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

```
#pragma once

#include<iostream>

#include "point.hpp"

enum class Figures {Rhombus, Pentagon, Hexagon};

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

rhombus.hpp

```
#pragma once

#include <array>

#include "figure.hpp"
#include "point.hpp"

class Rhombus : public Figure {
public:
    Rhombus(Point* p, int id);
    Rhombus(std::istream& is);
    Point Center() const override;
    double Square() const override;
    void Print(std::ostream& os) const override;
    int getID() const override;
    void serialize(std::ostream& os) const override;
private:
    std::array<Point, 4> points;
    double smallerDiagonal, biggerDiagonal;
    int id;
};
```

rhombus.cpp

```
#include "rhombus.hpp"

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(Point* p, int id) {
    Point p1, p2, p3, p4;
    p1 = p[0];
    p2 = p[1];
    p3 = p[2];
    p4 = p[3];
    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[0] = p1;
    points[1] = p2;
    points[2] = p3;
    points[3] = p4;
    this->id = id;
}

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

```cpp
        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[0] = p1;
    points[1] = p2;
    points[2] = p3;
    points[3] = 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;
}

void Rhombus::Print(std::ostream& os) const {
    os << "Rhombus: ";
    for(const auto& p : points) {
        os << p << ' ';
    }
    os << "Center: " << this->Center() << ' ';
    os << "Area: " << this->Square() << ' ';
    os << "ID: " << id;
    os << std::endl;
}

int Rhombus::getID() const {
    return id;
}

void Rhombus::serialize(std::ostream& os) const {
    os << points.size() << ' ';
    for(const auto& p : points) {
        os << p.x << ' ' << p.y << ' ';
    }
    os << std::endl;
}
```

pentagon.hpp

```cpp
#pragma once

#include <iostream>
#include <array>

#include "figure.hpp"
#include "point.hpp"

class Pentagon : public Figure {
public:
    Pentagon(Point* p, int id);
    Pentagon(std::istream& is);
    Point Center() const override;
    double Square() const override;
    void Print(std::ostream& os) const override;
    int getID() const override;
    void serialize(std::ostream& os) const override;
private:
    std::array<Point, 5> points;
    int id;
};
```

pentagon.cpp

```cpp
#include <cmath>

#include "pentagon.hpp"

Pentagon::Pentagon(Point* p, int id) {
    for(int i = 0; i < 5; ++i) {
        points[i] = p[i];
    }
    this->id = id;
}

Pentagon::Pentagon(std::istream& is) {
    is >> points[0] >> points[1] >> points[2] >> points[3] >>
   points[4];
}

Point Pentagon::Center() const {
    Point insideFigure{0, 0};
    Point result{0, 0};
    double square = this->Square();
    for(unsigned i = 0; i < points.size(); ++i) {
        insideFigure.x += points[i].x;
        insideFigure.y += points[i].y;
    }
    insideFigure.x /= points.size();
    insideFigure.y /= points.size();
    for(unsigned i = 0; i < points.size(); ++i) {
        double tempSquare = triangleSquare(points[i], points[(i + 1) % points.size()],
                insideFigure);
        result.x += tempSquare * (points[i].x + points[(i + 1) % points.size()].x
                + insideFigure.x) / 3.0;
        result.y += tempSquare * (points[i].y + points[(i + 1) % points.size()].y
                + insideFigure.y) / 3.0;
    }
    result.x /= square;
    result.y /= square;
    return result;
}

double Pentagon::Square() const {
    double result = 0;
    for(unsigned i = 0; i < points.size(); ++i) {
        Point p1 = i ? points[i - 1] : points[points.size() - 1];
        Point p2 = points[i];
        result += (p1.x - p2.x) * (p1.y + p2.y);
    }
    return fabs(result) / 2.0;
}

void Pentagon::Print(std::ostream& os) const {
    os << "Pentagon: ";
    for(const auto& p : points) {
        os << p << ' ';
    }
    os << "Center: " << this->Center() << ' ';
    os << "Area: " << this->Square() << ' ';
    os << "ID: " << id;
    os << std::endl;
}

int Pentagon::getID() const {
    return id;
}

void Pentagon::serialize(std::ostream& os) const {
    os << points.size() << ' ';
    for(const auto& p : points) {
        os << p.x << ' ' << p.y << ' ';
    }
    os << std::endl;
}
```

