

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](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, int id);
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, int id) {
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     this->id = id;
94 }
95
96 Point Rhombus::Center() const {
97     if(calculateDistance(points[0], points[1]) == smallerDiagonal
98     ||
99         calculateDistance(points[0], points[1]) ==
100 biggerDiagonal) {
101         return {(points[0].x + points[1].x) / 2.0, ((points[0].y
102 + points[1].y) / 2.0)};
103     } else if(calculateDistance(points[0], points[2]) ==
104 smallerDiagonal ||
105         calculateDistance(points[0], points[2]) ==
106 biggerDiagonal) {
107         return {(points[0].x + points[2].x) / 2.0, ((points[0].y
108 + points[2].y) / 2.0)};
109     } else {
110         return {(points[0].x + points[3].x) / 2.0, ((points[0].y
111 + points[3].y) / 2.0)};
112     }
113 }
114
115 double Rhombus::Square() const {
116     return smallerDiagonal * biggerDiagonal / 2.0;
117 }
118
119 void Rhombus::Print(std::ostream& os) const {
120     os << "Rhombus: ";
121     for(const auto& p : points) {
122         os << p << ' ';
123     }
124     os << "Center: " << this->Center() << ' ';
125     os << "Area: " << this->Square() << ' ';

```

```

119     os << "ID: " << id;
120     os << std::endl;
121 }
122
123 int Rhombus::getID() const {
124     return id;
125 }
126
127 void Rhombus::serialize(std::ostream& os) const {
128     os << 'R' << ' ';
129     for(const auto& p : points) {
130         os << p.x << ' ' << p.y << ' ';
131     }
132     os << std::endl;
133 }

```

## 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, int id);
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, int id) {
13     is >> points[0] >> points[1] >> points[2] >> points[3] >>
        points[4];
14     this->id = id;
15 }
16

```

```

17 Point Pentagon::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 +
1) % points.size()],
29             insideFigure);
30         result.x += tempSquare * (points[i].x + points[(i + 1) %
points.size()].x
31             + insideFigure.x) / 3.0;
32         result.y += tempSquare * (points[i].y + points[(i + 1) %
points.size()].y
33             + insideFigure.y) / 3.0;
34     }
35     result.x /= square;
36     result.y /= square;
37     return result;
38 }
39
40 double Pentagon::Square() const {
41     double result = 0;
42     for(unsigned i = 0; i < points.size(); ++i) {
43         Point p1 = i ? points[i - 1] : points[points.size() - 1];
44         Point p2 = points[i];
45         result += (p1.x - p2.x) * (p1.y + p2.y);
46     }
47     return fabs(result) / 2.0;
48 }
49
50 void Pentagon::Print(std::ostream& os) const {
51     os << "Pentagon: ";
52     for(const auto& p : points) {
53         os << p << ' ';
54     }
55     os << "Center: " << this->Center() << ' ';
56     os << "Area: " << this->Square() << ' ';
57     os << "ID: " << id;
58     os << std::endl;
59 }
60
61 int Pentagon::getID() const {
62     return id;
63 }
64
65 void Pentagon::serialize(std::ostream& os) const {
66     os << 'P' << ' ';
67     for(const auto& p : points) {
68         os << p.x << ' ' << p.y << ' ';
69     }
70     os << std::endl;
71 }

```

## 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, int id);
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, int id) {
13     is >> points[0] >> points[1] >> points[2] >> points[3] >>
14     points[4] >> points[5];
15     this->id = id;
16 }
17
18 Point Hexagon::Center() const {
19     Point insideFigure{0, 0};
20     Point result{0, 0};
21     double square = this->Square();
22     for(unsigned i = 0; i < points.size(); ++i) {
23         insideFigure.x += points[i].x;
24         insideFigure.y += points[i].y;
25     }
26     insideFigure.x /= points.size();
27     insideFigure.y /= points.size();
28     for(unsigned i = 0; i < points.size(); ++i) {
29         double tempSquare = triangleSquare(points[i], points[(i +
30 1) % points.size()],
31         insideFigure);
32         result.x += tempSquare * (points[i].x + points[(i + 1) %
33 points.size()].x
34 + insideFigure.x) / 3.0;
```

