Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

**Кафедра прикладной математики и кибернетики**

Современные технологии программирования

Лабораторная работа №4

«Разработка и модульное тестирование класса Матрица на C#»

Выполнил: студент 4 курса группы ИП-111

Кузьменок Денис Витальевич

Проверил преподаватель: Зайцев Михаил Георгиевич

Новосибирск, 2024 г.

**Цель:**

Сформировать практические навыки разработки на С# и модульного тестирования классов средствами Visual Studio.

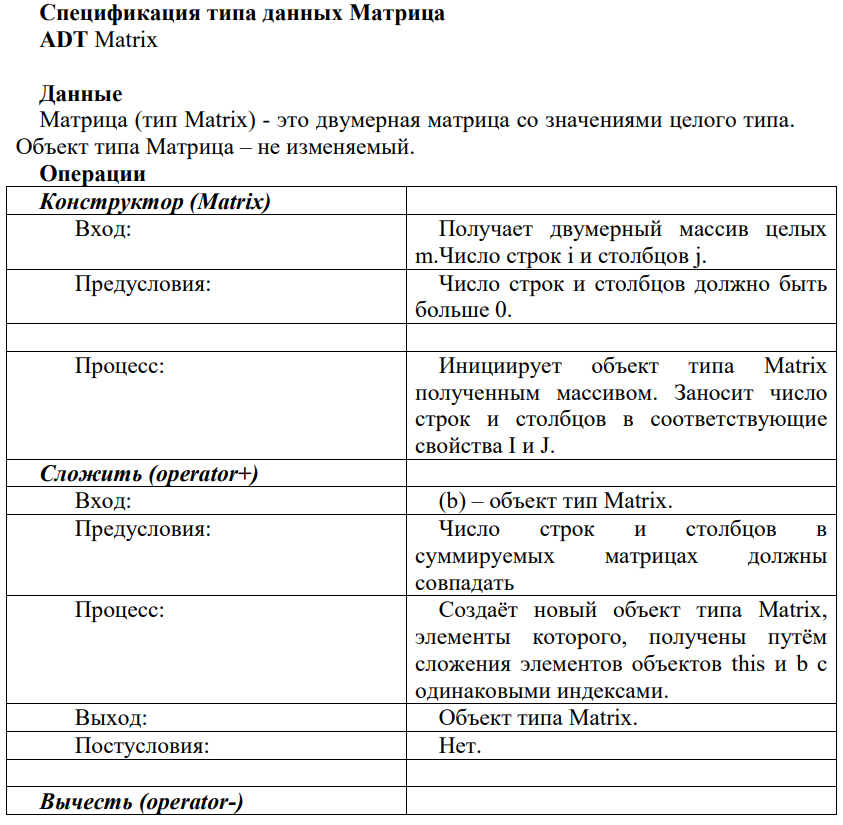
**Задание:**

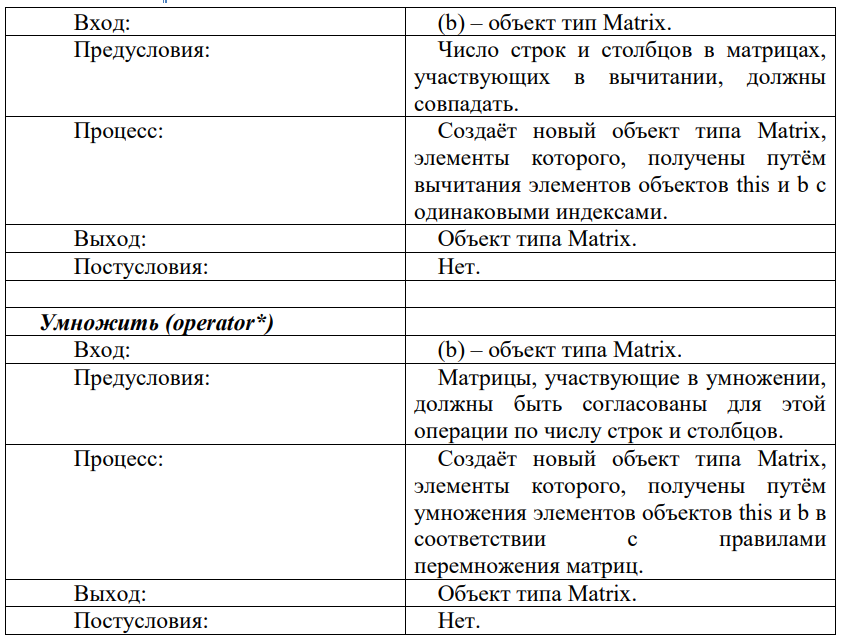
Разработайте класс Матрица (Matrix) для операций матричной алгебры в соответствии с предложенной ниже спецификацией требований.

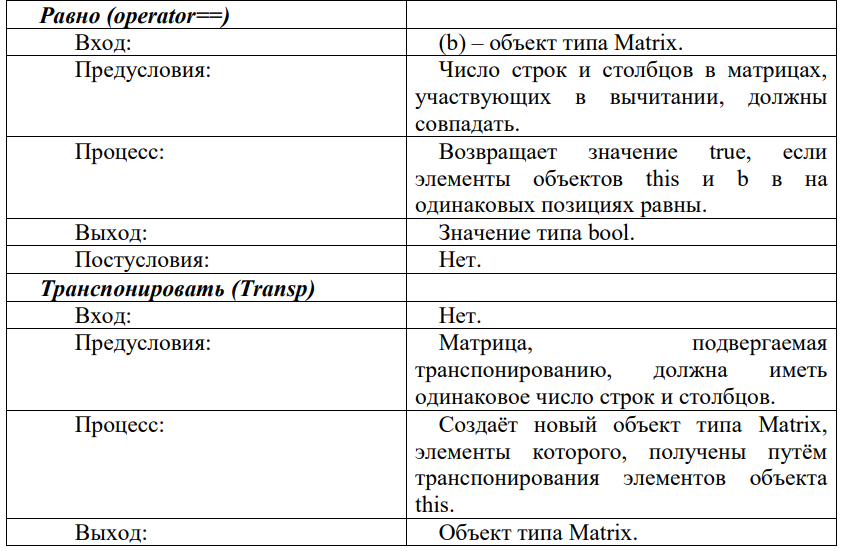
Разработайте тестовые наборы для тестирования методов класса на основе по критерию С2 (путей).

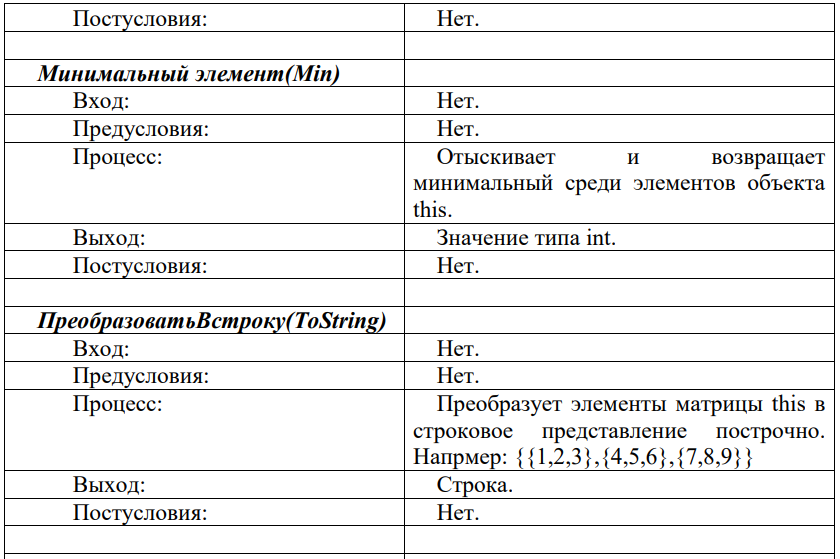
Выполните модульное тестирование класса средствами модульного тестирования Visual Studio.