hexagon.hpp

```cpp
#pragma once

#include <iostream>
#include <array>

#include "figure.hpp"
#include "point.hpp"

class Hexagon : public Figure {
public:
    Hexagon(Point* p, int id);
    Hexagon(std::istream& is);
    Point Center() const override;
    double Square() const override;
    void Print(std::ostream& os) const override;
    int getID() const override;
    void serialize(std::ostream& os) const override;
private:
    std::array<Point, 6> points;
    int id;
};
```

hexagon.cpp

```cpp
#include <cmath>

#include "hexagon.hpp"

Hexagon::Hexagon(Point* p, int id) {
    for(int i = 0; i < 6; ++i) {
        points[i] = p[i];
    }
    this->id = id;
}

Hexagon::Hexagon(std::istream& is) {
    is >> points[0] >> points[1] >> points[2] >> points[3] >>
   points[4] >> points[5];
}

Point Hexagon::Center() const {
    Point insideFigure{0, 0};
    Point result{0, 0};
    double square = this->Square();
    for(unsigned i = 0; i < points.size(); ++i) {
        insideFigure.x += points[i].x;
        insideFigure.y += points[i].y;
    }
    insideFigure.x /= points.size();
    insideFigure.y /= points.size();
    for(unsigned i = 0; i < points.size(); ++i) {
        double tempSquare = triangleSquare(points[i], points[(i +
   1) % points.size()],
                insideFigure);
        result.x += tempSquare * (points[i].x + points[(i + 1) %
   points.size()].x
                + insideFigure.x) / 3.0;
        result.y += tempSquare * (points[i].y + points[(i + 1) %
   points.size()].y
```

```cpp
                    + insideFigure.y) / 3.0;
    }
    result.x /= square;
    result.y /= square;
    return result;
}

double Hexagon::Square() const {
    double result = 0;
    for(unsigned i = 0; i < points.size(); ++i) {
        Point p1 = i ? points[i - 1] : points[points.size() - 1];
        Point p2 = points[i];
        result += (p1.x - p2.x) * (p1.y + p2.y);
    }
    return fabs(result) / 2.0;
}

void Hexagon::Print(std::ostream& os) const {
    os << "Hexagon:";
    for(const auto& p : points) {
        os << p << ' ';
    }
    os << "Center: " << this->Center() << ' ';
    os << "Area: " << this->Square() << ' ';
    os << "ID: " << id;
    os << std::endl;
}

int Hexagon::getID() const {
    return id;
}

void Hexagon::serialize(std::ostream& os) const {
    os << points.size() << ' ';
    for(const auto& p : points) {
        os << p.x << ' ' << p.y << ' ';
    }
    os << std::endl;
}
```

point.hpp

```cpp
#pragma once

#include <iostream>

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);
double triangleSquare(const Point& p1, const Point& p2, const
    Point& p3);
```

point.cpp

```cpp
#include <iostream>
#include <cmath>
#include <iomanip>

#include "point.hpp"

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

double triangleSquare(const Point& p1, const Point& p2, const Point& p3) {
    return 0.5 * fabs((p1.x - p3.x) * (p2.y - p3.y) - (p2.x - p3.x) * (p1.y - p3.y));
}

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;
}
```

command.hpp

```cpp
#pragma once

#include <memory>

#include "figure.hpp"
#include "document.hpp"

class Command {
public:
    virtual void exec() = 0;
    virtual void undo() = 0;
    virtual ~Command() = default;
protected:
    std::shared_ptr<Document> document;
};

class InsertCommand : public Command {
public:
    InsertCommand(std::shared_ptr<Document> document, Figures figure,
            Point* points) : figure(figure), points(points) {this->document = document;};

    void exec() override {
       document->insert(figure, points);
    }

    void undo() override {
        document->popBack();
    }
private:
    Figures figure;
    Point* points;
};

class RemoveCommand : public Command {
public:
    RemoveCommand(std::shared_ptr<Document> document, int id) :
        id(id), position(-1), figure(nullptr) {this->document =
   document;};

    void exec() override {
        try {
            figure = document->getFigure(id);
            position = document->getPosition(id);
        } catch(std::exception& e) {
            std::cout << e.what() << std::endl;
            return;
        }
        document->remove(id);
    }

    void undo() override {
        document->insert(position, figure);
    }
private:
    int id;
    int position;
    std::shared_ptr<Figure> figure;
};
```

