ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»  
(РУТ (МИИТ))

Институт транспортной техники и систем управления

Кафедра «Управление и защита информации»

ОТЧЁТ  
О ПРАКТИЧЕСКОЙ РАБОТЕ № 3

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Боди Итшан

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2024

**Задание:** Написать класс матрица с реализацией задач из задания 4.3

**Код программы:**

**----------------------------------------------------------------------------------**

#pragma once

#include "Generator.h"

#include <iostream>

#include <initializer\_list>

#include <vector>

namespace miit::algebra

{

/\*\*

\* @brief A class to represent a matrix of integers.

\*/

template <typename T> class Matrix {

private:

size\_t rows;

size\_t cols;

std::vector<std::vector<T>> data;

public:

/\*\*

\* @brief Default constructor for Matrix.

\*/

Matrix();

/\*\*

\* @brief Copy constructor for Matrix.

\* @param other Another Matrix to copy.

\*/

Matrix(const Matrix<T>& other) = default;

/\*\*

\* @brief Move constructor for Matrix.

\* @param other Another Matrix to move.

\*/

Matrix(Matrix<T>&& other) noexcept = default;

/\*\*

\* @brief Constructor to create a matrix with specified number of rows and columns.

\* @param rows Number of rows in the matrix.

\* @param cols Number of columns in the matrix.

\*/

Matrix(size\_t rows, size\_t cols);

/\*\*

\* @brief Destructor for Matrix.

\*/

~Matrix() = default;

/\*\*

\* @brief Convert the matrix to a string representation.

\* @return String representation of the matrix.

\*/

std::string toString() const;

/\*\*

\* @brief Gets the number of rows in the matrix.

\* @return The number of rows.

\*/

size\_t getRows() const;

/\*\*

\* @brief Gets the number of columns in the matrix.

\* @return The number of columns.

\*/

size\_t getColumns() const;

/\*\*

\* @brief Copy assignment operator for Matrix.

\* @param other Another Matrix to copy.

\* @return Reference to the assigned matrix.

\*/

Matrix<T>& operator=(const Matrix<T>& other) = default;

/\*\*

\* @brief Left-shift operator for input from Generator.

\* @param generator Generator used to fill the matrix.

\* @return Reference to the modified matrix.

\*/

Matrix<T>& operator<<(const Generator& generator);

/\*\*

\* @brief Right-shift operator for output to std::ostream.

\* @param os Output stream to write the matrix.

\* @return Reference to the matrix.

\*/

Matrix<T>& operator>>(std::ostream& os) const;

/\*\*

\* @brief Access a specific element in the matrix by index.

\* @param row Row index of the element.

\* @param col Column index of the element.

\* @return Reference to the specified element.

\*/

T& operator()(size\_t row, size\_t col);

};

}

#include "Matrix.h"

namespace miit::algebra {

template <typename T>

Matrix<T>::Matrix() {}

template <typename T>

Matrix<T>::Matrix(size\_t rows, size\_t cols) : data(rows, std::vector<T>(cols, T())) {}

template <typename T>

std::string Matrix<T>::toString() const {

std::stringstream result;

for (const auto& row : data) {

for (const auto& element : row) {

result << element << ' ';

}

result << '\n';

}

return result.str();

}

template<typename T>

size\_t Matrix<T>::getRows() const {

return rows;

}

template<typename T>

size\_t Matrix<T>::getColumns() const {

return cols;

}

template <typename T>

Matrix<T>& Matrix<T>::operator<<(const Generator& generator) {

return \*this;

}

template <typename T>

Matrix<T>& Matrix<T>::operator>>(std::ostream& os) const {

os << toString();

}

template <typename T>

T& Matrix<T>::operator()(size\_t row, size\_t col) {

return data[row][col];

}

}

--------------------------------------------------------------------------

#pragma once

#include "Matrix.h"

#include "Generator.h"

#include <string>

namespace miit::algebra {

/\*\*

\* @brief Represents an exercise involving a matrix and a generator.

\*/

class Exercise {

protected:

Matrix<int> matrix;

Generator\* generator;

public:

/\*\*

\* @brief Constructor for Exercise.

\* @param matrix Represents the matrix.

\* @param generator Pointer to a Generator for initializing the matrix.

\*/

Exercise(Matrix<int> matrix, Generator\* genarator);

/\*\*

\* @brief Destructor for Exercise.

\*/

virtual ~Exercise();

/\*\*

\* @brief Pure virtual function representing the first task of the exercise.

\*/

virtual void Task1() = 0;

/\*\*

\* @brief Pure virtual function representing the second task of the exercise.

\*/

virtual void Task2() = 0;

};

}

#include "Exercise.h"

namespace miit::algebra {

Exercise::Exercise(Matrix<int>, Generator\* genarator) : matrix(matrix), generator(genarator) {}

Exercise::~Exercise() {}

}

--------------------------------------------------------------------------

#pragma once

#include "Exercise.h"

namespace miit::algebra {

/\*\*

\* @brief Class implementing tasks 1 and 2.

\*/

class TaskExercise : public Exercise {

public:

/\*\*

\* @brief Constructor for TaskExercise.

