МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

Направление подготовки: «ПМИ»

**ОТЧЕТ**

по лабораторной работе

**ФИГУРЫ**

**Выполнил:** студент группы

3822Б1ПМ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Р.Д. Махнёв

Подпись

**Проверил:** лаборант ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Черных

Подпись

Нижний Новгород  
2023

**Содержание**

1. [Введение 3](#_Toc135005730)
2. [Постановка задачи 3](#_Toc135005731)
3. [Описание алгоритмов 4](#_Toc135005732)
   1. [Вычисление площади 4](#_Toc135005733)
   2. [Вычисление периметра 4](#_Toc135005734)
4. [Описание программы 5](#_Toc135005735)
   1. [Описание классов 5](#_Toc135005736)
5. [Результаты работы 11](#_Toc135005737)
6. [Заключение 13](#_Toc135005738)
7. [Приложения 15](#_Toc135005739)

# **Введение**

**Актуальность.** Консольная программа, которая работает с фигурами, находит площадь и периметр, является актуальной для решения различных задач. Например, такая программа может быть полезна для архитекторов, инженеров, дизайнеров и других специалистов, которым нужно быстро и точно вычислить характеристики геометрических объектов. Кроме того, такая программа может служить образовательным инструментом для студентов и школьников, изучающих математику и геометрию. Таким образом, консольная программа, которая работает с фигурами, находит площадь и периметр, имеет практическую и теоретическую значимость.

**Цель работы:** освоение техники наследования классов и переопределения методов.

**Постановка задачи**

1. изучить различные источники по данной теме;
2. исследовать наследование, полиморфизм;
3. реализовать программу для работы с фигурами;
4. провести сравнение различных методов сортировки.

**Описание алгоритмов**

Наследование в C++ позволяет создавать классы, которые расширяют функциональность других классов. Наследование упрощает код и избегает повторения. Существуют разные типы наследования, такие как public, protected и private, которые определяют уровень доступа к членам базового класса. [1]

Полиморфизм - это свойство языка C++, позволяющее работать с разными типами данных, имеющими одинаковый интерфейс. Полиморфизм реализуется с помощью виртуальных функций, которые могут быть переопределены в производных классах. Виртуальные функции позволяют вызывать нужную реализацию функции в зависимости от типа объекта, для которого она вызывается. [3]

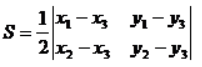
Абстрактный класс в C++ - это класс, в котором объявлена хотя бы одна чисто виртуальная функция. Чисто виртуальная функция - это функция, которая не имеет реализации в абстрактном классе и должна быть переопределена в производном классе. [2]

**Вычисление площади**

Прямая и точка имеют площадь равную 0.

Площадь прямоугольника равна length \* width, length – длина, width - ширина.

Площадь треугольника равна 1/2 модуля определителя. [4]



Площадь круга равна . R – радиус круга.

Площадь квадрата равна width \* width, width - ширина.

**Вычисление периметра**

Точка имеет периметр 0.

Прямая имеет периметр, равный её длине x\*x+y\*y.

Периметр треугольника равен сумме длин его сторон.

Периметр прямоугольника равен 2\*(width + length). Length – длина, width - ширина.

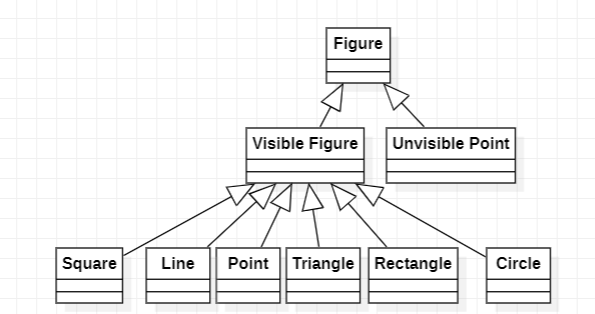
Периметр квадрата равен 4\*width, width - ширина.

Периметр круга равен , r – радиус круга.

**Описание программы**

Данная программа является консольным приложением, она может хранить фигуры, выполнять операции с ними.

