Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Проектирование структуры классов.

Студент:	Вахрамян К.О.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	3
Оценка:	
Дата:	

Москва
2019

1.**Код на C++:**
comand.h:

```
#include "figure.h"
#include "point.h"
#include "rectangle.h"
#include "rhombus.h"
#include "trapezoid.h"
#include "document.h"
#include <memory>
#include <vector>
#include <string>


struct TComand {
	std::shared_ptr<TFigure> f;
	std::string comand;
	int index;
	TComand(std::shared_ptr<TFigure> fig, std::string cmd, int idx) {
		comand = cmd;
		index = idx;
		f = std::move(fig);
	}

};
```

figure.h:

```
#ifndef FIGURE_H
#define FIGURE_H

#include <iostream>
#include <cassert>
#include <stdexcept>
#include "point.h"
#include <cmath>
#include <exception>
#include <string>


class TFigure {
public:
	virtual std::string Name() const = 0;
	virtual void Print(std::ostream&) const = 0;
	virtual TPoint Center() const = 0;
	virtual double Square() const = 0;
};
#endif
```

rectangle.h:

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "figure.h"
```

```
class TRectangle : public TFigure {
private:
	TPoint a, b, c, d;
public:
	void Print(std::ostream& os) const override;
	TPoint Center() const override;
	double Square() const override;
	std::string Name() const override;
	TRectangle();
	TRectangle(TPoint p1, TPoint p2, TPoint p3, TPoint p4);
	TRectangle(std::istream& is);
};


#endif // RECTANGLE_H
```

rectangle.cpp:

```
#include "rectangle.h"

TRectangle::TRectangle (const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4) {
	a = p1;
	b = p2;
	c = p3;
	d = p4;
	TPoint ab, ad, cb, cd;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	ad.x = d.x - a.x;
	ad.y = d.y - a.y;
	cb.x = b.x - c.x;
	cb.y = b.y - c.y;
	cd.x = d.x - c.x;
	cd.y = d.y - c.y;
	if (acos((ab.x * ad.x + ab.y * ad.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(ad.x * ad.x +
ad.y * ad.y))) / M_PI != 0.5 || acos((cb.x * cd.x + cb.y * cd.y) / (sqrt(cb.x * cb.x + cb.y * cb.y) *
sqrt(cd.x * cd.x + cd.y * cd.y))) / M_PI != 0.5) {
		throw std::logic_error("it's not rectangle\n");
	}
	//assert(acos((ab.x * ad.x + ab.y * ad.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(ad.x *
ad.x + ad.y * ad.y))) / M_PI == 0.5 && acos((cb.x * cd.x + cb.y * cd.y) / (sqrt(cb.x * cb.x + cb.y
* cb.y) * sqrt(cd.x * cd.x + cd.y * cd.y))) / M_PI == 0.5);
}

TRectangle::TRectangle(std::istream& is) {
	is >> a >> b >> c >> d;
	TPoint ab, ad, cb, cd;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	ad.x = d.x - a.x;
```

```
	ad.y = d.y - a.y;
	cb.x = b.x - c.x;
	cb.y = b.y - c.y;
	cd.x = d.x - c.x;
	cd.y = d.y - c.y;
	if (acos((ab.x * ad.x + ab.y * ad.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(ad.x * ad.x + ad.y * ad.y))) / M_PI != 0.5 || acos((cb.x * cd.x + cb.y * cd.y) / (sqrt(cb.x * cb.x + cb.y * cb.y) * sqrt(cd.x * cd.x + cd.y * cd.y))) / M_PI != 0.5) {
		throw std::logic_error("it's not rectangle\n");
	}
}

double TRectangle::Square () const {

	double ans = (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
	return fabs(ans);
}

TPoint TRectangle::Center() const {
	TPoint p;
	double x = (a.x + b.x + c.x + d.x) / 4;
	double y = (a.y + b.y + c.y + d.y) / 4;
	p.x = x;
	p.y = y;
	return p;
}

void TRectangle::Print(std::ostream& os) const {
	os <<  a << " " << b << " " << c << " " << d << "\n";
}

std::string TRectangle::Name() const{
	return "rectangle ";
}
```

rhombus.h:

