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:	Касимов М.М.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	6
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

Figure.h:

```
#ifndef OOP7_FIGURE_H
#define OOP7_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 square() const = 0;
    virtual ~figure() = default;
};
```

Octagon.cpp:

```
#ifndef OOP7_OCTAGON_H
#define OOP7_OCTAGON_H


#include "figure.h"

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


#endif //OOP7_OCTAGON_H
```

Octagon.h:

```
#ifndef OOP7_OCTAGON_H
```

```
#define OOP7_OCTAGON_H


#include "figure.h"

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


#endif //OOP7_OCTAGON_H

#endif //OOP_EXERCISE_03_8GON_H
```

Hexagon.h

```
#ifndef OOP7_HEXAGON_H
#define OOP7_HEXAGON_H

#include "figure.h"

struct hexagon : figure{
private:
    point a1,a2,a3,a4,a5,a6;
public:
    point center() const override ;
    void print(std::ostream&) const  override ;
    void printFile(std::ofstream&) const  override ;
    double square() const override ;
    hexagon() = default;
    hexagon(std::istream& is);
    hexagon(std::ifstream& is);
};


#endif //OOP7_HEXAGON_H
```

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& os) const {
    os << "hexagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 <<
'\n' << a6 << "\n";
}

void hexagon::printFile(std::ofstream &of) const {
    of << "hexagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 <<
'\n' << a6 << "\n";
}

double hexagon::square() const {
    return (-0.5) * ((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& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;
}
```

Pentagon.h:

```
#include "pentagon.h"

#include <cmath>
```

```
#include "point.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& os) const {
    os << "pentagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 <<
'\n';
}

void pentagon::printFile(std::ofstream& of) const {
    of << "pentagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 <<
'\n';
}

double pentagon::square() const{
    //метод Гаусса(алгоритм шнурования)
    return (-0.5) * ((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;
}
```

Pentagon.cpp

```
#include "pentagon.h"

#include <cmath>
#include "point.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& os) const {
    os << "pentagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 <<
'\n';
}

void pentagon::printFile(std::ofstream& of) const {
    of << "pentagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 <<
'\n';
}

double pentagon::square() const{
    //метод Гаусса(алгоритм шнурования)
    return (-0.5) * ((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;
}
```

Point.cpp:

```
#include "point.h"

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;
}
```

Point.h:

```
#ifndef OOP_EXERCISE_03_POINT_H
#define OOP_EXERCISE_03_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 //OOP_EXERCISE_03_POINT_H
```

Main.cpp:

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

void help() {
    std::cout << "help\n"
                 "create\n"
                 "load\n"
                 "save\n"
                 "add\n"
                 "remove\n"
                 "print\n"
                 "undo\n"
                 "exit\n";
}

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.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::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;
    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;
}
```

Command.h

```
#ifndef OOP7_COMMAND_H
#define OOP7_COMMAND_H
#include "document.h"

struct Acommand {
    virtual ~Acommand() = default;
    virtual void UnExecute() = 0;
    void SetDocument(std::shared_ptr<document>& doc) {
        doc_ = doc;
    }

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

```
struct InsertCommand : public Acommand {
public:
    void UnExecute() override {
        doc_->RemoveLast();
    }
};

struct DeleteCommand : public Acommand {
public:
    DeleteCommand(std::shared_ptr<figure>& newFigure, uint32_t newIndex) {
        figure_ = newFigure;
        index_ = newIndex;
    }
    void UnExecute() override {
        doc_->InsertIndex(figure_,index_);
    }

private:
    std::shared_ptr<figure> figure_;
    uint32_t index_;
};
#endif //OOP7_COMMAND_H
```

Document.h

```
#ifndef OOP7_DOCUMENT_H
#define OOP7_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 {
        if (buffer_.empty()) {
            std::cout << "Buffer is empty\n";
        }
        for (auto elem : buffer_) {
            elem->print(std::cout);
```

```
    }
};

document(std::string& newName): name_(newName), factory_(), buffer_(0) {};

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

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

void 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 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> GetFigure(uint32_t index) {
    return buffer_[index];
}

void 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 GetName() {
        return this->name_;
    }

