Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Лабораторная работа №4

Выполнил:
студенты 4 курса
группы ИП-216
Андрущенко Ф.А.
Литвинов А. Е.
Русецкий А. С.

Проверил:
преподаватель кафедры ПМиК
Агалаков Антон Александрович

Новосибирск, 2025 г.

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.

Спецификация типа данных Матрица ADT Matrix Данные Матрица (тип Matrix) - это двумерная матрица со значениями целого типа. Объект типа Матрица – не изменяемый.

Операции:

Конструктор (Matrix)	
Вход:	Получает двумерный массив целых m.Число строк i и столбцов j.
Предусловия:	Число строк и столбцов должно быть больше 0.
Процесс:	Инициирует объект типа Matrix полученным массивом. Заносит число строк и столбцов в соответствующие свойства I и J.
Сложить (operator+)	
Вход:	(b) – объект тип Matrix.
Предусловия:	Число строк и столбцов в суммируемых матрицах должны совпадать
Процесс:	Создаёт новый объект типа Matrix, элементы которого, получены путём сложения элементов объектов this и b с одинаковыми индексами.
Выход:	Объект типа Matrix.
Постусловия:	Нет.
Вычесть (operator-)	
Вход:	(b) – объект тип Matrix.
Предусловия:	Число строк и столбцов в матрицах, участвующих в вычитании, должны совпадать.
Процесс:	Создаёт новый объект типа Matrix, элементы которого, получены путём вычитания элементов объектов this и b с одинаковыми индексами.
Выход:	Объект типа Matrix.
Постусловия:	Нет.

Умножить (operator*)	
Вход:	(b) – объект типа Matrix.
Предусловия:	Матрицы, участвующие в умножении, должны быть согласованы для этой операции по числу строк и столбцов.
Процесс:	Создаёт новый объект типа Matrix, элементы которого, получены путём умножения элементов объектов this и b в соответствии с правилами перемножения матриц.
Выход:	Объект типа Matrix.
Постусловия:	Нет.
Равно (operator==)	
Вход:	(b) – объект типа Matrix.
Предусловия:	Число строк и столбцов в матрицах, участвующих в вычитании, должны совпадать.
Процесс:	Возвращает значение true, если элементы объектов this и b в на одинаковых позициях равны.
Выход:	Значение типа bool.
Постусловия:	Нет.
Транспонировать (Transp)	
Вход:	Нет.
Предусловия:	Матрица, подвергаемая транспонированию, должна иметь одинаковое число строк и столбцов.
Процесс:	Создаёт новый объект типа Matrix, элементы которого, получены путём транспонирования элементов объекта this.
Выход:	Объект типа Matrix.

Минимальный элемент(Min)	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Отыскивает и возвращает минимальный среди элементов объекта this.
Выход:	Значение типа int.
Постусловия:	Нет.
ПреобразоватьВстроку(ToString)	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Преобразует элементы матрицы this в строковое представление построчно. Напрмер: {{1,2,3},{4,5,6},{7,8,9}}
Выход:	Строка.
Постусловия:	Нет.
Взять элемент с индексами i,j (this [i,j])	
Вход:	Значения i, j типа int.
Предусловия:	Значения i, j должны находиться в допустимых диапазонах.
Процесс:	Возвращает элемент матрицы с индексами ***i,j***.
Выход:	Значение типа int.
Постусловия:	Нет.
Взять число строк I	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает число строк в матрице.
Выход:	Целое – число строк.
Постусловия:	Нет.
Взять число столбцов J	
Вход:	Нет.
Предусловия:	Нет.
Процесс:	Возвращает число столбцов в матрице.
Выход:	Целое – число столбцов.
Постусловия:	Нет.