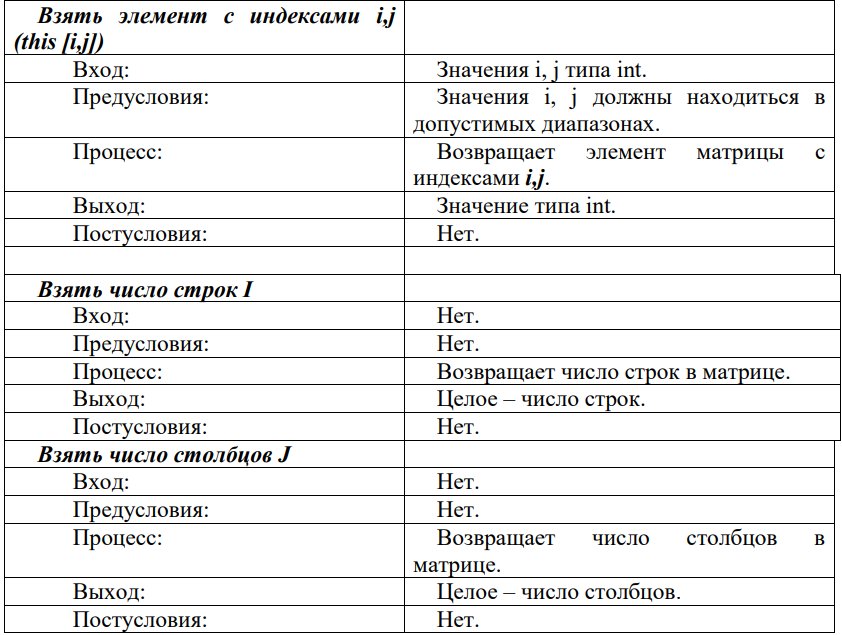
Выполните анализ покрытия кода методов тестами.











**Реализация**

В ходе выполнения лабораторной работы мною были созданы два файла: Program.cs, содержащий в себе точку входа в программу -–функцию main и создание объектов класса Matrix; и Matrix.cs, в котором реализуется новый тип данных Matrix.

Содержащиеся методы:

public Matrix(int i, int j); - Конструктор класса Matrix. Принимает два параметра: количество строк и столбцов. Генерирует исключения MyException, если значения i или j не натуральные. Конструктор инициализирует поля I и J, которые хранят размерность матрицы.

public int this[int i, int j]; - Индексатор, который позволяет обращаться к элементам объектов класса Matrix с помощью квадратных скобок, как к обычным двумерным массивам. Есть два метода: get и set. Оба будут генерировать исключение MyException если значения i или j выходят за допустимые границы. Get позволяет получить элемент m[i, j], а set – записывает переданное значение в элемент m[i, j].

public static Matrix operator+(Matrix a, Matrix b); - Метод принимает два объекта класса типа Matrix. Перегружает оператор “+”, что позволяет работать с типом данных Matrix – его же и будет возвращать метод. Предусмотрена генерация исключения MyException в случае, если размеры матриц a и b не совпадают.

public static Matrix operator-(Matrix a, Matrix b); Метод принимает два объекта класса типа Matrix. Перегружает оператор “-”, что позволяет работать с типом данных Matrix – его же и будет возвращать метод. Предусмотрена генерация исключения MyException в случае, если размеры матриц a и b не совпадают.

public static Matrix operator\*(Matrix a, Matrix b); - Метод принимает два объекта класса типа Matrix. Перегружает оператор “\*”, что позволяет работать с типом данных Matrix – его же и будет возвращать метод. Предусмотрена генерация исключения MyException в случае, если количество столбцов в матрице a не соответствует количеству строк в матрице b – главное условие при умножении матриц.

public void Transp(); - Данный метод будет транспонировать матрицу, которая указывается при вызове.

public int Min(); - Метод ищет в текущей матрице минимальное значение. Возвращает самое маленькое значение элемента, найденного в матрице.

public override string ToString(); - Метод перегружает метод ToString() для работы с объектами типа Matrix. Возвращает строковое представление матрицы.

public int TakeElement(int row, int col); - Метод возвращает элемент в матрице, находящийся по индексу, переданному в метод.

public int CountRows(); - Метод считает и возвращает количество строк в текущей матрице.

public int CountCols(); - Метод считает и возвращает количество столбцов в текущей матрице.

public static bool operator==(Matrix a, Matrix b); - Метод принимает два объекта класса типа Matrix. Перегружает оператор “==”, что позволяет работать с типом данных Matrix. Возвращает тип bool: true – если матрицы равны между собой; false – если есть несоответствие по количеству строк, столбцов или значению элементов.

public static bool operator!=(Matrix a, Matrix b); - Метод принимает два объекта класса типа Matrix. Перегружает оператор “!=”, что позволяет работать с типом данных Matrix. Возвращает тип bool: true – если есть несоответствие по количеству строк, столбцов или значению элементов; false – если матрицы равны между собой.

public void Show(); - Метод выводит на экран текущую матрицу.

public override bool Equals(object obj); - Переопределяется метод Equals, который в данном случае позволяет вернуть значение true, если матрицы равны по количеству строк, столбам и содержащимся элементам (обычный метод Equals возвращал true только в том случае, если два объекта ссылаются на один и тот же экземпляр в памяти).

