обработку исключений, в т.ч. порождаемых классами библиотеки, организацию повторения решения.
2. Допускается использование абстрактных типов данных, таких как List, ArrayList, Set, Stack, их обобщённых реализаций и проч., но не в качестве замены определённых вариантом структур данных.
3. Все данные приложения читают из текстовых файлов, результат работы **и** выводится на экран, **и** сохраняется в другом текстовом файле. Файлы размещаются обязательно рядом с исполняемым файлом консольного приложения (.EXE). Имена входного и выходного файлов вводятся пользователем с клавиатуры.

Общие требования к работе

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 (https://learn.microsoft.com/ru-ru/dotnet/csharp/fundamentals/coding-style/coding-conventions).
4. Весь программный код должен быть написан на языке программирования C# с учётом использования .net 6.0;
5. исходный код должен содержать комментарии, объясняющие неочевидные фрагменты и решения, резюме кода, описание целей кода (см. материалы лекции 1, модуль 1);
6. 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;
7. текстовые данные, включая данные на русском языке, успешно декодируются при представлении пользователю и человекочитаемы;
8. программа не допускает пользователя до решения задач, пока с клавиатуры не будут введены корректные данные;
9. консольное приложение обрабатывает исключительные ситуации, связанные (1) со вводом и преобразованием / приведением данных как с клавиатуры, так и из файлов; (2) с созданием, инициализацией, обращением к элементам массивов и строк; (3) вызовом методов библиотеки.
10. представленная к проверке библиотека классов должна решать все поставленные задачи, успешно компилироваться.

Что сдаём?

Архив с папкой решения (Solution), содержащий проекты консольного приложения и библиотеки классов, определённых индивидуальным вариантом.

Индивидуальные варианты

Вариант 1

В библиотеке классов объявить класс **`CharArr2D`**:

- Поле класса – **`char[][] _charArr`**.
- Конструктор с одним параметром **`String sentence`**. В качестве параметра **`sentence`** передаётся предложение. Предложение – любая последовательность символов, завершающаяся точкой. Если предложение состоит только из символа «точка» - считать это предложением нулевой длины. Слово – любая последовательность латинских символов, не содержащая пробелов (пробел или пустая строка не считаются словом). Слова предложения разделены ровно одним пробелом. Поле**`_charArr`** заполняется, по следующему правилу: каждый массив содержит символы одного слова из **`sentence`**. При некорректном значении **`sentence`** конструктор должен выбрасывать исключение.
- Конструктор с одним параметром **`char[][] arr`**. По ссылке **`_charArr`** создаётся глубокая копия массива **`arr`**. При некорректном значении **`arr`** конструктор выбрасывает исключение наиболее подходящего типа.
- Свойство **`char[][] OnlyVowels`**, возвращает массивы из **`_charArr`**, состоящие только из гласных букв латинского алфавита.

В основной программе, используя библиотеку классов, получить из входного файла строки латинских символов, разделённых пробелом. На основе этих строк инициализировать массив объектов **`A`** типа **`CharArr2D`**. На основе **`A`** создать новый массив объектов **`B`** типа **`CharArr2D`**, состоящий из массивов, включающих только гласные буквы, данные из массива **`B`** сохранить в выходной файл.

Вариант 2

В библиотеке классов объявить класс **`CharArr2D`**.

- Поле класса – **`char[][] _charArr`**.
- Конструктор с параметром **`String sentence`**. В **`sentence`** передаётся предложение. Предложение – любая последовательность символов, завершающаяся точкой. Если предложение состоит только из символа «точка» - считать это предложением нулевой длины. Слово – любая последовательность латинских символов, не содержащая пробелов (пробел или пустая строка не считаются словом). Слова предложения разделены одним пробелом. Массив, связанный с **`_charArr`** заполняется, по следующему правилу: каждый массив символов содержит символы одного слова предложения. При некорректном значении **`sentence`** конструктор должен выбрасывать исключения.

