Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа**

**по курсу «Объектно-ориентированное программирование»**

**III Семестр**

**Задание 7  
Вариант 8**

**Проектирование структуры классов**

|  |  |
| --- | --- |
| Студент: | Жерлыгин М.А |
| Группа: | М8О-208Б-18 |
| Преподаватель: | Журавлев А.А. |
| Оценка: |  |
| Дата: |  |
| Подпись: |  |

# 1. Код программы на языке С++

**point.h**

#ifndef D\_POINT\_H\_

#define D\_POINT\_H\_

#include <istream>

#include <ostream>

class Point {

public:

double x, y;

Point();

Point(double a, double b);

Point& operator=(const Point& other);

Point operator+(const Point& other);

Point operator-(const Point& other);

Point operator/(const double num);

~Point() = default;

friend std::istream& operator>> (std::istream& is, Point& p);

friend std::ostream& operator<< (std::ostream& os, const Point& p);

};

#endif //D\_POINT\_H\_

**point.cpp**

#include "point.h"

#include <cmath>

Point::Point(): x(0), y(0) {

}

Point::Point(double a, double b): x(a), y(b) {

}

Point& Point::operator=(const Point& other) {

this->x = other.x;

this->y = other.y;

return \*this;

}

Point Point::operator+(const Point& other) {

Point result;

result.x = this->x + other.x;

result.y = this->y + other.y;

return result;

}

Point Point::operator-(const Point& other) {

Point result;

result.x = this->x - other.x;

result.y = this->y - other.y;

return result;

}

Point Point::operator/(const double num) {

Point result;

result.x = this->x / num;

result.y = this->y / num;

return result;

}

std::istream& operator>> (std::istream& is, Point& p) {

return is >> p.x >> p.y;

}

std::ostream& operator<< (std::ostream& os, const Point& p) {

return os << "(" << p.x << ", " << p.y << ")" << std::endl;

}

**figure.h**

#ifndef FIGURE\_H\_

#define FIGURE\_H\_

#include <fstream>

#include <map>

#include <memory>

#include "point.h"

namespace figure {

class Figure {

public:

virtual Point center() const = 0;

virtual double area() const = 0;

virtual void print(std::ostream& os) const = 0;

virtual void save(std::ofstream& os) const = 0;

virtual void load(std::ifstream& is) = 0;

virtual uint32\_t get\_ID() const = 0;

virtual ~Figure() = default;

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

};

}

enum figure\_t {

OCTAGON,

TRIANGLE,

SQUARE

};

class Fact\_Interface {

public:

virtual std::shared\_ptr<figure::Figure> Create\_figure() const = 0;

virtual std::shared\_ptr<figure::Figure> Create\_figure(uint32\_t id, std::istream& is) const = 0;

};

#endif // FIGURE\_H\_

**figure.cpp**

#include "figure.h"

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

f.print(os);

return os;

}

**octagon.h**

#ifndef OCTAGON\_H\_

#define OCTAGON\_H\_

#include "figure.h"

namespace figure {

class Octagon : public Figure {

private:

Point coordinate[8];

uint32\_t id\_;

public:

Octagon();

Octagon(uint32\_t id, std::istream& is);

Point center() const override;

double area() const override;

void print(std::ostream& os) const override;

uint32\_t get\_ID() const override;

void save(std::ofstream& os) const override;

void load(std::ifstream& is) override;

};

}

class Oct\_factory: public Fact\_Interface {

public:

std::shared\_ptr<figure::Figure> Create\_figure() const override;

std::shared\_ptr<figure::Figure> Create\_figure(uint32\_t id, std::istream& is) const override;

};

#endif // OCTAGON\_H\_

**octagon.cpp**

#include <iostream>

#include <cmath>

#include "octagon.h"

namespace figure {

Octagon::Octagon(): id\_(0) {

for(int i = 0; i < 8; i++) {

coordinate[i].x = 0.0;

coordinate[i].y = 0.0;

}

}

Octagon::Octagon(uint32\_t id, std::istream& is): id\_(id) {

for(int i = 0; i < 8; i++) {

is >> coordinate[i];

