Московский авиационный институт
(Национальный исследовательский университет)
Факультет "Информационные технологии и прикладная математика"

Лабораторная работа №3 по курсу “Объектно-ориентированное программирование”

Студент: Живалев Е.А.

Группа: М8О-206Б

Преподаватель: Журавлев А.А.

Вариант: 5

Оценка: _____

Дата: _____

Москва
2019

1 Исходный код

Ссылка на github : https://github.com/QElderDelta/oop_exercise_03

Figures.hpp

```
#pragma once

#include <iostream>
#include <vector>

struct Point {
    double x, y;
};

double calculateDistance(const Point& lhs, const Point& rhs);
bool operator<(const Point& lhs, const Point& rhs);
std::istream& operator>>(std::istream& is, Point& p);
std::ostream& operator<<(std::ostream& os, const Point& p);
std::ostream& operator<<(std::ostream& os, const std::vector<Point
    > v);

class Figure {
public:
    virtual Point Center() const = 0;
    virtual double Square() const = 0;
    virtual void Print(std::ostream& os) const = 0;
    virtual ~Figure() = default;
};

class Rhombus : public Figure {
public:
    Rhombus(const Point& p1, const Point& p2, const Point& p3,
    const Point& p4);
    Point Center() const override;
    double Square() const override;
    void Print(std::ostream& os) const override;
private:
    std::vector<Point> points;
    double smallerDiagonal, biggerDiagonal;
};

class Pentagon : public Figure {
public:
    Pentagon(const Point& p1, const Point& p2, const Point& p3,
    const Point& p4, const Point& p5);
    Point Center() const override;
    double Square() const override;
    void Print(std::ostream& os) const override;
private:
    std::vector<Point> points;
};

class Hexagon : public Figure {
public:
    Hexagon(const Point& p1, const Point& p2, const Point& p3,
    const Point& p4, const Point& p5, const Point& p6);
    Point Center() const override;
    double Square() const override;
```

```
    void Print(std::ostream& os) const override;
private:
    std::vector<Point> points;
};
```

