**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 №3

по дисциплине: «Объектно-ориентированное программирование»

на тему: **«**Операции и перегруженные методы класса.**»**

Выполнил: студент гр. ИТИ-21

Мусафиров А.Ю.

Принял: преподаватель

Башаримов Ю. С.

Гомель 2023

**Цель работы**: научится разрабатывать классы для решения задач и модульные тесты для тестирования созданного класса; уметь выполнять перегрузку операций и операторов, реализовывать конструкторы с параметрами.

**Задание:**

1. Необходимо решить задачу, согласно варианту (Вариант 2).
2. При создании классов руководствоваться Code Convention.
3. Весь код должен содержать элементы документирования.
4. В реализованных классах должны быть конструкторы с параметрами и по умолчанию, а также необходимые свойства и методы.
5. Для сравнения объектов использовать перегруженный оператор отношения.
6. Разработать модульные тесты для верификации созданных классов.
7. Все классы должны быть размещены в библиотеке классов.
8. Модульные тесты – в отдельном проекте.
9. В отдельном проекте реализовать консольный интерфейс.

**Ход работы**

Необходимо создать класс «Матрица», в котором описать закрытые поля – матрица чисел и имя матрицы, свойства для определения количества строк и столбцов массива, индексатор для доступа к элементами поля-массива, конструктор с параметрами, метод ввода и вывода матрицы, перегруженные методы для вычисления произведения отрицательных элементов матрицы с выводом результата (параметр – имя матрицы) и без вывода, операция поэлементного сложения матрицы одинаковой размерности, операция true и false (массив истинный, если в нём есть ненулевой элемент).

На рисунке 1 изображен процесс создания матрицы при нажатии 1 и вывод консольного меню с пользователем.

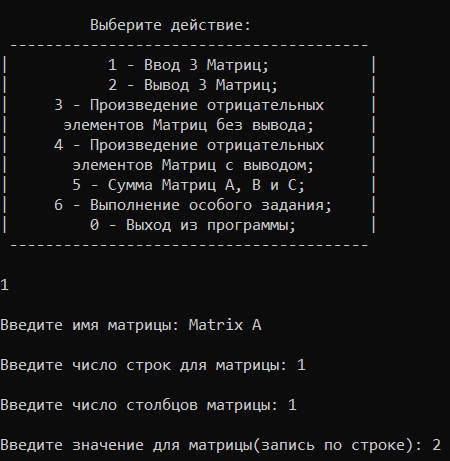


Рисунок 1 – Пример создания матрицы и вывод консольного меню.

На выбор даётся 6 действия, которые можно произвести над матрицами. На рисунке 2 уже изображён результат работы программы при нажатии 2, этот выбор выводит три матрицы.

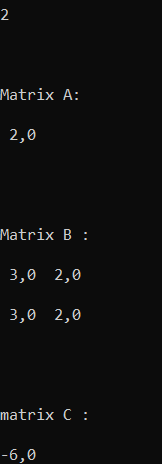


Рисунок 2 – Окно консоли при выводе трёх матриц.

Уже на рисунке 3 изображён результат при выборе действия 3, как можно видеть, в консоль выводится сообщение о том, что произведения отрицательных элементов подсчитаны, но так как этот выбор без вывода результата, узнать, что он посчитал нельзя, однако, можно узнать при создании модульных тестов:



Рисунок 3 – Окно консоли при выборе действия 3.

На рисунке 4 изображена работа действия 4, уже при выполнении этого действия можно оценить правильность его подсчётов, так как оно выводит результат произведения для каждой матрицы:

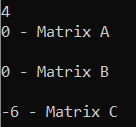


Рисунок 4 – Окно консоли при выборе действия 4 и результат умножения отрицательных элементов матрицы.

На рисунке пять представлен результат действия 5, а именно сложение одинаковых по размерности матриц:

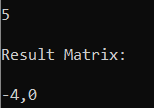


Рисунок 5 – Окно консоли при выборе действия 5 и результат сложения матриц.

