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: Суханов Егор Алексеевич
Группа: 80-206
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.

Так как мой вариант задачи 21, то мне нужно работать с ромбом, 5-ти и 6-ти угольниками.

Разобьем задачу на подзадачи:

- Разработка командного интерфейса:
    - Создать фигуру;
    - Удалить фигуру;
    - Отмена последнего действия;
    - Вывод примитивов на экран.
- Нужно использовать паттерн Factory для создания примитивов;
- Работа с файлами:
    - Загрузка;

- Сохранение.

Очень важно с самого начала спроектировать хороший дизайн. Для этого я хочу во-первых ознакомится с предложенным паттерном: Фабрика. А также с паттернами, которые, по-моему мнению, могут помочь улучшить архитектуру кода.

Паттерн “абстрактная фабрика” служит для имплементации создания конкретных классов. В абстрактном классе “фабрика” объявлены функции, которые создают абстрактные продукты, а в классах наследниках этой фабрики, эти функции определяются. Это позволяет добавлять новое поведение конструирования уже существующих продуктов. Если же требуется добавить новый продукт, то нужно изменить все классы. В итоге этот паттерн удобен, если нужно создавать объекты по разным стратегиям.

Паттерн “фабричный метод” служит для имплементации создания класса. Отличие от “абстрактной фабрики” заключается в имплементации создания не группы классов, а только одного. Этот паттерн позволяет незаметно для внешнего мира создавать разные виды продуктов.

Паттерн “команда” позволяет имплементировать выполнение какой-то команды. Код, который выполняет какую-то команду, может не знать о том, что он делает. Кроме этого, данный паттерн позволяет строить цепочки команд, передавать их. А если добавить функцию отмены изменений, можно реализовать операцию отмены. Этот паттерн удобен, если нужно отделить место создания команды с местом ее выполнения.

Для реализации класса Factory будем использовать паттерн “фабричный метод”. Это позволит достаточно легко добавлять новые виды фигур. А для операции отмены будем использовать паттерн “команда”.

2. Описание программы

Точка входа и интерактивная обработка команд находится в файле **main.cpp**:

- Функция main. Считывание и выполнение интерактивных команд. Для удобства, команды вынесены в отдельные функции;
- Семейство классов Command. Реализация паттерна “команда”. С их помощью достаточно легко реализовать отмену команд.

Для хранения команд используется стек. При вводе команды добавления или удаления фигуры, создается экземпляр соответствующего класса-наследника Command. После чего данная команда выполняется. И добавляется в стек команд. Фигуры же хранятся в векторе фигур. Если команду нужно отменить, то сначала выполняется отменяющая функция у команды, а затем она удаляется из стека команд.

Файлы **figure.hpp**, **figure.cpp**, **point.hpp** и **point.cpp** имплементируют работу с фигурами и вершинами. Большая часть кода была взята из 3-ей ЛР. Была добавлена функция Save для сохранения фигуры. Функцию загрузки имплементирует фабричный метод. Возможно это не самый хороший ход. Но, зато, менее затратный.

Файлы **figure_factory.hpp** и **figure_factory.cpp** имплементируют паттерн “фабричный метод”. Данные классы создают различные типы фигур:

- Класс FigureInputMaker строит фигуры из потока ввода;
- Класс FigureLoadMaker строит фигуры из бинарного представления;

Алгоритм их работы схож: сначала считывается тип фигуры и координаты вершин. Затем происходит вызов соответствующего конструктора. Если требуется модифицировать семейство фигур, нужно реализовать метод Load внутри классов фигур.

3. Набор тестов

test_01.txt - проверка добавления, удаления фигур и операции отмена.

```
add r 0 0 1 1

add h 0 0 1 1

add p 0 0 1 1

print

remove 2

remove 0

remove 0

print

undo

print

add r 0 0 2 2

print

undo

undo

undo

print

undo

undo

undo

undo

print
```

Результат выполнения теста:

```
Фигура добавлена!
Фигура добавлена!
Фигура добавлена!
Фигура 0 :Ромб, координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Фигура 1 :Шестиугольник, координаты: (1,1) (-0.366025,1.36603)
(-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Фигура 2 :Пятиугольник, координаты: (1,1) (-0.64204,1.26007)
(-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Фигура 2 удалена!
Фигура 0 удалена!
Фигура 0 удалена!
Удаление фигуры отменнено!
```

```
Фигура    0    :Шестиугольник,    координаты:    (1,1)    (-0.366025,1.36603)
(-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Фигура добавлена!
Фигура    0    :Шестиугольник,    координаты:    (1,1)    (-0.366025,1.36603)
(-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Фигура 1 :Ромб, координаты: (2,2) (-2,2) (-2,-2) (2,-2)
Добавление фигуры отмененно!
Удаление фигуры отменнено!
Удаление фигуры отменнено!
Фигура 0 :Ромб, координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Фигура    1    :Шестиугольник,    координаты:    (1,1)    (-0.366025,1.36603)
(-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
Фигура    2    :Пятиугольник,    координаты:    (1,1)    (-0.64204,1.26007)
(-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Добавление фигуры отмененно!
Добавление фигуры отмененно!
Добавление фигуры отмененно!
Стек команд уже пустой
```

