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: Пермяков Никита Александрович

Группа: 80-208

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2020

1. Постановка задачи

Цель:

Получение практических навыков в хороших практиках проектирования структуры классов приложения;

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;

Вариант: 33

реализовать:

· Ромб
· 5 угольник
· 6 угольник

2. Описание программы

Репозиторий содержит файлы:

- main.cpp - файл с заданием работы
- CMakeLists.txt - файл с конфигураций CMake
- report.doc - отчет о лабораторной работе
- figures.h - содержит реализацию фигур и все операции связанные с ними.
- factory.h - содержит класс для создания графических примитиве фигур.
- document.h - содержит все операции с документов, а также чтение и запись в документ информации об объекте фигуры с помощью сериализации.
- command.h - класс для вставки и удаления фигуры в контейнер и так же отмены удаления/вставки.
- editor.h - файл с обобщенным классом, который имеет операции над документом, вставкой/удалением фигуры и отменой вставки/удаления.

Пользователь может:

- добавить
- удалить
- сохранить
- импортировать данные из файла
- выполнить операцию undo (отмена последнего действия добавления/удаления фигуры).

В случае, если файл не сохранен, выдаются предупреждающие сообщения.

3. Набор тестов

Пояснение:

Программа получает на вход ключ команды, затем дополнительные опции, такие как название файла, для создания, удаления и сохранения. Тестируется операция undo, print, load, save, add, remove, exit, create, menu.

Test 1

```
2 firstfile

5 rec

0 0  5 0 5 10  0 10

5 trap

0 0  4 0  3 2  1 2

8

5 rec

5 5 10 5  10 12  5 12

5 rhomb

-2 0  0 -3  2 0  0 3

1

8

6 2

8

7

8

1

4 firstfile

1

5 rhomb

0 0  5 0  -3 2  4 5
```

9

Test 2

3 firstfile

8

5 rec

-5 -5 0 -5 0 10 -5 10

5 rhomb

0 0 2 -5 4 0 2 5

8

6 6

8

7

7

8

6 2

8

1

4 firstfile

9

Test 3

1

2 firstfile

1

5 Rhomb

0 0 5 0 -3 2 4 5

9

1. Результаты выполнения тестов

Test 1

```
1 - menu

2 - create new file

3 - load

4 - save [name]

5 - add [type]

6 - remove [id]

7 - undo

8 - print

9 - exit

2

Enter name of new project

firstfile

Document firstfile is created

5

Enter shape type:

        - rec

        - trap

        - rhomb

rec

Enter coordinates separated by space

0 0 5 0 5 10 0 10

Primitive is added

5

Enter shape type:

        - rec

        - trap

        - rhomb
```

```
trap

Enter the coordinates separated by space: 0 0 4 0 3 2 1 2

Primitive is added

8

Rectangle - id: 1

<0, 0> <5, 0> <5, 10> <0, 10>


Trapezoid id: 2

Coords: <0, 0><4, 0><3, 2><1, 2

7

OK

8

Rectangle - id: 1

<0, 0> <5, 0> <5, 10> <0, 10>

7

OK

8
```

Test 2

```
-5 -5 0 -5 0 10 -5 10

5 rhomb

Enter name of file to upload: Document loaded from file firstfile

Rectangle - id: 1

<0, 0> <5, 0> <5, 10> <0, 10>


Trapezoid id: 2

Coords: <0, 0><4, 0><3, 2><1, 2>


Rectangle - id: 3
```

```
<5, 5> <10, 5> <10, 12> <5, 12>

id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Enter shape type:

        - rec

        - trap

        - rhomb

Enter coordinates separated by space

Primitive is added

Enter shape type:

        - rec

        - trap

        - rhomb

Enter the coordinates separated by a space: 0 0  2 -5  4 0  2 5

8

6 6

8

7

7

8

6 2

8

1
```

```
4 firstfPrimitive is added

Rectangle - id: 1

<0, 0> <5, 0> <5, 10> <0, 10>

Trapezoid id: 2

Coords: <0, 0><4, 0><3, 2><1, 2>

Rectangle - id: 3

<5, 5> <10, 5> <10, 12> <5, 12>

id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Rectangle - id: 4

<-5, -5> <0, -5> <0, 10> <-5, 10>

id: 5

Figure: Trapezoid

Coords:

<0, 0>

<2, -5>

<4, 0>

<2, 5>

Remove primitive at id: 6

Rectangle - id: 1

<0, 0> <5, 0> <5, 10> <0, 10>

Trapezoid id: 2

Coords: <0, 0><4, 0><3, 2><1, 2>
```

```
Rectangle - id: 3

<5, 5> <10, 5> <10, 12> <5, 12>

id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Rectangle - id: 4

<-5, -5> <0, -5> <0, 10> <-5, 10>

id: 5

Figure: Trapezoid

Coords:

<0, 0>

<2, -5>

<4, 0>

<2, 5>

OK

OK
```

