**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №9

по дисциплине: «Объектно-ориентированное проектирование и   
программирование»

на тему: **«**Обработка исключительных ситуаций**»**

Выполнил: студент гр. ИТИ-21

Мусафиров А.Ю.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2023

**Цель работы**: изучить особенности обработки исключительных ситуаций в объектно-ориентированном языке программирования *С#*, а также дополнить решение задачи из лабораторной №2 обработкой всевозможных исключительных ситуаций и обеспечить надёжное функционирование классов.

**Задание:**

1. Необходимо дополнить решение задачи из лабораторной работы №2 (таблица 2) обработкой всевозможных исключительных ситуаций и обеспечить надёжное функционирование классов.
2. При обработке исключений использовать собственные классы исключений, которые необходимо разместить в отдельной библиотеке классов.
3. При создании классов руководствоваться *Code Convention*.
4. Весь код должен содержать элементы документирования.
5. Разработать модульные тесты для верификации созданных классов.
6. Классы должен быть размещены в библиотеке классов.
7. Модульные тесты – в отдельном проекте.

**Ход работы**

Необходимо разработать тип для работы с матрицами. Реализовать методы, позволяющие выполнять операции умножения матриц, умножение матрицы на число, предусмотрев возможность их выполнение.

На рисунке 1 изображен процесс создания двух матриц и вывод консольного меню с пользователем.

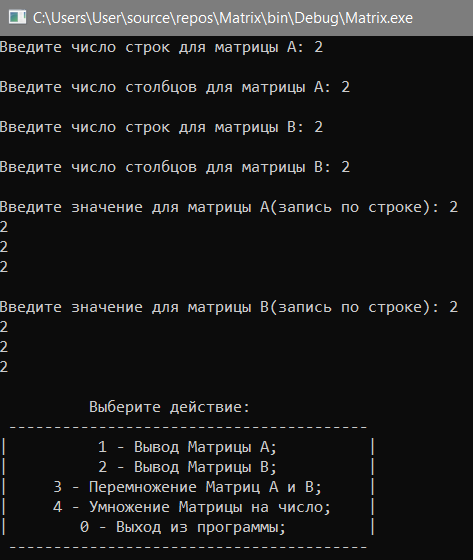


Рисунок 1 – Создание двух матриц и вывод консольного меню.

Уже на рисунке 2 изображен результат ввода некорректного ввода в данные с определённым форматом.



Рисунок 2 – Создание двух матриц и вывод консольного меню.

На выбор даётся 4 действия, которые можно произвести над матрицами. На рисунке 2 уже изображён результат работы программы при нажатии 1 или 2, этот выбор выводит матрицу *A* и *B*.

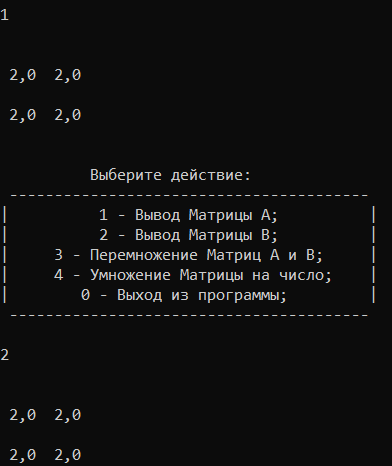


Рисунок 2 – Окно консоли при выводе матрицы *A* или *B*.

Уже на рисунке 3 изображён результат при выборе действия 3, как можно видеть, в консоль выводится результирующая, третья матрица *C*.

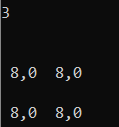


Рисунок 3 – Окно консоли при выборе действия 3 и результат умножения матриц.

На рисунке 4 изображена работа действия 4, как можно заметить на выбор нам даётся три матрицы, которые можно изменить с помощью умножения матрицы на число:

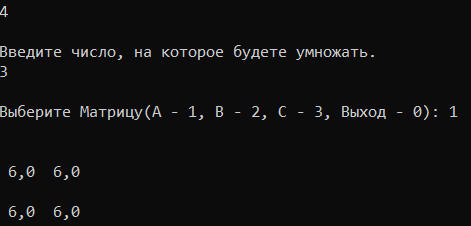


Рисунок 4 – Окно консоли при выборе действия 4 и результат умножения матрицы на число.

В приложении А и Б представлен код программы и код библиотеки.