На рисунке 6 изображена работа программы для особого задания. Задание заключается в том, чтобы, в случае если произведение отрицательных элементов первого массива больше заданного числа, а в третьей матрице есть ненулевые элементы, увеличить все отрицательные элементы этого массива на значение минимального среди элементов последней строки третьего массива:

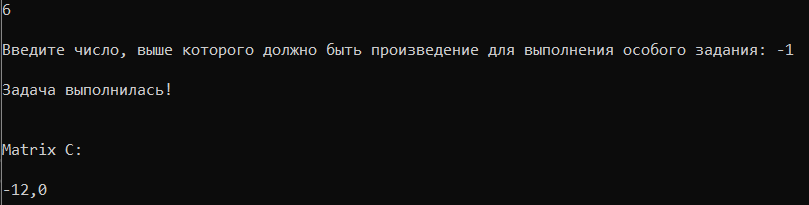


Рисунок 6 – Окно консоли при выборе действия 6 и результат выполнения особого задания.

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

На рисунке 7 уже изображён результат тестов, разработанных для проверки операторов и операций, созданных внутри класса для матриц:

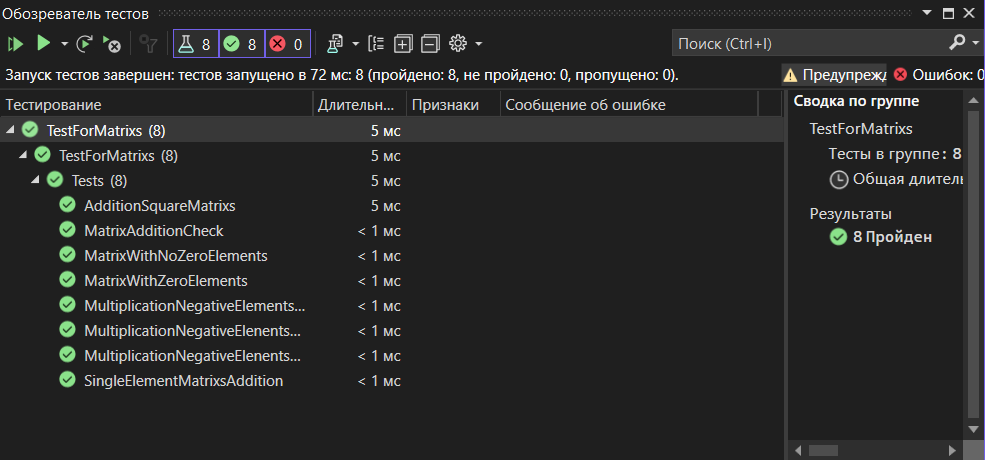


Рисунок 7 – 8 пройденных модульных тестов.

На рисунке 8 показан журнал Git для данной программы:

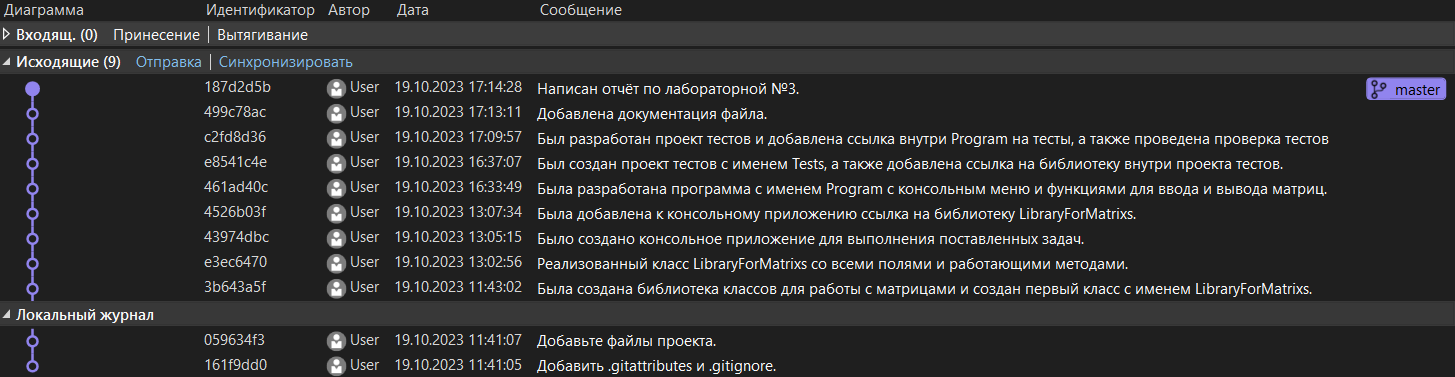


Рисунок 8 – Журнал Git.