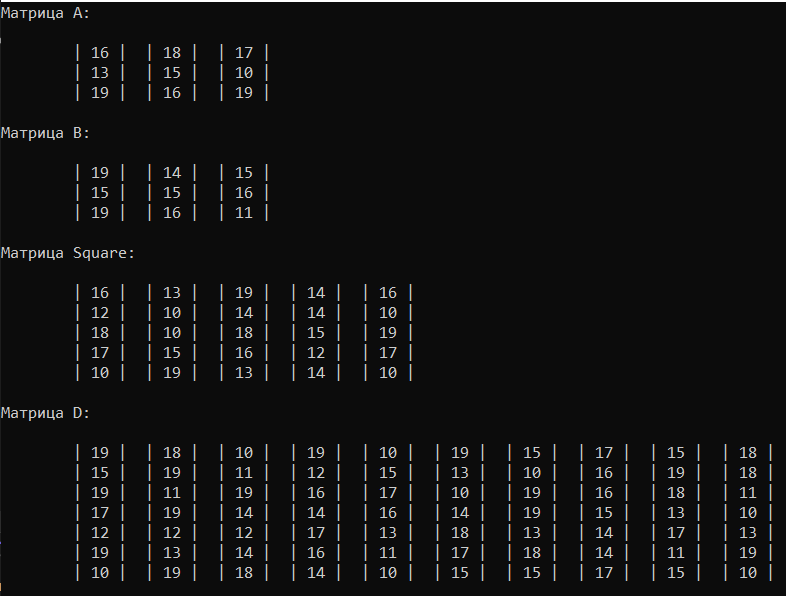


Рис. 1 – Результат запуска программы

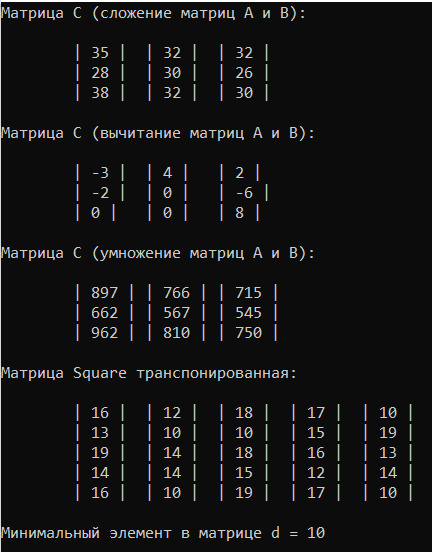


Рис. 2 – Результат запуска программы

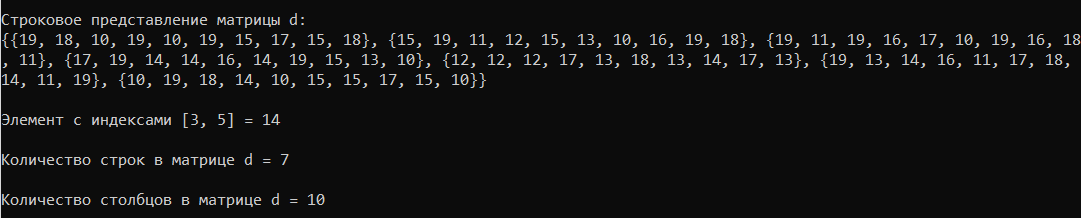
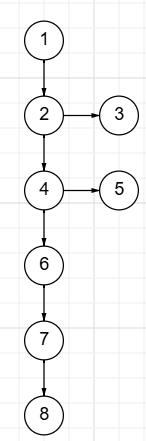


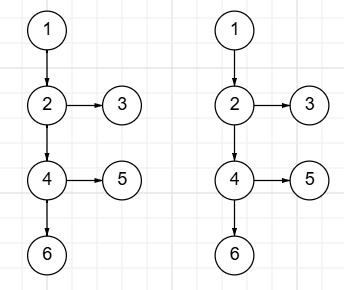
Рис. 3 – Результат запуска программы

В результате написания методов были созданы управляющие графы для каждого из них:

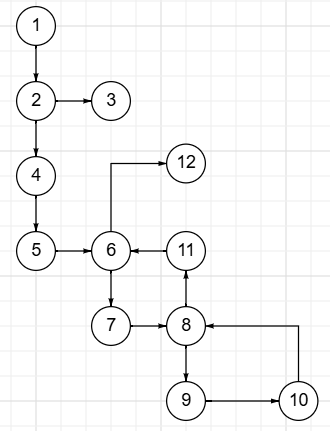
Для public Matrix(int i, int j):



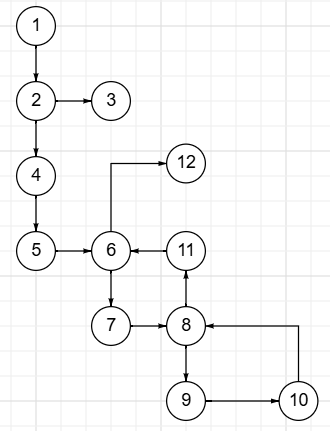
Для public int this[int i, int j] (get и set):



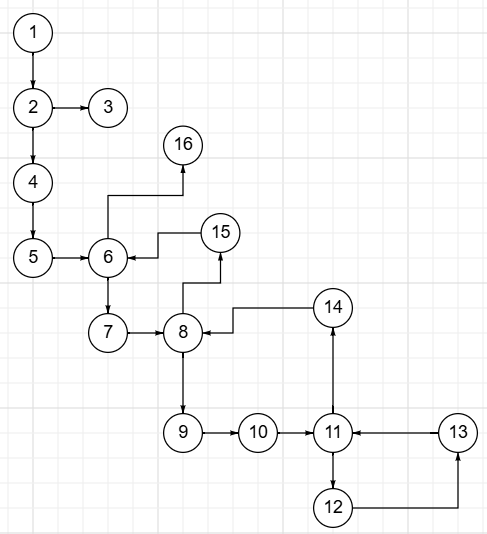
Для public static Matrix operator+(Matrix a, Matrix b):



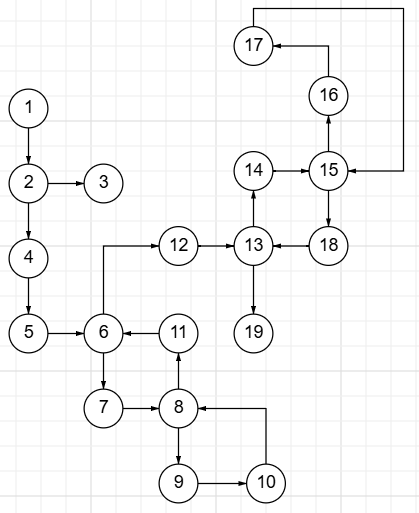
Для public static Matrix operator-(Matrix a, Matrix b):



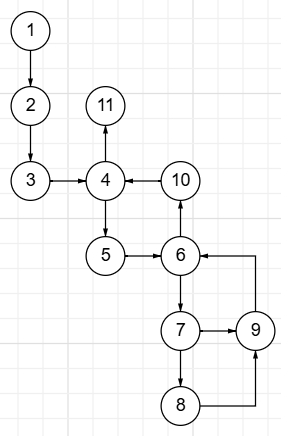
Для public static Matrix operator\*(Matrix a, Matrix b):



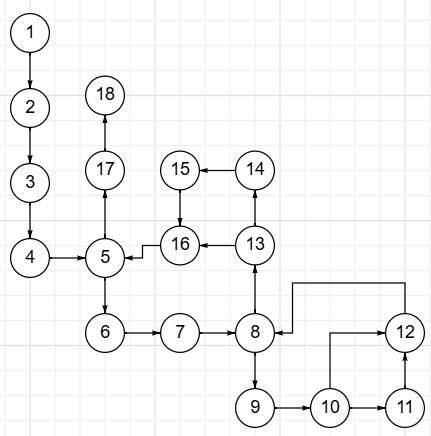
Для public void Transp():



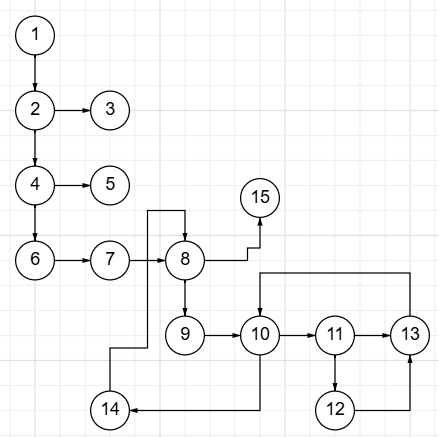
Для public int Min():



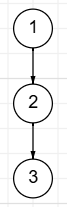
Для public override string ToString():