На рисунке 5 уже изображён результат тестов, разработанных для проверки операторов, созданных внутри класса для матриц:

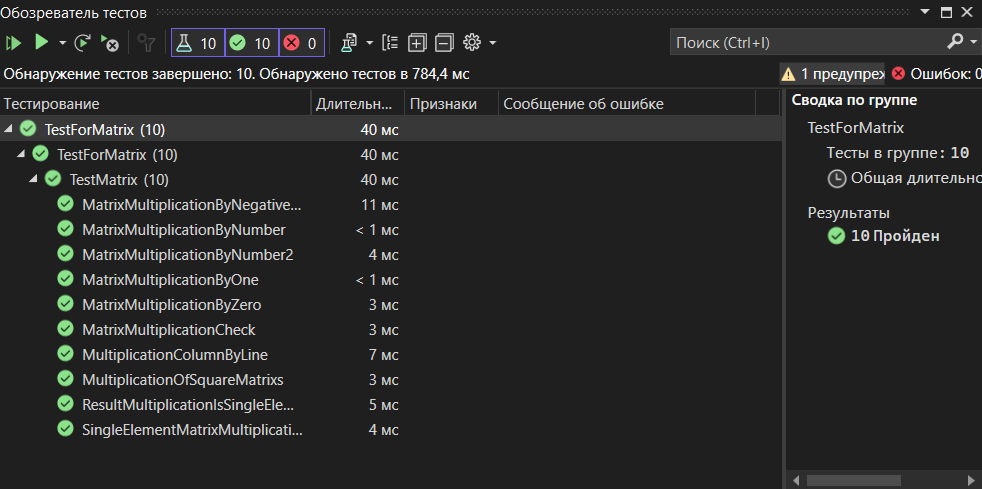


Рисунок 5 – 10 пройденных модульных тестов.

На рисунке 6 показан журнал *Git* с путями:

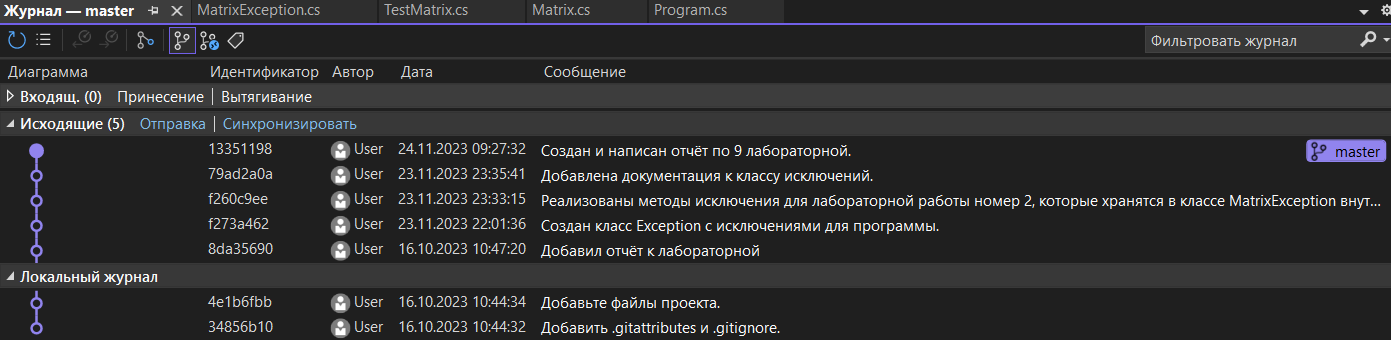


Рисунок 6 – Журнал *Git*.

Так как тесты все верны, то можно сделать вывод, что операторы были разработаны правильно, а именно умножение матрицы на матрицу и матрицу на число. Для умножения матрицы на матрицу прежде, чем их умножить, нужно было проверить равны ли количество столбцов первой матрицы и количество строчек второй, а уже потом по правилу умножения матриц их перемножить.

Уже умножение матрицы на число проще в реализации, чем умножение матриц, так как чтобы умножить число на матрицу, нужно всего лишь умножить каждый элемент на число, которое было заданно.

В приложении В представлен код модульных тестов.

