Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Проектирование структуры классов.

Студент:	Николаев В.А.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	14
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

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& npt,point& p );
std::ostream& operator << (std::ostream& out,const point& p);
```

point.cpp:

```
#include "point.h"

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

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

figure.h:

```
#pragma once
#include <iostream>
#include "point.h"

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

pentagon.h:

```
#pragma once
#include "figure.h"

struct pentagon : figure{
    point a1,a2,a3,a4,a5;
    point center() const override;
    void print(std::ostream& out) override;
    double area() const override;
    pentagon() = default;
    pentagon(std::istream& is);
```

```
    pentagon(std::ifstream& is);
};
```
pentagon.cpp:
```
#include "pentagon.h"

point pentagon::center() const {
    double x,y;
    x = (a1.x + a2.x + a3.x + a4.x + a5.x) / 5;
    y = (a1.y + a2.y + a3.y + a4.y + a5.y) / 5;
    point p(x,y);
    return p;
}

void pentagon::print(std::ostream& out) {
    out << "Coordinates are:\n"<<"{\n"<< a1 << a2 << a3 << a4 << a5 << "}\n";
}
double pentagon::area() const {
    return (0.5) * std::abs((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a1.y)
- ( a1.y*a2.x + a2.y*a3.x + a3.y*a4.x + a4.y*a5.x + a5.y*a1.x ));
}
pentagon::pentagon(std::istream& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5;
}
pentagon::pentagon(std::ifstream& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5;
}
```
hexagon.h:
```
#pragma once

#include "figure.h"

struct hexagon : figure
{
    point a1, a2, a3, a4, a5, a6;
    point center() const override;
    void print(std::ostream& out) override;
    double area() const override;
    hexagon() = default;
    hexagon(std::istream& is);
    hexagon(std::ifstream& npt);
};
```
hexagon.cpp:
```
#include "hexagon.h"

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

```
    double x,y;
    x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x) / 6;
    y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y) / 6;
    point p(x,y);
    return p;
}
void hexagon::print(std::ostream& out) {
    out << "Coordinates are:\n{\n"<< a1 << a2 << a3 << a4 << a5 << a6 << "}\n";
}

double hexagon::area() const {
    return 0.5 * std::abs((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a6.y +
a6.x*a1.y) - ( a1.y*a2.x + a2.y*a3.x + a3.y*a4.x + a4.y*a5.x + a5.y*a6.x + a6.y*a1.x
));
}

hexagon::hexagon(std::istream& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;
}

hexagon::hexagon(std::ifstream& npt) {
    npt >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;
}
```

octagon.h:

```
#pragma once

#include "figure.h"

struct octagon : figure
{
    point a1, a2, a3, a4, a5, a6, a7, a8;
    point center() const override;
    void print(std::ostream& out) override;
    double area() const override;
    octagon() = default;
    octagon(std::istream& is);
    octagon(std::ifstream& is);
};
```

octagon.cpp:

```
#include "octagon.h"

point octagon::center() const {
    double x,y;
    x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x + a7.x + a8.x) / 8;
    y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y + a7.y + a8.y) / 8;
```

```
    point p(x,y);
    return p;
}
void octagon::print(std::ostream& out) {
    out << "Coordinates are:\n{\n"<< a1 << a2 << a3 << a4 << a5 << a6 << a7 << a8
<< "}\n";
}

double octagon::area() const {
    return 0.5 * std::abs((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a6.y +
a6.x*a7.y + a7.x*a8.y + a8.x*a1.y) - ( a1.y*a2.x + a2.y*a3.x + a3.y*a4.x + a4.y*a5.x
+ a5.y*a6.x + a6.y*a7.x + a7.y*a8.x + a8.y*a1.x ));
}

octagon::octagon(std::istream& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;
}
octagon::octagon(std::ifstream& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;
}
```

command.h:

```
#pragma once
#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_;
};
```
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_);
}
```
factory.h:
```
#pragma once
#include <memory>
#include <iostream>
#include <fstream>
#include "hexagon.h"
#include "octagon.h"
#include "pentagon.h"
#include <string>

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

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

};
```
factory.cpp:
```
#include "factory.h"
```

```
std::shared_ptr<figure> factory::fig(std::istream &is) {
    std::string name;
    is >> name;
    if ( name == "pentagon" ) {
        return std::shared_ptr<figure> ( new pentagon(is));
    } else if ( name == "hexagon") {
        return std::shared_ptr<figure> ( new hexagon(is));
    } else if ( name == "octagon") {
        return std::shared_ptr<figure> ( new octagon(is));
    } else {
        throw std::logic_error("There is no such figure\n");
    }
}

std::shared_ptr<figure> factory::fig_from_file(std::ifstream &is) {
    std::string name;
    is >> name;
    if ( name == "pentagon" ) {
        return std::shared_ptr<figure> ( new pentagon(is));
    } else if ( name == "hexagon") {
        return std::shared_ptr<figure> ( new hexagon(is));
    } else if ( name == "octagon") {
        return std::shared_ptr<figure> ( new octagon(is));
    } else {
        throw std::logic_error("There is no such figure\n");
    }
}
```

editor.h:

```
#pragma once
#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();

};
```

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();
```

```
}
```
document.h:
```
#pragma once
#include <fstream>
#include <cstdint>
#include <memory>
#include <string>
#include <algorithm>
#include "figure.h"
#include <vector>
#include "factory.h"

struct document {
    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();
    factory factory_;
    std::string name_;
    std::vector<std::shared_ptr<figure>> buffer_;

    void RemoveLast();

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

main.cpp:

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

void create(editor& edit) {
    std::string tmp;
    std::cout << "Enter name of new document\n";
    std::cin >> tmp;
    edit.CreateDocument(tmp);
    std::cout << "Document create\n";
}

void load(editor& edit) {
    std::string tmp;
    std::cout << "Enter path to the file\n";
    std::cin >> tmp;
    try {
        edit.LoadDocument(tmp);
        std::cout << "Document loaded\n";

    } catch (std::runtime_error& e) {
        std::cout << e.what();
    }
}
```

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

void add(editor& edit) {
    factory fac;
    try {
        std::shared_ptr<figure> newElem = fac.fig(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::cout << "Enter index\n";
    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;
    char action;
    while (true) {
        std::cout << "Enter letter:\n"
                  "a)create\n"
                  "b)load\n"
                  "c)save\n"
                  "d)add\n"
                  "e)remove\n"
                  "f)print\n"
```

```
            "g)undo\n"
            "h)exit\n";
std::cin >> action;
if (action == 'a') {
    std::string tmp;
    std::cout << "Enter name of new document\n";
    std::cin >> tmp;
    edit.CreateDocument(tmp);
    std::cout << "Document created\n";
} else if (action == 'b') {
    std::string tmp;
    std::cout << "Enter path to the file\n";
    std::cin >> tmp;
    try {
        edit.LoadDocument(tmp);
        std::cout << "Document loaded\n";

    } catch (std::runtime_error& e) {
        std::cout << e.what();
    }
} else if (action == 'c') {
    std::string tmp;
    try {
        edit.SaveDocument();
        std::cout << "Document saved\n";
    } catch (std::runtime_error& e) {
        std::cout << e.what();
    }
} else if (action == 'd') {
    factory fac;
    try {
        std::shared_ptr<figure> newElem = fac.fig(std::cin);
        edit.InsertInDocument(newElem);
    } catch (std::logic_error& e) {
        std::cout << e.what() << '\n';
    }
    std::cout << "Ok\n";
} else if (action == 'e') {
    uint32_t index;
    std::cout << "Enter index\n";
    std::cin >> index;
    try {
        edit.DeleteInDocument(index);
        std::cout << "Ok\n";
    } catch (std::logic_error& err) {
```

```
                std::cout << err.what() << "\n";
            }
        } else if (action == 'f') {
            edit.PrintDocument();
        } else if (action == 'g') {
            try {
                edit.Undo();
            } catch (std::logic_error& e) {
                std::cout << e.what();
            }
        }  else if (action == 'h') {
                break;
        }
        else {
            std::cout << "Unknown command\n";
        }
    }
    return 0;
}
```

Makefile:

```
all:
	g++ main.cpp hexagon.cpp pentagon.cpp octagon.cpp point.cpp factory.cpp editor.cpp document.cpp command.cpp -o lab6
```

2. Ссылка на репозиторий на GitHub.

https://github.com/a1dv/oop_exercise_07.git

3. Набор тестов.

test_01.txt:

```
a out.txt
d pentagon 0 0 2 0 2 2 1 3 0 2
d hexagon 0 0 1 -1 2 0 2 2 1 3 0 2
d octagon 0 0 1 -1 2 0 3 1 2 2 1 3 0 2 -1 1
f
e 2
f
g
f
c
b in.txt
f
h
```

4. Результаты выполнения тестов.

test_01.result:
Document created
Ok
Ok
Ok
Coordinates are:
{
0 0
2 0
2 2
1 3
0 2
}
Coordinates are:
{
0 0
1 -1
2 0
2 2
1 3
0 2
}
Coordinates are:
{
0 0
1 -1
2 0
3 1
2 2
1 3
0 2
-1 1
}
Ok
Coordinates are:
{
0 0
2 0
2 2
1 3
0 2
}
Coordinates are:
{
0 0

1 -1
2 0
2 2
1 3
0 2
}
Coordinates are:
{
0 0
2 0
2 2
1 3
0 2
}
Coordinates are:
{
0 0
1 -1
2 0
2 2
1 3
0 2
}
Coordinates are:
{
0 0
1 -1
2 0
3 1
2 2
1 3
0 2
-1 1
}
Document saved
Document loaded
Coordinates are:
{
1 1
1 1
1 1
1 1
1 1
}

5. Объяснение результатов работы программы.

Пользователь вводит команды в терминале. В программе реализованы функции создания нового документа, чтения из файла и запись в него, добавление и удаление фигур. Так же реализована функция undo, отменяющая добавление или удаление фигуры.

6. Вывод.

Изучил проектирование структуры классов. Спроектировал простейший текстовый редактор.