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, №12
Студент:	Коростелев Дмитрий Васильевич
Преподаватель:	Журавлёв Андрей Андреевич
Оценка:	
Дата:	09.12.2019

Москва, 2019

1. Задание

Разработать программу на языке C++ согласно варианту задания. Программа на C++ должна собираться с помощью системы сборки CMake. Программа должна получать данные из стандартного ввода и выводить данные в стандартный вывод.

Необходимо настроить сборку лабораторной работы с помощью CMake. Собранная программа должна называться oop_exercise_07 (в случае использования Windows oop_exercise_07.exe)

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)

• 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;

2. Адрес репозитория на GitHub

https://github.com/Dmitry4K/oop_exercise_07

3. Код программы на C++

Figure.h

```
#pragma once
#include<cmath>
struct figure {
    virtual void render(const sdl::renderer& renderer) const = 0;
    virtual void save(std::ostream& os) const = 0;
    virtual ~figure() = default;
};
```

```
struct vertex {
    int32_t x, y;
};

double distance(int x1, int y1, int x2, int y2) {
    return sqrt(pow((x1 - x2),2)+ pow((y1 - y2), 2));
}


double distance(vertex a, vertex b) {
    return sqrt(pow(a.x - b.x, 2) + pow(a.y - b.y, 2));
}
```

Figures.h

```
#pragma once
#include"figure.h"
#include"pentagon.h"
#include"rhombus.h"
#include"trapeze.h"
```

Main.cpp

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

#include "sdl.h"
#include "imgui.h"
#include "figures.h"
#include "tools.h"


int main() {
  sdl::renderer renderer("Editor");//создание рендера с именем Editor
  bool quit = false;//вектор с уникальными указателями на фигуры
  std::unique_ptr<builder> active_builder = nullptr;// создаем уникальный указатель на
строителя
  const int32_t file_name_length = 128;//максимальная длина названия файла
  char file_name[file_name_length] = "";//имя файла
  int32_t remove_id = 0;
  while(!quit){
    //создаем пустое черное окно
    renderer.set_color(0, 0, 0);
    renderer.clear();

    sdl::event event;

    while(sdl::event::poll(event)){
      sdl::quit_event quit_event;//событие выхода
      sdl::mouse_button_event mouse_button_event;//события щелчка мыши
      if(event.extract(quit_event)){//если событие
        quit = true;
        break;
      }else if(event.extract(mouse_button_event)){
        if(active_builder && mouse_button_event.button() == sdl::mouse_button_event::left
&& //если есть строитель и леый щелчок мыши
           mouse_button_event.type() == sdl::mouse_button_event::down){
          std::unique_ptr<figure> figure =//nullptr, если в строителе не 3 веришны, иначе
фигура
           active_builder->add_vertex(vertex{mouse_button_event.x(),
mouse_button_event.y()});//в строитель добавляется очередная вершина
          if(figure){
            figures.emplace_back(std::move(figure));//добавить в вектор фигур
            active_builder = nullptr;
          }
```

```
        }
      }
    }

    for(const std::unique_ptr<figure>& figure: figures){
      figure->render(renderer);//рисуем фигуры
    }

    ImGui::Begin("Menu");
    if (ImGui::Button("New canvas")) {
         jl.figs = std::move(figures);
         jl.push(0, nullptr);
    }
    ImGui::InputText("File name", file_name, file_name_length - 1);
    if(ImGui::Button("Save")){
      std::ofstream os(file_name);
      if(os){
        for(const std::unique_ptr<figure>& figure: figures){
          figure->save(os);
        }
      }
    }
    ImGui::SameLine();
    if (ImGui::Button("Load")) {
        std::ifstream is(file_name);
        if (is) {
             loader loader;
            figures = loader.load(is);
            jl.push(2, nullptr);
        }
    }

    if(ImGui::Button("Trapeze")){
      active_builder = std::make_unique<trapeze_builder>();
    }