Test 3

```
1 - menu

2 - create new file

3 - load

4 - save [name]

5 - add [type]

6 - remove [id]

7 - undo
```

```
8 - print

9 - exit

1

2 firstfile

1

5 Rhomb

0 0  5 0  -3 2  4 5

91 - menu

2 - create new file

3 - load

4 - save [name]

5 - add [type]

6 - remove [id]

7 - undo

8 - print

9 - exit

Enter name of new project

Document firstfile is created

1 - menu

2 - create new file

3 - load

4 - save [name]

5 - add [type]

6 - remove [id]

7 - undo

8 - print

9 - exit

Enter shape type:
        - rec

        - trap

        - rhomb

Primitive isn't added

Enter shape type:

        - rec

        - trap

        - rhomb

Primitive isn't added

Save old document? Yes/No Enter name of new project

Document 5 is created
```

5. Листинг программы

main.cpp

```cpp
#include <iostream>

#include <string>

#include "editor.h"


void create(Editor &editor) {

    std::string cmd;

    if (editor.DocumentExist()) {

        std::cout << "Save old document? Yes/No ";

        std::cin >> cmd;

        if (cmd == "Yes" || cmd == "Y") {

            std::string filename;

            std::cout << "Enter name of file: ";

            std::cin >> filename;

            try {

                editor.SaveDocument(filename);

            } catch (std::runtime_error &err) {
                std::cout << err.what() << std::endl;
            }
        }
    }
    std::cout << "Enter name of new project" << std::endl;
    std::cin >> cmd;
    editor.CreateDocument(cmd);
    std::cout << "Document " << cmd << " is created" << std::endl;
}

void save(Editor &editor) {
    if (!editor.DocumentExist())
        throw std::runtime_error("Document does not exist");

    std::string filename;
    std::cin >> filename;

    try {
        editor.SaveDocument(filename);
        std::cout << "Document save in file " << filename << std::endl;
    } catch (std::runtime_error &err) {
        std::cout << err.what() << std::endl;
    }
}

void load(Editor &editor) {
    std::string cmd;
    std::string filename;
    if (editor.DocumentExist()) {
        std::cout << "Save old document? Yes/No ";
        std::cin >> cmd;
        if (cmd == "Yes") {
            std::cout << "Enter name of file ";
            std::cin >> filename;
            try {
                editor.SaveDocument(filename);
            } catch (std::runtime_error& err) {
                std::cout << err.what() << std::endl;
            }
        }
    }
    std::cout << "Enter name of file to upload: ";
    std::cin >> filename;
    try {
        editor.LoadDocument(filename);
        std::cout << "Document loaded from file " << filename << std::endl;
    } catch (std::runtime_error& err) {
        std::cout << err.what() << std::endl;
    }
}

void add(Editor &editor) {
    if (!editor.DocumentExist())
        throw std::runtime_error("Document does not exist");
    std::string type;
    std::cout << "Enter shape type: \n\t- rec \n\t- trap \n\t- rhomb\n";
    std::cin >> type;

    std::pair<double, double> *vertices = new std::pair<double, double>[4];
    if (type == "rec") {
        std::cout << "Enter coordinates separated by space\n";
        for (int i = 0; i < 4; ++i) {
            std::cin >> vertices[i];
        }
        try {
            editor.InsertPrimitive(rec, vertices);
            delete [] vertices;
            vertices = nullptr;
        } catch (std::logic_error &err) {
            std::cout << err.what() << std::endl;
            delete [] vertices;
            vertices = nullptr;
            return;
        }
