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 № 1

По дисциплине «Языки программирования»

Выполнил: ст. гр. ТКИ – 241

Кузмичев Владимир Андреевич

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2024

**Задание:** написать класс Матрица и задания к ней

**Код задания:**

***Matrix.h:***

#pragma once

#include "../Generators/Generator.h"

#include <iostream>

#include <vector>

namespace miit

{

/\*\*

\* @brief Класс матрица

\*/

class Matrix

{

public:

/\*\*

\* @brief Инициализирует объект типа матрица по параметрам пользователя

\* @param user\_rows Количество строк в матрице

\* @param user\_columns Количество столбцов в матрице

\* @param generator Генератор, которым необходимо заполнить матрицу

\*/

Matrix(int user\_rows, int user\_columns, Generator\* generator);

/\*\*

\* @brief Инициализирует объект тип матрица по параметрам пользователя

\* @param user\_data Информация о значениях матрицы в виде двумерного массива

\*/

Matrix(std::initializer\_list<std::initializer\_list<int>> user\_data);

/\*\*

\* @brief Инициализирует пустой объект тип матрица

\*/

Matrix();

/\*\*

\* @brief Перегрузка оператора вывода

\* @param os Поток вывода

\* @param matrix Матрица, данные которой будут выведены в поток

\* @return Поток вывода с информацией матрицы

\*/

friend std::ostream& operator<<(std::ostream& os, Matrix& matrix) noexcept;

/\*\*

\* @brief Перегрузка оператора сравнения

\* @param lha Первый аргумент для сравнения

\* @param rha Второй аргумент для сравнения

\* @return true - если равны, false - если нет

\*/

friend bool operator==(Matrix& lha, Matrix& rha) noexcept;

/\*\*

\* @brief Перегрузка оператора квадратные скобки

\* @param index Индекс элемента в массиве

\* @return Элемент из матрцы стоящий под нужным индексом

\*/

std::vector<int>& operator[](size\_t index);

/\*\*

\* @brief Перегрузка оператора квадратные скобки для константных объектов данного класса

\* @param index Индекс элемента в массиве

\* @return Элемент из матрцы стоящий под нужным индексом

\*/

const std::vector<int>& operator[](size\_t index) const;

/\*\*

\* @brief Получение количества строк в матрице

\* @return Количество строк в матрице

\*/

size\_t rows\_counts() const;

/\*\*

\* @brief Получение количества столбцов в матрице

\* @return Количество столбцов в матрице

\*/

size\_t columns\_counts() const;

/\*\*

\* @brief Преобразование объекта матрицы в строку

\* @return Строка с информацией о матрице

\*/

std::string to\_string() const noexcept;

/\*\*

\* @brief Получение максимального по модолю элемента в матрице

\* @return Максимальный по модолю елемент в матрице

\*/

int get\_abs\_max\_element() const;

/\*\*

\* @brief Добавляет строку в матрицу по индексу

\* @param user\_data Строка для добавления

\* @param index Индекс на который строка будет добавлена

\*/

void add\_row(std::vector<int> user\_data, size\_t index);

/\*\*

\* @brief Добавляет строку в конец матрицы

\* @param user\_data Строка для добавления

\*/

void add\_row(std::vector<int> user\_data);

/\*\*

\* @brief Добавляет столбец в матрицу по индексу

\* @param user\_data Столбец для добавления

\* @param index Индекс на который столбец будет добавлен

\*/

void add\_column(std::vector<int> user\_data, size\_t index);

/\*\*

\* @brief Добавляет столбец в конец матрицы

\* @param user\_data Столбец для добавления

\*/

void add\_column(std::vector<int> user\_data);

/\*\*

\* @brief Проверка на наличие элемента в строке матрицы

\* @param value Элемент для проверки

\* @param row\_id Индекс строки

\* @return true - если есть в сроке, false - если нет

\*/

bool is\_element\_in\_row(int value, size\_t row\_id);

/\*\*

\* @brief Транспонирование матрицы

\*/

void transpose();

private:

std::vector<std::vector<int>> data;

};

};

