

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

**Лабораторная работа №7 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_07

figure.hpp

```
1 #pragma once
2
3 #include<iostream>
4
5 #include "point.hpp"
6
7 enum class Figures {Rhombus, Pentagon, Hexagon};
8
9 class Figure {
10 public:
11     virtual Point Center() const = 0;
12     virtual double Square() const = 0;
13     virtual void Print(std::ostream& os) const = 0;
14     virtual ~Figure() = default;
15     virtual void serialize(std::ostream& os) const = 0;
16     virtual int getID() const = 0;
17 };
```

rhombus.hpp

```
1 #pragma once
2
3 #include <array>
4
5 #include "figure.hpp"
6 #include "point.hpp"
7
8 class Rhombus : public Figure {
9 public:
10     Rhombus(Point* p, int id);
11     Rhombus(std::istream& is);
12     Point Center() const override;
13     double Square() const override;
14     void Print(std::ostream& os) const override;
15     int getID() const override;
16     void serialize(std::ostream& os) const override;
17 private:
18     std::array<Point, 4> points;
19     double smallerDiagonal, biggerDiagonal;
20     int id;
21 };
```

rhombus.cpp

```
1 #include "rhombus.hpp"
2
3 double checkIfRhombus(const Point& p1, const Point& p2, const
4     Point& p3, const Point& p4) {
5     double d1 = calculateDistance(p1, p2);
6     double d2 = calculateDistance(p1, p3);
7     double d3 = calculateDistance(p1, p4);
8     if(d1 == d2) {
```

```

9         return d3;
10    } else if(d1 == d3) {
11        return d2;
12    } else if(d2 == d3) {
13        return d1;
14    } else {
15        throw std::invalid_argument("Entered coordinates are not
forming Rhombus. Try entering new coordinates");
16    }
17 }
18
19 Rhombus::Rhombus(Point* p, int id) {
20     Point p1, p2, p3, p4;
21     p1 = p[0];
22     p2 = p[1];
23     p3 = p[2];
24     p4 = p[3];
25     try {
26         double d1 = checkIfRhombus(p1, p2, p3, p4);
27         double d2 = checkIfRhombus(p2, p1, p3, p4);
28         double d3 = checkIfRhombus(p3, p1, p2, p4);
29         double d4 = checkIfRhombus(p4, p1, p2, p3);
30         if(d1 == d2 || d1 == d4) {
31             if(d1 < d3) {
32                 smallerDiagonal = d1;
33                 biggerDiagonal = d3;
34
35             } else {
36                 smallerDiagonal = d3;
37                 biggerDiagonal = d1;
38             }
39         } else if(d1 == d3) {
40             if(d1 < d2) {
41                 smallerDiagonal = d1;
42                 biggerDiagonal = d2;
43             } else {
44                 smallerDiagonal = d2;
45                 biggerDiagonal = d1;
46             }
47         }
48     } catch(std::exception& e) {
49         throw std::invalid_argument(e.what());
50         return;
51     }
52     points[0] = p1;
53     points[1] = p2;
54     points[2] = p3;
55     points[3] = p4;
56     this->id = id;
57 }
58
59 Rhombus::Rhombus(std::istream& is) {
60     Point p1, p2, p3, p4;
61     is >> p1 >> p2 >> p3 >> p4;
62     try {
63         double d1 = checkIfRhombus(p1, p2, p3, p4);
64         double d2 = checkIfRhombus(p2, p1, p3, p4);
65         double d3 = checkIfRhombus(p3, p1, p2, p4);
66         double d4 = checkIfRhombus(p4, p1, p2, p3);