При выполнении данной работы были сделаны следующие вещи:

– создан класс для описания матриц.;

– создана и подключена к основной части библиотека;

– разработаны методы и операции для выполнения поставленных задач;

– организовано консольное меню пользователя;

– сформированы методы исключения для ввода данных;

– выполнены модульные тесты;

**Вывод:** обучились обработке исключительных ситуаций в объектно-ориентированном языке программирования *С#*, а также изучили как разрабатывать свои классы исключительных ситуаций и методы внутри их.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

**Program.cs:**

using System;

using LibraryForMatrix;

namespace Matrix

{

internal class Program

{

static void Main(string[] args)

{

Matrixs matrixA = new Matrixs(), matrixB = new Matrixs(), matrixC = new Matrixs(); // Создание переменных типа Matrixs.

int lineA, columnA, lineB, columnB, choice = 0; // Создание переменных для обозначения строчек и столбцов матриц.

double[,] valuesA = new double[1000, 1000], valuesB = new double[1000, 1000]; // Создание двумерных массивов для обозначения элементов матриц.

bool inputValid = false;

while (!inputValid)

{

try

{

Console.Write("Введите число строк для матрицы А: ");

string inputLineA = Console.ReadLine();

MatrixException.ValidateInt(inputLineA, out lineA);

Console.Write("\nВведите число столбцов для матрицы А: ");

string inputColumnA = Console.ReadLine();

MatrixException.ValidateInt(inputColumnA, out columnA);

Console.Write("\nВведите число строк для матрицы B: ");

string inputLineB = Console.ReadLine();

MatrixException.ValidateInt(inputLineB, out lineB);

Console.Write("\nВведите число столбцов для матрицы B: ");

string inputColumnB = Console.ReadLine();

MatrixException.ValidateInt(inputColumnB, out columnB);

Console.Write("\nВведите значение для матрицы A(запись по строке): ");

valuesA = Program.InputMatrix(lineA, columnA);

Console.Write("\nВведите значение для матрицы B(запись по строке): ");

valuesB = Program.InputMatrix(lineB, columnB);

matrixA = new Matrixs(lineA, columnA, valuesA); // Метод создания матрицы из полученной информации.

matrixB = new Matrixs(lineB, columnB, valuesB);

inputValid = true;

}

catch (MatrixException ex)

{

Console.Clear();

Console.WriteLine($"Ошибочный ввод (Код {ex.ErrorCode}): {ex.Message}");

}

}

bool isRun = true; // Присваивание переменной IsRun значения true.

while (isRun) // Консольное меню.

{

Console.WriteLine("\n Выберите действие:");

Console.WriteLine(" ----------------------------------------");

Console.WriteLine("| 1 - Вывод Матрицы A; |");

Console.WriteLine("| 2 - Вывод Матрицы B; |");

Console.WriteLine("| 3 - Перемножение Матриц A и B; |");

Console.WriteLine("| 4 - Умножение Матрицы на число; |");

Console.WriteLine("| 0 - Выход из программы; |");

Console.WriteLine(" ----------------------------------------\n");

bool ValidateChoice = false;

while (!ValidateChoice)

{

try

{

string inputChoice = Console.ReadLine();

MatrixException.ValidateChoice(inputChoice, out choice);

ValidateChoice = true;

}

catch (MatrixException ex)

{

Console.WriteLine($"Ошибочный ввод (Код {ex.ErrorCode}): {ex.Message}");

}

}

switch (choice)

{

case 1:

Console.WriteLine("\n");

Program.OutputMatrix(matrixA); // Метод вывода матрицы А.

break;

case 2:

Console.WriteLine("\n");

Program.OutputMatrix(matrixB); // Метод вывода матрицы Б.

break;

case 3:

Console.WriteLine("\n");

matrixC = matrixA \* matrixB; // Операция умножения матриц.

if (matrixC.Line != 0)

{

Program.OutputMatrix(matrixC);

}

else

{

Console.WriteLine("Умножение не возможно.");

}

break;

case 4:

bool input = true;

double number = 0;

Console.WriteLine("\nВведите число, на которое будете умножать.");

bool ValidateNumber = false;

while (!ValidateNumber)

{

try

{

string inputNumber = Console.ReadLine();

MatrixException.ValidateDouble(inputNumber, out number);

ValidateNumber = true;

}

catch (MatrixException ex)

{

Console.WriteLine($"Ошибочный ввод (Код {ex.ErrorCode}): {ex.Message}");

}

}

while (input)

{

Console.Write("\nВыберите Матрицу(A - 1, B - 2, C - 3, Выход - 0): ");

ValidateChoice = false;

while (!ValidateChoice)

{

try

{

string inputChoice = Console.ReadLine();

MatrixException.ValidateChoice(inputChoice, out choice);

ValidateChoice = true;

}

catch (MatrixException ex)

{

Console.WriteLine($"Ошибочный ввод (Код {ex.ErrorCode}): {ex.Message}");

}

}

switch (choice)

{

case 1:

Console.WriteLine("\n");

matrixA = number \* matrixA; // Операция умножения матрицы на число.

