**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: Минибаев Айдар

Группа: 80-201Б-18

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2019

1. **Постановка задачи**

Вариант 3:

Реализовать простейший графический редактор с возможностью сохранения документов в файлы, и загрузки из них. Применить паттерн “Factory”.

|  |  |  |  |
| --- | --- | --- | --- |
| 3. | Прямоугольник | Трапеция | Ромб |

1. **Описание программы**

Программа состоит из 6 файлов:

1) figures.h - содержит реализацию фигур и все операции связанные с ними.

2) factory.h - содержит класс для создания графических примитиве фигур.

3) document.h - содержит все операции с документов, а также чтение и запись в документ информации об объекте фигуры с помощью сериализации.

4) command.h - класс для вставки и удаления фигуры в контейнер и так же отмены удаления/вставки.

5) editor.h - файл с обобщенным классом, который имеет операции над документом, вставкой/удалением фигуры и отменой вставки/удаления.

6) main.cpp - файл с взаимодействием с пользователем.

Пользователь может добавить или удалить одну из 3 фигур, сохранить, импортировать данные из файла, выполнить операцию *undo(отмена последнего действия добавления/удаления фигуры)*. В случае, если файл не сохранен, выдаются предупреждающие сообщения.

1. **Набор тестов**

В тестах создадим файл, добавим в него пару фигур и сохраним,

*test\_01.txt*:

Создаем файл, добавляем пару фигур, проверяем работу комнады undo, задаем неверно заданные фигуры

Create firstfile

Add Rec

0 0 5 0 5 10 0 10

Add Trapeze

0 0 4 0 3 2 1 2

Print

Add Rec

5 5 10 5 10 12 5 12

Add Rhomb

-2 0 0 -3 2 0 0 3

Menu

Print

Remove 3

Print

Undo

Print

Menu

Save firstfile

Menu

Add Rhomb

0 0 5 0 -3 2 4 5

Add Rec

gfg

Exit

*test\_02.txt*:

Загрузим документ с фигурами из 1 теста и добавим несколько фигур и применим несколько раз операцию undo.

Load firstfile

Print

Add Rec

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

Add Rhomb

0 0 2 -5 4 0 2 5

Print

Remove 6

Print

Undo

Undo

Print

Remove 2

Print

Menu

Save firstfile

Exit

1. **Результаты выполнения тестов.**

***test\_01.txt*:**

Menu

Create[Create] new file

Load

Save <fileName>

Add <figureType>

Remove <figure ID>

Undo

Print

Exit

Create

Enter name of new project

firstfile

Document firstfile is created

Add

Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].

Rec

Enter the coordinates separated by a space

0 0

5 0

5 10

0 10

Primitive is added

Add

Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].

Trapeze

Enter the coordinates separated by a space

0 0 4 0 3 2 1 2

Primitive is added

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 2

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 2>

<1, 2>

Add

Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].

Rec

Enter the coordinates separated by a space

5 5

10 5

10 12

5 12

Primitive is added

Add

Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].

Rhomb

Enter the coordinates separated by a space

-2 0 0 -3 2 0 0 3

Primitive is added

Menu

Menu

Create[Create] new file

Load

Save <fileName>

Add <figureType>

Remove <figure ID>

Undo

Print

Exit

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 2

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 2>

<1, 2>

Id: 3

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 12>

<5, 12>

Id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Remove 3

Remove primitive at id: 3

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 2

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 2>

<1, 2>

Id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Undo

OK

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 2

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 2>

<1, 2>

Id: 3

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 12>

<5, 12>

Id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Menu

Menu

Create[Create] new file

Load

Save <fileName>

Add <figureType>

Remove <figure ID>

Undo

Print

Exit

Save firstfile

Document save in file firstfile

Menu

Menu

Create[Create] new file

Load

Save <fileName>

Add <figureType>

Remove <figure ID>

Undo

Print

Exit

Add

Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].

Rhomb

Enter the coordinates separated by a space

0 0 5 0 -3 2 4 5

The entered coordinates of the vertices do not belong to the trapezoid.

Add

Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].

