Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 7: «Проектирование структуры классов»

Группа:	М8О-208Б-18, №3
Студент:	Анисимов Валерий Алексеевич
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	25.12.2019

Москва, 2019

1. **Тема**: Проктирование структуры классов

2. **Цель работы**: Получение практических навыков в хороших практиках проектирования структуры классов приложения

3. **Задание** (*вариант № 3*):
Фигуры — прямоугольник, трапеция, ромб.

4. **Адрес репозитория на GitHub**
https://github.com/wAlienUFOx/oop_exercise_07

5. **Код программы на C++**
main.cpp

```
#include <iostream>
#include <algorithm>
#include <map>
#include "rectangle.h"
#include "containers/stack.h"
#include "allocators/allocator.h"

#include <iostream>
#include "editor.h"

int main(){
    Editor editor;
    char cmd;
    std::cout << "Input command. Input 'h' for help;" << std::endl;
    while (std::cin >> cmd) {
        switch (cmd) {
            case 'h':
                std::cout << "h - help\n"
                          << "c - create\n"
                          << "l - load\n"
                          << "s - save\n"
                          << "a - add\n"
                          << "r - remove\n"
                          << "u - undo\n"
                          << "p - print\n"
                          << "q - quit\n";
                break;
            case 'c': {
                std::string name;
                std::cin >> name;
```

```
            editor.CreateDocument(name);
            std::cout << "Document " << name << " is created" << std::endl;
            break;
        }
        case 'l':{
            std::string filename;
            std::cin >> filename;
            try {
                editor.LoadDocument(filename);
                std::cout << "Document loaded" << std::endl;
            } catch (std::runtime_error& err) {
                std::cout << err.what() << std::endl;
            }
            break;
        }
        case 's': {
            std::string filename;
            std::cin >> filename;

            try {
                editor.SaveDocument(filename);
                std::cout << "Document save" << std::endl;
            } catch (std::runtime_error &err) {
                std::cout << err.what() << std::endl;
            }
            break;
        }
        case 'a': {
            try {
                editor.InsertPrimitive(std::cin);
                std::cout << "Added" << std::endl;
            }catch (std::logic_error& err) {
                std::cout << err.what() << std::endl;
            }
            break;
        }
        case 'r':{
            int id;
            std::cin >> id;
            try {
                editor.RemovePrimitive(id);
                std::cout << "Removed" << std::endl;
            } catch (std::logic_error& err) {
                std::cout << err.what() << std::endl;
            }
```

```
            break;
        }
        case 'u':
            editor.Undo();
            break;
        case 'p':
            editor.PrintDocument();
            break;
        case 'q':
            return 0;
        default:
            std::cout << "Wrong. Input 'h' for help\n";
      }
   }
   return 0;
}
```

