Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа 7
по курсу «ООП»

Тема:
Проектирование структуры классов.

Студент:	Болдырев А. К.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	10
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

figure.h:

```
#ifndef _FIGURE_H_
#define _FIGURE_H_

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

struct figure {
   virtual point center() const = 0;
   virtual void print(std::ostream&) const = 0 ;
   virtual void printFile(std::ofstream&) const = 0 ;
   virtual double area() const = 0;
   virtual ~figure() = default;
};


#endif //_FIGURE_H_
```

point.h:

```
#ifndef  _POINT_H_
#define  _POINT_H_

#include <iostream>

struct point {
   double x, y;
   point (double  a,double b) { x = a, y = b;};
   point() = default;
};

std::istream& operator >> (std::istream& is,point& p );
std::ostream& operator << (std::ostream& os,const point& p);


#endif // _POINT_H_
```

square.h:

```
#ifndef _SQUARE_H_
#define _SQUARE_H_

#include "figure.h"

struct square : figure {
private:
   point a1,a2,a3,a4;
public:
   point center() const override ;
   void print(std::ostream&) const  override ;
   void printFile(std::ofstream&) const override ;
   double area() const override ;
   square() = default;
   square(std::istream& is);
   square(std::ifstream& is);
};

#endif //_SQUARE_H_
```

square.cpp:

```
#include "square.h"

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

point square::center() const {
   double x,y;
   x = (a1.x + a2.x + a3.x + a4.x) / 4;
   y = (a1.y + a2.y + a3.y + a4.y) / 4;
   point p(x,y);
   return p;
}

void square::print(std::ostream& os) const {
   os << "square\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n';
   os << "Center: " << center() << '\n' << "Area:" << area() << '\n';
}

void square::printFile(std::ofstream& of) const {
   of << "square\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n';
}
```

```
double square::area() const{
    double vecX = a2.x - a1.x;
    double vecY = a2.y - a1.y;

    return vecX * vecX + vecY * vecY;
}

square::square(std::istream& is) {
    is >> a1 >> a2 >> a3 >> a4;
}

square::square(std::ifstream& is) {
    is >> a1 >> a2 >> a3 >> a4;
}
```

rectangle.h:

```
#ifndef _RECTANGLE_H_
#define _RECTANGLE_H_

#include "figure.h"

struct rectangle : figure{
private:
    point a1,a2,a3,a4;
public:
    point center() const override ;
    void print(std::ostream&) const override ;
    void printFile(std::ofstream&) const override ;
    double area() const override ;
    rectangle() = default;
    rectangle(std::istream& is);
    rectangle(std::ifstream& is);
};

#endif //_RECTANGLE_H_
```

rectangle.cpp

```
#include <cmath>

#include "rectangle.h"

point rectangle::center() const {
```

```
    double x,y;
    x = (a1.x + a2.x + a3.x + a4.x) / 4;
    y = (a1.y + a2.y + a3.y + a4.y) / 4;
    point p(x,y);
    return p;
}

void rectangle::print(std::ostream& os) const {
    os << "rectangle\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n';
    os << "Center: " << center() << '\n' << "Area:" << area() << '\n';
}

void rectangle::printFile(std::ofstream &of) const {
    of << "rectangle\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n';
}

double rectangle::area() const {
    double xHeight = a2.x - a1.x;
    double yHeight = a2.y - a1.y;

    double xWidth = a3.x - a2.x;
    double yWidth = a3.y - a2.y;

     return sqrt(xHeight * xHeight + yHeight * yHeight) * sqrt(xWidth * xWidth +
yWidth * yWidth);
}

rectangle::rectangle(std::istream& is) {
    is >> a1 >> a2 >> a3 >> a4;
}

rectangle::rectangle(std::ifstream& is) {
    is >> a1 >> a2 >> a3 >> a4;
}
```

command.h:

```
#ifndef _COMMAND_H_
#define _COMMAND_H_
#include "document.h"

struct Acommand {
    virtual ~Acommand() = default;
    virtual void UnExecute() = 0;
```

```
protected:
   std::shared_ptr<document> doc_;
};

struct InsertCommand : public Acommand {
public:
   void UnExecute() override;
   InsertCommand(std::shared_ptr<document>& doc);
};

struct DeleteCommand : public Acommand {
public:
DeleteCommand(std::shared_ptr<figure>& newFigure, uint32_t
newIndex,std::shared_ptr<document>& doc);
          void UnExecute() override;

private:
   std::shared_ptr<figure> figure_;
   uint32_t index_;
};
#endif //_COMMAND_H_
```