Rec

Enter the coordinates separated by a space

gfg

Is not a number

You did not choose an action

Exit

Process finished with exit code 0

***test\_02.txt*:**

Menu

Create[Create] new file

Load

Save <fileName>

Add <figureType>

Remove <figure ID>

Undo

Print

Exit

Load firstfile

Enter the name of the file to upload:

Document loaded from file firstfile

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 2

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 2>

<1, 2>

Id: 3

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 12>

<5, 12>

Id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Add Rec

Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].

Enter the coordinates separated by a space

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

Primitive is added

Add Rhomb

Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].

Enter the coordinates separated by a space

0 0 2 -5 4 0 2 5

Primitive is added

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 2

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 2>

<1, 2>

Id: 3

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 12>

<5, 12>

Id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Id: 5

Figure: Rectangle

Coords:

<-5, -5>

<0, -5>

<0, 10>

<-5, 10>

Id: 6

Figure: Trapezoid

Coords:

<0, 0>

<2, -5>

<4, 0>

<2, 5>

Remove 6

Remove primitive at id: 6

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 2

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 2>

<1, 2>

Id: 3

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 12>

<5, 12>

Id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Id: 4

Figure: Rectangle

Coords:

<-5, -5>

<0, -5>

<0, 10>

<-5, 10>

Undo

OK

Undo

OK

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 2

Figure: Trapezoid

Coords:

<0, 0>

<4, 0>

<3, 2>

<1, 2>

Id: 3

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 12>

<5, 12>

Id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Id: 5

Figure: Rectangle

Coords:

<-5, -5>

<0, -5>

<0, 10>

<-5, 10>

Remove 2

Remove primitive at id: 2

Print

Id: 1

Figure: Rectangle

Coords:

<0, 0>

<5, 0>

<5, 10>

<0, 10>

Id: 3

Figure: Rectangle

Coords:

<5, 5>

<10, 5>

<10, 12>

<5, 12>

Id: 4

Figure: Trapezoid

Coords:

<-2, 0>

<0, -3>

<2, 0>

<0, 3>

Id: 5

Figure: Rectangle

Coords:

<-5, -5>

<0, -5>

<0, 10>

<-5, 10>

Menu

Menu

Create[Create] new file

Load

Save <fileName>

Add <figureType>

Remove <figure ID>

Undo

Print

Exit

Save firstfile

Document save in file firstfile

Exit

Process finished with exit code 0

1. **Листинг программы**

**main.cpp**

/\*

Простейший графический редактор

Вариант 3: Прямоугольник, трапеция, ромб.

Минибаев Айдар

М8О-201Б-18

\*/

#include <iostream>

#include <string>

#include "editor.h"

void menu() {

std::cout << "\nMenu\n";

std::cout << "Create[Create] new file\n";

std::cout << "Load\n";

std::cout << "Save <fileName>\n";

std::cout << "Add <figureType>\n";

std::cout << "Remove <figure ID>\n";

std::cout << "Undo\n";

std::cout << "Print\n";

std::cout << "Exit\n\n";

}

void create(Editor &editor) {

std::string cmd;

if (editor.DocumentExist()) {

std::cout << "Save old document or don't save? Yes/No\n";

std::cin >> cmd;

if (cmd == "Yes") {

std::string filename;

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

std::cin >> filename;

try {

editor.SaveDocument(filename);

std::cout << "Document save in file " << filename << "\n";

} catch (std::runtime\_error &err) {

std::cout << err.what() << "\n";

}

}

}

std::cout << "Enter name of new project\n";

std::cin >> cmd;

editor.CreateDocument(cmd);

std::cout << "Document " << cmd << " is created\n";

}

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 << "\n";

} catch (std::runtime\_error &err) {

std::cout << err.what() << "\n";

}

}

void load(Editor &editor) {

std::string cmd;

std::string filename;

if (editor.DocumentExist()) {

std::cout << "Save old document or don't save? Yes/No\n";

std::cin >> cmd;

if (cmd == "Yes") {

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

std::cin >> filename;

try {

editor.SaveDocument(filename);

std::cout << "Document save in file " << filename << "\n";

} catch (std::runtime\_error& err) {

std::cout << err.what() << "\n";