    ImGui::SameLine();
    if (ImGui::Button("Rhombus")) {
        active_builder = std::make_unique<rhombus_builder>();
    }
    ImGui::SameLine();
    if (ImGui::Button("Pentagon")) {
        active_builder = std::make_unique<pentagon_builder>();
    }
    ImGui::InputInt("Remove id", &remove_id);
    if(ImGui::Button("Remove")){
        if (remove_id < figures.size()) {
            jl.push(-1, std::move(*(figures.begin() + remove_id)));
            figures.erase(figures.begin() + remove_id);
        }
    }

    if (ImGui::Button("Undo")) {
        jl.undo();
    }
    ImGui::End();

    renderer.present();
  }
}
```

Pentagon.h

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

```
#include<memory>
struct  pentagon : figure {
    pentagon(const std::array<vertex, 5>& vertices) : vertices_(vertices) {}

    void render(const sdl::renderer& renderer) const override {//рисует фигуру
        renderer.set_color(255, 0, 0);
        for (int32_t i = 0; i < 5; ++i) {
            renderer.draw_line(vertices_[i].x, vertices_[i].y,
                vertices_[(i + 1) % 5].x, vertices_[(i + 1) % 5].y);
        }
    }

    void save(std::ostream& os) const override {//сохраняет фигуру
        os << "pentagon" << std::endl;
        for (int32_t i = 0; i < 5; ++i) {
            os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
        }
    }

private:
    std::array<vertex, 5> vertices_;//хранилище вершин треугльника

};
```

Rhombus.h

```
#pragma once
#include"figure.h"
#include<memory>
struct  rhombus : figure {
    rhombus(const std::array<vertex, 4>& vertices) : vertices_(vertices) {}

    void render(const sdl::renderer& renderer) const override {//рисует фигуру
        renderer.set_color(255, 0, 0);
        for (int32_t i = 0; i < 4; ++i) {
            renderer.draw_line(vertices_[i].x, vertices_[i].y,
                vertices_[(i + 1) % 4].x, vertices_[(i + 1) % 4].y);
        }
    }

    void save(std::ostream& os) const override {//сохраняет фигуру
        os << "rhombus" << std::endl;
        for (int32_t i = 0; i < 4; ++i) {
            os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
        }
    }

private:
    std::array<vertex, 4> vertices_;//хранилище вершин треугльника

};
```

Tools.h

```
#pragma once
#include"figure.h"
#include"trapeze.h"
#include"pentagon.h"
#include"rhombus.h"
#include<string>
#include<vector>
#include<memory>
#include<stack>
```

```
static std::vector<std::unique_ptr<figure>> figures;

struct journal {
    struct j_element {
        int exp;
        std::unique_ptr<figure> fig;
        j_element(int _exp, std::unique_ptr<figure> _fig) : exp(_exp),
fig(std::move(_fig)) {}
    };
    std::stack<j_element> stack;
    std::vector<std::unique_ptr<figure>> figs;
    void push(int exp, std::unique_ptr<figure> value) {
        this->stack.push(j_element(exp, std::move(value)));
    }
    void undo() {
        //stack.top();
        if (stack.size() > 0) {
            if (stack.top().exp == 1) {
                figures.pop_back();
            }
            else if (stack.top().exp == -1) {
                figures.emplace_back(std::move(stack.top().fig));
            }
            else if (stack.top().exp == 0) {
                figures = std::move(figs);
            }
            else if (stack.top().exp == 2) {
                figures.clear();
            }
            stack.pop();
        }
    }
};
static journal jl;

struct builder {
    virtual std::unique_ptr<figure> add_vertex(const vertex& v) = 0;//функция добавления
новой вершины в фигуру

    virtual ~builder() = default;//деструктор, ненужная фигня, но должна быть

};