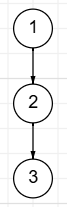
Для public int TakeElement(int row, int col):



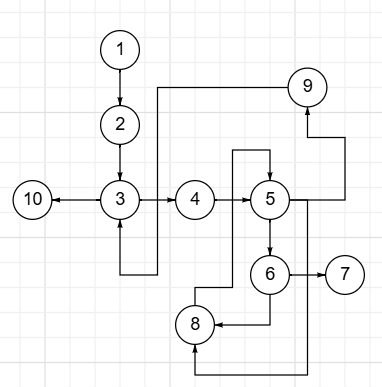
Для public int CountRows():



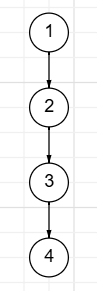
Для public int CountCols():



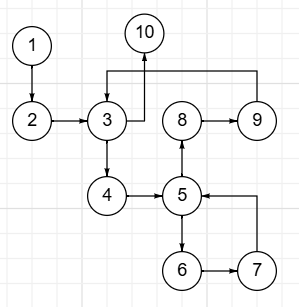
Для public static bool operator==(Matrix a, Matrix b):



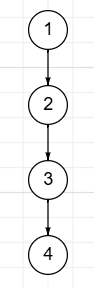
Для public static bool operator!=(Matrix a, Matrix b):



Для public void Show():



Для public override bool Equals(object obj):



1. **Matrix**
2. **Корректные данные:**

* Входные данные: i = 5, j = 2
* Ожидаемый результат: создание объекта Matrix с размерами m[5, 2]

1. **Выход за допустимый диапазон:**

* Входные данные: i = 0, j = 3
* Ожидаемый результат: генерация исключения MyException

1. **get{ }**
2. **Корректные данные:**

* Входные данные: i = 2, j = 2
* Ожидаемый результат: возврат матрицы m[2, 2]

1. **Выход за допустимый диапазон:**

* Входные данные: i = 2, j = -1
* Ожидаемый результат: генерация исключения MyException

1. **set{ }**

**a) Корректные данные:**

* Входные данные: int a = 5, i = 3, j = 2
* Ожидаемый результат: установление в матрице m[3, 2] значения a

1. **Выход за допустимый диапазон:**

* Входные данные: i = 2, j = -1
* Ожидаемый результат: генерация исключения MyException

1. **Matrix operator+**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]], b = [[5, 6], [7, 8]]
* Ожидаемый результат: создание результирующей матрицы c = [[6, 8], [10, 12]]

1. **Матрицы разных размеров:**

* Входные данные: a = [[1, 2]], b = [[5, 6], [7, 8]]
* Ожидаемый результат: генерация исключения MyException

1. **Matrix operator-**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]], b = [[5, 6], [7, 8]]
* Ожидаемый результат: создание результирующей матрицы c = [[-4, -4], [-4, -4]]

1. **Матрицы разных размеров:**

* Входные данные: a = [[1, 2]], b = [[5, 6], [7, 8]]
* Ожидаемый результат: генерация исключения MyException

1. **Matrix operator\***
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4], [5, 6]], b = [[7, 8, 9], [10, 11, 12]]
* Ожидаемый результат: создание результирующей матрицы c = [[27, 30, 33], [61, 68, 75], [95, 106, 117]]

1. **Количество столбцов в первой матрице не равно количеству строк во второй матрице:**

* Входные данные: a = [[1], [2], [3]], b = [[7, 8, 9], [10, 11, 12]]
* Ожидаемый результат: генерация исключения MyException

1. **Transp()**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]]
* Ожидаемый результат: транспонирование матрицы a(transp) = [[1, 3], [2, 4]]

1. **Матрица не квадратная:**

* Входные данные: a = [[1, 2], [3, 4], [5, 6]]
* Ожидаемый результат: генерация исключения MyException

1. **Min()**
2. **Корректные данные:**

* Входные данные: a = [[3, 2], [1, 4]]
* Ожидаемый результат: нахождение и возвращение минимального значения min = 1

1. **ToString()**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]]
* Ожидаемый результат: строка, содержащая в себе изначальную матрицу, записанную в необходимом виде: “{{1, 2}, {3, 4}}”

1. **TakeElement()**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]], i = 1, j = 0
* Ожидаемый результат: поиск и возврат элемента в матрице по данному индексу: result = 3

1. **Выход за допустимый диапазон:**

* Входные данные: a = [[1, 2], [3, 4]], i = -1, j = 1
* Ожидаемый результат: генерация исключения MyException

1. **CountRows()**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]]
* Ожидаемый результат: подсчет и возврат количества строк в матрице: count = 2

1. **CountCols()**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]]
* Ожидаемый результат: подсчет и возврат количества столбцов в матрице: count = 2

1. **Matrix operator==**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]], b = [[1, 2], [3, 4]]
* Ожидаемый результат: true, т.к. матрицы равны по количеству строк, столбцов и значениями элементов

1. **Разные матрицы:**

* Входные данные: a = [[1, 2], [3, 4]], b = [[1, 2]]
* Ожидаемый результат: false, т.к. матрицы различные

1. **Matrix operator!=**
2. **Корректные данные:**

* Входные данные: a = [[1, 2]], b = [[1, 2], [3, 4]]
* Ожидаемый результат: true, т.к. матрицы различные

1. **Одинаковые матрицы:**

* Входные данные: a = [[1, 2], [3, 4]], b = [[1, 2], [3, 4]]
* Ожидаемый результат: false, т.к. матрицы равны по количеству строк, столбцов и значениями элементов

1. **Show()**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]]
* Ожидаемый результат: вывод матрицы на экран

1. **Equals()**
2. **Корректные данные:**

* Входные данные: a = [[1, 2], [3, 4]], b = [[1, 2], [3, 4]]
* Ожидаемый результат: true, т.к. матрицы равны по количеству строк, столбцов и значениями элементов

1. **Разные матрицы:**

* Входные данные: a = [[1, 2]], b = [[1, 2], [3, 4]]
* Ожидаемый результат: false, т.к. матрицы различные

Тестирование по структуре критериев C2 для каждого из методов предполагает:

1. Конструктор Matrix:

* Можно выделить следующие пути: (1, 2, 3), (1, 2, 4, 5), (1, 2, 4, 6, 7, 8). Необходимо будет реализовать три тестовых метода (X, W, Y) = {(i = 0, j = 2, MyException), (i = 2, j = -2, MyException), (i = 2, j = 2, “m[i, j]”)}, чтобы все вышеуказанные покрыть пути.

1. Метод get{ }:

* Можно выделить следующие пути: (1, 2, 3), (1, 2, 4, 5), (1, 2, 4, 6). Необходимо будет реализовать три тестовых метода (X, Y, Z) = {(i = -1, j = 1, MyException), (i = 0, j = -2, MyException), (i = 0, j = 0, “m[0, 0]”)}, чтобы покрыть все указанные пути.

1. Метод set{ }:

* Можно выделить следующие пути: (1, 2, 3), (1, 2, 4, 5), (1, 2, 4, 6). Необходимо будет реализовать три тестовых метода (X, Y, Z) = {(i = -1, j = 1, value = 5, MyException), (i = 0, j = -2, value = 5, MyException), (i = 0, j = 0, value = 5, “m[0, 0] = 5”)}, чтобы покрыть все указанные пути.