Так как тесты все верны, то можно сделать вывод, что операторы и операции были разработаны правильно, а именно сложение матриц одинаковой размерности, произведение отрицательных элементов матриц с выводом и без, перегрузка операторов true и false для проверки матриц на наличие ненулевого элемента. Для сложения матриц прежде, чем их сложить, нужно было проверить равны ли их размерность, а уже потом по правилу сложения матриц их сложить.

Уже произведение отрицательных элементов матрицы проще в реализации, чем сложение матриц, так как для того, чтобы её реализовать, надо пройти по матрице и все отрицательные элементы записать в одномерный массив, а потом поочередно их перемножить.

Реализация true и false для матриц ещё проще. Всего лишь надо пройти по матрице и при обнаружении ненулевого элемента вернуть значение true.

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, операторы и операции для выполнения поставленных задач;

– организовано консольное меню пользователя;

– выполнены модульные тесты;

**Вывод:** в результате выполнения лабораторной работы научились реализовывать перегрузку операторов и перегрузку операций, также использовать индексаторы для вызова значений у закрытых полей, сделали класс для отображения матриц, повторили использование модульные тесты.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

**Program.cs:**

using LibraryForMatrixs;

namespace Matrix

{

internal class Program

{

static void Main(string[] args)

{

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

double negativeA = 0, negativeB = 0, negativeC = 0, number = 0;

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

while (isRun) // Консольное меню.

{

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

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

Console.WriteLine("| 1 - Ввод 3 Матриц; |");

Console.WriteLine("| 2 - Вывод 3 Матриц; |");

Console.WriteLine("| 3 - Произведение отрицательных |");

Console.WriteLine("| элементов Матриц без вывода; |");

Console.WriteLine("| 4 - Произведение отрицательных |");

Console.WriteLine("| элементов Матриц с выводом; |");

Console.WriteLine("| 5 - Сумма Матриц A, B и С; |");

Console.WriteLine("| 6 - Выполнение особого задания; |");

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

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

switch (Convert.ToInt32(Console.ReadLine()))

{

case 1:

matrixA = Program.InputMatrix(); // Метод создания матрицы из полученной информации.

matrixB = Program.InputMatrix();

matrixC = Program.InputMatrix();

break;

case 2:

Console.WriteLine("\n");

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

Console.WriteLine("\n");

Program.OutputMatrix(matrixB);

Console.WriteLine("\n");

Program.OutputMatrix(matrixC);

break;

case 3:

negativeA = Matrixs.MultiplyNegativeElements(matrixA); // Метод произведения отрицательных элементов матриц(без вывода).

negativeB = Matrixs.MultiplyNegativeElements(matrixB);

negativeC = Matrixs.MultiplyNegativeElements(matrixC);

Console.WriteLine("Произведения посчитаны.");

break;

case 4:

negativeA = Matrixs.MultiplyNegativeElements(matrixA, negativeA); // Операция произведения отрицательных элементов матриц(с выводом).

Console.WriteLine(Matrixs.ToString(negativeA, matrixA[0]));

negativeB = Matrixs.MultiplyNegativeElements(matrixB, negativeB);

Console.WriteLine(Matrixs.ToString(negativeB, matrixB[0]));

negativeC = Matrixs.MultiplyNegativeElements(matrixC, negativeC);

Console.WriteLine(Matrixs.ToString(negativeC, matrixC[0]));

break;

case 5:

Matrixs D = new Matrixs();

D = matrixA + matrixB; // Операция сложения матриц.

