Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»
Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа
Дисциплина: «Объектно-ориентированное программирование»
III семестр
Задание 7: «Проектирование структуры классов»

Группа:	М8О-208Б-18, №3
Студент:	Овечкин Виталий Андреевич
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	30.12.2019

Москва, 2019

1. **Задание** (*вариант № 19*):
Реализовать графический редактор, позволяющий отрисовывать четырёхугольник, многоугольник, круг, ломаную, задавать их цвета, реализовать функцию undo

2. **Адрес репозитория на GitHub** https://github.com/vitalouivi/oop_exercise_7
3. **Код программы на C++**

main.cpp

```
#include <array>
#include <memory>
#include <vector>
#include <stack>

#include "sdl.h"
#include "imgui.h"

#include "quadrangle.h"
#include "curve_line.h"
#include "circle.h"
#include "polygon.h"

#include "color.h"
#include "Document.h"

int main() {
	sdl::renderer renderer("Editor");
	bool quit = false;

	std::unique_ptr<builder> active_builder = nullptr;
	bool active_deleter = false;
	const int32_t file_name_length = 128;
	char file_name[file_name_length] = "";
	int32_t remove_id = 0;
	std::vector<int> color(3);

	Document currentDocument;

	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) {
					std::unique_ptr<figure> figure = active_builder-
>add_vertex(vertex{mouse_button_event.x(), mouse_button_event.y()});
					if (figure) {
						figure -> setColor(color);

						currentDocument.addFigure(std::move(figure));

						active_builder = nullptr;
					}
				}
```

```
                        if (active_builder && mouse_button_event.button() ==
sdl::mouse_button_event::right && mouse_button_event.type() ==
sdl::mouse_button_event::down) {
                                std::unique_ptr<figure> figure = active_builder-
>add_vertex(vertex{-1, -1});
                                if (figure) {
                                        figure -> setColor(color);

                                        currentDocument.addFigure(std::move(figure));

                                        active_builder = nullptr;
                                }
                        }
                        if (active_deleter && mouse_button_event.button() ==
sdl::mouse_button_event::left && mouse_button_event.type() ==
sdl::mouse_button_event::down) {


        currentDocument.removeByClick(vertex{mouse_button_event.x(),
mouse_button_event.y()});

                                        active_deleter = false;
                                }
                        }
                }

                currentDocument.render(renderer);



                ImGui::Begin("Menu");
                if (ImGui::Button("New canvas")) {
                        currentDocument.clear();
                }

                ImGui::InputText("File name", file_name, file_name_length - 1);

                if (ImGui::Button("Save")) {
                        std::ofstream os(file_name);

                        if (os) {
                                currentDocument.Save(os);
                        }

                }

                ImGui::SameLine();

                if (ImGui::Button("Load")) {
                        std::ifstream is(file_name);

                        if (is) {
                                currentDocument.Load(is);
                        }

                }


                ImGui::InputInt("R", &color[0]);
                ImGui::InputInt("G", &color[1]);
                ImGui::InputInt("B", &color[2]);
                if (ImGui::Button("Red")) {
                        color[0] = 255;
                        color[1] = 0;
```

```
                color[2] = 0;
            }
            ImGui::SameLine();
            if (ImGui::Button("Green")) {
                color[0] = 0;
                color[1] = 255;
                color[2] = 0;
            }
            ImGui::SameLine();
            if (ImGui::Button("Blue")) {
                color[0] = 0;
                color[1] = 0;
                color[2] = 255;
            }

            if (ImGui::Button("Quadrangle")) {
                active_builder = std::make_unique<quadrangle_builder>();
            }
            if (ImGui::Button("Broken Line")) {
                active_builder = std::make_unique<curve_line_builder>();
            }
            if (ImGui::Button("Circle")) {
                active_builder = std::make_unique<circle_builder>();
            }
            if (ImGui::Button("Polygon")) {
                active_builder = std::make_unique<polygon_builder>();
            }

            ImGui::InputInt("Remove id", &remove_id);
            if (ImGui::Button("Remove")) {
                if (remove_id >= 0 && remove_id < (currentDocument.figures).size()) {

                    currentDocument.removeFigure(remove_id);

                }
            }
            if (ImGui::Button("Remove by click")) {
                active_deleter = true;
            }
            if (ImGui::Button("UNDO")) {

                currentDocument.undo();
            }

            ImGui::End();

            renderer.present();
        }


}
```

