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: Мариничев Иван Александрович
Группа: 80-208
Преподаватель: Чернышов Л.Н.
Дата: 21.12.20
Оценка:

Москва, 2020

1. Постановка задачи. Вариант 13

   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;

   Фигуры:

   - Ромб;
   - Пятиугольник;
   - Шестиугольник;

2. Описание программы

Таблица 1 – Функции и классы

Название	Описание

Класс **figure**	Класс figure - это абстрактный базовый класс для остальных фигур. Класс содержит в себе чисто виртуальные функции Area() для вычисления площади, print() для печати фигуры, WritToFile() для записи в файл. Единственный атрибут - координаты центра фигуры.
Классы **rhombus, pentagon, hexagon**	Классы rhombus, pentagon и hexagon - это классы-наследники от figure, в которых описаны ромб, пятиугольник и шестиугольник соответственно. В этих классах переопределены все виртуальные функции из базового класса, а также переопределен оператор вывода. Класс rhombus дополнительно содержит два атрибута - длины диагоналей. Остальные классы содержат атрибут radius - радиус описанной окружности.
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 FigureType. Класс возвращает умный указатель на созданную фигуру.
Функция **main**	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 сохранён 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 (для ромба - длины диагоналей, для пятиугольника и шестиугольника - длину радиуса описанной окружности). Есть возможность удаления фигуры по индексу. Если индекс, введённый пользователем, некорректный, то программа выдаст соответствующее сообщение. Пользователь может распечатать все содержимое документа: фигуры, их площади и координаты их центров. 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.

3. Результаты выполнения тестов

Таблица 2 – Тесты и результаты работы с ними

Название тестового файла	Входные данные	Результат
test_01.txt	2 3 test 4 5 6 7 0	1. Create new document 2. Save document 3. Open document 4. Add figure 5. Remove figure 6. Print figures 7. Undo 8. Help 0. Exit Select option: Open document first Select option: Enter file's name: No such file Select option: Open document first Select option: Open document first Select option: Open document first Select option: Open document first Select option:
test_02.txt	1 test 2 3 test 4 1 0 0 10 100 4 2 0 0 10 4 3 0 0 20 6 2 7 6 3 test 6 0	1. Create new document 2. Save document 3. Open document 4. Add figure 5. Remove figure 6. Print figures 7. Undo 8. Help 0. Exit Select option: Enter document's name: Created new document

		Select option: Successfully saved Select option: Enter file's name: File was successfully loaded Select option: Enter figure id (1 - rhombus, 2 - pentagon, 3 - hexagon) Enter coords of the center and lengths of diagonals Figure was successfully added Select option: Enter figure id (1 - rhombus, 2 - pentagon, 3 - hexagon) Enter coords of the center and length of radius Figure was successfully added Select option: Enter figure id (1 - rhombus, 2 - pentagon, 3 - hexagon) Enter coords of the center and length of side Figure was successfully added Select option: Rhombus: 1. Coordinates: (5; 0), (0; 50), (-5; 0), (0; -50) 2. Area of figure: 500 3. Center: (0; 0) Pentagon: 1. Coordinates: (9.5; 3.1), (6.1e-16; 10), (-9.5; 3.1), (-5.9; -8.1), (5.9; -8.1) 2. Area of figure: 2.4e+02 3. Center: (0; 0) Hexagon: 1. Coordinates: (20; 0), (10; 17), (-10; 17), (-20; 2.4e-15), (-10; -17), (10; -17) 2. Area of figure: 1e+03 3. Center: (0; 0) Select option: Successfully saved Select option: Done Select option: Rhombus: 1. Coordinates: (5; 0), (0; 50), (-5; 0), (0; -50) 2. Area of figure: 5e+02 3. Center: (0; 0) Pentagon: 1. Coordinates: (9.5; 3.1), (6.1e-16; 10), (-9.5; 3.1), (-5.9; -8.1), (5.9; -8.1) 2. Area of figure: 2.4e+02 3. Center: (0; 0) Select option: Enter file's name: File was successfully loaded

		Select option: Rhombus: 1. Coordinates: (5; 0), (0; 50), (-5; 0), (0; -50) 2. Area of figure: 5e+02 3. Center: (0; 0) Pentagon: 1. Coordinates: (9.5; 3.1), (6.1e-16; 10), (-9.5; 3.1), (-5.9; -8.1), (5.9; -8.1) 2. Area of figure: 2.4e+02 3. Center: (0; 0) Hexagon: 1. Coordinates: (20; 0), (10; 17), (-10; 17), (-20; 2.4e-15), (-10; -17), (10; -17) 2. Area of figure: 1e+03 3. Center: (0; 0) Select option:

4. Листинг программы

main.cpp

```
/*
 *  Мариничев И. А.
 *  М8О-208Б-19
 *  github.com/IvaMarin/oop_exercise_07
 *  Вариант 13:
 *  Фигура:
 *  - ромб
 *  - пятиугольник
 *  - шестиугольник
 */


#include <iostream>
#include <memory>
#include <vector>

#include "document.hpp"

void 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 << "8. Help" << std::endl;
    std::cout << "0. Exit" << std::endl;
    std::cout << std::endl;
}

int main() {
    Menu();
```

```
    std::shared_ptr<document> doc;
    int cmd;

    std::cout << "Select option: ";
    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 -
pentagon, 3 - hexagon)" << 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 center and length
of radius" << std::endl;
                } else if (type == 3) {
                    std::cout << "Enter coords of the center and length
of side" << std::endl;
```

```
                }
                std::shared_ptr<Figure>             fig             =
factory::Create((FigureType) 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 == 8) {
            Menu();
        } else if (cmd == 0) {
            break;
        } else {
            std::cout << "Wrong cmd" << std::endl;
        }
        std::cout << std::endl;
        std::cout << "Select option: "; // Repeat input
    }
}
```

document.hpp

```
#ifndef DOCUMENT_HPP
#define DOCUMENT_HPP

#include <stack>
#include <fstream>

#include "factory.hpp"
```

```
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 << "2. Area of figure: " << f->Area() <<
std::endl;
            auto center = f->GetCenter();
            std::cout << "3. Center: (" << center.first << "; " <<
center.second << ")" << 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->WriteToFile(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::ReadFromFile((FigureType) type,
in));
        }
    }

};

#endif //DOCUMENT_HPP
```

factory.hpp

```
#ifndef FACTORY_HPP
#define FACTORY_HPP

#include "rhombus.hpp"
#include "pentagon.hpp"
#include "hexagon.hpp"

enum class FigureType {
    RHOMBUS = 1,
    PENTAGON = 2,
    HEXAGON = 3
};

struct factory {
    static    std::shared_ptr<Figure>    Create(FigureType    t)    {
//Initializing figures from input
        switch (t) {
            case FigureType::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 FigureType::PENTAGON: {
                std::pair<double, double> center;
                double r;
                std::cin >> center.first >> center.second >> r;
                return std::make_shared<Pentagon>(center, r);
            }
            case FigureType::HEXAGON: {
                std::pair<double, double> center;
                double r;
                std::cin >> center.first >> center.second >> r;
                return std::make_shared<Hexagon>(center, r);
            }
            default:
                throw std::logic_error("Wrong figure id");
        }
    }

    static    std::shared_ptr<Figure>    ReadFromFile(FigureType    t,
std::ifstream &in) { //Initializing figures from file
        switch (t) {
            case FigureType::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 FigureType::PENTAGON: {
                std::pair<double, double> center;
                double r;
                in.read((char *) &center.first, sizeof(double));
                in.read((char *) &center.second, sizeof(double));
                in.read((char *) &r, sizeof(double));
                return std::make_shared<Pentagon>(center, r);
            }
            case FigureType::HEXAGON: {
                std::pair<double, double> center;
                double r;
                in.read((char *) &center.first, sizeof(double));
                in.read((char *) &center.second, sizeof(double));
                in.read((char *) &r, sizeof(double));
                return std::make_shared<Hexagon>(center, r);
            }
            default:
                throw std::logic_error("Wrong figure id");
        }
    }
};

#endif //FACTORY_HPP
```

figure.hpp

```
#ifndef FIGURE_HPP
#define FIGURE_HPP

#include <cmath>
```

```
class Figure {
public:
    Figure() = default;

    Figure(std::pair<double, double> &center_) : center(center_) {}

    virtual double Area() = 0;

    virtual void Print() = 0;

    virtual void WriteToFile(std::ofstream &out) = 0;

    std::pair<double, double> GetCenter() {
        return center;
    }

protected:
    std::pair<double, double> center;
};

#endif // FIGURE_HPP
```

