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

Омельченко.А.А

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

Задание 1

## Формулировка задачи

Тема 2 Фигура на плоскости

Абстрактный базовый класс Фигура на плоскости, который «рисует» заданную (в варианте) конкретную фигуру. «Рисует» в данном задании означает выводит информацию (сериализованную в строку посредством метода ToString) о координатах, задающих фигуру, в стандартный поток вывода и имеет метод чтения из стандартного потока ввода.

Реализовать класс Конкретная фигура (по вариантам, название взять из варианта). Предусмотреть создание фигуры как посредством задания координат её вершин, которые имеют тип Точка, так и посредством задания вершин парами чисел (абсцисса, ордината). Типом данных, определяющих абсциссу и ординату, является целый тип данных без знака. Максимальное значение абсциссы и ординаты ограничено разрешающей способностью экрана монитора (является константой, которая передается извне, например из main). Определить статический метод ToString(). Предусмотреть невозможность создания объекта при несоблюдении правил создания фигуры на плоскости (сторона многоугольника и радиус должны быть неотрицательным числом). Определить статический метод чтения из стандартного потока ввода, использующий переопределенный оператор сдвига вправо для класса Точка.

Для класса Точка определить все необходимые для дальнейшего использования операторы (операторы сравнения, сдвига влево и вправо, сложение и вычитание Точки и целого числа).

## UML - диаграмма алгоритма

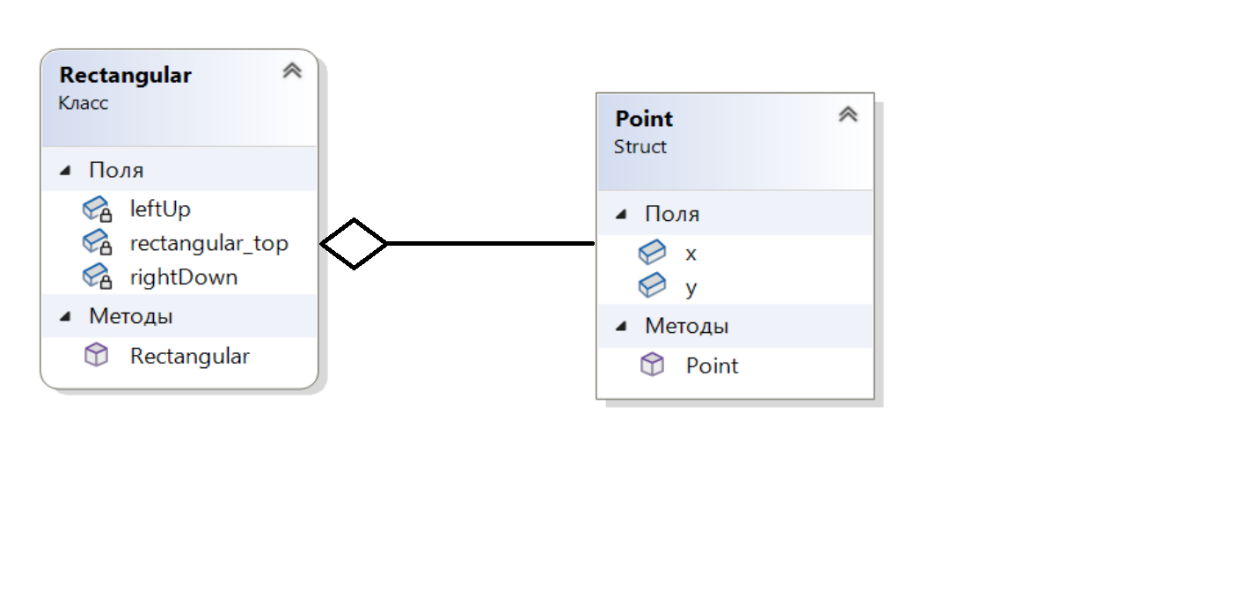


Рисунок 2 – UML диаграмма класса.

## Решение задачи на языке программирования C++

main.cpp:

#include "..\rectangular\rectangular.h"

#include<iostream>

#include<string>

using namespace std;

/\*

\*\brief Функция считывающая размеры

\*\param message - подсказка для пользователя

\*\return Возвращает размеры

\*/

double GetSize(const string& message);

void RecTwoPoints();

enum class creatingrec {

c\_OnePoint,

c\_TwoPoints

};

int main(){

setlocale(LC\_ALL, "rus");

int input = 0;

RecTwoPoints();

return 0;

}

void RecTwoPoints() {

double PointXU;

double PointYU;

double PointXD;

double PointYD;

std::cout << "Введите х левой верхней вершины" << "\n";

std::cin >> PointXU;

std::cout << "Введите у левой верхней вершины" << "\n";

std::cin >> PointYU;

std::cout << "Введите х правой нижней вершины" << "\n";

std::cin >> PointXD;

std::cout << "Введите у правой нижней вершины" << "\n";

std::cin >> PointYD;

auto PointUp=std::make\_shared<Point>(PointXU, PointYU);

auto pointDown=std::make\_shared<Point>(PointXD, PointYD);

Rectangular rec(PointUp, pointDown);

cout << rec;