```

```

67         if(d1 == d2 || d1 == d4) {
68             if(d1 < d3) {
69                 smallerDiagonal = d1;
70                 biggerDiagonal = d3;
71             } else {
72                 smallerDiagonal = d3;
73                 biggerDiagonal = d1;
74             }
75         } else if(d1 == d3) {
76             if(d1 < d2) {
77                 smallerDiagonal = d1;
78                 biggerDiagonal = d2;
79             } else {
80                 smallerDiagonal = d2;
81                 biggerDiagonal = d1;
82             }
83         }
84     }
85     } catch(std::exception& e) {
86         throw std::invalid_argument(e.what());
87         return;
88     }
89     points[0] = p1;
90     points[1] = p2;
91     points[2] = p3;
92     points[3] = p4;
93 }
94
95 Point Rhombus::Center() const {
96     if(calculateDistance(points[0], points[1]) == smallerDiagonal
97        ||
98        calculateDistance(points[0], points[1]) ==
99        biggerDiagonal) {
100         return {(points[0].x + points[1].x) / 2.0, ((points[0].y
101            + points[1].y) / 2.0)};
102     } else if(calculateDistance(points[0], points[2]) ==
103        smallerDiagonal ||
104        calculateDistance(points[0], points[2]) ==
105        biggerDiagonal) {
106         return {(points[0].x + points[2].x) / 2.0, ((points[0].y
107            + points[2].y) / 2.0)};
108     } else {
109         return {(points[0].x + points[3].x) / 2.0, ((points[0].y
110            + points[3].y) / 2.0)};
111     }
112 }
113
114 double Rhombus::Square() const {
115     return smallerDiagonal * biggerDiagonal / 2.0;
116 }
117
118 void Rhombus::Print(std::ostream& os) const {
119     os << "Rhombus: ";
120     for(const auto& p : points) {
121         os << p << ' ';
122     }
123     os << "Center: " << this->Center() << ' ';
124     os << "Area: " << this->Square() << ' ';
125     os << "ID: " << id;

```

```

119     os << std::endl;
120 }
121
122 int Rhombus::getID() const {
123     return id;
124 }
125
126 void Rhombus::serialize(std::ostream& os) const {
127     os << points.size() << ' ';
128     for(const auto& p : points) {
129         os << p.x << ' ' << p.y << ' ';
130     }
131     os << std::endl;
132 }

```

pentagon.hpp

```

1 #pragma once
2
3 #include <iostream>
4 #include <array>
5
6 #include "figure.hpp"
7 #include "point.hpp"
8
9 class Pentagon : public Figure {
10 public:
11     Pentagon(Point* p, int id);
12     Pentagon(std::istream& is);
13     Point Center() const override;
14     double Square() const override;
15     void Print(std::ostream& os) const override;
16     int getID() const override;
17     void serialize(std::ostream& os) const override;
18 private:
19     std::array<Point, 5> points;
20     int id;
21 };

```

pentagon.cpp

```

1 #include <cmath>
2
3 #include "pentagon.hpp"
4
5 Pentagon::Pentagon(Point* p, int id) {
6     for(int i = 0; i < 5; ++i) {
7         points[i] = p[i];
8     }
9     this->id = id;
10 }
11
12 Pentagon::Pentagon(std::istream& is) {
13     is >> points[0] >> points[1] >> points[2] >> points[3] >>
        points[4];
14 }
15
16 Point Pentagon::Center() const {
17     Point insideFigure{0, 0};

```

```

18     Point result{0, 0};
19     double square = this->Square();
20     for(unsigned i = 0; i < points.size(); ++i) {
21         insideFigure.x += points[i].x;
22         insideFigure.y += points[i].y;
23     }
24     insideFigure.x /= points.size();
25     insideFigure.y /= points.size();
26     for(unsigned i = 0; i < points.size(); ++i) {
27         double tempSquare = triangleSquare(points[i], points[(i +
1) % points.size()],
28             insideFigure);
29         result.x += tempSquare * (points[i].x + points[(i + 1) %
points.size()].x
30             + insideFigure.x) / 3.0;
31         result.y += tempSquare * (points[i].y + points[(i + 1) %
points.size()].y
32             + insideFigure.y) / 3.0;
33     }
34     result.x /= square;
35     result.y /= square;
36     return result;
37 }
38
39 double Pentagon::Square() const {
40     double result = 0;
41     for(unsigned i = 0; i < points.size(); ++i) {
42         Point p1 = i ? points[i - 1] : points[points.size() - 1];
43         Point p2 = points[i];
44         result += (p1.x - p2.x) * (p1.y + p2.y);
45     }
46     return fabs(result) / 2.0;
47 }
48
49 void Pentagon::Print(std::ostream& os) const {
50     os << "Pentagon: ";
51     for(const auto& p : points) {
52         os << p << ' ';
53     }
54     os << "Center: " << this->Center() << ' ';
55     os << "Area: " << this->Square() << ' ';
56     os << "ID: " << id;
57     os << std::endl;
58 }
59
60 int Pentagon::getID() const {
61     return id;
62 }
63
64 void Pentagon::serialize(std::ostream& os) const {
65     os << points.size() << ' ';
66     for(const auto& p : points) {
67         os << p.x << ' ' << p.y << ' ';
68     }
69     os << std::endl;
70 }