Исходные тексты приложения

```
class Matrix:
    def __init__(self, data):
        """
        Конструктор матрицы
        Вход: двумерный массив целых чисел
        Предусловия: число строк и столбцов > 0
        """
        if not data or len(data) == 0 or len(data[0]) == 0:
            raise ValueError("Матрица должна иметь хотя бы одну строку и столбец")
        # Проверяем, что все строки одинаковой длины
        row_length = len(data[0])
        for row in data:
            if len(row) != row_length:
                raise ValueError("Все строки матрицы должны быть одинаковой длины")
        self._data = [row[:] for row in data]  # Глубокая копия
        self._i = len(data)  # число строк
        self._j = len(data[0])  # число столбцов

    @property
    def I(self):
        """Число строк"""
        return self._i

    @property
    def J(self):
        """Число столбцов"""
        return self._j

    def __getitem__(self, indices):
        """
        Взять элемент с индексами i,j
        Предусловия: индексы в допустимых диапазонах
        """
        i, j = indices
        if not (0 <= i < self._i and 0 <= j < self._j):
            raise IndexError("Индексы выходят за границы матрицы")
        return self._data[i][j]

    def __add__(self, other):
        """
        Сложение матриц
        Предусловия: размеры матриц совпадают
        """
        if self._i != other._i or self._j != other._j:
            raise ValueError("Размеры матриц должны совпадать для сложения")
        result = []
        for i in range(self._i):
            row = []
            for j in range(self._j):
                row.append(self._data[i][j] + other._data[i][j])
```

```
            result.append(row)
        return Matrix(result)

    def __sub__(self, other):
        """
        Вычитание матриц
        Предусловия: размеры матриц совпадают
        """
        if self._i != other._i or self._j != other._j:
            raise ValueError("Размеры матриц должны совпадать для вычитания")
        result = []
        for i in range(self._i):
            row = []
            for j in range(self._j):
                row.append(self._data[i][j] - other._data[i][j])
            result.append(row)
        return Matrix(result)

    def __mul__(self, other):
        """
        Умножение матриц
        Предусловия: матрицы согласованы для умножения
        """
        if self._j != other._i:
            raise ValueError(
                "Число столбцов первой матрицы должно равняться числу строк второй
матрицы"
            )
        result = []
        for i in range(self._i):
            row = []
            for j in range(other._j):
                sum_val = 0
                for k in range(self._j):
                    sum_val += self._data[i][k] * other._data[k][j]
                row.append(sum_val)
            result.append(row)
        return Matrix(result)

    def __eq__(self, other):
        """
        Проверка равенства матриц
        Предусловия: размеры матриц совпадают
        """
        if self._i != other._i or self._j != other._j:
            return False
        for i in range(self._i):
            for j in range(self._j):
                if self._data[i][j] != other._data[i][j]:
                    return False
        return True
```

```
    def transp(self):
        """
        Транспонирование матрицы
        Предусловия: матрица квадратная
        """
        if self._i != self._j:
            raise ValueError("Матрица должна быть квадратной для транспонирования")
        result = []
        for j in range(self._j):
            row = []
            for i in range(self._i):
                row.append(self._data[i][j])
            result.append(row)
        return Matrix(result)

    def min(self):
        """
        Поиск минимального элемента
        """
        if self._i == 0 or self._j == 0:
            raise ValueError("Матрица пустая")
        min_val = self._data[0][0]
        for i in range(self._i):
            for j in range(self._j):
                if self._data[i][j] < min_val:
                    min_val = self._data[i][j]
        return min_val

    def __str__(self):
        """
        Преобразование матрицы в строку
        """
        rows = []
        for row in self._data:
            rows.append("{" + ",".join(map(str, row)) + "}")
        return "{" + ",".join(rows) + "}"

    def __repr__(self):
        return self.__str__()
```

