МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 3**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Динамический выбор типа объектов»**

*Вариант № 7*

Выполнил:

Студент группы

Ктбо2-8

Жалнин Д. И.

Проверил:

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# Оглавление

[Оглавление 2](#_Toc58966437)

[1 Формулировка задания 4](#_Toc58966438)

[2 Спецификация классов 5](#_Toc58966439)

[2.1 Класс Point 5](#_Toc58966440)

[2.2 Класс MyShape 6](#_Toc58966441)

[2.3 Класс Square 8](#_Toc58966442)

[2.4 Класс Rectangle 8](#_Toc58966443)

[2.5 Класс Shapes 9](#_Toc58966444)

[2.6 Класс Interactor 9](#_Toc58966445)

[2.7 Класс MyException 10](#_Toc58966446)

[2.8 Класс ShapeType 11](#_Toc58966447)

[3 Диаграмма классов 11](#_Toc58966448)

[4 Листинг программы 12](#_Toc58966449)

[4.1 lab3.cpp 12](#_Toc58966450)

[4.2 Interactor.h 12](#_Toc58966451)

[4.3 Interactor.cpp 12](#_Toc58966452)

[4.4 Shapes.h 15](#_Toc58966453)

[4.5 Shapes.cpp 16](#_Toc58966454)

[4.6 MyShape.h 17](#_Toc58966455)

[4.7 MyShape.cpp 18](#_Toc58966456)

[4.8 Rectangle.h 19](#_Toc58966457)

[4.9 Rectangle.cpp 20](#_Toc58966458)

[4.10 Square.h 20](#_Toc58966459)

[4.11 Square.cpp 21](#_Toc58966460)

[4.12 MyException.h 21](#_Toc58966461)

[4.13 ShapeType.h 21](#_Toc58966462)

[4.14 Point.h 21](#_Toc58966463)

[4.15 Point.cpp 22](#_Toc58966464)

# 1 Формулировка задания

В лабораторной работе должна быть создана программа, создающая объекты двух классов – квадрат (**Square**) и прямоугольник (**Rectangle**). Эти классы должны быть производными от класса **Shape**.

Для каждого типа фигуры вычисляются площадь, центр тяжести и другие атрибуты. Кроме этого, должны быть предусмотрены виртуальные методы Вращения (**Rotate**) и Перемещения (**Move**). При выполнении работы необходимо: − разработать соответствующие классы, конструкторы, поля и методы − поля класса сделать закрытыми; для чтения и изменения их значений определить открытые методы; – предусмотреть во всех вариантах консольный ввод данных для создания объектов и консольный вывод результатов. Запрещено пользоваться контейнерами STL, за исключением **string**.

# 2 Спецификация классов

## 2.1 Класс Point

Класс-обертка для значений координат в плоскости.

Приватные поля:

**double x = 0**

**double y = 0**

Публичные поля:

**Point() = default**

/\* Стандартный конструктор без параметров, для возможности поздней инициализации полей объекта \*/

**Point(double x, double y)**

/\* Конструктор, устанавливающий передаваемые значения в соответствующие поля объекта \*/

**Point& operator = (const Point& old)**

**= default;**

/\* Определение стандартной операции копирования через оператор «**=**» (Значения **x** и **y** будут скопированы из объекта **old**) \*/

**double getX() const**

**double getY() const**

**void setX(double newX)**

**void setY(double newY)**

// Геттеры и сеттеры для переменных **x** и **y**

**void moveX(double l)**

**void moveY(double l)**

//Методы, изменяющие поля **x** и **y** в зависимости от передаваемых значений

## 2.2 Класс MyShape

Родительский класс для создаваемых фигур.