command.cpp:

```
#include "command.h"

void InsertCommand::UnExecute() {
   doc_->RemoveLast();
}

InsertCommand::InsertCommand(std::shared_ptr<document> &doc) {
   doc_ = doc;
}

DeleteCommand::DeleteCommand(std::shared_ptr<figure>   &newFigure,   uint32_t
newIndex, std::shared_ptr<document> &doc) {
   doc_ = doc;
   figure_ = newFigure;
   index_ = newIndex;
}

void DeleteCommand::UnExecute() {
   doc_->InsertIndex(figure_,index_);
}
```

document.h:

```
#ifndef _DOCUMENT_H_
#define _DOCUMENT_H_

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

struct document {
public:
    void Print() const ;
    document(std::string& newName): name_(newName), factory_(), buffer_(0) {};
    void Insert(std::shared_ptr<figure>& ptr);
    void Rename(const std::string& newName);
    void Save (const std::string& filename) const;
    void Load(const std::string& filename);
    std::shared_ptr<figure> GetFigure(uint32_t index);
    void Erase(uint32_t index);
    std::string GetName();
    size_t Size();

private:
    friend class InsertCommand;
    friend class DeleteCommand;
    factory factory_;
    std::string name_;
    std::vector<std::shared_ptr<figure>> buffer_;

    void RemoveLast();

    void InsertIndex(std::shared_ptr<figure>& newFigure, uint32_t index);
};

#endif //_DOCUMENT_H_
```

document.cpp:

```
#include "document.h"
```

```
void document::Print() const {
    {
        if (buffer_.empty()) {
            std::cout << "Buffer is empty\n";
        }
        for (auto elem : buffer_) {
            elem->print(std::cout);
        }
    }
}

void document::Insert(std::shared_ptr<figure> &ptr) {
    buffer_.push_back(ptr);
}

void document::Rename(const std::string &newName) {
    name_ = newName;
}

void document::Save(const std::string &filename) const {
    std::ofstream fout;
    fout.open(filename);
    if (!fout.is_open()) {
        throw std::runtime_error("File is not opened\n");
    }
    fout << buffer_.size() << '\n';
    for (auto elem : buffer_) {
        elem->printFile(fout);
    }
}

void document::Load(const std::string &filename) {
    std::ifstream fin;
    fin.open(filename);
    if (!fin.is_open()) {
        throw std::runtime_error("File is not opened\n");
    }
    size_t size;
    fin >> size;
    buffer_.clear();
    for (int i = 0; i < size; ++i) {
        buffer_.push_back(factory_.FigureCreateFile(fin));
    }
    name_ = filename;
```

```
}

std::shared_ptr<figure> document::GetFigure(uint32_t index) {
    return buffer_[index];
}

void document::Erase(uint32_t index) {
    if ( index >= buffer_.size()) {
        throw std::logic_error("Out of bounds\n");
    }
    buffer_[index] = nullptr;
    for (; index < buffer_.size() - 1; ++index) {
        buffer_[index] = buffer_[index + 1];
    }
    buffer_.pop_back();
}

std::string document::GetName() {
    return this->name_;
}

size_t document::Size() {
    return buffer_.size();
}

void document::RemoveLast() {
    if (buffer_.empty()) {
        throw std::logic_error("Document is empty");
    }
    buffer_.pop_back();
}

void document::InsertIndex(std::shared_ptr<figure> &newFigure, uint32_t index) {
    buffer_.insert(buffer_.begin() + index, newFigure);
}
```

editor.h:

```
#ifndef _D_EDITOR_H_
#define _D_EDITOR_H_

#include "figure.h"
#include "document.h"
#include <stack>
#include "command.h"
```

```
struct editor {
private:
    std::shared_ptr<document> doc_;
    std::stack<std::shared_ptr<Acommand>> history_;
public:
    ~editor() = default;
    void PrintDocument();
    void CreateDocument(std::string& newName);
    bool DocumentExist();
    editor() : doc_(nullptr), history_() {}
    void InsertInDocument(std::shared_ptr<figure>& newFigure);
    void DeleteInDocument(uint32_t index);
    void SaveDocument();
    void LoadDocument(std::string& name);
    void Undo();

};

#endif //_D_EDITOR_H_
```