}

}

}

std::cout << "Enter the name of the file to upload:\n";

std::cin >> filename;

try {

editor.LoadDocument(filename);

std::cout << "Document loaded from file " << filename << "\n";

} catch (std::runtime\_error& err) {

std::cout << err.what() << "\n";

}

}

void add(Editor &editor) {

if (!editor.DocumentExist()) {

throw std::runtime\_error("Document does not exist");

}

std::string type;

std::cout << "Enter shape type: Rectangle[Rec], Trapezoid[Trapeze] or Rhombus[Rhomb].\n";

std::cin >> type;

if (type == "Rec") {

Vertex \*vertices = new Vertex[4];

std::cout << "Enter the coordinates separated by a 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() << "\n";

delete [] vertices;

vertices = nullptr;

return;

}

std::cout << "Primitive is added\n";

}

else if (type == "Trapeze") {

Vertex \*vertices = new Vertex[4];

std::cout << "Enter the coordinates separated by a space\n";

for (int i = 0; i < 4; i++) {

std::cin >> vertices[i];

}

try {

editor.InsertPrimitive(Trapeze, vertices);

delete [] vertices;

vertices = nullptr;

} catch (std::logic\_error &err) {

std::cout << err.what() << "\n";

delete [] vertices;

vertices = nullptr;

return;

}

std::cout << "Primitive is added\n";

}

else if (type == "Rhomb") {

Vertex \*vertices = new Vertex[4];

std::cout << "Enter the coordinates separated by a space\n";

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() << "\n";

delete [] vertices;

vertices = nullptr;

return;

}

std::cout << "Primitive is added\n";

}

else {

std::cout << "Primitive isn't added\n";

return;

}

}

void remove(Editor &editor) {

if (!editor.DocumentExist()) {

std::cout << "Document does not exist\n";

}

int id;

std::cin >> id;

if (id <= 0) {

std::cout << "Invalid id\n";

return;

}

try {

editor.RemovePrimitive(id);

} catch (std::exception &e) {

std::cout << "Invalid id\n";

return;

}

std::cout << "Remove primitive at id: " << id << "\n";

}

int main() {

Editor editor;

std::string cmd = "Menu";

while(cmd != "Ex") {

if (cmd == "Menu") {

menu();

}

else if (cmd == "Create") {

create(editor);

}

else if (cmd == "Save") {

try {

save(editor);

} catch (std::runtime\_error &err) {

}

}

else if (cmd == "Load") {

try {

load(editor);

} catch (std::runtime\_error &err) {

std::cout << err.what() << "\n\n";

}

}

else if (cmd == "Add") {

try {

add(editor);

} catch (std::runtime\_error &err) {

std::cout << err.what() << "\n\n";

}

}

else if (cmd == "Remove") {

try {

remove(editor);

} catch (std::exception &err) {

std::cout << err.what() << "\n";

}

}

else if (cmd == "Undo") {

try {

editor.Undo();

std::cout << "OK\n";

} catch (std::logic\_error &err) {

std::cout << err.what() << "\n\n";

}

}

else if (cmd == "Print") {

if (!editor.DocumentExist()) {

std::cout << "Document does not exist" << "\n\n";

continue;

}

editor.PrintDocument();

}

else if(cmd == "Exit"){

return 0;

} else {

std::cout << "You did not choose an action\n";

}

std::cin >> cmd;

std::cout << "\n";

}

return 0;

}

**editor.h**

#ifndef EDITOR\_H

#define EDITOR\_H

#include "document.h"

#include "command.h"

#include <stack>

class Editor {

public:

Editor() : Doc(nullptr), History() {};

void CreateDocument(const std::string &name) {

Doc = std::make\_shared<Document>(name);

}

void InsertPrimitive(FigureType type, Vertex \*vertices) {

std::shared\_ptr<Command> command = std::shared\_ptr<Command>(new InsertCommand(type, vertices));

command->SetDocument(Doc);

command->Execute();

History.push(command);

}

void RemovePrimitive(int id) {

try {

std::shared\_ptr<Command> command = std::shared\_ptr<Command>(new RemoveCommand(id));

command->SetDocument(Doc);

command->Execute();