test_02.txt - тестирование сохранения и загрузки.

```
save test
load test
add r 0 0 1 1
add p 0 0 1 1
add h 0 0 1 1
save test
load test
print
```

Результаты выполнения:

```
0 кол-во фигур
Фигура добавлена!
Фигура добавлена!
Фигура добавлена!
>save test
>load test
3 кол-во фигур
Ромб, координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Пятиугольник,  координаты:  (1,1)  (-0.64204,1.26007)  (-1.3968,-0.221232)
(-0.221232,-1.3968) (1.26007,-0.64204)
Шестиугольник,  координаты:  (1,1)  (-0.366025,1.36603)  (-1.36603,0.366025)
(-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)
>print
Фигура 0 :Ромб, координаты: (1,1) (-1,1) (-1,-1) (1,-1)
Фигура    1    :Пятиугольник,    координаты:    (1,1)    (-0.64204,1.26007)
(-1.3968,-0.221232) (-0.221232,-1.3968) (1.26007,-0.64204)
Фигура    2    :Шестиугольник,    координаты:    (1,1)    (-0.366025,1.36603)
```

(-1.36603,0.366025) (-1,-1) (0.366025,-1.36603) (1.36603,-0.366025)

4. Листинг программы

Исходный код ЛР можно найти на github: https://github.com/Reterer/oop_exercise_07 .