- Конструктор с параметром **char[][] arr**. По ссылке **_charArr** сохраняется копия элементов из массива **arr**. При некорректном значении **arr** конструктор выбрасывает исключение наиболее подходящего типа.
- Свойство **char[][] OnlyConsonants** возвращает массив массивов символов, состоящий только из тех массивов **_charArr**, которые состоят только из согласных букв латинского алфавита

В основной программе, используя библиотеку классов, получить из входного файла строки латинских символов, разделённых пробелом, на их основе проинициализировать объекты массива **A** объектами типа **CharArr2D**. На основе **A** построить массив **B** с объектами типа **CharArr2D**, включающий в себя только те массивы ненулевой длины, в которых нет гласных букв. Данные из массива **B** сохранить в выходной файл..

Вариант 3

В библиотеке классов объявить класс **MyStrings:**

- Поле класса **string[] _sentences**.
- Конструктор с параметрами **string str** и **char ch**. В конструкторе создаётся массив строк, состоящий из частей строки **str**, разделённых символом **char**. Полученный массив строк связывается со ссылкой **_sentences**.
- Свойство **ACRO** возвращает ссылку на массив строк, состоящий из аббревиатур элементов массива **_sentences**. Аббревиатуры формируются по первым символам слов, строчные символы капитализируются (т.е. преобразуются в заглавные буквы).

В основной программе, используя библиотеку классов, получить из файла строки, содержащие предложения. Предложение состоит из слов, разделённых пробелами (никаких символов кроме допустимых для слов и пробелов предложение не содержит). Каждое слово – это последовательность символов латинского или русского алфавита, не содержащая пробелов и каких-либо иных символов. Предложения размещены в строках файла. Концом предложения считается символ «точкой с запятой», строка файла может содержать несколько предложений. По полученным строкам и символу **';'** (точка с запятой) построить массив объектов типа **MyStrings**. В выходной файл вывести на каждую строку пары: строки из массива **MyStrings** и соответствующие им аббревиатуры.

Вариант 4

В библиотеке классов объявить класс **MyStrings**

- Поле класса – **string[] _sentences**.
- Конструктор с параметрами **string str** и **char ch**. В конструкторе создаётся массив строк, состоящий из частей строки **str**, разделённых символом **char**. Полученный массив строк связывается со ссылкой **_sentences**
- Свойство **ABBR**, возвращающее ссылку на массив строк. Каждая строка - аббревиатура элемента массива, связанного с **_sentences**. Аббревиатуры строятся по принципу до первой гласной включительно, например, *Software engineering – SoE*.

В основной программе, используя библиотеку классов, получить из файла те строки, которые содержат предложения. Предложение состоит из слов, разделённых пробелами (никаких символов кроме допустимых для слов и пробелов предложение не содержит). Каждое слово – это последовательность символов латинского или русского алфавита, не содержащая пробелов и каких-либо иных символов. Предложения отделены друг от друга символом '.' (точка). По полученной строке и символу-разделителю (точка) необходимо построить массив объектов класса **MyStrings**. В выходной файл вывести на отдельных строках исходную строку входного файла и через двоеточие, полученные из неёстроки массива **MyStrings** с соответствующими им аббревиатурами.

Вариант 5

В библиотеке классов объявить класс **MyDate**

- Поле класса – **DateTime[] _dates**
- Поле класса – **String[] _events**
- Конструктор с параметром **string str**. Конструктор получает на вход строку вида: **YYYY-MM-DD:event_1; YYYY-MM-DD:event_2;…** По данным из строки формируются два равновеликих массива, связанные со ссылками **_dates** и **_events**, соответственно. В массив дат сохраняются строки с датой, в массив событий сохраняются названия событий. Если событие или дата пропущены или указаны в неверном формате, проставляется строка **N/A».** Корректная дата в строке состоит только из цифр и разделителей - дефисов. Корректное событие – последовательность латинских строчных и заглавных символов и пробелов длиной не более 70 символов.
- Открытые свойства для чтения данных массивов дат и событий.