Figures.cpp

```
#include "Figures.hpp"
#include <cmath>
#include <algorithm>
#include <iomanip>

double calculateDistance(const Point& lhs, const Point& rhs) {
    return sqrt(pow(rhs.x - lhs.x, 2) + pow(rhs.y - lhs.y, 2));
}

bool operator<(const Point& lhs, const Point& rhs) {
    if(lhs.x != rhs.x) {
        return lhs.x < rhs.x;
    }
    return lhs.y < rhs.y;
}

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

std::ostream& operator<<(std::ostream& os, const Point& p) {
    os << std::fixed << std::setprecision(3) << "[" << p.x << ", "
     << p.y << "]";
    return os;
}

std::ostream& operator<<(std::ostream& os, const std::vector
        <Point> v) {
    for(unsigned i = 0; i < v.size(); ++i) {
        os << v[i];
        if(i != v.size() - 1) {
            os << ", ";
        }
    }
    return os;
}

double checkIfRhombus(const Point& p1, const Point& p2, const
        Point& p3, const Point& p4) {
    double d1 = calculateDistance(p1, p2);
    double d2 = calculateDistance(p1, p3);
    double d3 = calculateDistance(p1, p4);
    if(d1 == d2) {
        return d3;
    } else if(d1 == d3) {
        return d2;
    } else if(d2 == d3) {
        return d1;
    } else {
        throw std::invalid_argument("Entered coordinates are not
    forming Rhombus. Try entering new coordinates");
```

```
    }
}

Rhombus::Rhombus(const Point& p1, const Point& p2, const Point& p3
    , const Point& p4) {
    try {
        double d1 = checkIfRhombus(p1, p2, p3, p4);
        double d2 = checkIfRhombus(p2, p1, p3, p4);
        double d3 = checkIfRhombus(p3, p1, p2, p4);
        double d4 = checkIfRhombus(p4, p1, p2, p3);
        if(d1 == d2 || d1 == d4) {
            if(d1 < d3) {
                smallerDiagonal = d1;
                biggerDiagonal = d3;

            } else {
                smallerDiagonal = d3;
                biggerDiagonal = d1;
            }
        } else if(d1 == d3) {
            if(d1 < d2) {
                smallerDiagonal = d1;
                biggerDiagonal = d2;
            } else {
                smallerDiagonal = d2;
                biggerDiagonal = d1;
            }
        }
    } catch(std::exception& e) {
        throw std::invalid_argument(e.what());
        return;
    }
    points.push_back(p1);
    points.push_back(p2);
    points.push_back(p3);
    points.push_back(p4);
}

Point Rhombus::Center() const {
    if(calculateDistance(points[0], points[1]) == smallerDiagonal
    ||
            calculateDistance(points[0], points[1]) ==
    biggerDiagonal) {
        return {((points[0].x + points[1].x) / 2.0), ((points[0].y
     + points[1].y) / 2.0)};
    } else if(calculateDistance(points[0], points[2]) ==
    smallerDiagonal ||
            calculateDistance(points[0], points[2]) ==
    biggerDiagonal) {
        return {((points[0].x + points[2].x) / 2.0), ((points[0].y
     + points[2].y) / 2.0)};
    } else {
        return {((points[0].x + points[3].x) / 2.0), ((points[0].y
     + points[3].y) / 2.0)};
    }
}

double Rhombus::Square() const {
    return smallerDiagonal * biggerDiagonal / 2.0;
```

```
102 }
103
104 void Rhombus::Print(std::ostream& os) const {
105     if(points.size()) {
106         os << "Rhombus: " << points << std::endl;
107     }
108 }
109
110 Pentagon::Pentagon(const Point& p1, const Point& p2, const Point&
    p3,
111         const Point& p4, const Point& p5) {
112     points.push_back(p1);
113     points.push_back(p2);
114     points.push_back(p3);
115     points.push_back(p4);
116     points.push_back(p5);
117 }
118
119 double triangleSquare(const Point& p1, const Point& p2, const
    Point& p3) {
120     return 0.5 * fabs((p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x
    ) * (p1.y - p3.y));
121 }
122
123 Point Pentagon::Center() const {
124     Point insideFigure{0, 0};
125     Point result{0, 0};
126     double square = this->Square();
127     for(unsigned i = 0; i < points.size(); ++i) {
128         insideFigure.x += points[i].x;
129         insideFigure.y += points[i].y;
130     }
131     insideFigure.x /= points.size();
132     insideFigure.y /= points.size();
133     for(unsigned i = 0; i < points.size(); ++i) {
134         double tempSquare = triangleSquare(points[i], points[(i +
    1) % points.size()],
135                 insideFigure);
136         result.x += tempSquare * (points[i].x + points[(i + 1) %
    points.size()].x
137                 + insideFigure.x) / 3.0;
138         result.y += tempSquare * (points[i].y + points[(i + 1) %
    points.size()].y
139                 + insideFigure.y) / 3.0;
140     }
141     result.x /= square;
142     result.y /= square;
143     return result;
144 }
145
146 double Pentagon::Square() const {
147     double result = 0;
148     for(unsigned i = 0; i < points.size(); ++i) {
149         Point p1 = i ? points[i - 1] : points[points.size() - 1];
150         Point p2 = points[i];
151         result += (p1.x - p2.x) * (p1.y + p2.y);
152     }
153     return fabs(result) / 2.0;
154 }
```

```
155
156 void Pentagon::Print(std::ostream& os) const {
157     os << "Pentagon: " << points << std::endl;
158 }
159
160 Hexagon::Hexagon(const Point& p1, const Point& p2, const Point& p3
    ,
161         const Point& p4, const Point& p5, const Point& p6) {
162     points.push_back(p1);
163     points.push_back(p2);
164     points.push_back(p3);
165     points.push_back(p4);
166     points.push_back(p5);
167     points.push_back(p6);
168 }
169
170 Point Hexagon::Center() const {
171     Point insideFigure{0, 0};
172     Point result{0, 0};
173     double square = this->Square();
174     for(unsigned i = 0; i < points.size(); ++i) {
175         insideFigure.x += points[i].x;
176         insideFigure.y += points[i].y;
177     }
178     insideFigure.x /= points.size();
179     insideFigure.y /= points.size();
180     for(unsigned i = 0; i < points.size(); ++i) {
181         double tempSquare = triangleSquare(points[i], points[(i +
    1) % points.size()],
182                 insideFigure);
183         result.x += tempSquare * (points[i].x + points[(i + 1) %
    points.size()].x
184                 + insideFigure.x) / 3.0;
185         result.y += tempSquare * (points[i].y + points[(i + 1) %
    points.size()].y
186                 + insideFigure.y) / 3.0;
187     }
188     result.x /= square;
189     result.y /= square;
190     return result;
191 }
192
193 double Hexagon::Square() const {
194     double result = 0;
195     for(unsigned i = 0; i < points.size(); ++i) {
196         Point p1 = i ? points[i - 1] : points[points.size() - 1];
197         Point p2 = points[i];
198         result += (p1.x - p2.x) * (p1.y + p2.y);
199     }
200     return fabs(result) / 2.0;
201 }
202
203 void Hexagon::Print(std::ostream& os) const {
204     os << "Hexagon: " << points << std::endl;
205 }
```