Приватные поля:

**Point \*\*pointArray = nullptr**

/\* Указатель на массив указателей на объекты **Point**. Каждый объект представляет собой координату вершины фигуры \*/

**int pointsCount = 0**

// Количество вершин в фигуре

**Point \*massCenter = nullptr**

// Указатель на объект **Point**, обозначающий центр масс фигуры

«protected» поля:

**Point \*calculateMassCenter()**

// Метод для вычисления центра масс фигуры

**void setPoint(int i, Point \*p)**

// Метод для установки указателя **p** в массив **pointArray** по индексу **i**

Публичные поля:

**MyShape(int pointsCount, Point \*\*pointArray)**

/\* Конструктор, который принимает на вход значение количества точек в фигуре и координаты этих точек и, затем, сохраняет их в соответствующие поля \*/

**virtual ShapeType getType()**

/\* Виртуальный метод, возвращающий тип созданного объекта в виде **enum** – SQUARE, RECTANGLE или NONE \*/

**virtual double getArea()**

/\* Виртуальный метод, вычисляющий и возвращающий площадь фигуры \*/

**virtual void rotate(double alpha)**

/\* Виртуальный метод, реализующий вращение фигуры на заданный угол (в радианах) в двумерном пространстве \*/

**virtual void move(double x, double y)**

/\* Виртуальный метод, реализующий перемещение фигуры на заданные величины по оси **x** и **y** в двумерном пространстве \*/

**~MyShape()**

/\*Деструктор класса, удаляющий объекты **pointArray** (в том числе объекты Point, которые в нем хранятся) и **massCenter** \*/

**Point \*getPoint(int index)**

/\* Метод, возвращающий указатель на объект Point из массива **pointArray** по переданному индексу \*/

**double getOutRadius()**

/\* Метод, вычисляющий и возвращающий размер описанного вокруг фигуры радиуса \*/

**Point \*getMassCenter()**

/\* Метод, возвращающий вычисленный центр масс фигуры \*/

**int getCount() const**

/\* Метод, возвращающий число точек в фигуре \*/

## 2.3 Класс Square

Дочерний от класса **MyShape**, реализующий тип фигуры «Квадрат».

Публичные поля:

**Square(Point\* p1, Point\* p2)**

/\* Объект принимает на вход две точки в пространстве – верхнюю левую и верхнюю правую, и затем вычисляет координаты двух оставшихся точек. Если первая координата будет расположена правее второй, остальные точки будут расположены выше переданных в конструктор, иначе – ниже \*/

**double getArea() override**

// Переопределенный метод, возвращающий вычисленную площадь фигуры

**ShapeType getType() override**

// Переопределенный метод, возвращающий тип фигуры (**SQUARE**)

## 2.4 Класс Rectangle

Дочерний от класса **MyShape**, реализующий тип фигуры «Прямоугольник».

Публичные поля:

**Rectangle(Point\* p1, Point\* p2, Point\* p3, Point\* p4);**

**Rectangle(Point\* p1, Point\* p2, Point\* p3);**

/\* В отличие от квадрата, конструкторы класса прямоугольника могут принимать на вход или 3 точки (верхняя левая, верхняя правая, нижняя левая) или все 4. В первом случае будет вычислена оставшаяся точка \*/

**double getArea() override;**

// Переопределенный метод, возвращающий вычисленную площадь фигуры

**ShapeType getType() override {**

**return RECTANGLE;**

**}**// Переопределенный метод, возвращающий тип фигуры (**RECTANGLE**)

## 2.5 Класс Shapes

Утилитарный класс для различных взаимодействий с объектами класса **MyShape**. Содержит методы создания новых объектов-фигур.