```

hexagon.hpp

```

1 #pragma once
2
3 #include <iostream>
4 #include <array>
5
6 #include "figure.hpp"
7 #include "point.hpp"
8
9 class Hexagon : public Figure {
10 public:
11     Hexagon(Point* p, int id);
12     Hexagon(std::istream& is);
13     Point Center() const override;
14     double Square() const override;
15     void Print(std::ostream& os) const override;
16     int getID() const override;
17     void serialize(std::ostream& os) const override;
18 private:
19     std::array<Point, 6> points;
20     int id;
21 };

```

hexagon.cpp

```

1 #include <cmath>
2
3 #include "hexagon.hpp"
4
5 Hexagon::Hexagon(Point* p, int id) {
6     for(int i = 0; i < 6; ++i) {
7         points[i] = p[i];
8     }
9     this->id = id;
10 }
11
12 Hexagon::Hexagon(std::istream& is) {
13     is >> points[0] >> points[1] >> points[2] >> points[3] >>
14     points[4] >> points[5];
15 }
16
17 Point Hexagon::Center() const {
18     Point insideFigure{0, 0};
19     Point result{0, 0};
20     double square = this->Square();
21     for(unsigned i = 0; i < points.size(); ++i) {
22         insideFigure.x += points[i].x;
23         insideFigure.y += points[i].y;
24     }
25     insideFigure.x /= points.size();
26     insideFigure.y /= points.size();
27     for(unsigned i = 0; i < points.size(); ++i) {
28         double tempSquare = triangleSquare(points[i], points[(i +
29 1) % points.size()],
30         insideFigure);
31         result.x += tempSquare * (points[i].x + points[(i + 1) %
32 points.size()]).x
33         + insideFigure.x) / 3.0;
34         result.y += tempSquare * (points[i].y + points[(i + 1) %
35 points.size()]).y

```

```

32         + insideFigure.y) / 3.0;
33     }
34     result.x /= square;
35     result.y /= square;
36     return result;
37 }
38
39 double Hexagon::Square() const {
40     double result = 0;
41     for(unsigned i = 0; i < points.size(); ++i) {
42         Point p1 = i ? points[i - 1] : points[points.size() - 1];
43         Point p2 = points[i];
44         result += (p1.x - p2.x) * (p1.y + p2.y);
45     }
46     return fabs(result) / 2.0;
47 }
48
49 void Hexagon::Print(std::ostream& os) const {
50     os << "Hexagon:";
51     for(const auto& p : points) {
52         os << p << ' ';
53     }
54     os << "Center: " << this->Center() << ' ';
55     os << "Area: " << this->Square() << ' ';
56     os << "ID: " << id;
57     os << std::endl;
58 }
59
60 int Hexagon::getID() const {
61     return id;
62 }
63
64 void Hexagon::serialize(std::ostream& os) const {
65     os << points.size() << ' ';
66     for(const auto& p : points) {
67         os << p.x << ' ' << p.y << ' ';
68     }
69     os << std::endl;
70 }

```

point.hpp

```

1 #pragma once
2
3 #include <iostream>
4
5 struct Point {
6     double x, y;
7 };
8
9 double calculateDistance(const Point& lhs, const Point& rhs);
10 bool operator<(const Point& lhs, const Point& rhs);
11 std::istream& operator>>(std::istream& is, Point& p);
12 std::ostream& operator<<(std::ostream& os, const Point& p);
13 double triangleSquare(const Point& p1, const Point& p2, const
    Point& p3);