main.cpp

```
1 #include <iostream>
```

```cpp
#include "Figures.hpp"

int get_command() {
    int command;
    std::cin >> command;
    return command;
}

int main() {
    int command1, command2;
    std::vector<Figure*> figures;
    std::cout << "1 - add figure to the vector\n"
                 "2 - delete figure from the vector\n"
                 "3 - call common functions for the whole vector\n"
                 "4 - get total area of figures in vector\n"
                 "0 - exit\n";
    while((command1 = get_command()) != 0) {
        if(command1 == 1) {
            Figure* f;
            std::cout << "1 - Rhombus, 2 - Pentagon, 3 - Hexagon" << std::endl;
            std::cin >> command2;
            if(command2 == 1) {
                Point p1, p2, p3, p4;
                std::cin >> p1 >> p2 >> p3 >> p4;
                try {
                    f = new Rhombus{p1, p2, p3, p4};
                    figures.push_back(f);
                } catch(std::exception& e) {
                    std::cerr << e.what() << std::endl;
                }
            } else if(command2 == 2) {
                Point p1, p2, p3, p4, p5;
                std::cin >> p1 >> p2 >> p3 >> p4 >> p5;
                f = new Pentagon{p1, p2, p3, p4, p5};
                figures.push_back(f);
            } else if(command2 == 3) {
                Point p1, p2, p3, p4, p5, p6;
                std::cin >> p1 >> p2 >> p3 >> p4 >> p5 >> p6;
                f = new Hexagon{p1, p2, p3, p4, p5, p6};
                figures.push_back(f);
            } else {
                std::cout << "Wrong input" << std::endl;
            }
        } else if(command1 == 2) {
            std::cout << "Enter index" << std::endl;
            std::cin >> command2;
            if(command2 < static_cast<int>(figures.size())) {
                delete figures[command2];
                figures.erase(figures.begin() + command2);
            } else {
                std::cout << "Element with such index doesn't exist" << std::endl;
            }
        } else if(command1 == 3) {
            for(const auto& figure : figures) {
                figure->Print(std::cout);
                std::cout << figure->Center() << std::endl;
```

```
                std::cout << figure->Square() << std::endl;
            }
        } else if(command1 == 4) {
            double result = 0;
            for(const auto& figure : figures) {
                result += figure->Square();
            }
            std::cout << result << std::endl;
        } else if(command1 == 5) {
            break;
        } else {
            std::cout << "Wrong command" << std::endl;
        }
    }
    return 0;
}
```

