Московский авиационный институт

(Национальный исследовательский университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа № 7

Тема: Проектирование структуры классов

Студент: Семенов Илья

Преподаватель: Журавлев А.А.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

Спроектировать простейший графический векторный редактор. Требования 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
- должна быть реализована операция undo, отменяющая последнее сделанное действие. Должно действовать на операции добавления/удаления фигур.

Дополнительно реализовать ввода произвольного многоугольника и произвольной ломаной кривой(описание завершается нажатием правой кнопки мыши), окружности, строящейся по двум точкам. Добавить возможность выбора цвета для каждой фигуры. Добавить возможность удаления фигуры по клику правой кнопкой мыши внутри нее.

Вариант задания 20:

- Трапеция, Ромб, Пятиугольник

2. **Репозиторий github**

https://github.com/ilya89099/oop_exercise_07/

3. **Описание программы**

Реализованы классы для всех необходимых фигур, с применением наследования, для уменьшения дублирования кода(для всех многоугольников был создан абстрактный класс, содержащий в себе логику отрисовки). Для работы со всеми этими фигурами разработан класс Document, содержащий в себе функции для добавления и удаления фигур из документа, абстрактный класс для команд удаления и добавления(для реализации функции отмены команды). Так же этот класс отвечает за сохранение в файл и загрузку из файла. Класс Redactor предоставляет интерфейс для работы с документами. В основной функции описан графический интерфейс, взаимодействующий с классом Redactor.

4. Набор testcases

В связи с тем, что используется графический интерфейс, тесты можно произвести, взаимодействуя с ним. Мною было проведено всестороннее тестирования с его использованием, и оно не дало повода усомниться в правильности работы программы.