point.h

```
#ifndef OOP_LAB7_POINT_H
#define OOP_LAB7_POINT_H

#include <iostream>


struct point {
   point(): x(0), y(0) {}
   point(double a, double b): x(a), y(b) {}
   double x;
   double y;
};


std::istream& operator>>(std::istream& is, point& p) {
   is >> p.x >> p.y;
   return is;
}


std::ostream& operator<<(std::ostream& os, point p) {
   os << '(' << p.x << ' ' << p.y << ')';
   return os;
}
```

```
#endif
figure.h

#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>
#include <cmath>
#include "point.h"

namespace figures{

    enum FigureType {
        rhombus,
        rectangle,
        trapeze
    };

    class Figure {
    public:
    virtual std::ostream& print(std::ostream& out) const = 0;
    virtual void Serialize(std::ofstream& os) const = 0;
    virtual void Deserialize(std::ifstream& is) = 0;
    ~Figure() = default;
    };


    class Rectangle : public Figure {
    public:
    point A , B, C, D;

    Rectangle(): A{0, 0}, B{0, 0}, C{0, 0}, D{0,0} {}

    explicit Rectangle(std::istream& is) {
        is >> A >> B >> C >> D;
        double a, b, c, d, d1, d2, ABC, BCD, CDA, DAB;
        a = sqrt((B.x- A.x) * (B.x - A.x) + (B.y - A.y) * (B.y - A.y));
        b = sqrt((C.x- B.x) * (C.x - B.x) + (C.y - B.y) * (C.y - B.y));
        c = sqrt((C.x- D.x) * (C.x - D.x) + (C.y - D.y) * (C.y - D.y));
        d = sqrt((D.x- A.x) * (D.x - A.x) + (D.y - A.y) * (D.y - A.y));
        d1 = sqrt((B.x- D.x) * (B.x - D.x) + (B.y - D.y) * (B.y - D.y));
        d2 = sqrt((C.x- A.x) * (C.x - A.x) + (C.y - A.y) * (C.y - A.y));
        ABC = (a * a + b * b - d2 * d2) / 2 * a * b;
        BCD = (b * b + c * c - d1 * d1) / 2 * b * c;
        CDA = (d * d + c * c - d2 * d2) / 2 * d * c;
```

```
        DAB = (a * a + d * d - d1 * d1) / 2 * a * d;
        if(ABC != BCD || ABC != CDA || ABC != DAB)
            throw std::logic_error("It`s not a rectangle");
        }

    std::ostream& print(std::ostream& os) const override {
        os << "rectangle: " <<  A << " " << B << " " << C << " " << D << std::endl;
        return os;
    }

    void Serialize(std::ofstream& os) const override {
        FigureType type = rectangle;
        os.write(reinterpret_cast<char*>(&type), sizeof(type));
        os << A.x << " " << A.y << " " << B.x << " " << B.y << " " << C.x << " " <<
C.y << " " << D.x << " " << A.y;
    }

    void Deserialize(std::ifstream& is) override {

        is >> A >> B >> C >> D;
    }

    };


    class Trapeze : public Figure {
    public:
    point A, B, C, D;

    Trapeze(): A{0, 0}, B{0, 0}, C{0, 0}, D{0,0} {}

    explicit Trapeze(std::istream& is){
        is >> A >> B >> C >> D;
        if((C.y - B.y) / (C.x - B.x) != (D.y - A.y) / (D.x - A.x))
            throw std::logic_error("It`s not a trapeze");
    }

    std::ostream& print(std::ostream& os) const override {
        os << "trapeze: " << A << " " << B << " " << C << " " << D << std::endl;
        return os;
    }

    void Serialize(std::ofstream& os) const override {
        FigureType type = trapeze;
        os.write(reinterpret_cast<char*>(&type), sizeof(type));
```

```
        os << A.x << " " << A.y << " " << B.x << " " << B.y << " " << C.x << " " << C.y << " " << D.x << " " << A.y;
    }

    void Deserialize(std::ifstream& is) override {
        is >> A >> B >> C >> D;
    }


    };


    class Rhombus : public Figure {
    public:
    point A, B, C, D;


    Rhombus(): A{0, 0}, B{0, 0}, C{0, 0}, D{0,0} {}

    explicit Rhombus(std::istream& is){
        is >> A >> B >> C >> D;
        double a, b, c, d;
        a = sqrt((B.x - A.x) * (B.x - A.x) + (B.y - A.y) * (B.y - A.y));
        b = sqrt((C.x - B.x) * (C.x - B.x) + (C.y - B.y) * (C.y - B.y));
        c = sqrt((C.x - D.x) * (C.x - D.x) + (C.y - D.y) * (C.y - D.y));
        d = sqrt((D.x - A.x) * (D.x - A.x) + (D.y - A.y) * (D.y - A.y));
        if(a != b || a != c || a != d)
            throw std::logic_error("It`s not a rhombus");
    }

    std::ostream& print(std::ostream& os) const override {
         os << "rhombus: " << A << " " << B << " " << C << " " << D << std::endl;
        return os;
    }

    void Serialize(std::ofstream& os) const override {
        FigureType type = rhombus;
        os.write(reinterpret_cast<char*>(&type), sizeof(type));
        os << A.x << " " << A.y << " " << B.x << " " << B.y << " " << C.x << " " << C.y << " " << D.x << " " << A.y;
    }

    void Deserialize(std::ifstream& is) override {
        is >> A >> B >> C >> D;
    }
```

```
};

}

#endif
```

editor.h

```
#ifndef EDITOR_H
#define EDITOR_H

#include "document.h"
#include "command.h"
#include <iostream>
#include <stack>

class Editor {
public:
    Editor() : Doc(nullptr), History()
    {
    }

    void CreateDocument(const std::string& name) {
        Doc = std::make_shared<Document>(name);
     while(!History.empty())
            History.pop();
    }

    void InsertPrimitive(std::istream& is){
        std::shared_ptr<Command> command = std::shared_ptr<Command>(new
InsertCommand(is));
        command->SetDocument(Doc);
        command->Execute();
        History.push(command);
    }

    void RemovePrimitive(uint32_t id) {
        std::shared_ptr<Command> command = std::shared_ptr<Command>(new
RemoveCommand(id));
        command->SetDocument(Doc);
        command->Execute();
        History.push(command);
    }

    void SaveDocument(const std::string& filename) {
```

```
        Doc->Save(filename);
    }

    void LoadDocument(const std::string& filename) {
        Doc = std::make_shared<Document>("NewDoc");
        Doc->Load(filename);
     while(!History.empty())
            History.pop();
    }

    void Undo() {
        if (History.empty()) {
            throw std::logic_error("History is empty");
        }
        std::shared_ptr<Command> last = History.top();
        last->UnExecute();
        History.pop();
    }

    void PrintDocument() {
        Doc->Print();
    }

    ~Editor() = default;
private:
    std::shared_ptr<Document> Doc;
    std::stack<std::shared_ptr<Command>> History;
};

#endif
```