В основной программе, используя библиотеку классов, получить из входного файла строки, содержащие даты и события, заданные в формате, предполагаемом конструктором класса **MyDate**. Данные в файле могут быть расположены как построчно, так и в несколько строк. Создать массив объектов типа **MyDate**. Вывести в выходной файл исходные строки, а также отформатированную и отсортированную по возрастанию дат таблицу дат и событий, с учётом пропущенных значений, т.е. строк с символами **N/A**

Вариант 6

В библиотеке классов объявить класс **MyDate**

- Поле класса - **DateTime[] _dates**
- Поле класса - **String[] _events**
- Конструктор с параметром **string str**. Конструктор получает строку вида: **YYYY/Mon/DD:event_1; YYYY/Mon/DD:event_2;…** По данным строки формируются два равновеликих массива, связанные со ссылками **_dates** и **_events**, соответственно. В массив дат сохраняются полученные из строк даты, в массив событий сохраняются названия событий. Если событие или дата пропущены или указаны в неверном формате, проставляется строка «**N/A».** Корректная дата в строке состоит только из цифр, латинской трёхбуквенной аббревиатуры месяца и символа '/'. Корректное событие – последовательность латинских строчных и заглавных символов и пробелов длиной не более 65 символов. Трёхбуквенные имена месяцев можно найти

по ссылке (https://www.mymathtables.com/playschool-preschool-kindergarten/names-of-twelve-months-table.html)

- Открытые свойства для чтения данных из массивов дат и событий.

В основной программе, используя библиотеку классов, получить из файла строки, содержащие даты и события, заданные в формате, предполагаемом конструктором класса **MyDate**. Создать массив объектов типа **MyDate**. Вывести в выходной файл исходные строки, а также таблицу дат и событий, включая пропущенные значения, т.е. строк с символами **N/A.**

Вариант 7

В библиотеке классов объявить класс **Matrix3x3**

- Поле класса – **int[,] _mtr**, представляющее целочисленную матрицу **3x3.**
- Конструктор с параметром **int[,] newMtr** – ссылкой на целочисленную матрицу. Конструктор связывает ссылку **_mtr** с матрицей, доступной по **newMtr**. Если доступная матрица не размера **3x3** или массив по ссылке отсутствует – конструктор выбрасывает исключение.
- Свойстводоступа к матрице **_mtr**.
- Открытое свойство **Minor**, возвращающее многомерный массив, содержащий матрицу миноров матрицы **_mtr**, доступной по **_mtr**. Минор – определитель матрицы, полученной из исходной, путем вычёркивания из неё **i**-ой строки и **j**-го столбца.
- Закрытый метод вычисления определителя матрицы размера **2x2**

В основной программе, используя библиотеку классов, сформировать массив матриц **3x3**. Заполнить их случайными целыми числами из диапазона [**Min**, **Max**]. **Min**, **Max** – целые числа, необходимо получать из строк входного файла. На основе матриц создавать объекты массива **Matrix3x3[] A**. По объектам из **A**, используя свойство **Minor**, сформировать массив объектов **B** состоящий из значений миноров матриц **A**. Значения объектов **A** и **B** вывести в табличной форме в выходной файл: сначала записываются данные объекта из массива A в формате: элементы матрицы записаны в одну строку, элементы каждоый строки матрицы разделены одним пробелом, строки друг от друга отделены точкой с запятой. Далее в отдельной строке представлены данные о минорах, значения миноров отделены друг от друга одним пробелом. Перед следующей матрицей и ее минорами добавлять пустую строку.