5. Листинг программы

main.cpp

```
#include <array>
#include <fstream>
#include <iostream>
#include <memory>
#include <vector>
#include "FiguresAndFactories/Factory.h"
#include "FiguresAndFactories/Circle/CircleFactory.h"
#include "FiguresAndFactories/Polygon/Trapeze/TrapezeFactory.h"
#include "FiguresAndFactories/Polygon/Rhombus/RhombusFactory.h"
#include "FiguresAndFactories/Polygon/RandomPolygon/RandomPolygonFactory.h"
#include "FiguresAndFactories/Polygon/Line/LineFactory.h"
#include "Redactor.h"
#include "sdl.h"
#include "imgui.h"
#define FILENAME_LENGTH 128
int main() {
    Redactor redactor;
    sdl::renderer renderer("Editor");
    bool quit = false;
    bool building_polygon = false;
    std::unique_ptr<Factory> active_builder = nullptr;
    char file_name[FILENAME_LENGTH] = "";
    std::string color_result;
    int32_t remove_id = 0;
    int red = 255, green = 0, blue = 0;
    bool remove_figure = false;
    while (!quit) {
        renderer.set_color(0, 0, 0);
        renderer.clear();
        sdl::event event;
        while (sdl::event::poll(event)) {
            sdl::quit_event quit_event;
            sdl::mouse_button_event mouse_button_event;
            if (event.extract(quit_event)) {
                quit = true;
                break;
            } else if (event.extract(mouse_button_event)) {
                if (active_builder && mouse_button_event.button() ==
```

```
sdl::mouse_button_event::left &&
                    mouse_button_event.type() ==
sdl::mouse_button_event::down) {
                    active_builder->AddPoint(Point{(double)
mouse_button_event.x(), (double) mouse_button_event.y()});
                    if
(dynamic_cast<RandomPolygonFactory*>(active_builder.get()) == nullptr
                        &&
dynamic_cast<LineFactory*>(active_builder.get()) == nullptr) {
                        std::shared_ptr<Figure> new_figure =
active_builder->TryCreateObject();
                        if (new_figure) {
                            redactor.InsertFigure(new_figure);
                            active_builder = nullptr;
                        }
                    }
                }
                if (mouse_button_event.button() ==
sdl::mouse_button_event::right && mouse_button_event.type() ==
sdl::mouse_button_event::down) {
                    if (active_builder &&
(dynamic_cast<RandomPolygonFactory*>(active_builder.get()) != nullptr
                        || dynamic_cast<LineFactory*>(active_builder.get())
!= nullptr)) {
                        std::shared_ptr<Figure> new_figure =
active_builder->TryCreateObject();
                        if (new_figure) {
                            redactor.InsertFigure(new_figure);
                            active_builder = nullptr;
                        }
                    } else if (!active_builder) {

redactor.RemoveFigureOnPoint(Point{(double)mouse_button_event.x(),
(double) mouse_button_event.y()});
                    }
                }
            }
        }
        redactor.Render(renderer);
        ImGui::Begin("Menu");
        ImGui::InputText("File name", file_name, FILENAME_LENGTH - 1);
        if (ImGui::Button("Create file")) {
            std::string file_name_str(file_name);
            if (!file_name_str.empty()) {
                redactor.CreateDocument(file_name_str);
            }
        }
        if (ImGui::Button("Save file")) {
            if (redactor.HasDocument()) {
                redactor.SaveDocument();
            }
        }
```

```
        if (ImGui::Button("Load file")) {
            std::string filename(file_name);
            if (!filename.empty()) {
                redactor.LoadDocument(filename);
            }
        }
        if (ImGui::Button("Trapeze")) {
            active_builder = std::make_unique<TrapezeFactory>();
        }
        ImGui::SameLine();
        if (ImGui::Button("Circle")) {
            active_builder = std::make_unique<CircleFactory>();
        }
        if (ImGui::Button("Random Polygon")) {
            active_builder = std::make_unique<RandomPolygonFactory>();
        }
        ImGui::SameLine();
        if (ImGui::Button("Rhombus")) {
            active_builder = std::make_unique<RhombusFactory>();
        }
        if (ImGui::Button("Line")) {
            active_builder = std::make_unique<LineFactory>();
        }
        ImGui::InputInt("Remove id", &remove_id);
        if (ImGui::Button("Remove")) {
            redactor.RemoveFigureOnIndex(remove_id);
        }
        ImGui::LabelText("","Enter color %s", color_result.c_str());
        ImGui::InputInt("Red", &red);
        ImGui::InputInt("Green", &green);
        ImGui::InputInt("Blue", &blue);
        if (ImGui::Button("Set color")) {
            if (red < 0 || red > 255 || green < 0 || green > 255 || blue <
0 || blue > 255) {
                color_result = "Error, invalid color";
                red = 255;
                green = 0;
                blue = 0;
            } else {
                redactor.SetColor(red,green,blue);
                color_result = "";
            }
        }
        if (ImGui::Button("Undo")) {
            redactor.Undo();
        }
        ImGui::End();
        renderer.present();
    }
}
```

Document.h

```
#pragma once
#include <string>
#include <vector>
#include <fstream>
#include <memory>
#include "FiguresAndFactories/Figure.h"
#include "FiguresAndFactories/Circle/Circle.h"
#include "FiguresAndFactories/Polygon/Trapeze/Trapeze.h"
#include "FiguresAndFactories/Polygon/RandomPolygon/RandomPolygon.h"
#include "FiguresAndFactories/Polygon/Rhombus/Rhombus.h"
class Document;
class Command;
class Command {
    friend Document;
public:
    Command(Document& doc);
    virtual ~Command() = default;
    virtual void Undo() = 0;
protected:
    Document& associated_document_;
};
class AddCommand : public Command {
public:
    AddCommand(Document& doc);
    void Undo() override;
};
class DeleteCommand : public Command {
public:
    DeleteCommand(Document& doc, std::shared_ptr<Figure> fig, size_t
index);
    void Undo() override;
private:
    size_t index_;
    std::shared_ptr<Figure> contained_figure_;
};
class Document {
    friend Command;
    friend AddCommand;
    friend DeleteCommand;
public:
    Document(const std::string& str);
    void SaveToFile() const;
    void LoadFromFile();
    void Render(const sdl::renderer& r);
    void AddFigure(std::shared_ptr<Figure>);
    void DeleteFigure(size_t index);
    void DeleteFigure(Point p);
    void SetColor(int r, int g, int b);
    void Undo();
private:
    std::string document_name_;
```

```
    std::vector<std::shared_ptr<Figure>> figures_;
    std::vector<std::shared_ptr<Command>> commands_;
    int rcolor = 255, gcolor = 0, bcolor = 0;
};
```