struct loader {
    std::vector<std::unique_ptr<figure>> load(std::ifstream& is) {
        std::string figure_name;
        std::vector<std::unique_ptr<figure>> figures;
        while (is >> figure_name) {
            vertex v;
            if (figure_name == std::string("trapeze")) {
                std::array<vertex, 4> vertices;
                for (int i = 0; i < 4; ++i) {
                    is >> v.x >> v.y;
                    vertices[i] = v;
                }
                figures.emplace_back(std::make_unique<trapeze>(vertices));
            }
            else if (figure_name == std::string("pentagon")) {
                std::array<vertex, 5> vertices;
                for (int i = 0; i < 5; ++i) {
                    is >> v.x >> v.y;
                    vertices[i] = v;
                }
                figures.emplace_back(std::make_unique<pentagon>(vertices));
```

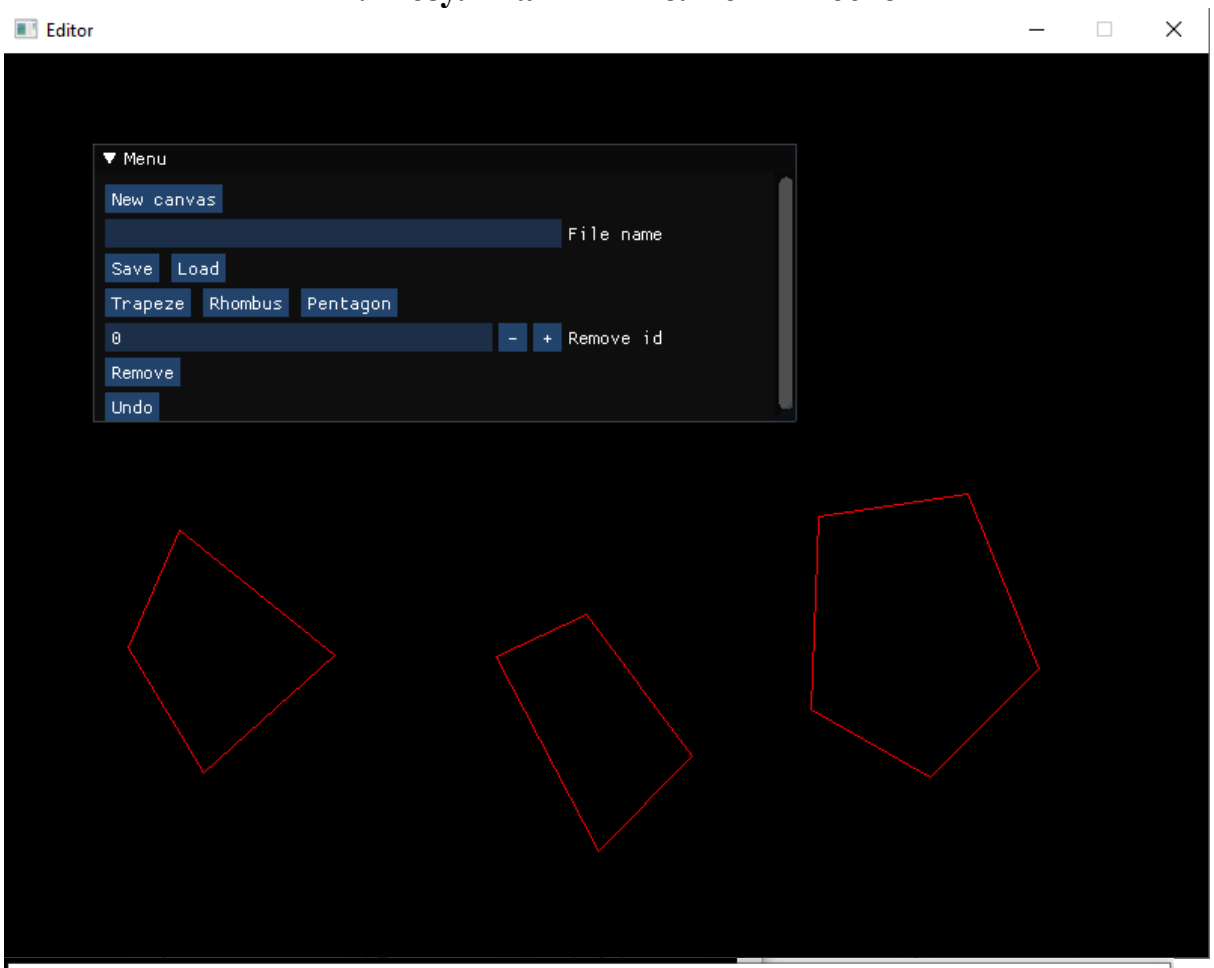
```
            }
            else if (figure_name == std::string("rhombus")) {
                std::array<vertex, 4> vertices;
                for (int i = 0; i < 4; ++i) {
                    is >> v.x >> v.y;
                    vertices[i] = v;
                }
                figures.emplace_back(std::make_unique<rhombus>(vertices));
            }
        }
        return figures;
    }
     ~loader() = default;
};