}

}

double Octagon::area() const {

double result = 0;

for(int i = 0; i < 7; i++) {

result += (coordinate[i].x \* coordinate[i+1].y) - (coordinate[i+1].x \* coordinate[i].y);

}

result = std::abs(result + (coordinate[7].x \* coordinate[0].y) - (coordinate[0].x \* coordinate[7].y));

return result / 2.0;

}

Point Octagon::center() const {

Point result;

for(int i = 0; i < 8; i++) {

result = result + coordinate[i];

}

return result / 8.0;

}

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

os << "=================================\n";

os << "id - " << id\_ << "\nFigure - Octagon" << "\nArea: " << area() << "\nCenter: " << center();

std::cout << "Octagon coordinates:" << std::endl;

os << this->coordinate[0];

os << this->coordinate[1];

os << this->coordinate[2];

os << this->coordinate[3];

os << this->coordinate[4];

os << this->coordinate[5];

os << this->coordinate[6];

os << this->coordinate[7];

}

uint32\_t Octagon::get\_ID() const {

return id\_;

}

void Octagon::save(std::ofstream& os) const {

figure\_t t = OCTAGON;

os.write(reinterpret\_cast<char\*>(&t), sizeof(t));

os.write((char\*)(&id\_), sizeof(id\_));

for(int i = 0; i <= 7 ; i++) {

os << coordinate[i].x << ' ' << coordinate[i].y;

if(i != 7) {

os << "\t";

}

}

}

void Octagon::load(std::ifstream& is) {

is.read((char\*)(&id\_), sizeof(id\_));

for(int i = 0; i <= 7; i++) {

is >> coordinate[i].x >> coordinate[i].y;

}

}

}// end of namespace

std::shared\_ptr<figure::Figure> Oct\_factory::Create\_figure() const {

return std::shared\_ptr<figure::Figure>(new figure::Octagon());

}

std::shared\_ptr<figure::Figure> Oct\_factory::Create\_figure(uint32\_t id, std::istream& is) const {

return std::shared\_ptr<figure::Figure>(new figure::Octagon(id, is));

}

**triangle.h**

#ifndef D\_TRIANGLE\_H\_

#define D\_TRIANGLE\_H\_

#include "figure.h"

namespace figure {

class Triangle : public Figure {

public:

Point coordinate[3];

uint32\_t id\_;

Triangle();

Triangle(uint32\_t id, std::istream& is);

Point center() const override;

double area() const override;

void print(std::ostream& os) const override;

uint32\_t get\_ID() const override;

void save(std::ofstream& os) const override;

void load(std::ifstream& is) override;

};

} // end of namespace

class Tri\_factory: public Fact\_Interface {

public:

std::shared\_ptr<figure::Figure> Create\_figure() const override;

std::shared\_ptr<figure::Figure> Create\_figure(uint32\_t id, std::istream& is) const override;

};

#endif //D\_TRIANGLE\_H\_

**triangle.cpp**

#include <iostream>

#include <cmath>

#include "triangle.h"

namespace figure {

Triangle::Triangle(): id\_(0) {

//coordinate = new Point[3];

for(int i = 0; i < 3; i++) {

coordinate[i].x = 0.0;

coordinate[i].y = 0.0;

}

}

Triangle::Triangle(uint32\_t id, std::istream& is): id\_(id) {

//coordinate = new Point[3];

for(int i = 0; i < 3; i++) {

is >> coordinate[i];

}

double AB, BC, AC;

AB = sqrt(pow(coordinate[1].x - coordinate[0].x, 2) + pow(coordinate[1].y - coordinate[0].y, 2));

BC = sqrt(pow(coordinate[2].x - coordinate[1].x, 2) + pow(coordinate[2].y - coordinate[1].y, 2));

AC = sqrt(pow(coordinate[2].x - coordinate[0].x, 2) + pow(coordinate[2].y - coordinate[0].y, 2));

if(AB + BC <= AC || AB + AC <= BC || BC + AC <= AB) throw std::logic\_error("This is not Triange");