Статические публичные поля:

**static MyShape \*createShape(Point \*p1, Point \*p2);**

**static MyShape \*createShape(Point \*p1, Point \*p2, Point \*p3, Point \*p4);**

**static MyShape \*createShape(Point \*p1, Point \*p2, Point \*p3);**

/\* Три метода, которые создают правильный тип фигуры в зависимости от переданных координат. Если координаты не являются координатами квадрата или прямоугольника, будет брошено исключение \*/

**static MyShape\* createShape(int count, Point\*\* points);**

/\* Метод, принимающий на вход количество точек и массив с их координатами и вызывающий соответствующие методы создания. Если кол-во точек не лежит в диапазоне от 2 до 4, будет брошено исключение \*/

**static double compare(MyShape \*s1, MyShape \*s2);**

/\* Метод, вычисляющий разницу между площадями первой и второй фигуры \*/

**static bool isIntersect(MyShape \*s1, MyShape \*s2);**

// Метод, определяющий факт пересечения двух фигур

**static bool isInclude(MyShape \*s1, MyShape \*s2);**

// Метод, определяющий факт полного наложения **s1** на **s2**

## 2.6 Класс Interactor

Реализует взаимодействие пользователя с программой через ввод/вывод в консоли.

Приватные поля:

**MyShape\* shape1 = nullptr;**

**MyShape\* shape2 = nullptr;**

// Указатели на две фигуры, с которыми может работать пользователь

**bool running = true;**

// Флаг работы программы (выполнения циклического ввода и вывода)

Публичные поля:

**void showMenu()**

**int getMenuNum()**

**void execute(int cmd)**

**static bool boolChoice(const char\* text, const char\* v1, const char\* v2)**

**void loop()**

**~Interactor(){**

**delete shape1**

**delete shape2**

**}**

## 2.7 Класс MyException

Реализация класса для исключений.

**class MyException : public std::exception {**

**private:**

**const char\* reason;**

**public:**

**MyException(const char\* reason) {**

**this->reason = reason;**

**}**

**const char\* what() const noexcept override {**

**return reason;**

**}**

**};**

## 2.8 Класс ShapeType

Класс-перечисление, используемый в работе программы для указания типа фигуры:

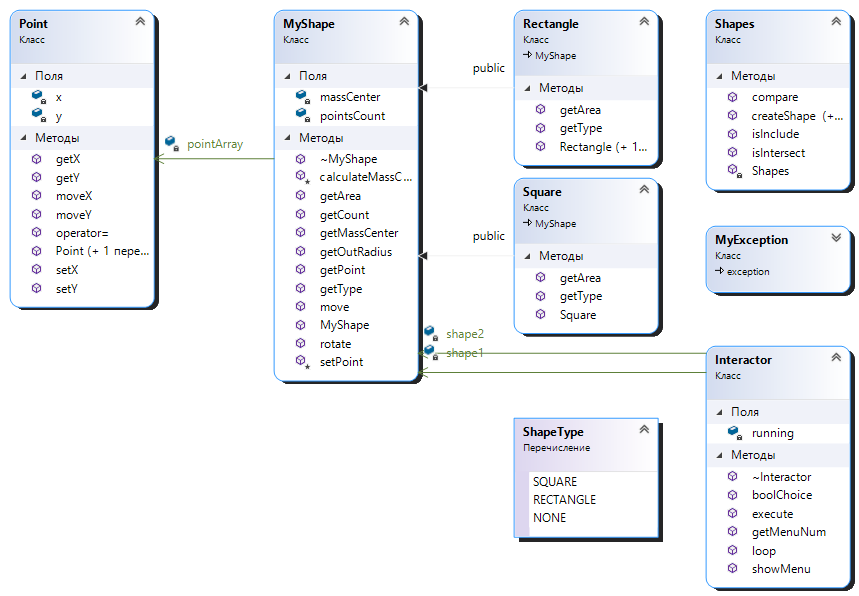
**enum ShapeType**

**{**

**SQUARE, RECTANGLE, NONE**

**};**

# 3 Диаграмма классов



*Рисунок 1 – UML диаграмма классов.*

# 4 Листинг программы

## 4.1 lab3.cpp

#include "Interactor.h"

#include <vld.h>

int main()

{

Interactor\* interactor = new Interactor();

interactor->loop();

delete interactor;

}

## 4.2 Interactor.h

#pragma once

#include "MyShape.h"

#include "Shapes.h"

#include "ShapeType.h"

class Interactor {

private:

MyShape\* shape1 = nullptr;

MyShape\* shape2 = nullptr;

bool running = true;

public:

void showMenu();

int getMenuNum();

void execute(int cmd);

static bool boolChoice(const char\* text, const char\* v1, const char\* v2);

void loop();

~Interactor(){

delete shape1;

delete shape2;

}

};

## 4.3 Interactor.cpp

#include "Interactor.h"

#include "MyException.h"

#include <iostream>

using namespace std;

void Interactor::showMenu() {

cout<<"1 - Create/recreate\n"

<<"2 - Get type\n"

<<"3 - Get coords\n"

<<"4 - Get area\n"

<<"5 - Get radius of the circumscribed circle\n"

<<"6 - Move\n"

<<"7 - Rotate\n"

<<"8 - Compare\n"

<<"9 - Center of mass\n"

<<"0 - Exit\n";

}

int Interactor::getMenuNum() {

int x;

cin >> x;

if (x<0 || x>10){

throw MyException("Incorrect menu number!");

}

return x;

}

void Interactor::execute(int cmd) {

bool b;

MyShape\*\* localShape = nullptr;

if (cmd != 8 && cmd != 0) {

if (boolChoice("Choice figure", "First", "Second")) {

localShape = &shape1;

} else {

localShape = &shape2;

}

if (cmd!=1 && (\*localShape == nullptr || localShape == nullptr)) {

throw MyException("Shape is not initialized!");

}

}

switch (cmd) {

case 1: {

int c;

cout << "Enter count of points (2-4):\n";

cin >> c;

if (c <= 0){

throw MyException("Incorrect count of points!");

break;

}

Point\*\* points = new Point \* [c] {};

for (int i = 0; i < c; i++) {

if (points[i] && points[i] != nullptr) {

delete points[i];

}

points[i] = new Point();

cout << "\nPoint " << i + 1 << "\n";

cout << "X>\n";

int x;

cin >> x;

points[i]->setX(x);

cout << "Y>\n";

int y;

cin >> y;

points[i]->setY(y);

cout<<"\n";

}

if (\*localShape!=nullptr)

delete \*localShape;

\*localShape = Shapes::createShape(c, points);

cout << "OK\n";

break;

}

case 2:

cout<< (((\*localShape)->getType() == SQUARE) ? "\nSQUARE\n" : (((\*localShape)->getType() == RECTANGLE) ? "\nRECTANGLE\n" : "NONE"));

break;

case 3:

for (int i = 0; i < (\*localShape)->getCount(); i++) {

cout<<"\nPoint "<<i<<"\n";

cout << "X - "

<<(\*localShape)->getPoint(i)->getX()

<<"\nY - "

<<(\*localShape)->getPoint(i)->getY()

<<"\n";

}

break;

case 4:

cout<<"Area - "<<(\*localShape)->getArea()<<"\n";

break;

case 5:

cout<<"Radius - "<<(\*localShape)->getOutRadius()<<"\n";

break;

case 6:

double x,y;

cout<<"X>";

cin>>x;

cout<<"\nY>";

cin>>y;

(\*localShape)->move(x,y);

break;

case 7:

double alpha;

cout<<"Rad>";

cin>>alpha;

(\*localShape)->rotate(alpha);

break;

case 8: {

if (shape1 == nullptr || shape2 == nullptr) {

throw MyException("Shapes are not initialized!");

}

double dif = Shapes::compare(shape1, shape2);

if (dif == 0) {

cout << "Shapes are equals\n";

} else {

if (dif < 0) {

cout << "Shape 2 is " << -dif << " more then Shape 1";

} else {

cout << "Shape 1 is " << dif << " more then Shape 2";

}

}

break;

}

case 9:

cout<<"Center of mass - X = "<<(\*localShape)->getMassCenter()->getX()<<" Y = "<<(\*localShape)->getMassCenter()->getY()<<"\n";

break;

case 0:

delete localShape;

running = false;

default: return;