1. Метод Matrix operator+:

* Можно выделить следующие пути: (1, 2, 3), (1, 2, 4, 5, 6, 12), (1, 2, 4, 5, 6, 7, 8, 11, 6), (1, 2, 4, 5, 6, 7, 8, 9, 10, 8),. Необходимо будет реализовать два тестовых метода (X, Y) = {(Matrix a = [[1, 2], [3, 4]], Matrix b = [[5, 6], [7, 8]], “Matrix c = [[6, 8], [10, 12]]”), (Matrix a = [[1, 2], [3, 4]], Matrix b = [[5, 6]], MyException)}, чтобы покрыть все указанные пути.

1. Метод Matrix operator-:

* Можно выделить следующие пути: (1, 2, 3), (1, 2, 4, 5, 6, 12), (1, 2, 4, 5, 6, 7, 8, 11, 6), (1, 2, 4, 5, 6, 7, 8, 9, 10, 8),. Необходимо будет реализовать два тестовых метода (X, Y) = {(Matrix a = [[1, 2], [3, 4]], Matrix b = [[5, 6], [7, 8]], “Matrix c = [[-4, -4], [-4, -4]]”), (Matrix a = [[1, 2], [3, 4]], Matrix b = [[5, 6]], MyException)}, чтобы покрыть все указанные пути.

1. Метод Matrix operator\*:

* Можно выделить следующие пути: (1, 2, 3), (1, 2, 4, 5, 6, 16), (1, 2, 4, 5, 6, 7, 8, 15, 6), (1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 11), (1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 14, 8). Необходимо будет реализовать два тестовых метода (X, Y) = {(Matrix a = [[1, 2], [3, 4], [5, 6]], Matrix b = [[7, 8], [9, 10]], “Matrix c = [[25, 28], [57, 64], [89, 100]]”), (Matrix a = [[1, 2], [3, 4]], Matrix b = [[5]], MyException)}, чтобы покрыть все указанные пути.

1. Метод Transp:

* Можно выделить следующие пути: (1, 2, 3), (1, 2, 4, 5, 6, 7, 8, 11, 6), (1, 2, 4, 5, 6, 7, 8, 9, 10, 8), (1, 2, 4, 5, 6, 12, 13, 19), (1, 2, 4, 5, 6, 12, 13, 14, 15, 18, 13), (1, 2, 4, 5, 6, 12, 13, 14, 15, 16, 17, 15). Необходимо будет реализовать два тестовых метода (X, Y) = {(Matrix a = [[1, 2], [3, 4]], “Matrix a = [[1, 3], [2, 4]”), (Matrix a = [[1, 2], [3, 4], [5, 6]], MyException)}, чтобы покрыть все указанные пути.

1. Метод Min:

* Можно выделить следующие пути: (1, 2, 3, 4, 11), (1, 2, 3, 4, 5, 6, 10, 4), (1, 2, 3, 4, 5, 6, 7, 8, 9, 6), (1, 2, 3, 4, 5, 6, 7, 9, 6). Необходимо будет реализовать один тестовый метод (X) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], “1”}, чтобы покрыть все указанные пути.

1. Метод ToString:

* Можно выделить следующие пути: (1, 2, 3, 4, 5, 17, 18), (1, 2, 3, 4, 5, 6, 7, 8, 13, 14, 15, 16, 5), (1, 2, 3, 4, 5, 6, 7, 8, 13, 16, 5), (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 8), (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 8). Необходимо будет реализовать один тестовый метод (X) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], “{{3, 5}, {2, 1}, {6, 4}}”}, чтобы покрыть все указанные пути.

1. Метод TakeElement:

* Можно выделить следующие пути: (1, 2, 3), (1, 2, 4, 5), (1, 2, 4, 6, 7, 8, 15), (1, 2, 4, 6, 7, 8, 9, 10, 14, 8), (1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 10), (1, 2, 4, 6, 7, 8, 9, 10, 11, 13, 10). Необходимо будет реализовать три тестовых метода (X, Y, Z) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], row = 2, col = 1, “4”), (Matrix a = [[3, 5], [2, 1], [6, 4]], row = 3, col = 1, MyException), (Matrix a = [[3, 5], [2, 1], [6, 4]], row = 2, col = -1, MyException)}, чтобы покрыть все указанные пути.

1. Метод CountRows:

* Можно выделить следующие пути: (1, 2, 3). Необходимо будет реализовать один тестовый метод (X) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], “3”)}, чтобы покрыть все указанные пути.

1. Метод CountCols:

* Можно выделить следующие пути: (1, 2, 3). Необходимо будет реализовать один тестовый метод (X) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], “3”)}, чтобы покрыть все указанные пути.

1. Метод bool operator==:

* Можно выделить следующие пути: (1, 2, 3, 10), (1, 2, 3, 4, 5, 6, 7), (1, 2, 3, 4, 5, 6, 8, 5), (1, 2, 3, 4, 5, 8, 5), (1, 2, 3, 4, 5, 9, 3). Необходимо будет реализовать два тестовых метода (X, Y) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], Matrix b = [[3, 5], [2, 1], [6, 4]], “true”), (Matrix a = [[2, 1], [6, 4]], Matrix b = [[3, 5], [2, 1], [6, 4]], “false”)}, чтобы покрыть все указанные пути.

1. Метод bool operator!=:

* Можно выделить следующие пути: (1, 2, 3, 4). Необходимо будет реализовать два тестовых метода (X, Y) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], Matrix b = [[3, 5], [2, 1], [6, 4]], “false”), (Matrix a = [[2, 1], [6, 4]], Matrix b = [[3, 5], [2, 1], [6, 4]], “true”)}, чтобы покрыть все указанные пути.

1. Метод Show:

* Можно выделить следующие пути: (1, 2, 3, 10), (1, 2, 3, 4, 5, 8, 9, 3), (1, 2, 3, 4, 5, 6, 7). Необходимо будет реализовать один тестовый метод (X) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], “| 3 | | 5 |\n| 2 | | 1 |\n| 6 | | 4 |\n”)}, чтобы покрыть все указанные пути.

1. Метод Equals:

* Можно выделить следующие пути: (1, 2, 3, 4). Необходимо будет реализовать два тестовых метода (X, Y) = {(Matrix a = [[3, 5], [2, 1], [6, 4]], Matrix b = [[3, 5], [2, 1], [6, 4]], “true”), (Matrix a = [[2, 1], [6, 4]], Matrix b = [[3, 5], [2, 1], [6, 4]], “false”)}, чтобы покрыть все указанные пути.