}

Point Triangle::center() const {

Point result;

for(int i = 0; i < 3; i++) {

result = result + coordinate[i];

}

return result / 3.0;

}

double Triangle::area() const {

return fabs(((coordinate[0].x - coordinate[2].x) \* (coordinate[1].y - coordinate[2].y) - (coordinate[1].x - coordinate[2].x) \* (coordinate[0].y - coordinate[2].y)) / 2);

}

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

os << "=================================\n";

os << "id - " << id\_ << "\nFigure - Triangle" << "\nArea: " << area() << "\nCenter: " << center();

std::cout << "Triangle coordinates" << std::endl;

os << Point(coordinate[0].x, coordinate[0].y) << "\n"

<< Point(coordinate[1].x, coordinate[1].y) << "\n"

<< Point(coordinate[2].x, coordinate[2].y) << std::endl;

}

uint32\_t Triangle::get\_ID() const {

return id\_;

}

void Triangle::load(std::ifstream& is) {

is.read((char\*)(&id\_), sizeof(id\_));

for (int i = 0; i < 3; ++i) {

is >> coordinate[i].x >> coordinate[i].y;

}

}

void Triangle::save(std::ofstream& os) const {

figure\_t t = TRIANGLE;

os.write(reinterpret\_cast<char\*>(&t), sizeof(t));

os.write((char\*)(&id\_), sizeof(id\_));

for (int i = 0; i <= 2; ++i) {

os << coordinate[i].x << ' ' << coordinate[i].y;

if (i != 2) os << '\t';

}

}

}// end of namespace

std::shared\_ptr<figure::Figure> Tri\_factory::Create\_figure() const {

return std::shared\_ptr<figure::Figure>(new figure::Triangle());

}

std::shared\_ptr<figure::Figure> Tri\_factory::Create\_figure(uint32\_t id, std::istream& is) const {

return std::shared\_ptr<figure::Figure>(new figure::Triangle(id, is));

}

**square.h**

#ifndef D\_Square\_H\_

#define D\_Square\_H\_

#include "figure.h"

namespace figure {

struct Square : public Figure {

private:

Point coordinate[4];

uint32\_t id\_;

public:

Square();

Square(uint32\_t id, std::istream& is);

Point center() const override;

double area() const override;

void print(std::ostream& os) const override;

void save(std::ofstream& os) const override;

void load(std::ifstream& is) override;

uint32\_t get\_ID() const override;

};

}// end of namespace

class Squ\_factory: public Fact\_Interface {

public:

std::shared\_ptr<figure::Figure> Create\_figure() const override;

std::shared\_ptr<figure::Figure> Create\_figure(uint32\_t id, std::istream& is) const override;

};

#endif // D\_Square\_H\_

**square.cpp**

#include <iostream>

#include "square.h"

#include <cmath>

#include <algorithm>

namespace figure {

Square::Square(): id\_(0) {

for(int i = 0; i < 4; i++) {

coordinate[i].x = 0.0;

coordinate[i].y = 0.0;

}

}

Square::Square(uint32\_t id, std::istream& is): id\_(id) {

double a, b, c, d;

is >> coordinate[0];

is >> coordinate[1];

is >> coordinate[2];

is >> coordinate[3];

a = sqrt((coordinate[1].x - coordinate[0].x)\*(coordinate[1].x - coordinate[0].x) + (coordinate[1].y - coordinate[0].y)\*(coordinate[1].y - coordinate[0].y));

b = sqrt((coordinate[2].x - coordinate[1].x)\*(coordinate[2].x - coordinate[1].x) + (coordinate[2].y - coordinate[1].y)\*(coordinate[2].y - coordinate[1].y));

c = sqrt((coordinate[3].x - coordinate[2].x)\*(coordinate[3].x - coordinate[2].x) + (coordinate[3].y - coordinate[2].y)\*(coordinate[3].y - coordinate[2].y));

d = sqrt((coordinate[0].x - coordinate[3].x)\*(coordinate[0].x - coordinate[3].x) + (coordinate[0].y - coordinate[3].y)\*(coordinate[0].y - coordinate[3].y));