Program.OutputMatrix(matrixA);

input = false;

break;

case 2:

Console.WriteLine("\n");

matrixB = number \* matrixB;

Program.OutputMatrix(matrixB);

input = false;

break;

case 3:

Console.WriteLine("\n");

matrixC = number \* matrixC;

Program.OutputMatrix(matrixC);

input = false;

break;

case 0:

input = false; // Выход из программы.

break;

default:

Console.WriteLine("Некорректный выбор функции!");

break;

}

}

break;

case 0:

isRun = false; // Выход из программы.

break;

default:

Console.WriteLine("Некорректный выбор функции!");

break;

}

}

}

/// <summary>

/// Метод ввода матрицы из консоли.

/// </summary>

/// <param name="line"></param>

/// <param name="column"></param>

/// <returns></returns>

public static double[,] InputMatrix(int line, int column)

{

int i = 0;

double[,] value = new double[1000, 1000];

while (i < line)

{

int j = 0;

while (j < column)

{

string inputValue = Console.ReadLine();

MatrixException.ValidateDouble(inputValue, out value[i,j]);

j++;

}

i++;

}

return value;

}

/// <summary>

/// Метод вывода матрицы в консоль.

/// </summary>

/// <param name="A"></param>

public static void OutputMatrix(Matrixs A)

{

int i = 0;

while (i < A.Line)

{

int j = 0;

while (j < A.Column)

{

Console.Write(String.Format("{0,4:0.0}", A.Value[i, j]));

Console.Write(" ");

j++;

}

Console.WriteLine("\n");

i++;

}

}

}

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Тексты библиотеки**

**Matrix.cs:**

using System;

namespace LibraryForMatrix

{

/// <summary>

/// Класс, в котором находятся данные косающиеся матрицы, методы и операции, совершаемые над матрицами.

/// </summary>

public class Matrixs

{

public int Line;

public int Column;

public double[,] Value = new double[1000, 1000];

/// <summary>

/// Конструктор пустой матрицы.

/// </summary>

public Matrixs()

{

this.Line = 0;

this.Column = 0;

this.Value[Line, Column] = 0;

}

/// <summary>

/// Конструктор матрицы по вводимым данным.

/// </summary>

/// <param name="line"></param>

/// <param name="column"></param>

/// <param name="values"></param>

public Matrixs(int line, int column, double[,] values)

{

this.Line = line;

this.Column = column;

int stepLine = 0, stepColumn = 0;

while (stepLine < line)

{

stepColumn = 0;

while (stepColumn < column)

{

Value[stepLine, stepColumn] = values[stepLine, stepColumn];

stepColumn++;

}

stepLine++;

}

}

/// <summary>

/// Операция умножения матрицы на матрицу.

/// </summary>

/// <param name="A"></param>

/// <param name="B"></param>

/// <returns></returns>

public static Matrixs operator \*(Matrixs A, Matrixs B)

{

if (A.Column == B.Line)

{

int stepLineA = 0, stepColumnB = 0;

Matrixs C = new Matrixs();

C.Line = A.Line;

C.Column = B.Column;

while (stepLineA < A.Line)

{

stepColumnB = 0;

while (stepColumnB < B.Column)

{

int stepColumnA = 0, stepLineB = 0;

while (stepColumnA < A.Column)

{

C.Value[stepLineA, stepColumnB] = C.Value[stepLineA, stepColumnB] + A.Value[stepLineA, stepColumnA] \* B.Value[stepLineB, stepColumnB];

stepColumnA++;

stepLineB++;

}

stepColumnB++;

}

stepLineA++;

}

return C;

}

Matrixs D = new Matrixs();

return D;

}

/// <summary>

/// Операция умножения матрицы на число.

/// </summary>

/// <param name="number"></param>

/// <param name="A"></param>

/// <returns></returns>

public static Matrixs operator \*(double number, Matrixs A)

{

int stepLine = 0, stepColumn = 0;

Matrixs C = new Matrixs();

C.Line = A.Line;

C.Column = A.Column;

while (stepLine < A.Line)

{

stepColumn = 0;

while (stepColumn < A.Column)

{

C.Value[stepLine, stepColumn] = A.Value[stepLine, stepColumn] \* number;

stepColumn++;

}

stepLine++;

}

return C;

}

}

}

**MatrixException.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace LibraryForMatrix

{

/// <summary>

/// Класс с методами исключения.