Например, для матрицы $\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix}$, один из миноров: $M_{11} = \begin{vmatrix} a_{00} & a_{02} \\ a_{20} & a_{22} \end{vmatrix}$

Вариант 8

В библиотеке классов объявить класс **Matrix4x4**

- Поле класса **double[,] _mtr** представляет матрицу **4x4.**
- Конструктор с параметром **newMtr** – ссылкой на вещественную матрицу. Конструктор копирует в массив по ссылке **_mtr** элементы, доступные по ссылке **newMtr**. Если матрица **newMtr** не размера **4x4** или массив по ссылке отсутствует– конструктор создаёт объект исключения.
- Свойство доступа к матрице **_mtr**.
- Открытый метод формирования массива, содержащего значения главных миноров матрицы, доступной по **_mtr**. Минор – определитель матрицы, полученной из исходной вычёркиванием её **i**-ой строки и **j**-го столбца. Для главного минора **i = j**.
- Закрытый метод вычисления определителя матрицы **3x3.**

В основной программе, используя библиотеку классов, сформировать массив матриц **4x4**. Заполнить их случайными целыми числами из диапазона [**Min**, **Max**]. **Min**, **Max** – целые числа, необходимо получать из строк входного файла. На основе матриц создавать объекты массива **Matrix4x4[] A**. По объектам из **A**, используя метод формирования массива главных миноров, сформировать массив объектов **B**, состоящий из значений главных миноров матриц **A**. Значения объектов **A** и **B** вывести в табличной форме в выходной файл, точность вывода вещественных значений – два знака после запятой

Например, для матрицы $\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$, один из миноров: $M_{11} = \begin{vmatrix} a_{00} & a_{02} & a_{03} \\ a_{20} & a_{22} & a_{23} \\ a_{30} & a_{32} & a_{33} \end{vmatrix}$

Вариант 9

В библиотеке классов объявить класс **JaggedMatrix**

- Поле **x** – массив вещественных значений
- **arr** – массив ссылок на вещественные одномерные массивы, каждый одномерный массив – значения членов ряда Тейлора, получаемых при разложении функции **sin(x)** в ряд, при заданном значении аргумента **x** (см. метод **SinArray()**)

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

.

- Конструктор с параметрами: **min**, **max**, **n** (количество элементов в массивах **arr** и **x**). Конструктор создаёт массив **x** из **n** случайных вещественных значений из диапазона (**min**, **max**), не включая границы. По значению элемента **x[i]** формируется, с помощью метода **SinArray(),** массив по ссылке **arr[i]**.

- Метод **SinArray()** – закрытый метод, формирующий вещественный массив, содержащий значения элементов разложения функции **sin(x)** в ряд Тейлора в точке **x0** (параметр метода), количество элементов разложения определяется неразличимостью для компьютера добавления очередного члена ряда (машинная точность).
- Метод **AsStrings()** – открытый метод, формирующий массив строк, каждая из которых представляет в символьном виде элементы массива **arr[i]**. Форматирование задавать в формате экспоненты, точность – **3** знака после десятичного разделителя.

В основной программе, используя библиотеку классов, создать объект класса **JaggedMatrix**. Количество элементов **n** во внутреннем массиве **x** и значения **min**, **max** – получать из входного файла. Значения элементов массива массивов **arr** и объекта класса **JaggedMatrix** вывести с выходной файл. Для формирования строк использовать метод **AsStrings()**.

Вариант 10

В библиотеке классов объявить класс **JaggedMatrix**

- Поле **x** – массив вещественных значений
- **Double[][] arr** – каждый массив – значения членов ряда Маклорена, получаемых при разложении функции **cos(x)** в ряд, при заданном значении аргумента **x** (см. метод **CosArray()**)

$$\cos\left(x\right)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}\ldots=\sum_{n=0}^{\infty}\frac{\left(-1\right)^n}{\left(2n\right)!}x^{2n}$$

- Конструктор с параметрами **min**, **max, n** (количество элементов массивов **arr** и **x**). Конструктор создаёт массив **x** из **n** случайных вещественных значений из диапазона (**min**, **max**) не включая границы. По значению элемента **x[i]** формируется, с помощью метода **CosArray(),** каждая ссылка массив по ссылке **arr[i]**.
- Метод **CosArray()** – закрытый метод, формирующий вещественный массив, содержащий значения элементов разложения функции **cos(x)** в ряд Маклорена в точке **x0** (параметр метода), количество элементов разложения определяется неразличимостью для компьютера текущей суммы ряда и полученной на предыдущем шаге вычислений (машинная точность).
- Метод **AsStrings()** – открытый метод, формирующий массив строк, каждая из которых представляет элементы массивов **arr**. Форматирование задавать в формате экспоненты, точность – **3** знака после десятичного разделителя.

