Московский авиационный институт
(Национальный исследовательский университет)

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 7

Тема: Проектирование структуры классов

Студент: Попов Илья Павлович
Группа: 80-206
Преподаватель: Чернышов Л.Н.
Дата: 16.12
Оценка:

Москва, 2021

1. Постановка задачи

Спроектировать простейший «графический» векторный редактор.

Требования 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 в std::cout)
- 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;

Вариант № 22.
Фигуры: 5 - угольник 6 - угольник 8 - угольник

2. Описание программы

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

Программа состоит из 16 файлов:

1. figure.h - содержит реализацию родительского класса фигур;
   pentagon.cpp и pentagon.h — реализация класса 5-угольника;
   hexagon.cpp и hexagon.h — реализация класса 6-угольника;
   octagon.cpp и octagon.h — реализация класса 8-угольника;
2. factory.cpp и factory.h — класс для создания графических примитивов фигур ;
3. document.cpp и document.h — реализация класса, содержащего непосредственно методы работы с файлом;
4. editor.cpp и editor.h — реализация класса, необходимого согласованной работы document и command;
5. command.cpp и command.h — реализация класса, реализующего перемещение по истории изменения документа, необходимое для операции undo;
6. main.cpp - файл с взаимодействием с пользователем.

Пользователю доступны команды:

1. create <path> — создание файла;
2. save — сохранить хранящиеся в document фигуры созданный ранее файл
3. add <hexagon, pentagon trapezoid> — добавить фигуру

4. remove <index> — удалить фигуру по индексу
5. print — печать document
6. undo — отмена последнего действия
7. load <path> — загрузить данные из указанного файла

3. Набор тестов

Тест №1 (демонстрирует работу программы при корректно введенных входных данных)

```
_____
Available commands:
create <path> - create a new file
save - save data to a file
add <hexagon, pentagon trapezoid>
remove <index> - remove a figure by index
print
undo - undo the last action
load <path> - load data from a file
_____

create test1.txt
Create success

_____
add pentagon
side = 5

Add success

_____
add hexagon
side = 6

Add success

_____
add octagon
side = 8

Add success

_____
print
Pentagon
Side = 5
Area:43.0119
Hexagon
Side = 6
Area:93.5307
Octagon
Side = 8
Area:309.019
_____
remove 0
Remove success
```

---
print
Hexagon
Side = 6
Area:93.5307
Octagon
Side = 8
Area:309.019

---
undo
Undo success

---
print
Pentagon
Side = 5
Area:43.0119
Hexagon
Side = 6
Area:93.5307
Octagon
Side = 8
Area:309.019

---
save
Save success

---
create test2.txt
Create success

---
add pentagon
side = 55

Add success

---
print
Pentagon
Side = 55
Area:5204.44

---
save
Save success

---
load test1.txt
Load success

---
print
Pentagon
Side = 5
Area:43.0119
Hexagon
Side = 6
Area:93.5307
Octagon

Side = 8
Area:309.019

---

exit

Тест №2 (демонстрирует работу программы при некорректно введенных входных данных)

Available commands:
create <path> - create a new file
save - save data to a file
add <hexagon, pentagon trapezoid>
remove <index> - remove a figure by index
print
undo - undo the last action
load <path> - load data from a file

---

create test1.txt
Create success

---

dsafsd
Error input!

---

add fdsf
There is no such figure!

---

add pentagon
side = sfd

Pentagon input error!
Repeat input!
side = 3

Add success!

---

remove dsfsd
Index error!

---

remove 333
Index error!

---

undo
Undo success

---

undo
History is empty!

```
load safaf
File is not opened!


_____________________________________
exit
```

4. Результаты выполнения тестов

Представлены выше, с целью упростить прочтение.

5. Листинг программы

figure.h

```
#ifndef _FIGURE_H_
#define _FIGURE_H_

#include <iostream>
#include <fstream>
#include <cmath>
#include <string>

using namespace std;

struct figure {
   virtual void print(ostream&) const = 0 ;
   virtual void printFile(ofstream&) const = 0 ;
   virtual double area() const = 0;
   virtual ~figure() = default;
   virtual bool is_fig_num(const string& s) {
      bool point = false;
      for (int i = 0; i < s.length(); ++i) {
         if (s[i] == '.') {
            if ((i == 0 || i == s.length() - 1) || point) {
               return false;
            }
            else {
               point = true;
            }
         }
         else if (s[i] < '0' || s[i] > '9') { return false; }
      }
      return true;
   }
};

#endif
```