        std::cout << "Primitive is added" << std::endl;
    }
    else if (type == "trap") {
        std::cout << "Enter the coordinates separated by space: ";
        for (int i = 0; i < 4; ++i) {
            std::cin >> vertices[i];
        }
        try {
            editor.InsertPrimitive(trap, vertices);
            delete [] vertices;
            vertices = nullptr;
        } catch (std::logic_error &err) {
            std::cout << err.what() << std::endl;
            delete [] vertices;
            vertices = nullptr;
            return;
        }
        std::cout << "Primitive is added"  << std::endl;
    }
    else if (type == "rhomb") {
        std::cout << "Enter the coordinates separated by a space: ";
        for (int i = 0; i < 4; ++i) {
            std::cin >> vertices[i];
        }
        try {
            editor.InsertPrimitive(rhomb, vertices);
            delete [] vertices;
            vertices = nullptr;
        } catch (std::logic_error &err) {
            std::cout << err.what() << std::endl;
            delete [] vertices;
            vertices = nullptr;
            return;
        }
        std::cout << "Primitive is added" << std::endl;
    }
    else {
        std::cout << "Primitive isn't added" << std::endl;
        return;
    }
}

void remove(Editor &editor) {
    if (!editor.DocumentExist())
        std::cout << "Document does not exist" << std::endl;
    int id;
    std::cin >> id;
    if (id <= 0) {
        std::cout << "Invalid id" << std::endl;
        return;
    }
    try {
        editor.RemovePrimitive(id);
    } catch (std::exception &e) {
        std::cout << "Invalid id" << std::endl;
        return;
    }
    std::cout << "Remove primitive at id: " << id << std::endl;
}

void menu() {
    std::cout << "1 - menu" << std::endl;
    std::cout << "2 - create new file" << std::endl;
    std::cout << "3 - load" << std::endl;
    std::cout << "4 - save [name]" << std::endl;
    std::cout << "5 - add [type]" << std::endl;
    std::cout << "6 - remove [id]" << std::endl;
    std::cout << "7 - undo" << std::endl;
    std::cout << "8 - print" << std::endl;
    std::cout << "9 - exit" << std::endl;
}

int main() {
    Editor editor;
    uint16_t cmd = 1;

    while(cmd != 9) {
        if (cmd == 1) {
            menu();
        }
        else if (cmd == 2) {
            create(editor);
        }
        else if (cmd == 3) {
            try {
                load(editor);
            } catch (std::runtime_error &err) {
                std::cout << err.what() << "\n\n";
            }
        }
        else if (cmd == 4) {
            try {
                save(editor);
            } catch (std::runtime_error &err) {
            }
        }
```

```
else if (cmd == 5) {
    try {
        add(editor);
    } catch (std::runtime_error &err) {
        std::cout << err.what() << "\n\n";
    }
}
else if (cmd == 6) {
    try {
        remove(editor);
    } catch (std::exception &err) {
        std::cout << err.what() << std::endl;
    }
}
else if (cmd == 7) {
    try {
        editor.Undo();
        std::cout << "OK\n";
    } catch (std::logic_error &err) {
        std::cout << err.what() << "\n\n";
    }
}
else if (cmd == 8) {
    if (!editor.DocumentExist()) {
        std::cout << "Document does not exist" << "\n\n";
        continue;
    }
    editor.PrintDocument();
}
else if(cmd == 9){
    return 0;
} else {
    std::cout << "You did not choose an action\n";
}
```

```
        std::cin >> cmd;
        std::cout << std::endl;
    }
    return 0;
}
```