```
#ifndef RHOMBUS_H
#define RHOMBUS_H

#include "figure.h"
#include <iostream>

class TRhombus : public TFigure{
private:
	TPoint a, b, c, d;
public:
	std::string name = "rhombus ";
	double Square() const override;
	TPoint Center() const override;
	void Print(std::ostream&) const override;
```

```
        std::string Name() const override;
        TRhombus();
        TRhombus(const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4);
        TRhombus(std::istream& is);
};


#endif
```

rhombus.cpp:

```
#include "rhombus.h"

TRhombus::TRhombus (const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4) {
        a = p1;
        b = p2;
        c = p3;
        d = p4;
        TPoint ab, bc, cd, da;
        ab.x = b.x - a.x;
        ab.y = b.y - a.y;
        bc.x = c.x - b.x;
        bc.y = c.y - b.y;
        cd.x = d.x - c.x;
        cd.y = d.y - c.y;
        da.x = a.x - d.x;
        da.y = a.y - d.y;
        if (sqrt(ab.x * ab.x + ab.y * ab.y) !=  sqrt(bc.x * bc.x + bc.y * bc.y) || sqrt(bc.x * bc.x +
bc.y * bc.y) != sqrt(cd.x * cd.x + cd.y * cd.y) || sqrt(cd.x * cd.x + cd.y * cd.y) != sqrt(da.x * da.x
+ da.y * da.y)) {
                throw std::logic_error("it's not rhombus\n");
        }
        //assert(sqrt(ab.x * ab.x + ab.y * ab.y) ==  sqrt(bc.x * bc.x + bc.y * bc.y) && sqrt(bc.x *
bc.x + bc.y * bc.y) == sqrt(cd.x * cd.x + cd.y * cd.y) && sqrt(cd.x * cd.x + cd.y * cd.y) ==
sqrt(da.x * da.x + da.y * da.y));

}

TRhombus::TRhombus(std::istream& is) {
        is >> a >> b >> c >> d;
        TPoint ab, bc, cd, da;
        ab.x = b.x - a.x;
        ab.y = b.y - a.y;
        bc.x = c.x - b.x;
        bc.y = c.y - b.y;
        cd.x = d.x - c.x;
        cd.y = d.y - c.y;
        da.x = a.x - d.x;
        da.y = a.y - d.y;
```

```
	if (sqrt(ab.x * ab.x + ab.y * ab.y) !=  sqrt(bc.x * bc.x + bc.y * bc.y) || sqrt(bc.x * bc.x + bc.y * bc.y) != sqrt(cd.x * cd.x + cd.y * cd.y) || sqrt(cd.x * cd.x + cd.y * cd.y) != sqrt(da.x * da.x + da.y * da.y)) {
		throw std::logic_error("it's not rhombus\n");
	}
}

double TRhombus::Square() const {
	double ans = 0.5 * sqrt(pow(a.x - c.x, 2) + pow(a.y - c.y, 2)) * sqrt(pow(b.x - d.x, 2) + pow(b.y - d.y, 2));
	return fabs(ans);
}

TPoint TRhombus::Center() const {
	TPoint p;
	double x = (a.x + b.x + c.x + d.x) / 4;
	double y = (a.y + b.y + c.y + d.y) / 4;
	p.x = x;
	p.y = y;
	return p;
}

void TRhombus::Print(std::ostream& os) const {
	os << a << " " << b << " " << c << " " << d << "\n";
}

std::string TRhombus::Name() const{
	return "rhombus ";
}
```

trapezoid.h:

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

#include "figure.h"
#include <cmath>

class TTrapezoid : public TFigure{
private:
	TPoint a, b, c, d;
public:
	std::string name = "trapezoid ";
	double Square() const override;
	TPoint Center() const override;
	void Print(std::ostream&) const override;
	std::string Name() const override;
	TTrapezoid();
	TTrapezoid(const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4);
	TTrapezoid(std::istream& is);
};
```

