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

Райхман И. А.

Проверил: к.т.н., доц.

Васильева М. А.

Москва 2023

**Содержание**

Задание 33

Задание 1

## Формулировка задачи

**Вариант 10 Линия**

Задается двумя точками. Координаты имеют тип **Точка** (структура, содержащая абсциссу и ординату). Стиль и цвет линии задается в конструкторе, также предусмотреть метод изменения стиля линии и цвета.

**Стиль линии** – класс, который содержит тип перечисление **Тип линии** (сплошная – solid, пунктирная – dash, точки – dot, точка-тире – dash-dot, две точки тире – dashdotdot) и толщину линии (целое число больше нуля). Предусмотрите стиль линии по умолчанию (с цветом линии – черный и толщиной линии, равной 1).

**Цвет** является классом, имеющим три поля red, green, blue, хранящие значение от 0 до 255 (в виде числа типа uint8\_t). Предусмотреть конструктор, принимающий три значения типа int. Если входные значения не принадлежат диапазону от 0 до 255, выбрасывать исключение. Переопределить операторы сдвига.

Для класса **Линия** переопределить операторы сдвига влево, написать статический метод чтения, для структуры **Точка** переопределить операторы сдвига и равенства.

## UML - диаграмма алгоритма

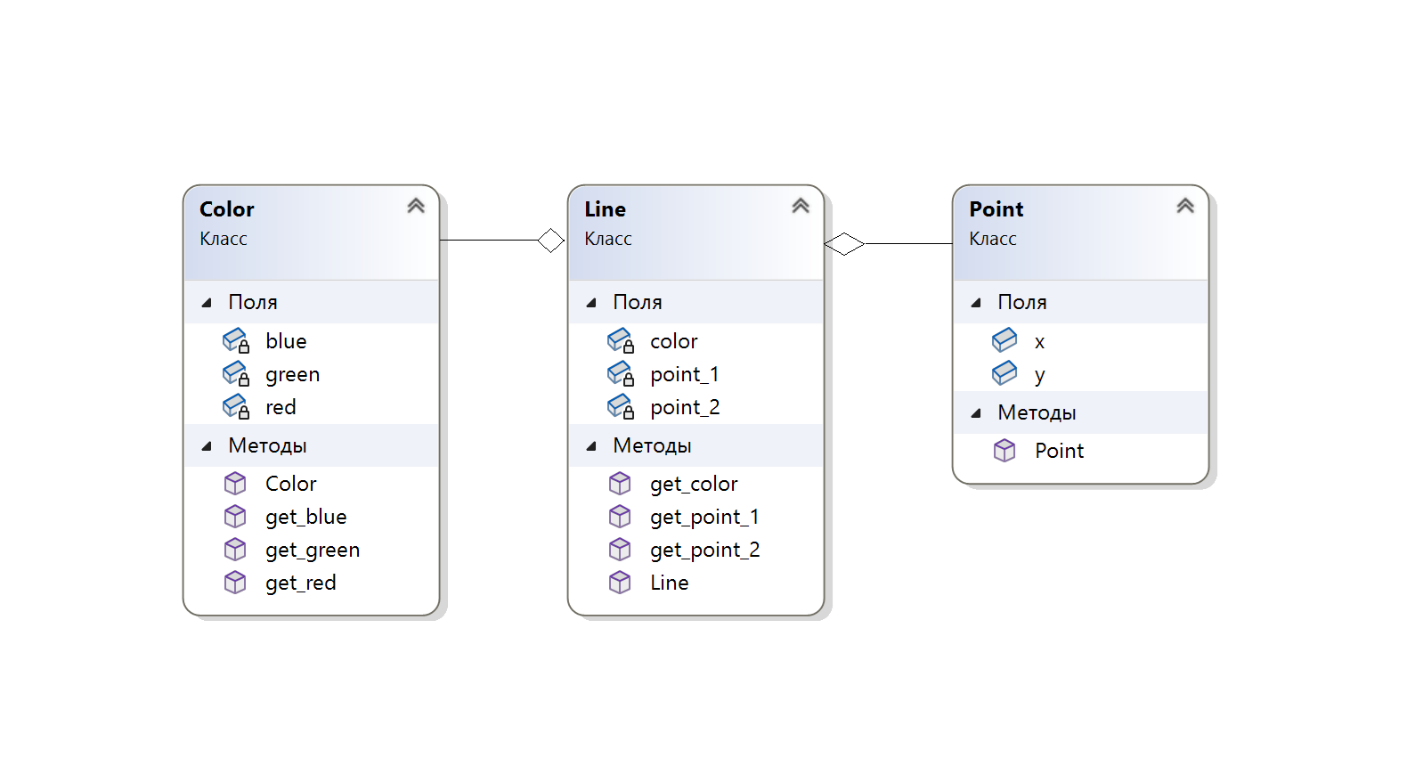


Рисунок 1 – UML диаграмма класса

## Решение задачи на языке программирования C++

main.cpp:

#include <iostream>

#include "..\Line\Line.h"

/\*\*

\* \brief Точка входа в программу.

\* \return Возвращает 0 в случае успеха.

\*/

int main()

{

std::shared\_ptr<Point> point\_1 = std::make\_shared<Point>(123.0, 21.0);

std::shared\_ptr<Point> point\_2 = std::make\_shared<Point>(57.0, 35.0);

std::shared\_ptr<Color> color = std::make\_shared<Color>(12, 247, 102);

Line line\_1(point\_1, point\_2, color);

std::cout << line\_1;

return 0;

}

Line.h:

#pragma once

#include <iostream>

#include "Point.h"

#include "Color.h"

#include <memory>

/\*\*

\* \class line

\* \brief Класс, представляющий линию, определенную двумя точками и цветом.

\*/

