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: | |

Москва 2019

# 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_07

## 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.