Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Институт информатики и вычислительной техники

09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

Современные технологии программирования

Лабораторная работа №4

Разработка и модульное тестирование класса

Матрица

Вариант 1

Выполнил:

студент гр.ИП-213 Дмитриев Антон Александрович
ФИО студента

«__» _____ 2025 г.

Проверил:

Преподаватель

ФИО преподавателя

«__» _____ 2025 г. Оценка_____

Новосибирск 2025 г.

1. **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 C2 (путей).

   Выполните модульное тестирование класса средствами модульного тестирования Visual Studio.

   Выполните анализ покрытия кода методов тестами.

2. Исходные тексты программ.

Class.py:

```
class Matrix:
    def __init__(self, data):
        if not data or len(data) == 0 or len(data[0]) == 0:
            raise ValueError("Число строк и столбцов должно быть больше 0")

        self._data = [row[:] for row in data]  # создаем копию данных
        self._I = len(data)
        self._J = len(data[0])

        for row in data:
            if len(row) != self._J:
                raise ValueError("Все строки должны иметь одинаковую длину")

    @property
    def I(self):
        return self._I

    @property
    def J(self):
        return self._J

    def __getitem__(self, indices):
        i, j = indices
        if not (0 <= i < self._I and 0 <= j < self._J):
            raise IndexError(f"Индексы ({i}, {j}) выходят за границы матрицы
{self._I}x{self._J}")
        return self._data[i][j]

    def __add__(self, other):
        if self._I != other.I or self._J != other.J:
            raise ValueError("Размерности матриц должны совпадать для сложения")

        result = [
            [self._data[i][j] + other._data[i][j] for j in range(self._J)]
            for i in range(self._I)
        ]
        return Matrix(result)

    def __sub__(self, other):
        if self._I != other.I or self._J != other.J:
            raise ValueError("Размерности матриц должны совпадать для вычитания")
```

```
        result = [
            [self._data[i][j] - other._data[i][j] for j in range(self._J)]
            for i in range(self._I)
        ]
        return Matrix(result)

    def __mul__(self, other):
        if self._J != other.I:
            raise ValueError(f"Матрицы не согласованы для умножения: {self._I}x{self._J} *
{other.I}x{other.J}")

        result = [[0 for _ in range(other.J)] for _ in range(self._I)]

        for i in range(self._I):
            for j in range(other.J):
                for k in range(self._J):
                    result[i][j] += self._data[i][k] * other._data[k][j]

        return Matrix(result)

    def __eq__(self, other):
        if not isinstance(other, Matrix):
            return False

        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):
        result = [[self._data[j][i] for j in range(self._I)] for i in range(self._J)]
        return Matrix(result)

    def Min(self):
        return min(min(row) for row in self._data)

    def __str__(self):
        rows = []
        for i in range(self._I):
            row_str = "[" + ", ".join(str(x) for x in self._data[i]) + "]"
            rows.append(row_str)
```

```
        return "[" + ", ".join(rows) + "]"

    def __repr__(self):
        return f"Matrix({self.__str__()})"
```