}

}

void Interactor::loop() {

while(running) {

try {

showMenu();

execute(getMenuNum());

cout<<"\n";

} catch (MyException &e) {

cout << "\n" << e.what() << "\n\n";

}

}

}

bool Interactor::boolChoice(const char \*text, const char \*v1, const char \*v2) {

int x;

cout<<text<<"\n"<<"1 - "<<v1<<"\n"<<"2 - "<<v2<<"\n";

cin >> x;

if (x==1){

return true;

}

if (x == 2){

return false;

}

throw MyException("Incorrect choice!");

}

## 4.4 Shapes.h

#pragma once

#include <iostream>

#include "Point.h"

#include "Rectangle.h"

#include "Square.h"

class Shapes {

private:

Shapes() {}

public:

static MyShape \*createShape(Point \*p1, Point \*p2);

static MyShape \*createShape(Point \*p1, Point \*p2, Point \*p3, Point \*p4);

static MyShape \*createShape(Point \*p1, Point \*p2, Point \*p3);

static MyShape\* createShape(int count, Point\*\* points);

static double compare(MyShape \*s1, MyShape \*s2);

static bool isIntersect(MyShape \*s1, MyShape \*s2);

static bool isInclude(MyShape \*s1, MyShape \*s2);

};

## 4.5 Shapes.cpp

#include "Shapes.h"

MyShape \*Shapes::createShape(Point \*p1, Point \*p2) {

if (p1== nullptr || p2 == nullptr){

delete p1, p2;

throw MyException("Null pointer exception! (createShape2)");

}

Square \*s = new Square(p1, p2);

return s;

}

MyShape \*Shapes::createShape(Point \*p1, Point \*p2, Point \*p3, Point \*p4) {

if (p1== nullptr || p2 == nullptr || p3 == nullptr || p4 == nullptr){

delete p1, p2, p3, p4;

throw MyException("Null pointer exception! (createShape4)");

}

if (p3->getX() - p1->getX() != p2->getX() - p4->getX() && p3->getY() - p1->getY() != p2->getY() - p4->getY()) {

delete p1, p2, p3, p4;

throw MyException("Wrong coordinates!");

}

MyShape \*s = nullptr;

if (

(p1->getX() + (p2->getY() - p1->getY())) == p3->getX() &&

(p1->getY() - (p2->getX() - p1->getX())) == p3->getY()

) {

delete p3, p4;

s = new Square(p1, p2);

return s;

} else {

s = new Rectangle(p1, p2, p3, p4);

return s;

}

}

MyShape \*Shapes::createShape(Point \*p1, Point \*p2, Point \*p3) {

if (p1== nullptr || p2 == nullptr || p3 == nullptr){

delete p1, p2, p3;

throw MyException("Null pointer exception! (createShape3)");

}

if (

((p2->getX() - p1->getX()) \* (p3->getX() - p1->getX()) +

(p2->getY() - p1->getY()) \* (p3->getY() - p1->getY())) /

(sqrt(pow(p2->getX() - p1->getX(), 2) + pow(p2->getY() - p1->getY(), 2)) \*

sqrt(pow(p3->getX() - p1->getX(), 2) + pow(p3->getY() - p1->getY(), 2))) != 0

) {

throw MyException("Wrong coordinates!");

}

if (

(p1->getX() + (p2->getY() - p1->getY())) == p3->getX() &&

(p1->getY() - (p2->getX() - p1->getX())) == p3->getY()

) {

delete p3;

return new Square(p1, p2);

} else {

return new Rectangle(p1, p2, p3);

}

}

MyShape \*Shapes::createShape(int count, Point \*\*points) {

if (points == nullptr){

throw MyException("Null pointer exception! (createShape0)");