document.h

```
#ifndef DOCUMENT_H
#define DOCUMENT_H

#include <fstream>
#include <cstdint>
#include <memory>
#include <string>
#include <algorithm>
#include "figure.h"
#include "factory.h"
#include <vector>

class Document {
```

```
public:
    Document(): Name(""), Vec(0), Factory()
    {}

    explicit Document(std::string name):
            Name(std::move(name)),
            Vec(0),
            Factory()
    {}

    ~Document() = default;

    void Save(const std::string& filename) const {
        std::ofstream os;
        os.open(filename, std::ios_base::binary | std::ios_base::out);
        if (!os.is_open()) {
            throw std::runtime_error("File is not opened");
        }
        uint32_t nameLen = Name.size();
        os.write((char*)&nameLen, sizeof(nameLen));
        os.write((char*)(Name.c_str()), nameLen);
        for (const auto& shape : Vec) {
            shape->Serialize(os);
        }
    }

    void Load(const std::string& filename) {
        std::ifstream is;
        is.open(filename, std::ios_base::binary | std::ios_base::in);
        if (!is.is_open()) {
            throw std::runtime_error("File is not opened");
        }
        uint32_t nameLen;
        is.read((char*)&nameLen, sizeof(nameLen));
        char* name = new char[nameLen + 1];
        name[nameLen] = 0;
        is.read(name, nameLen);
        Name = std::string(name);
        delete[] name;
        figures::FigureType type;
        while(is.read((char*)&type, sizeof(type))) {
            Vec.push_back(Factory.FigureCreate(type));
            Vec.back()->Deserialize(is);
        }
    }
```

```
    void Print() {
        int it = 0;
        std::for_each(Vec.begin(), Vec.end(), [&it](std::shared_ptr<figures::Figure>&
fig) {
            std::cout << it << ") ";
            fig->print(std::cout);
            it++;
        });
    }

    void Remove(uint32_t Id) {
        if (Id >= Vec.size())
            throw std::logic_error("Wrong index");
        Vec.erase(Vec.begin() + Id);
    }

    void Insert(std::istream& is) {
        Vec.push_back(Factory.FigureCreate(is));
    }

    void  Insert(uint32_t Id, std::shared_ptr<figures::Figure> figure) {
        Vec.insert(Vec.begin() + Id, figure);
    }

private:
    std::string Name;
    std::vector<std::shared_ptr<figures::Figure>> Vec;
    factory::Factory Factory;

    friend class InsertCommand;
    friend class RemoveCommand;

    void RemoveLastPrimitive() {
        if (Vec.empty())
            throw std::logic_error("Document is empty");
        Vec.pop_back();
    }

    std::shared_ptr<figures::Figure> GetFigure(uint32_t Id) {
        if (Id >= Vec.size())
            throw std::logic_error("Wrong index");
        auto it = Vec.begin() + Id;
        return *it;
    }
};
```

```
#endif
```
factory.h

```
#ifndef FACTORY_H
#define FACTORY_H

#include <iostream>
#include "figure.h"

namespace factory {

    class Factory {
    public:

        std::shared_ptr<figures::Figure> FigureCreate(figures::FigureType type)
const {
            if (type == figures::rhombus) {
                return std::shared_ptr<figures::Figure>(new figures::Rhombus());
            } else if (type == figures::rectangle) {
                return std::shared_ptr<figures::Figure>(new figures::Rectangle());
            } else if (type == figures::trapeze) {
                return std::shared_ptr<figures::Figure>(new figures::Trapeze());
            }
            throw std::logic_error("Wrong. Figures: rhombus, rectangle, trapeze");
        }

        std::shared_ptr<figures::Figure> FigureCreate(std::istream &is) const {
            std::string type;
            std::cin >> type;
            if (type == "rhombus") {
                return std::shared_ptr<figures::Figure>(new figures::Rhombus(is));
            } else if (type == "rectangle") {
                return std::shared_ptr<figures::Figure>(new figures::Rectangle(is));
            } else if (type == "trapeze") {
                return std::shared_ptr<figures::Figure>(new figures::Trapeze(is));
            }
            throw std::logic_error("Wrong. Figures: rhombus, rectangle, trapeze");
        }
    };
}
#endif
```