#endif

trapezoid.cpp:

```
#include "trapezoid.h"

TTrapezoid::TTrapezoid (const TPoint p1, const TPoint p2, const TPoint p3, const TPoint p4) {
	a = p1;
	b = p2;
	c = p3;
	d = p4;
	TPoint ab, ad, bc, dc;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	ad.x = d.x - a.x;
	ad.y = d.y - a.y;
	bc.x = c.x - b.x;
	bc.y = c.y - b.y;
	dc.x = c.x - d.x;
	dc.y = c.y - d.y;
	if (acos((ab.x * dc.x + ab.y * dc.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(dc.x * dc.x + dc.y * dc.y))) != 0 && acos((ad.x * bc.x + ad.y * bc.y) / (sqrt(ad.x * ad.x + ad.y * ad.y) * sqrt(bc.x * bc.x + bc.y * bc.y))) != 0) {
		throw std::logic_error("it's not trapezoid\n");
	}

	//assert(acos((ab.x * dc.x + ab.y * dc.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(dc.x * dc.x + dc.y * dc.y))) == 0 || acos((ad.x * bc.x + ad.y * bc.y) / (sqrt(ad.x * ad.x + ad.y * ad.y) * sqrt(bc.x * bc.x + bc.y * bc.y))) == 0);

}

TTrapezoid::TTrapezoid(std::istream& is) {
	is >> a >> b >> c >> d;
	TPoint ab, ad, bc, dc;
	ab.x = b.x - a.x;
	ab.y = b.y - a.y;
	ad.x = d.x - a.x;
	ad.y = d.y - a.y;
	bc.x = c.x - b.x;
	bc.y = c.y - b.y;
	dc.x = c.x - d.x;
	dc.y = c.y - d.y;
	if (acos((ab.x * dc.x + ab.y * dc.y) / (sqrt(ab.x * ab.x + ab.y * ab.y) * sqrt(dc.x * dc.x + dc.y * dc.y))) != 0 && acos((ad.x * bc.x + ad.y * bc.y) / (sqrt(ad.x * ad.x + ad.y * ad.y) * sqrt(bc.x * bc.x + bc.y * bc.y))) != 0) {
		throw std::logic_error("it's not trapezoid\n");
	}

}
```

```
TPoint TTrapezoid::Center() const {
	TPoint p;
	double x = (a.x + b.x + c.x + d.x) /4;
	double y = (a.y + b.y + c.y + d.y) /4;
	p.x = x;
	p.y = y;

	return p;

}
double TTrapezoid::Square() const {
	TPoint p = this->Center();
	double t1 = 0.5 * fabs((b.x - a.x) * (p.y - a.y) - (p.x - a.x) * (b.y - a.y));
	double t2 = 0.5 * fabs((c.x - b.x) * (p.y - b.y) - (p.x - b.x) * (c.y - b.y));
	double t3 = 0.5 * fabs((d.x - c.x) * (p.y - c.y) - (p.x - c.x) * (d.y - c.y));
	double t4 = 0.5 * fabs((a.x - d.x) * (p.y - d.y) - (p.x - d.x) * (a.y - d.y));
	return t1 + t2 + t3 + t4;
}

void TTrapezoid::Print(std::ostream& os) const {
	os << a << " " << b << " " << c << " " << d << "\n";
}

std::string TTrapezoid::Name() const{
	return "trapezoid ";
}
```

document.h:

```
#ifndef DOCUMENT_H
#define DOCUMENT_H


#include "figure.h"
#include "point.h"
#include "rectangle.h"
#include "rhombus.h"
#include "trapezoid.h"
#include "comand.h"
#include <memory>
#include <vector>
#include <string>



class TDocument {
private:
	std::vector<std::shared_ptr<TFigure>> figures;
	std::vector<std::shared_ptr<TComand>> comands;
public:
	TDocument() {}
```

```
        void Save(std::ostream& os);
        void Load(std::istream& is);
        void Print(std::ostream& os);
        void Add(std::shared_ptr<TFigure> f, int idx);
        void Delete(int idx);
        void Undo();
        void AddCmd(TComand cmd);
};


#endif
```

