Московский Авиационный Институт

(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Лабораторной работе № 07
по курсу «Объектно-ориентированное программирование»

Тема:
« Проектирование структуры классов»

Студент:	Пшеницын А. А.
Группа:	М80-208Б-18
Преподаватель:	Журавлев А. А.
Вариант:	17
Оценка:	
Дата:	

Москва
2019

Цель:

Цель:
• Получение практических навыков в хороших практиках проектирования структуры классов приложения;

Задание

Спроектировать простейший графический векторный редактор.
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;

Код

main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <memory>
#include "figure.h"
#include "factory.h"
#include "command.h"

int main() {
    factory fact;
    std::unique_ptr<doc_oper> doc = std::make_unique<doc_oper>();
    while(1){
        std::string command;
        std::cin >> command;
        if(command == "save"){
            std::string path;
            std::cin >> path;
            std::ofstream os(path);
            doc -> save(os);
            os.close();
        }else if(command=="load"){
            std::string path;
            std::cin >> path;
            std::ifstream is(path);
            if(is) {
                doc -> load(is);
            }else {
```

```
            std::cout << "No such file\n";
        }
        is.close();
    }else if(command=="add"){
        size_t id;
        std::cin >> id;
        doc -> add(std::cin, id);
    }else if(command=="erase"){
        size_t id;
        std::cin >> id;
        doc -> erase(id);
    }else if(command=="print"){
        doc -> print(std::cout);
    }else if(command == "undo"){
        doc -> undo();
    } else if(command == "exit"){
        break;
    } else{
        std::cout << "EROOR";
    }
  }
  return 0;
}
```

document.h

```
#ifndef D_DOCUMENT_H_
#define D_DOCUMENT_H_


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



struct document{
public:
  document() = default;

  void save_fig(std::ostream& os) const {
    for (size_t i = 0; i < figs.size(); ++i) {
      figs[i] -> print_dop(os);
    }
  }

  void load_fig(std::ifstream& is){
```

```
        while(!is.eof()){
            std::shared_ptr<fig> ptr;
            ptr = fact.fig_create_file(is);
            if(ptr == nullptr){
                break;
            }
            figs.push_back(ptr);
        }
    }


    void print_fig(std::ostream& os) const {
        if(figs.size() == 0) {
            os << "Empty\n";
            return;
        }
        for (size_t i = 0; i < figs.size(); ++i) {
            os << "figure " << i << ":" << "\n";
            figs[i] -> print(os);
            os << "center :" << "[" << figs[i] -> center().x << ", " << figs[i] -> center().y << "]" <<
"\n";
            os << "square :" << figs[i] -> square() << "\n";
        }
    }

    void add_fig(std::istream& is, size_t id){
        std::shared_ptr<fig> ptr = fact.fig_create(is);
        figs.insert(figs.begin() + id, ptr);
    }

    void erase_fig(size_t id){
        figs.erase(figs.begin() + id);
    }

    std::shared_ptr<fig> get_fig(size_t id) {
        if (id >= figs.size()) {
            return nullptr;
        }
        return figs[id];
    }

    void add_fig_dop(std::shared_ptr<fig>& ptr, size_t id){
        figs.insert(figs.begin() + id, ptr);
    }



private:
    factory fact;
    std::vector<std::shared_ptr<fig>> figs;
};
```

```
#endif
```

command.h

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

#include "document.h"

struct doc_oper : private document{
private:
    struct command{
        size_t id;
        std::shared_ptr<fig> ptr;
        virtual void undo(document &doc) = 0;
    };

    std::vector<std::shared_ptr<command>> opers;

    struct add_com : public command{
        void undo(document &doc) override {
            doc.erase_fig(id);
        }
    };

    struct remove_com : public command{
        void undo(document &doc) override {
            doc.add_fig_dop(ptr, id);
        }
    };

public:

    void add(std::istream& is, size_t id){
        add_fig(is, id);
        std::shared_ptr<fig> fig = get_fig(id);
        std::shared_ptr<add_com> op = std::make_shared<add_com>();
        op -> id = id;
        op -> ptr = fig;
        opers.push_back(op);
    }

    void erase(size_t id){
        std::shared_ptr<fig> fig = get_fig(id);
        erase_fig(id);
        std::shared_ptr<remove_com> op = std::make_shared<remove_com>();
        op -> id = id;
        op -> ptr = fig;
        opers.push_back(op);
```

```
    }

    void undo(){
        if(opers.size() == 0){
            std::cout << "EMPTY";
            return;
        }
        opers[opers.size() - 1] -> undo(*this);
        opers.pop_back();
    }

    void save(std::ostream& os){
        save_fig(os);
    }

    void load(std::ifstream& is){
        load_fig(is);
    }

    void print(std::ostream& os){
        print_fig(os);
    }
};

#endif //OOP7_COMMAND_H
```

