**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 7**

Тема: Проектирование структуры классов

Студент: Хренов Геннадий

Группа: 80-207

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. Постановка задачи

Спроектировать простейший графический векторный редактор со следующими требованиями: 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, которая отменяет последнее сделанное действие. Должно действовать для операций добавления/удаления фигур.

1. Описание программы

Программа позволяет создавать графические примитивы фигур квадрат, прямоугольник, трапеция, записывать и считывать их из файла, удалять, а также отменять действие последней операции с помощью команды undo. Запись производится в бинарный файл. Все операции с файлами производит редактор. Если происходит выход с рабочего документа, редактор запрашивает необходимость сохранения данного файла, а затем продолжает работу. Также редактор хранит историю команд, и в случае необходимости может удалить последнюю.

1. Набор testcases
2. Create 1.bin Add S 1 1 2 1 2 2 2 1 Add R 0 0 3 0 3 4 0 4 Remove 1 Remove 2 Print- проверка на вставку и удаление элементов
3. Create 2.bin Add S 1 1 2 1 2 2 2 1 Add T 0 0 1 1 2 1 3 0 Save 2.bin Remove 1 Load 2.bin Print - проверка на загрузку и выгрузку из файла
4. Load 2.bin Remove 1 Print Undo Print - проверка правильности работы Undo
5. Результаты выполнения тестов.

а) Create 1.bin

Document 1.bin is created

Add S 1 1 2 1 2 2 2 1

Figure is added

Add R 0 0 3 0 3 4 0 4

Figure is added

Remove 1

Primitive with 1 is removed

Remove 2

Primitive with 2 is removed

Print // фигуры успешно добавлены и удалены

b) Create 2.bin

Document 2.bin is created

Add S 1 1 2 1 2 2 2 1

Figure is added

Add T 0 0 1 1 2 1 3 0

Figure is added

Save 2.bin

Document save in file 2.bin

Remove 1

Primitive with 1 is removed

Load

Save old document? Yes/No

No

2.bin

Document loaded from file 2.bin

Print

ID: 1

Figure: Square

Coors:

A = (1.00, 1.00) B = (2.00, 1.00)

C = (2.00, 2.00) D = (2.00, 1.00)

ID: 2

Figure: Trape

Coors:

A = (0.00, 0.00) B = (1.00, 1.00)

C = (2.00, 1.00) D = (3.00, 0.00)// загрузка-выгрузка произошла успешно

c) Load 2.bin

Document loaded from file 2.bin

Remove 1

Primitive with 1 is removed

Print

ID: 2

Figure: Trape

Coors:

A = (0.00, 0.00) B = (1.00, 1.00)

C = (2.00, 1.00) D = (3.00, 0.00)

Undo

OK

Print

ID: 1

Figure: Square

Coors:

A = (1.00, 1.00) B = (2.00, 1.00)

C = (2.00, 2.00) D = (2.00, 1.00)

ID: 2

Figure: Trape

Coors:

A = (0.00, 0.00) B = (1.00, 1.00)

C = (2.00, 1.00) D = (3.00, 0.00) // удаленная фигура восстановилась

1. Листинг программы

**main.cpp**

/\*Хренов Геннадий M80-207Б

Спроектировать простейший графический векторный редактор

со следующими требованиями:

создание нового документа,

импорт документа из файла,

экспорт документа в файл,

создание графического примитива,

удаление графического примитива,

отображение документа на экран

\*/

#include <iostream>

#include "editor.h"

#include "figure.h"

void help() {

std::cout << "you can use:\n";

std::cout << "\tCreate\n";

std::cout << "\tLoad\n";

std::cout << "\tSave filename\n";

std::cout << "\tAdd S/R/T coords\n";

std::cout << "\tRemove ID\n";

std::cout << "\tUndo\n";

std::cout << "\tPrint\n";

}

void create(Editor& editor) {

std::string cmd;

if (editor.DocumentExist()) {

std::cout << "Save old document? Yes/No\n";

std::cin >> cmd;

if (cmd == "Yes") {

std::string filename;

std::cout << "Enter name of file\n";

std::cin >> filename;

try {

editor.SaveDocument(filename);

std::cout << "Document save in file " << filename << "\n";

} catch (std::runtime\_error& err) {

std::cout << err.what() << "\n";

}

}

std::cout << "Enter name of new project\n";

}

std::cin >> cmd;

editor.CreateDocument(cmd);

std::cout << "Document " << cmd << " is created\n";

}

void load(Editor& editor) {

std::string cmd;

if (editor.DocumentExist()) {

std::cout << "Save old document? Yes/No\n";

std::cin >> cmd;

if (cmd == "Yes") {

std::string filename;

std::cout << "Enter name of file\n";

std::cin >> filename;

try {

editor.SaveDocument(filename);

std::cout << "Document save in file " << filename << "\n";

} catch (std::runtime\_error& err) {

std::cout << err.what() << "\n";

}

}

}

std::string filename;

std::cin >> filename;

try {

editor.LoadDocument(filename);

std::cout << "Document loaded from file " << filename << "\n";

} catch (std::runtime\_error& err) {

std::cout << err.what() << "\n";