editor.hpp

```
#pragma once

#include <stack>

#include "command.hpp"
#include "document.hpp"
#include "figure.hpp"

class Editor {
public:
    Editor() : document(nullptr) {};

    void createDocument() {
        document = std::make_shared<Document>();
    }

    void insert(Figures figure, Point* points) {
        std::shared_ptr<Command> command = std::shared_ptr<Command
   >(new InsertCommand(document, figure, points));
```

```
        command->exec();
        commandStack.push(command);
    }

    void remove(int id) {
        try {
            std::shared_ptr<Command> command = std::shared_ptr<
Command>(new RemoveCommand(document, id));
            command->exec();
            commandStack.push(command);
        } catch(std::exception& e) {
            std::cout << e.what() << std::endl;
        }
    }

    void saveDocument(const std::string& filename) {
        document->save(filename);
    }

    void loadDocument(const std::string& filename) {
        createDocument();
        document->load(filename);
    }

    void undo() {
        if(commandStack.empty()) {
            throw std::logic_error("Nothing to undo");
        }
        std::shared_ptr<Command> command = commandStack.top();
        command->undo();
        commandStack.pop();
    }

    void print() {
        document->print();
    }
private:
    std::shared_ptr<Document> document;
    std::stack<std::shared_ptr<Command>> commandStack;
};
```

factory.hpp

```
#pragma once

#include <memory>

#include "figure.hpp"
#include "rhombus.hpp"
#include "pentagon.hpp"
#include "hexagon.hpp"

class Factory {
public:
    std::shared_ptr<Figure> createFigure(Figures type, Point* p,
int id) {
        if(type == Figures::Rhombus) {
            try {
                Rhombus{p, id};
```

```cpp
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
                return nullptr;
            }
            return std::make_shared<Rhombus>(Rhombus{p, id});
        } else if(type == Figures::Pentagon) {
            return std::make_shared<Pentagon>(Pentagon{p, id});
        } else if(type == Figures::Hexagon) {
            return std::make_shared<Hexagon>(Hexagon{p, id});
        } else {
            return nullptr;
        }
    }
};
```

document.hpp

```cpp
#pragma once

#include <vector>
#include <string>
#include <algorithm>
#include <fstream>
#include <stack>

#include "figure.hpp"
#include "factory.hpp"
#include "command.hpp"

class Document {
friend class Command;
public:
    Document() : currentFigureID(0) {};

    void newDocument() {
        content.clear();
        currentFigureID = 0;
    }

    void save(const std::string& fileName) {
        serialize(fileName);
    }

    void load(const std::string& fileName) {
        deserialize(fileName);
    }

    void print() {
        for(const auto& figure : content) {
            figure->Print(std::cout);
        }
    }

    void insert(Figures type, Point* points) {
        if(type == Figures::Rhombus) {
            try {
                Rhombus{points, currentFigureID};
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
                return;
            }
            content.push_back(factory.createFigure(Figures::Rhombus, points, currentFigureID));
        } else if(type == Figures::Pentagon) {
            content.push_back(factory.createFigure(Figures::Pentagon, points, currentFigureID));
        } else if(type == Figures::Hexagon) {
            content.push_back(factory.createFigure(Figures::Hexagon, points, currentFigureID));
        }
        currentFigureID++;
    }

    void insert(unsigned position, std::shared_ptr<Figure> figure) {
        auto it = content.begin();
        std::advance(it, position);
        content.insert(it, figure);
    }

    void remove(int id) {
        unsigned temp = content.size();
        auto it = std::remove_if(content.begin(), content.end(), [id](std::shared_ptr<Figure> f)
                {return id == f->getID();});
        content.erase(it, content.end());
        if(temp == content.size()) {
            throw std::invalid_argument("Figure this such ID doesn't exist");
        }
    }

    void popBack() {
        if(!content.size()) {
            throw std::logic_error("Document is empty");
        }
        content.pop_back();
    }

    std::shared_ptr<Figure> getFigure(int id) {
        for(const auto& figure : content) {
            if(id == figure->getID()) {
                return figure;
            }
        }
        throw std::invalid_argument("1:No figure this such ID");
    }

    int getPosition(int id) {
        int n = content.size();
        for(int i = 0; i < n; ++i) {
            if(id == content[i]->getID()) {
                return i;
            }
        }
        throw std::invalid_argument("2:No figure with such ID");
    }
private:
    int currentFigureID;
    std::vector<std::shared_ptr<Figure>> content;
    Factory factory;
    void serialize(const std::string& fileName) {
        std::ofstream os(fileName, std::ios::trunc);
        if(!os) {
            throw std::runtime_error("Couldn't open file");
        }
        os << content.size() << std::endl;
        for(const auto& figure : content) {
            figure->serialize(os);
        }
    }

    void deserialize(const std::string& fileName) {
        std::ifstream is(fileName);
        if(!is) {
            throw std::runtime_error("Couldn't open file");
        }
        this->newDocument();
        int numberOfFigures;
        is >> numberOfFigures;
        int numberOfPoints;
        Point p;
        while(numberOfFigures--) {
            is >> numberOfPoints;
            Point* points = new Point[numberOfPoints];
            for(int i = 0; i < numberOfPoints; ++i) {
                is >> p;
                points[i] = p;
            }
            if(numberOfPoints == 4) {
                content.push_back(factory.createFigure(Figures::Rhombus, points, currentFigureID));
            } else if(numberOfPoints == 5) {
                content.push_back(factory.createFigure(Figures::Pentagon, points, currentFigureID));
            } else if(numberOfPoints == 6) {
                content.push_back(factory.createFigure(Figures::Hexagon, points, currentFigureID));
            }
            this->currentFigureID++;
            delete[] points;
        }
    }
};
```

