Московский Авиационный Институт
(Национальный Исследовательский Университет)

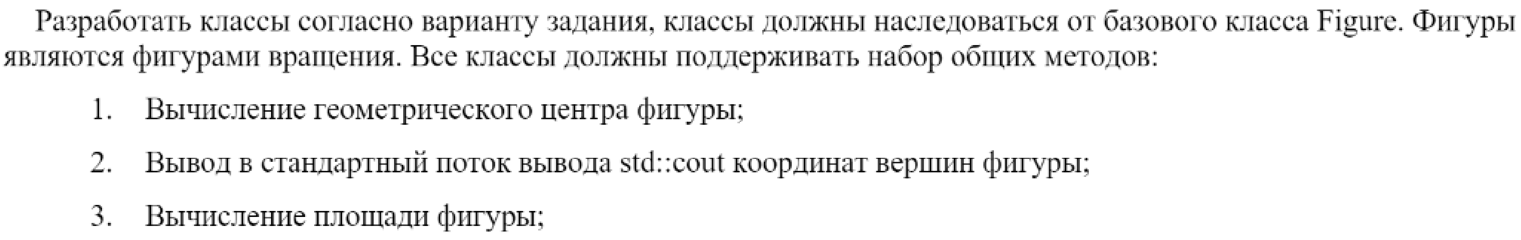
Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 3: «Наследование. полиморфизм»

Группа:	М8О-206Б-18, №27
Студент:	Шорохов Алексей Павлович
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	

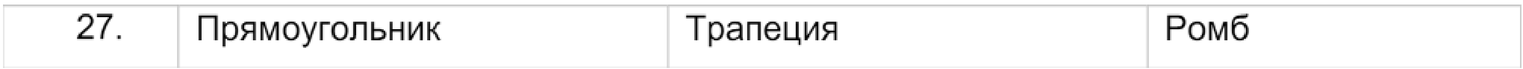
Москва, 2019

Задание



Разработать классы согласно варианту задания, классы должны наследоваться от базового класса Figure. Фигуры являются фигурами вращения. Все классы должны поддерживать набор общих методов:

1. Вычисление геометрического центра фигуры;
2. Вывод в стандартный поток вывода std::cout координат вершин фигуры;
3. Вычисление площади фигуры;



27.	Прямоугольник	Трапеция	Ромб

Адрес репозитория на GitHub

Код программы на С++

```
cmake_minimum_required(VERSION 3.2)

project(BitString)

add_executable(lab3
        Source.cpp
        point.cpp
        figure.cpp
        rectangle.cpp
        rhombus.cpp
        trapezoid.cpp
        vector.cpp)

set_property(TARGET BitString PROPERTY CXX_STANDART 11)
```

point.h

```
#ifndef D_POINT_H_
#define D_POINT_H_

#include <iostream>

struct Point {

        double x, y;

};

std::istream& operator>> (std::istream& is, Point &p);
std::ostream& operator<< (std::ostream& os, const Point &p);
```

```
bool operator == (Point a, Point b);
```

Point.cpp

```
#include "point.h"

std::istream& operator >> (std::istream& is, Point &p) {
        is >> p.x >> p.y;
}

std::ostream& operator << (std::ostream& os, const Point &p) {
   return os << p.x << " " << p.y << '\n';
}

bool operator == (Point a, Point b) {
        return (a.x == b.x && a.y == b.y);
}
```

Figure.h

```
#ifndef D_FIGURE_H_
#define D_FIGURE_H_

#include <iostream>


#include "point.h"
#include "vector.h"


class Figure {
public:
        virtual Point center() const = 0;
        virtual void print(std::ostream&) const = 0;
        virtual double square() const = 0;
        virtual ~Figure() = default;
};

std::ostream& operator << (std::ostream& os, const Figure& f);

e
```

Figure.cpp

```
#include "figure.h"
i
std::ostream& operator << (std::ostream& os, const Figure& f) {
D
F
I
G
U
R
```