Tests.py:

```
import pytest
from lab4 import Matrix

class TestMatrixC2Coverage:

    def test_constructor_valid(self):
        """Тест корректного создания матрицы"""
        data = [[1, 2], [3, 4]]
        matrix = Matrix(data)
        assert matrix.I == 2
        assert matrix.J == 2

    def test_constructor_invalid_empty(self):
        """Тест создания пустой матрицы"""
        with pytest.raises(ValueError):
            Matrix([])

    def test_constructor_invalid_empty_row(self):
        """Тест создания матрицы с пустой строкой"""
        with pytest.raises(ValueError):
            Matrix([[]])

    def test_constructor_invalid_different_lengths(self):
        """Тест создания матрицы с разной длиной строк"""
        with pytest.raises(ValueError):
            Matrix([[1, 2], [3]])

    def test_getitem_valid(self):
        """Тест получения элемента по корректным индексам"""
        matrix = Matrix([[1, 2, 3], [4, 5, 6]])
        assert matrix[0, 0] == 1
        assert matrix[1, 2] == 6

    @pytest.mark.parametrize("i,j", [
        (-1, 0),   # отрицательная строка
        (2, 0),    # строка за границами
        (0, -1),   # отрицательный столбец
        (0, 3)     # столбец за границами
    ])
```

```
def test_getitem_invalid_index(self, i, j):
    """Тест получения элемента с неверными индексами"""
    matrix = Matrix([[1, 2, 3], [4, 5, 6]])
    with pytest.raises(IndexError):
        _ = matrix[i, j]

def test_addition_valid(self):
    """Тест корректного сложения матриц"""
    m1 = Matrix([[1, 2], [3, 4]])
    m2 = Matrix([[5, 6], [7, 8]])
    result = m1 + m2
    expected = Matrix([[6, 8], [10, 12]])
    assert result == expected

@pytest.mark.parametrize("m2_data,expected_error", [
    ([[1, 2]], "different number of lines"),
    ([[1, 2, 3], [4, 5, 6]], "different number of colums")
])
def test_addition_invalid_size(self, m2_data, expected_error):
    """Тест сложения матриц разного размера"""
    m1 = Matrix([[1, 2], [3, 4]])
    m2 = Matrix(m2_data)
    with pytest.raises(ValueError):
        _ = m1 + m2

def test_subtraction_valid(self):
    """Тест корректного вычитания матриц"""
    m1 = Matrix([[5, 6], [7, 8]])
    m2 = Matrix([[1, 2], [3, 4]])
    result = m1 - m2
    expected = Matrix([[4, 4], [4, 4]])
    assert result == expected

@pytest.mark.parametrize("m2_data", [
    [[1, 2]],          # разное число строк
    [[1, 2, 3], [4, 5, 6]]  # разное число столбцов
])
def test_subtraction_invalid_size(self, m2_data):
    """Тест вычитания матриц разного размера"""
    m1 = Matrix([[1, 2], [3, 4]])
    m2 = Matrix(m2_data)
    with pytest.raises(ValueError):
        _ = m1 - m2

@pytest.mark.parametrize("m1_data,m2_data,expected_data", [
```

```
    ([[1, 2], [3, 4]], [[2, 0], [1, 3]], [[4, 6], [10, 12]]),  # 2x2 * 2x2
    ([[1, 2, 3]], [[1, 2], [3, 4], [5, 6]], [[22, 28]])        # 1x3 * 3x2
])
def test_multiplication_valid(self, m1_data, m2_data, expected_data):
    """Тест корректного умножения матриц"""
    m1 = Matrix(m1_data)
    m2 = Matrix(m2_data)
    result = m1 * m2
    expected = Matrix(expected_data)
    assert result == expected

def test_multiplication_invalid_size(self):
    """Тест умножения несогласованных матриц"""
    m1 = Matrix([[1, 2], [3, 4]])
    m2 = Matrix([[1, 2]])
    with pytest.raises(ValueError):
        _ = m1 * m2

def test_equality_same(self):
    """Тест сравнения одинаковых матриц"""
