**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: Кочисов Алексей

Группа: М8О-201Б-20

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

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 в std::cout)
* реализовать операцию undo, которая отменяет 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 9: треугольник, квадрат, прямоугольник.

1. Описание программы

Программа описывает объектно ориентированную структуру классов в основе которой лежит сущность Document, представляющая собой документ с возможностью вставки или удаления фигур, распечатывания его содержимого, а также сохранения и загрузки. В качестве дополнительных операций для каждой рабочей сессии документ предоставляет возможность откатить последние изменение или восстановить их (после отката).

В main.cpp находится интерактивная часть программы, в файлах figure.hpp, triangle.cpp, square.cpp, rectangle.cpp находится описание и реализация классов для работы с геометрическими формами. В файле factory.hpp — фабрика, создающая конкретный объект по его строковому описанию.

1. Набор тестов

Описание команд:

0. Create new document

1. Insert figure: <figure-type> <coord-center> <coord0>

2. Delete figure: <index>

3. Save document: <document-name>

4. Load document: <document-name>

5. Undo last operation

6. Redo last undone operation

7. Print Document

test01.txt — проверяет возможность добавить элементы, отменить и повторить одно предыдущее действие, вывести элементы на экран, удалить элемент по индексу, сохранить документ в файл, попытка выполнить несуществующую команду.

1 triangle 1 1 2 2

1 square 2 2 4 -4

7

5

7

6

7

1 rectangle 0 0 5 10

7

2 1

7

3 saved.file

10

test02.txt — проверяет загрузку из файла, отмену и восстановление нескольких действий.

4 saved.file

7

1 triangle -1 -2 0 11

7

2 0

2 1

7

5

7

5

7

5

7

6

6

7

3 saved2.file

2 0

2 0

7

4 saved2.file

7

1. Результаты выполнения тестов

answer01:

Inserted!

Inserted!

Print:

triangle: [x: 2 y: 2] [x: -0.36 y: 1.36] [x: 1.36 y: -0.36]

square: [x: 4 y: -4] [x: 0 y: 8]

Undone!

Print:

triangle: [x: 2 y: 2] [x: -0.36 y: 1.36] [x: 1.36 y: -0.36]

Redone!

Print:

triangle: [x: 2 y: 2] [x: -0.36 y: 1.36] [x: 1.36 y: -0.36]

square: [x: 4 y: -4] [x: 0 y: 8]

Inserted!

Print:

triangle: [x: 2 y: 2] [x: -0.36 y: 1.36603] [x: 1.36 y: -0.36]

square: [x: 4 y: -4] [x: 0 y: 8]

rectangle: [x: 5 y: 10] [x: -5 y: -10]

Removed!

Print:

triangle: [x: 2 y: 2] [x: -0.36 y: 1.36] [x: 1.36 y: -0.36]

rectangle: [x: 5 y: 10] [x: -5 y: -10]

Saved!

No such option :)

Goodbye!

answer02:

Loaded!

Print:

triangle: [x: 2 y: 2] [x: -0.366025 y: 1.36603] [x: 1.36603 y: -0.366025]

rectangle: [x: 5 y: 10] [x: -5 y: -10]

Inserted!

Print:

triangle: [x: 2 y: 2] [x: -0.366025 y: 1.36603] [x: 1.36603 y: -0.366025]

rectangle: [x: 5 y: 10] [x: -5 y: -10]

triangle: [x: 0 y: 11] [x: -12.7583 y: -7.63397] [x: 9.75833 y: -9.36603]

Removed!

Removed!

Print:

rectangle: [x: 5 y: 10] [x: -5 y: -10]

Undone!

Print:

rectangle: [x: 5 y: 10] [x: -5 y: -10]

triangle: [x: 0 y: 11] [x: -12.75 y: -7.63] [x: 9.75 y: -9.36]

Undone!

Print:

triangle: [x: 2 y: 2] [x: -0.36 y: 1.36] [x: 1.36 y: -0.36]

rectangle: [x: 5 y: 10] [x: -5 y: -10]

triangle: [x: 0 y: 11] [x: -12.75 y: -7.63] [x: 9.75 y: -9.36]

Undone!