struct  pentagon_builder : builder {//строитель треугольника
    std::unique_ptr<figure> add_vertex(const vertex& v) {//добавить вершну V в
треугольник, наверно еще не построенный
         vertices_[n_] = v;
         n_ += 1;
         if (n_ != 5) {
             return nullptr;
         }
         jl.push(1, nullptr);
         return std::make_unique<pentagon>(vertices_);
    }
    private:
        int32_t n_ = 0; //кол-во вершин.
        std::array<vertex, 5> vertices_;
};

struct  rhombus_builder : builder {//строитель треугольника
    std::unique_ptr<figure> add_vertex(const vertex& v) {//добавить вершну V в
треугольник, наверно еще не построенны
        vertices_[n_] = v;
        n_ += 1;
        if (n_ != 4) {
            return nullptr;
        }
        jl.push(1, nullptr);
        return std::make_unique<rhombus>(vertices_);
    }

private:
    int32_t n_ = 0; //кол-во вершин.
    std::array<vertex, 4> vertices_;

};



struct  trapeze_builder : builder {//строитель треугольника
    std::unique_ptr<figure> add_vertex(const vertex& v) {//добавить вершну V в
треугольник, наверно еще не построенный
        vertices_[n_] = v;
        n_ += 1;
        if (n_ != 4) {
            return nullptr;
        }
        jl.push(1, nullptr);
        return std::make_unique<trapeze>(vertices_);
    }

private:
```

```
    int32_t n_ = 0; //кол-во вершин.
    std::array<vertex, 4> vertices_;

};
```

Trapeze.h

```
#pragma once
#include"figure.h"
#include<memory>
struct  trapeze : figure {
     trapeze(const std::array<vertex, 4>& vertices) : vertices_(vertices) {}

    void render(const sdl::renderer& renderer) const override {//рисует фигуру
        renderer.set_color(255, 0, 0);
        for (int32_t i = 0; i < 4; ++i) {
            renderer.draw_line(vertices_[i].x, vertices_[i].y,
                vertices_[(i + 1) % 4].x, vertices_[(i + 1) % 4].y);
        }
    }

    void save(std::ostream& os) const override {//сохраняет фигуру
        os << "trapeze" << std::endl;
        for (int32_t i = 0; i < 4; ++i) {
            os << vertices_[i].x << ' ' << vertices_[i].y << '\n';
        }
    }