**Схема выводимости**



**Описание классов**

**class Figure** – абстрактный базовый класс фигуры с виртуальными методами double perimetr(), double area() и void move(double X, double Y)

double perimetr() – периметр фигуры

double area() – площадь фигуры

void move(double X, double Y) – переместить на (X, Y) (Приложение №3)

**class VisibleFigure** - это производный класс от класса Figure. Он имеет публичное поле bool visible, конструкторы VisibleFigure(bool \_visible) и VisibleFigure(), а также виртуальный метод show(). (Приложение №3)

**class UnvisiblePoint** – производный класс от класса Figure, невидимая точка. (Приложение №3)

**Поля**:

double x, y; координаты x, y

**Конструкторы:**

UnvisiblePoint(double \_x, double \_y) \_x, \_y – координаты новой точки

UnvisiblePoint() – конструктор по умолчанию

UnvisiblePoint(const UnvisiblePoint& point) – конструктор копирования

**Методы:**

double abs() – модуль радиус вектора точки

**Перегрузки:**

const bool operator== (const UnvisiblePoint& point) сравнение двух точек, входные параметры константная ссылка point;

const UnvisiblePoint operator-(const UnvisiblePoint& point) покоординатное вычитание, входные параметры: константная ссылка point.

UnvisiblePoint& operator= (const UnvisiblePoint& point) оператор присваивания. Входные параметры: константная ссылка point.

**class Point** - это производный класс от класса VisibleFigure. Точка, которую можно вывести. Описывает точку. (Приложение №3)

**Поля**:

UnvisiblePoint point; - точка

**Конструкторы**:

Point(const Point& point) – конструктор копирования

Point(bool \_visible) \_visible – видима ли фигура

Point() – конструктор по умолчанию

Point(double \_x, double \_y, bool \_visible) - \_x, \_y – координаты новой точки, \_visible – видима ли фигура

Point(double \_x, double \_y) \_x, \_y – координаты новой точки

**Перегрузки:**

Point& operator= (const Point& point) оператор присваивания. Входные параметры: константная ссылка point.

**class Line** – производный класс от VisibleFigure. Описывает отрезок. (Приложение №3)

**Поля:**

UnvisiblePoint first, second; - точки начала и конца.

**Конструкторы**:

Line(const Line& line) – конструктор копирования

Line(bool \_visible) \_visible – видима ли фигура

Line() – конструктор по умолчанию

Line(const UnvisiblePoint& \_first, const UnvisiblePoint& \_second, bool \_visible) \_first, \_second – точки начала и конца, \_visible – видима ли фигура

Line(UnvisiblePoint \_first, UnvisiblePoint \_second) \_first, \_second – точки начала и конца.

**Перегрузки:**

Line& operator= (const Line& line) оператор присваивания. Входные параметры: константная ссылка line.

**class Triangle** – производный класс от VisibleFigure. Описывает треугольник. (Приложение №3)

**Поля:**

UnvisiblePoint point\_1, point\_2, point\_3; - вершины треугольника.

**Конструкторы:**

Triangle(const UnvisiblePoint& \_point\_1, const UnvisiblePoint& \_point\_2, const UnvisiblePoint& \_point\_3, bool visible = IS\_VISIBLE\_BY\_DEFAULT) - \_point\_1, \_point\_2, \_point\_3 – вершины треугольника, visible - видима ли фигура

Triangle() – конструктор по умолчанию

Triangle(const Triangle& triangle) – конструктор копирования

**Методы**

void setPoint\_1(const UnvisiblePoint& point) – установить 1 точку, входные параметры: константная ссылка point.

void setPoint\_2(const UnvisiblePoint& point) – установить 2 точку, входные параметры: константная ссылка point.

void setPoint\_3(const UnvisiblePoint& point) – установить 3 точку,

Входные параметры: константная ссылка point.

UnvisiblePoint getPoint\_1() – получить 1 точку

UnvisiblePoint getPoint\_2() – получить 2 точку

UnvisiblePoint getPoint\_3() – получить 3 точку

**Перегрузки:**

Triangle & operator= (const Triangle & triangle) оператор присваивания. Входные параметры: константная ссылка triangle.

**class Circle** – производный класс от VisibleFigure. Описывает круг. (Приложение №3)

**Поля:**

double radius; - радиус

UnvisiblePoint center; - центр

**Конструкторы:**

Circle(bool visible = IS\_VISIBLE\_BY\_DEFAULT) visible – видима ли фигура

Circle(double x, double y, double \_radius, bool visible = IS\_VISIBLE\_BY\_DEFAULT) x, y – координаты центра круга, visible – видима ли фигура, \_radius – радиус круга.