class Line

{

public:

/\*\*

\* \brief Конструктор класса line.

\* \param point\_1 Первая точка линии.

\* \param point\_2 Вторая точка линии.

\* \param color Цвет линии.

\*/

Line(std::shared\_ptr<Point> point\_1, std::shared\_ptr<Point> point\_2, std::shared\_ptr<Color> color);

/\*\*

\* \brief Получает первую точку линии.

\* \return Первая точка линии.

\*/

std::shared\_ptr<Point> get\_point\_1() const;

/\*\*

\* \brief Получает вторую точку линии.

\* \return Вторая точка линии.

\*/

std::shared\_ptr<Point> get\_point\_2() const;

/\*\*

\* \brief Получает цвет линии.

\* \return Цвет линии.

\*/

std::shared\_ptr<Color> get\_color() const;

/\*\*

\* \brief Перегруженный оператор вывода в поток.

\* \param out Поток вывода.

\* \param line Линия для вывода.

\* \return Поток вывода.

\*/

friend std::ostream& operator<< (std::ostream& out, const Line& line);

private:

std::shared\_ptr<Color> color;

std::shared\_ptr<Point> point\_1;

std::shared\_ptr<Point> point\_2;

};

Line.cpp:

#include "Line.h"

Line::Line(std::shared\_ptr<Point> point\_1, std::shared\_ptr<Point> point\_2, std::shared\_ptr<Color> color)

: point\_1(point\_1), point\_2(point\_2), color(color)

{

if (point\_1 == point\_2)

{

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

}

}

std::shared\_ptr<Point> Line::get\_point\_1() const

{

return this->point\_1;

}

std::shared\_ptr<Point> Line::get\_point\_2() const

{

return this->point\_2;

}

std::shared\_ptr<Color> Line::get\_color() const

{

return this->color;

}

std::ostream& operator<<(std::ostream& out, const Line& line)

{

out << "line(" << \*line.get\_point\_1() << ", " << \*line.get\_point\_2() << ", " << \*line.get\_color() << ")";

return out;

}

Point.h:

#pragma once

#include <iostream>

#include "HelpMath.h"

/\*\*

\* \struct point

\* \brief Структура, представляющая точку в двумерном пространстве.

\*/

class Point

{

public:

/\*\*

\* \brief Конструктор по умолчанию.

\* \param value\_x Значение координаты x точки.