В основной программе, используя библиотеку классов, создать объект класса **JaggedMatrix**. Количество элементов **n** во внутреннем массиве **x** и значения **min**, **max** – получать из входного файла. Значения элементов массива массивов **arr** и объекта класса **JaggedMatrix** вывести с выходной файл. Для формирования строк использовать метод **AsStrings()**.

Вариант 11

В библиотеке классов разместить класс **NumbJagged**

- Поле **int[][] jagArr** - зубчатый массив.
- Конструктор с целочисленным неотрицательным параметром **N** – число строк зубчатого массива. Конструктор инициализирует поле **jagArr**. Каждая строка **jagArr** – одномерный массив, состоящий из заранее неизвестного количества случайно выбираемых элементов, каждый элемент лежит в диапазоне [0,5]. Каждая строка последовательно «растет», пока случайным образом не появится нулевое значение, которое становится значением последнего элемента.
- Метод **AsString()** для представления значений элементов строк-массивов из ступенчатого массива (в символьной строке - все значения одной строки).
- Метод **TriangleNumber()** для вычисления количества треугольников, которые можно построить со сторонами из произвольно выбираемых троек значений элементов строки ступенчатого массива с номером, который задан в качестве параметра.

В основной программе, получить из файла значения **N** и определить объекты класса **NumbJagged** из **N** строк разной длины. В выходной файл вывести значения элементов из **NumbJagged**. Для формирования строковых представлений использовать **AsString()**, и количество возможных треугольников для каждой строки массива.

Вариант 12

В библиотеке классов разместить класс **NumbJagged**

- Поле **int[][] jagArr** - ступенчатый массив.
- Конструктор с целочисленным неотрицательным параметром **N** – число строк зубчатого массива. Конструктор инициализирует поле **jagArr**. Каждая строка **jagArr** – одномерный массив из заранее неизвестного количества случайно выбираемых из диапазона [0,5] элементов. Каждая строка последовательно «растет», пока не появится случайное нулевое значение, которое становится значением последнего элемента.
- Метод **StringOut()** для представления значений элементов строк-массивов из ступенчатого массива в виде массива символьных строк (в символьной строке все значения одной строки).
- Метод **MinSquareNumb()** возвращающий значения тех трех элементов заданной параметром строки массива **jagArr**, треугольник с длинами сторон которых имеет максимальную площадь.

В основной программе, получать из файла значения **N** и определить объекты класса **NumbJagged** из **N** строк разной длины. В выходной файл вывести значения элементов из **NumbJagged**, для формирования строковых представлений использовать **AsString()**, и количество стороны треугольника с максимальной площадью.

Вариант 13

Класс **Rectangle** – прямоугольник на плоскости, стороны которого параллельны координатным осям.

- Автореализуемые вещественные свойства для координат левого верхнего и правого нижнего углов прямоугольника..
- Открытые свойства только для чтения – площадь и периметр прямоугольника. Свойства не связаны с конкретными полями и вычисляются при обращении из вызывающего кода.
- Конструктор с четырьмя вещественными параметрами – координатами углов прямоугольника.
- Открытый метод **`Union()`** формирующий объект класса **`Rectangle`**. Объект представляет собой минимальный (описанный) прямоугольник, включающий данный объект-прямоугольник и прямоугольник, переданный в параметре метода.