private:
    std::array<vertex, 4> vertices_;//хранилище вершин треугльника

};
```

4. Результаты выполнения тестов



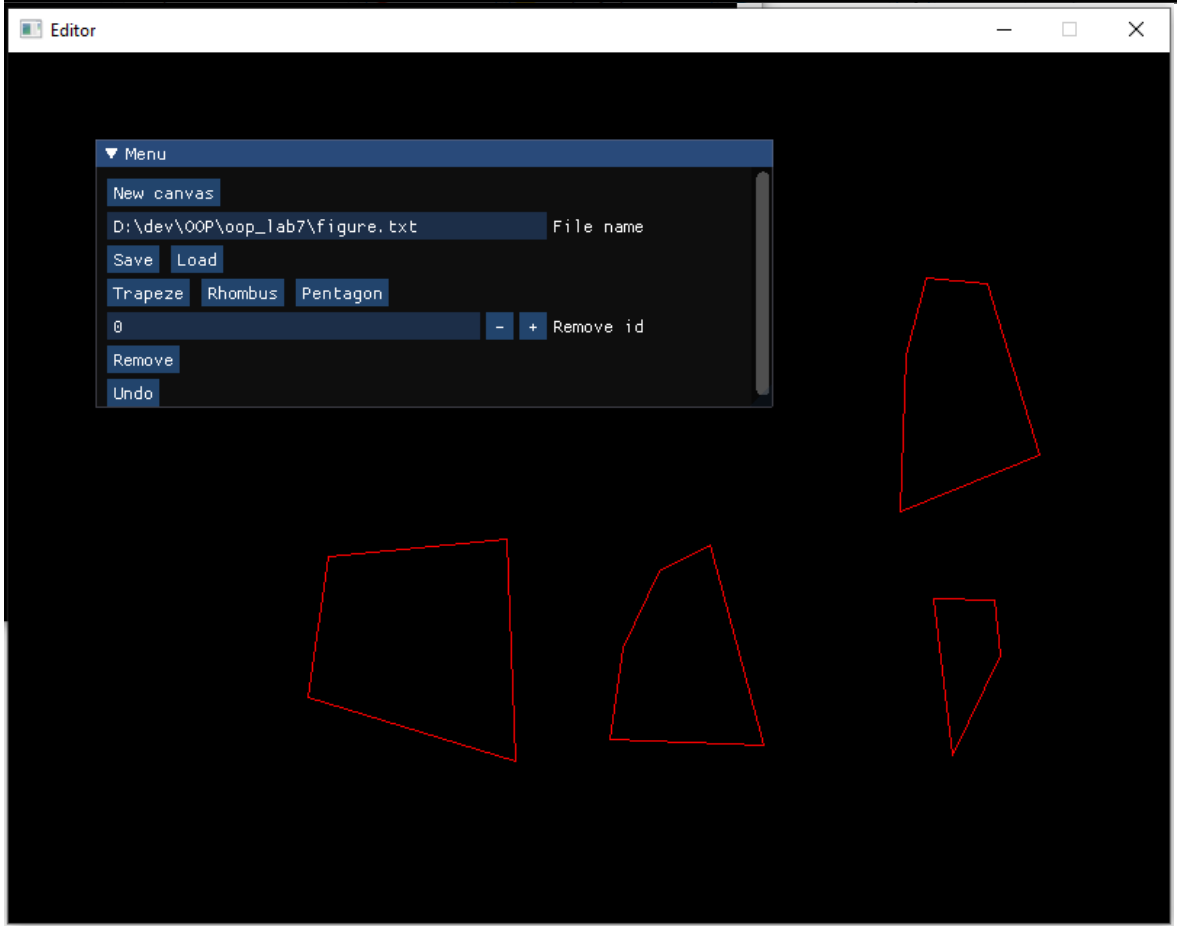


Figure.txt
trapeze
206 444
349 488
343 335
220 347
rhombus
650 484
683 415
679 377
637 376
pentagon
520 477
414 473
423 410
448 357
483 339
pentagon
710 277
614 316
618 208
632 155
674 159

5. Объяснение результатов работы программы

Программа запускается с появлением черного полотна и меню с функциями для рисования и удаления различных фигур. Пользователь может нарисовать сразу несколько видов фигур а затем удалить, либо сохранить их. Также есть возможность отмены прошлых действий при помощи кнопки undo. Внутри программы было реализовано несколько классов для построения, рендеринга и сохранения фигур, журнал изменения состояния полотна.

6. Вывод

Благодаря данной лабораторной работе студент может улучшить свои навыки в проектировании более сложных программ. Умение проектировать структуру классов позволяет сделать дальнейшую разработку более гибкой и простой, повысить читаемость кода, кроме того, в ходе выполнения работы студент может познакомится с графическими библиотеками и написать свой пользовательский интерфейс.