**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: Бирюков Виктор Владимирович

Группа: 80-207

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, 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 17: треугольник, квадрат, прямоугольник.

1. Описание программы

Классы “графических” примитивов Triangle, Square, Rectangle наследуют абстрактый класс Figure, что позволяет единообразно их использовать и хранить в одной коллекции.

Ввод и вывод фигур осуществляется при помощи класса Factory, который позволяет вводить/выводить фигуры как в стандартные потоки, так и в бинарные файлы. Тип фигур определяется при помощи перечислимого типа Figures.

Класс Document определяет поведение документа. Документ хранить список фигур, позволяя добавлять и удалять фигуры по индексам. Функции сохранения и загрузки из файла записывают/считывают количество фигур, а затем сами фигуры. Отмена операций реализована через сохранение в стеке противодействий к совершённым действиям, действия наследуют абстрактный класс Action и содержат метод apply, который применяет действие к списку фигур.

1. Набор тестов

Программа принимает на вход команды, вводимые пользователем. Реализованы следующие команды: создание нового документа (create), добавление фигуры в список фигур по индексу (add INDEX FIGURE), удаление фигуры из списка по индексу (remove INDEX), отмена последнего действия (undo), печать всех фигур (print), сохранение документа в файл (save NAME), загрузка документа из файла (load NAME).

Для ввода фигур необходимо указать тип фигуры (triangle, square, rectangle) и данные о фигуре (координаты левой нижней вершины и длина стороны (сторон))

Тест 1:

> create

> add 0

triangle 1 1 1

> add 1

square 0 0 5

> add 1

rectangle 2 2 3 4

> print

Triangle:

(1, 1) (1.5, 1.86603) (2, 1)

Rectangle:

(2, 2) (2, 6) (5, 6) (5, 2)

Square:

(0, 0) (0, 5) (5, 5) (5, 0)

> undo

> print

Triangle:

(1, 1) (1.5, 1.86603) (2, 1)

Square:

(0, 0) (0, 5) (5, 5) (5, 0)

> add 0

rectangle 0 0 3 4

> print

Rectangle:

(0, 0) (0, 4) (3, 4) (3, 0)

Triangle:

(1, 1) (1.5, 1.86603) (2, 1)

Square:

(0, 0) (0, 5) (5, 5) (5, 0)

> save 3figures

> create

> print

> load 3figures

> print

Rectangle:

(0, 0) (0, 4) (3, 4) (3, 0)

Triangle:

(1, 1) (1.5, 1.86603) (2, 1)

Square:

(0, 0) (0, 5) (5, 5) (5, 0)

> remove 1

> remove 0

> print

Square:

(0, 0) (0, 5) (5, 5) (5, 0)

> save 1square

> undo

> undo

> print

Rectangle:

(0, 0) (0, 4) (3, 4) (3, 0)

Triangle:

(1, 1) (1.5, 1.86603) (2, 1)

Square:

(0, 0) (0, 5) (5, 5) (5, 0)

> load 1square

> print

Square:

(0, 0) (0, 5) (5, 5) (5, 0)

> exit

Тест 2 (исключительные ситуации):

> add -1 square 0 0 0

Error: invalid index for insert

> add 0 square 0 0 0

Error: Invalid square parameters

> add 0 square 0 0 1

> remove 2

Error: invalid index for remove

> print

Square:

(0, 0) (0, 1) (1, 1) (1, 0)

> undo

> undo

Nothing to undo

> print

> load unexisted\_file

Error: Error opening file

> add 0 cube

Error: Unknown figure type

> exit

1. Листинг программы

// Абстрактный класс фигуры, а также перечислимый тип со всеми фигурами.

#pragma once

#include <fstream>

#include <iostream>

enum Figures : int {

triangle = 0,

square = 1,

rectangle = 2

};

class Figure {

public:

virtual void print(std::ostream&) = 0;

virtual void write(std::ofstream&) = 0;

virtual void read(std::ifstream&) = 0;

virtual Figures type() = 0;

virtual ~Figure() = default;

};

// Прямоугольник. Хранит данные как координаты левой нижней вершины и длины сторон.