Print:

triangle: [x: 2 y: 2] [x: -0.36 y: 1.36] [x: 1.36 y: -0.36]

rectangle: [x: 5 y: 10] [x: -5 y: -10]

Redone!

Redone!

Print:

rectangle: [x: 5 y: 10] [x: -5 y: -10]

triangle: [x: 0 y: 11] [x: -12.75 y: -7.63] [x: 9.75 y: -9.36]

Saved!

Removed!

Removed!

Print:

Loaded!

Print:

rectangle: [x: 5 y: 10] [x: -5 y: -10]

triangle: [x: 0 y: 11] [x: -12.75 y: -7.63] [x: 9.75 y: -9.36]

Goodbye!

1. Листинг программы

main.cpp

#include <iostream>

#include <iomanip>

#include "document.hpp"

int main() {

uint32\_t option;

Document doc;

while (std::cin >> option) {

if (option == 0)

doc.Create();

else if (option == 1) {

std::string figureType;

std::cin >> figureType;

std::unique\_ptr<Figure> obj = Factory<Figure>::GetInstance().Create(figureType);

std::cin >> \*(obj.get());

doc.Add(std::move(obj));

std::cout << "Inserted!" << std::endl;

}

else if (option == 2) {

uint32\_t index;

std::cin >> index;

doc.Remove(index);

std::cout << "Removed!" << std::endl;

}

else if (option == 3) {

std::string filename;

std::cin >> filename;

doc.Save(filename);

std::cout << "Saved!" << std::endl;

}

else if (option == 4) {

std::string filename;

std::cin >> filename;

doc.Load(filename);

std::cout << "Loaded!" << std::endl;

}

else if (option == 5) {

doc.Undo();

std::cout << "Undone!" << std::endl;

}

else if (option == 6) {

doc.Redo();

std::cout << "Redone!" << std::endl;

}

else if (option == 7)

doc.Print();

else

std::cout << "No such option :)" << std::endl;

}

std::cout << "Goodbye!" << std::endl;

return 0;

}

figure.hpp

#pragma once

#include <iostream>

#include <vector>

#define \_USE\_MATH\_DEFINES

#include <math.h>

class Point2D {

public:

double x;

double y;

Point2D(double x = 0, double y = 0);

friend std::ostream& operator<<(std::ostream& os, const Point2D& d);

friend std::istream& operator>>(std::istream& is, Point2D& d);

};

class Figure {

public:

static uint64\_t counter;

Figure();

virtual ~Figure();

[[nodiscard]] virtual Point2D GetCentroid() const = 0;

[[nodiscard]] virtual double GetArea() const = 0;

[[nodiscard]] const std::string& GetName() const;

virtual void Write(std::ostream& os) const;

friend std::ostream& operator<<(std::ostream& os, Figure& figure);

virtual void Read(std::istream& is) = 0;

friend std::istream& operator>>(std::istream& is, Figure& mc);

[[nodiscard]] const std::vector<Point2D>& GetVertices() const;

protected:

std::vector<Point2D> vertices;

std::string name;

private:

const uint64\_t id;

};

template <typename T>

class FigureCreator {

public:

static Figure\* Create() {

return new T();

}

};

class Triangle : public Figure {

public:

Triangle(const Point2D& a = Point2D(), const Point2D& b = Point2D(), const Point2D& c = Point2D());

~Triangle() override;

void Read(std::istream& is) override;

[[nodiscard]] Point2D GetCentroid() const override;

[[nodiscard]] double GetArea() const override;

};

class Square : public Figure {

public:

Square(const Point2D& a = Point2D(), const Point2D& b = Point2D());

~Square() override;

void Read(std::istream& is) override;

[[nodiscard]] Point2D GetCentroid() const override;

[[nodiscard]] double GetArea() const override;

};

class Rectangle : public Figure {

public:

Rectangle(const Point2D& a = Point2D(), const Point2D& b = Point2D());

~Rectangle() override;

void Read(std::istream& is) override;

[[nodiscard]] Point2D GetCentroid() const override;

[[nodiscard]] double GetArea() const override;

};

figure.cpp

#include "figure.hpp"