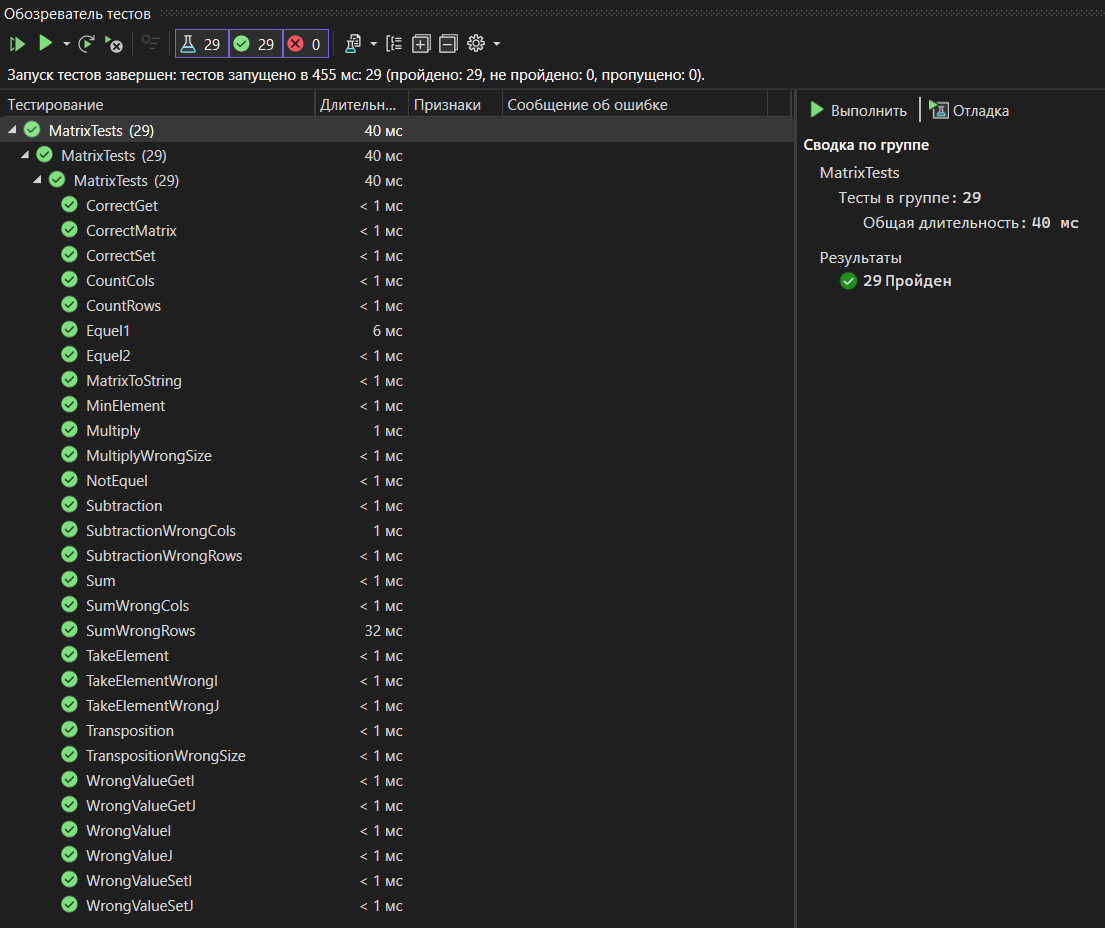


Рис. 4 – Результат выполнения модульных тестов.

**Вывод:**

Разработанный класс Matrix предоставляет набор функционала для выполнения базовых операций матричной алгебры. Использование стратегии тестирования по критерию C2 и проведение модульного тестирования с помощью Visual Studio позволили обеспечить высокое качество кода и уверенность в его корректной работе.

**Листинг программы:**

**Program.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**namespace** **lab4**

{

**class** **Program**

{

**static** **void** Main(string[] args)

{

Random rnd = **new** Random();

**try**

{

Matrix a = **new** Matrix(3, 3);

Matrix b = **new** Matrix(3, 3);

Matrix square = **new** Matrix(5, 5);

Matrix d = **new** Matrix(7, 10);

Matrix c;

**for** (int i = 0; i < a.I; i++)

{

**for** (int j = 0; j < a.J; j++)

{

a[i, j] = rnd.Next(10, 20);

}

}

Console.WriteLine("Матрица A: \n");

a.Show();

**for** (int i = 0; i < b.I; i++)

{

**for** (int j = 0; j < b.J; j++)

{

b[i, j] = rnd.Next(10, 20);

}

}

Console.WriteLine("\nМатрица B: \n");

b.Show();

**for** (int i = 0; i < square.I; i++)

{

**for** (int j = 0; j < square.J; j++)

{

square[i, j] = rnd.Next(10, 20);

}

}

Console.WriteLine("\nМатрица Square: \n");

square.Show();

**for** (int i = 0; i < d.I; i++)

{

**for** (int j = 0; j < d.J; j++)

{

d[i, j] = rnd.Next(10, 20);

}

}

Console.WriteLine("\nМатрица D: \n");

d.Show();

c = a + b;

Console.WriteLine("\nМатрица C (сложение матриц A и B): \n");

c.Show();

c = a - b;

Console.WriteLine("\nМатрица C (вычитание матриц A и B): \n");

c.Show();

c = a \* b;

Console.WriteLine("\nМатрица C (умножение матриц A и B): \n");

c.Show();

Console.WriteLine();

Console.WriteLine("Матрица Square транспонированная: \n");

square.Transp();

square.Show();

int min = d.Min();

Console.WriteLine($"\nМинимальный элемент в матрице d = {min}\n");

string str = d.ToString();

Console.WriteLine($"Строковое представление матрицы d:\n{str}\n");

int[] indexes = { 3, 5 };

int element = d.TakeElement(indexes[0], indexes[1]);

Console.WriteLine($"Элемент с индексами [{indexes[0]}, {indexes[1]}] = {element}\n");

int rows = d.CountRows();

Console.WriteLine($"Количество строк в матрице d = {rows}\n");

int cols = d.CountCols();

Console.WriteLine($"Количество столбцов в матрице d = {cols}\n");

}

**catch** (MyException ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

**Matrix.cs**

**using** **System**;

**using** **System.Collections.Generic**;

**using** **System.Linq**;

**using** **System.Text**;

**using** **System.Threading.Tasks**;

**namespace** **lab4**

{

**public** **class** **MyException** : Exception

{

**public** MyException(string s) : **base**(s) { }

}

**public** **class** **Matrix**

{

int[,] m;

**public** int I { **get**; }

**public** int J { **get**; }

**public** Matrix(int i, int j)

{

**if** (i <= 0) **throw** **new** MyException($"Недопустимое значение строки = {i}");

**if**(j <= 0) **throw** **new** MyException($"Недопустимое значение столбца = {j}");

I = i;

J = j;

m = **new** int[i, j];

}

**public** int **this**[int i, int j]

{

**get**

{

**if** (i < 0 | i > I - 1) **throw** **new** MyException($"Неверное значение i = {i}");

**if** (j < 0 | j > J - 1) **throw** **new** MyException($"неверное значение j = {j}");

**return** m[i, j];

}

**set**

{

**if** (i < 0 | i > I - 1) **throw** **new** MyException($"Неверное значение i = {i}");

**if** (j < 0 | j > J - 1) **throw** **new** MyException($"неверное значение j = {j}");

m[i, j] = **value**;

}

}

**public** **static** Matrix **operator**+(Matrix a, Matrix b)

{

**if** (a.m.GetLength(0) != b.m.GetLength(0) || a.m.GetLength(1) != b.m.GetLength(1))

**throw** **new** MyException("Матрицы должны быть одинакового размера!");

Matrix c = **new** Matrix(a.I, a.J);

**for**(int i = 0; i < a.I; i++)

{

**for**(int j = 0; j < a.J; j++)

{

c[i, j] = a.m[i, j] + b.m[i, j];