command.h

```
#ifndef COMMAND_H
#define COMMAND_H

#include "document.h"
#include <stack>
#include <iostream>

class Command {
protected:
    std::shared_ptr<Document> Doc;
public:
    virtual ~Command() = default;
    virtual void Execute() = 0;
    virtual void UnExecute() = 0;

    void SetDocument(std::shared_ptr<Document> doc) {
        Doc = doc;
    }
};

class InsertCommand : public Command {
public:
    explicit InsertCommand(std::istream& is):
            input(is)
    {}

    void Execute() override {
        Doc->Insert(input);
    }

    void UnExecute() override {
        Doc->RemoveLastPrimitive();
    }

private:
    std::istream& input;
};

class RemoveCommand : public Command {
public:
    explicit RemoveCommand(uint32_t id): Id(id)
    {}

    void Execute() override {
        Fig = Doc->GetFigure(Id);
```

```
        Doc->Remove(Id);
    }

    void UnExecute() override {
        Doc->Insert(Id, Fig);
    }

private:
    uint32_t Id;
    std::shared_ptr<figures::Figure> Fig;
};

#endif
```

CMakeLists.txt

```
cmake_minimum_required (VERSION 3.5)

project(lab7)

add_executable(oop_exercise_07
  main.cpp)

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -g3 -Wextra")

set_target_properties(oop_exercise_07 PROPERTIES CXX_STANDART 14
CXX_STANDART_REQUIRED ON)
```

6. Набор testcases

test_01.txt

```
c doc
a rectangle 0 0 0 2 2 2 2 0
a rectangle 0 0 0 3 5 3 5 0
a rhombus -2 0 0 4 2 0 0 -4
p
s doc.txt
r 0 r 0 r 0
p
l doc.txt
p
q
```

test_02.txt

```
c doc
a rectangle 0 0 0 1 4 1 4 0
a trapeze 0 0 1 1 2 1 3 0
p
r 0
p
u
p
a rhombus 0 0 -1 1 0 2 1 1
p
u
p
q
```

7. Результаты выполнения тестов

```
walien@PC-name:~/2kurs/OOP/lab7/tmp$ ./oop_exercise_07 < ../test_01.txt
Input command. Input 'h' for help;
Document doc is created
Added
Added
Added
0) rectangle: (0 0) (0 2) (2 2) (2 0)
1) rectangle: (0 0) (0 3) (5 3) (5 0)
2) rhombus: (-2 0) (0 4) (2 0) (0 -4)
Document save
Removed
Removed
Removed
Document loaded
0) rectangle: (0 0) (0 2) (2 2) (2 0)
1) rectangle: (0 0) (0 3) (5 3) (5 0)
2) rhombus: (-2 0) (0 4) (2 0) (0 0)
```

```
walien@PC-name:~/2kurs/OOP/lab7/tmp$ ./oop_exercise_07 < ../test_02.txt
Input command. Input 'h' for help;
Document doc is created
Added
Added
0) rectangle: (0 0) (0 1) (4 1) (4 0)
1) trapeze: (0 0) (1 1) (2 1) (3 0)
Removed
0) trapeze: (0 0) (1 1) (2 1) (3 0)
```

0) rectangle: (0 0) (0 1) (4 1) (4 0)
1) trapeze: (0 0) (1 1) (2 1) (3 0)
Added
0) rectangle: (0 0) (0 1) (4 1) (4 0)
1) trapeze: (0 0) (1 1) (2 1) (3 0)
2) rhombus: (0 0) (-1 1) (0 2) (1 1)
0) rectangle: (0 0) (0 1) (4 1) (4 0)
1) trapeze: (0 0) (1 1) (2 1) (3 0)

8. Объяснение результатов работы программы - вывод

В main.cpp посредством editor.h осуществяются действия с документом. В command.h реализованы создание, выполнение и обратное выполнение команды, необходимые для реализации undo; в document.h — действия с документом, в factory.h реализован класс Factory, создающий графические примитивы.

В ходе лабораторной работы были усовершенствованны навыки объектно-ориентированного программирования, укреплены знания о наследовании, полиморфизме, класах.