Vector.h

```
#ifndef VECTOR_H_
#define VECTOR_H_

#include "point.h"
#include <cmath>
#include <numeric>
#include <limits>

class Vector {
public:
	explicit Vector(Point a, Point b);
	double length() const;
	double x;
	double y;
	friend double operator* (Vector a, Vector b) ;
	bool operator== (Vector b);
};

bool isParallel(const Vector a, const Vector b);
bool isPerpendicular(const Vector a, const Vector b);



#endif
```

Vector.cpp

```
#include "vector.h"

Vector::Vector(Point a, Point b) {
	x = b.x - a.x;
	y = b.y - a.y;
}

double Vector::length() const{
	return sqrt(x * x + y * y);
}

bool isParallel(const Vector a, const Vector b) {
	return (a.x * b.y - a.y * b.y) == 0;
}

bool isPerpendicular(const Vector a, const Vector b) {
	return (a.x * b.x + a.y * b.y) == 0;
}
```

```
double operator* (Vector a, Vector b) {
        return a.x * b.x + a.y * b.y;
}

bool Vector::operator== (Vector b) {
        return std::abs(x - b.x) < std::numeric_limits<double>::epsilon() * 100
        && std::abs(y - b.y) < std::numeric_limits<double>::epsilon() * 100;
}
```

Rectangle.h

```
#ifndef D_RECTANGLE_H_
#define D_RECTANGLE_H_

#include "figure.h"

class Rectangle : public Figure {
public:
        Rectangle (std::istream&);

        Point center() const override;
        void print(std::ostream&) const override;
        double square() const override;
private:

P
o
i
nendif
tRectangle.cpp
p#include "rectangle.h"
p#include <iostream>
p#include <cmath>
p

Rectangle::Rectangle(std::istream& is) {
        is >> p1 >> p2 >> p3 >> p4;

        if (isPerpendicular(Vector(p1, p2), Vector(p1, p4)) &&
isPerpendicular(Vector(p1, p2), Vector(p2, p3)) &&
                isPerpendicular(Vector(p2, p3), Vector(p3, p4)) &&
isPerpendicular(Vector(p3, p4), Vector(p1, p4))) {
    } else if (isPerpendicular(Vector(p1, p4), Vector(p4, p2)) &&
isPerpendicular(Vector(p4, p2), Vector(p2, p3)) &&
        isPerpendicular(Vector(p2, p3), Vector(p3, p1)) &&
isPerpendicular(Vector(p1, p3), Vector(p1, p4))) {

            Point tmp;
            tmp = p1;
            p1 = p4;
            p4 = tmp;

    } else if (isPerpendicular(Vector(p1, p2), Vector(p2, p4)) &&
isPerpendicular(Vector(p2, p4), Vector(p4, p3)) &&
        isPerpendicular(Vector(p4, p3), Vector(p3, p1)) &&
isPerpendicular(Vector(p3, p1), Vector(p1, p2))) {

            Point tmp;
            tmp = p3;
            p3 = p4;
            p4 = tmp;

    } else if (p1 == p2 || p1 == p3 || p1 == p4 || p2 == p3 || p2 == p4 || p3 == p4)
{
        throw std::logic_error("No points are able to be equal");
    } else {
        throw std::logic_error("That's not a Rectangle, sides are not
Perpendicular");
    }

    if (!(Vector(p1, p2).length() == Vector(p3, p4).length() && Vector(p2,
p3).length() == Vector(p1, p4).length())) {
        throw std::logic_error("That's not a Rectangle, sides are not equal");


}

Point Rectangle::center() const{
    Point p;
    p.x = (p1.x + p2.x + p3.x + p4.x) / 4;
    p.y = (p1.y + p2.y + p3.y + p4.y) / 4;
    return p;
}

void Rectangle::print(std::ostream& os) const{
    os << "Rectangle\n";
    os << p1 << p2 << p3 << p4;
```

```
}

double Rectangle::square() const{

        return Vector(p1, p2).length() * Vector(p2, p3).length();
```

Rhombus.h

```
#ifndef D_RHOMBUS_H_
#define D_RHOMBUS_H_

#include "figure.h"

class Rhombus : public Figure {
public:
        Rhombus (std::istream&);

        Point center() const override;
        void print(std::ostream&) const override;
        double square() const override;
private:

P
o
i
n
endif
t
```