editor.cpp:

```
#include "editor.h"

void editor::PrintDocument() {
    if (doc_ == nullptr) {
        std::cout << "No document!\n";
        return;
    }
    doc_->Print();
}

void editor::CreateDocument(std::string &newName) {
    doc_ = std::make_shared<document>(newName);
}

bool editor::DocumentExist() {
    return doc_ != nullptr;
}

void editor::InsertInDocument(std::shared_ptr<figure> &newFigure) {
```

```
    if (doc_ == nullptr) {
        std::cout << "No document!\n";
        return;
    }
        std::shared_ptr<Acommand> command = std::shared_ptr<Acommand>(new
InsertCommand(doc_));
    doc_->Insert(newFigure);
    history_.push(command);
}

void editor::DeleteInDocument(uint32_t index) {
    if (doc_ == nullptr) {
        std::cout << "No document!\n";
        return;
    }
    if (index >= doc_->Size()) {
        std::cout << "Out of bounds\n";
        return;
    }
    std::shared_ptr<figure> tmp = doc_->GetFigure(index);
        std::shared_ptr<Acommand> command = std::shared_ptr<Acommand>(new
DeleteCommand(tmp,index,doc_));
    doc_->Erase(index);
    history_.push(command);
}

void editor::SaveDocument() {
    if (doc_ == nullptr) {
        std::cout << "No document!\nNot ";
        return;
    }
    std::string saveName = doc_->GetName();
    doc_ ->Save(saveName);
}

void editor::LoadDocument(std::string &name) {
    try {
        doc_ = std::make_shared<document>(name);
        doc_->Load(name);
        while (!history_.empty()){
            history_.pop();
        }
    } catch(std::logic_error& e) {
        std::cout << e.what();
    }
```

```
}

void editor::Undo() {
    if (history_.empty()) {
        throw std::logic_error("History is empty\n");
    }
    std::shared_ptr<Acommand> lastCommand = history_.top();
    lastCommand->UnExecute();
    history_.pop();
}
```

factory.h:

```
#ifndef _FACTORY_H_
#define _FACTORY_H_

#include <memory>
#include <iostream>
#include <fstream>
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"
#include <string>

struct factory {
    std::shared_ptr<figure> FigureCreate(std::istream& is);

    std::shared_ptr<figure> FigureCreateFile(std::ifstream& is);

};

#endif //_FACTORY_H_
```

factory.cpp:

```
#include "factory.h"

std::shared_ptr<figure> factory::FigureCreate(std::istream &is) {
    std::string name;
    is >> name;
    if ( name == "square" ) {
        return std::shared_ptr<figure> ( new square(is));
    } else if ( name == "rectangle") {
        return std::shared_ptr<figure> ( new rectangle(is));
    } else if ( name == "trapezoid") {
```

```
        return std::shared_ptr<figure> ( new trapezoid(is));
    } else {
        throw std::logic_error("There is no such figure\n");
    }
}

std::shared_ptr<figure> factory::FigureCreateFile(std::ifstream &is) {
    std::string name;
    is >> name;
    if ( name == "square" ) {
        return std::shared_ptr<figure> ( new square(is));
    } else if ( name == "rectangle") {
        return std::shared_ptr<figure> ( new rectangle(is));
    } else if ( name == "trapezoid") {
        return std::shared_ptr<figure> ( new trapezoid(is));
    } else {
        throw std::logic_error("There is no such figure\n");
    }
}
```

main.cpp:

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

void help() {
    std::cout << "help - print this menu\n"
                 "create <path> - create a new file\n"
                 "save - save data to a file\n"
                 "load <path> - load data from a file\n"
                 "add <square, rectangle or trapezoid> <vertices> - add a figure\n"
                 "remove <index> - remove a figure by index\n"
                 "print - print all the current figures\n"
                 "undo - undo the last addition / removal action\n"
                 "exit\n";
}

void create(editor& edit) {
    std::string tmp;
    std::cin >> tmp;
    edit.CreateDocument(tmp);
    std::cout << "OK\n";
}
```

