**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: Лисин Роман Сергеевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Создать простейший “графический” редактор. Требования к функционалу редактора:

● создание нового документа.

● импорт документа в файл.

● экспорт документа из файла.

● создание графического примитива (фигуры: ромб, прямоугольник, трапеция).

● удаление графического примитива.

● отображение документа на экране.

● реализовать операцию undo, отменяющую последнее сделанное действие.

Требования к реализации:

● Создание графических примитивов необходимо вынести в отдельный класс factory.

● Сделать упор на использование полиморфизма при работе с фигурами.

● Взаимодействие с пользователем реализовать в функции main.

1. **Описание программы**

***Класс figure***

Класс figure - это абстрактный базовый класс для остальных фигур. Класс содержит в себе чисто виртуальные функции square() для вычисления площади, print() для печати фигуры, write\_to\_file() для записи в файл. Единственный атрибут - координаты центра фигуры.

***Классы rhombus, rectangle, trapezoid***

Классы rhombus, rectangle и trapezoid - это классы-наследники от figure, в которых описаны ромб, прямоугольник и трапеция соответственно. В этих классах переопределены все виртуальные функции из базового класса, а также переопределен оператор вывода.

***Класс document***

Класс document описывает функционал для работы с “графическим” документом. Класс включает в себя методы для добавления фигуры в документ, удаления фигур, печати всех фигур, записи всех данных в файл, чтения всех данных из файла и отмены последнего действия. Для последнего метода используется объект класса originator, который в свою очередь является обёрткой для шаблона memento.

Атрибуты класса: name - название документа, buffer - вектор умных указателей на фигуры.

В случае возникновения логических ошибок в методах класса генерируются соответствующие исключения.

***Класс factory***

В данном классе реализован шаблон factory. Этот шаблон предназначен для упрощения создания новых объектов. Во время выполнения программы он сам определяет, какой объект необходимо создать, при помощи id фигуры. Фигуры и их id определены в enum class figure\_type. Класс возвращает умный указатель на созданную фигуру.

***Работа программы***

В функции main описано взаимодействие с пользователем. Пользователю предоставлен набор команд, позволяющий работать с документами и с фигурами в документе.

Пользователь может создать новый документ или открыть уже существующий. При запуске программы документ не открывается. Пока документ не будет открыт, функции для работы с фигурами будут недоступными. Для создания документа необходимо ввести его имя. Переименовать документ нельзя. Пользователю предоставлена возможность сохранить документ. Он будет сохранен в директории программы в файле с названием документа. В случае возникновения системных ошибок генерируются исключения.

В открытый документ пользователь может добавлять фигуры. Для добавления фигуры нужно ввести её id, координаты центра и дополнительные атрибуты. Есть возможность удаления фигуры по индексу. Если индекс, введенный пользователем, некорректный, то программа выдаст соответствующее сообщение.

Пользователь может распечатать все содержимое документа: фигуры, их площади. Также предусмотрена возможность отмены последнего совершенного действия.

Для выхода из программы пользователю нужно ввести соответствующую команду. В случае ввода неверной команды будет показано соответствующее предупреждение.

1. **Набор тестов**

1 hello // создать документ hello

4 1 0 0 5 10 // добавить ромб с центром (0, 0) и диагоналями 5 и 10

6 // печать всех фигур. будет напечатан только ромб

2 // сохранить документ

4 3 10 10 10 // добавить трапецию с основаниями и высотой 10

6 // печать всех фигур. будет напечатано 2 фигур

3 doc123 // загрузить документ doc123. будет ошибка, т.к. нет такого файла

3 hello // загрузить документ hello

6 // печать всех фигур. будет напечатан только ромб

0 // выход

1. **Результаты выполнения тестов**

1. Create new document

2. Save document

3. Open document

4. Add figure

5. Remove figure

6. Print figures

7. Undo

0. Exit

1

Enter document's name:

hello

Created new document

4 1 0 0 5 10

Enter figure id (1 - rhombus, 2 - rectangle, 3 - trapezoid)

Enter coords of the center and lengths of diagonals

Figure was successfully added

6

Rhombus {(-2.5, 0); (0, 5); (2.5, 0); (0, -5)}

Square: 25

2

Successfully saved

4 3 10 10 10

Enter figure id (1 - rhombus, 2 - rectangle, 3 - trapezoid)