double d1, d2;

d1 = sqrt((coordinate[1].x - coordinate[3].x)\*(coordinate[1].x - coordinate[3].x) + (coordinate[1].y - coordinate[3].y)\*(coordinate[2].y - coordinate[3].y));

d2 = sqrt((coordinate[2].x - coordinate[0].x)\*(coordinate[2].x - coordinate[0].x) + (coordinate[2].y - coordinate[0].y)\*(coordinate[2].y - coordinate[0].y));

double ABC = (a \* a + b \* b - d2 \* d2) / (2 \* a \* b);

double BCD = (b \* b + c \* c - d1 \* d1) / (2 \* b \* c);

double CDA = (c \* c + d \* d - d1 \* d1) / (2 \* c \* d);

double DAB = (d \* d + a \* a - d2 \* d2) / (2 \* d \* a);

if(ABC != BCD || ABC != CDA || ABC != DAB || a!=b || a!=c || a!=d) throw std::logic\_error("It`s not a square");

//if((coordinate[1].x - coordinate[2].x != coordinate[1].y - coordinate[2].y) || (coordinate[1].x == coordinate[2].x && coordinate[1].y == coordinate[2].y)) throw std::logic\_error("This are incorrect coordinates");

//if(coordinate[1].x - coordinate[2].x != coordinate[1].y - coordinate[2].y) throw std::logic\_error("This is not square");

}

Point Square::center() const {

return Point((coordinate[0].x + coordinate[2].x) / 2, (coordinate[0].y + coordinate[2].y) / 2);

}

double Square::area() const {

//const double dx = coordinate[1].x - coordinate[3].x;

//const double dy = coordinate[1].y - coordinate[3].y;

//return std::abs(dx \* dy);

return pow(sqrt((coordinate[0].x - coordinate[3].x)\*(coordinate[0].x - coordinate[3].x) + (coordinate[0].y - coordinate[3].y)\*(coordinate[0].y - coordinate[3].y)), 2);

}

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

os << "=================================\n";

os << "id - " << id\_ << "\nFigure - Square" << "\nArea: " << area() << "\nCenter: " << center();

std::cout << "Square coordinates:" << std::endl;

os << coordinate[0] << std::endl;

os << coordinate[1] << std::endl;

os << coordinate[2] << std::endl;

os << coordinate[3] << std::endl;

}

void Square::save(std::ofstream& os) const {

figure\_t t = SQUARE;

os.write(reinterpret\_cast<char\*>(&t), sizeof(t));

os.write((char\*)(&id\_), sizeof(id\_));

for (int i = 0; i < 2; ++i) {

os << coordinate[i].x << ' ' << coordinate[i].y;

if (i != 1) os << '\t';

}

}

void Square::load(std::ifstream& is) {

is.read((char\*)(&id\_), sizeof(id\_));

for (int i = 0; i < 2; ++i) {

is >> coordinate[i].x >> coordinate[i].y;

}

}

uint32\_t Square::get\_ID() const {

return id\_;

}

}// end of namespace

std::shared\_ptr<figure::Figure> Squ\_factory::Create\_figure() const {

return std::shared\_ptr<figure::Figure>(new figure::Square());

}

std::shared\_ptr<figure::Figure> Squ\_factory::Create\_figure(uint32\_t id, std::istream& is) const {

return std::shared\_ptr<figure::Figure>(new figure::Square(id, is));

}

**interface.h**

#ifndef INTERFACE\_H\_

#define INTERFACE\_H\_

#include <stack>

#include "doc.h"

#include "com.h"

class Editor {

private:

    std::stack<std::shared\_ptr<Command>> History\_;

    std::shared\_ptr<document\_class::Document> document\_;

public:

    Editor(): document\_(nullptr), History\_() {}

    ~Editor() = default;

    void Create\_document(const std::string& name) {

        document\_ = std::make\_shared<document\_class::Document>(name);

        while(!History\_.empty())

            History\_.pop();

    }

    void Save\_document(const std::string& filename) {

        document\_->Save(filename);