\* \param value\_y Значение координаты y точки.

\*/

Point(const double value\_x, const double value\_y);

/\*\*

\* \brief Перегруженный оператор сравнения.

\* \param lha Левый операнд.

\* \param rha Правый операнд.

\* \return true, если точки равны; false в противном случае.

\*/

friend bool operator== (const Point& lha, const Point& rha);

/\*\*

\* \brief Перегруженный оператор вывода в поток.

\* \param out Поток вывода.

\* \param point Точка для вывода.

\* \return Поток вывода.

\*/

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

double x;

double y;

};

Point.cpp:

#include "Point.h"

#include <sstream>

Point::Point(const double value\_x, const double value\_y)

: x(value\_x), y(value\_y)

{

}

bool operator==(const Point& lha, const Point& rha)

{

return (is\_double\_equal(lha.x, rha.x) && is\_double\_equal(lha.y, rha.y));

}

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

{

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

return out;

}

Color.h:

#pragma once

#include <iostream>

#include "Point.h"

/\*\*

\* \class color

\* \brief Класс, представляющий цвет в формате RGB.

\*/

class Color

{

public:

/\*\*

\* \brief Конструктор класса color.

\* \param red Значение красного цвета.

\* \param green Значение зеленого цвета.

\* \param blue Значение синего цвета.

\*/

Color(const int red = 0, const int green = 0, const int blue = 0);

/\*\*

\* \brief Получает значение красного цвета.

\* \return Значение красного цвета.

\*/

int get\_red() const;

/\*\*

\* \brief Получает значение зеленого цвета.

\* \return Значение зеленого цвета.

\*/

int get\_green() const;

/\*\*

\* \brief Получает значение синего цвета.

\* \return Значение синего цвета.

\*/

int get\_blue() const;

/\*\*

\* \brief Перегруженный оператор вывода в поток.

\* \param out Поток вывода.

\* \param color Цвет для вывода.

\* \return Поток вывода.

\*/

friend std::ostream& operator<< (std::ostream& out, const Color& color);

private:

int red; /\*\*< Значение красного цвета. \*/

int green; /\*\*< Значение зеленого цвета. \*/

int blue; /\*\*< Значение синего цвета. \*/

};

Color.cpp:

#include "Color.h"

Color::Color(const int red, const int green, const int blue)

: red(red), green(green), blue(blue)

{

if (red < 0 || red > 255 || green < 0 || green > 255 || blue < 0 || blue > 255)

{

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

}

}

int Color::get\_red() const

{

return this->red;

}

int Color::get\_green() const

{

return this->green;

}

int Color::get\_blue() const

{

return this->blue;

}

std::ostream& operator<<(std::ostream& out, const Color& color)

{

out << "color(" << color.get\_red() << ", " << color.get\_green() << "," << color.get\_blue() << ")";

return out;

}

HelpMath.h:

#pragma once

#include <limits>

/\*\*

\* \brief Сравнивает два значения типа double на равенство с учетом заданной точности.

\* \param value\_1 Первое значение типа double.

\* \param value\_2 Второе значение типа double.

\* \param precision Точность сравнения (по умолчанию - наименьшее положительное число типа double).

\* \return true, если значения value\_1 и value\_2 считаются равными с заданной точностью, иначе false.

\*/

bool is\_double\_equal(const double value\_1, const double value\_2, const double precision = std::numeric\_limits<double>::min());Department.cpp:

#include "Department.h"

Department::Department(const std::string nameDepartment): nameDepartment(nameDepartment)

{

}

std::ostream& operator<<(std::ostream& os, Department& department)

{

os << department.nameDepartment;

return os;

}

std::vector<std::shared\_ptr<Speciality>> Department::GetSpecialities()

{

return this->specialities;

}

void Department::AddSpeciality(std::shared\_ptr<Speciality>& speciality)

{

specialities.push\_back(speciality);

}

std::string Department::ToString()

{

std::stringstream buffer;

buffer << nameDepartment;

return buffer.str();

}

HelpMath.cpp:

#include "HelpMath.h"

#include <cmath>

/\*\*

\* \brief Сравнивает два значения типа double на равенство с учетом заданной точности.