Point2D::Point2D(double x, double y) : x(x), y(y) {};

std::ostream& operator<<(std::ostream& os, const Point2D& d) {

os << "[x: " << d.x << " y: " << d.y << ']';

return os;

}

std::istream& operator>>(std::istream& is, Point2D& d) {

is >> d.x >> d.y;

return is;

}

uint64\_t Figure::counter = 0;

Figure::Figure() : id(counter++) {

if (DEBUG) std::cout << "constructor called for Figure with ID of " << this->id << std::endl;

};

Figure::~Figure() {

if (DEBUG) std::cout << "destructor called for Figure with ID of " << this->id << std::endl;

};

[[nodiscard]] const std::string& Figure::GetName() const {

return this->name;

}

void Figure::Write(std::ostream& os) const {

os << this->name << ':';

for (auto vertex : this->vertices) {

os << ' ' << vertex;

}

}

std::ostream& operator<<(std::ostream& os, Figure& figure) {

figure.Write(os);

return os;

}

std::istream& operator>>(std::istream& is, Figure& mc) {

mc.Read(is);

return is;

}

[[nodiscard]] const std::vector<Point2D>& Figure::GetVertices() const {

return this->vertices;

}

triangle.cpp

#include "figure.hpp"

Triangle::Triangle(const Point2D& a, const Point2D& b, const Point2D& c) {

this->vertices.assign({ a,b,c });

this->name = "triangle";

if (DEBUG) std::cout << "constructor called for " << this->name << " with vertices: "

<< vertices.at(0) << ' '

<< vertices.at(1) << ' '

<< vertices.at(2) << std::endl;

}

Triangle::~Triangle() {

if (DEBUG) std::cout << "destructor called for " << this->name << " with vertices: "

<< vertices.at(0) << ' '

<< vertices.at(1) << ' '

<< vertices.at(2) << std::endl;

}

void Triangle::Read(std::istream& is) {

this->vertices.resize(3);

Point2D center;

is >> center >> vertices[0];

double alpha = 120.0 / 180.0 \* M\_PI;

for (int i = 1; i < 3; ++i) {

vertices[i].x = -sin(alpha) \* (vertices[i - 1].y - center.y) + cos(alpha) \* (vertices[i - 1].x - center.x) + center.x;

vertices[i].y = cos(alpha) \* (vertices[i - 1].y - center.y) + sin(alpha) \* (vertices[i - 1].x - center.x) + center.y;

}

//is >> this->vertices[0] >> this->vertices[1] >> this->vertices[2];

if (is.fail()) {

std::cerr << "Bad input!" << std::endl;

}

}

[[nodiscard]] Point2D Triangle::GetCentroid() const {

return Point2D((vertices[0].x + vertices[1].x + vertices[2].x) / 3,

(vertices[0].y + vertices[1].y + vertices[2].y) / 3);

}

[[nodiscard]] double Triangle::GetArea() const {

double a = (vertices[0].x - vertices[2].x) \* (vertices[1].y - vertices[2].y);

double b = (vertices[0].y - vertices[2].y) \* (vertices[1].x - vertices[2].x);

double c = (a - b) / 2.0;

if (c >= 0) {

return c;

}

else {

return -c;

}

}

square.cpp

#include "figure.hpp"

Square::Square(const Point2D& a, const Point2D& b) {

this->vertices.assign({ a,b });

this->name = "square";

if (DEBUG) std::cout << "constructor called for " << this->name << " with vertices: "

<< vertices.at(0) << ' '

<< vertices.at(1) << std::endl;

}

Square::~Square() {

if (DEBUG) std::cout << "destructor called for " << this->name << " with vertices: "

<< vertices.at(0) << ' '

<< vertices.at(1) << std::endl;

}

void Square::Read(std::istream& is) {

this->vertices.resize(2);

Point2D center;

is >> center >> vertices[0];

double alpha = M\_PI;

vertices[1].x = -sin(alpha) \* (vertices[0].y - center.y) + cos(alpha) \* (vertices[0].x - center.x) + center.x;

vertices[1].y = cos(alpha) \* (vertices[0].y - center.y) + sin(alpha) \* (vertices[0].x - center.x) + center.y;

if (is.fail()) {

std::cerr << "Bad input!" << std::endl;