D = D + matrixC;

if (D.Line != 0)

{

Program.OutputMatrix(D);

}

else

{

D = matrixA + matrixB;

if (D.Line != 0)

{

Program.OutputMatrix(D);

}

else

{

D = matrixA + matrixC;

if (D.Line != 0)

{

Program.OutputMatrix(D);

}

else

{

D = matrixB + matrixC;

if (D.Line != 0)

{

Program.OutputMatrix(D);

}

else

{

Console.WriteLine("Матрицы нельзя сложить.");

}

}

}

}

break;

case 6:

Console.Write("\nВведите число, выше которого должно быть произведение для выполнения особого задания: ");

number = Program.ReadDouble();

if (Matrixs.MultiplyNegativeElements(matrixA) > number)

{

if (matrixC) // Проверка на наличие ненулевых элементов внутри матрицы

{

double min = matrixC[matrixC.Line - 1, 0];

int j = 1;

while (j < matrixC.Column)

{

if (min > matrixC[matrixC.Line - 1, j])

{

min = matrixC[matrixC.Line - 1, j];

}

j++;

}

int i = 0;

while (i < matrixC.Line)

{

j = 0;

while (j < matrixC.Column)

{

if (matrixC[i, j] < 0)

{

matrixC[i, j] = matrixC[i, j] + min;

}

j++;

}

i++;

}

Console.WriteLine("\nЗадача выполнилась!\n");

Program.OutputMatrix(matrixC);

}

else

{

Console.WriteLine("В третьей матрице только нулевые элементы.");

}

}

else

{

Console.WriteLine("Произведение отрицательных элементов не привысило число.");

}

break;

case 0:

isRun = false; // Выход из программы.

break;

default:

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

break;

}

}

}

/// <summary>

/// Метод ввода двумерного массива 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)

{

value[i, j] = Program.ReadDouble();

j++;

}

i++;

}

return value;

}

/// <summary>

/// Метод создания матрица из консоли.

/// </summary>

public static Matrixs InputMatrix()

{

int line, column; // Создание переменных для обозначения строчек и столбцов матриц.

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

string name; // Создание переменной для обозначения имени матрицы.

Console.Write("\nВведите имя матрицы: ");

name = Console.ReadLine();

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

line = Program.ReadInt();

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

column = Program.ReadInt();

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

values = Program.InputMatrix(line, column);

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

return new Matrixs(line, column, values, name);

}

/// <summary>

/// Метод вывода матрицы в консоль.

/// </summary>

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

public static void OutputMatrix(Matrixs A)

{

int i = 0;

Console.WriteLine("\n" + A[0] + ":\n");

while (i < A.Line)

{

int j = 0;

while (j < A.Column)

{

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

Console.Write(" ");

j++;

}

Console.WriteLine("\n");

i++;

}

}

/// <summary>

/// Метод для проверки вводимого значения для вещественного числа.

/// </summary>

/// <returns></returns>

public static double ReadDouble()

{

string numeral = Console.ReadLine();

double value;

while (!Double.TryParse(numeral, out value))

{

Console.WriteLine("Вводный данные не подходят. Введите корректное значение: ");

numeral = Console.ReadLine();

}

return value;

}

/// <summary>

/// Метод для проверки вводимого значения для целочисленного числа.

/// </summary>

/// <returns></returns>

public static int ReadInt()

{

string numeral = Console.ReadLine();

int value;

while (!Int32.TryParse(numeral, out value))

{

Console.WriteLine("Вводный данные не подходят. Введите корректное значение: ");

numeral = Console.ReadLine();

}

return value;

}

}

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Текст библиотеки**

**LibraryForMatrixs.cs:**

namespace LibraryForMatrixs

{

/// <summary>

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

/// </summary>

public class Matrixs

{

private double[,] number = new double[1000, 1000];

private string[] name = new string[1000];

public int line, column;

public int Line

{

get { return line; }

set

{ line = value; }

}

public int Column

{

get { return column; }

set

{

column = value;

}

}

public double this[int i, int j]

{

get => number[i, j];

set => number[i, j] = value;

}

public string this[int i]

{

get => name[0];

set => name[0] = value;

}

/// <summary>

/// Конструктор пустой матрицы.

/// </summary>

public Matrixs()

{

this.Line = 0;

this.Column = 0;

this[Line, Column] = 0;

this[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, string name)

{

this.Line = line;

this.Column = column;

this[0] = name;

int stepLine = 0, stepColumn = 0;

while (stepLine < line)

{

stepColumn = 0;

while (stepColumn < column)

{

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

stepColumn++;

}

stepLine++;

}

}

/// <summary>

/// Операция сложения матрицы на матрицу.

