**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: Черемисинов Максим (староста)

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

Реализовать простейший графический редактор с возможностью сохранения документов в файлы, и загрузки из них. Применить паттерн “Фабрика”.

1. **Репозиторий github**

https://github.com/devepodete/oop\_exercise\_07

1. **Описание программы**

Пользователь может добавить или удалить одну из 3 фигур в начало или конец файла, сохранить, импортировать данные из файла, выполнить операцию *undo*. В случае, если файл не сохранен, выдаются предупреждающие сообщения.

1. **Набор testcases**

**test\_01.txt**

push\_begin square 0 0 1 0 1 1 0 1

push\_begin rectangle 0 0 3 0 3 1 0 1

exit

n

export myFigures.bin

exit

//Проверка предупреждения о том, что файл не сохранен

**test\_02.txt**

import myFigures.bin

undo

pop\_back

pop\_end

export myFigures.bin

y

exit

//проверка проверки ввода команд, импорта из файла, сохранения в существующий файл

1. **Результаты выполнения тестов.**

Исключения создаются и отлавливаются, программа выдает правильные ответы.

1. **Листинг программы**

**main.cpp**

/\*

Простейший графический редактор

Черемисинов Максим

М8О-201Б-18

\*/

#include <iostream>

#include <vector>

#include <memory>

#include <string>

#include "figures.hpp"

#include "serialize.hpp"

void print\_actions(){

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

std::cout << "[push\_begin,push\_end] <figure name> <cords>\n";

std::cout << "[pop\_begin, pop\_end]\n";

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

std::cout << "import <filename>\n";

std::cout << "export <filename>\n";

std::cout << "generate N\n";

}

int main(){

std::unique\_ptr<Square\_Factory> square\_factory(new Square\_Factory);

std::unique\_ptr<Rectangle\_Factory> rectangle\_factory(new Rectangle\_Factory);

std::unique\_ptr<Trapezoid\_Factory> trapezoid\_factory(new Trapezoid\_Factory);

Serializable\_Vector sv;

print\_actions();

std::string action = "";

while(action != "exit"){

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

sv.print();

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

std::cout << "> ";

std::cin >> action;

std::string figure\_name;

std::string file\_name;

if(action == "push\_begin" || action == "push\_end"){

std::cin >> figure\_name;

std::vector<Cords> v(4);

std::cin >> v;

if(figure\_name == "square"){

if(action == "push\_begin"){

sv.push\_begin(square\_factory->create\_figure(v));

}else{

sv.push\_end(square\_factory->create\_figure(v));

}

}else if(figure\_name == "rectangle"){

if(action == "push\_begin"){

sv.push\_begin(rectangle\_factory->create\_figure(v));

}else{

sv.push\_end(rectangle\_factory->create\_figure(v));

}

}else if(figure\_name == "trapezoid"){

if(action == "push\_begin"){

sv.push\_begin(trapezoid\_factory->create\_figure(v));

}else{

sv.push\_end(trapezoid\_factory->create\_figure(v));

}

}else{

std::cout << "Error: wrong figure name\n";

}

}else if(action == "pop\_begin"){

sv.pop\_begin();

}else if(action == "pop\_end"){

sv.pop\_end();

}else if(action == "import" || action == "export"){

std::cin >> file\_name;

if(action == "import"){

sv.import\_file(file\_name);

}else{

sv.export\_file(file\_name);

}

}else if(action == "undo"){

sv.undo();

}else if(action == "generate"){

unsigned n;

std::cin >> n;

sv.generate(n);

}else if(action == "exit"){

if(!sv.is\_saved()){

std::cout << "Warning! Your buffer is not saved.\n";

std::cout << "Continue? (y/n): ";

char ans;

std::cin >> ans;

if(ans != 'y'){

action = "";

}

}

}else{

std::cout << "Wrong command\n";

}

}

}

**serialize.hpp**

#ifndef \_SERIALIZE\_HPP\_

#define \_SERIALIZE\_HPP\_

#include <fstream>

#include <iostream>

#include <stack>

#include <vector>

#include <string>

#include <ctime>

#include <cstdlib>

#include <cmath>

#include "figures.hpp"

struct command\_figure{

char figure\_type;

std::string cmd;

Figure \*f;

};

class Serializable\_Vector{

private:

Square\_Factory \*s\_factory;

Rectangle\_Factory \*r\_factory;

Trapezoid\_Factory \*t\_factory;

std::vector<Figure\*> storage;

std::fstream file;

bool file\_opened;

bool buffer\_saved;

std::stack<command\_figure> undo\_commands;

public:

Serializable\_Vector(){

srand(time(NULL));

buffer\_saved = true;

file\_opened = false;

s\_factory = new Square\_Factory;

r\_factory = new Rectangle\_Factory;

t\_factory = new Trapezoid\_Factory;

}

~Serializable\_Vector(){

if(file.is\_open()){

file.close();

}

clear\_storage();

clear\_stack();

delete s\_factory;

delete r\_factory;

delete t\_factory;

}

bool open\_document(std::string str){

bool file\_exist = check\_exist(str);

while(file\_exist){

std::cout << "Warning! File with name \'"<< str << "\' already exists.\n" <<

"Continuing will cause full data loss\n" <<

"Continue? (y/n): ";

char ans;

std::cin >> ans;

if(ans != 'y'){

return false;

}

}

std::ofstream ofs;

ofs.open(str);

if(ofs.is\_open()){

ofs.close();

file.open(str, std::ios::in | std::ios::out | std::ios::binary);

if(file.is\_open()){

file\_opened = true;

return true;

}else{

std::cout << "Can not open fstream file\n";

return false;

}

}else{

std::cout << "Can not open ofstream file\n";

return false;

}

}

void push\_begin(Figure \*f){

char type = f->get\_figure();

storage.insert(storage.begin(), f);

undo\_commands.push({type, "pop\_begin", nullptr});

buffer\_saved = false;

}

void pop\_begin(){

if(!storage.empty()){

Figure \*f = storage.front();

char type = f->get\_figure();

storage.erase(storage.begin());

undo\_commands.push({type, "push\_begin", f});

buffer\_saved = false;

}

}

void push\_end(Figure \*f){

char type = f->get\_figure();

storage.push\_back(f);

undo\_commands.push({type, "pop\_end", nullptr});

buffer\_saved = false;

}

void pop\_end(){

if(!storage.empty()){

Figure \*f = storage.back();

char type = f->get\_figure();

storage.pop\_back();

undo\_commands.push({type, "push\_end", f});

buffer\_saved = false;

}

}

bool export\_file(std::string str){

bool file\_exist = check\_exist(str);

if(file\_exist){

std::cout << "Warning! File with name \'"<< str << "\' already exists.\n" <<

"Continuing will cause full data loss\n" <<

"Continue? (y/n): ";

char ans;

std::cin >> ans;

if(ans != 'y'){

return false;

}

}

std::ofstream temp;

temp.open(str);

temp.close();

std::fstream file\_to\_export;

file\_to\_export.open(str, std::ios::out | std::ios::binary);

if(!file\_to\_export.is\_open()){

std::cout << "Can not open file for export data\n";

return false;

}

size\_t storage\_size = storage.size();

file\_to\_export.write( (char\*)&storage\_size, sizeof(storage.size()) );

for(size\_t i = 0; i < storage\_size; i++){

char figure\_type = storage[i]->get\_figure();

file\_to\_export.write((char\*)&figure\_type, sizeof(figure\_type));

for(size\_t j = 0; j < storage[i]->cords.size(); j++){

file\_to\_export.write( (char\*)(&storage[i]->cords[j]), sizeof(Cords) );

}

}

file\_to\_export.close();

buffer\_saved = true;

return true;

}

bool import\_file(std::string str){

if(!buffer\_saved){

std::cout << "Warning! You have unsaved buffer\n" <<

"Continuing will cause full data loss\n" <<

"Continue? (y/n): ";

char ans;

std::cin >> ans;

if(ans != 'y'){

return false;

}

}

clear\_storage();

clear\_stack();

if(file.is\_open()){

file.close();

}

file.open(str, std::ios::in | std::ios::out | std::ios::binary);

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

std::cout << "Error. Can not import file\n";

file\_opened = false;

return false;

}else{

size\_t figures\_number;

file.read((char\*)&figures\_number, sizeof(figures\_number));

for(size\_t i = 0; i < figures\_number; i++){

char figure\_type;

file.read((char\*)&figure\_type, sizeof(figure\_type));

Cords figure\_cords[4];

for(size\_t j = 0; j < 4; j++){

file.read((char\*)&figure\_cords[j], sizeof(figure\_cords[j]));

}

switch(figure\_type){

case 's':{

storage.push\_back(s\_factory->create\_figure(figure\_cords[0],figure\_cords[1],figure\_cords[2],figure\_cords[3]));

break;

}

case 'r':{

storage.push\_back(r\_factory->create\_figure(figure\_cords[0],figure\_cords[1],figure\_cords[2],figure\_cords[3]));

break;

}

case 't':{

storage.push\_back(t\_factory->create\_figure(figure\_cords[0],figure\_cords[1],figure\_cords[2],figure\_cords[3]));

break;