Circle(const UnvisiblePoint& point, double \_radius, bool visible = IS\_VISIBLE\_BY\_DEFAULT) point – центральная точка, visible – видима ли фигура, \_radius – радиус круга.

Circle(const Circle& circle) – конструктор копирования

**Методы**

void setRadius(double radius) – установить радиус.

double getRadius() – получить радиус.

**Перегрузки:**

Circle& operator= (const Circle& circle) оператор присваивания. Входные параметры: константная ссылка circle.

**class Rectangle**– производный класс от VisibleFigure. Описывает прямоугольник. (Приложение №3)

**Поля:**

UnvisiblePoint point; - левая верхняя точка

double width; - ширина

double length; - длина

**Конструкторы:**

Rectangle(const UnvisiblePoint& point, double width, double length, bool visible = IS\_VISIBLE\_BY\_DEFAULT) point – левая верхняя вершина, width – ширина, length – длина, visible - видима ли фигура.

Rectangle() – конструктор по умолчанию.

Rectangle(const Rectangle& rectangle) – конструктор копирования.

**Перегрузки:**

Rectangle& operator= (const Rectangle& rectangle) оператор присваивания. Входные параметры: константная ссылка rectangle.

**class Square** – производный класс от VisibleFigure. Описывает квадрат. (Приложение №3)

**Поля:**

UnvisiblePoint point; - левая верхняя точка

double length; - длина

**Конструкторы:**

Square(const UnvisiblePoint& point, double length, bool visible = IS\_VISIBLE\_BY\_DEFAULT) point – левая верхняя вершина, length – длина, visible - видима ли фигура.

Square() – конструктор по умолчанию.

Square(const Square& square) – конструктор копирования.

**Перегрузки:**

Square & operator= (const Square& square) оператор присваивания. Входные параметры: константная ссылка square.

**class Image** – класс, описывающий набор фигур. (Приложение №3)

**Поля:**

size\_t size; - размер

VisibleFigure\*\* image; - массив указателей на фигуры.

**Конструкторы:**

Image(size\_t size) size – размер

Image(const Image& \_shape) - конструктор копирования

**Деструкторы:**

~Image()

**class Exception** – класс, описывающий набор фигур. (Приложение №2)

**Поля:**

std::string name; - имя ошибки

std::string time; - время ошибки

std::string date; - дата ошибки

size\_t line; - линия ошибки

**Конструкторы:**

Exception(std::string \_name, size\_t \_line) \_name – имя ошибка, \_line – линия

**Перегрузки:**

friend std::ostream& operator << (std::ostream& stream, const Exception& exception) exception – исключение

**class LineException** – производный класс от Exception. Исключение прямой (Приложение №2)

**Конструкторы:**

LineException(size\_t line) line – линия на которой произошло исключение.

**class TriangleException**– производный класс от Exception. Исключение прямой (Приложение №2)

**Конструкторы:**

TriangleException (size\_t line) line – линия на которой произошло исключение.

**class CircleException**– производный класс от Exception. Исключение прямой (Приложение №2)

**Конструкторы:**

CircleException (size\_t line) line – линия на которой произошло исключение.