Figure.h

```
#ifndef D_FIGURE_H
#define D_FIGURE_H 1

#include "vertex.h"

#include "sdl.h"
#include <array>
#include <vector>
#include <memory>
//#include <iostream>
//#include <fstream>
#include <string>
```

```
struct figure {
	virtual void render(const sdl::renderer& renderer) const = 0;
	virtual void save(std::ostream& os) const = 0;
	virtual bool isPointInside(vertex v) const = 0;
	virtual void setColor(std::vector<int> color) = 0;
	virtual ~figure() = default;

};

#endif // !D_FIGURE_H
```

vertex.h

```
#ifndef D_VERTEX_H
#define D_VERTEX_H 1

#include <memory>
#include <fstream>
#include <iostream>

struct vertex {
	int32_t x, y;

};

inline std::istream& operator>> (std::istream& is, vertex& p) {
	is >> p.x >> p.y;
	return is;
}

#endif // !D_VERTEX_H
```

circle.h

```
#ifndef D_CIRCLE_H
#define D_CIRCLE_H 1

#include "figure.h"
#include "builder.h"
#include <math.h>

struct circle : figure {
	circle(const std::vector<vertex>& vertices);

	void setColor(std::vector<int> color) override;

	void render(const sdl::renderer& renderer) const override;

	void save(std::ostream& os) const override;

	bool isPointInside(vertex v) const override;

private:
	std::vector<int> color_;
	std::vector<vertex> vertices_;
	int radius;
};

struct circle_builder : builder {

	std::unique_ptr<figure> add_vertex(const vertex& v) override;

	std::string getType() override;

private:

	int32_t n_ = 0;
```

```
	std::vector<vertex> vertices_;

};


#endif
```