    }

    std::shared\_ptr<document\_class::Document> get\_document() {

        return document\_;

    }

    void Load\_document(const std::string& filename) {

        document\_ = std::make\_shared<document\_class::Document>("NoName");

        document\_->Load(filename);

        while(!History\_.empty())

            History\_.pop();

    }

    void Insert\_figure(figure\_t type, std::istream& is) {

        std::shared\_ptr<Command> command = std::shared\_ptr<Command>(new Command\_insert(type, is));

        command->Set\_Doc(document\_);

        command->Run();

        History\_.push(command);

    }

    void Remove\_figure(uint32\_t id) {

        std::shared\_ptr<Command> command = std::shared\_ptr<Command>(new Command\_remove(id));

        command->Set\_Doc(document\_);

        command->Run();

        History\_.push(command);

    }

    void Print\_document() {

        document\_->Print();

    }

    bool Document\_exist() {

        return document\_ != nullptr;

    }

    void Undo() {

        if (History\_.empty()) {

            std::cout << "History is empty" << std::endl;

        } else {

            std::shared\_ptr<Command> last\_cmd = History\_.top();

            last\_cmd->Abort();

            History\_.pop();

        }

    }

};

#endif

**com.h**

#ifndef COM\_H

#define COM\_H

#include <stack>

#include <utility>

#include "doc.h"

class Command {

protected:

std::shared\_ptr<document\_class::Document> document\_;

public:

virtual ~Command() = default;

virtual void Run() = 0;

virtual void Abort() = 0;

void Set\_Doc(std::shared\_ptr<document\_class::Document> doc) { document\_ = std::move(doc); }

};

class Command\_insert: public Command {

private:

figure\_t fig\_type\_;

std::istream& is\_;

public:

Command\_insert(figure\_t type, std::istream& is): fig\_type\_(type), is\_(is) {}

void Run() override {

document\_->figure\_add(fig\_type\_, is\_);

}

void Abort() override {

document\_->Remove\_last\_figure();

}

};

class Command\_remove : public Command {

private:

uint32\_t id\_;

uint32\_t position\_;

std::shared\_ptr<figure::Figure> figure;

public:

explicit Command\_remove(uint32\_t id): id\_(id), position\_(0), figure(nullptr) {}

void Run() override {

figure = document\_->Get\_figure(id\_);

position\_ = document\_->Get\_position(id\_);

document\_->Remove\_figure(id\_);

}

void Abort() override {

document\_->Insert\_figure(position\_, figure);

}

};

#endif //COM\_H

**doc.h**

#ifndef DOCUMENT\_H

#define DOCUMENT\_H

#include <fstream>

#include <memory>

#include <list>

#include "figure.h"

#include "octagon.h"

#include "square.h"

#include "triangle.h"

const uint32\_t FORMAT\_CODE = 06032001;

namespace document\_class {

class Factory {

public:

Factory() {

plants.emplace(TRIANGLE, std::make\_shared<Tri\_factory>());

plants.emplace(SQUARE, std::make\_shared<Squ\_factory>());

plants.emplace(OCTAGON, std::make\_shared<Oct\_factory>());

figure\_names.emplace("triangle", TRIANGLE);

figure\_names.emplace("square", SQUARE);

figure\_names.emplace("octagon", OCTAGON);

}

std::map<figure\_t, std::shared\_ptr<Fact\_Interface>> plants;

std::map<std::string, figure\_t> figure\_names;

};

class Document {

private:

uint32\_t id\_;

std::string doc\_name;

std::list<std::shared\_ptr<figure::Figure>> buffer;

void Save\_private(const std::string& file\_name) const;

void Load\_private(const std::string& file\_name);

public:

Document();

explicit Document(std::string name);

void Save(const std::string &file\_name) const;

void Load(const std::string &file\_name);

void Print() const;

void Remove\_figure(uint32\_t id);

void Remove\_last\_figure();

void figure\_add(figure\_t type, std::istream &is);

uint32\_t Get\_position(uint32\_t id);

std::shared\_ptr<figure::Figure> Get\_figure(uint32\_t id);

void Insert\_figure(uint32\_t pos, std::shared\_ptr<figure::Figure>& figure);