Rectangle.h

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

class Rectangle : public Figure {
public:
    Rectangle(): id{0}, vertices{new std::pair<double, double>[4]} {
        for (uint16_t i = 0; i < 4; ++i){
            this->vertices[i] = std::make_pair(0, 0);
        }
    }

    Rectangle(
        std::pair<double, double> &a,
        std::pair<double, double> &b,
        std::pair<double, double> &c,
        std::pair<double, double> &d,
        uint16_t id
        ) : id{id}, vertices{new std::pair<double, double>[4]} {
        if (a == b || a == c || b == c || a == d ||
            !(perpendicular(a, b, a, d)) ||
            !collinear(a, d, c, b) ||
            !collinear(a, b, d, c)
        ) {
            throw std::logic_error("Entered coordinates of vertices do not belong to rectangle.");
    } else {
        this->vertices[0] = a;
        this->vertices[1] = b;
        this->vertices[2] = c;
        this->vertices[3] = d;
    }
}

~Rectangle() override {
    delete [] this->vertices;
    this->vertices = nullptr;
}

std::pair<double, double> Center() const override {
    uint16_t number = 4;
    return getCenter(this->vertices, number);
}

double Area() const override {
    auto AB = dist(this->vertices[0], vertices[1]);
    auto AD = dist(this->vertices[0],vertices[3]);
    return AD * AB;
}

std::ostream &Print(std::ostream &out) const override {
    out << "Rectangle - id: " << id << "\n";
    for (uint16_t i = 0; i < 4; ++i) {
        out << vertices[i] << " ";
    }
    out << "\n";
    return out;
}

void Serialize(std::ofstream &os) const override {
        FigureType type = rec;
        os.write((char *) &type, sizeof(type));
        os.write((char *) &id, sizeof(id));
        for (uint16_t i = 0; i < 4; ++i) {
            os.write((char*) &(vertices[i].first), sizeof(vertices[i].first));
            os.write((char*) &(vertices[i].second), sizeof(vertices[i].second));
        }
    }

    void Deserialize(std::ifstream &is) override {
        is.read((char *) &id, sizeof(id));
        for (uint16_t i = 0; i < 4; ++i) {
            is.read((char *) &(vertices[i].first), sizeof(vertices[i].first));
            is.read((char *) &(vertices[i].second), sizeof(vertices[i].second));
        }
    }

    int getId() const override {
        return (int)id;
    }

private:
    uint16_t id;
    std::pair<double, double> *vertices;
};
```

Trapedzoid.h

```
#pragma once

#include"figures.h"




class Trapezoid : public Figure {
```

```
public:
    Trapezoid() : id{0}, vertices{new std::pair<double, double>[4]} {
        for (uint16_t i = 0; i < 4; ++i){
            this->vertices[i] = std::make_pair(0,0);
        }
    }

    Trapezoid(std::pair<double, double> &a, std::pair<double, double> &b, std::pair<double,
double> &c, std::pair<double, double> &d, uint16_t id) :
    id{id}, vertices{new std::pair<double, double>[4]} {
        if (a == b || a == c || b == c || a == d || b == d || c == d ||
            collinear(a, b, c, a) || collinear(a, b, d, a) || collinear(a, c, d, a)
            || collinear(b, c, d, b)) {
            throw std::logic_error("The entered coordinates of the vertices do not belong to the
trapezoid.");
        } else {
            this->vertices[0] = a;
            this->vertices[1] = b;
            this->vertices[2] = c;
            this->vertices[3] = d;
        }
    }