**class RectangleException**– производный класс от Exception. Исключение прямой (Приложение №2)

**Конструкторы:**

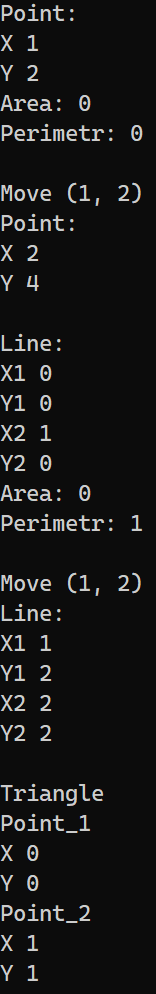
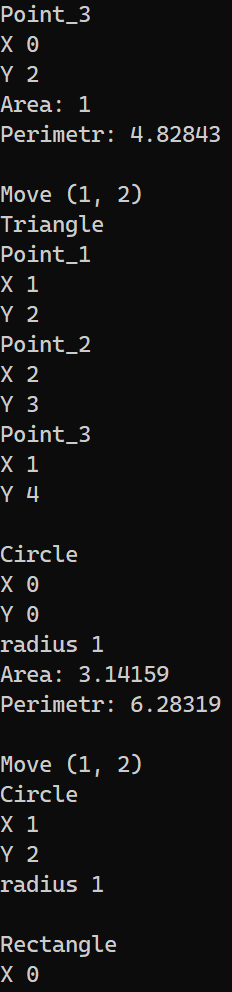
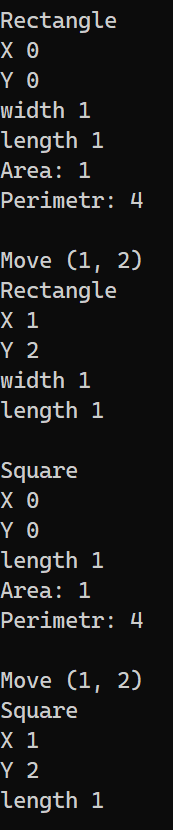
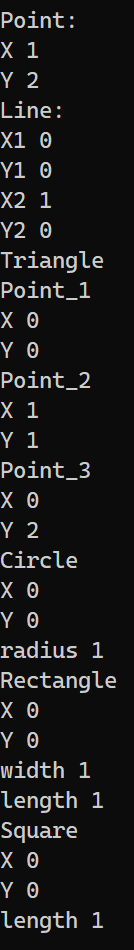
RectangleException (size\_t line) line – линия на которой произошло исключение.

**class ImageException**– производный класс от Exception. Исключение прямой (Приложение №2)

Конструкторы:

**ImageException** (size\_t line) line – линия на которой произошло исключение.

**Результаты работы**

1. создаётся массив указателей на фигур длины 6;
2. добавление в массив указателя на точку с координатами (1, 2);
3. добавление в массив указателя на линию с точками (0, 0) (1, 0);
4. добавление в массив указателя на треугольник с точками (0, 0) (1, 1) (0, 2);
5. добавление в массив указателя на круг с центром (0, 0) и радиусом 1;
6. добавление в массив указателя на прямоугольник с левой верхней точкой (0, 0), шириной 1 и длиной 1;
7. добавление в массив указателя на квадрат с левой верхней точкой (0, 0), шириной 1;
8. отображение фигур, поиск площади, периметра, перемещение на (1, 2), отображение фигур;
9. создание Image размером 6;
10. добавление в image на точку с координатами (1, 2);
11. добавление в image линии с точками (0, 0) (1, 0);
12. добавление в image треугольник с точками (0, 0) (1, 1);
13. добавление в image круг с центром (0, 0) и радиусом 1;
14. добавление в image прямоугольник с левой верхней точкой (0, 0), шириной 1 и длиной 1;
15. добавление в image квадрат с левой верхней точкой (0, 0), шириной 1;
16. отображение фигур. (Приложение №1)

**Заключение**

В этой работе были изучены и разработаны способы работы с фигурами. Изучены и применены техники наследования классов и переопределения методов. По полученным данным можно сделать следующие выводы:

Программа выполняет поставленные задачи.

**Литература**

1. C++ | Наследование – Metanit <https://metanit.com/cpp/tutorial/5.10.php>
2. C++ | Чистые виртуальные функции и абстрактные классы <https://metanit.com/cpp/tutorial/5.12.php>
3. Объектно-ориентированное программирование – Википедия <https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование>
4. Площадь треугольника через координаты - Fxyz.ru <https://www.fxyz.ru/формулы_по_математике/аналитическая_геометрия/на_плоскости/точки_и_прямые_в_прямоугольной_системе_координат/площадь_треугольника_через_координаты/>

**Приложения**

Приложение №1

#include "Figures.hpp"

int main() try {

VisibleFigure\*\* figure = new VisibleFigure\*[6];

figure[0] = new Point(1, 2);

figure[1] = new Line();

figure[2] = new Triangle();

figure[3] = new Circle();

figure[4] = new Rectangle();

figure[5] = new Square();

for (size\_t i = 0; i < 6; i++)

{

figure[i]->show();

std::cout << "Area: " << figure[i]->area() << std::endl;

std::cout << "Perimetr: " << figure[i]->perimetr() << std::endl;

std::cout << std::endl << "Move (1, 2)" << std::endl;

figure[i]->move(1, 2);

figure[i]->show();

std::cout << std::endl;

delete figure[i];

}

delete[] figure;

std::cout << std::endl << std::endl;