    m1 = Matrix([[1, 2], [3, 4]])
    m2 = Matrix([[1, 2], [3, 4]])
    assert m1 == m2

def test_equality_different_elements(self):
    """Тест сравнения матриц с разными элементами"""
    m1 = Matrix([[1, 2], [3, 4]])
    m2 = Matrix([[1, 2], [3, 5]])
    assert m1 != m2

@pytest.mark.parametrize("other_data", [
    [[1, 2]],          # разное число строк
    [[1, 2], [3, 4], [5, 6]],  # разное число строк
    [[1, 2, 3], [4, 5, 6]]     # разное число столбцов
])
def test_equality_different_size(self, other_data):
    """Тест сравнения матриц разного размера"""
    m1 = Matrix([[1, 2], [3, 4]])
    m2 = Matrix(other_data)
    assert m1 != m2

@pytest.mark.parametrize("other", [
    "not a matrix",
    123,
    None,
```

```
    [1, 2, 3]
])
def test_equality_non_matrix(self, other):
    """Тест сравнения с не-Matrix объектом"""
    m1 = Matrix([[1, 2], [3, 4]])
    assert m1 != other

@pytest.mark.parametrize("input_data,expected_data", [
    ([[1, 2], [3, 4]], [[1, 3], [2, 4]]),                # квадратная 2x2
    ([[1, 2, 3], [4, 5, 6]], [[1, 4], [2, 5], [3, 6]]),  # прямоугольная 2x3
    ([[1, 2], [3, 4], [5, 6]], [[1, 3, 5], [2, 4, 6]])   # прямоугольная 3x2
])
def test_transpose(self, input_data, expected_data):
    """Тест транспонирования матриц"""
    matrix = Matrix(input_data)
    result = matrix.Transp()
    expected = Matrix(expected_data)
    assert result == expected

@pytest.mark.parametrize("matrix_data,expected_min", [
    ([[0, 2, 3], [4, 5, 6]], 0),      # минимальный в начале
    ([[5, 1, 8], [9, 7, 2]], 1),      # минимальный в середине
    ([[5, 3, 8], [2, 7, -1]], -1),    # минимальный в конце
    ([[5, 5], [5, 5]], 5),            # все элементы одинаковые
    ([[42]], 42)                      # матрица 1x1
])
def test_min_element(self, matrix_data, expected_min):
    """Тест поиска минимального элемента"""
    matrix = Matrix(matrix_data)
    assert matrix.Min() == expected_min

@pytest.mark.parametrize("matrix_data,expected_str", [
    ([[1]], "[[1]]"),                          # 1x1
    ([[1, 2, 3]], "[[1, 2, 3]]"),              # 1xN
    ([[1], [2], [3]], "[[1], [2], [3]]"),      # Mx1
    ([[1, 2], [3, 4]], "[[1, 2], [3, 4]]")     # MxN
])
def test_string_representation(self, matrix_data, expected_str):
    """Тест строкового представления"""
    matrix = Matrix(matrix_data)
    assert str(matrix) == expected_str

@pytest.mark.parametrize("matrix_data,expected_i,expected_j", [
    ([[1, 2], [3, 4]], 2, 2),                  # квадратная
    ([[1, 2, 3, 4], [5, 6, 7, 8]], 2, 4),      # прямоугольная
```

```
    ([[1]], 1, 1)                    # 1x1
])
def test_properties(self, matrix_data, expected_i, expected_j):
    """Тест свойств I и J"""
    matrix = Matrix(matrix_data)
    assert matrix.I == expected_i
    assert matrix.J == expected_j

def test_large_matrix(self):
    """Тест большой матрицы"""
    large_data = [[i * 10 + j for j in range(10)] for i in range(10)]
    large_matrix = Matrix(large_data)
    assert large_matrix.I == 10
    assert large_matrix.J == 10

def test_negative_numbers(self):
    """Тест матрицы с отрицательными числами"""
    negative_matrix = Matrix([[-1, -2], [-3, -4]])
    assert negative_matrix.Min() == -4

def test_zero_matrix(self):
    """Тест матрицы с нулями"""
    zero_matrix = Matrix([[0, 0], [0, 0]])
    assert zero_matrix.Min() == 0
```