rhombus.hpp

```
#ifndef RHOMBUS_HPP
#define RHOMBUS_HPP

#include "figure.hpp"

class Rhombus : public Figure {
public:
    Rhombus() = default;

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

    double Area() override {
        return diag1 * diag2 * 0.5;
    }

    void Print() override {
        std::cout << *this;
    }

    void WriteToFile(std::ofstream &out) override {
        int id = 1;
        out.write((char *) &id, sizeof(int));
        out.write((char *) &center.first, sizeof(double));
        out.write((char *) &center.second, sizeof(double));
        out.write((char *) &diag1, sizeof(double));
        out.write((char *) &diag2, sizeof(double));
    }

    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 << "\nRhombus:\n";
```

```
    out << "1. Coordinates: (";
    out << 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
<< ")";
    return out;
}


#endif //RHOMBUS_HPP
```

pentagon.hpp

```
#ifndef PENTAGON_HPP
#define PENTAGON_HPP

#include "figure.hpp"

class Pentagon : public Figure {
public:
    Pentagon() = default;

    Pentagon(std::pair<double, double> &center, double rad) :
Figure(center), radius(rad) {}

    double Area() override {
        double pi = acos(-1);
        double side = radius * cos(13 * pi / 10) - radius * cos(17 *
pi / 10);
        return sqrt(25 + 10 * sqrt(5)) * pow(side, 2) * 0.25;
    }

    void Print() override {
        std::cout << *this;
    }

    void WriteToFile(std::ofstream &out) override {
        int id = 2;
        out.write((char *) &id, sizeof(int));
        out.write((char *) &center.first, sizeof(double));
        out.write((char *) &center.second, sizeof(double));
        out.write((char *) &radius, sizeof(double));
    }

    friend std::ostream &operator<<(std::ostream &out, Pentagon &p);

private:
    double radius = 0;
};

std::ostream &operator<<(std::ostream &out, Pentagon &p) {
    std::cout << "\nPentagon:" << std::endl;
    std::cout << "1. Coordinates: ";
    double pi = acos(-1);
    for (int i = 0; i < 5; ++i) {
        double angle = 2 * pi * i / 5;
        std::cout.precision(2);
        std::cout << "(" << p.center.first + p.radius * cos(angle + pi
```

```
/ 10) << "; "
                  << p.center.second + p.radius * sin(angle + pi / 10)
<< ")";
        if (i != 4) {
            std::cout << ", ";
        }
    }
    return out;
}


#endif //PENTAGON_HPP
```

hexagon.hpp

```
#ifndef HEXAGON_HPP
#define HEXAGON_HPP

#include "figure.hpp"

class Hexagon : public Figure {
public:
    Hexagon() = default;

    Hexagon(std::pair<double,   double>   &center,   double   rad)   :
Figure(center), radius(rad) {}

    double Area() override {
        return pow(radius, 2) * 3 * sqrt(3) * 0.5;
    }

    void Print() override {
        std::cout << *this;
    }

    void WriteToFile(std::ofstream &out) override {
        int id = 3;
        out.write((char *) &id, sizeof(int));
        out.write((char *) &center.first, sizeof(double));
        out.write((char *) &center.second, sizeof(double));
        out.write((char *) &radius, sizeof(double));
    }

    friend std::ostream &operator<<(std::ostream &out, Hexagon &h);

private:
    double radius = 0;
};

std::ostream &operator<<(std::ostream &out, Hexagon &h) {
    std::cout << "\nHexagon:" << std::endl;
    std::cout << "1. Coordinates: ";
    double pi = acos(-1);
    for (int i = 0; i < 6; ++i) {
        double angle = pi * i / 3;
        std::cout.precision(2);
        std::cout << "(" << h.center.first + h.radius * cos(angle) <<
"; "
                  << h.center.second + h.radius * sin(angle) << ")";
        if (i != 5) {
            std::cout << ", ";
        }
```

```
    }
    return out;
}


#endif //HEXAGON_HPP
```

CmakeLists.txt

```
cmake_minimum_required(VERSION 3.17)
project(oop_lab7)

set(CMAKE_CXX_STANDARD 17)

add_executable(oop_lab7 main.cpp)
```

5. Выводы

В ходе лабораторной работы я получил практические навыки в хороших практиках проектирования структуры классов приложения.

Список литературы

1. Стефан К. Дьюхэрст Скользкие места C++. Как избежать проблем при проектировании и компиляции ваших программ — ISBN: 5-94074-083-9 — 266 с.
2. Руководство по языку C++ [Электронный ресурс]. URL: https://www.cplusplus.com/ (дата обращения 17.12.2020).
3. Шаблон memento [Электронный ресурс]. URL: http://cpp-reference.ru/patterns/behavioral-patterns/memento/ (дата обращения 18.12.2020).
4. Шаблон factory [Электронный ресурс]. URL: http://cpp-reference.ru/patterns/creational-patterns/factory-method/ (дата обращения 18.12.2020).