```

point.cpp


```

1 #include <iostream>
2 #include <cmath>
3 #include <iomanip>
4
5 #include "point.hpp"
6
7 double calculateDistance(const Point& lhs, const Point& rhs) {
8     return sqrt(pow(rhs.x - lhs.x, 2) + pow(rhs.y - lhs.y, 2));
9 }
10
11 double triangleSquare(const Point& p1, const Point& p2, const
    Point& p3) {
12     return 0.5 * fabs((p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x
        ) * (p1.y - p3.y));
13 }
14
15 bool operator<(const Point& lhs, const Point& rhs) {
16     if(lhs.x != rhs.x) {
17         return lhs.x < rhs.x;
18     }
19     return lhs.y < rhs.y;
20 }
21
22 std::istream& operator>>(std::istream& is, Point& p) {
23     is >> p.x >> p.y;
24     return is;
25 }
26
27 std::ostream& operator<<(std::ostream& os, const Point& p) {
28     os << std::fixed << std::setprecision(3) << "[" << p.x << ", "
        << p.y << "];";
29     return os;
30 }

```

command.hpp

```

1 #pragma once
2
3 #include <memory>
4
5 #include "figure.hpp"
6 #include "document.hpp"
7
8 class Command {
9 public:
10     virtual void exec() = 0;
11     virtual void undo() = 0;
12     virtual ~Command() = default;
13 protected:
14     std::shared_ptr<Document> document;
15 };
16
17 class InsertCommand : public Command {
18 public:
19     InsertCommand(std::shared_ptr<Document> document, Figures
        figure,
20         Point* points) : figure(figure), points(points) {this
        ->document = document;};
21

```

```

22     void exec() override {
23         document->insert(figure, points);
24     }
25
26     void undo() override {
27         document->popBack();
28     }
29 private:
30     Figures figure;
31     Point* points;
32 };
33
34 class RemoveCommand : public Command {
35 public:
36     RemoveCommand(std::shared_ptr<Document> document, int id) :
37         id(id), position(-1), figure(nullptr) {this->document =
document;};
38
39     void exec() override {
40         try {
41             figure = document->getFigure(id);
42             position = document->getPosition(id);
43         } catch(std::exception& e) {
44             std::cout << e.what() << std::endl;
45             return;
46         }
47         document->remove(id);
48     }
49
50     void undo() override {
51         document->insert(position, figure);
52     }
53 private:
54     int id;
55     int position;
56     std::shared_ptr<Figure> figure;
57 };

```

editor.hpp

```

1  #pragma once
2
3  #include <stack>
4
5  #include "command.hpp"
6  #include "document.hpp"
7  #include "figure.hpp"
8
9  class Editor {
10 public:
11     Editor() : document(nullptr) {};
12
13     void createDocument() {
14         document = std::make_shared<Document>();
15     }
16
17     void insert(Figures figure, Point* points) {
18         std::shared_ptr<Command> command = std::shared_ptr<Command>
>(new InsertCommand(document, figure, points));

```

```

19         command->exec();
20         commandStack.push(command);
21     }
22
23     void remove(int id) {
24         try {
25             std::shared_ptr<Command> command = std::shared_ptr<
Command>(new RemoveCommand(document, id));
26             command->exec();
27             commandStack.push(command);
28         } catch(std::exception& e) {
29             std::cout << e.what() << std::endl;
30         }
31     }
32
33     void saveDocument(const std::string& filename) {
34         document->save(filename);
35     }
36
37     void loadDocument(const std::string& filename) {
38         createDocument();
39         document->load(filename);
40     }
41
42     void undo() {
43         if(commandStack.empty()) {
44             throw std::logic_error("Nothing to undo");
45         }
46         std::shared_ptr<Command> command = commandStack.top();
47         command->undo();
48         commandStack.pop();
49     }
50
51     void print() {
52         document->print();
53     }
54 private:
55     std::shared_ptr<Document> document;
56     std::stack<std::shared_ptr<Command>> commandStack;
57 };

```

factory.hpp

```

1 #pragma once
2
3 #include <memory>
4
5 #include "figure.hpp"
6 #include "rhombus.hpp"
7 #include "pentagon.hpp"
8 #include "hexagon.hpp"
9
10 class Factory {
11 public:
12     std::shared_ptr<Figure> createFigure(Figures type, Point* p,
int id) {
13         if(type == Figures::Rhombus) {
14             try {
15                 Rhombus{p, id};