pentagon.cpp

```
#include "pentagon.h"
```

```
void pentagon::print(ostream& os) const {
    os << "Pentagon\n";
    os << "Side = " << side;
    os << "\nArea:" << area() << '\n';
}

void pentagon::printFile(ofstream &of) const {
    of << "pentagon\n"<< side << '\n';
}

double pentagon::area() const {
    long double cur = (double)pow(side, 2) / (double)4;
    long double res = cur * sqrt(25 + 10 * sqrt(5));
    return res;
}

pentagon::pentagon(istream& is) {
    string str;
    cout << "side = ";  is >> str; cout << endl;

    while (true) {
        if (is_fig_num(str)) {
            side = stod(str);
            break;
        }
        cout << "Pentagon input error!\n";
        cout << "Repeat input!\n";
        cout << "side = ";  is >> str; cout << endl;
    }
}

pentagon::pentagon(ifstream& is) {
    is >> side;
}
```

pentagon.h

```
#ifndef _PENTAGON_H_
#define _PENTAGON_H_

#include "figure.h"
using namespace std;

struct pentagon : figure{
private:
    double side;
public:
    void print(ostream&) const override ;
    void printFile(ofstream&) const override ;
    double area() const override ;
    pentagon() = default;
    pentagon(istream& is);
    pentagon(ifstream& is);
};

#endif
```

hexagon.cpp

```
#include "hexagon.h"

void hexagon::print(ostream& os) const {
    os << "Hexagon\n";
    os << "Side = " << side;
    os << "\nArea:" << area() << '\n';
}

void hexagon::printFile(ofstream& of) const {
    of << "hexagon\n" << side << '\n';
}

double hexagon::area() const{
    long double res = (double)((3 * sqrt(3)) / 2) * pow(side, 2);
    return res;
}

hexagon::hexagon(istream& is) {
    string str;
    cout << "side = ";  is >> str; cout << endl;

    while (true) {
        if (is_fig_num(str)) {
            side = stod(str);
            break;
        }
        cout << "Hexagon input error!\n";
        cout << "Repeat input!\n";
        cout << "side = ";  is >> str; cout << endl;
    }
}

hexagon::hexagon(ifstream& is) {
    is >> side;
}
```

hexagon.h

```
#ifndef _HEXAGON_H_
#define _HEXAGON_H_

#include "figure.h"
using namespace std;

struct hexagon : figure {
private:
    double side;
public:
    void print(ostream&) const  override ;
    void printFile(ofstream&) const override ;
    double area() const override ;
    hexagon() = default;
    hexagon(istream& is);
    hexagon(ifstream& is);
};
```

```
#endif
```

octagon.cpp

```
#include "octagon.h"

void octagon::print(ostream& os) const {
    os << "Octagon\n";
    os << "Side = " << side;
    os << "\nArea:" << area() << '\n';
}

void octagon::printFile(ofstream& of) const {
    of << "octagon\n" << side << '\n';
}

double octagon::area() const {
    long double res = (2 + 2 * sqrt(2)) * pow(side, 2);
    return res;
}

octagon::octagon(istream& is) {
    string str;
    cout << "side = ";  is >> str; cout << endl;

    while (true) {
        if (is_fig_num(str)) {
            side = stod(str);
            break;
        }
        cout << "Octagon input error!\n";
        cout << "Repeat input!\n";
        cout << "side = ";  is >> str; cout << endl;
    }
}

octagon::octagon(ifstream& is) {
    is >> side;
}
```

octagon.h

```
#ifndef _OCTAGON_H_
#define _OCTAGON_H_

#include "figure.h"
using namespace std;

struct octagon : figure {
private:
    double side;
public:
    void print(ostream&) const override;
    void printFile(ofstream&) const override;
    double area() const override;
    octagon() = default;
```

```
    octagon(istream& is);
    octagon(ifstream& is);
};

#endif
```

factory.cpp

```
#include "factory.h"

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

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

factory.h

```
#ifndef _FACTORY_H_
#define _FACTORY_H_

#include <memory>
#include <iostream>
#include <fstream>
#include <string>

#include "hexagon.h"
#include "pentagon.h"
#include "octagon.h"

struct factory {
    shared_ptr<figure> FigureCreate(istream& is);
    shared_ptr<figure> FigureCreateFile(ifstream& is);
};
```

```
#endif
```

document.cpp

```
#include "document.h"

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

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

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

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

void document::Load(const string &filename) {
    ifstream fin;
    fin.open(filename);
    if (!fin.is_open()) {
        throw 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;
}

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

void document::Erase(uint32_t index) {
    if ( index >= buffer_.size()) {
        throw logic_error("Out of bounds!\n");
```

```
    }
    buffer_[index] = nullptr;
    for (; index < buffer_.size() - 1; ++index) {
        buffer_[index] = buffer_[index + 1];
    }
    buffer_.pop_back();
}

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

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

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

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

document.h

```
#ifndef _DOCUMENT_H_
#define _DOCUMENT_H_

#include <cstdint>
#include <memory>
#include <string>
#include <algorithm>
#include <vector>

#include "figure.h"
#include "factory.h"

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

```
    vector<shared_ptr<figure>> buffer_;
    void RemoveLast();
    void InsertIndex(shared_ptr<figure>& newFigure, uint32_t index);
};


#endif
```

editor.cpp

```
#include "editor.h"