}

MyShape\* shape = nullptr;

switch (count) {

case (2):

shape = createShape(points[0], points[1]);

delete points;

return shape;

case (3):

shape = createShape(points[0], points[1], points[2]);

delete points;

return shape;

case (4):

shape = createShape(points[0], points[1], points[2], points[3]);

delete points;

return shape;

default:

throw MyException("Incorrect count of points!");

}

}

double Shapes::compare(MyShape \*s1, MyShape \*s2) {

if (s1 == nullptr || s2 == nullptr){

throw MyException("Null pointer exception! (compare)");

}

return s1->getArea() - s2->getArea();

}

bool Shapes::isIntersect(MyShape \*s1, MyShape \*s2) {

if (s1 == nullptr || s2 == nullptr){

throw MyException("Null pointer exception! (isIntersect)");

}

//Моих знаниий и умений в векторной геометрии слишком мало, чтоб реализовать этот метод без ошибок :(

return true;

}

bool Shapes::isInclude(MyShape \*s1, MyShape \*s2) {

if (s1 == nullptr || s2 == nullptr){

throw MyException("Null pointer exception! (isInclude)");

}

//Моих знаниий и умений в векторной геометрии слишком мало, чтоб реализовать этот метод без ошибок :(

return true;

}

## 4.6 MyShape.h

#pragma once

#include "Point.h"

#include "cmath"

#include "MyException.h"

#include "ShapeType.h"

class MyShape {

private:

Point \*\*pointArray = nullptr;

int pointsCount = 0;

Point \*massCenter = nullptr;

public:

MyShape(int pointsCount, Point \*\*pointArray);

virtual ShapeType getType();

virtual double getArea();

virtual void rotate(double alpha);

virtual void move(double x, double y);

~MyShape();

Point \*getPoint(int index);

double getOutRadius();

Point \*getMassCenter();

int getCount() const;

protected:

Point \*calculateMassCenter();

void setPoint(int i, Point \*p);

};

## 4.7 MyShape.cpp

#include "MyShape.h"

ShapeType MyShape::getType() {

return NONE;

}

double MyShape::getArea() {

return 0;

}

void MyShape::rotate(double alpha) {

if (!massCenter) {

massCenter = calculateMassCenter();

}

for (int i = 0; i < pointsCount; i++) {

pointArray[i]->setX(massCenter->getX() + (pointArray[i]->getX() - massCenter->getX()) \* cos(alpha) -

(pointArray[i]->getY() - massCenter->getY()) \* sin(alpha));

pointArray[i]->setY(massCenter->getY() + (pointArray[i]->getY() - massCenter->getY()) \* cos(alpha) -

(pointArray[i]->getX() - massCenter->getX()) \* sin(alpha));

}

}

MyShape::MyShape(int pointsCount, Point \*\*pointArray) {

this->pointArray = pointArray;

this->pointsCount = pointsCount;

}

void MyShape::move(double x, double y) {

for (int i = 0; i < pointsCount; i++) {

pointArray[i]->moveX(x);

pointArray[i]->moveY(y);

}

}

MyShape::~MyShape() {

for(int i = 0; i<pointsCount; i++){

if (pointArray!=nullptr)

delete pointArray[i];

}

if (pointArray!= nullptr)

delete pointArray;

if (massCenter!= nullptr)

delete massCenter;

}

Point \*MyShape::getPoint(int index) {

if (index < 0 || index > 3 || !pointArray[index - 1]) {

throw MyException("Incorrect Index\n");

}

return pointArray[index];

}

double MyShape::getOutRadius() {

return sqrt(pow(pointArray[0]->getX() - pointArray[3]->getX(), 2) +

pow(pointArray[0]->getX() - pointArray[3]->getX(), 2)) / 2;

}

Point \*MyShape::getMassCenter() {

if (massCenter!=nullptr)

delete massCenter;

massCenter = calculateMassCenter();

return massCenter;

}

int MyShape::getCount() const {

return pointsCount;

}

Point \*MyShape::calculateMassCenter() {

return new Point(getPoint(0)->getX() - (getPoint(0)->getX() - getPoint(3)->getX()) / 2,

getPoint(0)->getY() - (getPoint(0)->getY() - getPoint(3)->getY()) / 2);