main.cpp:

```
/*
  Лабораторная работа: 7
  Вариант: 21
  Группа: М8О-206Б-19
  Автор: Суханов Егор Алексеевич

  Спроектировать простейший «графический» векторный редактор.
  Редактор должен уметь создавать, выводить, удалять фигуры, делать отмену
  действия.


  Фигуры:
      Ромб, 5-угольник, 6-угольник
*/
#include <stack>
#include <vector>
#include <string>
#include <iostream>
#include <fstream>

#include "figure.hpp"
#include "figure_factory.hpp"

class Command {
public:
      virtual void Execute() = 0;
      virtual void UnExecute() = 0;
};


class CommandAddFigure : public Command {
public:
      CommandAddFigure(std::vector<Figure*>& figures, Figure* figure)
            : figures(figures), figure(figure)
      {};

      virtual void Execute() override;
      virtual void UnExecute() override;

private:
      std::vector<Figure*>& figures;
      Figure* figure;
};

void CommandAddFigure::Execute() {
      figures.push_back(figure);
```

```
	std::cout << "Фигура добавлена!\n";
}
void CommandAddFigure::UnExecute() {
	figures.pop_back();
	std::cout << "Добавление фигуры отмененно!\n";
}

class CommandRemoveFigure : public Command {
public:
	CommandRemoveFigure(std::vector<Figure*>& figures, int idx)
		: figures(figures), idx(idx)
	{};

	virtual void Execute() override;
	virtual void UnExecute() override;

private:
	std::vector<Figure*>& figures;
	Figure* figure;
	int idx;
};

void CommandRemoveFigure::Execute() {
	figure = figures[idx];
	figures.erase(figures.begin() + idx);
	std::cout << "Фигура " << idx << " удалена!\n";
}
void CommandRemoveFigure::UnExecute() {
	figures.insert(figures.begin() + idx, figure);
	std::cout << "Удаление фигуры отменнено!\n";
}

void clear() {
	std::cin.clear();
	std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
}
bool ansYesNo() {
	std::string ans;
	while (std::cin >> ans) {
		if (ans == "y")
			return true;
		else if (ans == "n")
			return false;

		std::cout << "Введите \"y\" (да) или \"n\" (нет)\n";
	}
	std::cout << "Произошла какая-то ошибка!\n";
	clear();
	return false;
}

void help() {
	std::cout <<
		"help - выводит эту справку.\n"
		"exit - выход;\n"
		"save <path> - сохранение сессии;\n"
		"load <path> - загрузка сессии;\n"
		"print - вывести список фигур;\n"
```

```
            "add <r|p|h> <center> <vertex> - добавить фигуру
(ромб|пяти|шестиугольник)\n"
            "                  с координатами центра в <center> и первой
вершиной в <vertex>;\n"
            "remove <id> - удалить фигуру с индексом <id>;\n"
            "undo - отменить последнее действие.\n";
}

void print(const std::vector<Figure*>& figures) {
      for (int i = 0; i < figures.size(); ++i) {
            std::cout << "Фигура " << i << " :";
            figures[i]->Print();
      }
}

void save(const std::vector<Figure*>& figures) {
      std::string fileName;
      if (!std::getline(std::cin, fileName)) {
            std::cout << "Ошибка ввода имени файла\n";
            clear();
            return;
      }
      if (std::ofstream ofs = std::ofstream(fileName, std::fstream::out |
std::fstream::binary | std::fstream::trunc)) {
            size_t size = figures.size();
            ofs.write(reinterpret_cast<char*>(&size), sizeof(size));
            for (auto f : figures) {
                  if (!f->Save(ofs)) {
                        std::cout << "Какая-то ошибка при сохранении
фигуры\n";
                        return;
                  }
            }
            ofs.close();
      }
      else {
            std::cout << "Ошибка при открытии файла\n";
            return;
      }
}

void load(std::vector<Figure*>& figures, std::stack<Command*>& execCommands)
{
      std::string fileName;
      if (!std::getline(std::cin, fileName)) {
            std::cout << "Ошибка ввода имени файла\n";
            clear();
            return;
      }
      if (std::ifstream ifs = std::ifstream(fileName, std::fstream::in |
std::fstream::binary)) {
            std::vector<Figure*> newFigures;
            FigureMaker* loaderFigures = new FigureLoadMaker(ifs);

            size_t figure_count;
            ifs.read(reinterpret_cast<char*>(&figure_count),
sizeof(figure_count));
            std::cout << figure_count << " кол-во фигур\n";
```

```
			try {
				while (Figure* figure = loaderFigures->Make()) {
					newFigures.push_back(figure);
					figure->Print();
				}
			}
			catch (std::invalid_argument& e) {
				std::cout << "Ошибка: " << e.what() << "\n";
				return;
			}

			if (figure_count == newFigures.size()) {
				figures = std::move(newFigures);
				execCommands = std::stack<Command*>();
			}
			else {
				std::cout << "Файл сохранен в неверном формате.\nВы
уверенны, что хотите открыть его (y/n)? ";
				if (ansYesNo()) {
					figures = std::move(newFigures);
					execCommands = std::stack<Command*>();
				}
			}
			ifs.close();
		}
		else {
			std::cout << "Ошибка при открытии файла\n";
			return;
		}
}

Command* add(std::vector<Figure*>& figures, FigureMaker* figureMaker) {
		try {
			if (Figure* figure = figureMaker->Make()) {
				return new CommandAddFigure(figures, figure);
			}
			else {
				std::cout << "Фигура задана в неверном формате\n";
				clear();
			}
		}
		catch (std::invalid_argument& e) {
				std::cout << "Ошибка: " << e.what() << "\n";
				return nullptr;
		}
		return nullptr;
}

Command* remove(std::vector<Figure*>& figures) {
		int idx;
		if (!(std::cin >> idx)) {
			std::cout << "Ошибка ввода\n";
			clear();
			return nullptr;
		}
		if (idx < 0 || idx >= figures.size()) {
			std::cout << "Задан неправильный индекс\n";
			clear();
```

```
			return nullptr;
		}

		return new CommandRemoveFigure(figures, idx);
}

void undo(std::stack<Command*>& execCommands) {
		if (!execCommands.empty()) {
			execCommands.top()->UnExecute();
			execCommands.pop();
		}
		else {
			std::cout << "Стек команд уже пустой\n";
		}
}


int main() {
		setlocale(LC_ALL, "russian");

		std::vector<Figure*> figures;
		std::stack<Command*> execCommands;
		FigureMaker* figureMaker = new FigureInputMaker(std::cin);
		bool run = true;

		while (run) {
			Command* figureCommand = nullptr;
			// Обработка пользовательских команд
			{
				std::string cmd;
				std::cout << '>';
				std::cin >> cmd;
				if (cmd == "help") {
					help();
				}
				else if (cmd == "exit") {
					run = false;
				}
				else if (cmd == "save") {
					save(figures);
				}
				else if (cmd == "load") {
					load(figures, execCommands);
				}
				else if (cmd == "print") {
					print(figures);
				}
				else if (cmd == "add") {
					figureCommand = add(figures, figureMaker);
				}
				else if (cmd == "remove") {
					figureCommand = remove(figures);
				}
				else if (cmd == "undo") {
					undo(execCommands);
				}
				else {
					std::cout << "Введена неизвестная команда\n";
```

```
                        clear();
                    }
                }


                // Выполнение команды
                if (figureCommand) {
                    figureCommand->Execute();
                    execCommands.push(figureCommand);
                }
        }

        return 0;
}
```