Image image(6);

image.push(Point(1, 2), 0);

image.push(Line(), 1);

image.push(Triangle(), 2);

image.push(Circle(), 3);

image.push(Rectangle(), 4);

image.push(Square(), 5);

image.show();

return 0;

}

catch (Exception exception) {

std::cout << exception;

}

Приложение №2

#pragma once

#include <string>

#include <ostream>

class Exception {

std::string name;

std::string time;

std::string date;

size\_t line;

public:

Exception(std::string \_name, size\_t \_line) :name(\_name), line(\_line), date(\_\_DATE\_\_), time(\_\_TIME\_\_) {}

friend std::ostream& operator << (std::ostream& stream, const Exception& exception){

stream << exception.name << std::endl << exception.time << std::endl << exception.date << std::endl << exception.line << std::endl;

return stream;

}

};

class LineException : public Exception {

public:

LineException(size\_t line) : Exception("Line Exception", line) {}

};

class TriangleException : public Exception {

public:

TriangleException(size\_t line) : Exception("Triangle Exception", line) {}

};

class CircleException : public Exception {

public:

CircleException(size\_t line) : Exception("Circle Exception", line) {}

};

class RectangleException : public Exception {

public:

RectangleException(size\_t line) : Exception("Rectangle Exception", line) {}

};

class ShapeException : public Exception {

public:

ShapeException(size\_t line) : Exception("Shape Exception", line) {}

};

Приложение №3

#pragma once

#include <iostream>

#define \_USE\_MATH\_DEFINES

#include <math.h>

#include <vector>

#include "Exception.hpp"

#define IS\_VISIBLE\_BY\_DEFAULT true

//интерфейс

class Figure {

public:

virtual double perimetr() = 0;

virtual double area() = 0;

virtual void move(double X, double Y) = 0;

Figure(){}

};

class VisibleFigure : public Figure {

public:

bool visible;

//конструкторы

VisibleFigure(bool \_visible) : Figure(), visible(\_visible) {}

VisibleFigure() : VisibleFigure(IS\_VISIBLE\_BY\_DEFAULT) {}

//методы

virtual void show() = 0;

};

class UnvisiblePoint : public Figure {

public:

double x, y; //координаты

//конструкторы

UnvisiblePoint(double \_x, double \_y) : Figure(), x(\_x), y(\_y) {}

UnvisiblePoint() : UnvisiblePoint(0, 0) {}

UnvisiblePoint(const UnvisiblePoint& point) : UnvisiblePoint(point.x, point.y) {}

//методы

void move(double X, double Y) override {

x += X;

y += Y;

}

double perimetr() override {

return 0;

}

double area() override {

return 0;

}

double abs() const { //радиус вектор

return sqrt(x \* x + y \* y);

}

//перегрузки

const bool operator== (const UnvisiblePoint& point) const {

return ((x == point.x) && (y == point.y));

}

const UnvisiblePoint operator-(const UnvisiblePoint& point) {

double X = x - point.x;

double Y = y - point.y;

return UnvisiblePoint(X, Y);

}

UnvisiblePoint& operator= (const UnvisiblePoint& point) {

x = point.x;

y = point.y;

return \*this;

}

};

class Point : public VisibleFigure {

public:

UnvisiblePoint point;

Point(const Point& point) : VisibleFigure(point.visible), point(point.point) {}

Point(bool \_visible) : VisibleFigure(\_visible), point() {}

Point() : Point(IS\_VISIBLE\_BY\_DEFAULT) {}

Point(double \_x, double \_y, bool \_visible) : VisibleFigure(\_visible), point(\_x, \_y) {}

Point(double \_x, double \_y) : VisibleFigure(IS\_VISIBLE\_BY\_DEFAULT), point(\_x, \_y) {}

void show() override {

if (visible) {

std::cout << "Point:"<< std::endl

<< "X " << point.x << std::endl

<< "Y " << point.y << std::endl;

}

}

double perimetr() override {

return 0;

}

double area() override {

return 0;

}

void move(double X, double Y) override {

point.x += X;

point.y += Y;

}

Point& operator= (const Point& point) {

this->point = point.point;

this->visible = point.visible;

return \*this;

}

};

class Line : public VisibleFigure {

private:

UnvisiblePoint first, second;

public:

Line(const Line& line) {

first = line.first;

second = line.second;