test.cpp

```
#include "Figures.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE testFigures

#include <boost/test/unit_test.hpp>

BOOST_AUTO_TEST_CASE(testRhombusSqaure) {
    Point p1{-2, 0};
    Point p2{0, 2};
    Point p3{2, 0};
    Point p4{0, -2};
    Rhombus r{p1,p2,p3,p4};
    BOOST_CHECK_EQUAL(r.Square(), 8);
    p2.y = 1;
    p4.y = -1;
    Rhombus r1{p1,p2,p3,p4};
    BOOST_CHECK_EQUAL(r1.Square(), 4);
}

BOOST_AUTO_TEST_CASE(testRhombusCenter) {
    Point p1{-2, 0};
    Point p2{0, 2};
    Point p3{2, 0};
    Point p4{0, -2};
    Rhombus r{p1,p2,p3,p4};
    BOOST_CHECK_EQUAL(r.Center().x, 0);
    BOOST_CHECK_EQUAL(r.Center().y, 0);
    p2.y = 1;
    p4.y = -1;
    Rhombus r1{p1,p2,p3,p4};
    BOOST_CHECK_EQUAL(r1.Center().x, 0);
    BOOST_CHECK_EQUAL(r1.Center().y, 0);
}

BOOST_AUTO_TEST_CASE(testPentagonSquare) {
    Point p1{-2, 0};
    Point p2{0, 2};
    Point p3{2, 0};
    Point p4{3, 0};
```

```
    Point p5{0, -2};
    Pentagon p{p1, p2, p3, p4, p5};
    BOOST_CHECK_EQUAL(p.Square(), 9);
}

BOOST_AUTO_TEST_CASE(testHexagonSquare) {
    Point p1{-2, 0};
    Point p2{0, 2};
    Point p3{2, 0};
    Point p4{2, -1};
    Point p5{1, -2};
    Point p6{0, -2};
    Hexagon h{p1, p2, p3, p4, p5, p6};
    BOOST_CHECK_EQUAL(h.Square(), 9.5);
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(lab3)

enable_testing()

set(Figures_source Figures.cpp)

add_library(figures STATIC ${Figures_source})

find_package(Boost COMPONENTS unit_test_framework REQUIRED)

add_executable(testFigures test.cpp)
target_link_libraries(testFigures ${Boost_LIBRARIES} figures)
add_test(NAME Test1 COMMAND test1)

add_executable(lab3
    main.cpp
    Figures.cpp)

set_property(TARGET lab3 PROPERTY CXX_STANDARD 17)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -Werror")
```

2 Тестирование

test_01.txt:

Попробуем добавить в вектор фигуру с координатами (-5, 0), (-4, -1), (-3, -1), (-2, 0), которая очевидно не является ромбом, рассчитывая получить сообщение об ошибке. Затем добавим в вектор ромб с координатами (-5, 0), (-3, 1), (-1, 0), (-3, -1), площадь которого равна 4, а центр находится в точке (-3, 0), а также пятиугольник с координатами (-3.000, 0.000), (-2.000, 1.000), (-1.000, 1.000), (0.000, 0.000), (-1.000, -1.000), площадь которого равна 3.5 и шестиугольник с координатами (-3.000, 0.000), (-2.000, 1.000), (-1.000, 1.000), (0.000, 0.000), (-1.000, -1.000), (-2.000, -1.000), (-1.500, -0.000) с площадью равной 4. Попробуем удалить из вектора элемент, находящийся на 3 позиции, надеясь получить сообщение об ошибке. Затем выведем все фигуры, а также найдем общую площадь фигур в массиве, которая должна быть равна 11.5, затем удалим шестиугольник и еще раз выведем все фигуры.

Результат:

1 - add figure to the vector
2 - delete figure from the vector
3 - call common functions for the whole vector
4 - get total area of figures in vector
0 - exit
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Entered coordinates are not forming Rhombus. Try entering new coordinates
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Enter index
Element with such index doesn't exist
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
7.5
Rhombus: [-5.000, 0.000], [-3.000, 1.000], [-1.000, 0.000], [-3.000, -1.000]
[-3.000, 0.000]
4.000
Pentagon: [-3.000, 0.000], [-2.000, 1.000], [-1.000, 1.000], [0.000, 0.000], [-1.000, -1.000]
[-1.429, 0.095]
3.500
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Rhombus: [-5.000, 0.000], [-3.000, 1.000], [-1.000, 0.000], [-3.000, -1.000]
[-3.000, 0.000]
4.000
Pentagon: [-3.000, 0.000], [-2.000, 1.000], [-1.000, 1.000], [0.000, 0.000], [-1.000, -1.000]
[-1.429, 0.095]
3.500
Hexagon: [-3.000, 0.000], [-2.000, 1.000], [-1.000, 1.000], [0.000, 0.000], [-1.000, -1.000], [-2.000, -1.000]
[-1.500, -0.000]
4.000
11.500

