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:	Ли А. И.
Группа:	М80-208Б-18
Преподаватель:	Журавлев А.А.
Вариант:	13
Оценка:	
Дата:	

Москва
2019

1. Задание

Спроектировать простейший графический векторный редактор, согласно варианту задания (квадрат, прямоугольник, трапеция).

Требование к функционалу редактора:

- создание нового документа
- импорт документа из файла
- экспорт документа в файл
- создание графического примитива (согласно варианту задания)
- удаление графического примитива
- отображение документа на экране (печать перечня графических объектов и их характеристик)
- реализовать операцию undo, отменяющую последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

Требования к реализации:

- Создание графических примитивов необходимо вынести в отдельный класс – Factory.
- Сделать упор на использовании полиморфизма при работе с фигурами;
- Взаимодействие с пользователем (ввод команд) реализовать в функции main;

2. Адрес репозитория на GitHub

https://github.com/elips0n/oop_exercise_07

3. Код программы на C++

main.cpp

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

void menu() {
   std::cout << "menu\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 == "menu") {
            menu();
        } 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;
}
```

point.h

```
#ifndef OOP_POINT_H
#define OOP_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);
```

```
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;
}
#endif
```

command.h

```
#ifndef OOP_COMMAND_H
#define OOP_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_;
};
//=====================================realize============================
==
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_);
}
#endif //OOP_COMMAND_H

document.h
#ifndef OOP_DOCUMENT_H
#define OOP_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 ;

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

    void Insert(std::shared_ptr<figure>& ptr);


    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);
};

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::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);
}
#endif
```

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();

    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();

};
//==================================realize=============================
===

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();
}
#endif //OOP7_EDITOR_H
```

factory.h

```
#ifndef OOP_FACTORY_H
#define OOP_FACTORY_H

#include <memory>
```

```
#include <iostream>
#include <fstream>
#include "square.h"
#include "rectangle.h"
#include "trapez.h"
#include <string>

struct factory {

   std::shared_ptr<figure> FigureCreate(std::istream &is) {
      std::string name;
      is >> name;
      if ( name == "rectangle" ) {
         return std::shared_ptr<figure> ( new Rectangle(is));
      } else if ( name == "trapez") {
         return std::shared_ptr<figure> ( new Trapez(is));
      } else if ( name == "square") {
         return std::shared_ptr<figure> ( new Square(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 == "rectangle" ) {
         return std::shared_ptr<figure> ( new Rectangle(is));
      } else if ( name == "trapez") {
         return std::shared_ptr<figure> ( new Trapez(is));
      } else if ( name == "square") {
         return std::shared_ptr<figure> ( new Square(is));
      } else {
         throw std::logic_error("There is no such figure\n");
      }
   }

};


#endif //OOP_FACTORY_H
```

figure.h

```
#ifndef OOP_FIGURE_H
#define OOP_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 //OOP_FIGURE_H
```

square.h

```
#ifndef OOP_SQUARE_H
#define OOP_SQUARE_H

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

struct Square : figure {
public:
    point a1, a2, a3, a4;


    point 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 print(std::ostream &os) const {
        os << "square "<< a1 <<" "<< a2 <<" "<< a3 <<" "<< a4  <<"\n";
    }
    void printFile(std::ofstream &of) const {
        of << "square "<< a1 <<" "<< a2 <<" "<< a3 <<" "<< a4  <<"\n";
    }

    double area() const {
        return (-0.5) * ((a1.x * a2.y + a2.x * a3.y + a3.x * a4.y + a4.x * a4.y) -
                         (a1.y * a2.x + a2.y * a3.x + a3.y * a4.x + a4.y * a1.x));
    }

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

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

```
#endif //OOP_SQUARE_H
```

rectangle.h

```
#ifndef OOP_RECTANGLE_H
#define OOP_RECTANGLE_H

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

struct Rectangle : figure {

    point a1, a2, a3, a4;

    point 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 print(std::ostream &os) const {
        os << "rectangle "<< a1 <<" "<< a2 <<" "<< a3 <<" "<< a4  <<"\n";
    }

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

    double area() const {
        return (-0.5) * ((a1.x * a2.y + a2.x * a3.y + a3.x * a4.y + a4.x * a1.y) -
                         (a1.y * a2.x + a2.y * a3.x + a3.y * a4.x + a4.y * a1.x));
    }

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

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

#endif //OOP_RECTANGLE_H
```

trepez.h

```
#ifndef OOP_TRAPEZ_H
#define OOP_TRAPEZ_H
```

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

struct Trapez : figure{

   point a1,a2,a3,a4;

   point 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 print(std::ostream& os) const {
      os << "trapez "<< a1 <<" "<< a2 <<" "<< a3 <<" "<< a4  <<"\n";
   }

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

   double area() const {
      return (-0.5) * ((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a1.y) - ( a1.y*a2.x + a2.y*a3.x +
a3.y*a4.x + a4.y*a1.x ));
   }

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

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


#endif //OOP_TRAPEZ_H
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 3.10.2)
project(oop_exercise_07)

set(CMAKE_CXX_STANDARD 17)

add_executable(oop_exercise_07
       main.cpp
       point.h
       trapez.h
       figure.h
```

rectangle.h
square.h
document.h
factory.h
command.h
editor.h)

4. Объяснение результатов работы программы - вывод

В main.cpp посредством editor.h, выступающим в роли редактора, осуществляются действия с документом: его создание, удаление, сохранение и тд. В command.h реализованы вставка, удаление и обратное выполнение команды, необходимые для реализации undo; в document.h — действия с документом, в factory.h реализовано создание фигур квадрат, прямоугольник и трапеция.

В ходе лабораторной работы мною были усовершенствованы навыки объектно-ориентированного программирования, укреплены знания о наследовании, полиморфизме, классах.