```

```

16         } catch(std::exception& e) {
17             std::cout << e.what() << std::endl;
18             return nullptr;
19         }
20         return std::make_shared<Rhombus>(Rhombus{p, id});
21     } else if(type == Figures::Pentagon) {
22         return std::make_shared<Pentagon>(Pentagon{p, id});
23     } else if(type == Figures::Hexagon) {
24         return std::make_shared<Hexagon>(Hexagon{p, id});
25     } else {
26         return nullptr;
27     }
28 }
29 };

```

document.hpp

```

1  #pragma once
2
3  #include <vector>
4  #include <string>
5  #include <algorithm>
6  #include <fstream>
7  #include <stack>
8
9  #include "figure.hpp"
10 #include "factory.hpp"
11 #include "command.hpp"
12
13 class Document {
14 friend class Command;
15 public:
16     Document() : currentFigureID(0) {};
17
18     void newDocument() {
19         content.clear();
20         currentFigureID = 0;
21     }
22
23     void save(const std::string& fileName) {
24         serialize(fileName);
25     }
26
27     void load(const std::string& fileName) {
28         deserialize(fileName);
29     }
30
31     void print() {
32         for(const auto& figure : content) {
33             figure->Print(std::cout);
34         }
35     }
36
37     void insert(Figures type, Point* points) {
38         if(type == Figures::Rhombus) {
39             try {
40                 Rhombus{points, currentFigureID};
41             } catch(std::exception& e) {
42                 std::cout << e.what() << std::endl;

```

```

43         return;
44     }
45     content.push_back(factory.createFigure(Figures::
Rhombus, points, currentFigureID));
46     } else if(type == Figures::Pentagon) {
47         content.push_back(factory.createFigure(Figures::
Pentagon, points, currentFigureID));
48     } else if(type == Figures::Hexagon) {
49         content.push_back(factory.createFigure(Figures::
Hexagon, points, currentFigureID));
50     }
51     currentFigureID++;
52 }
53
54 void insert(unsigned position, std::shared_ptr<Figure> figure)
55 {
56     auto it = content.begin();
57     std::advance(it, position);
58     content.insert(it, figure);
59 }
60
61 void remove(int id) {
62     unsigned temp = content.size();
63     auto it = std::remove_if(content.begin(), content.end(), [
id](std::shared_ptr<Figure> f)
64     {return id == f->getID();});
65     content.erase(it, content.end());
66     if(temp == content.size()) {
67         throw std::invalid_argument("Figure this such ID doesn
't exist");
68     }
69 }
70
71 void popBack() {
72     if(!content.size()) {
73         throw std::logic_error("Document is empty");
74     }
75     content.pop_back();
76 }
77
78 std::shared_ptr<Figure> getFigure(int id) {
79     for(const auto& figure : content) {
80         if(id == figure->getID()) {
81             return figure;
82         }
83     }
84     throw std::invalid_argument("1:No figure this such ID");
85 }
86
87 int getPosition(int id) {
88     int n = content.size();
89     for(int i = 0; i < n; ++i) {
90         if(id == content[i]->getID()) {
91             return i;
92         }
93     }
94     throw std::invalid_argument("2:No figure with such ID");
95 }
96 private:

```

```

96     int currentFigureID;
97     std::vector<std::shared_ptr<Figure>> content;
98     Factory factory;
99     void serialize(const std::string& fileName) {
100         std::ofstream os(fileName, std::ios::trunc);
101         if(!os) {
102             throw std::runtime_error("Couldn't open file");
103         }
104         os << content.size() << std::endl;
105         for(const auto& figure : content) {
106             figure->serialize(os);
107         }
108     }
109
110     void deserialize(const std::string& fileName) {
111         std::ifstream is(fileName);
112         if(!is) {
113             throw std::runtime_error("Couldn't open file");
114         }
115         this->newDocument();
116         int numberOfFigures;
117         is >> numberOfFigures;
118         int numberOfPoints;
119         Point p;
120         while(numberOfFigures--) {
121             is >> numberOfPoints;
122             Point* points = new Point[numberOfPoints];
123             for(int i = 0; i < numberOfPoints; ++i) {
124                 is >> p;
125                 points[i] = p;
126             }
127             if(numberOfPoints == 4) {
128                 content.push_back(factory.createFigure(Figures::
Rhombus, points, currentFigureID));
129             } else if(numberOfPoints == 5) {
130                 content.push_back(factory.createFigure(Figures::
Pentagon, points, currentFigureID));
131             } else if(numberOfPoints == 6) {
132                 content.push_back(factory.createFigure(Figures::
Hexagon, points, currentFigureID));
133             }
134             this->currentFigureID++;
135             delete[] points;
136         }
137     }
138 };