    ~Trapezoid() override {
        delete [] this->vertices;
        this->vertices = nullptr;
    }

    std::pair<double, double> Center() const override {
        uint16_t num = 4;
        return getCenter(this->vertices, num);
    }

    double Area() const override {
        auto area = ((vertices[0].first * vertices[1].second - vertices[1].first * vertices[0].second)
+
        (vertices[1].first * vertices[2].second - vertices[2].first * vertices[1].second) +
        (vertices[2].first * vertices[3].second - vertices[3].first * vertices[2].second)) / 2;
        return std::abs(area);
    }

    std::ostream &Print(std::ostream &out) const override{
        out << "Trapezoid id: " << this->id << "\n";
        out << "Coords: ";
        for (uint16_t i = 0; i < 4; ++i) {
            out << this->vertices[i];
        }
        out << "\n";
        return out;
    }

    void Serialize(std::ofstream &os) const override{
        FigureType type = trap;
        os.write((char *) &type, sizeof(type));
        os.write((char *) &id, sizeof(id));
        for (uint16_t i = 0; i < 4; ++i) {
            os.write((char *) &(this->vertices[i].first), sizeof(this->vertices[i].first));
            os.write((char *) &(this->vertices[i].second), sizeof(this->vertices[i].second));
        }
    }

    void Deserialize(std::ifstream &is) override {
        is.read((char *) &id, sizeof(id));
        for (uint16_t i = 0; i < 4; ++i) {
            is.read((char *) &(this->vertices[i].first), sizeof(this->vertices[i].first));
            is.read((char *) &(this->vertices[i].second),sizeof(this->vertices[i].second));
        }
    }
    int getId() const override {
        return this->id;
    }

private:
    uint16_t id;
    std::pair<double, double> *vertices;
};
```

editor.h

```
#pragma once

#include <stack>
#include "document.h"
#include "command.h"



class Editor {
public:
    Editor() : doc(nullptr), history() {};

    void CreateDocument(const std::string &name) {
        this->doc = std::make_shared<Document>(name);
    }

    void InsertPrimitive(FigureType type, std::pair<double, double> *vertices) {
        std::shared_ptr<Command> command = std::shared_ptr<Command>(new
InsertCommand(type, vertices));
        command->SetDocument(this->doc);
        command->Execute();
        this->history.push(command);
    }
    void RemovePrimitive(int id) {
        try {
            std::shared_ptr<Command> command = std::shared_ptr<Command>(new
RemoveCommand(id));
            command->SetDocument(this->doc);
            command->Execute();
            this->history.push(command);
        } catch (std::exception &err) {
            std::cout << err.what() << std::endl;
            throw;
        }
    }

    void SaveDocument(const std::string &filename) {
        this->doc->Save(filename);
    }

    void LoadDocument(const std::string &filename) {
        this->doc = std::make_shared<Document>(filename);
        this->doc->Load(filename);
    }

    void Undo() {
        if (this->history.empty())
            throw std::logic_error("empty");

        std::shared_ptr<Command> lastCommand = this->history.top();
        lastCommand->UnExecute();
        this->history.pop();
    }

    void PrintDocument() {
        this->doc->Print();
    }

    bool DocumentExist() {
        return this->doc != nullptr;
    }

    ~Editor() = default;

private:
    std::shared_ptr<Document> doc;
    std::stack<std::shared_ptr<Command>> history;
};
```

factory.h

```
#pragma once

#include "figures.h"
#include "Rectangle.h"
#include "Rhombus.h"
#include "Trapezoid.h"



class Factory {
public:
    std::shared_ptr<Figure> FigureCreate(FigureType type) const {
        std::shared_ptr<Figure> res;
        if (type == rec) {
            res = std::make_shared<Rectangle>();
        } else if (type == rhomb) {
            res = std::make_shared<Rhombus>();
        } else if (type == trap) {
            res = std::make_shared<Trapezoid>();
        }
        return res;
    }

    std::shared_ptr<Figure> FigureCreate(FigureType type, std::pair<double, double>
*vertices, int id) const {
        std::shared_ptr<Figure> res;
        if (type == rec) {
            res = std::make_shared<Rectangle>(vertices[0], vertices[1], vertices[2], vertices[3],
id);
        } else if (type == rhomb) {
            res = std::make_shared<Rhombus>(vertices[0], vertices[1], vertices[2], vertices[3],
id);
        } else if (type == trap) {
            res = std::make_shared<Trapezoid>(vertices[0], vertices[1], vertices[2], vertices[3],
id);
        }

        return res;
    }
};
```

document.h

```
#pragma once

#include <fstream>
#include <list>
#include <stdexcept>
#include <string>
#include <algorithm>
#include <utility>
#include "figures.h"
#include "factory.h"
```