В основной программе получить из входного файла данные о координатах вершин прямоугольника. Данные о координатах одного прямоугольника расположены в одной строке файла и разделены символом точка с запятой. Пустые строки и пропущенные данные считать некорректными и прямоугольники на их основе не создавать. Создать массив объектов типа **`Rectangle`**. При помощи метода Union() класса **Rectangle** провести объединение прямоугольников (каждый с каждым). В итоговый файл вывести сведения обо всех перебранных парах прямоугольников из массива и о прямоугольниках, полученных при объединении. Данные о паре и результате ее объединения записывать в одну строку, данные о прямоугольнике пишутся в том же формате, что и в во входном файле, между собой данные разделить пробелом дефисом и еще одним пробелом. Формат вывода вещественных значений – два знака после запятой.

Вариант 14

В библиотеке классов описать класс **`Sphere`** – сфера в трёхмерном пространстве.

- Закрытые поля класса: вещественные координаты центра и положительный радиус сферы
- Вещественные свойства доступа для чтения и записи значений полей. Свойство для радиуса должно проверять корректность передаваемых значений и для неверных данных выбрасывать исключение наиболее подходящего типа.
- Конструктор с тремя вещественными параметрами – координатами центра и радиусом сферы. Через конструктор в объект не должны попадать некорректные данные.
- Свойство только для чтения – объём шара, ограниченного сферой ($\frac{4}{3}\pi R^3$). Свойство не связано с полем и вычисляется при обращении к нему из вызывающего кода.
- Открытый метод **`bool IsContain(Sphere newSp)`**, который проверяет полное включение сферы, переданной в качестве параметра, в объект сферы.

В основной программе получить из входного файла данные о координатах и радиусах сфер. Данные о координатах и радиусе одной сферы расположены в одной строке файла и разделены одним пробелом. Пустые строки и пропущенные данные считать некорректными и объекты-сферы на их основе не создавать. Создать массив объектов класса **`Sphere`**. Проанализировать все сферы из массива, разместить в итоговый файл сведения обо всех тех парах сфер, где одна полностью включает вторую. Данные о парах размещать в отдельных строках следующим образом: в одной строке – данные о сферах в формате, заданном исходным файлом, данные сфер разделены точкой с запятой.

Информация о следующей паре – с новой строки. Формат вывода вещественных значений – три знака после запятой.

Вариант 15

В библиотеке классов описать класс **`Book`** – книга в библиотеке.

- Закрытые поля класса: заголовок, автор, наличие для выдачи.
- Строковые свойства для доступа к данным полей. При установке свойств учитывать, что заголовок может быть произвольной строкой, автор записывается в формате Фамилия И.О., отчество может отсутствовать, а фамилия допускает пробелы и дефисы, наличие для выдачи всегда имеет значение и не может быть неопределённым.
- Конструктор с двумя параметрами – заголовком, автором (книга по умолчанию доступна для выдачи); конструктор с тремя параметрами – заголовок, автор и статус готовности к выдаче. Через конструктор в объект не должны попадать некорректные данные.
- Открытый метод **`void Borrow()`**, переводящий книгу в статус «выдана», т.е. изменяющий значение поля «наличие для выдачи». Если книга уже выдана, то метод не должен повторно её выдавать.
- Открытый метод **`void Return()`**, переводящий книгу с статус «готова для выдачи» », т.е. изменяющий значение поля «наличие для выдачи». Если книга уже возвращена, то метод не должен повторно её возвращать.

В основной программе получить из входного файла данные о книгах. Данные о каждой книге расположены на отдельных строках файла. Строки, в которых нарушен формат имени автора считать некорректными; пустые строки и пропущенные данные считать некорректными. Для некорректных данных объекты-книги не создавать. Создать массив объектов класса **`Book`**. Проанализировать все книги, и разместить в итоговый файл данные о книгах с изменёнными на противоположные статусами о выдаче (т.е. было «выдана» - «стало – готова к выдаче»). Информация о каждой книге размещается в отдельной строке файла в формате: Фамилия И.О., Название, статус выдачи.