figure_factory.hpp:

```
#pragma once
#include "figure.hpp"

class FigureMaker {
public:
        virtual Figure* Make() = 0;
};

class FigureInputMaker: public FigureMaker {
public:
        FigureInputMaker(std::istream& in);
        Figure* Make() override;
private:
        std::istream& in;
};

class FigureLoadMaker : public FigureMaker {
public:
        FigureLoadMaker(std::ifstream& ifs);
        Figure* Make() override;
private:
        std::istream& ifs;
};
```

fgiure_factory.cpp:

```
#include <string>
#include "figure_factory.hpp"

FigureInputMaker::FigureInputMaker(std::istream& in) : in{in} {}

Figure* FigureInputMaker::Make() {
        std::string figureType;
        Point center, vertex;
        if(!(in >> figureType >> center >> vertex))
                return nullptr;
```

```
	if(figureType == "r")
		return new Rhombus(center, vertex);
	if(figureType == "p")
		return new Pentagon(center, vertex);
	if(figureType == "h")
		return new Hexagon(center, vertex);

	return nullptr;
}

FigureLoadMaker::FigureLoadMaker(std::ifstream& ifs) : ifs{ ifs } {}
Figure* FigureLoadMaker::Make() {
	std::string figureType;
	Point center, vertex;

	{
		char type;
		bool success = true;
		success &= static cast<bool>(ifs.read(&type, sizeof(type)));
		success                                                          &=
static cast<bool>(ifs.read(reinterpret_cast<char*>(&center),
sizeof(center)));
		success                                                          &=
static cast<bool>(ifs.read(reinterpret_cast<char*>(&vertex),
sizeof(vertex)));

		if (!success)
			return nullptr;

		figureType = type;
	}

	if (figureType == "r")
		return new Rhombus(center, vertex);
	if (figureType == "p")
		return new Pentagon(center, vertex);
	if (figureType == "h")
		return new Hexagon(center, vertex);

	return nullptr;
}
```

point.hpp:

```
#pragma once
#include <iostream>

struct Point {
	double x;
	double y;

	Point rotate(double radians);
```

```
        friend bool operator== (const Point a, const Point b);
        friend bool operator!= (const Point a, const Point b);

        friend Point operator+ (const Point a, const Point b);
        friend Point operator- (const Point a, const Point b);

        friend double operator* (const Point a, const Point b);
        friend Point operator* (const Point a, double b);

        friend std::ostream& operator<< (std::ostream& out, const Point& p);
        friend std::istream& operator>> (std::istream& in, Point& p);
};
```

point.cpp:

```
#include "point.hpp"

bool operator== (const Point a, const Point b)
{
        return a.x == b.x && a.y == b.y;
}
bool operator!= (const Point a, const Point b)
{
        return a.x != b.x || a.y != b.y;
}

Point operator+ (const Point a, const Point b)
{
        return { a.x + b.x, a.y + b.y };
}

Point operator- (const Point a, const Point b)
{
        return { a.x - b.x, a.y - b.y };
}

double operator*(const Point a, const Point b)
{
        return a.x * b.x + a.y * b.y;
}

Point operator*(const Point
a, double b)
{
        return Point{ a.x * b, a.y * b };
}

std::istream& operator>>(std::istream& in, Point& p)
{
        double x, y;
        in >> x >> y;
        p = Point{ x, y };
```

```
	return in;
}

std::ostream& operator<< (std::ostream& out, const Point& p)
{
	out << '(' << p.x << ',' << p.y << ')';
	return out;
}

Point Point::rotate(double radians) {
	Point rotated;
	rotated.x = x * std::cos(radians) - y * std::sin(radians);
	rotated.y = x * std::sin(radians) + y * std::cos(radians);
	return rotated;
}
```

figure.hpp:

```
#pragma once
#include <fstream>
#include "point.hpp"

class Figure {
public:
	Figure(Point center, Point vertex);
	Point GetCenter();
	virtual void Print() = 0;
	virtual bool Save(std::ofstream& ofs) = 0;

protected:
	Point center;
	Point vertex;
};

class Rhombus : public Figure {
public:
	Rhombus(Point center, Point vertex);
	virtual void Print() override;
	virtual bool Save(std::ofstream& ofs) override;
};

class Pentagon : public Figure {
public:
	Pentagon(Point center, Point vertex);
	virtual void Print() override;
	virtual bool Save(std::ofstream& ofs) override;

};

class Hexagon : public Figure {
public:
	Hexagon(Point center, Point vertex);
	virtual void Print() override;
	virtual bool Save(std::ofstream& ofs) override;
};
```