```
class Document {
public:
    Document() : id(1), name(""), buffer(0), factory() {};
    Document(std::string name) : id(1), name(name), buffer(0), factory() {};

    ~Document() = default;

    void Rename(const std::string &new_name) {
        this->name = new_name;
    }

    void Save(const std::string &filename) {
        SerialiseImpl(filename);
    }

    void Load(const std::string &filename) {
        DeserializeImpl(filename);
    }

    void Print() {
        std::for_each(this->buffer.begin(), this->buffer.end(), [](std::shared_ptr<Figure> shape) {
            shape->Print(std::cout) << std::endl;
        });
    }

    void RemovePrimitive(int id) {
        auto it = std::find_if(this->buffer.begin(), this->buffer.end(),
                [id](std::shared_ptr<Figure> shape) -> bool {
            return id == shape->getId();
        });

        if (it == this->buffer.end())
            throw std::logic_error("Figure with this id doesn't exist");
        this->buffer.erase(it);
    }

    void InsertPrimitive(FigureType type, std::pair<double, double> *vertices) {
        switch (type) {
            case rec:
                this->buffer.push_back(factory.FigureCreate(rec, vertices, id));
                break;
            case rhomb:
                this->buffer.push_back(factory.FigureCreate(rhomb, vertices, id));
                break;
            case trap:
                this->buffer.push_back(factory.FigureCreate(trap, vertices, id));
                break;
        }
        ++this->id;
    }

private:
    int id;
    std::string name;
    std::list<std::shared_ptr<Figure>> buffer;
    Factory factory;

    friend class InsertCommand;
    friend class RemoveCommand;

    void SerialiseImpl(const std::string &filename) const {
        std::ofstream os(filename, std::ios::binary | std::ios::out);
        if(!os)
            throw std::runtime_error("File is not opened");

        size_t len_name = this->name.size();
        os.write((char *) &len_name, sizeof(len_name));
        os.write((char *) this->name.c_str(), len_name);
        for (const auto &shape : this->buffer)
            shape->Serialize(os);
    }

    void DeserializeImpl(const std::string &filename) {
        std::ifstream is(filename, std::ios::binary | std::ios::in);
        if(!is)
            throw std::runtime_error("File is not opened");

        size_t len_name;
        is.read((char *) &len_name, sizeof(len_name));

        char *clear_name = new char[len_name + 1];
        clear_name[len_name] = 0;
        is.read(clear_name, len_name);
        this->name = std::string(clear_name);
        delete [] clear_name;

        FigureType type;
        while (true) {
            is.read((char *) &type, sizeof(type));
            if (is.eof())
                break;
            switch (type) {
                case rec:
                    this->buffer.push_back(factory.FigureCreate(rec));
                    break;
                case rhomb:
                    this->buffer.push_back(factory.FigureCreate(rhomb));
                    break;
                case trap:
                    this->buffer.push_back(factory.FigureCreate(trap));
                    break;
            }
            this->buffer.back()->Deserialize(is);
        }
        this->id = this->buffer.size();
    }

    std::shared_ptr<Figure> GetFigure(int id) {
        for (auto it = this->buffer.begin(); it != this->buffer.end(); ++it)
            if (id == (*it)->getId())
                return *it;
        return nullptr;
    }

    int getPos(int id) {
        int i = 0;
        for (auto it = this->buffer.begin(); it != this->buffer.end(); ++it) {
            if (id == (*it)->getId())
                return i;
            ++i;
        }
        return -1;
    }

    void InsertPrimitive(int pos, std::shared_ptr<Figure> figure) {
        auto it = this->buffer.begin();
        std::advance(it, pos);
        this->buffer.insert(it, figure);
    }

    void RemoveLastPrimitive() {
        if (this->buffer.empty())
            throw std::logic_error("Document is empty");
        this->buffer.pop_back();
    }
```