/// </summary>

public class MatrixException : Exception

{

public int ErrorCode { get; }

public MatrixException(string message, int errorCode) : base(message)

{

ErrorCode = errorCode;

}

/// <summary>

/// Метод исключения для переменных типа Double.

/// </summary>

/// <param name="input"></param>

/// <param name="result"></param>

/// <exception cref="MatrixException"></exception>

public static void ValidateDouble(string input, out double result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new MatrixException("Строка пуста!", 1);

}

if (!double.TryParse(input, out result))

{

throw new MatrixException("Строка содержит символ или текст!", 2);

}

}

/// <summary>

/// Метод исключения для переменных тип Int.

/// </summary>

/// <param name="input"></param>

/// <param name="result"></param>

/// <exception cref="MatrixException"></exception>

public static void ValidateInt(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new MatrixException("Строка пуста!", 1);

}

if (!Int32.TryParse(input, out result))

{

throw new MatrixException("Строка содержит вещественное число или текст!", 2);

}

}

/// <summary>

/// Метод исключения для значений выбора в меню.

/// </summary>

/// <param name="input"></param>

/// <param name="result"></param>

/// <exception cref="MatrixException"></exception>

public static void ValidateChoice(string input, out int result)

{

if (string.IsNullOrWhiteSpace(input))

{

throw new MatrixException("Строка пуста!\n", 1);

}

if (!Int32.TryParse(input, out result))

{

throw new MatrixException("Строка содержит вещественное число или текст!\n", 2);

}

}

}

}

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Текст тестов**

**TestMatrix.cs:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using LibraryForMatrix;

namespace TestForMatrix

{

/// <summary>

/// Тестовый класс, в котором хранятся модульные тесты.

/// </summary>

[TestClass]

public class TestMatrix

{

/// <summary>

/// Метод сравнения матриц между собой.

/// </summary>

/// <param name="A"></param>

/// <param name="B"></param>

/// <returns></returns>

public static bool CompareMatrixs(Matrixs A, Matrixs B)

{

if (A.Line != B.Line || A.Column != B.Column)

{

return false;

}

int i = 0;

while (i < A.Line)

{

int j = 0;

while (j < A.Column)

{

if (A.Value[i, j] != B.Value[i, j])

{

return false;

}

j++;

}

i++;

}

return true;

}

/// <summary>

/// Тест, проверяющий умножения одноэлементной матрицы.

/// </summary>

[TestMethod]

public void SingleElementMatrixMultiplication()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 1;

matrix1.Column = 1;

matrix1.Value[0, 0] = 3;

Matrixs matrix2 = new Matrixs();

matrix2.Line = 1;

matrix2.Column = 3;

matrix2.Value = new double[1, 3] { { 1, 2, 3 } };

Matrixs matrix3 = matrix1 \* matrix2;

Matrixs result = new Matrixs();

result.Line = 1;

result.Column = 3;

result.Value = new double[1, 3] { { 3, 6, 9 } };

Assert.IsTrue(CompareMatrixs(result, matrix3));

}

/// <summary>

/// Тест, проверяющий умножение матриц, в результате которого получается одноэлементная матрица.

/// </summary>

[TestMethod]

public void ResultMultiplicationIsSingleElementMatrix()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 1;

matrix1.Column = 3;

matrix1.Value = new double[1, 3] { { 1, 2, 3 } };

Matrixs matrix2 = new Matrixs();

matrix2.Line = 3;

matrix2.Column = 1;

matrix2.Value = new double[3, 1] { { 1 }, { 2 }, { 3 } };

Matrixs matrix3 = matrix1 \* matrix2;

Matrixs result = new Matrixs();

result.Line = 1;

result.Column = 1;

result.Value[0, 0] = 14;

Assert.IsTrue(CompareMatrixs(result, matrix3));

}

/// <summary>

/// Тест, проверяющий умножение квадратных матриц.

/// </summary>

[TestMethod]

public void MultiplicationOfSquareMatrixs()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1.Value = new double[2, 2] { { 2, 3 }, { 3, 2 } };

Matrixs matrix2 = new Matrixs();

matrix2.Line = 2;

matrix2.Column = 2;

matrix2.Value = new double[2, 2] { { 1, 2 }, { 3, 2 } };

Matrixs matrix3 = matrix1 \* matrix2;

Matrixs result = new Matrixs();

result.Line = 2;

result.Column = 2;

result.Value = new double[2, 2] { { 11, 10 }, { 9, 10 } };

Assert.IsTrue(CompareMatrixs(result, matrix3));

}

/// <summary>

/// Тест, проверяющий умножение столбца на строчку.