Enter the lengths of bases and height

Figure was successfully added

6

Rhombus {(-2.5, 0); (0, 5); (2.5, 0); (0, -5)}

Square: 25

Trapezoid {( Bases: 10, 10; Height: 10)}

Square: 100

3 doc123

Enter file's name:

No such file

3

6

1. **Листинг программы**

**main.cpp**

#include <iostream>

#include <memory>

#include <vector>

#include "document.h"

/\* Лабораторная работа №7

Вариант 19: Прямоугольник, трапеция, ромб

Выполнил Лисин Роман, группа М8О-206Б-20

\*/

void print\_menu() {

std::cout << "1. Create new document" << std::endl;

std::cout << "2. Save document" << std::endl;

std::cout << "3. Open document" << std::endl;

std::cout << "4. Add figure" << std::endl;

std::cout << "5. Remove figure" << std::endl;

std::cout << "6. Print figures" << std::endl;

std::cout << "7. Undo" << std::endl;

std::cout << "0. Exit" << std::endl;

std::cout << std::endl;

}

int main() {

print\_menu();

std::shared\_ptr<document> doc;

int cmd;

while (true) {

std::cin >> cmd;

if (cmd == 1) {

std::string name;

std::cout << "Enter document's name:" << std::endl;

std::cin >> name;

doc = std::make\_shared<document>(name);

std::cout << "Created new document" << std::endl;

}

else if (cmd == 2) {

if (!doc) {

std::cout << "Open document first" << std::endl;

}

else {

try {

doc->save();

std::cout << "Successfully saved" << std::endl;

}

catch (std::exception& ex) {

std::cout << ex.what() << std::endl;

}

}

}

else if (cmd == 3) {

std::string file\_name;

std::cout << "Enter file's name: " << std::endl;

std::cin >> file\_name;

std::ifstream in;

in.open(file\_name, std::ios::in | std::ios::binary);

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

std::cout << "No such file" << std::endl;

}

else {

doc = std::make\_shared<document>(file\_name);

try {

doc->open(in);

std::cout << "File was successfully loaded" << std::endl;

}

catch (std::exception& ex) {

std::cout << ex.what() << std::endl;

}

in.close();

}

}

else if (cmd == 4) {

if (!doc) {

std::cout << "Open document first" << std::endl;

}

else {

std::cout << "Enter figure id (1 - rhombus, 2 - rectangle, 3 - trapezoid)" << std::endl;

int type;

std::cin >> type;

if (type == 1) {

std::cout << "Enter coords of the center and lengths of diagonals" << std::endl;

}

else if (type == 2) {

std::cout << "Enter coords of the left highest point, height and width" << std::endl;

}

else if (type == 3) {

std::cout << "Enter the lengths of bases and height" << std::endl;

}

std::shared\_ptr<figure> fig = factory::create((figure\_type)type);

doc->add(fig);

std::cout << "Figure was successfully added" << std::endl;

}

}

else if (cmd == 5) {

if (!doc) {

std::cout << "Open document first" << std::endl;

}

else {

int id;

std::cout << "Enter id of the figure" << std::endl;

std::cin >> id;

try {

doc->remove(id);

std::cout << "Figure was successfully removed" << std::endl;

}

catch (std::exception& ex) {

std::cout << ex.what() << std::endl;

}

}

}

else if (cmd == 6) {

if (!doc) {

std::cout << "Open document first" << std::endl;

}

else {

doc->print();

}

}

else if (cmd == 7) {

if (!doc) {

std::cout << "Open document first" << std::endl;

}

else {

try {

doc->undo();

std::cout << "Done" << std::endl;

}

catch (std::exception& ex) {

std::cout << ex.what() << std::endl;

}

}

}

else if (cmd == 0) {

break;

}

else {

std::cout << "Wrong cmd" << std::endl;

}

}

}

**rectangle.h**

#ifndef OOP\_LAB7\_RECTANGLE\_H

#define OOP\_LAB7\_RECTANGLE\_H

#include "figure.h"

#include <iostream>

class rectangle : public figure {

public:

rectangle() = default;

rectangle(std::pair<double, double>& center, double h, double w) : figure(center), height(h), width(w) {}

double square() override {

return height \* width;

}

void print() override {

std::cout << \*this;

}

void write\_to\_file(std::ofstream& out) override {

int id = 1;

out << "Rectangle {(" << center.first << ", " << center.second << "); "

<< "(" << center.first + width << ", " << center.second << "); "