}

default:

std::cout << "Error while reading from import\_file file: wrong figure class\n";

break;

}

}

buffer\_saved = true;

file.close();

file\_opened = false;

return true;

}

}

void undo(){

if(undo\_commands.empty()){

std::cout << "There is no operations to undo\n";

}else{

char figure\_type = undo\_commands.top().figure\_type;

std::string action = undo\_commands.top().cmd;

if(action == "push\_begin"){

if(figure\_type == 's'){

storage.insert(storage.begin(), s\_factory->create\_figure(undo\_commands.top().f->cords));

}else if(figure\_type == 'r'){

storage.insert(storage.begin(), r\_factory->create\_figure(undo\_commands.top().f->cords));

}else if(figure\_type == 't'){

storage.insert(storage.begin(), t\_factory->create\_figure(undo\_commands.top().f->cords));

}else{

std::cout << "Error while undo: incorrect figure type\n";

}

delete undo\_commands.top().f;

}else if(action == "pop\_begin"){

if(!storage.empty()){

delete storage.front();

storage.erase(storage.begin());

}

}else if(action == "push\_end"){

if(figure\_type == 's'){

storage.push\_back(s\_factory->create\_figure(undo\_commands.top().f->cords));

}else if(figure\_type == 'r'){

storage.push\_back(r\_factory->create\_figure(undo\_commands.top().f->cords));

}else if(figure\_type == 't'){

storage.push\_back(t\_factory->create\_figure(undo\_commands.top().f->cords));

}else{

std::cout << "Error while undo: incorrect figure type\n";

}

delete undo\_commands.top().f;

}else if(action == "pop\_end"){

if(!storage.empty()){

delete storage.back();

storage.pop\_back();

}

}else if(action == "clear"){

clear\_storage();

}else{

std::cout << "Error while undo: wrong action\n";

}

undo\_commands.pop();

buffer\_saved = false;

}

}

bool check\_exist(std::string str){

std::fstream fs;

fs.open(str);

if(fs.is\_open()){

fs.close();

return true;

}else{

return false;

}

}

void print(){

for(size\_t i = 0; i < storage.size(); i++){

storage[i]->print();

}

}

size\_t size(){

return storage.size();

}

void clear\_storage(){

while(!storage.empty()){

delete storage.back();

storage.pop\_back();

}

}

void clear\_stack(){

while(!undo\_commands.empty()){

delete undo\_commands.top().f;

undo\_commands.pop();

}

}

void generate(unsigned number){

if(file\_opened && !buffer\_saved){

std::cout << "Warning! Generating random figures will cause full data loss\n";

std::cout << "Continue? (y/n): ";

char ans;

std::cin >> ans;

if(ans != 'y'){

return;

}

}

if(file.is\_open()){

file.close();

}

file\_opened = false;

clear\_stack();

clear\_storage();

for(size\_t i = 0; i < number; i++){

std::vector<Cords> svc(4);

unsigned figure\_type = rand()%3;

generate\_figure(svc, svc.size());

if(figure\_type == 0){

storage.push\_back(s\_factory->create\_figure(svc));

}else if(figure\_type == 1){

storage.push\_back(r\_factory->create\_figure(svc));

}else{

storage.push\_back(t\_factory->create\_figure(svc));

}

}

buffer\_saved = false;

undo\_commands.push({'c', "clear", nullptr});

}

void generate\_figure(std::vector<Cords> &v, unsigned n){

if(n == 0){

return;

}

double x0 = static\_cast<double>(rand()%20);

double y0 = static\_cast<double>(rand()%20);

double r = static\_cast<double>(rand()%20+1);

for(unsigned i = 0; i < n; i++){

v[i].x = x0 + r\*std::cos(2\*M\_PI\*i/n);

v[i].y = y0 + r\*std::sin(2\*M\_PI\*i/n);

}

}

bool is\_saved(){

return buffer\_saved;

}

};

#endif

**figures.hpp**

#ifndef \_FIGURES\_HPP\_

#define \_FIGURES\_HPP\_

#include <iostream>

#include <vector>

struct Cords{

double x, y;

};

std::istream &operator>>(std::istream &is, Cords &c){

is >> c.x >> c.y;

return is;

}

std::istream &operator>>(std::istream &is, std::vector<Cords> &v){

for(size\_t i = 0; i < v.size(); i++){

std::cin >> v[i];

}

return is;

}

std::ostream &operator<<(std::ostream &os, Cords c){

os << "<" << c.x << ", " << c.y << ">";

return os;

}

class Figure{

public:

std::vector<Cords> cords;