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

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

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

    void InsertIndex(std::shared_ptr<figure>& newFigure, uint32_t index) {
        /*buffer_.push_back(newFigure);
        for (int i = buffer_.size() - 1; i != index ; --i){
            std::shared_ptr<figure> tmp = buffer_[i];
            buffer_[i] = buffer_[i - 1];
            buffer_ [i - 1] = tmp;
        }*/
        buffer_.insert(buffer_.begin() + index, newFigure);
    }
};


#endif //OOP7_DOCUMENT_H
```

Editor.h

```
#ifndef OOP7_EDITOR_H
#define OOP7_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() {
      if (doc_ == nullptr) {
         std::cout << "No document!\n";
         return;
      }
      doc_->Print();
   }

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

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

   editor() : doc_(nullptr), history_()
   {
   }

   void 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_->Insert(newFigure);
        command -> SetDocument(doc_);
        history_.push(command);
    }

    void 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_->Erase(index);
        command -> SetDocument(doc_);
        history_.push(command);
    }

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

    void LoadDocument(std::string& name) {
        doc_ = std::make_shared<document>(name);
        doc_->Load(name);
        while (!history_.empty()){
            history_.pop();
        }
    }

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

```
        history_.pop();
    }

};

#endif //OOP7_EDITOR_H
```

Factory.h

```
#ifndef OOP7_FACTORY_H
#define OOP7_FACTORY_H

#include <memory>
#include <iostream>
#include <fstream>
#include "hexagon.h"
#include "octagon.h"
#include "pentagon.h"
#include <string>

struct factory {
    std::shared_ptr<figure> FigureCreate(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> FigureCreateFile(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");
        }
    };

};


#endif //OOP7_FACTORY_H
```

2. Ссылка на репозиторий на GitHub.

https://github.com/magomed2000kasimov/oop_exercise_07

3. Набор тестов.

test_01.test:

```
add pentagon 1 1 1 1 1 1 1 1 1 1
remove 0
undo
save
print
exit
```

test_02.test:

```
create buffer1.txt
add pentagon 5 5 5 5 5 5 5 5 5 5
add hexagon 6 6 6 6 6 6 6 6 6 6 6 6
add gldlfglg
add octagon 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
print
save
remove 1
remove 0
undo
undo
add pentagon 0 0 0 0 0 0 0 0 0 0
undo
undo
print
undo
print
undo
print
undo
exit
```

4. Результаты выполнения тестов.

test_01.res:

No document!
Ok
Enter index
No document!
Ok
History is empty
No document!
Not save document
No document!

test_02.result:

Enter name of new document
Document create
Ok
Ok
There is no such figure

Ok
Ok
pentagon
5 5
5 5
5 5
5 5
5 5
hexagon
6 6
6 6
6 6
6 6
6 6
6 6
octagon
8 8
8 8
8 8
8 8
8 8
8 8
8 8
8 8

save document
Enter index
Ok
Enter index
Ok
Ok
pentagon
5 5
5 5
5 5
5 5
5 5
hexagon
6 6
6 6
6 6
6 6
6 6
6 6
pentagon
5 5
5 5
5 5
5 5
5 5
Buffer is empty
History is empty

5. Объяснение результатов работы программы.

1) Метод center() const возвращает точку с x –деление суммы иксов всех точек данной фигуры на их количество, y – аналогично x.

2) Метод print(std::ostream&) const печатает координаты всех точек данной фигуры.
3) Метод square() const вычисляет площадь данной фигуры по методу Гаусса (формула землемера, метод шунтирования) и возвращает это значение.
4) Удаление в main.cpp фигуры из вектора по индексу происходит:
- удаляается фигура с помощью delete.
- элементы вектора сдвигаются влево циклом for , чтобы закрыть индекс удаленного элемента.
- используется метод вектора pop_back();.

6. Вывод.

Выполняя данную лабораторную, я получил опыт работы с наследованием в C++. Узнал о виртуальных функциях, абстрактных классах. Также я вынес отдельные классы, которые отвечают за что-то свое, реализовал импорт и экспорт документа в файл.

7. Литература.

1)лекции по ООП МАИ.
2) Г.Шилдт «C++».