}

void MyShape::setPoint(int i, Point \*p) {

pointArray[i] = p;

}

## 4.8 Rectangle.h

#pragma once

#include <cmath>

#include "MyShape.h"

#include "ShapeType.h"

class Rectangle : public MyShape

{

public:

Rectangle(Point\* p1, Point\* p2, Point\* p3, Point\* p4);

Rectangle(Point\* p1, Point\* p2, Point\* p3);

double getArea() override;

ShapeType getType() override {

return RECTANGLE;

}

};

## 4.9 Rectangle.cpp

#include "Rectangle.h"

double Rectangle::getArea() {

return sqrt(pow(getPoint(1)->getX() - getPoint(0)->getX(), 2) + pow(getPoint(1)->getY() - getPoint(0)->getY(), 2)) \*

sqrt(pow(getPoint(0)->getX() - getPoint(2)->getX(), 2) + pow(getPoint(0)->getY() - getPoint(2)->getY(), 2));

}

Rectangle::Rectangle(Point \*p1, Point \*p2, Point \*p3, Point \*p4) : MyShape(4, new Point\* [4]{ 0 }) {

setPoint(0, p1);

setPoint(1, p2);

setPoint(2, p3);

setPoint(3, p4);

}

Rectangle::Rectangle(Point \*p1, Point \*p2, Point \*p3) : MyShape(4, new Point\* [4]{ 0 }) {

setPoint(0, p1);

setPoint(1, p2);

setPoint(2, p3);

setPoint(3, new Point(p3->getX() + (p2->getX() - p1->getX()), p3->getY() + (p2->getY() - p1->getY())));

}

## 4.10 Square.h

#pragma once

#include "MyShape.h"

#include "ShapeType.h"

class Square: public MyShape {

public:

Square(Point\* p1, Point\* p2);

double getArea() override;

ShapeType getType() override;

};

## 4.11 Square.cpp

#include "Square.h"

Square::Square(Point \*p1, Point \*p2) : MyShape(4, new Point\* [4]{0 }){

// p1 верхняя левая точка p2 - верхняя правая

setPoint(0, p1);

setPoint(1, p2);

setPoint(2 , new Point(p1->getX() + (p2->getY() - p1->getY()), p1->getY() - (p2->getX() - p1->getX())));

setPoint(3, new Point(p2->getX() + (p2->getY() - p1->getY()), p2->getY() - (p2->getX() - p1->getX())));

}

double Square::getArea() {

return pow(sqrt(pow(getPoint(0)->getX() - getPoint(1)->getX(), 2) + pow(getPoint(0)->getY() - getPoint(1)->getY(), 2)), 2);

}

ShapeType Square::getType() {

return SQUARE;

}

## 4.12 MyException.h

#pragma once

#include <exception>

class MyException : public std::exception {

private:

const char\* reason;

public:

MyException(const char\* reason) {

this->reason = reason;

}

const char\* what() const noexcept override {

return reason;

}

};

## 4.13 ShapeType.h

#pragma once

enum ShapeType{

SQUARE, RECTANGLE, NONE

};

## 4.14 Point.h

#pragma once

class Point

{

private:

double x = 0;

double y = 0;

public:

Point() = default;

Point(double x, double y);

Point& operator = (const Point& old)

= default;

double getX() const;

double getY() const;

void setX(double newX);

void setY(double newY);

void moveX(double l);

void moveY(double l);

};

## 4.15 Point.cpp

#include "Point.h"

Point::Point(double x, double y) {

this->x = x;

this->y = y;

}

void Point::moveY(double l) { this->y += l; }

double Point::getX() const { return x; }

double Point::getY() const { return y; }

void Point::setX(double newX) { this->x = newX; }

void Point::setY(double newY) { this->y = newY; }

void Point::moveX(double l) { this->x += l; }