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

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

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

void editor::InsertInDocument(shared_ptr<figure> &newFigure) {
    if (doc_ == nullptr) {
        cout << "No document!\n\n";
        return;
    }
    shared_ptr<Acommand> command = shared_ptr<Acommand>(new InsertCommand(doc_));
    doc_->Insert(newFigure);
    history_.push(command);
    cout << "Add success!\n\n";
}

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

void editor::SaveDocument() {
    if (doc_ == nullptr) {
        cout << "No document!\n\n";
```

```
        return;
    }
    string saveName = doc_->GetName();
    doc_ ->Save(saveName);
    cout << "Save success\n\n";
}

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

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

editor.h

```
#ifndef _D_EDITOR_H_
#define _D_EDITOR_H_

#include <stack>
#include "command.h"

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

#endif
```

command.cpp

```
#include "command.h"

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

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

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

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

command.h

```
#ifndef _COMMAND_H_
#define _COMMAND_H_

#include "document.h"

struct Acommand {
public:
   virtual ~Acommand() = default;
   virtual void UnExecute() = 0;
protected:
   shared_ptr<document> doc_;
};

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

struct DeleteCommand : public Acommand {
public:
   DeleteCommand(shared_ptr<figure>& newFigure, uint32_t newIndex,shared_ptr<document>& doc);
   void UnExecute() override;
private:
   shared_ptr<figure> figure_;
   uint32_t index_;
};
#endif
```

main.cpp

```
//Попов Илья Павлович
//M80-206Б-20

//Лабораторная работа №7
//Вариант № 22.
//Фигуры: 5 - угольник 6 - угольник 8 - угольник


//Взаимодействуем с документом(document.h) происходит посредством editor.h, в котором лежат ссылки на документ(вектор фигур) и стек команд
//Перенаправление записи и вывода из файла происходят в factory.h
//Команды отслеживаются в command.h, что необходимо для реализации undo

/*
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 в std::cout)
   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;
*/

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

void usage() {
   cout << "_____" << endl;
   cout << "Available commands:\n";
   cout << "create <path> - create a new file\n";
   cout << "save - save data to a file\n";
   cout << "add <hexagon, pentagon trapezoid>\n";
   cout << "remove <index> - remove a figure by index\n";
   cout << "print\n";
   cout << "undo - undo the last action\n";
   cout << "load <path> - load data from a file\n";
   cout << "_____" << endl << endl;
}

bool is_number(const string& s) {
   bool point = false;
   for (int i = 0; i < s.length(); ++i) {
      if (s[i] == '-' && i == 0) {
         continue;
      }
      else if (s[i] == '.') {
         if ((i == 0 || i == s.length() - 1) || point) {
            return false;
         }
```

```
            else {
                point = true;
            }
        }
        else if (s[i] < '0' || s[i] > '9') { return false; }
    }
    return true;
}

void create(editor& edit) {
    string tmp;
    cin >> tmp;
    edit.CreateDocument(tmp);
    cout << "Create success\n\n";
}

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

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

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

void remove(editor& edit) {
    try {
        string str;
        cin >> str;
        if (!is_number(str)) {
            cout << "Index error!\n\n";
        }
        else {
            uint32_t index = stoi(str);
            edit.DeleteInDocument(index);
        }

    } catch (logic_error& err) {
        cout << err.what() << "\n\n";
```

```
    }
}

int main() {
    usage();
    editor edit;
    string cmd;
    while (true) {
        cin >> cmd;
        if (cmd == "help") {
            usage();
        } else if (cmd == "create") {
            create(edit);
        } else if (cmd == "load") {
            load(edit);
        } else if (cmd == "save") {
            save(edit);
        } else if (cmd == "exit") {
            break;
        } else if (cmd == "add") {
            add(edit);
        } else if (cmd == "remove") {
            remove(edit);
        } else if (cmd == "print") {
            edit.PrintDocument();
        } else if (cmd == "undo") {
            try {
                edit.Undo();
                cout << "Undo success\n\n";
            } catch (logic_error& err) {
                cout << err.what() << "\n\n";
            }
        } else {
            cout << "Error input!\n\n";
        }
        cout << "_____" << endl;
    }
    return 0;
}
```

6. Вывод

Почти все серьезные коммерческие проекты используют работу с файлами, в парадигме объектно-ориентированного программирования реализация этого механизма представлена довольно просто и ознакомиться с ней должен каждый программист.

Также в ходе выполнения лабораторной работы была решена непростая задача — реализована функция undo, которая отменяет последнее удаление и добавление фигуры. Данную задачу удалось решить путем реализации класса command, который хранит историю(стек команд) изменения document.

ЛИТЕРАТУРА

1. Работа с файлами язык программирования C++ [Электронный ресурс].

URL:http://cppstudio.com/post/446/ (дата обращения: 16.12.2021).
2. Базовый файловый вывод и ввод в C++ [Электронный ресурс].
URL:https://ravesli.com/urok-212-bazovyj-fajlovyj-vvod-vyvod/ (дата обращения: 16.12.2021).