\* \param value\_1 Первое значение типа double.

\* \param value\_2 Второе значение типа double.

\* \param precision Точность сравнения.

\* \return true, если значения value\_1 и value\_2 считаются равными с заданной точностью, иначе false.

\*/

bool is\_double\_equal(const double value\_1, const double value\_2, const double precision)

{

return std::abs(value\_1 - value\_2) < precision;

}

## Решение тестов

Код тестов:

#include "CppUnitTest.h"

#include "../Line/Line.h"

#include "../Line/Point.h"

#include "../Line/HelpMath.h"

#include <iostream>

#include <memory>

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace LineTests

{

TEST\_CLASS(LineTests)

{

public:

TEST\_METHOD(ConstructorTest\_ValidData\_Success)

{

std::shared\_ptr<Point> point\_1 = std::make\_shared<Point>(123.0, 21.0);

std::shared\_ptr<Point> point\_2 = std::make\_shared<Point>(57.0, 35.0);

std::shared\_ptr<Color> color = std::make\_shared<Color>(12, 247, 102);

Line line\_1(point\_1, point\_2, color);

Assert::IsNotNull(&line\_1);

}

TEST\_METHOD(GetPoint1Test\_ValidData\_Success)

{

std::shared\_ptr<Point> point\_1 = std::make\_shared<Point>(123.0, 21.0);

std::shared\_ptr<Point> point\_2 = std::make\_shared<Point>(57.0, 35.0);

std::shared\_ptr<Color> color = std::make\_shared<Color>(12, 247, 102);

Line line\_1(point\_1, point\_2, color);

std::stringstream buffer;

std::string expected = "Point(123, 21)";

buffer << \*line\_1.get\_point\_1();

Assert::AreEqual(expected, buffer.str());

}

TEST\_METHOD(GetPoint2Test\_ValidData\_Success)

{

std::shared\_ptr<Point> point\_1 = std::make\_shared<Point>(123.0, 21.0);

std::shared\_ptr<Point> point\_2 = std::make\_shared<Point>(57.0, 35.0);

std::shared\_ptr<Color> color = std::make\_shared<Color>(12, 247, 102);

Line line\_1(point\_1, point\_2, color);

std::stringstream buffer;

std::string expected = "Point(57, 35)";

buffer << \*line\_1.get\_point\_2();

Assert::AreEqual(expected, buffer.str());

}

TEST\_METHOD(GetColorTest\_ValidData\_Success)

{

std::shared\_ptr<Point> point\_1 = std::make\_shared<Point>(123.0, 21.0);

std::shared\_ptr<Point> point\_2 = std::make\_shared<Point>(57.0, 35.0);

std::shared\_ptr<Color> color = std::make\_shared<Color>(12, 247, 102);

Line line\_1(point\_1, point\_2, color);

std::stringstream buffer;

std::string expected = "color(12, 247,102)";

buffer << \*line\_1.get\_color();

Assert::AreEqual(expected, buffer.str());

}

TEST\_METHOD(OutputOperatorTest\_ValidData\_Success)

{

std::shared\_ptr<Point> point\_1 = std::make\_shared<Point>(123.0, 21.0);

std::shared\_ptr<Point> point\_2 = std::make\_shared<Point>(57.0, 35.0);

std::shared\_ptr<Color> color = std::make\_shared<Color>(12, 247, 102);

Line line\_1(point\_1, point\_2, color);

std::string expected = "line(Point(123, 21), Point(57, 35), color(12, 247,102))";

std::stringstream buffer;

buffer << line\_1;

Assert::AreEqual(expected, buffer.str());

}

};

TEST\_CLASS(ColorTests)

{

public:

TEST\_METHOD(ConstructorTest\_ValidData\_Success)

{

Color color\_1(10,20,30);

Assert::IsNotNull(&color\_1);

}

TEST\_METHOD(GetRedColorTest\_ValidData\_Success)