factory.h

```
#ifndef D_FACTORY_H
#define D_FACTORY_H

#include <memory>
#include <iostream>
#include <fstream>
#include "trigon.h"
#include "rectangle.h"
#include "quadrate.h"
#include <string>

struct factory {
    std::shared_ptr<fig> fig_create(std::istream& is);
    std::shared_ptr<fig> fig_create_file(std::ifstream& is);

};


std::shared_ptr<fig> factory::fig_create(std::istream &is) {
    try {
        std::string comm;
        is >> comm;
        if (comm == "trigon") {
            return std::shared_ptr<fig>(new trigon(is));
        } else if (comm == "quadrate") {
```

```
            return std::shared_ptr<fig>(new quadrate(is));
        } else if (comm == "rectangle") {
            return std::shared_ptr<fig>(new rectangle(is));
        } else {
            throw "This is not a figure";
        }
    } catch (const char* f){
        std::cout << f << "\n";
        return nullptr;
    }
}

std::shared_ptr<fig> factory::fig_create_file(std::ifstream &is) {
    try {
        std::string comm;
        is >> comm;
        if (comm == "trigon") {
            return std::shared_ptr<fig>(new trigon(is));
        } else if (comm == "quadrate") {
            return std::shared_ptr<fig>(new quadrate(is));
        } else if (comm == "rectangle") {
            return std::shared_ptr<fig>(new rectangle(is));
        } else {
            throw " ";
        }
    } catch (const char* f){
        std::cout << f << "\n";
        return nullptr;
    }
}


#endif
```

Ссылка на репозиторий на GitHub

https://github.com/AlexPshen/oop_exercise_06.git

Тесты

test_01.txt

```
add 0 trigon 0 0 0 1 1 0
print
add 1 quadrate 0 0 1 1 0 1 1 0
print
undo
print
save file
exit
```

res_01.txt

```
add 0 trigon 0 0 0 1 1 0
print
```

figure 0:
trigon: (0, 0) (0, 1) (1, 0)
center :[0.333333, 0.333333]
square :0.5
add 1 quadrate 0 0 1 1 0 1 1 0
print
figure 0:
trigon: (0, 0) (0, 1) (1, 0)
center :[0.333333, 0.333333]
square :0.5
figure 1:
quadrate: (0, 0) (1, 1) (0, 1) (1, 0)
center :[0.5, 0.5]
square :1
undo
print
figure 0:
trigon: (0, 0) (0, 1) (1, 0)
center :[0.333333, 0.333333]
square :0.5
save file
exit

test_02.txt

load file
print
add 0 rectangle 0 0 0 1 1 0 1 1
print
erase
0
print
indo
print
save

Объяснение результатов работы программы

Данная программа предназначена для обработки фигур, таких как треугольник, квадрат, прямоугольник. Опишем команды-обработчики:

- save – сохранение фигур в файл
- load – загрузка фигур из файла
- add – добавление фигуры
- erase – удаление фигуры
- print – печать характеристик всех фигур
- undo – отмена последней команды (может отменять только add и erase)
- exit – конец программы

Вывод

В данной лабораторной работе мы попробовали спроектировать нормальную структуру классов. Данная работа направлена на приучение к написанию хороших классов, которые должны иметь наименьшее возможное количество связей между собой. Это делается для того, чтобы было проще читать код, а также для более легкого редактирования данного кода впоследствии.