3. Результат тестирования методов класса Matrix.

```
test.py::TestMatrixC2Coverage::test_constructor_valid PASSED
test.py::TestMatrixC2Coverage::test_constructor_invalid_empty PASSED
test.py::TestMatrixC2Coverage::test_constructor_invalid_empty_row PASSED
test.py::TestMatrixC2Coverage::test_constructor_invalid_different_lengths PASSED
test.py::TestMatrixC2Coverage::test_getitem_valid PASSED
test.py::TestMatrixC2Coverage::test_getitem_invalid_index[-1-0] PASSED
test.py::TestMatrixC2Coverage::test_getitem_invalid_index[2-0] PASSED
test.py::TestMatrixC2Coverage::test_getitem_invalid_index[0--1] PASSED
test.py::TestMatrixC2Coverage::test_getitem_invalid_index[0-3] PASSED
test.py::TestMatrixC2Coverage::test_addition_valid PASSED
test.py::TestMatrixC2Coverage::test_addition_invalid_size[m2_data0-different number of lines] PASSED
test.py::TestMatrixC2Coverage::test_addition_invalid_size[m2_data1-different number of colums] PASSED
test.py::TestMatrixC2Coverage::test_subtraction_valid PASSED
test.py::TestMatrixC2Coverage::test_subtraction_invalid_size[m2_data0] PASSED
test.py::TestMatrixC2Coverage::test_subtraction_invalid_size[m2_data1] PASSED
test.py::TestMatrixC2Coverage::test_multiplication_valid[m1_data0-m2_data0-expected_data0] PASSED
test.py::TestMatrixC2Coverage::test_multiplication_valid[m1_data1-m2_data1-expected_data1] PASSED
test.py::TestMatrixC2Coverage::test_multiplication_invalid_size PASSED
test.py::TestMatrixC2Coverage::test_equality_same PASSED
test.py::TestMatrixC2Coverage::test_equality_different_elements PASSED
test.py::TestMatrixC2Coverage::test_equality_different_size[other_data0] PASSED
test.py::TestMatrixC2Coverage::test_equality_different_size[other_data1] PASSED
test.py::TestMatrixC2Coverage::test_equality_different_size[other_data2] PASSED
test.py::TestMatrixC2Coverage::test_equality_non_matrix[not a matrix] PASSED
test.py::TestMatrixC2Coverage::test_equality_non_matrix[123] PASSED
test.py::TestMatrixC2Coverage::test_equality_non_matrix[None] PASSED
test.py::TestMatrixC2Coverage::test_equality_non_matrix[other3] PASSED
test.py::TestMatrixC2Coverage::test_transpose[input_data0-expected_data0] PASSED
test.py::TestMatrixC2Coverage::test_transpose[input_data1-expected_data1] PASSED
test.py::TestMatrixC2Coverage::test_transpose[input_data2-expected_data2] PASSED
test.py::TestMatrixC2Coverage::test_min_element[matrix_data0-0] PASSED
test.py::TestMatrixC2Coverage::test_min_element[matrix_data1-1] PASSED
test.py::TestMatrixC2Coverage::test_min_element[matrix_data2--1] PASSED
test.py::TestMatrixC2Coverage::test_min_element[matrix_data3-5] PASSED
test.py::TestMatrixC2Coverage::test_min_element[matrix_data4-42] PASSED
test.py::TestMatrixC2Coverage::test_string_representation[matrix_data0-[[1]]] PASSED
test.py::TestMatrixC2Coverage::test_string_representation[matrix_data1-[[1, 2, 3]]] PASSED
test.py::TestMatrixC2Coverage::test_string_representation[matrix_data2-[[1], [2], [3]]] PASSED
test.py::TestMatrixC2Coverage::test_string_representation[matrix_data3-[[1, 2], [3, 4]]] PASSED
test.py::TestMatrixC2Coverage::test_properties[matrix_data0-2-2] PASSED
test.py::TestMatrixC2Coverage::test_properties[matrix_data1-2-4] PASSED
test.py::TestMatrixC2Coverage::test_properties[matrix_data2-1-1] PASSED
test.py::TestMatrixC2Coverage::test_large_matrix PASSED
test.py::TestMatrixC2Coverage::test_negative_numbers PASSED
test.py::TestMatrixC2Coverage::test_zero_matrix PASSED
```

```
Name      Stmts   Miss  Cover   Missing
---------------------------------------
lab4.py      63      1    98%   89
---------------------------------------
TOTAL        63      1    98%
---------------------------------------
```