```
void load(editor& edit) {
    std::string tmp;
    std::cin >> tmp;
    try {
        edit.LoadDocument(tmp);
        std::cout << "OK\n";
    } catch (std::runtime_error& e) {
        std::cout << e.what();
    }
}

void save(editor& edit) {
    std::string tmp;
    try {
        edit.SaveDocument();
        std::cout << "OK\n";
    } catch (std::runtime_error& e) {
        std::cout << e.what();
    }
}

void add(editor& edit) {
    factory fac;
    try {
        std::shared_ptr<figure> newElem = fac.FigureCreate(std::cin);
        edit.InsertInDocument(newElem);
    } catch (std::logic_error& e) {
        std::cout << e.what() << '\n';
    }
    std::cout << "Ok\n";
}

void remove(editor& edit) {
    uint32_t index;
    std::cin >> index;
    try {
        edit.DeleteInDocument(index);
        std::cout << "Ok\n";
    } catch (std::logic_error& err) {
        std::cout << err.what() << "\n";
    }
}

int main() {
    editor edit;
```

```
    std::string command;
    while (true) {
        std::cin >> command;
        if (command == "help") {
            help();
        } else if (command == "create") {
            create(edit);
        } else if (command == "load") {
            load(edit);
        } else if (command == "save") {
            save(edit);
        } else if (command == "exit") {
            break;
        } else if (command == "add") {
            add(edit);
        } else if (command == "remove") {
            remove(edit);
        } else if (command == "print") {
            edit.PrintDocument();
        } else if (command == "undo") {
            try {
                edit.Undo();
            } catch (std::logic_error& e) {
                std::cout << e.what();
            }
        } else {
            std::cout << "Unknown command\n";
        }
    }
    return 0;
}
```

2. Ссылка на репозиторий на GitHub.

https://github.com/Anton-Boldyrev/oop_exercise_07

3. Набор тестов.

test_01.test:

```
create figures1.txt
add square 0 0 0 0 0 0 0 0
add rectangle 0 0 2 0 2 1 0 1
add trapezoid 0 0 2 0 2 1 0 1
print
remove 0
```

```
print
save
create figures2.txt
add square 1 1 1 1 1 1 1 1
add trapezoid 1 1 1 1 1 1 1 1
save
print
load figures1.txt
print
remove 1
print
undo
print
load figures2.txt
print
add square 10 10 10 10 10 10 10 10
print
undo
print
exit
```

test_02.test:

```
create figures1.txt
print
add square 0 0 0 0 0 0 0 0
undo
print
exit
```

4. Результаты выполнения тестов.

test_01.result:

```
OK
Ok
Ok
Ok
square
0 0
0 0
0 0
0 0
Center: 0 0
Area:0
```

rectangle
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:2
trapezoid
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:1
Ok
rectangle
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:2
trapezoid
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:1
OK
OK
Ok
Ok
OK
square
1 1
1 1
1 1
1 1
Center: 1 1
Area:0
trapezoid
1 1
1 1
1 1

1 1
Center: 1 1
Area:0
OK
rectangle
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:2
trapezoid
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:1
Ok
rectangle
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:2
rectangle
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:2
trapezoid
0 0
2 0
2 1
0 1
Center: 1 0.5
Area:1
OK
square
1 1
1 1
1 1

1 1
Center: 1 1
Area:0
trapezoid
1 1
1 1
1 1
1 1
Center: 1 1
Area:0
Ok
square
1 1
1 1
1 1
1 1
Center: 1 1
Area:0
trapezoid
1 1
1 1
1 1
1 1
Center: 1 1
Area:0
square
10 10
10 10
10 10
10 10
Center: 10 10
Area:0
square
1 1
1 1
1 1
1 1
Center: 1 1
Area:0
trapezoid
1 1
1 1
1 1
1 1
Center: 1 1

Area:0

test_02.result:

OK
Buffer is empty
OK
Buffer is empty

5. Объяснение результатов работы программы.

В программе реализованы функции сохранения фигур (квадрата, прямогульника и трапеции) в файл, загрузки из файла и отмены последнего добавления / удаления фигуры в файл.

6. Вывод.

В ходе выполнения лабораторной работы я улучшил навыки работы с наследованием классов, реализовал функцию undo, которая отменяет последнее удаление и добавление фигуры, поработал с чтением и записью фигур в файл.