/// </summary>

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

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

/// <returns></returns>

public static Matrixs operator +(Matrixs A, Matrixs B)

{

if (A.Line == B.Line && A.Column == B.Column)

{

int stepLine = 0, stepColumn = 0;

Matrixs C = new Matrixs();

C.Line = A.Line;

C.Column = A.Column;

C[0] = "Result Matrix";

while (stepLine < A.Line)

{

stepColumn = 0;

while (stepColumn < A.Column)

{

C[stepLine, stepColumn] = A[stepLine, stepColumn] + B[stepLine, stepColumn];

stepColumn++;

}

stepLine++;

}

return C;

}

Matrixs D = new Matrixs();

return D;

}

/// <summary>

/// Переменная для определения ненулевого элемента внутри матрицы.

/// </summary>

public bool HaveNozero

{

get

{

int stepLine = 0, stepColumn = 0;

while (stepLine < this.Line)

{

stepColumn = 0;

while (stepColumn < this.Column)

{

if (this[stepLine, stepColumn] != 0)

{

return true;

}

stepColumn++;

}

stepLine++;

}

return false;

}

}

/// <summary>

/// Операция true для матрицы.

/// </summary>

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

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

/// <returns></returns>

public static bool operator true(Matrixs A)

{

return A.HaveNozero;

}

/// <summary>

/// Операция false для матрицы.

/// </summary>

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

/// <returns></returns>

public static bool operator false(Matrixs A)

{

return !A.HaveNozero;

}

/// <summary>

/// Метод для нахождения произведения отрицательных элементов матрицы без вывода результата.

/// </summary>

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

/// <returns></returns>

public static double MultiplyNegativeElements(Matrixs A)

{

int stepLine = 0, sizeArray = 0;

double[] negative = new double[1000];

while (stepLine < A.Line)

{

int stepColumn = 0;

while (stepColumn < A.Column)

{

if (A[stepLine, stepColumn] < 0)

{

negative[sizeArray] = A[stepLine, stepColumn];

sizeArray++;

}

stepColumn++;

}

stepLine++;

}

if (sizeArray != 0)

{

int i = 0;

double number = 1;

while (i < sizeArray)

{

number = number \* negative[i];

i++;

}

return number;

}

else

{

double number = 0;

return number;

}

}

/// <summary>

/// Метод для нахождения произведения отрицательных элементов матрицы с выводом результата

/// </summary>

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

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

/// <returns></returns>

public static double MultiplyNegativeElements(Matrixs A, double number)

{

int stepLine = 0, sizeArray = 0;

double[] negative = new double[1000];

while (stepLine < A.Line)

{

int stepColumn = 0;

while (stepColumn < A.Column)

{

if (A[stepLine, stepColumn] < 0)

{

negative[sizeArray] = A[stepLine, stepColumn];

sizeArray++;

}

stepColumn++;

}

stepLine++;

}

if (sizeArray != 0)

{

int i = 0;

number = 1;

while (i < sizeArray)

{

number = number \* negative[i];

i++;

}

return number;

}

else

{

number = 0;

return number;

}

}

public static string ToString(double number, string name)

{

string res = number + " - " + name + "\n";

return res;

}

}

}

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Текст тестов**

**Tests.cs:**

using LibraryForMatrixs;

namespace TestForMatrixs

{

[TestClass]

public class Tests

{

/// <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 || A[0] != B[0])

{

return false;

}

int i = 0;

while (i < A.Line)

{

int j = 0;

while (j < A.Column)

{

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

{

return false;

}

j++;

}

i++;

}

return true;

}

/// <summary>

/// Тест, проверяющий сложение одноэлементных матриц.

/// </summary>

[TestMethod]

public void SingleElementMatrixsAddition()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 1;

matrix1.Column = 1;

matrix1[0, 0] = 3;

Matrixs matrix2 = new Matrixs();

matrix2.Line = 1;

matrix2.Column = 1;

matrix2[0, 0] = 1;

Matrixs matrix3 = matrix1 + matrix2;

Matrixs result = new Matrixs();

result.Line = 1;

result.Column = 1;

result[0,0] = 4;

result[0] = "Result Matrix";

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

}

/// <summary>

/// Тест, проверяющий сложение квадратных матриц.