}

}

[[nodiscard]] Point2D Square::GetCentroid() const {

return Point2D(

(vertices[0].x + vertices[1].x) / 2,

(vertices[0].y + vertices[1].y) / 2);

}

[[nodiscard]] double Square::GetArea() const {

return abs((vertices[0].x - vertices[1].x) \* (vertices[0].y - vertices[1].y));

}

rectangle.cpp

#include "figure.hpp"

Rectangle::Rectangle(const Point2D& a, const Point2D& b) {

this->vertices.assign({ a,b });

this->name = "rectangle";

if (DEBUG) std::cout << "constructor called for Rectangle with vertices: "

<< vertices.at(0) << ' '

<< vertices.at(1) << std::endl;

}

Rectangle::~Rectangle() {

if (DEBUG) std::cout << "destructor called for Rectangle with vertices: "

<< vertices.at(0) << ' '

<< vertices.at(1) << std::endl;

}

void Rectangle::Read(std::istream& is) {

this->vertices.resize(2);

Point2D center;

is >> center >> vertices[0];

double alpha = M\_PI;

vertices[1].x = -sin(alpha) \* (vertices[0].y - center.y) + cos(alpha) \* (vertices[0].x - center.x) + center.x;

vertices[1].y = cos(alpha) \* (vertices[0].y - center.y) + sin(alpha) \* (vertices[0].x - center.x) + center.y;

if (is.fail()) {

std::cerr << "Bad input!" << std::endl;

}

}

[[nodiscard]] Point2D Rectangle::GetCentroid() const {

return Point2D(

(vertices[0].x + vertices[1].x) / 2,

(vertices[0].y + vertices[1].y) / 2);

}

[[nodiscard]] double Rectangle::GetArea() const {

return abs((vertices[0].x - vertices[1].x) \* (vertices[0].y - vertices[1].y));

}

factory.hpp

#pragma once

#include <memory>

#include <functional>

#include <map>

#include <string>

#include "figure.hpp"

template <typename T>

class Factory {

public:

bool IsRegistered(std::string typeRepr) {

return (registered.find(typeRepr) != registered.end());

}

void RegisterClass(std::string typeRepr, std::function<T\* ()> typeCreator) {

if (registered.find(typeRepr) != registered.end()) {

throw std::runtime\_error("Already registered!");

}

registered.insert(std::make\_pair(typeRepr, typeCreator));

}

std::unique\_ptr<T> Create(std::string typeRepr) {

for (auto& pair : registered) {

if (pair.first == typeRepr) {

auto typeCreator = registered.at(typeRepr);

return std::unique\_ptr<T>(typeCreator());

}

}

throw std::runtime\_error("No such creator!");

}

static Factory& GetInstance() {

static Factory<T> factory;

return factory;

}

private:

Factory() = default;

std::map < std::string, std::function<T\* ()>> registered;

};

document.hpp

#pragma once

#include <vector>

#include <stack>

#include <fstream>

#include <variant>

#include "factory.hpp"

#include "figure.hpp"

class Document {

public:

Document();

~Document() = default;

bool Create(std::string name = "unnamed");

void Save(std::string name);

void Load(std::string name);

void Add(std::unique\_ptr<Figure>&& figure);

void Remove(uint32\_t index);

void Print();

void Undo();

void Redo();

private:

std::string name;

std::vector<std::unique\_ptr<Figure>> container;

std::fstream file;

Factory<Figure>& factory = Factory<Figure>::GetInstance();

using vertices = std::pair<Point2D, Point2D>;

using figure\_cmd = std::pair<std::string, vertices>;

using cmd\_and\_index = std::pair<figure\_cmd, uint32\_t>;

using command = std::pair<uint32\_t, cmd\_and\_index>;

std::stack<command> undo\_history;

std::stack<command> redo\_history;

command ComposeCommand(uint32\_t option, uint32\_t index, Figure\* figure);

};

document.cpp

#include "document.hpp"