document.cpp:

```
#include "document.h"

void TDocument::Save(std::ostream& os) {
        for (auto &tmp : figures) {
                os << tmp->Name();
                tmp->Print(os);
        }
}

void TDocument::Add(std::shared_ptr<TFigure> f, int idx) {
        std::shared_ptr<TComand> tmp(new TComand(f, "insert", idx));
        figures.insert(figures.begin() + idx, std::move(f));
        comands.push_back(std::move(tmp));
}

void TDocument::Delete(int idx) {
        std::shared_ptr<TComand> tmp(new TComand(figures[idx], "delete", idx));
        figures.erase(figures.begin() + idx);
        comands.push_back(std::move(tmp));

}

void TDocument::Print(std::ostream& os) {
        for (auto &tmp : figures) {
                std::cout << tmp->Name();
                tmp->Print(os);
                std::cout << "Square: " << tmp->Square() << std::endl;
                std::cout << "Center: " << tmp->Center() << std::endl;
        }
}

void TDocument::Load(std::istream& is) {
        std::string name;
        while(is >> name)
        if (name == "rectangle") {
                std::shared_ptr<TFigure> f(new TRectangle(is));
                figures.push_back(std::move(f));
        } else if (name == "rhombus") {
```

```
                std::shared_ptr<TFigure> f(new TRhombus(is));
                figures.push_back(std::move(f));
        } else if (name == "trapezoid") {
                std::shared_ptr<TFigure> f(new TTrapezoid(is));
                figures.push_back(std::move(f));
        }
}

void TDocument::Undo() {
        if (comands.size() == 0) {
                throw std::logic_error("list of comands is empty\n");
        }
        if (comands.back()->comand == "delete") {
                figures.insert(figures.begin() + comands.back()->index,
std::move(comands.back()->f));
                comands.pop_back();

        } else if (comands.back()->comand == "insert") {
                figures.erase(figures.begin() + comands.back()->index);
                comands.pop_back();
        }
}
```

comand.h:

```
#include "figure.h"
#include "point.h"
#include "rectangle.h"
#include "rhombus.h"
#include "trapezoid.h"
#include "document.h"
#include <memory>
#include <vector>
#include <string>


struct TComand {
        std::shared_ptr<TFigure> f;
        std::string comand;
        int index;
        TComand(std::shared_ptr<TFigure> fig, std::string cmd, int idx) {
                comand = cmd;
                index = idx;
                f = std::move(fig);
        }

};
```

main.cpp:

```
#include "rectangle.h"
```

```
#include "rhombus.h"
#include "trapezoid.h"
#include "document.h"
#include <vector>
#include <memory>
#include <algorithm>
#include <string>
#include <fstream>

int main() {
	std::string cmd;
	std::cout << "new - to create new document\n"
			<< "insert idx- to insert figure to document to position idx\n"
			<< "delete idx- to delete figure on position idx from document\n"
			<< "undo - to cancel last comand\n"
			<< "save - to save document in file\n"
			<< "load - to load document from file\n"
			<< "print - to print all figure\n"
			<< "help - to show this page\n"
			<< "exit - to finish execution of program\n";
	while (true) {
		std::cin >> cmd;
		if (cmd == "new") {
			TDocument doc;
			while (true) {
				std::cin >> cmd;
				if (cmd == "insert") {
				int i;
				std::cin >> i;
				int chose;
				std::cout << "1 - rectangle\n2 - rhombus\n3 - trapezoid\n";
				std::cin >> chose;
				std::shared_ptr<TFigure> f;
				if (chose == 1) {
					try {
						f =
std::make_shared<TRectangle>(TRectangle(std::cin));
					} catch (std::exception& err) {
						std::cout << err.what();
						continue;
					}

				} else if (chose == 2) {
					try {
						f =
std::make_shared<TRhombus>(TRhombus(std::cin));
					} catch (std::exception& err) {
						std::cout << err.what();
						continue;
					}
				} else if (chose == 3) {
					try {
```

```
                                        f =
std::make_shared<TTrapezoid>(TTrapezoid(std::cin));
                                } catch(std::exception& err) {
                                        std::cout << err.what();
                                        continue;
                                }
                        } else {
                                std::cout << "error, try again\n";
                                continue;
                        }
                                doc.Add(f, i);
                        } else if (cmd == "delete") {
                                int i;
                                std::cin >> i;
                                doc.Delete(i);
                        } else if (cmd == "save") {
                                std::string path;
                                std::cin >> path;
                                std::ofstream os(path);
                                doc.Save(os);
                        } else if (cmd == "load") {
                                std::string path;
                                std::cin >> path;
                                std::ifstream is(path);
                                doc.Load(is);
                        } else if (cmd == "print") {
                                doc.Print(std::cout);
                        } else if (cmd == "undo") {
                                try {
                                        doc.Undo();
                                } catch (std::exception& err) {
                                        std::cout << err.what();
                                }

                        } else if (cmd == "help") {
                                std::cout << "new - to create new document\n"
                                                << "insert idx- to insert figure to document
to position idx\n"
                                                << "delete idx- to delete figure on position

idx from document\n"
                                                << "undo - to cancel last comand\n"
                                                << "save - to save document in file\n"
                                                << "load - to load document from file\n"
                                                << "print - to print all figure\n"
                                                << "help - to show this page\n"
                                                << "exit - to finish execution of program\n";
                                                continue;
                        } else if (cmd == "exit") {
                                return 0;
                        } else {
                                std::cout << "wrong comand, enter 'help' to show man\n";
                        }
```

```
                }
            } else if (cmd == "exit") {
                break;
            } else {
                std::cout << "firstly create new document\n";
                continue;
            }

        }
}
```