}

}

**return** c;

}

**public** **static** Matrix **operator**-(Matrix a, Matrix b)

{

**if** (a.m.GetLength(0) != b.m.GetLength(0) || a.m.GetLength(1) != b.m.GetLength(1))

**throw** **new** MyException("Матрицы должны быть одинакового размера!");

Matrix c = **new** Matrix(a.I, a.J);

**for**(int i = 0; i < a.I; i++)

{

**for**(int j = 0; j < a.J; j++)

{

c[i, j] = a.m[i, j] - b.m[i, j];

}

}

**return** c;

}

**public** **static** Matrix **operator**\*(Matrix a, Matrix b)

{

**if** (a.m.GetLength(1) != b.m.GetLength(0))

**throw** **new** MyException("Количество столбцов первой матрицы должно быть равно количеству строк второй матрицы.");

Matrix c = **new** Matrix(a.I, b.J);

**for**(int i = 0; i < a.I; i++)

{

**for**(int j = 0; j < b.J; j++)

{

c[i, j] = 0;

**for**(int k = 0; k < a.J; k++)

{

c[i, j] += a.m[i, k] \* b.m[k, j];

}

}

}

**return** c;

}

**public** **void** Transp()

{

**if** (I != J)

**throw** **new** MyException("Для транспонирования матрица должна быть квадратной!");

int[,] temp = **new** int[J, I];

**for** (int i = 0; i < I; i++)

{

**for** (int j = 0; j < J; j++)

{

temp[j, i] = **this**[i, j];

}

}

**for** (int i = 0; i < I; i++)

{

**for**(int j = 0; j < J; j++)

{

**this**[i, j] = temp[i, j];

}

}

}

**public** int Min()

{

int min = **this**[0, 0];

**for**(int i = 0; i < I; i++)

{

**for**(int j = 0; j < J; j++)

{

**if** (**this**[i, j] < min)

{

min = **this**[i, j];

}

}

}

**return** min;

}

**public** **override** string ToString()

{

StringBuilder stringBuilder = **new** StringBuilder();

stringBuilder.Append("{");

**for** (int i = 0; i < I; i++)

{

stringBuilder.Append("{");

**for** (int j = 0; j < J; j++)

{

stringBuilder.Append(**this**[i, j]);

**if** (j < J - 1)

stringBuilder.Append(", ");

}

stringBuilder.Append("}");

**if**(i < I - 1)

stringBuilder.Append(", ");

}

stringBuilder.Append("}");

**return** stringBuilder.ToString();

}

**public** int TakeElement(int row, int col)

{

**if** (row < 0 | row > I - 1)

**throw** **new** MyException("i должна находиться в диапазонах размерности матрицы!");

**if** (col < 0 | col > J - 1)

**throw** **new** MyException("j должна находиться в диапазонах размерности матрицы!");

int result = 0;

**for** (int i = 0; i < I; i++)

{

**for** (int j = 0; j < J; j++)

{

**if**(i == row && j == col)

result = **this**[i, j];

}

}

**return** result;

}

**public** int CountRows()

{

int count = m.GetLength(0);

**return** count;

}

**public** int CountCols()

{

int count = m.GetLength(1);

**return** count;

}

**public** **static** bool **operator**==(Matrix a, Matrix b)

{

**for**(int i = 0; i < a.I; i++)

{

**for**(int j = 0; j < a.J; j++)

{

**if** (a[i,j] != b[i, j])

{

**return** **false**;

}

}

}

**return** **true**;

}

**public** **static** bool **operator**!=(Matrix a, Matrix b)

{

**return** !(a==b);

}

**public** **void** Show()

{

**for**(int i = 0; i < I; i++)

{

**for**(int j = 0; j < J; j++)

{

Console.Write("\t" + "| " + **this**[i, j] + " |");

}

Console.WriteLine();

}

}

**public** **override** bool Equals(object obj)

{

**return** (**this** **as** Matrix)==(obj **as** Matrix);

}

}

}

**MatrixTests.cs**

**using** **Microsoft.VisualStudio.TestTools.UnitTesting**;

**using** **System**;

**using** **lab4**;

**namespace** **MatrixTests**

{

[TestClass]

**public** **class** **MatrixTests**

{

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** WrongValueI()

{

*//act*

Matrix a = **new** Matrix(0, 2);

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** WrongValueJ()

{

*//act*

Matrix a = **new** Matrix(2, 0);

}

[TestMethod]

**public** **void** CorrectMatrix()

{

*//act*

Matrix a = **new** Matrix(2, 2);

}

[TestMethod]

**public** **void** CorrectSet()

{

*//act*

Matrix a = **new** Matrix(2, 2);

a[1, 1] = 3;

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** WrongValueSetI()

{

*//act*

Matrix a = **new** Matrix(2, 2);

a[5, 1] = 2;

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** WrongValueSetJ()

{

*//act*

Matrix a = **new** Matrix(2, 2);

a[1, 3] = 2;

}

[TestMethod]

**public** **void** CorrectGet()

{

*//act*

Matrix a = **new** Matrix(2, 2);

int r = a[1, 1];

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** WrongValueGetI()

{

*//act*

Matrix a = **new** Matrix(2, 2);

int r = a[3, 1];

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** WrongValueGetJ()

{

*//act*

Matrix a = **new** Matrix(2, 2);

int r = a[1, 5];

}

[TestMethod]

**public** **void** Sum()

{

*//arrange*

Matrix a = **new** Matrix(2, 2);

a[0, 0] = 1; a[0, 1] = 1; a[1, 0] = 1; a[1, 1] = 1;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 1; b[0, 1] = 1; b[1, 0] = 1; b[1, 1] = 1;

Matrix expected = **new** Matrix(2, 2);

expected[0, 0] = 2; expected[0, 1] = 2; expected[1, 0] = 2; expected[1, 1] = 2;

Matrix actual = **new** Matrix(2, 2);

*//act*

actual = a + b;

*//assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** SumWrongRows()

{

*//arrange*

Matrix a = **new** Matrix(3, 2);

a[0, 0] = 1; a[0, 1] = 1; a[1, 0] = 1; a[1, 1] = 1; a[2, 0] = 1; a[2, 1] = 1;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 1; b[0, 1] = 1; b[1, 0] = 1; b[1, 1] = 1;

Matrix actual = **new** Matrix(2, 2);

*//act*

actual = a + b;

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** SumWrongCols()

{

*//arrange*

Matrix a = **new** Matrix(2, 3);

a[0, 0] = 1; a[0, 1] = 1; a[0, 2] = 1; a[1, 0] = 1; a[1, 1] = 1; a[1, 2] = 1;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 1; b[0, 1] = 1; b[1, 0] = 1; b[1, 1] = 1;

Matrix actual = **new** Matrix(2, 2);

*//act*

actual = a + b;

}

[TestMethod]

**public** **void** Subtraction()

{

*//arrange*

Matrix a = **new** Matrix(2, 2);

a[0, 0] = 5; a[0, 1] = 3; a[1, 0] = 10; a[1, 1] = -7;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 9; b[0, 1] = 3; b[1, 0] = 15; b[1, 1] = -9;

Matrix expected = **new** Matrix(2, 2);

expected[0, 0] = -4; expected[0, 1] = 0; expected[1, 0] = -5; expected[1, 1] = 2;

Matrix actual = **new** Matrix(2, 2);