figure.cpp:

```cpp
#include <stdexcept>
#include <iostream>
#include <cmath>
#include "figure.hpp"

const double PI = 3.141592653589793;

Figure::Figure(Point center, Point vertex)
{
	if (center == vertex)
		throw std::invalid_argument("center can be eq vertex");

	this->center = center;
	this->vertex = vertex;
}

Point Figure::GetCenter()
{
	return this->center;
}

template <int VERTEX_COUNT>
void _printCords(const Point center, const Point vertex)
{
	const double ANGLE = 2 * PI / VERTEX_COUNT; // угол между двумя соседними вершинами и центром
	std::cout << "координаты: " << vertex;

	Point vec = vertex - center;
	for (int i = 2; i <= VERTEX_COUNT; ++i)
	{
		vec = vec.rotate(ANGLE);
		std::cout << ' ' << center + vec;
	}
	std::cout << std::endl;
}

template <int VERTEX_COUNT>
double _calcArea(const Point center, const Point vertex)
{
	Point vecRadius = vertex - center; // Вектор-радиус описанной окружности
	double sqRadius = vecRadius * vecRadius; // Квадрат радуиса описанной окружности
	return VERTEX_COUNT / 2. * sqRadius * std::sin(2 * PI / VERTEX_COUNT);
}

Rhombus::Rhombus(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Rhombus::Print()
{
	std::cout << "Ромб, ";
```

```
	_printCords<4>(center, vertex);
}

bool Rhombus::Save(std::ofstream& ofs)
{
	const char type = 'r';
	bool success = true;

	success &= static cast<bool>(ofs.write(&type, sizeof(type)));
	success                                                                    &=
static cast<bool>(ofs.write(reinterpret_cast<char*>(&center),
sizeof(center)));
	success                                                                    &=
static cast<bool>(ofs.write(reinterpret_cast<char*>(&vertex),
sizeof(vertex)));
	return success;
}


Pentagon::Pentagon(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Pentagon::Print()
{
	std::cout << "Пятиугольник, ";
	_printCords<5>(center, vertex);
}

bool Pentagon::Save(std::ofstream& ofs)
{
	const char type = 'p';
	bool success = true;

	success &= static cast<bool>(ofs.write(&type, sizeof(type)));
	success                                                                    &=
static cast<bool>(ofs.write(reinterpret_cast<char*>(&center),
sizeof(center)));
	success                                                                    &=
static cast<bool>(ofs.write(reinterpret_cast<char*>(&vertex),
sizeof(vertex)));
	return success;
}

Hexagon::Hexagon(Point center, Point vertex)
	: Figure(center, vertex)
{}

void Hexagon::Print()
{
	std::cout << "Шестиугольник, ";
	_printCords<6>(center, vertex);
}

bool Hexagon::Save(std::ofstream& ofs)
{
	const char type = 'h';
	bool success = true;
```

```
	success &= static cast<bool>(ofs.write(&type, sizeof(type)));
	success &= static cast<bool>(ofs.write(reinterpret_cast<char*>(&center), sizeof(center)));
	success &= static cast<bool>(ofs.write(reinterpret_cast<char*>(&vertex), sizeof(vertex)));
	return success;
}
```

5. Выводы

Выполняя данную лабораторную работу я узнал о нескольких паттернах. Зачем нужно использовать паттерны? Во-первых, знание паттернов позволяет быстро определить вектор решения задачи. Очень важно понимать условия, в которых паттерн можно применить. Иначе, вместо помощи в решении задачи, вы получите слишком усложненную систему. Во-вторых, зная название паттерна, его зону применимости, само действие, можно достаточно емко объяснить коллеге проблему или, например, ввести в курс дела. В-третьих, знание паттернов позволяет проще читать чужой код. Не стоит из них делать что-то непонятное и сложное, каждый хотя бы раз использовал, сам того не понимая, определенный паттерн. Паттерн - это ничто иное как абстрактное отработанное решение задачи, с которой сталкиваются много людей.

6. Список литературы

1. Страуструп, Бьёрн. Язык программирования C++. Краткий курс, 2-е изд. : Пер. с англ. - СПб.: ООО "Диалектика", 2019. - 320 с.: ил. - Парал. тит. англ.;
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного проектирования. - СПб.: Питер, 2021. - 448 с.: ил. - (Серия “Библиотека программиста”).