{

Color color\_1(10, 20, 30);

std::stringstream buffer;

std::string expected = "10";

buffer << color\_1.get\_red();

Assert::AreEqual(expected, buffer.str());

}

TEST\_METHOD(GetGreenColorTest\_ValidData\_Success)

{

Color color\_1(10, 20, 30);

std::stringstream buffer;

std::string expected = "20";

buffer << color\_1.get\_green();

Assert::AreEqual(expected, buffer.str());

}

TEST\_METHOD(GetBlueColorTest\_ValidData\_Success)

{

Color color\_1(10, 20, 30);

std::stringstream buffer;

std::string expected = "30";

buffer << color\_1.get\_blue();

Assert::AreEqual(expected, buffer.str());

}

TEST\_METHOD(OutputOperatorTest\_ValidData\_Success)

{

Color color\_1(10, 20, 30);

std::string expected = "color(10, 20,30)";

std::stringstream buffer;

buffer << color\_1;

Assert::AreEqual(expected, buffer.str());

}

};

TEST\_CLASS(HelpMathTests)

{

public:

TEST\_METHOD(IsDoubleEqualTest\_ValidData\_Success)

{

is\_double\_equal(10.0, 10.0);

Assert::IsFalse(is\_double\_equal == 0);

}

};

TEST\_CLASS(PointTests)

{

public:

TEST\_METHOD(ConstructorTest\_ValidData\_Success)

{

Point point\_1(1.0,2.0);

Assert::IsNotNull(&point\_1);

}

TEST\_METHOD(EqualOperatorTest\_ValidData\_Success)

{

Point point\_1(1.0, 2.0);

Point point\_2(1.0, 2.0);

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

}

TEST\_METHOD(OutputOperatorTest\_ValidData\_Success)

{

Point point\_1(1.0, 2.0);

std::string expected = "Point(1, 2)";

std::stringstream buffer;

buffer << point\_1;

Assert::AreEqual(expected, buffer.str());

}

};

}

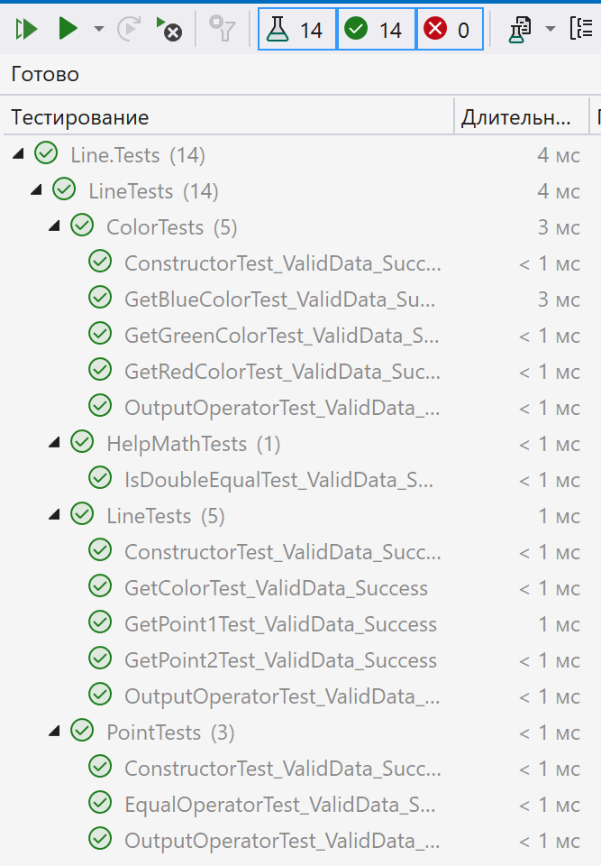


Рисунок 2 – Успешные тесты

## Зачет задания в GitHub

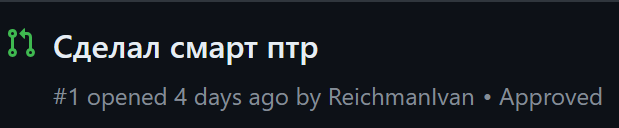


Рисунок 3 - Зачет задания в GitHub