<< "(" << center.first + width << ", " << center.second - height << "); "

<< "(" << center.first << ", " << center.second - height << ")}\n";

}

friend std::ostream& operator<<(std::ostream& out, rectangle& r);

private:

double height = 0;

double width = 0;

};

std::ostream& operator<<(std::ostream& out, rectangle& r) {

out << "Rectangle {(" << r.center.first << ", " << r.center.second << "); "

<< "(" << r.center.first + r.width << ", " << r.center.second << "); "

<< "(" << r.center.first + r.width << ", " << r.center.second - r.height << "); "

<< "(" << r.center.first << ", " << r.center.second - r.height << ")}\n";

return out;

}

#endif //OOP\_LAB7\_RECTANGLE\_H

**rhombus.h**

#ifndef OOP\_LAB7\_RHOMBUS\_H

#define OOP\_LAB7\_RHOMBUS\_H

#include "figure.h"

#include <iostream>

class rhombus : public figure {

public:

rhombus() = default;

rhombus(std::pair<double, double>& center, double d1, double d2) : figure(center), diag1(d1), diag2(d2) {}

double square() override {

return diag1 \* diag2 \* 0.5;

}

void print() override {

std::cout << \*this;

}

void write\_to\_file(std::ofstream& out) override {

int id = 1;

out << "Rhombus {(" << center.first - diag1 \* 0.5 << ", " << center.second << "); (";

out << center.first << ", " << center.second + diag2 \* 0.5 << "); (";

out << center.first + diag1 \* 0.5 << ", " << center.second << "); (";

out << center.first << ", " << center.second - diag2 \* 0.5 << ")}\n";

}

friend std::ostream& operator<<(std::ostream& out, rhombus& r);

private:

double diag1 = 0;

double diag2 = 0;

};

std::ostream& operator<<(std::ostream& out, rhombus& r) {

out << "Rhombus {(" << r.center.first - r.diag1 \* 0.5 << ", " << r.center.second << "); (";

out << r.center.first << ", " << r.center.second + r.diag2 \* 0.5 << "); (";

out << r.center.first + r.diag1 \* 0.5 << ", " << r.center.second << "); (";

out << r.center.first << ", " << r.center.second - r.diag2 \* 0.5 << ")}\n";

return out;

}

#endif //OOP\_LAB7\_RHOMBUS\_H

**trapezoid.h**

#ifndef OOP\_LAB7\_TRAPEZOID\_H

#define OOP\_LAB7\_TRAPEZOID\_H

#include "figure.h"

#include <iostream>

class trapezoid : public figure {

public:

trapezoid() = default;

trapezoid(double a, double b, double h) : a(a), b(b), h(h) {}

double square() override {

return (a + b) / 2 \* h;

}

void print() override {

std::cout << \*this;

}

void write\_to\_file(std::ofstream& out) override {

int id = 1;

out << "Trapezoid {(" << " Bases: " << a << ", " << b << "; Height: " << h << ")}\n";

}

friend std::ostream& operator<<(std::ostream& out, trapezoid& r);

private:

double a = 0;

double b = 0;

double h = 0;

};

std::ostream& operator<<(std::ostream& out, trapezoid& t) {

out << "Trapezoid {(" << " Bases: "<< t.a << ", " << t.b << "; Height: " << t.h << ")}\n";

return out;

}

#endif //OOP\_LAB7\_TRAPEZOID\_H

**factory.h**

#ifndef OOP\_LAB7\_FACTORY\_H

#define OOP\_LAB7\_FACTORY\_H

#include "rhombus.h"

#include "rectangle.h"

#include "trapezoid.h"

enum class figure\_type {

rhombus = 1,

rectangle = 2,

trapezoid = 3

};

struct factory {

static std::shared\_ptr<figure> create(figure\_type t) {

switch (t) {

case figure\_type::rhombus: {

std::pair<double, double> center;

double d1, d2;

std::cin >> center.first >> center.second >> d1 >> d2;

return std::make\_shared<rhombus>(center, d1, d2);

}

case figure\_type::rectangle: {

std::pair<double, double> center;

double h, w;

std::cin >> center.first >> center.second >> h >> w;

return std::make\_shared<rectangle>(center, h, w);

}

case figure\_type::trapezoid: {

std::pair<double, double> center;

double a, b, h;

std::cin >> a >> b >> h;

return std::make\_shared<trapezoid>(a, b, h);