#include <sstream>

Document::Document() {

if (!factory.IsRegistered("triangle"))

factory.RegisterClass("triangle", FigureCreator<Triangle>::Create);

if (!factory.IsRegistered("square"))

factory.RegisterClass("square", FigureCreator<Square>::Create);

if (!factory.IsRegistered("rectangle"))

factory.RegisterClass("rectangle", FigureCreator<Rectangle>::Create);

}

bool Document::Create(std::string name) {

std::cout << "This unsaved document will be lost, are you sure about that? (yes/no)" << std::endl;

std::string ans;

std::cin >> ans;

if (ans != "yes")

return false;

this->name = name;

container.clear();

std::cout << "Document created" << std::endl;

return true;

}

void Document::Save(std::string name) {

file.open(name, std::fstream::out);

if (file.is\_open()) {

for (auto& elem : container) {

file << elem->GetName() << ' ';

auto center = elem->GetCentroid();

file << center.x << ' ' << center.y << ' ';

auto vertices = elem->GetVertices();

file << vertices[0].x << ' ' << vertices[0].y << ' ';

file << '\n';

}

file.close();

}

else throw std::runtime\_error("File couldn't be opened!");

}

void Document::Load(std::string name) {

file.open(name, std::fstream::in);

if (file.is\_open()) {

while (!file.eof()) {

std::string figureType;

file >> figureType;

if (figureType.length() == 0)

break;

container.emplace\_back(factory.Create(figureType));

file >> \*(container.back());

}

file.close();

}

else throw std::runtime\_error("File couldn't be opened!");

}

void Document::Add(std::unique\_ptr<Figure>&& figure) {

try {

auto figureName = figure->GetName();

container.emplace\_back(std::move(figure));

}

catch (std::exception& ex) {

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

}

undo\_history.emplace(ComposeCommand(1, container.size() - 1, container.back().get()));

}

void Document::Remove(uint32\_t index) {

if (index >= container.size()) {

throw std::out\_of\_range("Bad index provided!");

}

auto figure = container.at(index).get();

undo\_history.emplace(ComposeCommand(2, index, figure));

container.erase(container.begin() + index);

}

void Document::Print() {

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

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

std::cout << '\t' << \*(container[i]) << std::endl;

}

void Document::Undo() {

if (undo\_history.size() == 0)

throw std::runtime\_error("Nothing to redo!");

command cmd = undo\_history.top();

undo\_history.pop();

if (cmd.first == 2) {

auto figure = factory.Create(cmd.second.first.first);

std::stringstream ss;

ss

<< cmd.second.first.second.first.x << ' '

<< cmd.second.first.second.first.y << ' '

<< cmd.second.first.second.second.x << ' '

<< cmd.second.first.second.second.y;

ss >> \*(figure.get());

container.insert(container.begin() + cmd.second.second, std::move(figure));

}

else if (cmd.first == 1)

container.erase(container.begin() + cmd.second.second);

else throw std::runtime\_error("Corrupted history!");

redo\_history.push(cmd);

}

void Document::Redo() {

if (redo\_history.size() == 0)

throw std::runtime\_error("Nothing to redo!");

command cmd = redo\_history.top();

redo\_history.pop();

if (cmd.first == 1) {

auto figure = factory.Create(cmd.second.first.first);

std::stringstream ss;

ss

<< cmd.second.first.second.first.x << ' '

<< cmd.second.first.second.first.y << ' '

<< cmd.second.first.second.second.x << ' '

<< cmd.second.first.second.second.y;

ss >> \*(figure.get());

container.insert(container.begin() + cmd.second.second, std::move(figure));

}

else if (cmd.first == 2)

container.erase(container.begin() + cmd.second.second);

else throw std::runtime\_error("Corrupted history!");

undo\_history.push(cmd);

}

Document::command Document::ComposeCommand(uint32\_t option, uint32\_t index, Figure\* figure) {

return command(option, cmd\_and\_index(figure\_cmd(figure->GetName(), vertices(figure->GetCentroid(), figure->GetVertices().at(0))), index));

}

1. Вывод

При проектировании структуры классов очень важно как можно раньше определить основные сущности программы, их поведение, и взаимоотношение — только тогда можно будет понять как писать код так, чтобы потом не пришлось его переписывать, то есть чтобы он был не только эффективным и красивым, но и удобным, масштабируемым, стабильным…