***Matrix.cpp:***

#include "Matrix.h"

#include <random>

#include <sstream>

miit::Matrix::Matrix(int user\_rows, int user\_columns, Generator\* generator)

:data({})

{

if (user\_rows < 0 || user\_columns < 0)

{

throw std::out\_of\_range("Incorrect matrix size");

}

for (size\_t i = 0; i < user\_rows; i++)

{

std::vector<int> temp{};

for (size\_t j = 0; j < user\_columns; j++)

{

temp.push\_back(generator->generate());

}

this->data.push\_back(temp);

}

}

miit::Matrix::Matrix(std::initializer\_list<std::initializer\_list<int>> user\_data)

:data({})

{

int constant\_column\_size = user\_data.begin()->size();

for (auto& row : user\_data)

{

if (row.size() != constant\_column\_size)

{

throw std::out\_of\_range("Not Matrix");

}

std::vector<int> temp{};

for (auto& column : row)

{

temp.push\_back(column);

}

this->data.push\_back(temp);

}

}

miit::Matrix::Matrix()

:data(std::vector<std::vector<int>> {})

{

}

void miit::Matrix::add\_row(std::vector<int> user\_data, size\_t index)

{

if (index >= this->rows\_counts())

{

throw std::out\_of\_range("incorrect index");

}

this->data.insert(data.begin() + index, user\_data);

}

void miit::Matrix::add\_row(std::vector<int> user\_data)

{

this->data.push\_back(user\_data);

}

void miit::Matrix::add\_column(std::vector<int> user\_data, size\_t index)

{

transpose();

this->add\_row(user\_data, index);

transpose();

}

void miit::Matrix::add\_column(std::vector<int> user\_data)

{

transpose();

this->add\_row(user\_data);

transpose();

}

int miit::Matrix::get\_abs\_max\_element() const

{

int max = 0;

for (size\_t i = 0; i < this->rows\_counts(); i++)

{

for (size\_t j = 0; j < this->columns\_counts(); j++)

{

if (std::abs(this->data[i][j]) >= std::abs(max))

{

max = this->data[i][j];

}

}

}

return max;

}

bool miit::Matrix::is\_element\_in\_row(int value, size\_t row\_id)

{

for (size\_t i = 0; i < this->data[row\_id].size(); i++)

{

if (this->data[row\_id][i] == value)

{

return true;

}

}

return false;

}

size\_t miit::Matrix::rows\_counts() const

{

return this->data.size();

}

size\_t miit::Matrix::columns\_counts() const

{

if (this->data.size() == 0)

{

return 0;

}

return this->data[0].size();

}

void miit::Matrix::transpose()

{

std::vector<std::vector<int>> temp\_data;

for (size\_t j = 0; j < this->data[0].size(); j++)

{

std::vector<int> temp{};

for (size\_t i = 0; i < this->data.size(); i++)

{

temp.push\_back(this->data[i][j]);

}

temp\_data.push\_back(temp);

}

this->data = temp\_data;

}

std::vector<int>& miit::Matrix::operator[](size\_t index)

{

if (index >= this->rows\_counts())

{

throw std::out\_of\_range("Incorrect index");

}

return data[index];

}

const std::vector<int>& miit::Matrix::operator[](size\_t index) const

{

if (index >= this->rows\_counts())

{

throw std::out\_of\_range("Incorrect index");

}

return data[index];

}

std::string miit::Matrix::to\_string() const noexcept

{

std::stringstream temp;

for (size\_t i = 0; i < this->rows\_counts(); i++)

{

for (size\_t j = 0; j < this->columns\_counts(); j++)

{

temp << this->data[i][j] << "\t";

}

temp << "\n";

}

return temp.str();

}

std::ostream& miit::operator<<(std::ostream& os, Matrix& matrix) noexcept

{

return os << matrix.to\_string();

}

bool miit::operator==(Matrix& lha, Matrix& rha) noexcept

{

return lha.to\_string() == rha.to\_string();

}

***Generator.h:***

#pragma once

/\*\*

\* @brief Базовый класс генератора

\*/

class Generator

{

public:

/\*\*

\* @brief Чисто виртуальный деструктор

\*/

virtual ~Generator() = 0 {};

/\*\*

\* @brief Чисто виртуальный метод для генерации чисел в классах наследниках

\*/

virtual int generate() = 0;

};

***IStreamGenerator.h:***

#pragma once

#include "Generator.h"

#include <iostream>

/\*\*

\* @brief Генератор для ввода значений от пользователя

\*/

class IStreamGenerator : public Generator

{

private:

std::istream& in;

public:

/\*\*

\* @brief Инициализация объекта типа IStreamGenerator

\* @param input Поток ввода

\*/

IStreamGenerator(std::istream& input = std::cin);

/\*\*

\* @brief Запрашивает у пользователся число

\* @return Элемент от пользователя

\*/

int generate() override;

};

***IStreamGenerator.cpp:***

#include "IStreamGenerator.h"

IStreamGenerator::IStreamGenerator(std::istream& input)

:in{ input }

{

}

int IStreamGenerator::generate()

{

int value = 0;

this->in >> value;

return value;

}

***RandomGenerator.h:***

#pragma once

#include <random>

#include "Generator.h"

/\*\*

\* @brief Генератор случайного числа в заданном диапазоне

\*/

class RandomGenerator : public Generator

{

private:

std::uniform\_int\_distribution<int> distribution;

std::mt19937 generator;

public:

/\*\*

\* @brief Инициализация объекта типа RandomGenerator

\* @param min Минимальное значение для рандома

\* @param max Максимальное значение для рандома

\*/

RandomGenerator(int min, int max);

/\*\*

\* @brief Генерирует случайное число из заданного ранее диапазона

\* @return Случайные элемент из заданного диапазона

\*/

int generate() override;

};

***RandomGenerator.cpp:***

#include "RandomGenerator.h"

RandomGenerator::RandomGenerator(int min, int max)

{

this->generator = std::mt19937(std::random\_device{}());

this->distribution = std::uniform\_int\_distribution<int>(min, max);

}

int RandomGenerator::generate()

{

return this->distribution(this->generator);

}

***TaskBase.h:***

#pragma once

#include "../MatrixTask/Matrix.h"

using namespace miit;

/\*\*

\* @brief Базовый класс задания от которого можно отнаследовать свое задание с 2 подзаданиями

\*/

struct TaskBase

{

public:

/\*\*

\* @brief Чисто виртуальный деструктор

\*/

virtual ~TaskBase() = 0 {};

/\*\*

\* @brief Чисто виртуальный метод для выполениея первого подзадания

\*/

virtual Matrix task\_1() = 0;

/\*\*

\* @brief Чисто виртуальный метод для выполениея второго подзадания

\*/

virtual Matrix task\_2() = 0;

};

***Task.h:***

#pragma once

#include "TaskBase.h"

#include "../MatrixTask/Matrix.h"

/\*\*

\* @brief Класс задание, наследник базового задания, с 2 подзаданиями

\*/

struct Task : public TaskBase

{

private:

Matrix matrix;

public:

/\*\*

\* @brief Инициализирует объект типа "Task"

\* @param matrix Матрица для выполнения задания

\*/

Task(Matrix matrix);

/\*\*

\* @brief Решение первой подзадачи

\* @return Матрица выполненного задания

\*/

Matrix task\_1() override;

/\*\*

\* @brief Решение второй подзадачи

\* @return Матрица выполненного задания

\*/

Matrix task\_2() override;

};

***Task.cpp:***

#include "Task.h"

Task::Task(Matrix matrix)

:matrix{ matrix }

{

}

Matrix Task::task\_1()

{

Matrix result{ this->matrix };

int max = this->matrix.get\_abs\_max\_element();

for (size\_t i = 0; i < result.rows\_counts(); i++)

{

for (size\_t j = 0; j < result.columns\_counts(); j++)

{

if (result[i][j] == max)

{

result[i][j] = 0;

}

}

}

return result;

}

Matrix Task::task\_2()