```

32         result.y += tempSquare * (points[i].y + points[(i + 1) %
points.size()].y
33             + insideFigure.y) / 3.0;
34     }
35     result.x /= square;
36     result.y /= square;
37     return result;
38 }
39
40 double Hexagon::Square() const {
41     double result = 0;
42     for(unsigned i = 0; i < points.size(); ++i) {
43         Point p1 = i ? points[i - 1] : points[points.size() - 1];
44         Point p2 = points[i];
45         result += (p1.x - p2.x) * (p1.y + p2.y);
46     }
47     return fabs(result) / 2.0;
48 }
49
50 void Hexagon::Print(std::ostream& os) const {
51     os << "Hexagon:";
52     for(const auto& p : points) {
53         os << p << ' ';
54     }
55     os << "Center: " << this->Center() << ' ';
56     os << "Area: " << this->Square() << ' ';
57     os << "ID: " << id;
58     os << std::endl;
59 }
60
61 int Hexagon::getID() const {
62     return id;
63 }
64
65 void Hexagon::serialize(std::ostream& os) const {
66     os << 'H' << ' ';
67     for(const auto& p : points) {
68         os << p.x << ' ' << p.y << ' ';
69     }
70     os << std::endl;
71 }

```

## 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 "document.hpp"
6 #include "figure.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) {this->
        document = document;};
20     void exec() override;
21     void undo() override;
```

```

22 };
23
24 class RemoveCommand : public Command {
25 public:
26     RemoveCommand(std::shared_ptr<Document> document, int id) :
27         id(id), position(-1), figure(nullptr) {this->document =
        document;};
28     void exec() override;
29     void undo() override;
30 private:
31     int id;
32     int position;
33     std::shared_ptr<Figure> figure;
34 };

```

## command.cpp

```

1 #include "command.hpp"
2
3
4 void InsertCommand::exec() {
5     document->insert();
6 }
7
8 void InsertCommand::undo() {
9     document->popBack();
10 }
11
12
13 void RemoveCommand::exec() {
14     try {
15         figure = document->getFigure(id);
16         position = document->getPosition(id);
17     } catch(std::exception& e) {
18         std::cout << e.what() << std::endl;
19         return;
20     }
21     document->remove(id);
22 }
23
24 void RemoveCommand::undo() {
25     document->insert(position, figure);
26 }

```

## 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     void createDocument();
13     void insert();

```

```

14     void remove(int id);
15     void saveDocument(const std::string& filename);
16     void loadDocument(const std::string& filename);
17     void undo();
18     void print();
19 private:
20     std::shared_ptr<Document> document;
21     std::stack<std::shared_ptr<Command>> commandStack;
22 };

```

## editor.cpp

```

1  #include "editor.hpp"
2
3  void Editor::createDocument() {
4      document = std::make_shared<Document>();
5      while(!commandStack.empty()) {
6          commandStack.pop();
7      }
8  }
9
10 void Editor::insert() {
11     std::shared_ptr<Command> command = std::shared_ptr<Command>(
12     new InsertCommand(document));
13     command->exec();
14     commandStack.push(command);
15 }
16
17 void Editor::remove(int id) {
18     try {
19         std::shared_ptr<Command> command = std::shared_ptr<Command>
20         >(new RemoveCommand(document, id));
21         command->exec();
22         commandStack.push(command);
23     } catch(std::exception& e) {
24         std::cout << e.what() << std::endl;
25     }
26 }
27
28 void Editor::saveDocument(const std::string& filename) {
29     document->save(filename);
30 }
31
32 void Editor::loadDocument(const std::string& filename) {
33     createDocument();
34     document->load(filename);
35 }
36
37 void Editor::undo() {
38     if(commandStack.empty()) {
39         throw std::logic_error("Nothing to undo");
40     }
41     std::shared_ptr<Command> command = commandStack.top();
42     command->undo();
43     commandStack.pop();
44 }
45
46 void Editor::print() {
47     document->print();

```

46 }

## factory.hpp

```
1 #pragma once
2
3 #include <memory>
4 #include <string>
5
6 #include "figure.hpp"
7 #include "rhombus.hpp"
8 #include "pentagon.hpp"
9 #include "hexagon.hpp"
10
11 class Factory {
12 public:
13     std::shared_ptr<Figure> createFigure(int id);
14 };
```

## factory.cpp

```
1 #include "factory.hpp"
2
3 std::shared_ptr<Figure> Factory::createFigure(int id) {
4     std::string figureType;
5     std::cin >> figureType;
6     std::shared_ptr<Figure> figure;
7     if(figureType == "R") {
8         try {
9             figure = std::make_shared<Rhombus>(Rhombus{std::cin,
10 id});
11         } catch(std::exception& e) {
12             std::cout << e.what() << std::endl;
13         }
14     } else if(figureType == "P") {
15         figure = std::make_shared<Pentagon>(Pentagon{std::cin, id
16 });
17     } else if(figureType == "H") {
18         figure = std::make_shared<Hexagon>(Hexagon{std::cin, id});
19     } else {
20         std::cout << "Unknown figure" << std::endl;
21     }
22     return figure;
23 }
```

## 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
```

```

12 class Document {
13 friend class Command;
14 public:
15     Document() : currentFigureID(0) {};
16     void newDocument();
17     void save(const std::string& fileName);
18     void load(const std::string& fileName);
19     void print();
20     void insert();
21     void insert(unsigned position, std::shared_ptr<Figure> figure)
        ;
22     void remove(int id);
23     void popBack();
24     std::shared_ptr<Figure> getFigure(int id);
25     int getPosition(int id);
26 private:
27     int currentFigureID;
28     std::vector<std::shared_ptr<Figure>> content;
29     Factory factory;
30     void serialize(const std::string& fileName);
31     void deserialize(const std::string& fileName);
32 };