History.push(command);

} catch (std::exception &err) {

std::cout << err.what() << "\n";

throw;

}

}

void SaveDocument(const std::string &filename) {

Doc->Save(filename);

}

void LoadDocument(const std::string &filename) {

Doc = std::make\_shared<Document>(filename);

Doc->Load(filename);

}

void Undo() {

if (History.empty()) {

throw std::logic\_error("empty");

}

std::shared\_ptr<Command> lastCommand = History.top();

lastCommand->UnExecute();

History.pop();

}

void PrintDocument() {

Doc->Print();

}

bool DocumentExist() {

return Doc != nullptr;

}

~Editor() = default;

private:

std::shared\_ptr<Document> Doc;

std::stack<std::shared\_ptr<Command>> History;

};

#endif //EDITOR\_H

**command.h**

#ifndef COMMAND\_H

#define COMMAND\_H 1

#include "document.h"

#include <stack>

class Command {

protected:

std::shared\_ptr<Document> Doc;

public:

virtual void Execute() = 0;

virtual void UnExecute() = 0;

virtual ~Command() = default;

void SetDocument(std::shared\_ptr<Document> doc) {

Doc = doc;

}

};

class InsertCommand : public Command {

public:

InsertCommand(FigureType type, Vertex \*vertices) :

Type{type}, Vertices{vertices} {};

void Execute() override {

Doc->InsertPrimitive(Type, Vertices);

}

void UnExecute() override {

Doc->RemoveLastPrimitive();

}

private:

FigureType Type;

Vertex \*Vertices;

};

class RemoveCommand : public Command {

public:

RemoveCommand(int id) : Id(id), Pos(0), figure(nullptr) {};

void Execute() override {

figure = Doc->GetFigure(Id);

if (figure == nullptr) {

std::cout << "ERROR\n";

return;

}

Pos = Doc->GetPos(Id);

if (Pos == 0) {

std::cout << "ERROR\n";

return;

}

Doc->RemovePrimitive(Id);

}

void UnExecute() override {

Doc->InsertPrimitive(Pos, figure);

}

private:

int Id;

int Pos;

std::shared\_ptr<Figure> figure;

};

#endif //COMMAND\_H

**document.h**

#ifndef DOCUMENT\_H

#define DOCUMENT\_H

#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 &newName) {

Name = newName;

}

void Save(const std::string &filename) {

SerialiseImpl(filename);

}

void Load(const std::string &filename) {

DeserializeImpl(filename);

}

void Print() {

std::for\_each(Buffer.begin(), Buffer.end(), [](std::shared\_ptr<Figure> shape) {

shape->Print(std::cout) << "\n";

});

}

void RemovePrimitive(int id) {

auto it = std::find\_if(Buffer.begin(), Buffer.end(),

[id](std::shared\_ptr<Figure> shape) -> bool {

return id == shape->getId();

});

if (it == Buffer.end()) {

throw std::logic\_error("Figure with this id doesn't exist");

}

Buffer.erase(it);

}

void InsertPrimitive(FigureType type, Vertex \*vertices) {

switch (type) {

case Rec:

Buffer.push\_back(factory.FigureCreate(Rec,vertices, Id));

break;

case Rhomb:

Buffer.push\_back(factory.FigureCreate(Rhomb, vertices, Id));

break;

case Trapeze:

Buffer.push\_back(factory.FigureCreate(Trapeze, vertices, Id));

break;

}

Id++;

}

private:

int Id;

std::string Name;

std::list<std::shared\_ptr<Figure>> Buffer;

Factory factory;

/\*TriangleFactory triangleFactory;

SquareFactory squareFactory;

RectangleFactory rectangleFactory;\*/

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 nameLen = Name.size();

os.write((char \*) &nameLen, sizeof(nameLen)); //пишем длину имени(при чтении нужно это)

os.write((char \*) Name.c\_str(), nameLen); //пишем само имя c\_str перевол в массив char имя

for (const auto &shape : 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 nameLen;

is.read((char \*) &nameLen, sizeof(nameLen)); //прочли длину имени

char \*name = new char[nameLen + 1]; //создали имя такой то длины

name[nameLen] = 0;

is.read(name, nameLen);