```

main.cpp

```

1  #include <iostream>
2  #include <string>
3
4  #include "editor.hpp"
5
6  void help() {
7      std::cout << "new - Creates new document" << std::endl;
8      std::cout << "save <path to file> - saves document to file" <<
std::endl;
9      std::cout << "load <path to file> - loads document from file"

```

```

10     << std::endl;
11     std::cout << "add R/P/H <coordinates> - adds Rhombus/Pentagon/
Hexagon to the document" << std::endl;
12     std::cout << "remove <Figure ID> - removes figure with given
ID if it is present" << std::endl;
13     std::cout << "undo - undo last action" << std::endl;
14     std::cout << "print - prints information about all figures
from document" << std::endl;
15     std::cout << "help - do I really need to explain what help
does?" << std::endl;
16     std::cout << "exit - exit editor" << std::endl;
17 }
18
19 int main() {
20     int id;
21     std::string command;
22     std::string filepath;
23     std::string figureType;
24     Editor e;
25     help();
26     while(std::cin >> command) {
27         if(command == "new") {
28             e.createDocument();
29         } else if(command == "save") {
30             std::cin >> filepath;
31             try {
32                 e.saveDocument(filepath);
33             } catch(std::exception& e) {
34                 std::cout << e.what() << std::endl;
35             }
36         } else if(command == "load") {
37             std::cin >> filepath;
38             try {
39                 e.loadDocument(filepath);
40             } catch(std::exception& e) {
41                 std::cout << e.what() << std::endl;
42             }
43         } else if(command == "add") {
44             std::cin >> figureType;
45             if(figureType == "R") {
46                 Point* p = new Point[4];
47                 for(int i = 0; i < 4; ++i) {
48                     std::cin >> p[i];
49                 }
50                 try {
51                     e.insert(Figures::Rhombus, p);
52                 } catch(std::exception& e) {
53                     std::cout << e.what() << std::endl;
54                 }
55                 delete[] p;
56             } else if(figureType == "P") {
57                 Point* p = new Point[5];
58                 for(int i = 0; i < 5; ++i) {
59                     std::cin >> p[i];
60                 }
61                 e.insert(Figures::Pentagon, p);
62                 delete[] p;
63             } else if(figureType == "H") {
64                 Point* p = new Point[6];

```

```

64         for(int i = 0; i < 6; ++i) {
65             std::cin >> p[i];
66         }
67         e.insert(Figures::Hexagon, p);
68         delete[] p;
69     } else {
70         std::cout << "Unknown figure" << std::endl;
71     }
72 } else if(command == "remove") {
73     std::cin >> id;
74     try {
75         e.remove(id);
76     } catch(std::exception& e) {
77         std::cout << e.what() << std::endl;
78     }
79 } else if(command == "undo") {
80     try {
81         e.undo();
82     } catch(std::exception& e) {
83         std::cout << e.what() << std::endl;
84     }
85 } else if(command == "print") {
86     e.print();
87 } else if(command == "help") {
88     help();
89 } else if(command == "exit") {
90     break;
91 } else {
92     std::cout << "Unknown figure" << std::endl;
93 }
94 }
95 return 0;
96 }

```

CMakeLists.txt

```

1 cmake_minimum_required(VERSION 3.1)
2
3 project(lab7)
4
5 add_executable(lab7
6     main.cpp
7     point.cpp
8     rhombus.cpp
9     pentagon.cpp
10    hexagon.cpp)
11
12 set_property(TARGET lab7 PROPERTY CXX_STANDARD 17)
13
14 set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -Werror")

```


2 Тестирование

test_01.txt:

Попробуем добавить в документ фигуру с координатами (-5, 0), (-4, -1), (-3, -1), (-2, 0), которая очевидно не является ромбом, рассчитывая получить сообщение об ошибке. Затем добавим 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. Затем удалим шестиугольник и пятиугольник, еще раз выведем содержимое документа и сделаем undo.