#pragma once

#include <utility>

#include <fstream>

#include <iostream>

#include <stdexcept>

#include "figure.hpp"

template <typename T>

class Rectangle : public Figure {

public:

std::pair<T,T> x;

T a;

T b;

Rectangle() = default;

Rectangle(T x1, T x2, T a, T b) : x(x1,x2), a(a), b(b) {

if (a <= 0 || b <= 0) {

throw std::invalid\_argument("Invalid rectangle parameters");

}

}

~Rectangle() = default;

void print(std::ostream& os) override {

os << "(" << x.first << ", " << x.second << ") " <<

"(" << x.first << ", " << x.second + b << ") " <<

"(" << x.first + a << ", " << x.second + b << ") " <<

"(" << x.first + a << ", " << x.second << ")" << std::endl;

}

Figures type() override {

return Figures::rectangle;

}

void read(std::ifstream& ifs) override {

ifs.read(reinterpret\_cast<char\*>(&x.first), sizeof(T));

ifs.read(reinterpret\_cast<char\*>(&x.second), sizeof(T));

ifs.read(reinterpret\_cast<char\*>(&a), sizeof(T));

ifs.read(reinterpret\_cast<char\*>(&b), sizeof(T));

}

void write(std::ofstream& ofs) override {

ofs.write(reinterpret\_cast<char\*>(&x.first), sizeof(T));

ofs.write(reinterpret\_cast<char\*>(&x.second), sizeof(T));

ofs.write(reinterpret\_cast<char\*>(&a), sizeof(T));

ofs.write(reinterpret\_cast<char\*>(&b), sizeof(T));

}

template <class U>

friend std::istream& operator>>(std::istream&, Rectangle<U>&);

};

template <class T>

std::istream& operator>>(std::istream& is, Rectangle<T>& rect) {

is >> rect.x.first >> rect.x.second >> rect.a >> rect.b;

if (rect.a <= 0 || rect.b <= 0) {

throw std::invalid\_argument("Invalid rectangle parameters");

}

return is;

}

// Квадрат. Хранит данные как координаты левой нижней вершины и длину стороны.

#pragma once

#include <utility>

#include <fstream>

#include <iostream>

#include <stdexcept>

#include "figure.hpp"

template <typename T>

class Square : public Figure {

public:

std::pair<T,T> x;

T a;

Square() = default;

Square(T x1, T x2, T a) : x(x1,x2), a(a) {

if (a <= 0) {

throw std::invalid\_argument("Invalid square parameters");

}

}

~Square() = default;

void print(std::ostream& os) override {

os << "(" << x.first << ", " << x.second << ") " <<

"(" << x.first << ", " << x.second + a << ") " <<

"(" << x.first + a << ", " << x.second + a << ") " <<

"(" << x.first + a << ", " << x.second << ")" << std::endl;

}

Figures type() override {

return Figures::square;

}

void read(std::ifstream& ifs) override {

ifs.read(reinterpret\_cast<char\*>(&x.first), sizeof(T));

ifs.read(reinterpret\_cast<char\*>(&x.second), sizeof(T));

ifs.read(reinterpret\_cast<char\*>(&a), sizeof(T));

}

void write(std::ofstream& ofs) override {

ofs.write(reinterpret\_cast<char\*>(&x.first), sizeof(T));

ofs.write(reinterpret\_cast<char\*>(&x.second), sizeof(T));

ofs.write(reinterpret\_cast<char\*>(&a), sizeof(T));

}

template <class U>

friend std::istream& operator>>(std::istream&, Square<U>&);

};

template <class T>

std::istream& operator>>(std::istream& is, Square<T>& sq) {

is >> sq.x.first >> sq.x.second >> sq.a;

if (sq.a <= 0) {

throw std::invalid\_argument("Invalid square parameters");

}

return is;

}

// Треугольник. Хранит данные как координаты левой вершины и длину стороны.

#pragma once

#include <cmath>

#include <utility>

#include <fstream>

#include <iostream>

#include <stdexcept>

#include "figure.hpp"

template <typename T>

class Triangle : public Figure {

public:

std::pair<T,T> x;

T a;

Triangle() = default;

Triangle(T x1, T x2, T a) : x(x1,x2), a(a) {

if (a <= 0) {

throw std::invalid\_argument("Invalid triangle parameters");

}

}

~Triangle() = default;

void print(std::ostream& os) override {

os << "(" << x.first << ", " << x.second << ") " <<

"(" << x.first + 1.0 / 2 \* a << ", " <<

x.second + sqrt(3) / 2 \* a << ") " <<

"(" << x.first + a << ", " << x.second << ")" << std::endl;

}

Figures type() override {

return Figures::triangle;