Name = std::string(name);

delete [] name;

FigureType type;

while (true) {

is.read((char \*) &type, sizeof(type));

if (is.eof()) {

break;

}

switch (type) {

case Rec:

Buffer.push\_back(factory.FigureCreate(Rec));

break;

case Rhomb:

Buffer.push\_back(factory.FigureCreate(Rhomb));

break;

case Trapeze:

Buffer.push\_back(factory.FigureCreate(Trapeze));

break;

}

Buffer.back()->Deserialize(is);

}

Id = Buffer.size();

}

std::shared\_ptr<Figure> GetFigure(int id) {

for (auto it = Buffer.begin(); it != Buffer.end(); it++) {

if (id == (\*it)->getId()) {

return \*it;

}

}

return nullptr;

}

int GetPos(int id) {

int i = 0;

for (auto it = Buffer.begin(); it != Buffer.end(); it++) {

if (id == (\*it)->getId()) {

return i;

}

i++;

}

return -1;

}

void InsertPrimitive(int pos, std::shared\_ptr<Figure> figure) {

auto it = Buffer.begin();

std::advance(it,pos);

Buffer.insert(it,figure);

}

void RemoveLastPrimitive() {

if (Buffer.empty()) {

throw std::logic\_error("Document is empty");

}

Buffer.pop\_back();

}

};

#endif //DOCUMENT\_H

**factory.h**

#ifndef FACTORY\_H

#define FACTORY\_H 1

#include "figures.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 == Trapeze) {

res = std::make\_shared<Trapezoid>();

}

return res;

}

std::shared\_ptr<Figure> FigureCreate(FigureType type, Vertex \*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 == Trapeze) {

res = std::make\_shared<Trapezoid>(vertices[0], vertices[1], vertices[2], vertices[3], id);

}

return res;

}

};

#endif //FACTORY\_H

**figures.h**

#ifndef FIGURES\_H

#define FIGURES\_H 1

#include <iostream>

#include <fstream>

#include <utility> // for pair

#include <memory>

#include <cmath>

#include <stdexcept>

enum FigureType {

Rec,

Rhomb,

Trapeze

};

using Vertex = std::pair<double, double>;

class Figure {

public:

virtual double Area() const = 0;

virtual Vertex 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;

};

Vertex Get\_Center(const Vertex \*vertices, int n) {

double x = 0, y = 0;

for (int i = 0; i < n; i++) {

x += vertices[i].first;

y += vertices[i].second;

}

return std::make\_pair(x / n, y / n);

}

Vertex operator-(const Vertex &p1, const Vertex &p2) {

return {p1.first - p2.first, p1.second - p2.second};

}

bool collinear(const Vertex &a, const Vertex &b, const Vertex &c, const Vertex &d){

return (b.second-a.second)\*(d.first-c.first) - (d.second-c.second)\*(b.first-a.first) <= 1e-9;

}

bool perpendic(const Vertex &a, const Vertex &b, const Vertex &c, const Vertex &d){

using vect = std::pair<double, double>;

vect AC = c-a;

vect BD = d-b;

double dotProduct = AC.first\*BD.first + AC.second\*BD.second;

if(dotProduct <= 1e-9 && dotProduct >= -1e-9) return true;

else return false;

}

double dist(const Vertex &a, const Vertex &b){

return sqrt( ((b.first - a.first) \* (b.first - a.first)) + ((b.second - a.second) \* (b.second - a.second)));

}

bool operator==(const Vertex &a, const Vertex &b){

return (a.first == b.first) && (a.second == b.second);

}

std::ostream& operator<<(std::ostream &o, const Vertex &p){

o << "<" << p.first << ", " << p.second << ">";

return o;

}

bool isNumber(const std::string& s){

return !s.empty() && s.find\_first\_not\_of("-.0123456789") == std::string::npos;

}

std::istream& operator>>(std::istream &is, Vertex &p){

std::string checker;

is >> checker;

if(isNumber(checker) == false){

throw std::overflow\_error("Is not a number");