Figure(){}

Figure(const Figure &f){

for(size\_t i = 0; i < f.cords.size(); i++){

cords.push\_back(f.cords[i]);

}

}

virtual ~Figure(){}

void read\_cords(){

for(size\_t i = 0; i < cords.size(); i++){

std::cin >> cords[i];

}

}

virtual void print\_name() = 0;

virtual char get\_figure() = 0;

void print(){

print\_name();

for(size\_t i = 0; i < cords.size(); i++){

std::cout << cords[i] << " ";

}

std::cout << '\n';

}

};

class Square: public Figure{

public:

Square(){

for(size\_t i = 0; i < 4; i++){

cords.push\_back({0, 0});

}

}

Square(const Cords &c1, const Cords &c2, const Cords &c3, const Cords &c4){

cords.push\_back(c1);

cords.push\_back(c2);

cords.push\_back(c3);

cords.push\_back(c4);

}

Square(const Square &s){

cords = s.cords;

}

Square(const std::vector<Cords> &v){

cords = v;

}

~Square(){}

void print\_name(){

std::cout << "Square: ";

}

char get\_figure(){

return 's';

}

};

class Rectangle: public Figure{

public:

Rectangle(){

for(size\_t i = 0; i < 4; i++){

cords.push\_back({0,0});

}

}

Rectangle(const Cords &c1, const Cords &c2, const Cords &c3, const Cords &c4){

cords.push\_back(c1);

cords.push\_back(c2);

cords.push\_back(c3);

cords.push\_back(c4);

}

Rectangle(const Rectangle &r){

cords = r.cords;

}

Rectangle(const std::vector<Cords> &v){

cords = v;

}

~Rectangle(){}

void print\_name(){

std::cout << "Rectangle: ";

}

char get\_figure(){

return 'r';

}

};

class Trapezoid: public Figure{

public:

Trapezoid(){

for(size\_t i = 0; i < 4; i++){

cords.push\_back({0,0});

}

}

Trapezoid(const Cords &c1, const Cords &c2, const Cords &c3, const Cords &c4){

cords.push\_back(c1);

cords.push\_back(c2);

cords.push\_back(c3);

cords.push\_back(c4);

}

Trapezoid(const Trapezoid &t){

cords = t.cords;

}

Trapezoid(const std::vector<Cords> &v){

cords = v;

}

~Trapezoid(){}

void print\_name(){

std::cout << "Trapezoid: ";

}

char get\_figure(){

return 't';

}

};

class Factory{

public:

virtual Figure \*create\_figure() = 0;

virtual ~Factory(){}

};

class Square\_Factory: public Factory{

public:

Figure \*create\_figure(){

return reinterpret\_cast<Figure\*>(new Square());

}

Figure \*create\_figure(const Cords &c1, const Cords &c2, const Cords &c3, const Cords &c4){

return reinterpret\_cast<Figure\*>(new Square(c1, c2, c3, c4));

}

Figure \*create\_figure(const std::vector<Cords> &v){

return reinterpret\_cast<Figure\*>(new Square(v));

}

};

class Rectangle\_Factory: public Factory{

public:

Figure \*create\_figure(){

return reinterpret\_cast<Figure\*>(new Rectangle());

}

Figure \*create\_figure(const Cords &c1, const Cords &c2, const Cords &c3, const Cords &c4){

return reinterpret\_cast<Figure\*>(new Rectangle(c1, c2, c3, c4));

}

Figure \*create\_figure(const std::vector<Cords> &v){

return reinterpret\_cast<Figure\*>(new Rectangle(v));

}

};

class Trapezoid\_Factory: public Factory{

public:

Figure \*create\_figure(){

return reinterpret\_cast<Figure\*>(new Trapezoid());

}

Figure \*create\_figure(const Cords &c1, const Cords &c2, const Cords &c3, const Cords &c4){

return reinterpret\_cast<Figure\*>(new Trapezoid(c1, c2, c3, c4));

}

Figure \*create\_figure(const std::vector<Cords> &v){

return reinterpret\_cast<Figure\*>(new Trapezoid(v));

}

};

#endif

1. **Вывод**

Применение определенных паттернов программирования позволяет во многих случаях легче поддерживать код в долгосрочных проектах, изменять меньшую его часть при внесении правок.

Программа позволяет выполнять простейшие действия над файлом. Возможно также расширение программы операциями вставки и удаления элемента с любой позиции в файле, добавление команды *redo*.

**Список литературы**

1. Справочник по языку С++ [Электронный ресурс]. URL:

<https://en.cppreference.com/w/> (дата обращения: 10.12.2019).