}

Line(bool \_visible) : VisibleFigure(\_visible), first(), second(1, 0) {}

Line() : Line(IS\_VISIBLE\_BY\_DEFAULT) {}

Line(const UnvisiblePoint& \_first, const UnvisiblePoint& \_second, bool \_visible) : VisibleFigure(\_visible) {

if (\_first == \_second)

throw LineException(\_\_LINE\_\_);

first = \_first;

second = \_second;

}

Line(const UnvisiblePoint& \_first, const UnvisiblePoint& \_second) : Line(\_first, \_second, IS\_VISIBLE\_BY\_DEFAULT) {}

void show() override {

if (visible) {

std::cout<< "Line:"<< std::endl

<< "X1 " << first.x<< std::endl

<< "Y1 " << first.y << std::endl

<< "X2 " << second.x<< std::endl

<< "Y2 " << second.y << std::endl;

}

}

double perimetr() override {

return (second - first).abs();

}

double area() override {

return 0;

}

void move(double X, double Y) override {

first.move(X, Y);

second.move(X, Y);

}

Line& operator= (const Line& line) {

this->first = line.first;

this->second = line.second;

this->visible = line.visible;

return \*this;

}

};

class Triangle : public VisibleFigure {

private:

UnvisiblePoint point\_1, point\_2, point\_3;

public:

//Конструкторы

Triangle(const UnvisiblePoint& \_point\_1, const UnvisiblePoint& \_point\_2, const UnvisiblePoint& \_point\_3, bool visible = IS\_VISIBLE\_BY\_DEFAULT) : VisibleFigure(visible), point\_1(\_point\_1), point\_2(\_point\_2), point\_3(\_point\_3) {

if (area() <= 0)

throw TriangleException(\_\_LINE\_\_);

}

Triangle() : Triangle(UnvisiblePoint(0, 0), UnvisiblePoint(1, 1), UnvisiblePoint(0, 2)) {}

Triangle(const Triangle& triangle) : Triangle(triangle.point\_1, triangle.point\_2, triangle.point\_3, triangle.visible) {}

//методы

void setPoint\_1(const UnvisiblePoint& point) {

point\_1 = point;

if (area() <= 0) throw TriangleException(\_\_LINE\_\_);

}

void setPoint\_2(const UnvisiblePoint& point) {

point\_2 = point;

if (area() <= 0) throw TriangleException(\_\_LINE\_\_);

}

void setPoint\_3(const UnvisiblePoint& point) {

point\_3 = point;

if (area() <= 0) throw TriangleException(\_\_LINE\_\_);

}

UnvisiblePoint getPoint\_1(){ return point\_1; }

UnvisiblePoint getPoint\_2() { return point\_2; }

UnvisiblePoint getPoint\_3() { return point\_3; }

void move(double X, double Y) override {

point\_1.move(X, Y);

point\_2.move(X, Y);

point\_3.move(X, Y);

}

double perimetr() override {

return (point\_2 - point\_1).abs() + (point\_3 - point\_2).abs() + (point\_3 - point\_1).abs();

}

double area() override {

//double a = (point\_2 - point\_1).abs();

//double b = (point\_3 - point\_2).abs();

//double c = (point\_3 - point\_1).abs();

//double p = (a + b + c) / 2;

//return sqrt(p \* (p - a) \* (p - b) \* (p - c));

return abs(0.5\*((point\_1.x - point\_3.x) \* (point\_2.y - point\_3.y) - (point\_1.y - point\_3.y) \* (point\_2.x - point\_3.x)));

}

void show() override {

if (visible) {

std::cout << "Triangle" << std::endl

<< "Point\_1 " << std::endl

<< "X " << point\_1.x << std::endl

<< "Y " << point\_1.y << std::endl

<< "Point\_2 " << std::endl

<< "X " << point\_2.x << std::endl

<< "Y " << point\_2.y << std::endl

<< "Point\_3 " << std::endl

<< "X " << point\_3.x << std::endl

<< "Y " << point\_3.y << std::endl;

}

}

Triangle& operator= (const Triangle& triangle) {

this->point\_1 = triangle.point\_1;

this->point\_2 = triangle.point\_2;

this->point\_3 = triangle.point\_3;

this->visible = triangle.visible;

return \*this;

}

};

class Circle : public VisibleFigure {

private:

double radius;

public:

UnvisiblePoint center;

//конструкторы

Circle(bool visible = IS\_VISIBLE\_BY\_DEFAULT) : VisibleFigure(visible), center(), radius(1.0) {}