{

Matrix result{};

std::vector<int> zero\_row;

for (size\_t i = 0; i < matrix.rows\_counts(); i++)

{

zero\_row.push\_back(0);

}

int max = this->matrix.get\_abs\_max\_element();

for (size\_t i = 0; i < matrix.rows\_counts(); i++)

{

result.add\_row(matrix[i]);

if (this->matrix.is\_element\_in\_row(max, i))

{

result.add\_row(zero\_row);

}

}

return result;

}

**Тесты:**

#include "pch.h"

#include "CppUnitTest.h"

#include "../MatrixTask/Matrix.h"

#include "../Generators/RandomGenerator.h"

using namespace miit;

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace Tests

{

TEST\_CLASS(Tests)

{

public:

TEST\_METHOD(Matrix\_EqualOperator\_ValidData\_Success)

{

// arrange

Matrix matrix\_1 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

Matrix matrix\_2 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

//act & assert

Assert::IsTrue(matrix\_1 == matrix\_2);

}

TEST\_METHOD(Matrix\_IndexOperator\_ValidData\_Success)

{

// arrange

Matrix matrix = Matrix({ {1, 2}, {3, 4} });

int success = 3;

// act & assert

Assert::IsTrue(matrix[1][0] == success);

}

TEST\_METHOD(Matrix\_IndexOperator\_IncorrectIndex\_ExeptionExpected)

{

// arrange

Matrix matrix = Matrix({ {1, 2}, {3, 4} });

// act & assert

Assert::ExpectException<std::out\_of\_range>([matrix]()

{double x = matrix[3][4]; });

}

TEST\_METHOD(Matrix\_RowsCounts\_ValidData\_Success)

{

// arrange

Matrix matrix = Matrix({ {1, 2}, {3, 4} });

int answer = 2;

// act

int temp = matrix.rows\_counts();

// assert

Assert::IsTrue(temp == answer);

}

TEST\_METHOD(Matrix\_ColumnsCounts\_ValidData\_Success)

{

// arrange

Matrix matrix = Matrix({ {1, 2}, {3, 4} });

int answer = 2;

// act

int temp = matrix.columns\_counts();

// assert

Assert::IsTrue(temp == answer);

}

TEST\_METHOD(Matrix\_ToString\_ValidData\_Success)

{

// arrange

Matrix matrix = Matrix({ {1, 2}, {3, 4}, {5, 6} });

// act & assert

Assert::IsTrue(matrix.to\_string() == "1\t2\t\n3\t4\t\n5\t6\t\n");

}

TEST\_METHOD(Matrix\_Transpose\_ValidData\_Success)

{

// arrange

Matrix matrix\_1 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

Matrix transposed = Matrix({ {1, 4, 7}, {2, 5, 8}, {3, 6, 9} });

// act

matrix\_1.transpose();

// assert

Assert::IsTrue(matrix\_1 == transposed);

}

TEST\_METHOD(Matrix\_AddRow\_ValidData\_Success)

{

// arrange

Matrix matrix\_1 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

std::vector<int> row{ 1,1,1 };

Matrix result = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {1, 1, 1} });

// act

matrix\_1.add\_row(row);

// assert

Assert::IsTrue(matrix\_1 == result);

}

TEST\_METHOD(Matrix\_AddRowIndex\_ValidData\_Success)

{

// arrange

Matrix matrix\_1 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

std::vector<int> row{ 1,1,1 };

Matrix result = Matrix({ {1, 1, 1}, {1, 2, 3}, {4, 5, 6}, {7, 8, 9}});

// act

matrix\_1.add\_row(row, 0);

// assert

Assert::IsTrue(matrix\_1 == result);

}

TEST\_METHOD(Matrix\_AddRowIndex\_IncorrectIndex\_ExeptionExpected)

{

// arrange & act & assert

Assert::ExpectException<std::out\_of\_range>([]()

{

Matrix matrix\_1 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

std::vector<int> row{ 1, 1, 1 };

matrix\_1.add\_row(row, 10);

});

}

TEST\_METHOD(Matrix\_AddColumn\_ValidData\_Success)