```

## document.cpp

```

1 #include "document.hpp"
2
3
4 void Document::newDocument() {
5     content.clear();
6     currentFigureID = 0;
7 }
8
9 void Document::save(const std::string& fileName) {
10     serialize(fileName);
11 }
12
13 void Document::load(const std::string& fileName) {
14     deserialize(fileName);
15 }
16
17 void Document::print() {
18     for(const auto& figure : content) {
19         figure->Print(std::cout);
20     }
21 }
22
23 void Document::insert() {
24     std::shared_ptr<Figure> figure = this->factory.createFigure(
        currentFigureID);
25     if(figure) {
26         content.push_back(figure);
27         currentFigureID++;
28     }
29 }
30
31 void Document::insert(unsigned position, std::shared_ptr<Figure>
        figure) {
32     auto it = content.begin();

```

```

33     std::advance(it, position);
34     content.insert(it, figure);
35 }
36
37 void Document::remove(int id) {
38     unsigned temp = content.size();
39     auto it = std::remove_if(content.begin(), content.end(), [id](
std::shared_ptr<Figure> f)
40         {return id == f->getID();});
41     content.erase(it, content.end());
42     if(temp == content.size()) {
43         throw std::invalid_argument("Figure this such ID doesn't
exist");
44     }
45 }
46
47 void Document::popBack() {
48     if(!content.size()) {
49         throw std::logic_error("Document is empty");
50     }
51     content.pop_back();
52 }
53
54 std::shared_ptr<Figure> Document::getFigure(int id) {
55     for(const auto& figure : content) {
56         if(id == figure->getID()) {
57             return figure;
58         }
59     }
60     throw std::invalid_argument("1:No figure this such ID");
61 }
62
63 int Document::getPosition(int id) {
64     int n = content.size();
65     for(int i = 0; i < n; ++i) {
66         if(id == content[i]->getID()) {
67             return i;
68         }
69     }
70     throw std::invalid_argument("2:No figure with such ID");
71 }
72
73 void Document::serialize(const std::string& fileName) {
74     std::ofstream os(fileName, std::ios::trunc);
75     if(!os) {
76         throw std::runtime_error("Couldn't open file");
77     }
78     os << content.size() << std::endl;
79     for(const auto& figure : content) {
80         figure->serialize(os);
81     }
82 }
83
84 void Document::deserialize(const std::string& fileName) {
85     std::ifstream is(fileName);
86     if(!is) {
87         throw std::runtime_error("Couldn't open file");
88     }
89     this->newDocument();

```

```

90     int numberOfFigures;
91     is >> numberOfFigures;
92     while(numberOfFigures-->0) {
93         this->insert();
94     }
95 }

```

## 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" <<
9      std::endl;
10     std::cout << "load <path to file> - loads document from file"
11     << std::endl;
12     std::cout << "add R/P/H <coordinates> - adds Rhombus/Pentagon/
13     Hexagon to the document" << std::endl;
14     std::cout << "remove <Figure ID> - removes figure with given
15     ID if it is present" << std::endl;
16     std::cout << "undo - undo last action" << std::endl;
17     std::cout << "print - prints information about all figures
18     from document" << std::endl;
19     std::cout << "help - do I really need to explain what help
20     does?" << std::endl;
21     std::cout << "exit - exit editor" << std::endl;
22 }
23
24 int main() {
25     int id;
26     std::string command;
27     std::string filepath;
28     std::string figureType;
29     Editor e;
30     help();
31     while(std::cin >> command) {
32         if(command == "new") {
33             e.createDocument();
34         } else if(command == "save") {
35             std::cin >> filepath;
36             try {
37                 e.saveDocument(filepath);
38             } catch(std::exception& e) {
39                 std::cout << e.what() << std::endl;
40             }
41         } else if(command == "load") {
42             std::cin >> filepath;
43             try {
44                 e.loadDocument(filepath);
45             } catch(std::exception& e) {
46                 std::cout << e.what() << std::endl;
47             }
48         } else if(command == "add") {
49             e.insert();
50         } else if(command == "remove") {

```

```

45         std::cin >> id;
46         try {
47             e.remove(id);
48         } catch(std::exception& e) {
49             std::cout << e.what() << std::endl;
50         }
51     } else if(command == "undo") {
52         try {
53             e.undo();
54         } catch(std::exception& e) {
55             std::cout << e.what() << std::endl;
56         }
57     } else if(command == "print") {
58         e.print();
59     } else if(command == "help") {
60         help();
61     } else if(command == "exit") {
62         break;
63     } else {
64         std::cout << "Unknown figure" << std::endl;
65     }
66 }
67 return 0;
68 }

```

## 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    editor.cpp
11    factory.cpp
12    hexagon.cpp
13    command.cpp
14    document.cpp)
15
16 set_property(TARGET lab7 PROPERTY CXX_STANDARD 17)
17
18 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.