2. **Ссылка на репозиторий в GitHub:**
https://github.com/vebcreatex7/oop_exercise_07

3.**Результаты выполнения программы:**

```
new - to create new document
insert idx- to insert figure to document to position idx
delete idx- to delete figure on position idx from document
undo - to cancel last comand
save - to save document in file
load - to load document from file
print - to print all figure
help - to show this page
exit - to finish execution of program
new
insert 0
1 - rectangle
2 - rhombus
3 - trapezoid
1
0 0 0 1 1 1 1 0
insert
0
1 - rectangle
2 - rhombus
3 - trapezoid
3
0 0 3 3 6 3 9 0
print
trapezoid 0 0 3 3 6 3 9 0
Square: 18
Center: 4.5 1.5
rectangle 0 0 0 1 1 1 1 0
Square: 1
Center: 0.5 0.5
save ../in.txt
```

print
trapezoid 0 0 3 3 6 3 9 0
Square: 18
Center: 4.5 1.5
rectangle 0 0 0 1 1 1 1 0
Square: 1
Center: 0.5 0.5
undo
print
rectangle 0 0 0 1 1 1 1 0
Square: 1
Center: 0.5 0.5
undo
print
undo
list of comands is empty
load ../in.txt
print
trapezoid 0 0 3 3 6 3 9 0
Square: 18
Center: 4.5 1.5
rectangle 0 0 0 1 1 1 1 0
Square: 1
Center: 0.5 0.5
undo
list of comands is empty
ls
wrong comand, enter 'help' to show man
print
trapezoid 0 0 3 3 6 3 9 0
Square: 18
Center: 4.5 1.5
rectangle 0 0 0 1 1 1 1 0
Square: 1
Center: 0.5 0.5
delete 0
print
rectangle 0 0 0 1 1 1 1 0
Square: 1
Center: 0.5 0.5
undo
print
trapezoid 0 0 3 3 6 3 9 0
Square: 18
Center: 4.5 1.5
rectangle 0 0 0 1 1 1 1 0
Square: 1
Center: 0.5 0.5
exit

4. Объяснение результатов работы программы:
Пользователь вводит команды с терминала. В программе реализованы функции создания нового документа, чтение и запись в файл. Команда undo отменяет последнее добавление или удаление фигуры.

5.**Вывод:**
В данной лабораторной работе реализован примитивный текстовый редактор. Проведена работа с чтением и записью в файл. Также осознано и реализовано то, как работает отмена последнего действия при помощи команды undo.