main.cpp

```
#include <iostream>
#include <string>

#include "editor.hpp"

void help() {
    std::cout << "new - Creates new document" << std::endl;
    std::cout << "save <path to file> - saves document to file" << std::endl;
    std::cout << "load <path to file> - loads document from file"
```

```cpp
    << std::endl;
    std::cout << "add R/P/H <coordinates> - adds Rhombus/Pentagon/Hexagon to the document" << std::endl;
    std::cout << "remove <Figure ID> - removes figure with given ID if it is present" << std::endl;
    std::cout << "undo - undo last action" << std::endl;
    std::cout << "print - prints information about all figures from document" << std::endl;
    std::cout << "help - do I really need to explain what help does?" << std::endl;
    std::cout << "exit - exit editor" << std::endl;
}

int main() {
    int id;
    std::string command;
    std::string filepath;
    std::string figureType;
    Editor e;
    help();
    while(std::cin >> command) {
        if(command == "new") {
            e.createDocument();
        } else if(command == "save") {
            std::cin >> filepath;
            try {
                e.saveDocument(filepath);
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
            }
        } else if(command == "load") {
            std::cin >> filepath;
            try {
                e.loadDocument(filepath);
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
            }
        } else if(command == "add") {
            std::cin >> figureType;
            if(figureType == "R") {
                Point* p = new Point[4];
                for(int i = 0; i < 4; ++i) {
                    std::cin >> p[i];
                }
                try {
                    e.insert(Figures::Rhombus, p);
                } catch(std::exception& e) {
                    std::cout << e.what() << std::endl;
                }
                delete[] p;
            } else if(figureType == "P") {
                Point* p = new Point[5];
                for(int i = 0; i < 5; ++i) {
                    std::cin >> p[i];
                }
                e.insert(Figures::Pentagon, p);
                delete[] p;
            } else if(figureType == "H") {
                Point* p = new Point[6];
                for(int i = 0; i < 6; ++i) {
                    std::cin >> p[i];
                }
                e.insert(Figures::Hexagon, p);
                delete[] p;
            } else {
                std::cout << "Unknown figure" << std::endl;
            }
        } else if(command == "remove") {
            std::cin >> id;
            try {
                e.remove(id);
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
            }
        } else if(command == "undo") {
            try {
                e.undo();
            } catch(std::exception& e) {
                std::cout << e.what() << std::endl;
            }
        } else if(command == "print") {
            e.print();
        } else if(command == "help") {
            help();
        } else if(command == "exit") {
            break;
        } else {
            std::cout << "Unknown figure" << std::endl;
        }
    }
    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)

project(lab7)

add_executable(lab7
    main.cpp
    point.cpp
    rhombus.cpp
    pentagon.cpp
    hexagon.cpp)

set_property(TARGET lab7 PROPERTY CXX_STANDARD 17)

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