Результат:

new - Creates new document

save <path to file> - saves document to file

load <path to file> - loads document from file

add R/P/H <coordinates> - adds Rhombus/Pentagon/Hexagon to the document

remove <Figure ID> - removes figure with given ID if it is present

undo - undo last action

print - prints information about all figures from document

help - do I really need to explain what help does?

exit - exit editor

new

add R -5 0 -4 -1 -3 -1 -2 0

Entered coordinates are not forming Rhombus. Try entering new coordinates

print

add R -5 0 -3 1 -1 0 -3 -1

add P -3 0 -2 1 -1 1 0 0 -1 -1

add H -3 0 -2 1 -1 1 0 0 -1 -1 -2 -1

print

Rhombus: [-5.000, 0.000] [-3.000, 1.000] [-1.000, 0.000] [-3.000, -1.000] Center: [-3.000, 0.000]

Area: 4.000 ID: 0

Pentagon: [-3.000, 0.000] [-2.000, 1.000] [-1.000, 1.000] [0.000, 0.000] [-1.000, -1.000] Center:

[-1.429, 0.095] Area: 3.500 ID: 1

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] Center: [-1.500, -0.000] Area: 4.000 ID: 2

remove 2

remove 1

print

Rhombus: [-5.000, 0.000] [-3.000, 1.000] [-1.000, 0.000] [-3.000, -1.000] Center: [-3.000, 0.000]

Area: 4.000 ID: 0

undo

print

Rhombus: [-5.000, 0.000] [-3.000, 1.000] [-1.000, 0.000] [-3.000, -1.000] Center:
 [-3.000, 0.000]
 Area: 4.000 ID: 0
 Pentagon: [-3.000, 0.000] [-2.000, 1.000] [-1.000, 1.000] [0.000, 0.000] [-1.000,
 -1.000] Center:
 [-1.429, 0.095] Area: 3.500 ID: 1
 exit

test_02.txt

Добавим в документ 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 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, сделаем undo, удалим пятиугольник и квадрат и еще раз выведем все фигуры.

Результат:

new - Creates new document
 save <path to file> - saves document to file
 load <path to file> - loads document from file
 add R/P/H <coordinates> - adds Rhombus/Pentagon/Hexagon to the document
 remove <Figure ID> - removes figure with given ID if it is present
 undo - undo last action
 print - prints information about all figures from document
 help - do I really need to explain what help does?
 exit - exit editor
 Rhombus: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000] Center:
 [8.000, 0.000]
 Area: 16.000 ID: 0
 Rhombus: [4.000, 2.000] [8.000, 2.000] [8.000, -2.000] [4.000, -2.000] Center:
 [6.000, 0.000]
 Area: 16.000 ID: 1
 Pentagon: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000] [6.000, -
 2.000] Center:
 [7.778, -0.148] Area: 18.000 ID: 2
 Rhombus: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000] Center:
 [8.000, 0.000]
 Area: 16.000 ID: 0
 Rhombus: [4.000, 2.000] [8.000, 2.000] [8.000, -2.000] [4.000, -2.000] Center:
 [6.000, 0.000]
 Area: 16.000 ID: 1
 Pentagon: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000] [6.000, -
 2.000] Center:
 [7.778, -0.148] Area: 18.000 ID: 2
 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] Center: [8.000, 0.000] Area: 20.000 ID: 3

Rhombus: [4.000, 0.000] [8.000, 2.000] [12.000, 0.000] [8.000, -2.000] Center:
[8.000, 0.000]
Area: 16.000 ID: 0

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 них 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 работы я познакомился с некоторыми принципами и паттернами проектирования программ, что позволило достаточно неплохо организовать структуру классов моей программы.