}

void read(std::ifstream& ifs) override {

ifs.read(reinterpret\_cast<char\*>(&x.first), sizeof(T));

ifs.read(reinterpret\_cast<char\*>(&x.second), sizeof(T));

ifs.read(reinterpret\_cast<char\*>(&a), sizeof(T));

}

void write(std::ofstream& ofs) override {

ofs.write(reinterpret\_cast<char\*>(&x.first), sizeof(T));

ofs.write(reinterpret\_cast<char\*>(&x.second), sizeof(T));

ofs.write(reinterpret\_cast<char\*>(&a), sizeof(T));

}

template <class U>

friend std::istream& operator>>(std::istream&, Triangle<U>&);

};

template <class T>

std::istream& operator>>(std::istream& is, Triangle<T>& tr) {

is >> tr.x.first >> tr.x.second >> tr.a;

if (tr.a <= 0) {

throw std::invalid\_argument("Invalid triangle parameters");

}

return is;

}

// Класс, содержащий функции для ввода/вывода фигур в различные виды стандартных потоков.

#pragma once

#include <memory>

#include <fstream>

#include <iostream>

#include <stdexcept>

#include "figure.hpp"

#include "triangle.hpp"

#include "square.hpp"

#include "rectangle.hpp"

template <class T>

class Factory {

public:

std::shared\_ptr<Figure> create(std::istream& is) const {

std::string type;

is >> type;

std::shared\_ptr<Figure> figure;

if (type == "triangle") {

Triangle<T>\* tr = new Triangle<T>;

is >> \*tr;

figure =

std::shared\_ptr<Figure>(reinterpret\_cast<Figure\*>(tr));

}

else if (type == "square") {

Square<T>\* sq = new Square<T>;

is >> \*sq;

figure =

std::shared\_ptr<Figure>(reinterpret\_cast<Figure\*>(sq));

}

else if (type == "rectangle") {

Rectangle<T>\* rect = new Rectangle<T>;

is >> \*rect;

figure =

std::shared\_ptr<Figure>(reinterpret\_cast<Figure\*>(rect));

}

else {

throw std::runtime\_error("Unknown figure type");

}

return figure;

}

void print(const std::shared\_ptr<Figure>& figure, std::ostream& os) {

switch (figure->type()) {

case Figures::triangle:

os << "Triangle:\n";

break;

case Figures::square:

os << "Square:\n";

break;

case Figures::rectangle:

os << "Rectangle:\n";

break;

}

figure->print(os);

}

std::shared\_ptr<Figure> load(std::ifstream& ifs) const {

if (!ifs) {

throw std::runtime\_error("File is over");

}

int type;

ifs.read(reinterpret\_cast<char\*>(&type), sizeof(type));

std::shared\_ptr<Figure> figure;

switch (type) {

case Figures::triangle: {

Triangle<T>\* tr = new Triangle<T>;

tr->read(ifs);

figure =

std::shared\_ptr<Figure>(reinterpret\_cast<Figure\*>(tr));

break;

}

case Figures::square: {

Square<T>\* sq = new Square<T>;

sq->read(ifs);

figure =

std::shared\_ptr<Figure>(reinterpret\_cast<Figure\*>(sq));

break;

}

case Figures::rectangle: {

Rectangle<T>\* rect = new Rectangle<T>;

rect->read(ifs);

figure =

std::shared\_ptr<Figure>(reinterpret\_cast<Figure\*>(rect));

break;

}

}

return figure;

}

void save(const std::shared\_ptr<Figure>& figure, std::ofstream& ofs) const {

int type = figure->type();

ofs.write(reinterpret\_cast<char\*>(&type), sizeof(type));

figure->write(ofs);

}

Factory() = default;

~Factory() = default;

};

// Класс документа, хранит фигуры в списке.

#pragma once

#include <list>

#include <stack>

#include <memory>

#include <string>

#include <fstream>

#include <iostream>

#include <stdexcept>

#include <algorithm>

#include "figure.hpp"

#include "triangle.hpp"

#include "square.hpp"

#include "rectangle.hpp"

#include "factory.hpp"

struct Action {

virtual ~Action() = default;

virtual void apply(std::list<std::shared\_ptr<Figure>>&) = 0;

};

struct AddAction : public Action {

std::shared\_ptr<Figure> figure;

size\_t index;

AddAction(std::shared\_ptr<Figure> figure, size\_t index) : figure(figure), index(index) {}

~AddAction() = default;

void apply(std::list<std::shared\_ptr<Figure>>& figures) override {

auto iter = figures.begin();

while (index--) {

iter++;

