Завдання № 1

Реалізуйте клас `MyMatrix`, який зберігає прямокутну числову матрицю і вміє виконувати дії додавання матриць та множення матриць.

Матриця повинна містити єдине поле рівня доступу `private` або `protected` — масив із власне елементами матриці (типу `double[,]`).

Слід реалізувати конструктори, оператори, властивості та методи (усі публічні, якщо тільки не вказано іншого). Слід зробити цей клас `partial`, розбивши на два `.cs`:-файли:

Файл `MatrixData.cs`:

- Конструктори:
  - копіюючий з іншого примірника цього самого класу `MyMatrix`;
  - з двовимірного масиву типу `double[,]`;
  - з «зубчастого» масиву `double`-ів, ***якщо*** він фактично прямокутний;
  - з масиву рядків, ***якщо*** фактично ці рядки містять записані через пробіли та/або числа, а кількість цих чисел у різних рядках однакова.
  - з рядка, що містить як пробіли та/або табуляції (їх трактувати як роздільники чисел у одному рядку матриці), так і символи переведення рядка (їх трактувати як роздільники рядків) — ***якщо*** фактичні дані того рядка утворюють прямокутну числову матрицю; зокрема, щоб цим конструктором можна було створити матрицю з рядка, раніше сформованого методом `ToString` (див. далі))
- Властивості (Properties) `Height` та `Width`, що дозволяють взнати (але не дозволяють змінити) «висоту» (кількість рядків) та «ширину» (кількість стовпчиків)
- Java-style getter-и (без setter-ів) кількості рядків `getHeight()` та кількості стовпчиків `getWidth()`
- Індексатори, що дозволяють публічно доступатися до будь-якого окремого елемента матриці (як взнавати його значення, так і змінювати)
- `Java`-style getter та setter для окремого елемента (getter має два параметри — номер рядка і номер стовпчика, setter має три параметра — номер рядка, номер стовпчика, і значення, яке записати у той рядок і стовпчик)
- `override public String ToString()`, який формуватиме (табуляціями та переведеннями рядка) зручне для сприйняття людиною прямокутне подання числової матриці; метод повинен мати саме вищенаведений заголовок, бо саме так вдається забезпечити, щоб примірники цього класу можна було виводити просто через `Console.WriteLine(A)` (де `A` – примірник цього самого класу `MyMatrix`)

Файл `MatrixOperations.cs`:

- `operator +` додавання двох матриць (лише якщо вони мають однаковий розмір)
- `operator *` множення двох матриць (лише якщо кількість стовпчиків першої дорівнює кількості рядків другої)
- Метод (***не*** статичний; `private` або `protected`) `GetTransponedArray()`, що повертає новостворений масив `double[,]` (***не*** `MyMatrix`, а просто масив), у якому вміст елементів транспонований відносно тієї матриці, для якої він викликався
- Метод (***не*** статичний) `MyMatrix GetTransponedCopy()`, який би створював новий примірник `MyMatrix`, у якому вміст матриці транспонований відносно тієї, для якої він викликався; технічну роботу зі власне транспонування не повторювати, а використати результат `GetTransponedArray()`

- Метод (***не*** статичний) `void TransponeMe()`, який би замінював матрицю, для якої викликається, на транспоновану (теж використати `GetTransponedArray()`, але щоб у результаті змінився сам `this`-примірник `MyMatrix`).

В усіх випадках неправильних даних (намагання створити матрицю з фактично не прямокутного «зубчастого» масиву; намагання створити матрицю з масиву рядків, який фактично містить взагалі не числа; намагання доступитися до неіснуючого елемента; намагання додати матриці різного розміру, тощо) пропонується, щоб виконання програми просто аварійно завершувалося. Рятування ситуації шляхом коректної роботи з виключними ситуаціями (exception-ами) – правильний підхід, вітається (хоч і не вимагається). Намагання нівелювати ці помилки будь-якими іншими засобами забороняються.

Частина завдання на бонусні бали; її можна не робити, але якщо робити, то захищати, а просто мовчки здавати, не захищаючи, заборонено (хто так зробить — вважається, що списа(в/ла), бали замість збільшитися стрімко зменшаться):

- Додати у `MatrixOperations.cs` також метод `double CalcDeterminant()` (***не*** статичний), який би обчислював детермінант (визначник), лише для квадратної матриці, не змінюючи вміст матриці. (Зрозуміло, доведеться створити копію масиву і змінювати вміст цієї копії.) Оскільки обчислення детермінанта є вельми повільною операцією, зробити запам'ятоване й закешоване значення визначника, щоб не витрачати багато часу на обчислення вже обчисленого. (Для цього слід порушити другий абзац цієї постановки задачі, де сказано, що масив із власне елементами матриці — єдине поле класу; однак, повне виконання цього бонусного завдання є єдиною дозволеною причиною порушувати ту вимогу.) Додаткову складність дає також те, що матриця не `immutable`, а може змінюватися, тому після будь-якої зміни вмісту матриці не можна користуватися раніше обчисленим значенням детермінанту. Як врахувати це — визначити самостійно, але все це повинно бути поєднаним із розділенням класу (підтримкою його як `partial`), причому з можливістю включати в якийсь проект лише `MatrixData.cs` без `MatrixOperations.cs`

Завдання № 2

Взявши за основу завдання блок 1 лабораторної «структури та рядки–1» за 2-й семестр 1-го курсу (де про структури `my_time` та `my_frac`), реалізуйте клас `MyTime` або `MyFrac` відповідно (розподіл варіантів вказується викладачем окремо), що має описану функціональність. Деталі вимог щодо всяких різних деталей переосмислити самостійно, згідно того, що там просили зробити структуру й не просили дотримуватися ООП-підходу, а тепер треба зробити клас, дотримуючись ООП-підходу.

Ще раз: Ваше завдання для цієї лабораторної з ООП включає в себе необхідність вирішити, де з точки зору ООП слід робити інакше, а де краще просто дотриматися вимог, сформульованих у тій лабораторній для першого курсу.