Document.cpp

```
#include "Document.h"
Command::Command(Document& doc)
: associated_document_(doc) {}
Document::Document(const std::string &str)
: document_name_(str){}
AddCommand::AddCommand(Document& doc)
: Command(doc) {}
void AddCommand::Undo() {
    associated_document_.figures_.pop_back();
}
DeleteCommand::DeleteCommand(Document& doc, std::shared_ptr<Figure> fig,
size_t index)
: Command(doc), contained_figure_(move(fig)), index_(index) {}
void DeleteCommand::Undo() {
    auto it = associated_document_.figures_.begin();
    associated_document_.figures_.insert(it + index_, contained_figure_);
}
void Document::AddFigure(std::shared_ptr<Figure> fig) {
    figures_.push_back(move(fig));
    figures_.back()->SetColor(rcolor, gcolor, bcolor);
    commands_.push_back(std::make_shared<AddCommand>(*this));
}
void Document::DeleteFigure(Point p) {
    for (size_t i = 0; i < figures_.size(); ++i) {
        size_t cur_index = figures_.size() - i - 1;
        if (figures_[cur_index]->PointBelongsTo(p)) {
            DeleteFigure(cur_index);
            break;
        }
    }
}
void Document::DeleteFigure(size_t index) {
    if (index >= figures_.size()) {
        return;
    }
    auto ptr = std::make_shared<DeleteCommand>(*this, figures_[index],
index);
    figures_.erase(figures_.begin() + index);
    commands_.push_back(ptr);
}
void Document::Render(const sdl::renderer& r) {
    for (auto i : figures_) {
        i->Render(r);
    }
}
void Document::Undo() {
```

```
    if (commands_.empty()) {
        return;
    }
    std::shared_ptr<Command> ptr = commands_.back();
    commands_.pop_back();
    ptr->Undo();
}
void Document::SetColor(int r, int g, int b) {
    rcolor = r;
    gcolor = g;
    bcolor = b;
}
void Document::SaveToFile() const {
    std::ofstream os(document_name_, std::ios::out | std::ios::trunc);
    if (!os.is_open()) {
        return;
    }
    os << figures_.size() << "\n";
    for (auto ptr : figures_) {
        ptr->Save(os);
    }
    os.close();
}
void Document::LoadFromFile() {
    std::ifstream is(document_name_, std::ios::in);
    if (!is) {
        return;
    }
    figures_.clear();
    commands_.clear();
    size_t figures_count;
    is >> figures_count;
    for (size_t i = 0; i < figures_count; ++i) {
        std::string type;
        is >> type;
        std::shared_ptr<Figure> ptr;
        if (type == "circle") {
            ptr = std::make_shared<Circle>();
        } else if (type == "random_polygon") {
            ptr = std::make_shared<RandomPolygon>();
        } else if (type == "trapeze") {
            ptr = std::make_shared<Trapeze>();
        } else  if (type == "rhombus") {
            ptr = std::make_shared<Rhombus>();
        }
        ptr->Load(is);
        figures_.push_back(ptr);
    }
    is.close();
}
```

Redactor.h

```h
#pragma once
#include <string>
#include <memory>
#include "FiguresAndFactories/Point.h"
#include "Document.h"
class Redactor {
public:
    void CreateDocument(const std::string& document_name);
    void LoadDocument(const std::string& filename);
    void SaveDocument();
    bool HasDocument() const;
    void Render(const sdl::renderer& r);
    void InsertFigure(std::shared_ptr<Figure> figure);
    void RemoveFigureOnPoint(Point p);
    void RemoveFigureOnIndex(size_t index);
    void SetColor(int r, int g, int b);
    void Undo();
private:
    std::shared_ptr<Document> document = nullptr;
};
```