{

// arrange

Matrix matrix\_1 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

std::vector<int> column{ 1,1,1 };

Matrix result = Matrix({ {1, 2, 3, 1}, {4, 5, 6, 1}, {7, 8, 9, 1} });

// act

matrix\_1.add\_column(column);

// assert

Assert::IsTrue(matrix\_1 == result);

}

TEST\_METHOD(Matrix\_AddColumnIndex\_ValidData\_Success)

{

// arrange

Matrix matrix\_1 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

std::vector<int> column{ 1,1,1 };

Matrix result = Matrix({ {1, 1, 2, 3}, {1, 4, 5, 6}, {1, 7, 8, 9} });

// act

matrix\_1.add\_column(column, 0);

// assert

Assert::IsTrue(matrix\_1 == result);

}

TEST\_METHOD(Matrix\_AddColumnIndex\_IncorrectIndex\_ExeptionExpected)

{

// arrange & act & assert

Assert::ExpectException<std::out\_of\_range>([]()

{

Matrix matrix\_1 = Matrix({ {1, 2, 3}, {4, 5, 6}, {7, 8, 9} });

std::vector<int> column{ 1, 1, 1 };

matrix\_1.add\_column(column, 10);

});

}

TEST\_METHOD(MatrixGetMaxAbsElement\_ValidData\_Success)

{

// arrange

Matrix matrix = Matrix({{22, 1, -30}, {7, 23, 5}, {-3, 4, -12}});

int success\_value = -30;

// act

int temp = matrix.get\_abs\_max\_element();

// assert

Assert::IsTrue(temp == success\_value);

}

TEST\_METHOD(MatrixIsElementInRow\_ValidData\_Success)

{

// arrange

Matrix matrix = Matrix({ {22, 1, -30}, {7, 23, 5}, {-3, 4, -12} });

// act

int temp\_1 = matrix.is\_element\_in\_row(23, 1);

int temp\_2 = matrix.is\_element\_in\_row(23, 0);

// assert

Assert::IsTrue(temp\_1 == true && temp\_2 == false);

}

TEST\_METHOD(Matrix\_CtorGenerator\_IncorrectSize\_ExeptionExpected)

{

// arrange & act & assert

Assert::ExpectException<std::out\_of\_range>([]()

{

Generator\* gen = new RandomGenerator(10, 100);

Matrix matrix\_1 = Matrix(-3, -2, gen);

});

}

TEST\_METHOD(Matrix\_CtorMatrix\_IncorrectMatrixData\_ExeptionExpected)

{

// arrange & act & assert

Assert::ExpectException<std::out\_of\_range>([]()

{

Matrix matrix\_1 = Matrix({{ 1, 2, 3 }, { 1, 2, 3, 4 }, { 1, 2, 3 }});

});

}

};