}

}

void save(Editor& editor) {

std::string filename;

std::cin >> filename;

try {

editor.SaveDocument(filename);

std::cout << "Document save in file " << filename << "\n";

} catch (std::runtime\_error& err) {

std::cout << err.what() << "\n";

}

}

using Point = Pair<double,double>;

void add (Editor& editor) {

char type;

Point a, b, c, d;

std::cin >> type;

std::cin >> a >> b >> c >> d;

switch(type) {

case 'S':

editor.InsertPrimitive(SQUARE, a, b, c, d);

break;

case 'R':

editor.InsertPrimitive(RECTANGLE, a, b, c, d);

break;

case 'T':

editor.InsertPrimitive(TRAPE, a, b, c, d);

break;

default:

std::cout << "Unknown figure\n";

return;

}

std::cout << "Figure is added\n";

}

void remove(Editor& editor) {

int id;

std::cin >> id;

try {

editor.RemovePrimitive(id);

std::cout << "Primitive with " << id << " is removed\n";

} catch (std::logic\_error& err) {

std::cout << err.what() << "\n";

}

}

int main() {

Editor editor;

std::string cmd;

std::cout << "enter 'help' for instructions\n\n";

std::cout << "you need to create or load document for work(cmd Create, Load)\n";

while (std::cin >> cmd) {

if (cmd == "help") {

help();

} else if (cmd == "Create") {

create(editor);

} else if (cmd == "Load") {

load(editor);

} else if (cmd == "Save") {

save(editor);

} else if (cmd == "Add") {

add(editor);

} else if (cmd == "Remove") {

remove(editor);

} else if (cmd == "Undo") {

editor.Undo();

std::cout << "OK\n";

} else if (cmd == "Print") {

editor.PrintDocument();

}

}

return 0;

}

**editor.h**

#ifndef EDITOR\_H

#define EDITOR\_H

#include "doc.h"

#include "com.h"

#include "figure.h"

#include <stack>

class Editor {

public:

Editor() : Doc\_(nullptr), History\_()

{

}

void CreateDocument(const std::string& name) {

Doc\_ = std::make\_shared<Document>(name);

}

using Point = Pair<double,double>;

void InsertPrimitive(FigureType type, const Point a, const Point b, const Point c, const Point d) {

std::shared\_ptr<ICommand>command = std::shared\_ptr<ICommand>(new InsertCommand(type, a, b, c, d));

command->SetDocument(Doc\_);

command->Execute();

History\_.push(command);

}

void RemovePrimitive(int id) {

std::shared\_ptr<ICommand> command = std::shared\_ptr<ICommand>(new RemoveCommand(id));

command->SetDocument(Doc\_);

command->Execute();

History\_.push(command);

}

void SaveDocument(const std::string& filename) {

Doc\_->Save(filename);

}

void LoadDocument(const std::string& filename) {

Doc\_ = std::make\_shared<Document>("NoName");

Doc\_->Load(filename);

}

void Undo() {

if (History\_.empty()) {

throw std::logic\_error("History is empty");

} else {

std::shared\_ptr<ICommand> lastCommand = History\_.top();

lastCommand->UnExecute();

History\_.pop();

}

}

void PrintDocument() {

Doc\_->Print();

}

bool DocumentExist() {

return Doc\_ != nullptr;

}

~Editor() = default;

private:

std::shared\_ptr<Document> Doc\_;

std::stack<std::shared\_ptr<ICommand>> History\_;

};

#endif

**com.h**

#ifndef COMMAND\_H

#define COMMAND\_H

#include "doc.h"

#include "figure.h"

#include <stack>

class ICommand {

protected:

std::shared\_ptr<Document> Doc;

public:

virtual ~ICommand() = default;

virtual void Execute() = 0;

virtual void UnExecute() = 0;

void SetDocument(std::shared\_ptr<Document> doc) {

Doc = doc;

}

};

using Point = Pair<double,double>;

class InsertCommand : public ICommand {

public:

InsertCommand(FigureType type, Point a, Point b, Point c, Point d):

Type\_(type), A\_(std::move(a)), B\_(std::move(b)), C\_(std::move(c)), D\_(std::move(d))

{

}

void Execute() override {

Doc->InsertPrimitive(Type\_, A\_,B\_, C\_, D\_);

}

void UnExecute() override {

Doc->RemoveLastPrimitive();

}

private:

FigureType Type\_;

Point A\_, B\_, C\_, D\_;

};

class RemoveCommand : public ICommand {

public:

explicit RemoveCommand(int id): Id\_(id), Pos\_(0), Figure\_(nullptr)

{

}

void Execute() override {

Figure\_ = Doc->GetFigure(Id\_);

Pos\_ = Doc->GetPos(Id\_);

Doc->RemovePrimitive(Id\_);

}

void UnExecute() override {

Doc->InsertPrimitive(Pos\_, Figure\_);

}

private:

int Id\_;

int Pos\_;

std::shared\_ptr<IFigure> Figure\_;

};

#endif

**doc.h**

#ifndef DOCUMENT\_H

#define DOCUMENT\_H

#include <fstream>

#include <cstdint>

#include <memory>

#include <string>

#include <algorithm>

#include "figure.h"