Redactor.cpp

```cpp
#include "Redactor.h"
void Redactor::CreateDocument(const std::string& document_name) {
    document = std::make_shared<Document>(document_name);
}
void Redactor::LoadDocument(const std::string &filename) {
    document = std::make_shared<Document>(filename);
    document->LoadFromFile();
}
void Redactor::Render(const sdl::renderer& r) {
    if (HasDocument()) {
        document->Render(r);
    }
}
void Redactor::SaveDocument() {
    if (HasDocument()) {
        document->SaveToFile();
    }
}
bool Redactor::HasDocument() const {
    return document != nullptr;
}
void Redactor::InsertFigure(std::shared_ptr<Figure> figure) {
    if (HasDocument()) {
        document->AddFigure(figure);
    }
}
```

```
void Redactor::RemoveFigureOnPoint(Point p) {
    if (HasDocument()) {
        document->DeleteFigure(p);
    }
}
void Redactor::RemoveFigureOnIndex(size_t index) {
    if (HasDocument()) {
        document->DeleteFigure(index);
    }
}
void Redactor::Undo() {
    if (HasDocument()) {
        document->Undo();
    }
}
void Redactor::SetColor(int r, int g, int b) {
    if (HasDocument()) {
        document->SetColor(r,g,b);
    }
}
```

Point.cpp

```
#pragma once
#include <numeric>
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>
#include <SDL.h>
#include "../sdl.h"
#include "imgui.h"
class Vector;
struct Point;
struct Point {
    Point() = default;
    Point(const Vector& v);
    Point (double new_x, double new_y);
    double x = 0;
    double y = 0;
};
std::ostream& operator << (std::ostream& os, const Point& p);
std::istream& operator >> (std::istream& is, Point& p);
Point operator + (Point lhs, Point rhs);
Point operator - (Point lhs, Point rhs);
Point operator / (Point lhs, double a);
Point operator * (Point lhs, double a);
class Vector {
public:
    explicit Vector(double a, double b);
    explicit Vector(Point a, Point b);
    bool operator == (Vector rhs);
```

```
    Vector operator - ();
    friend double operator * (Vector lhs, Vector rhs);
    double length() const;
    double x;
    double y;
};
// точка сначала, прямые потом
Vector normalize_vector(const Vector& v);
Point projection (Point p1, Point p2, Point p3); // проекция точки p3 на
прямую p1p2
bool is_parallel(const Vector& lhs, const Vector& rhs);
bool is_perpendecular(const Vector& lhs, const Vector& rhs);
double point_and_line_distance(Point p1, Point p2, Point p3);
```

Point.cpp

```
#include "Point.h"
Point::Point(const Vector& v)
: x(v.x), y(v.y) {}
Point::Point(double new_x, double new_y)
: x(new_x), y(new_y) {}
Point operator + (Point lhs, Point rhs) {
    return {lhs.x + rhs.x, lhs.y + rhs.y};
}
Point operator - (Point lhs, Point rhs) {
    return {lhs.x - rhs.x, lhs.y - rhs.y};
}
Point operator / (Point lhs, double a) {
    return { lhs.x / a, lhs.y / a};
}
Point operator * (Point lhs, double a) {
    return {lhs.x * a, lhs.y * a};
}
bool operator < (Point lhs, Point rhs) {
    return (lhs.x * lhs.x + lhs.y * lhs.y) < (lhs.x * lhs.x + lhs.y *
lhs.y);
}
double operator * (Vector lhs, Vector rhs) {
    return lhs.x * rhs.x + lhs.y * rhs.y;
}
bool is_parallel(const Vector& lhs, const Vector& rhs) {
    return (lhs.x * rhs.y - lhs.y * rhs.x) == 0;
}
bool Vector::operator == (Vector rhs) {
    return
        std::abs(x - rhs.x) < std::numeric_limits<double>::epsilon() * 100
        && std::abs(y - rhs.y) < std::numeric_limits<double>::epsilon() *
100;
}
double Vector::length() const {
    return sqrt(x*x + y*y);
}
```

```
Vector::Vector(double a, double b)
: x(a), y(b) {
}
Vector::Vector(Point a, Point b)
: x(b.x - a.x), y(b.y - a.y){
}
Vector Vector::operator - () {
    return Vector(-x, -y);
}
bool is_perpendecular(const Vector& lhs, const Vector& rhs) {
    return (lhs * rhs) == 0;
}
Vector normalize_vector(const Vector& v) {
    return Vector(v.x / v.length(), v.y / v.length());
}
Point projection (Point p1, Point p2, Point p3) { // проекция точки p3 на
прямую p1p2
    double x = p2.y - p1.y; //x и y - координаты вектора,
перпендикулярного к AB
    double y = p1.x - p2.x;
    double L = (p1.x * p2.y - p2.x * p1.y + p1.y * p3.x - p2.y * p3.x +
p2.x * p3.y - p1.x * p3.y) / (x * (p2.y - p1.y) + y * (p1.x - p2.x));
    Point H;
    H.x = p3.x + x * L;
    H.y = p3.y + y * L;
    return H;
}
double point_and_line_distance(Point p1, Point p2, Point p3) {
    double A = p2.y - p3.y;
    double B = p3.x - p2.x;
    double C = p2.x*p3.y - p3.x*p2.y;
    return (std::abs(A*p1.x + B*p1.y + C) / std::sqrt(A*A + B*B));
}
std::ostream& operator << (std::ostream& os, const Point& p) {
    return os << p.x << " " << p.y;
}
std::istream& operator >> (std::istream& is, Point& p) {
    return is >> p.x >> p.y;
}
```