Rhombus.cpp

```
p
#include "rhombus.h"
p
#include <iostream>
p
#include <cmath>
p

Rhombus::Rhombus(std::istream& is) {
        is >> p1 >> p2 >> p3 >> p4;

        if (Vector(p1, p2).length() == Vector(p2, p3).length() && Vector(p2,
p3).length() == Vector(p3, p4).length()
         && Vector(p1, p2).length() == Vector(p1, p4).length()) {

        } else if (Vector(p1, p2).length() == Vector(p2, p4).length() && Vector(p2,
p4).length() == Vector(p3, p4).length()
         && Vector(p1, p2).length() == Vector(p1, p3).length()) {
                Point tmp = p4;
                p4 = p3;
        p3 = tmp;
    } else if (Vector(p1, p3).length() == Vector(p4, p3).length() && Vector(p4,
p3).length() == Vector(p2, p4).length()
     && Vector(p1, p2).length() == Vector(p1, p3).length()) {
        Point tmp = p4;
        p4 = p3;
        p3 = tmp;
    } else if (p1 == p2 || p1 == p3 || p1 == p4 || p2 == p3 || p2 == p4 || p3 == p4)
{
        throw std::logic_error("No points are able to be equal");
    } else {
        throw std::logic_error("This is not a Rhombus, sides are not equal");
    }


    Vector v1(p1, p2);
    Vector v2(p2, p3);
    Vector v3(p3, p4);
    Vector v4(p4, p1);

    double cos1 = v1 * v2 / (v1.length() * v2.length());
    double cos2 = v2 * v3 / (v2.length() * v3.length());
    double cos3 = v3 * v4 / (v3.length() * v4.length());
    double cos4 = v1 * v4 / (v1.length() * v4.length());

    if (cos1 != cos3 || cos2 != cos4) {
        throw std::logic_error("This is not a Rhombus, opposite angles are not
equal");
    }
}

Point Rhombus::center() const{
    Point p;
    p.x = (p1.x + p2.x + p3.x + p4.x) / 4;
    p.y = (p1.y + p2.y + p3.y + p4.y) / 4;



void Rhombus::print(std::ostream& os) const{

    os << p1 << p2 << p3 << p4;
}

double Rhombus::square() const{
    return Vector(p1, p3).length() * Vector(p2, p4).length() / 2;
```

Trapezoid.h

```
#ifndef D_TRAPEZOID_H_
#define D_TRAPEZOID_H_

#include "figure.h"

class Trapezoid : public Figure {
public:
	Trapezoid (std::istream&);

	Point center() const override;
	void print(std::ostream&) const override;
	double square() const override;
private:

P
o
i
endif
```

Trapezoid.cpp

```
#include "trapezoid.h"
#include <iostream>
#include <cmath>
p
Trapezoid::Trapezoid(std::istream& is) {
	is >> p1 >> p2 >> p3 >> p4;

	if (isParallel(Vector(p1, p4), Vector(p2, p3))) {

	} else if (isParallel(Vector(p1, p3), Vector(p4, p2))) {

		Point tmp;
		tmp = p2;
		p2 = p4;
		p4 = tmp;
		tmp = p3;
		p3 = p4;
		p4 = tmp;

	} else if (isParallel(Vector(p1, p3), Vector(p2, p4))) {

		Point tmp;
		tmp = p3;
		p3 = p4;
```

```
			throw std::logic_error("No points are able to be equal");
		} else {
			throw std::logic_error("At least 2 sides of trapeze must be parallel");
		}
}

Point Trapezoid::center() const{
	Point p;
	p.x = (p1.x + p2.x + p3.x + p4.x) / 4;
	p.y = (p1.y + p2.y + p3.y + p4.y) / 4;
	return p;
}

void Trapezoid::print(std::ostream& os) const{
	os << "Trapezoid\n";
	os << p1 << p2 << p3 << p4;
}

double Trapezoid::square() const{

	double a = p2.y - p3.y;
    double b = p3.x - p2.x;
    double c = p2.x * p3.y - p3.x * p2.y;
    double height = (std::abs(a * p1.x + b * p1.y + c) / sqrt(a * a + b * b));
    return (Vector(p1, p2).length() + Vector(p3, p4).length()) * height / 2;
```