/// </summary>

[TestMethod]

public void MultiplicationColumnByLine()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 1;

matrix1.Column = 2;

matrix1.Value = new double[1, 2] { { 1, 2 } };

Matrixs matrix2 = new Matrixs();

matrix2.Line = 2;

matrix2.Column = 1;

matrix2.Value = new double[2, 1] { { 3 }, { 2 } };

Matrixs matrix3 = matrix2 \* matrix1;

Matrixs result = new Matrixs();

result.Line = 2;

result.Column = 2;

result.Value = new double[2, 2] { { 3, 6 }, { 2, 4 } };

Assert.IsTrue(CompareMatrixs(result, matrix3));

}

/// <summary>

/// Тест, проверяющий возможность умножения матриц.

/// </summary>

[TestMethod]

public void MatrixMultiplicationCheck()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 1;

matrix1.Column = 1;

matrix1.Value[0, 0] = 1;

Matrixs matrix2 = new Matrixs();

matrix2.Line = 2;

matrix2.Column = 1;

matrix2.Value = new double[2, 1] { { 1 }, { 2 } };

Matrixs matrix3 = matrix1 \* matrix2;

Matrixs result = new Matrixs();

Assert.IsTrue(CompareMatrixs(result, matrix3));

}

/// <summary>

/// Тест, проверяющий умножение матрицы на число.

/// </summary>

[TestMethod]

public void MatrixMultiplicationByNumber()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1.Value = new double[2, 2] { { 2, 2 }, { 2, 2 } };

double number = 5;

Matrixs matrix2 = number \* matrix1;

Matrixs result = new Matrixs();

result.Line = 2;

result.Column = 2;

result.Value = new double[2, 2] { { 10, 10 }, { 10, 10 } };

Assert.IsTrue(CompareMatrixs(result, matrix2));

}

/// <summary>

/// Тест, проверяющий умножение матрицу на вещественное число.

/// </summary>

[TestMethod]

public void MatrixMultiplicationByNumber2()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1.Value = new double[2, 2] { { 5, 5 }, { 5, 5 } };

double number = 2.5;

Matrixs matrix2 = number \* matrix1;

Matrixs result = new Matrixs();

result.Line = 2;

result.Column = 2;

result.Value = new double[2, 2] { { 12.5, 12.5 }, { 12.5, 12.5 } };

Assert.IsTrue(CompareMatrixs(result, matrix2));

}

/// <summary>

/// Тест, проверяющий умножение матрицы на единицу.

/// </summary>

[TestMethod]

public void MatrixMultiplicationByOne()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 3;

matrix1.Column = 3;

matrix1.Value = new double[3, 3] { { 3, 3, 3 }, { 3, 3, 3 }, { 3, 3, 3 } };

double number = 1;

Matrixs matrix2 = number \* matrix1;

Matrixs result = new Matrixs();

result.Line = 3;

result.Column = 3;

result.Value = new double[3, 3] { { 3, 3, 3 }, { 3, 3, 3 }, { 3, 3, 3 } };

Assert.IsTrue(CompareMatrixs(result, matrix2));

}

/// <summary>

/// Тест, проверяющий умножение матрицы на отрицательное число.

/// </summary>

[TestMethod]

public void MatrixMultiplicationByNegativeNumber()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1.Value = new double[2, 2] { { 2, 2 }, { 2, 2 } };

double number = -3;

Matrixs matrix2 = number \* matrix1;

Matrixs result = new Matrixs();

result.Line = 2;

result.Column = 2;

result.Value = new double[2, 2] { { -6, -6 }, { -6, -6 } };

Assert.IsTrue(CompareMatrixs(result, matrix2));

}

/// <summary>

/// Тест, проверяющий умножение матрицы на ноль.

/// </summary>

[TestMethod]

public void MatrixMultiplicationByZero()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1.Value = new double[2, 2] { { 2, 2 }, { -2, 2 } };

double number = 0;

Matrixs matrix2 = number \* matrix1;

Matrixs result = new Matrixs();

result.Line = 2;

result.Column = 2;

result.Value = new double[2, 2] { { 0, 0 }, { 0, 0 } };

}

}

}