~Document() = default;

Factory factory;

};

}

#endif //OOP\_LAB7\_DOCUMENT\_H

**doc.cpp**

#include <algorithm>

#include <cstdint>

#include <iostream>

#include "doc.h"

document\_class::Document::Document(): id\_(1), doc\_name(""), buffer(0), factory() {}

document\_class::Document::Document(std::string name): id\_(1), doc\_name(std::move(name)), buffer(0), factory() {}

void document\_class::Document::Save(const std::string &file\_name) const {

Save\_private(file\_name);

}

void document\_class::Document::Load(const std::string &file\_name) {

Load\_private(file\_name);

}

void document\_class::Document::Save\_private(const std::string &file\_name) const {

std::ofstream os;

os.open(file\_name, std::ios\_base::binary | std::ios\_base::out);

if (!os.is\_open()) {

throw std::runtime\_error("File is not opened");

}

uint32\_t format = FORMAT\_CODE;

uint32\_t nameLen = doc\_name.size();

os.write((char\*)&format, sizeof(format));

os.write((char\*)&nameLen, sizeof(nameLen));

os.write((char\*)(doc\_name.c\_str()), nameLen);

std::for\_each(buffer.begin(), buffer.end(), [&](const std::shared\_ptr<figure::Figure>& shape) {

shape->save(os);

});

}

void document\_class::Document::Load\_private(const std::string &file\_name) {

std::ifstream is;

is.open(file\_name, std::ios\_base::binary | std::ios\_base::in);

if (!is.is\_open()) {

throw std::runtime\_error("File is not opened");

}

uint32\_t format;

uint32\_t nameLen;

is.read((char\*)&format, sizeof(format));

if (format != FORMAT\_CODE)

throw std::runtime\_error("Bad file");

is.read((char\*)&nameLen, sizeof(nameLen));

char\* name = new char[nameLen + 1];

name[nameLen] = 0;

is.read(name, nameLen);

doc\_name = std::string(name);

delete[] name;

figure\_t type;

while(true) {

is.read((char\*)&type, sizeof(type));

if (is.eof()) break;

buffer.push\_back(factory.plants[type]->Create\_figure());

buffer.back()->load(is);

}

id\_ = buffer.size();

}

void document\_class::Document::Print() const {

std::for\_each(buffer.begin(), buffer.end(), [&](const std::shared\_ptr<figure::Figure>& shape) {

shape->print(std::cout);

});

}

void document\_class::Document::Remove\_figure(uint32\_t id) {

auto it = std::find\_if(buffer.begin(), buffer.end(), [id](const std::shared\_ptr<figure::Figure>& shape) -> bool {

return id == shape->get\_ID();

});

if (it == buffer.end())

throw std::logic\_error("Figure with this id doesn't exist");

buffer.erase(it);

}

void document\_class::Document::Remove\_last\_figure() {

if (buffer.empty()) {

throw std::logic\_error("Doc is empty");

}

buffer.pop\_back();

}

void document\_class::Document::figure\_add(figure\_t type, std::istream& is) {

buffer.push\_back(factory.plants[type]->Create\_figure(id\_++, is));

}

uint32\_t document\_class::Document::Get\_position(uint32\_t id) {

auto it = std::find\_if(buffer.begin(), buffer.end(), [id](std::shared\_ptr<figure::Figure>& shape) -> bool {

return id == shape->get\_ID();

});

return std::distance(buffer.begin(), it);

}

std::shared\_ptr<figure::Figure> document\_class::Document::Get\_figure(uint32\_t id) {

auto it = std::find\_if(buffer.begin(), buffer.end(), [id](std::shared\_ptr<figure::Figure>& shape) -> bool {

return id == shape->get\_ID();

});

return \*it;

}

void document\_class::Document::Insert\_figure(uint32\_t pos, std::shared\_ptr<figure::Figure>& figure) {

auto it = buffer.begin();

std::advance(it, pos);

buffer.insert(it, figure);