Enter index
Rhombus: [-5.000, 0.000], [-3.000, 1.000], [-1.000, 0.000], [-3.000, -1.000]
[-3.000, 0.000]
4.000
Pentagon: [-3.000, 0.000], [-2.000, 1.000], [-1.000, 1.000], [0.000, 0.000], [-1.000, -1.000]
[-1.429, 0.095]
3.500

test_02.txt

Добавим в вектор ромб с координатами [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000], центром в точке [8, 0] и площадью равной 16, квадрат с координатами [4.000, 2.000], [8.000, 2.000], [8.000, -2.000], [4.000, -2.000] с центром в точке [6, 0] и площадью равной 16, пятиугольник с координатами [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000], [6.000, -2.000] и площадью равной 18. Затем выведем все фигуры и найдем общую площадь, которая должна быть равна 50. Добавим шестиугольник с координатами [4.000, 0.000], [8.000, 2.000], [10.000, 2.000], [12.000, 0.000], [8.000, -2.000], [6.000, -2.000] и площадью равной 20. Еще раз выведем все фигуры и найдем общую площадь, которая должна быть равна 70. Затем удалим пятиугольник и шестиугольник и еще раз выведем все фигуры.

Результат:
1 - add figure to the vector
2 - delete figure from the vector
3 - call common functions for the whole vector
4 - get total area of figures in vector
0 - exit
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Rhombus: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000]
[8.000, 0.000]
16.000
Rhombus: [4.000, 2.000], [8.000, 2.000], [8.000, -2.000], [4.000, -2.000]
[6.000, 0.000]
16.000
Pentagon: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000], [6.000, -2.000]
[7.778, -0.148]
18.000
50.000
1 - Rhombus, 2 - Pentagon, 3 - Hexagon
Rhombus: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000]
[8.000, 0.000]
16.000
Rhombus: [4.000, 2.000], [8.000, 2.000], [8.000, -2.000], [4.000, -2.000]
[6.000, 0.000]
16.000

Pentagon: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000], [6.000, -2.000]
[7.778, -0.148]
18.000
Hexagon: [4.000, 0.000], [8.000, 2.000], [10.000, 2.000], [12.000, 0.000], [8.000, -2.000],
[6.000, -2.000]
[8.000, 0.000]
20.000
70.000
Enter index
Enter index
Rhombus: [4.000, 0.000], [8.000, 2.000], [12.000, 0.000], [8.000, -2.000]
[8.000, 0.000]
16.000
Rhombus: [4.000, 2.000], [8.000, 2.000], [8.000, -2.000], [4.000, -2.000]
[6.000, 0.000]
16.000

3 Объяснение результатов работы программы

При вводе координат для создания ромба производится проверка этих координат, ведь они могут не образовывать ромб. Для этого реализована функция checkIfRhombus, которая вычисляет расстояния от одной точки до трёх остальных, а поскольку фигура является ромбом, то два из низ должны быть равны. Третье же значение функция возвращает ведь оно равно длине одной из диагоналей. Площадь ромба вычисляется как половина произведения диагоналей, центр - точка пересечения диагоналей. Методы вычисления площади и центра для пяти- и шестиугольника совпадают. Чтобы найти площадь необходимо перебрать все ребра и сложить площади трапеций, ограниченных этими ребрами. Чтобы найти центр необходимо разбить фигуры на треугольники(найти одну точку внутри фигуры), для каждого треугольника найти центроид и площадь и перемножить их, просуммировать полученные величины и разделить на общую площадь фигуры.

4 Выводы

В ходе выполнения лабораторной работы я познакомился с таким понятием как runtime-полиморфизм. Также я познакомился с библиотекой для юнит-тестов из коллекций библиотек boost, которую уже могу сравнить с ранее использованным googletest. На мой взгляд, googletest предоставляет чуть больше возможностей для тестирования кода и имеет более приятную организацию тестов.