```
};
```

command.h

```
#pragma once

#include <stack>
#include "document.h"




class Command {
public:
    virtual void Execute() = 0;
    virtual void UnExecute() = 0;
    virtual ~Command() = default;

    void SetDocument(std::shared_ptr<Document> doc) {
        this->doc = doc;
    }
protected:
    std::shared_ptr<Document> doc;
};

class InsertCommand : public Command {
public:
    InsertCommand(FigureType type, std::pair<double, double> *vertices) :
            type{type}, vertices{vertices} {};

    void Execute() override {
        this->doc->InsertPrimitive(this->type, this->vertices);
    }

    void UnExecute() override {
        this->doc->RemoveLastPrimitive();
    }

private:
    FigureType type;
    std::pair<double, double> *vertices;
};



class RemoveCommand : public Command {
public:
    RemoveCommand(int id) : id(id), pos(0), figure(nullptr) {};

    void Execute() override {
        this->figure = this->doc->GetFigure(this->id);
        if (this->figure == nullptr) {
            return;
        }
        this->pos = this->doc->getPos(this->id);
        if (this->pos == 0) {
            return;
        }
        this->doc->RemovePrimitive(this->id);
    }

    void UnExecute() override {
        this->doc->InsertPrimitive(this->pos, this->figure);
    }

private:
    int id;
    int pos;
    std::shared_ptr<Figure> figure;
};
```

figures.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <utility>
#include <memory>
#include <cmath>
#include <stdexcept>


enum FigureType {
    rec,
    rhomb,
    trap,
};

class Figure{
    public:
    virtual double Area() const = 0;
    virtual std::pair<double, double> Center() const = 0;
    virtual std::ostream& Print(std::ostream& out) const = 0;
    virtual void Serialize(std::ofstream& os) const = 0;
    virtual void Deserialize(std::ifstream& is) = 0;
    virtual int getId() const = 0;
    virtual ~Figure() = default;
};

std::pair<double, double> getCenter(
    const std::pair<double, double> *vertices,
    uint16_t& n
    ) {
    double x = 0, y = 0;
    for (uint16_t i = 0; i < n; ++i) {
        x += vertices[i].first;
        y += vertices[i].second;
    }
    return {x / n, y / n};
}

std::pair<double, double> operator- (
    const std::pair<double, double> &p1,
    const std::pair<double, double> &p2
    ) {
    return {p1.first - p2.first, p1.second - p2.second};
}

bool collinear(
    const std::pair<double, double> &a,
    const std::pair<double, double> &b,
    const std::pair<double, double> &c,
    const std::pair<double, double> &d
    ){
    return (b.second-a.second)*(d.first-c.first) - (d.second-c.second)*(b.first-a.first) <= 1e-9;
}



bool perpendicular(
    const std::pair<double, double> &a,
    const std::pair<double, double> &b,
    const std::pair<double, double> &c,
    const std::pair<double, double> &d
    ){
    std::pair<double, double> AC = c - a;
    std::pair<double, double> BD = d - b;

    double normaAC = sqrt(pow(AC.second, 2) + pow(AC.first, 2));
    double normaDB = sqrt(pow(BD.second, 2) + pow(BD.first, 2));

    double hightAC = AC.second / normaAC;
    double widthAC = AC.first / normaAC;
    double hightBD = BD.second / normaDB;
    double widthBD = BD.first / normaDB;

    double cos_theta = hightAC * hightBD + widthAC * widthBD;

    return abs(cos_theta) < 1e-9;
}

double dist(
    const std::pair<double, double> &a,
    const std::pair<double, double> &b
    ){
    return sqrt(((b.first - a.first) * (b.first - a.first)) + ((b.second - a.second) * (b.second -
a.second)));
}

bool operator==(
    const std::pair<double, double> &a,
    const std::pair<double, double> &b
    ){
    return (a.first == b.first) && (a.second == b.second);
}

std::ostream& operator<<(std::ostream &o, const std::pair<double, double> &p){
    o << "<" << p.first << ", " << p.second << ">";
    return o;
}

std::istream& operator>>(std::istream &is, std::pair<double, double> &p){
    std::string checker;
    p.first = static_cast<double>(std::stod(checker));
    p.second = static_cast<double>(std::stod(checker));
    return is;
}
```