}

**main.cpp**

#include <iostream>

#include "interface.h"

bool quit (Editor& editor) {

    char c;

    std::cout << "You want save file? y/n: ";

    std::cin >> c;

    if (c == 'N' || c == 'n') {

        return true;

    }

    else if (c == 'Y' || c == 'y') {

        std::string name;

        std::cout << "Enter name for savefile: ";

        std::cin >> name;

        try {

            editor.Save\_document(name);

            std::cout << "Successfully saved in " << name << '\n';

        } catch  (std::runtime\_error& err) {

            std::cout << err.what() << "\n";

            return false;

        }

        return true;

    } else {

        std::cout << "so yes or no?\n";

        return false;

    }

}

void man () {

    std::cout << "create: create new document\n"

    << "save: save document to file\n"

    << "load: load document from file\n"

    << "add: add figure\n"

    << "print: print the document\n"

    << "delete: delete figure by it`s ID\n"

    << "undo: undo previous operation\n"

    << "quit: close program and exit\n";

}

bool create(Editor& editor) {

    char c;

    if (editor.Document\_exist()) {

        std::cout << "Save document? y/n\n";

        std::cin >> c;

        if (c == 'N' || c == 'n') {

        }

        else if (c == 'Y' || c == 'y') {

            std::string name;

            std::cout << "Enter the name for file: ";

            std::cin >> name;

            try {

                editor.Save\_document(name);

                std::cout << "Successfully saved in " << name << '\n';

            } catch  (std::runtime\_error& err) {

                std::cout << err.what() << "\n";

                return false;

            }

        } else {

            std::cout << "yes or no?\n";

            return false;

        }

    }

    std::string document\_name;

    std::cout << "Enter the name of project\n";

    std::cin >> document\_name;

    editor.Create\_document(document\_name);

    std::cout << "Document " << document\_name << " is created\n";

    return true;

}

bool load(Editor& editor) {

    char c;

    if (editor.Document\_exist()) {

        std::cout << "Save document? y/n\n";

        std::cin >> c;

        if (c == 'N' || c == 'n') {

        }

        else if (c == 'Y' || c == 'y') {

            std::string name;

            std::cout << "Enter the name for file: ";

            std::cin >> name;

            try {

                editor.Save\_document(name);

                std::cout << "Successfully saved in " << name << '\n';

            } catch  (std::runtime\_error& err) {

                std::cout << err.what() << "\n";

                return false;

            }

        } else {

            std::cout << "so yes or no?\n";

            return false;

        }

    }

    std::string file\_name;

    std::cout << "Enter name of load file\n";

    std::cin >> file\_name;

    try {

        editor.Load\_document(file\_name);

        std::cout << "Successfully loaded from " << file\_name << "\n";

    } catch (std::runtime\_error& err) {

        std::cout << err.what() << "\n";

        return false;

    }

    return true;

}

bool save(Editor& editor) {

    std::string file\_name;

    std::cout << "Enter name for savefile: ";

    std::cin >> file\_name;

    try {

        editor.Save\_document(file\_name);

        std::cout << "Successfully saved in " << file\_name << '\n';

    } catch (std::runtime\_error& err) {

        std::cout << err.what() << "\n";

        return false;

    }

    return true;

}

void add (Editor& editor) {

    std::string name;

    std::cin >> name;

    editor.Insert\_figure(editor.get\_document()->factory.figure\_names[name], std::cin);

    std::cout << "Figure is added\n";

}

bool remove(Editor& editor) {

    uint32\_t id;

    std::cout << "enter ID of figure you want to delete (from 1 to ...): ";

    std::cin >> id;

    try {

        editor.Remove\_figure(id);

        std::cout << "Figure with ID " << id << " is removed\n";

    } catch (std::logic\_error& err) {

        std::cout << err.what() << "\n";

        return false;

    }

    return true;

}

int main() {

    Editor editor;

    std::string cmd;

    while (cmd != "quit") {

        std::cin >> cmd;

        if (cmd == "quit") {

           if (quit(editor)) return 0;

        } else if (cmd == "man") {

            man();

        } else if (cmd == "create") {

            create(editor);

