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»

Москва, 2019

1. **Задание** (*вариант № 7* ):

Реализовать графический редактор, позволяющий рисовывать треугольник, восьмиугольник и шестиугольник.

2. **Адрес репозитория на GitHub**

https://github.com/DragonKeker/oop_exercise_07

3. **Код программы на C++**

***main.cpp***

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

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

#include "triangle.h"
#include "hexagon.h"
#include "octagon.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;
                                        }
                                }

                        }
                }

                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);
                        }

                }


                        color[0] = 255;
                        color[1] = 0;
                        color[2] = 0;


                if (ImGui::Button("Triangle")) {
                        active_builder = std::make_unique<triangle_builder>();
                }
                if (ImGui::Button("Hexagon")) {
                        active_builder = std::make_unique<hexagon_builder>();
                }
                if (ImGui::Button("Octagon")) {
                        active_builder = std::make_unique<octagon_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("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 <string>


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

};

#endif
```

***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
```

***triangle.h***

```
#ifndef D_TRIANGLE_H
#define D_TRIANGLE_H 1

#include "figure.h"
```

```
#include "builder.h"

struct triangle : figure {
    triangle(const std::array<vertex, 3>& vertices);
    void setColor(std::vector<int> color);

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

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


private:
    std::array<vertex, 3> vertices_;
    std::vector<int> color_;

};

struct triangle_builder : builder {
    std::unique_ptr<figure> add_vertex(const vertex& v);
    std::string getType();

private:
    int32_t n_ = 0;
    std::array<vertex, 3> vertices_;

};

#endif
```

***Hexagon.h***

```
#ifndef D_HEXAGON_H
#define D_HEXAGON_H 1

#include "figure.h"
#include "builder.h"

struct hexagon : figure {
    hexagon(const std::array<vertex, 6>& vertices);
    void setColor(std::vector<int> color);

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

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


private:
    std::array<vertex, 6> vertices_;
    std::vector<int> color_;

};

struct hexagon_builder : builder {
    std::unique_ptr<figure> add_vertex(const vertex& v);
    std::string getType();

private:
    int32_t n_ = 0;
    std::array<vertex, 6> vertices_;

};

#endif
```

***octagon.h***

```
#ifndef D_OCTAGON_H
#define D_OCTAGON_H 1
```

```
#include "figure.h"
#include "builder.h"

struct octagon : figure {
    octagon(const std::array<vertex, 8>& vertices);
    void setColor(std::vector<int> color);

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

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

private:
    std::array<vertex, 8> vertices_;
    std::vector<int> color_;

};

struct octagon_builder : builder {
    std::unique_ptr<figure> add_vertex(const vertex& v);
    std::string getType();

private:
    int32_t n_ = 0;
    std::array<vertex, 8> vertices_;

};

#endif
```

***Document.h***

```
#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 "triangle.h"
#include "hexagon.h"
#include "octagon.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 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
```

```
  Document.cpp
  triangle.cpp
  octagon.cpp
  hexagon.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)
```

4. **Объяснение результатов работы программы - вывод**

Я познакомился с написанием и сборкой многофайльвых проектов, а так же подключением сторонних библиотек, углубил свои знания в использовании полиморфизма.