}

figures.insert(iter, figure);

}

};

struct DelAction : public Action {

size\_t index;

DelAction(size\_t index) : index(index) {}

~DelAction() = default;

void apply(std::list<std::shared\_ptr<Figure>>& figures) override {

auto iter = figures.begin();

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

iter++;

}

figures.erase(iter);

}

};

template <class T>

class Document {

private:

std::list<std::shared\_ptr<Figure>> figures;

std::stack<std::shared\_ptr<Action>> actions;

Factory<T> factory;

public:

void create() {

figures.clear();

while (!actions.empty()) {

actions.pop();

}

}

void add(int index) {

if (index < 0) {

throw std::out\_of\_range("Invalid index for insert");

}

std::shared\_ptr<Figure> figure = factory.create(std::cin);

actions.push(std::shared\_ptr<Action>(new DelAction(std::min(index,

(int)figures.size()))));

if (index >= (int)figures.size()) {

figures.push\_back(figure);

} else {

auto iter = figures.begin();

while (index--) {

iter++;

}

figures.insert(iter, figure);

}

}

void remove(int index) {

if (index < 0 || index >= (int)figures.size()) {

throw std::out\_of\_range("Invalid index for remove");

}

auto iter = figures.begin();

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

iter++;

}

actions.push(std::shared\_ptr<Action>(new AddAction(\*iter, index)));

figures.erase(iter);

}

void undo() {

if (actions.empty()) {

std::cout << "Nothing to undo" << std::endl;

return;

}

actions.top()->apply(figures);

actions.pop();

}

void save(std::string& name) {

std::ofstream ofs(name, std::ios::binary);

if (ofs.fail()) {

throw std::runtime\_error("Error opening file");

}

size\_t size = figures.size();

ofs.write(reinterpret\_cast<char\*>(&size), sizeof(size));

for (const std::shared\_ptr<Figure>& figure : figures) {

factory.save(figure, ofs);

}

ofs.close();

}

void load(std::string& name) {

create();

std::ifstream ifs(name, std::ios::binary);

if (ifs.fail()) {

throw std::runtime\_error("Error opening file");

}

size\_t size;

ifs.read(reinterpret\_cast<char\*>(&size), sizeof(size));

while (size--) {

figures.push\_back(factory.load(ifs));

}

ifs.close();

}

void print() {

for (const std::shared\_ptr<Figure>& figure : figures) {

factory.print(figure, std::cout);

}

}

Document() : figures(), actions(), factory() {};

~Document() = default;

};

/\*

Бирюков В.В. М8О-107Б-19

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, 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 17: треугольник, квадрат, прямоугольник.

\*/

#include <string>

#include <iostream>

#include "document.hpp"

int main() {

std::string command;

Document<int> document;

std::cout << "> ";

while (std::cin >> command && command != "exit") {

try {

if (command == "help") {

std::cout << "Usage:\n"

<< "help show this message\n"

<< "create create new document\n"

<< "add INDEX FIGURE add figure to document\n"

<< "remove INDEX remove figure from

document\n"

<< "undo cancel previous

operation\n"

<< "print print list of figures\n"

<< "save NAME save document to file\n"

<< "load NAME load document from file\n"

<< "exit exit program\n"

<< "FIGURE: TYPE COORDS\n"

<< "TYPE: triangle, square, rectangle\n"

<< "COORDS: triangle: X Y A, square: X Y A,

rectangle: X Y A B\n";

}

else if (command == "create") {

document.create();

}

else if (command == "add") {

int index;

std::cin >> index;

document.add(index);

}

else if (command == "remove") {

int index;

std::cin >> index;

document.remove(index);

}

else if (command == "undo") {

document.undo();

}

else if (command == "print") {

document.print();

}

else if (command == "save") {

std::string filename;

std::cin >> filename;

document.save(filename);

}

else if (command == "load") {

std::string filename;

std::cin >> filename;

document.load(filename);

}

else {

throw std::runtime\_error("Unknown command");

}

std::cout << "> ";

} catch (const std::exception& e) {

std::cout << "Error: " << e.what() << std::endl;

std::cout << "> ";

}

}

}

1. Выводы

В ходе лабораторной работы я познакомился с различными способами проектирования классов, а также применил некоторые из них на практике.

Литература

1. Справочник по языку C++ [Электронный ресурс]. URL: <https://ru.cppreference.com> (дата обращения: 11.12.20).