        } else if (cmd == "save") {

            save(editor);

            std::cout << "Saved successfully" << std::endl;

        } else if (cmd == "load") {

            load(editor);

        } else if (cmd == "add") {

            add(editor);

        } else if (cmd == "delete") {

            remove(editor);

        } else if (cmd == "undo") {

            editor.Undo();

            //std::cout << "Undo done\n";

        } else if (cmd == "print") {

            editor.Print\_document();

        }

    }

    return 0;

}

**2. Ссылка на репозиторий на Github**

[https://github.com/mmaxim2710/oop\_exercise\_0](https://github.com/mmaxim2710/oop_exercise_01)7

**3.Набор testcases**

**1)**

man

create doc

add

triangle

0 0 2 2 0 2

add

square

0 0

0 5

5 5

5 0

add

octagon

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

print

delete 3

print

save

1

quit

n

**2)**

create doc

add triangle 0 0 2 2 0 2

print

undo

print

add

square

0 0 0 5 5 5 5 0

delete 2

undo print

quit

n

**4. Результат выполнения тестов**

**1)**

man

create - create new document

save - save document to file

load - load document from file

add - add figure

print - print the document

delete - delete figure by it`s ID

undo - undo previous operation

quit - close program and exit

create doc

Enter name of new project

Document doc is created

add

triangle

0 0 2 2 0 2

Figure is added

add

square

0 0

0 5

5 5

5 0

Figure is added

add

octagon

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure is added

print

=================================

id - 1

Figure - Triangle

Area: 2

Center: (0.666667, 1.33333)

Triangle coordinates

(0, 0)

(2, 2)

(0, 2)

=================================

id - 2

Figure - Square

Area: 25

Center: (2.5, 2.5)

Square coordinates:

(0, 0)

(0, 5)

(5, 5)

(5, 0)

=================================

id - 3

Figure - Octagon

Area: 0

Center: (1, 1)

Octagon coordinates:

(1, 1)

(1, 1)

(1, 1)

(1, 1)

(1, 1)

(1, 1)

(1, 1)

(1, 1)

delete 3

enter ID of figure you fant to remove (you can see it in print): Figure with

ID 3 is removed

print

=================================

id - 1

Figure - Triangle

Area: 2

Center: (0.666667, 1.33333)

Triangle coordinates

(0, 0)

(2, 2)

(0, 2)

=================================

id - 2

Figure - Square

Area: 25

Center: (2.5, 2.5)

Square coordinates:

(0, 0)

(0, 5)

(5, 5)

(5, 0)

man

create - create new document

save - save document to file

load - load document from file

add - add figure

print - print the document

delete - delete figure by it`s ID

undo - undo previous operation

quit - close program and exit

save

Enter name for savefile: 1

Successfully saved in 1

Saved successfully

quit

You want save file? y/n: n

**2)**

create doc

Enter name of new project

Document doc is created

add triangle 0 0 2 2 0 2

Figure is added

print

=================================

id - 1

Figure - Triangle

Area: 2

Center: (0.666667, 1.33333)

Triangle coordinates

(0, 0)

(2, 2)

(0, 2)

undo

Undo done

print

add

square

0 0 0 5 5 5 5 0

Figure is added

delete 2

enter ID of figure you fant to remove (you can see it in print): Figure with ID 2 is removed

undo

Undo done

print

=================================

id - 2

Figure - Square

Area: 25

Center: (2.5, 2.5)

Square coordinates:

(0, 0)

(0, 5)

(5, 5)

(5, 0)

quit

You want save file? y/n: n

**5. Объяснение результатов программы**

Классы фигур объединены в один неймспейс «figures» и взяты из лабораторной работы №3. Описан класс factory, который вызывает конструкторы фигур, класс editor, который является «оберткой» класса Document. Он вызывает его функции и записывает его в стек для отмены (undo).   
 Класс команд наследуется от абстрактного класса для удобного вызова команд из стека.

**Вывод:** Проделав данную работу я улучшил свои знания в принципах наследования, организации подобных структур, таких как моя программа. Создал сложную систему классов.