}

p.first = static\_cast<double>(std::stod(checker));

is >> checker;

if(isNumber(checker) == false){

throw std::overflow\_error("Is not a number");

}

p.second = static\_cast<double>(std::stod(checker));

return is;

}

class Rectangle : public Figure {

int Id;

Vertex \*vertices;

public:

Rectangle() : Id{0}, vertices{new Vertex[4]} {

for (int i = 0; i < 4; i++){

vertices[i] = std::make\_pair(0,0);

}

}

Rectangle(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :

Id{id}, vertices{new Vertex[4]} {

if (a == b || a == c || b == c || a == d ||

!(perpendic(a, b, a, d)) || !collinear(a, d, c, b)

|| !collinear(a, b, d, c)) {

throw std::logic\_error("The entered coordinates of the vertices do not belong to the rectangle.");

} else {

vertices[0] = a;

vertices[1] = b;

vertices[2] = c;

vertices[3] = d;

}

}

~Rectangle() override {

delete [] vertices;

vertices = nullptr;

}

Vertex Center() const override {

return Get\_Center(vertices,4);

}

double Area() const override {

auto AB = dist(vertices[0], vertices[1]);

auto AD = dist(vertices[0],vertices[3]);

return AD\*AB;

}

std::ostream &Print(std::ostream &out) const override{

out << "Id: " << Id << "\n";

out << "Figure: Rectangle\n";

out << "Coords:\n";

for (int i = 0; i < 4; i++) {

out << vertices[i] << "\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 (int 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 (int 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;

}

};

class Trapezoid : public Figure {

int Id;

Vertex \*vertices;

public:

Trapezoid() : Id{0}, vertices{new Vertex[4]} {

for (int i = 0; i < 4; i++){

vertices[i] = std::make\_pair(0,0);

}

}

Trapezoid(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :

Id{id}, vertices{new Vertex[4]} {

auto AB = dist(a, b);

auto AD = dist(a, d);

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 {

vertices[0] = a;

vertices[1] = b;

vertices[2] = c;

vertices[3] = d;

}

}

~Trapezoid() override {

delete [] vertices;

vertices = nullptr;

}

Vertex Center() const override {

return Get\_Center(vertices,4);

}

double Area() const override {

double x1,x2,x3,x4, y1,y2,y3,y4;

x1 = vertices[0].first; y1 = vertices[0].second;

x2 = vertices[1].first; y2 = vertices[1].second;

x3 = vertices[2].first; y3 = vertices[2].second;

x4 = vertices[3].first; y4 = vertices[3].second;

auto area = ( (x1\*y2-x2\*y1)+(x2\*y3-x3\*y2)+(x3\*y4-x4\*y3) ) / 2;

return std::abs(area);

}

std::ostream &Print(std::ostream &out) const override{

out << "Id: " << Id << "\n";

out << "Figure: Trapezoid\n";

out << "Coords:\n";

for (int i = 0; i < 4; i++) {

out << vertices[i] << "\n";

}

return out;

}

void Serialize(std::ofstream &os) const override{

FigureType type = Trapeze;

os.write((char \*) &type, sizeof(type));

os.write((char \*) &Id, sizeof(Id));

for (int 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 (int 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;

}

};

class Rhombus: public Figure {

int Id;

Vertex \*vertices;

public:

Rhombus() : Id{0}, vertices{new Vertex[4]} {

for (int i = 0; i < 4; i++){

vertices[i] = std::make\_pair(0,0);

}

}

Rhombus(Vertex &a, Vertex &b, Vertex &c, Vertex &d, int id) :

Id{id}, vertices{new Vertex[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) ||

!perpendic(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 {

vertices[0] = a;

vertices[1] = b;

vertices[2] = c;

vertices[3] = d;

}

}

~Rhombus() override {

delete [] vertices;

vertices = nullptr;

}

Vertex Center() const override {

return Get\_Center(vertices,4);

}

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 (int 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 (int 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 (int 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;

}

};

#endif //FIGURES\_H

1. **Выводы:**

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

**СПИСОК ЛИТЕРАТУРЫ**

1. Справочник по языку С++ [Электронный ресурс]. URL:

<https://en.cppreference.com/w/> .