document.h

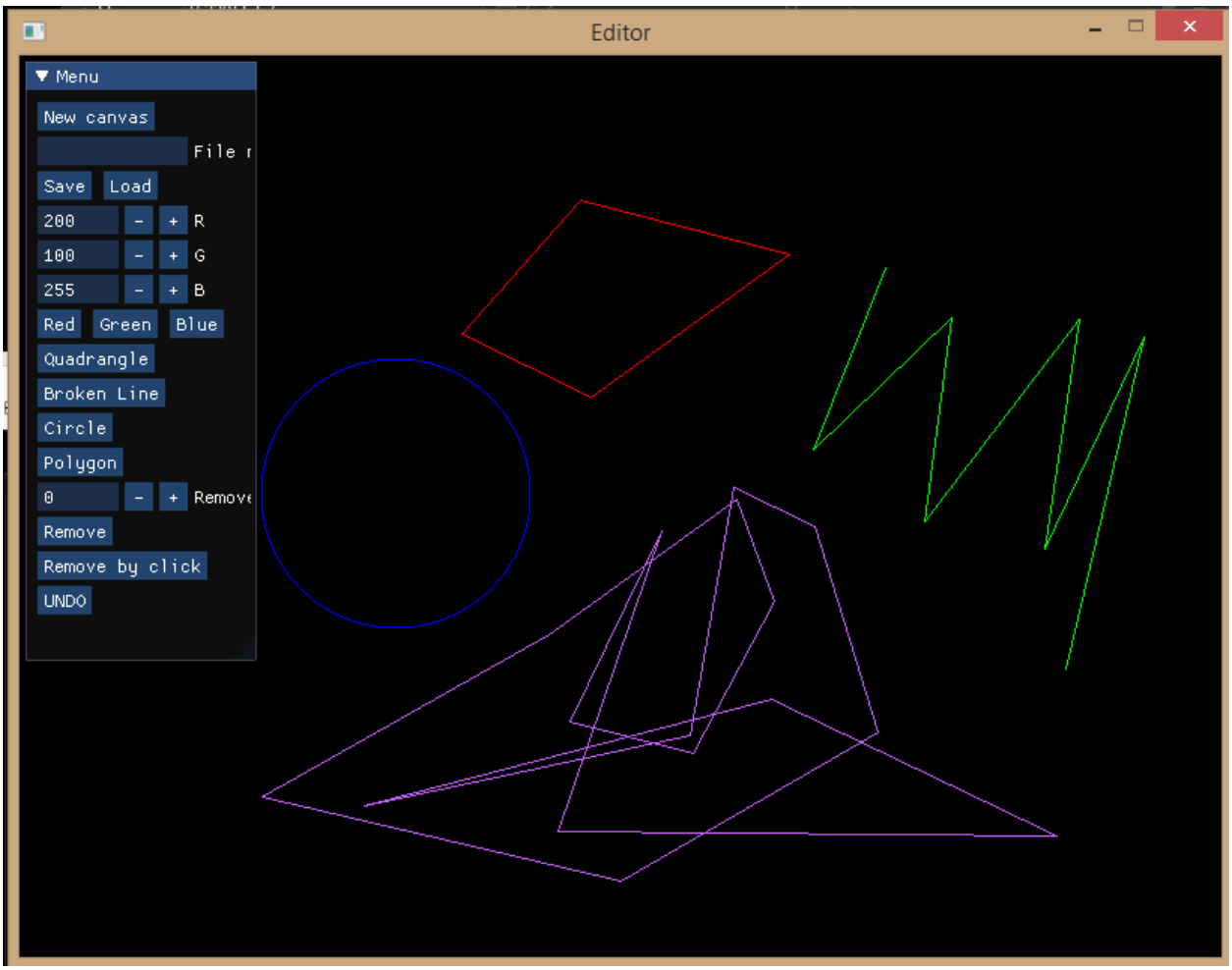
```
#pragma once

#ifndef D_DOCUMENT_H
#define D_DOCUMENT_H 1

#include <array>
#include <fstream>
#include <iostream>
#include <memory>
#include <vector>
#include <stack>

#include "sdl.h"
#include "imgui.h"

#include"quadrangle.h"
#include "curve_line.h"
#include "circle.h"
#include "polygon.h"

struct Command;
struct CommandAdd;
struct CommandRemove;
struct Document;


struct Document {
public:
	Document() = default;

	void addFigure(std::unique_ptr<figure> fig);
	void removeFigure(int id);
	void removeByClick(vertex v);
	void undo();
	void Save(std::ofstream& os);
	void Load(std::ifstream& is);
	void render(const sdl::renderer& renderer);
	void clear();

	std::vector<std::shared_ptr<figure>> figures;
	std::stack<std::unique_ptr<Command>> commandStack;
};



struct Command {

	virtual ~Command() = default;

	virtual void undo() = 0;

};

struct CommandAdd : Command {


	int index__;
```

```
        Document * doc__ = new Document();

        CommandAdd(int index, Document * doc) : index__(index), doc__(doc) {}

        void undo() {
                (doc__ -> figures).erase((doc__ -> figures).begin() + index__);
        }

};

struct CommandRemove : Command {

        Document * doc__;

        int index__;
        std::shared_ptr<figure> figure__ = nullptr;

        CommandRemove(int index, std::shared_ptr<figure> figure_, Document * doc) :
index__(index), figure__(figure_), doc__(doc) {}

        void undo() {
                if (index__ > (doc__ -> figures).size() - 1)
                        (doc__ -> figures).push_back(std::move(figure__));
                else
                        (doc__ -> figures).insert((doc__ -> figures).begin() + index__,
std::move(figure__));
        }

};

#endif
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0)

project(lab7)

set(CMAKE_CXX_STANDARD_REQUIRED YES)
set(CMAKE_CXX_STANDARD 14)

add_executable(lab7
  main.cpp
  sdl.cpp
  circle.cpp
  curve_line.cpp
  polygon.cpp
  quadrangle.cpp
  Document.cpp
)

add_subdirectory(lib/SDL2/)
target_link_libraries(lab7 SDL2-static)
target_include_directories(lab7 PRIVATE ${SDL2_INCLUDE_DIR})

add_subdirectory(lib/imgui/)
target_include_directories(imgui PRIVATE lib/SDL2/include/)
target_link_libraries(lab7 imgui)
```

Пример Использования



4. Объяснение результатов работы программы - вывод

В figure.h задаётся базовый класс Figure задающий структуру для классов — наследников — Quadrangle, Polygon и Circle.
Наследование позволяет избежать дублирования кода при написании классов, т. к. класс может использовать переменные и методы другого класса как свои собственные.
Программа представляет собой визуальное приложение, способное строить прямоугольник, ромб, трапецию, ломаную линию, многоугольник и круг. Возможно удаление по индексу, клику внутрь фигуры , операция undo, загрузка и сохранение в файл. Я познакомился с визуальными библиотеками в C++, углубил свои знания в области полиморфизма, познакомился с написанием и сборкой многофайльвых проектов, а так же подключением сторонних библиотек. Также в очередной раз я практиковался в разбиении кода на модули.