}

default:

throw std::logic\_error("Wrong figure id");

}

}

static std::shared\_ptr<figure> read\_from\_file(figure\_type t, std::ifstream& in) {

switch (t) {

case figure\_type::rhombus: {

std::pair<double, double> center;

double d1, d2;

in.read((char\*)&center.first, sizeof(double));

in.read((char\*)&center.second, sizeof(double));

in.read((char\*)&d1, sizeof(double));

in.read((char\*)&d2, sizeof(double));

return std::make\_shared<rhombus>(center, d1, d2);

}

case figure\_type::rectangle: {

std::pair<double, double> center;

double height, width;

in.read((char\*)&center.first, sizeof(double));

in.read((char\*)&center.second, sizeof(double));

in.read((char\*)&height, sizeof(double));

in.read((char\*)&width, sizeof(double));

return std::make\_shared<rectangle>(center, height, width);

}

case figure\_type::trapezoid: {

double a, b, h;

in.read((char\*)&a, sizeof(double));

in.read((char\*)&b, sizeof(double));

in.read((char\*)&h, sizeof(double));

return std::make\_shared<trapezoid>(a, b, h);

}

default:

throw std::logic\_error("Wrong figure id");

}

}

};

#endif //OOP\_LAB7\_FACTORY\_H

**figure.h**

#ifndef OOP\_LAB7\_FIGURE\_H

#define OOP\_LAB7\_FIGURE\_H

#include <cmath>

#include <fstream>

class figure {

public:

figure() = default;

figure(std::pair<double, double>& center\_) : center(center\_) {}

virtual double square() = 0;

virtual void print() = 0;

virtual void write\_to\_file(std::ofstream& out) = 0;

std::pair<double, double> get\_center() {

return center;

}

protected:

std::pair<double, double> center;

};

#endif //OOP\_LAB7\_FIGURE\_H

**document.h**

#ifndef OOP\_LAB7\_DOCUMENT\_H

#define OOP\_LAB7\_DOCUMENT\_H

#include <stack>

#include <fstream>

#include <vector>

#include "factory.h"

class document {

private:

struct memento {

std::vector<std::shared\_ptr<figure>> state;

memento() = default;

memento(std::vector<std::shared\_ptr<figure>>& other) : state(other) {}

};

struct originator {

std::stack<memento> mementos;

void create\_memento(std::vector<std::shared\_ptr<figure>>& state) {

mementos.emplace(state);

}

std::vector<std::shared\_ptr<figure>> restore() {

if (!mementos.empty()) {

std::vector<std::shared\_ptr<figure>> res = mementos.top().state;

mementos.pop();

return res;

}

throw std::logic\_error("Can't undo");

}

};

std::string name;

std::vector<std::shared\_ptr<figure>> buffer;

originator origin;

public:

document(std::string& name\_) : name(name\_) {}

void add(const std::shared\_ptr<figure>& figure) {

origin.create\_memento(buffer);

buffer.push\_back(figure);

}

void remove(int id) {

if (id >= 0 && id < buffer.size()) {

origin.create\_memento(buffer);

buffer.erase(buffer.begin() + id);

}

else {

throw std::logic\_error("Invalid position");

}

}

void undo() {

buffer = origin.restore();

}

void print() {

for (auto& f : buffer) {

f->print();

std::cout << std::endl;

std::cout << "Square: " << f->square() << std::endl;

//auto center = f->get\_center();

//std::cout << "Center: (" << center.first << "; " << center.second << ")" << std::endl << std::endl;

}

}

void save() {

std::ofstream out;

out.open(name, std::ios::out | std::ios::binary | std::ios::trunc);

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

throw std::logic\_error("Can't open file");

}

else {

int size = buffer.size();

out.write((char\*)&size, sizeof(int));

for (auto& f : buffer) {

f->write\_to\_file(out);

}

out.close();

}

}

void open(std::ifstream& in) {

int size;

in.read((char\*)&size, sizeof(int));

for (int i = 0; i < size; ++i) {

int type;

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

buffer.push\_back(factory::read\_from\_file((figure\_type)type, in));

}

}

};

#endif //OOP\_LAB7\_DOCUMENT\_H

**ЛИТЕРАТУРА**

1. Паттерн Factory [Электронный ресурс].

URL: http://cpp-reference.ru/patterns/creational-patterns/factory-method/ (дата обращения: 16.12.2021)