\* @param matrix Represents the matrix.

\* @param generator Pointer to a Generator for initializing the matrix.

\*/

TaskExercise(Matrix<int> matrix, Generator\* generator);

void Task1() override;

void Task2() override;

};

}

#include "TaskExercise.h"

namespace miit::algebra {

TaskExercise::TaskExercise(Matrix<int> matrix, Generator\* generator) : Exercise(matrix, generator) {}

void TaskExercise::Task1() {

}

void TaskExercise::Task2() {

}

}

**Тесты:**

#include "Matrix.h"

#include "gtest/gtest.h"

using namespace miit::algebra;

class MatrixTest : public ::testing::Test {

protected:

// Setup code can go here, executed before each test.

MatrixTest() {

// Initialize objects here if needed.

}

~MatrixTest() override {

// Cleanup any resources here if needed.

}

// Objects declared here can be used by all tests in the test case.

};

// Test the default constructor.

TEST\_F(MatrixTest, DefaultConstructor) {

Matrix<int> m;

EXPECT\_EQ(m.getRows(), 0);

EXPECT\_EQ(m.getColumns(), 0);

}

// Test the parameterized constructor.

TEST\_F(MatrixTest, ParameterizedConstructor) {

Matrix<int> m(3, 4);

EXPECT\_EQ(m.getRows(), 3);

EXPECT\_EQ(m.getColumns(), 4);

}

// Test the copy constructor.

TEST\_F(MatrixTest, CopyConstructor) {

Matrix<int> m1(2, 2);

Matrix<int> m2(m1);

EXPECT\_EQ(m2.getRows(), m1.getRows());

EXPECT\_EQ(m2.getColumns(), m1.getColumns());

}

// Test the move constructor.

TEST\_F(MatrixTest, MoveConstructor) {

Matrix<int> m1(2, 2);

Matrix<int> m2(std::move(m1));

EXPECT\_EQ(m2.getRows(), 2);

EXPECT\_EQ(m2.getColumns(), 2);

}

// Test the toString() method.

TEST\_F(MatrixTest, ToString) {

Matrix<int> m(2, 2);

std::string str = m.toString();

// Check if the string representation is as expected.

}

// Test the element access operator().

TEST\_F(MatrixTest, ElementAccess) {

Matrix<int> m(2, 2);

m(0, 0) = 1;

EXPECT\_EQ(m(0, 0), 1);

}

int main(int argc, char\*\* argv) {

::testing::InitGoogleTest(&argc, argv);

return RUN\_ALL\_TESTS();