*//act*

actual = a - b;

*//assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** SubtractionWrongRows()

{

*//arrange*

Matrix a = **new** Matrix(3, 2);

a[0, 0] = 1; a[0, 1] = 1; a[1, 0] = 1; a[1, 1] = 1; a[2, 0] = 1; a[2, 1] = 1;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 1; b[0, 1] = 1; b[1, 0] = 1; b[1, 1] = 1;

Matrix actual = **new** Matrix(2, 2);

*//act*

actual = a - b;

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** SubtractionWrongCols()

{

*//arrange*

Matrix a = **new** Matrix(2, 3);

a[0, 0] = 1; a[0, 1] = 1; a[0, 2] = 1; a[1, 0] = 1; a[1, 1] = 1; a[1, 2] = 1;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 1; b[0, 1] = 1; b[1, 0] = 1; b[1, 1] = 1;

Matrix actual = **new** Matrix(2, 2);

*//act*

actual = a - b;

}

[TestMethod]

**public** **void** Multiply()

{

*//arrange*

Matrix a = **new** Matrix(2, 2);

a[0, 0] = 3; a[0, 1] = 2; a[1, 0] = -6; a[1, 1] = 12;

Matrix b = **new** Matrix(2, 3);

b[0, 0] = 6; b[0, 1] = 21; b[0, 2] = 0; b[1, 0] = -2; b[1, 1] = 4; b[1, 2] = 7;

Matrix expected = **new** Matrix(2, 3);

expected[0, 0] = 14; expected[0, 1] = 71; expected[0, 2] = 14; expected[1, 0] = -60; expected[1, 1] = -78; expected[1, 2] = 84;

Matrix actual = **new** Matrix(2, 3);

*//act*

actual = a \* b;

*//assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** MultiplyWrongSize()

{

*//arrange*

Matrix a = **new** Matrix(2, 3);

a[0, 0] = 3; a[0, 1] = 2; a[0, 2] = 43; a[1, 0] = -6; a[1, 1] = 12; a[1, 2] = 0;

Matrix b = **new** Matrix(2, 3);

b[0, 0] = 6; b[0, 1] = 21; b[0, 2] = 0; b[1, 0] = -2; b[1, 1] = 4; b[1, 2] = 7;

Matrix actual = **new** Matrix(2, 3);

*//act*

actual = a \* b;

}

[TestMethod]

**public** **void** Transposition()

{

*//arrange*

Matrix expected = **new** Matrix(2, 2);

expected[0, 0] = 1; expected[0, 1] = 4;

expected[1, 0] = 2; expected[1, 1] = 5;

Matrix actual = **new** Matrix(2, 2);

actual[0, 0] = 1; actual[0, 1] = 2;

actual[1, 0] = 4; actual[1, 1] = 5;

*//act*

actual.Transp();

*//assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** TranspositionWrongSize()

{

*//arrange*

Matrix a = **new** Matrix(2, 3);

a[0, 0] = 1; a[0, 1] = 2; a[0, 2] = 3;

a[1, 0] = 4; a[1, 1] = 5; a[1, 2] = 6;

*//act*

a.Transp();

}

[TestMethod]

**public** **void** MinElement()

{

*//arrange*

Matrix a = **new** Matrix(3, 2);

a[0, 0] = 33; a[0, 1] = 5;

a[1, 0] = 0; a[1, 1] = -2;

a[2, 0] = 90; a[2, 1] = -13;

int expected = -13;

*//act*

int actual = a.Min();

*//assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

**public** **void** MatrixToString()

{

*//arrange*

Matrix a = **new** Matrix(3, 2);

a[0, 0] = 1; a[0, 1] = 2;

a[1, 0] = 3; a[1, 1] = 4;

a[2, 0] = 5; a[2, 1] = 6;

string expected = "{{1, 2}, {3, 4}, {5, 6}}";

*//act*

string actual = a.ToString();

*//assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

**public** **void** TakeElement()

{

*//arrange*

int row = 2;

int col = 1;

Matrix a = **new** Matrix(3, 3);

a[0, 0] = 1; a[0, 1] = 2; a[0, 2] = 3;

a[1, 0] = 4; a[1, 1] = 5; a[1, 2] = 6;

a[2, 0] = 7; a[2, 1] = 8; a[2, 2] = 9;

int expected = 8;

*//act*

int actual = a.TakeElement(row, col);

*//Assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** TakeElementWrongI()

{

*//arrange*

int row = -1;

int col = 1;

Matrix a = **new** Matrix(3, 3);

a[0, 0] = 1; a[0, 1] = 2; a[0, 2] = 3;

a[1, 0] = 4; a[1, 1] = 5; a[1, 2] = 6;

a[2, 0] = 7; a[2, 1] = 8; a[2, 2] = 9;

*//act*

int actual = a.TakeElement(row, col);

}

[TestMethod]

[ExpectedException(typeof(MyException))]

**public** **void** TakeElementWrongJ()

{

*//arrange*

int row = 2;

int col = 5;

Matrix a = **new** Matrix(3, 3);

a[0, 0] = 1; a[0, 1] = 2; a[0, 2] = 3;

a[1, 0] = 4; a[1, 1] = 5; a[1, 2] = 6;

a[2, 0] = 7; a[2, 1] = 8; a[2, 2] = 9;

*//act*

int actual = a.TakeElement(row, col);

}

[TestMethod]

**public** **void** CountRows()

{

*//arrange*

Matrix a = **new** Matrix(4, 1);

a[0, 0] = 1;

a[1, 0] = 2;

a[2, 0] = 3;

a[3, 0] = 4;

int expected = 4;

*//act*

int actual = a.CountRows();

*//assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

**public** **void** CountCols()

{

*//arrange*

Matrix a = **new** Matrix(4, 1);

a[0, 0] = 1;

a[1, 0] = 2;

a[2, 0] = 3;

a[3, 0] = 4;

int expected = 1;

*//act*

int actual = a.CountCols();

*//assert*

Assert.AreEqual(expected, actual);

}

[TestMethod]

**public** **void** Equel1()

{

*//arrange*

Matrix a = **new** Matrix(2, 2);

a[0, 0] = 1; a[0, 1] = 1; a[1, 0] = 1; a[1, 1] = 1;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 1; b[0, 1] = 1; b[1, 0] = 1; b[1, 1] = 1;

*//act*

bool r = a == b;

*//assert*

Assert.IsTrue(r);

}

[TestMethod]

**public** **void** Equel2()

{

*//arrange*

Matrix a = **new** Matrix(2, 2);

a[0, 0] = 2; a[0, 1] = 1; a[1, 0] = -8; a[1, 1] = 10;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 11; b[0, 1] = 1; b[1, 0] = 0; b[1, 1] = 32;

*//act*

bool r = a == b;

*//assert*

Assert.IsFalse(r);

}

[TestMethod]

**public** **void** NotEquel()

{

*//arrange*

Matrix a = **new** Matrix(2, 2);

a[0, 0] = 2; a[0, 1] = 1; a[1, 0] = -8; a[1, 1] = 10;

Matrix b = **new** Matrix(2, 2);

b[0, 0] = 11; b[0, 1] = 1; b[1, 0] = 0; b[1, 1] = 32;

*//act*

bool r = a != b;

*//assert*

Assert.IsTrue(r);

}

}

}