}

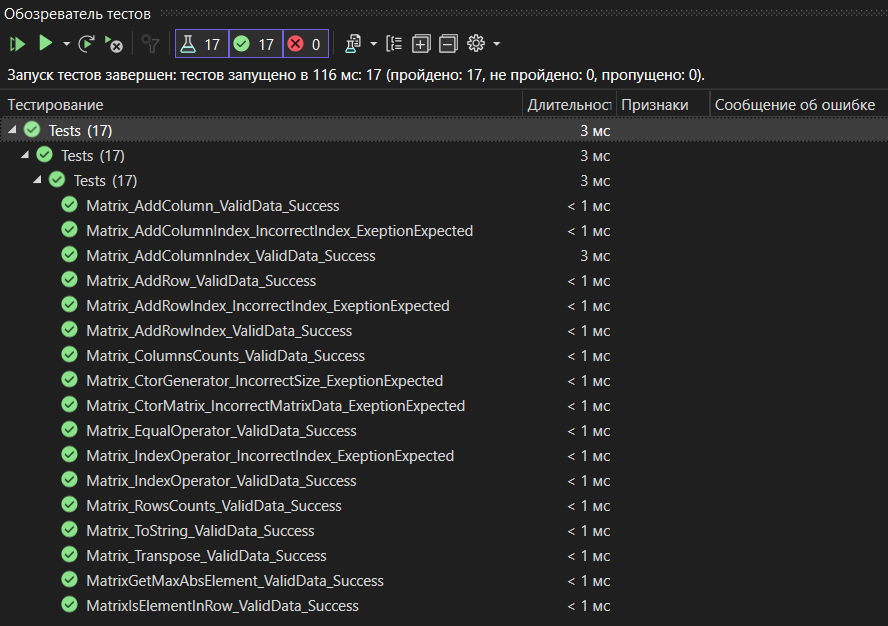
****

Рисунок 1 – прохождение тестов

**UML Диаграмма:**

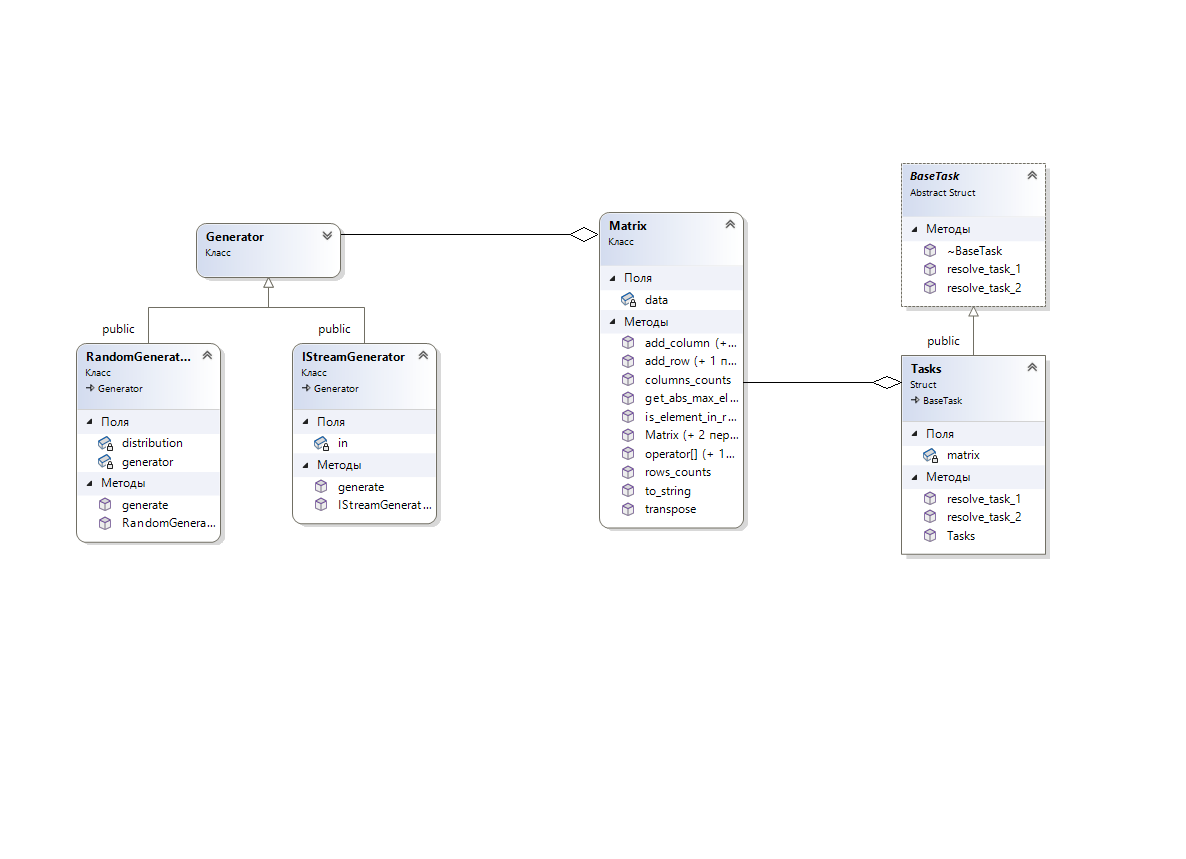


Рисунок 2 – UML диаграмма

****

Рисунок 3 – Approve