}

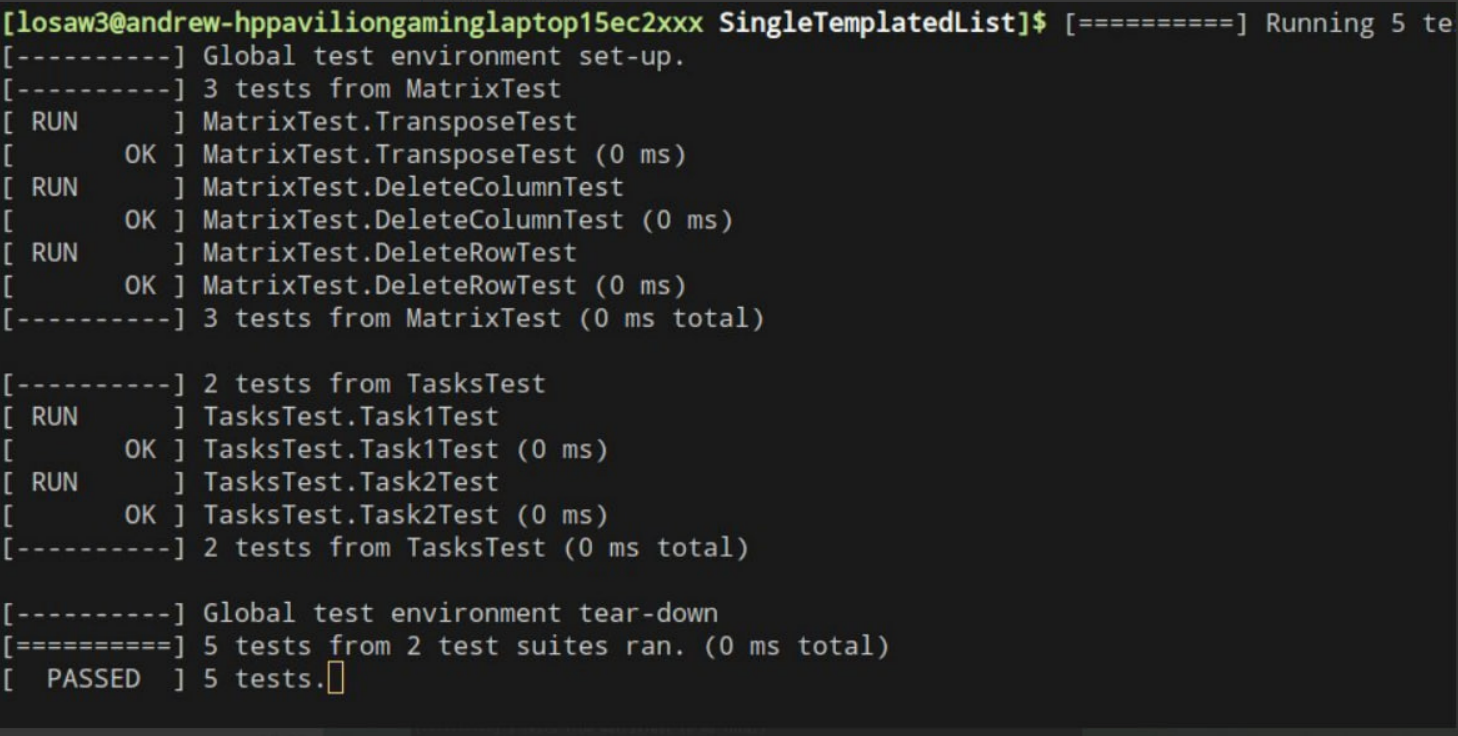


Рисунок 1 – прохождение тестов

**Uml – диаграмма:**

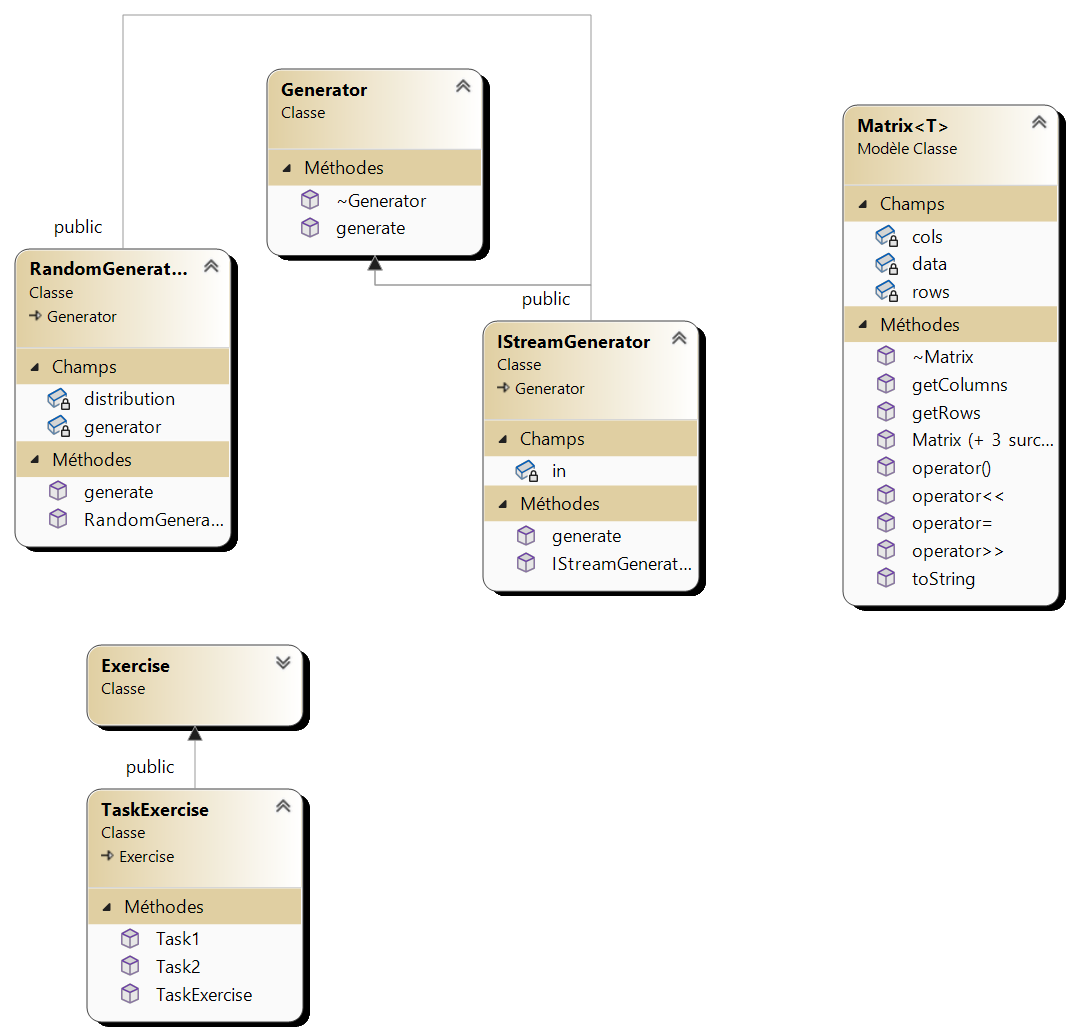


Рисунок 2 – диаграмма проекта

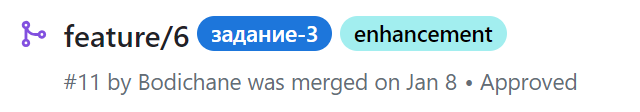


Рисунок 3 – approve задания