Модульные тесты для тестирования класса Matrix

```
import unittest
from matrix import Matrix
```

```
class TestMatrix(unittest.TestCase):
    def test_constructor_valid(self):
        """Тест конструктора с валидными данными"""
        data = [[1, 2], [3, 4]]
        matrix = Matrix(data)
        self.assertEqual(matrix.I, 2)
        self.assertEqual(matrix.J, 2)

    def test_constructor_invalid_empty(self):
        """Тест конструктора с пустой матрицей"""
        with self.assertRaises(ValueError):
            Matrix([])

    def test_constructor_invalid_empty_inner_array(self):
        """Тест конструктора с пустым внутренним массивом"""
        with self.assertRaises(ValueError):
            Matrix([[]])

    def test_constructor_invalid_irregular(self):
        """Тест конструктора с неправильной матрицей"""
        with self.assertRaises(ValueError):
            Matrix([[1, 2], [3]])

    def test_get_item_valid(self):
        """Тест получения элемента по индексу"""
        data = [[1, 2], [3, 4]]
        matrix = Matrix(data)
        self.assertEqual(matrix[0, 0], 1)
        self.assertEqual(matrix[1, 1], 4)

    def test_get_item_invalid(self):
        """Тест получения элемента с невалидным индексом"""
        data = [[1, 2], [3, 4]]
        matrix = Matrix(data)
        with self.assertRaises(IndexError):
            _ = matrix[2, 0]
        with self.assertRaises(IndexError):
            _ = matrix[0, 2]
        with self.assertRaises(IndexError):
            _ = matrix[-1, 0]
        with self.assertRaises(IndexError):
            _ = matrix[0, -1]

    def test_addition_valid(self):
        """Тест сложения матриц"""
        matrix1 = Matrix([[1, 2], [3, 4]])
        matrix2 = Matrix([[5, 6], [7, 8]])
        result = matrix1 + matrix2
        expected = Matrix([[6, 8], [10, 12]])
        self.assertEqual(result, expected)
```

```
    def test_addition_invalid_size(self):
        """Тест сложения матриц разного размера"""
        matrix1 = Matrix([[1, 2], [3, 4]])
        matrix2 = Matrix([[1, 2, 3], [4, 5, 6]])
        with self.assertRaises(ValueError):
            _ = matrix1 + matrix2

    def test_subtraction_valid(self):
        """Тест вычитания матриц"""
        matrix1 = Matrix([[5, 6], [7, 8]])
        matrix2 = Matrix([[1, 2], [3, 4]])
        result = matrix1 - matrix2
        expected = Matrix([[4, 4], [4, 4]])
        self.assertEqual(result, expected)

    def test_subtraction_invalid_size(self):
        """Тест вычитания матриц разного размера (покрытие исключения в sub)"""
        matrix1 = Matrix([[1, 2], [3, 4]])
        matrix2 = Matrix([[1, 2, 3], [4, 5, 6]])
        with self.assertRaises(ValueError):
            _ = matrix1 - matrix2

    def test_multiplication_valid(self):
        """Тест умножения матриц"""
        matrix1 = Matrix([[1, 2], [3, 4]])
        matrix2 = Matrix([[2, 0], [1, 2]])
        result = matrix1 * matrix2
        expected = Matrix([[4, 4], [10, 8]])
        self.assertEqual(result, expected)

    def test_multiplication_non_square(self):
        """Тест умножения неквадратных матриц"""
        matrix1 = Matrix([[1, 2, 3], [4, 5, 6]])
        matrix2 = Matrix([[1, 2], [3, 4], [5, 6]])
        result = matrix1 * matrix2
        expected = Matrix([[22, 28], [49, 64]])
        self.assertEqual(result, expected)

    def test_multiplication_invalid_size(self):
        """Тест умножения несогласованных матриц"""
        matrix1 = Matrix([[1, 2, 3]])
        matrix2 = Matrix([[1, 2]])
        with self.assertRaises(ValueError):
            _ = matrix1 * matrix2

    def test_equality_valid(self):
        """Тест проверки равенства матриц"""
        matrix1 = Matrix([[1, 2], [3, 4]])
        matrix2 = Matrix([[1, 2], [3, 4]])
        matrix3 = Matrix([[5, 6], [7, 8]])
        self.assertTrue(matrix1 == matrix2)
        self.assertFalse(matrix1 == matrix3)
```

```
    def test_equality_different_size(self):
        """Тест проверки равенства матриц разного размера"""
        matrix1 = Matrix([[1, 2]])
        matrix2 = Matrix([[1, 2, 3]])
        self.assertFalse(matrix1 == matrix2)

    def test_equality_with_different_values(self):
        """Тест проверки равенства матриц с разными значениями"""
        matrix1 = Matrix([[1, 2], [3, 4]])
        matrix2 = Matrix([[1, 2], [3, 5]])  # Одно значение отличается
        self.assertFalse(matrix1 == matrix2)

    def test_transpose_valid(self):
        """Тест транспонирования квадратной матрицы"""
        matrix = Matrix([[1, 2], [3, 4]])
        result = matrix.transp()
        expected = Matrix([[1, 3], [2, 4]])
        self.assertEqual(result, expected)

    def test_transpose_invalid(self):
        """Тест транспонирования неквадратной матрицы"""
        matrix = Matrix([[1, 2, 3], [4, 5, 6]])
        with self.assertRaises(ValueError):
            _ = matrix.transp()

    def test_min_element(self):
        """Тест поиска минимального элемента"""
        matrix = Matrix([[5, 2], [8, 1], [3, 7]])
        self.assertEqual(matrix.min(), 1)

    def test_min_element_single(self):
        """Тест поиска минимального элемента в матрице 1x1"""
        matrix = Matrix([[42]])
        self.assertEqual(matrix.min(), 42)

    def test_min_element_negative(self):
        """Тест поиска минимального отрицательного элемента"""
        matrix = Matrix([[5, -2], [8, 1], [3, -7]])
        self.assertEqual(matrix.min(), -7)

    def test_min_element_all_negative(self):
        """Тест поиска минимального элемента в матрице с отрицательными значениями"""
        matrix = Matrix([[-5, -2], [-8, -1], [-3, -7]])
        self.assertEqual(matrix.min(), -8)

    def test_min_element_empty_matrix(self):
        """Тест поиска минимального элемента в пустой матрице (покрытие исключения в
min())"""
        # Создаем "пустую" матрицу для тестирования исключения
        matrix = Matrix.__new__(Matrix)
        matrix._data = []
```

```
        matrix._i = 0
        matrix._j = 0
        with self.assertRaises(ValueError):
            matrix.min()

    def test_to_string(self):
        """Тест преобразования в строку"""
        matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
        expected = "{{1,2,3},{4,5,6},{7,8,9}}"
        self.assertEqual(str(matrix), expected)

    def test_to_string_single_element(self):
        """Тест преобразования в строку матрицы 1x1"""
        matrix = Matrix([[42]])
        expected = "{{42}}"
        self.assertEqual(str(matrix), expected)

    def test_to_string_single_row(self):
        """Тест преобразования в строку матрицы с одной строкой"""
        matrix = Matrix([[1, 2, 3]])
        expected = "{{1,2,3}}"
        self.assertEqual(str(matrix), expected)

    def test_to_string_single_column(self):
        """Тест преобразования в строку матрицы с одним столбцом"""
        matrix = Matrix([[1], [2], [3]])
        expected = "{{1},{2},{3}}"
        self.assertEqual(str(matrix), expected)

    def test_repr_method(self):
        """Тест метода repr (покрытие метода repr)"""
        matrix = Matrix([[1, 2], [3, 4]])
        repr_str = repr(matrix)
        self.assertEqual(repr_str, "{{1,2},{3,4}}")
        self.assertEqual(
            repr_str, str(matrix)
        )  # repr должен возвращать то же, что и str

    def test_properties(self):
        """Тест свойств I и J"""
        matrix = Matrix([[1, 2, 3], [4, 5, 6]])
        self.assertEqual(matrix.I, 2)
        self.assertEqual(matrix.J, 3)

    def test_properties_single_element(self):
        """Тест свойств I и J для матрицы 1x1"""
        matrix = Matrix([[42]])
        self.assertEqual(matrix.I, 1)
        self.assertEqual(matrix.J, 1)

    def test_properties_single_row(self):
        """Тест свойств I и J для матрицы с одной строкой"""
```

```
        matrix = Matrix([[1, 2, 3]])
        self.assertEqual(matrix.I, 1)
        self.assertEqual(matrix.J, 3)

    def test_properties_single_column(self):
        """Тест свойств I и J для матрицы с одним столбцом"""
        matrix = Matrix([[1], [2], [3]])
        self.assertEqual(matrix.I, 3)
        self.assertEqual(matrix.J, 1)


if __name__ == "__main__":
    unittest.main()
```