}

double GetSize(const string& message) {

double size = -1;

cout << message;

cin >> size;

if (size < 0)

{

throw out\_of\_range("Incorrect size");

}

return size;

}

Rectangular.cpp:

#include"rectangular.h"

#include <vector>

#include<sstream>

Rectangular::Rectangular(std::shared\_ptr<Point>& leftUp, std::shared\_ptr<Point>& rightDown):

leftUp(leftUp),rightDown(rightDown)

{

rectangular\_top.push\_back(leftUp);

auto rightUp= std::make\_shared<Point>(rightDown->x, leftUp->y);

rectangular\_top.push\_back(rightUp);

rectangular\_top.push\_back(rightDown);

auto leftDown= std::make\_shared<Point>(leftUp->x, rightDown->y );

rectangular\_top.push\_back(leftDown);

}

std::ostream& operator<<(std::ostream& out, const Rectangular& rec)

{

std::string f;

for (int i = 0; i < rec.rectangular\_top.size(); i++) {

out <<\*rec.rectangular\_top[i]<< "\n";

}

return out;

};

Rectangular.h:

#pragma once

#include "point.h"

#include <vector>

#include <iostream>

using namespace PointClass;

class Rectangular{

private:

std::shared\_ptr<Point>& leftUp;

std::shared\_ptr<Point>& rightDown;

std::vector<std::shared\_ptr<Point>> rectangular\_top;

public:

Rectangular(std::shared\_ptr<Point>& leftUp, std::shared\_ptr<Point>& rightDown);

friend std::ostream& operator<< (std::ostream& out, const Rectangular& rec);

};

Point.cpp:

#include <sstream>

#include"point.h"

using namespace PointClass;

Point::Point(const double valueX ,const double valueY):x(valueX),y(valueY) {

}

std::wstring ToString(Point\* point) {

std::wstringstream buffer;

buffer << L"{ ";

buffer << point->x << L" " <<point->y << L" }";

return buffer.str();

}

std::wstring PointClass::ToString(const Point& point)

{

return std::wstring();

}

std::ostream& PointClass::operator<<(std::ostream& out, const Point& point) {

out << "Point(" << point.x << ", " << point.y << ")";

return out;

}

Point.h:

#pragma once

#include<iostream>

namespace PointClass {

struct Point;

std::wstring ToString(const Point& point);

std::wstring ToString(Point\* point);

struct Point {

Point(const double valueX = 0, const double valueY = 0);

double x;

double y;

friend std::ostream& operator<< (std::ostream& out, const Point& point);

};

}

## Решение тестов

Код тестов:

#include "pch.h"

#include "CppUnitTest.h"

#include"..\rectangular\rectangular.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace RectangularTests

{

TEST\_CLASS(RectangularTests)

{

public:

TEST\_METHOD(Point\_Valid\_Coordinates\_Success)

{

// arrange

Point p = Point(2, 3);

// act

double resultX = p.x;

double resultY = p.y;

// assert

Assert::AreEqual(2., resultX);

Assert::AreEqual(3.,resultY);

}

TEST\_METHOD(Rectangular\_Constructor\_First\_IsNotNull) {

//arrange

auto p = std::make\_shared<Point>(2, 3);

auto p2 = std::make\_shared<Point>(4, 5);

Rectangular rec = Rectangular(p,p2);

std::stringstream buffer;

std::string testString;

//act

buffer << rec;

testString = buffer.str();

//assert

Assert::IsNotNull(testString.c\_str());

}

TEST\_METHOD(Point\_Output\_Operator\_Overloading\_Success) {

//arrange

auto p = std::make\_shared<Point>(2, 2);

std::stringstream buffer;

std::string testString;

std::string goodString;

//act

buffer << \*p;

testString = buffer.str();

goodString = "Point(2, 2)";

//assert

Assert::AreEqual(testString, goodString);

}

TEST\_METHOD(Rectangular\_Output\_Operator\_Overloading\_Success) {

//arrange

auto p = std::make\_shared<Point>(2, 2);

auto p2 = std::make\_shared<Point>(3, 4);

Rectangular rec = Rectangular(p,p2);

std::stringstream buffer;

std::string testString;

std::string goodString;

//act

buffer << rec;

testString = buffer.str();

goodString = "Point(2, 2)\nPoint(3, 2)\nPoint(3, 4)\nPoint(2, 4)\n";

//assert

Assert::AreEqual(testString, goodString);

}

//ostream point, 2 конструктора (not null, неправильный ввод)

};

}

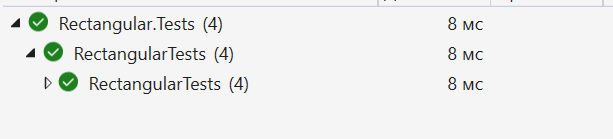


Рисунок 4 – Успешные тесты.

## Зачет задания в GitHub

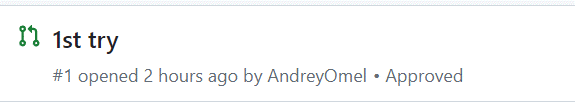


Рисунок 5 - Зачет задания в GitHub