Rhombus.h

```
#pragma once

#include"figures.h"




class Rhombus: public Figure {
public:
    Rhombus() : id{0}, vertices{new std::pair<double, double>[4]} {
        for (uint16_t i = 0; i < 4; ++i){
            this->vertices[i] = {0, 0};
        }
    }
    Rhombus(std::pair<double, double> &a, std::pair<double, double> &b, std::pair<double,
double> &c, std::pair<double, double> &d, uint16_t id) :
    id{id}, vertices{new std::pair<double, double>[4]} {
        auto AB = dist(a, b);
        auto AD = dist(a, d);
        auto BC = dist(b,c);
        auto CD = dist(c,d);
        if (a == b || a == c || b == c || a == d || b == d || c == d ||
            !(AB == AD) || !(CD == BC) || !(AB == CD) ||
            !perpendicular(a,b,c,d) ||
            !collinear(a, b, c, d) ||
            !collinear(a, d, c, b)
        ) {
            throw std::logic_error("The entered coordinates of the vertices do not belong to the
trapezoid.");
```

```
    } else {
        this->vertices[0] = a;
        this->vertices[1] = b;
        this->vertices[2] = c;
        this->vertices[3] = d;
    }
}

~Rhombus() override {
    delete [] this->vertices;
    this->vertices = nullptr;
}

std::pair<double, double> Center() const override {
    uint16_t num = 4;
    return getCenter(this->vertices, num);
}

double Area() const override {
    double AC = dist(vertices[0],vertices[2]);
    double DB = dist(vertices[3],vertices[1]);
    return (AC*DB) /2;
}

std::ostream &Print(std::ostream &out) const override{
    out << "id: " << id << "\n";
    out << "Figure: Trapezoid\n";
    out << "Coords:\n";
    for (uint16_t i = 0; i < 4; ++i) {
        out << vertices[i] << "\n";
    }
    return out;
}
```

```
    void Serialize(std::ofstream &os) const override{
        FigureType type = rhomb;
        os.write((char *) &type, sizeof(type));
        os.write((char *) &id, sizeof(id));
        for (uint16_t i = 0; i < 4; ++i) {
            os.write((char *) &(vertices[i].first),sizeof(vertices[i].first));
            os.write((char *) &(vertices[i].second),sizeof(vertices[i].second));
        }
    }

    void Deserialize(std::ifstream &is) override {
        is.read((char *) &id, sizeof(id));
        for (uint16_t i = 0; i < 4; ++i) {
            is.read((char *) &(vertices[i].first),
                    sizeof(vertices[i].first));
            is.read((char *) &(vertices[i].second),
                    sizeof(vertices[i].second));
        }
    }

    int getId() const override {
        return id;
    }

private:
    uint16_t id;
    std::pair<double, double> *vertices;
};
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)
project(lab7)
```

```
add_executable(lab7 main.cpp)
set_property(TARGET lab7 PROPERTY CXX_STANDARD 11)
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra -g")
```

6. Ссылка на репозиторий

https://github.com/nikit34/oop_exercise_07

7. Объяснение результатов работы программы

В программе реализованы функции сохранения Прямоугольника, Ромба, Трапеции в файл, загрузки из файла и отмены последнего добавления удаления фигуры в файл.

8. Вывод

Научился проектировать структуры классов для решения более сложных задач и грамотной организации кода. При проектировании структуры классов важно использовать паттерны проектирования, которые позволяют поддерживать код в долгосрочных и больших проектах и изменять меньшую его часть при внесении правок.

Список литературы

Проектирование классов C++ [Электронный ресурс]. URL: https://metanit.com/cpp/tutorial/5.14.php

(дата обращения: 16.12.2020).

2. Академическое программирование C++ [Электронный ресурс]. URL:http://www.c-cpp.ru/books/akkadem.pdf

(дата обращения: 16.12.2020).