Figure.h

```
#pragma once
#include "Point.h"
class Figure {
public:
    virtual void Render(const sdl::renderer& renderer) const = 0;
    virtual void Save(std::ostream& os) const = 0;
    virtual void Load(std::istream& is) = 0;
    virtual bool PointBelongsTo(Point p) const = 0;
    void SetColor(int r, int g, int b);
    virtual ~Figure() = default;
protected:
```

```
    int rcolor = 255, gcolor = 0, bcolor = 0;
};
```

Figure.cpp

```
#include "Figure.h"
void Figure::SetColor(int r, int g, int b) {
    rcolor = r;
    gcolor = g;
    bcolor = b;
}
```

Factory.h

```
#pragma once
#include "Figure.h"
#include <vector>
#include <memory>
#include <exception>
class Factory {
public:
    Factory() = default;
    virtual ~Factory() = default;
    virtual void AddPoint(const Point& p) = 0;
    virtual std::unique_ptr<Figure> TryCreateObject() const = 0;
protected:
    std::vector<Point> points_;
};
```

Polygon.h

```
#pragma once
#include "../Figure.h"
#include <math.h>
class Polygon : public Figure {
public:
    Polygon() = default;
    Polygon(int points_count);
    Polygon(const std::vector<Point>& points);
    virtual void Render(const sdl::renderer& renderer) const override;
    bool PointBelongsTo(Point p) const override;
protected:
    std::vector<Point> points_;
};
```

Polygon.cpp

```
#include "Polygon.h"
Polygon::Polygon(int points_count)
: points_(points_count) {}
Polygon::Polygon(const std::vector<Point>& points)
: points_(points) {}
```

```
void Polygon::Render(const sdl::renderer &renderer) const {
    renderer.set_color(rcolor, gcolor, bcolor);
    for (size_t i = 0; i < points_.size(); ++i) {
        size_t next_index = (i + 1) % points_.size();
        renderer.draw_line(points_[i].x, points_[i].y,
points_[next_index].x, points_[next_index].y);
    }
}
bool Polygon::PointBelongsTo(Point p) const {
    bool is_ok = true;
    int first = 0;
    for (size_t i = 0; i < points_.size(); ++i) {
        size_t next_index = (i + 1) % points_.size();
        int elem = (points_[i].x - p.x) * (points_[next_index].y -
points_[i].y) - (points_[next_index].x - points_[i].x) * (points_[i].y -
p.y);
        if (i == 0) {
            first = elem / std::abs(elem);
        } else {
            if (first * (elem / std::abs(elem)) < 0) {
                is_ok = false;
                break;
            }
        }
    }
    return is_ok;
}
```

6. Вывод

В результате данной лабораторной работы я научился работать с библиотеками, предоставляющими графический интерфейс. Так же(после 3 рефакторингов) я получил богатый опыт проектирования структуры классов(с рациональным использованием наследования и полиморфизма). Также я познакомился с паттернами builder и abstractfactory.