#include <list>

class Document {

public:

Document(): Id\_(1), Name\_(""), Buf\_(0), squareFactory(), rectangleFactory(), trapeFactory()

{

}

explicit Document(std::string name):

Id\_(1),

Name\_(std::move(name)),

Buf\_(0),

squareFactory(),

rectangleFactory(),

trapeFactory()

{

}

~Document() = default;

void Rename(const std::string& newName) {

Name\_ = newName;

}

void Save(const std::string& filename) const {

SerializeImpl(filename);

}

void Load(const std::string& filename) {

DeserializeImpl(filename);

}

void Print() {

std::for\_each(Buf\_.begin(), Buf\_.end(), [](std::shared\_ptr<IFigure> shape) {

shape->Print(std::cout) << "\n";

};

}

void RemovePrimitive(int id) {

auto it=std::find\_if(Buf\_.begin(),Buf\_.end(),[id](std::shared\_ptr<IFigure> shape) -> bool {

return id == shape->Id();

});

if (it == Buf\_.end())

throw std::logic\_error("Figure with this id doesn't exist");

Buf\_.erase(it);

}

using Point = Pair<double,double>;

void InsertPrimitive(FigureType type, const Point a, const Point b, const Point c, const Point d) {

switch (type) {

case SQUARE:

Buf\_.push\_back(squareFactory.FigureCreate(Id\_++, a, b, c, d));

break;

case RECTANGLE:

Buf\_.push\_back(rectangleFactory.FigureCreate(Id\_++, a, b, c, d));

break;

case TRAPE:

Buf\_.push\_back(trapeFactory.FigureCreate(Id\_++, a, b, c, d));

break;

}

}

private:

int Id\_;

std::string Name\_;

std::list<std::shared\_ptr<IFigure>> Buf\_;

SquareFactory squareFactory;

RectangleFactory rectangleFactory;

TrapeFactory trapeFactory;

friend class InsertCommand;

friend class RemoveCommand;

void SerializeImpl(const std::string& filename) const {

std::ofstream os;

os.open(filename, std::ios\_base::binary | std::ios\_base::out);

if (!os.is\_open()) {

throw std::runtime\_error("File is not opened");

}

uint32\_t nameLen = Name\_.size();

os.write((char\*)&nameLen, sizeof(nameLen));

os.write((char\*)(Name\_.c\_str()), nameLen);

for (const auto& shape : Buf\_) {

shape->Serialize(os);

}

}

void DeserializeImpl(const std::string& filename) {

std::ifstream is;

is.open(filename, std::ios\_base::binary | std::ios\_base::in);

if (!is.is\_open()) {

throw std::runtime\_error("File is not opened");

}

int nameLen;

is.read((char\*)&nameLen, sizeof(nameLen));

char\* name = new char[nameLen + 1];

name[nameLen] = 0;

is.read(name, nameLen);

Name\_ = std::string(name);

delete[] name;

FigureType type;

while(true) {

is.read((char\*)&type, sizeof(type));

if (is.eof())

break;

switch (type) {

case SQUARE:

Buf\_.push\_back(squareFactory.FigureCreate());

break;

case RECTANGLE:

Buf\_.push\_back(rectangleFactory.FigureCreate());

break;

case TRAPE:

Buf\_.push\_back(trapeFactory.FigureCreate());

break;

}

Buf\_.back()->Deserialize(is);

}

Id\_ = Buf\_.size();

}

std::shared\_ptr<IFigure> GetFigure(int id) {

auto it=std::find\_if(Buf\_.begin(),Buf\_.end(),[id](std::shared\_ptr<IFigure> shape) -> bool {

return id == shape->Id();

});

return \*it;

}

int GetPos(int id) {

auto it=std::find\_if(Buf\_.begin(),Buf\_.end(),[id](std::shared\_ptr<IFigure> shape) -> bool {

return id == shape->Id();

});

return std::distance(Buf\_.begin(), it);

}

void InsertPrimitive(int pos, std::shared\_ptr<IFigure> figure) {

auto it = Buf\_.begin();

std::advance(it, pos);

Buf\_.insert(it, figure);

}

void RemoveLastPrimitive() {

if (Buf\_.empty())

throw std::logic\_error("Document is empty");

Buf\_.pop\_back();

}

};

#endif

6. Выводы

Класс проектирования представляет абстракцию одного или нескольких классов в реализации системы. Классы определяют объекты, которые, в свою очередь, реализуют варианты использования. Класс основан на требованиях, предъявляемых реализациями вариантов использования объектам системы, а также на всех ранее разработанных моделях объектов. В проектировании это очень удобный инструмент, который позволяет структурировать код и избежать повторений.

СПИСОК ЛИТЕРАТУРЫ

1. Паттерн проектирования [Электронный ресурс]. URL:<http://cpp-reference.ru/patterns/creational-patterns/factory-method/>

(дата обращения: 20.12.2019).

2. Проектирование и наследование [Электронный ресурс]. URL:

<http://www.williamspublishing.com/PDF/5-8459-0859-0/part.pdf>

(дата обращения: 20.12.2019).