/// </summary>

[TestMethod]

public void AdditionSquareMatrixs()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1[0, 0] = 1; matrix1[0, 1] = 2; matrix1[1,0] = 3; matrix1[1,1] = 4;

Matrixs matrix2 = new Matrixs();

matrix2.Line = 2;

matrix2.Column = 2;

matrix2[0, 0] = 9; matrix2[0, 1] = 8; matrix2[1,0] = 7; matrix2[1,1] = 6;

Matrixs matrix3 = matrix1 + matrix2;

Matrixs result = new Matrixs();

result.Line = 2;

result.Column = 2;

result[0, 0] = 10; result[0, 1] = 10; result[1,0] = 10; result[1,1] = 10;

result[0] = "Result Matrix";

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

}

/// <summary>

/// Тест, проверяющий возможность сложения матриц.

/// </summary>

[TestMethod]

public void MatrixAdditionCheck()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 1;

matrix1.Column = 2;

matrix1[0, 0] = 2; matrix1[0 ,1 ] = 3;

Matrixs matrix2 = new Matrixs();

matrix2.Line = 1;

matrix2.Column = 1;

matrix2[0, 0] = 1;

Matrixs matrix3 = matrix1 + matrix2;

Matrixs result = new Matrixs();

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

}

/// <summary>

/// Тест, проверяющий наличие в матрице ненулевых элементов.

/// </summary>

[TestMethod]

public void MatrixWithNoZeroElements()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 3;

matrix1.Column = 2;

matrix1[0, 0] = 0; matrix1[0, 1] = 0; matrix1[1,0] = 0; matrix1[1,1] = 0; matrix1[2, 0] = 2; matrix1[2, 1] = 0;

bool noZero = false;

if (matrix1)

{

noZero = true;

}

Assert.IsTrue(noZero);

}

/// <summary>

/// Тест, проверяющий матрицу на все нулевые значения.

/// </summary>

[TestMethod]

public void MatrixWithZeroElements()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1[0, 0] = 0; matrix1[0, 1] = 0; matrix1[1, 0] = 0; matrix1[1, 1] = 0;

bool zero = true;

if(matrix1)

{

zero = false;

}

Assert.IsTrue(zero);

}

/// <summary>

/// Тест, проверяющий произвденеие отрицательных элементов матрицы(без вывода);.

/// </summary>

[TestMethod]

public void MultiplicationNegativeElenentsOfMatrixWithOutOutput()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1[0, 0] = -2; matrix1[0, 1] = -5; matrix1[1, 0] = -3; matrix1[1,1] = -1.1;

double negative = 33;

double result = Matrixs.MultiplyNegativeElements(matrix1);

Assert.AreEqual(result,negative);

}

/// <summary>

/// Тест, проверяющий произвденеие отрицательных элементов матрицы(с выводом).

/// </summary>

[TestMethod]

public void MultiplicationNegativeElenentsOfMatrixWithOutput()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 2;

matrix1.Column = 2;

matrix1[0, 0] = 2; matrix1[0, 1] = -5; matrix1[1, 0] = -5; matrix1[1,1] = -1;

double negative = -25;

double result = 0;

result = Matrixs.MultiplyNegativeElements(matrix1,result);

Assert.AreEqual(result,negative);

}

/// <summary>

/// Тест, проверяющий произведение отрицательных элементов матрицы, у которой все элементы положительны или ноль.

/// </summary>

[TestMethod]

public void MultiplicationNegativeElementsMatrix()

{

Matrixs matrix1 = new Matrixs();

matrix1.Line = 3;

matrix1.Column = 3;

matrix1[0, 0] = 3; matrix1[0, 1] = 3; matrix1[0, 2] = 3; matrix1[1, 0] = 1; matrix1[1, 1] = 1; matrix1[1, 2] = 1; matrix1[2, 0] = 0; matrix1[2, 1] = 0; matrix1[2,2] = 0;

double negative = 0;

double result = Matrixs.MultiplyNegativeElements(matrix1);

Assert.AreEqual(result,negative);

}

}

}