Circle(double x, double y, double \_radius, bool visible = IS\_VISIBLE\_BY\_DEFAULT) : VisibleFigure(visible) {

if (\_radius <= 0) throw CircleException(\_\_LINE\_\_);

radius = \_radius;

}

Circle(const UnvisiblePoint& point, double \_radius, bool visible = IS\_VISIBLE\_BY\_DEFAULT) : VisibleFigure(visible) {

if (\_radius <= 0) throw CircleException(\_\_LINE\_\_);

radius = \_radius;

}

Circle(const Circle& circle) : Circle(circle.center, circle.radius, circle.visible) {}

//методы

void setRadius(double radius) {

if (radius <= 0) throw CircleException(\_\_LINE\_\_);

this->radius = radius;

}

double getRadius() { return radius; }

double perimetr() override {

return 2 \* M\_PI \* radius;

}

double area() override {

return M\_PI \* radius \* radius;

}

void show() override {

if (visible) {

std::cout << "Circle" << std::endl

<< "X " << center.x << std::endl

<< "Y " << center.y << std::endl

<< "radius " << radius << std::endl;

}

}

void move(double X, double Y) override {

center.move(X, Y);

}

Circle& operator= (const Circle& circle) {

this->center = circle.center;

this->radius = circle.radius;

this->visible = circle.visible;

return \*this;

}

};

class Rectangle : public VisibleFigure {

public:

UnvisiblePoint point;

double width;

double length;

//конструкторы

Rectangle(const UnvisiblePoint& point, double width, double length, bool visible = IS\_VISIBLE\_BY\_DEFAULT) : VisibleFigure(visible) {

if (width <= 0) throw RectangleException(\_\_LINE\_\_);

if (length <= 0) throw RectangleException(\_\_LINE\_\_);

this->width = width;

this->length = length;

}

Rectangle() : Rectangle(UnvisiblePoint(), 1, 1) {}

Rectangle(const Rectangle& rectangle) : Rectangle(rectangle.point, rectangle.width, rectangle.length, rectangle.visible) {}

double perimetr() override {

return 2 \* (width + length);

}

double area() override {

return width \* length;

}

void show() override {

if (visible) {

std::cout << "Rectangle" << std::endl

<< "X " << point.x << std::endl

<< "Y " << point.y << std::endl

<< "width " << width << std::endl

<< "length " << length << std::endl;

}

}

void move(double X, double Y) override {

point.move(X, Y);

}

Rectangle& operator= (const Rectangle& rectangle) {

this->point = rectangle.point;

this->width = rectangle.width;

this->length = rectangle.length;

this->visible = rectangle.visible;

return \*this;

}

};

class Square : public VisibleFigure {

public:

UnvisiblePoint point;

double length;

//конструкторы

Square(const UnvisiblePoint& point, double length, bool visible = IS\_VISIBLE\_BY\_DEFAULT) : VisibleFigure(visible) {

if (length <= 0) throw RectangleException(\_\_LINE\_\_);

this->length = length;

}

Square() : Square(UnvisiblePoint(), 1) {}

Square(const Square& square) : Square(square.point, square.length, square.visible) {}

double perimetr() override {

return 4 \* length;

}

double area() override {

return length \* length;

}

void show() override {

if (visible) {

std::cout << "Square" << std::endl

<< "X " << point.x << std::endl

<< "Y " << point.y << std::endl

<< "length " << length << std::endl;

}

}

void move(double X, double Y) override {

point.move(X, Y);

}

Square& operator= (const Square& square) {

this->point = square.point;

this->length = square.length;

this->visible = square.visible;

return \*this;

}

};

class Image {

private:

size\_t size;

VisibleFigure\*\* image;

public:

template <class T>

void push(const T& figure, size\_t index) {

if (index >= size) throw ShapeException(\_\_LINE\_\_);

if (image[index] != nullptr)delete image[index];

image[index] = new T(figure);

}

void show() {

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

{

image[i]->show();

}

}

Image(size\_t size) {

this->size = size;

image = new VisibleFigure \* [size];

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

{

image[i] = nullptr;

}

}

Image(const Image& \_shape) {

this->image = new VisibleFigure \* [\_shape.size];

size = \_shape.size;

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

{

//push((\*\_shape.shape[i]), i);

}

}

~Image() {

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

{

delete image[i];

}

delete[] image;

}

};