Результат тестирования методов класса Matrix

```
Running: test_addition_invalid_size
✓ PASS: test_addition_invalid_size
Running: test_addition_valid
✓ PASS: test_addition_valid
Running: test_constructor_invalid_empty
✓ PASS: test_constructor_invalid_empty
Running: test_constructor_invalid_empty_inner_array
✓ PASS: test_constructor_invalid_empty_inner_array
Running: test_constructor_invalid_irregular
✓ PASS: test_constructor_invalid_irregular
Running: test_constructor_valid
✓ PASS: test_constructor_valid
Running: test_equality_different_size
✓ PASS: test_equality_different_size
Running: test_equality_valid
✓ PASS: test_equality_valid
Running: test_equality_with_different_values
✓ PASS: test_equality_with_different_values
Running: test_get_item_invalid
✓ PASS: test_get_item_invalid
Running: test_get_item_valid
✓ PASS: test_get_item_valid
Running: test_min_element
✓ PASS: test_min_element
Running: test_min_element_all_negative
✓ PASS: test_min_element_all_negative
Running: test_min_element_empty_matrix
✓ PASS: test_min_element_empty_matrix
Running: test_min_element_negative
✓ PASS: test_min_element_negative
Running: test_min_element_single
✓ PASS: test_min_element_single
Running: test_multiplication_invalid_size
✓ PASS: test_multiplication_invalid_size
Running: test_multiplication_non_square
✓ PASS: test_multiplication_non_square
Running: test_multiplication_valid
✓ PASS: test_multiplication_valid
Running: test_properties
✓ PASS: test_properties
Running: test_properties_single_column
✓ PASS: test_properties_single_column
Running: test_properties_single_element
✓ PASS: test_properties_single_element
Running: test_properties_single_row
✓ PASS: test_properties_single_row
Running: test_repr_method
✓ PASS: test_repr_method
Running: test_subtraction_invalid_size
✓ PASS: test_subtraction_invalid_size
Running: test_subtraction_valid
✓ PASS: test_subtraction_valid
Running: test_to_string
✓ PASS: test_to_string
Running: test_to_string_single_column
✓ PASS: test_to_string_single_column
Running: test_to_string_single_element
✓ PASS: test_to_string_single_element
Running: test_to_string_single_row
✓ PASS: test_to_string_single_row
Running: test_transpose_invalid
✓ PASS: test_transpose_invalid
Running: test_transpose_valid
✓ PASS: test_transpose_valid
Ran 32 tests in 0.010s

OK
 Test Results
 Passed: 32
 Failed: 0
 Errors: 0
 Total: 32
```

```
❯ coverage report -m
Name             Stmts   Miss  Cover   Missing
----------------------------------------------
matrix.py           89      0   100%
test_matrix.py     153      1    99%   223
----------------------------------------------
TOTAL              242      1    99%
```