Source.cpp

```
#include <iostream>
#include <cmath>
#include <vector>
#include "vector.h"
#include "point.h"
#include "rectangle.h"
#include "rhombus.h"
#include "trapezoid.h"
#include "figure.h"

void menu() {
	std::cout << "MENU\n0 : exit\n1 : input new figure\n	0 : Rectangle\n	1
: Rhombus\n	2 : Trapezoid\n2 : functions\n	0 : print figures\n	1 : print
squares\n	2 : print centers\n	3 : print sum of all squares\n3 : delete figure by
id\n";
```

```
}

int main() {

    std::vector<Figure *> figures;

    for (;;) {

        menu();
        int cmd;
        std::cin >> cmd;
        if (cmd == 0) {

            break;

        } else if (cmd == 1) {

            int figureId;
            std::cin >> figureId;
            Figure * newFigure;

            if (figureId == 0) {
                newFigure = new Rectangle(std::cin);
            } else if (figureId == 1) {
                newFigure = new Rhombus(std::cin);
            } else if (figureId == 2) {
                newFigure = new Trapezoid(std::cin);
            }

            figures.push_back(newFigure);
        } else if (cmd == 2) {

            int functionId;
            std::cin >> functionId;

            if (functionId == 0) {
                for (Figure * currentFigure : figures) {
                    currentFigure -> print(std::cout);
                }
            } else if (functionId == 1) {
                for (Figure * currentFigure : figures) {
                    std::cout << currentFigure -> square() << '\n';
                }
            } else if (functionId == 2) {
                for (Figure * currentFigure : figures) {
                    std::cout << currentFigure -> center();
                }
            } else if (functionId == 3) {
                double sum = 0;
                for (Figure * currentFigure : figures) {
                    sum += currentFigure -> square();
                }
                std::cout << sum << '\n';
            }

        } else if (cmd == 3) {

            int id;
            std::cin >> id;
            if (id >= 0 && id < figures.size()) {

                delete figures[id];
                figures.erase(figures.begin() + id);

            } else {
                throw std::logic_error("id is out of range");
            }

        }

    }

    for (size_t i = 0; i < figures.size(); i++) {
        delete figures[i];
    }

    return 0;
}
```

File01.test

```
1 2
0 0
0 20
15 20
21 0
2 1
2 2
3
0
2 0
```

```
0
```

File02.test

```
1
0
1 1
5 20
5 1
1 20
1 1
0 0
-1 2
1 2
0 4
2 0
2 1
2 2
2 3
3 0
0
```

Результаты тестов

```
:
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
1 2
0 0
0 20
15 20
21 0
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
```

```
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
2
1
408.806
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
2 2
9 10
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
3
0
MENU
0 : exit
1 : input new figure
        0 : Rectangle


2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
2 0
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
```

:

```
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
1
0
1 1
5 20
5 1
1 20
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
1 1
0 0
-1 2
1 2
0 4
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
2 0
Rectangle
1 20
5 20
5 1
1 1
Rhombus
0 0
-1 2
0 4
1 2
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
2 1
```

```
76
4
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
2 2
3 10.5
0 2
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
2 3
80
MENU
0 : exit
1 : input new figure
        0 : Rectangle
        1 : Rhombus
        2 : Trapezoid
2 : functions
        0 : print figures
        1 : print squares
        2 : print centers
        3 : print sum of all squares
3 : delete figure by id
```

```
	0 : Rectangle
	1 : Rhombus
	2 : Trapezoid
2 : functions
	0 : print figures
	1 : print squares
	2 : print centers
	3 : print sum of all squares
```

Объяснение результатов

Программа получает на вход команды из меню. В зависимости от команды совершается одно из действий: ввод фигуры, нахождение площади, центра, печать координат, нахождение суммы всех площадей, удаление фигуры из вектора по id

Вывод

Были изучены наследование и полиморфизм, применены в лабораторной работе. Применение наследования уменьшает количество программного кода, делает похожие классы типовыми и упрощает понимание кода. Полиморфизм позволяет объектам с одинаковой спецификацией иметь различную спецификацию.