**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: **Наследование**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7382 |  | Гаврилов А.В. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2019

**Цель работы.**

Ознакомиться с понятиями наследование, полиморфизм, абстрактный класс, изучить виртуальные функции, принцип их работы, способ организации в памяти, раннее и позднее связывания в языке C++. В соответствии с индивидуальным заданием разработать систему классов для представления геометрических фигур.

**Постановка задачи.**

Необходимо спроектировать систему классов для моделирования геометрических фигур (в соответствии с полученным индивидуальным заданием). Задание предполагает использование виртуальных функций в иерархии наследования, проектирование и использование абстрактного базового класса.  Разработанные классы должны быть наследниками абстрактного класса Shape, содержащего методы для перемещения в указанные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, установки и получения цвета, а также оператор вывода в поток.  
﻿﻿Необходимо также обеспечить однозначную идентификацию каждого объекта.

Решение должно содержать:

* условие задания;
* UML диаграмму разработанных классов;
* текстовое обоснование проектных решений;
* реализацию классов на языке С++.

Задание варианта: реализовать классы фигур – квадрата, трапеции, равнобедренной трапеции.

**Ход работы.**

Построена UML диаграмма классов, представленная на рис. 1.

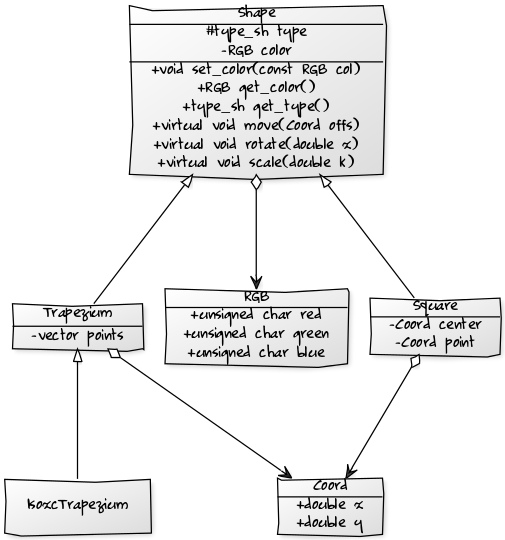


Рисунок 1-UML диаграмма классов

Код программы представлен в приложении А.

Программа состоит из 5 классов и 1 структуры.

Структура RGB задает цвет.

Класс Coord хранит координаты вектора или точки и поддерживает некоторые операции с ним.

Square – класс квадрата, который описывает квадрат через его центр и точку одного из углов.

Trapezium – класс, который хранит трапецию через 4 точки.

IsoxcTrapezium –класс, который хранит равнобедренную трапецию по 3 точкам.

**Выводы.**

В результате работы была разработана иерархия классов, которая необходима для реализации геометрических фигур в соответствии с заданием. Перед реализацией программы была составлена UML диаграмма классов этой иерархии. Программа может описать фигуру и работать с ней в 2-х мерном пространстве.

**ПРИЛОЖЕНИЕ А**

**Исходный код**

#include <iostream>

#include <cmath>

#include <vector>

enum type\_sh{SHAPE,SQUARE, TRAPEZIUM, ISOXC\_TRAPEZIUM};

struct RGB{

unsigned char red;

unsigned char green;

unsigned char blue;

RGB():red(0),green(0),blue(0){};

};

class Coord{

public:

double x;

double y;

Coord(){

x=0;

y=0;

}

Coord(double x, double y){

this->x=x;

this->y=y;

}

Coord& operator +=(const Coord& add){

this->x+=add.x;

this->y+=add.y;

return \*this;

}

Coord operator +(const Coord& b){

Coord res;

res.x=this->x+b.x;

res.y=this->y+b.y;

return res;

}

Coord operator +(double c){

this->x+=c;

this->y+=c;

}

Coord operator -(const Coord& b){

Coord res;

res.x=this->x-b.x;

res.y=this->y-b.y;

return res;

}

Coord operator \*(const double k){

Coord res;

res.x= this->x\*k;

res.y=this->y\*k;

return res;

}

double operator \*(const Coord& b){

return this->x\*b.x+this->y\*b.y;

}

double mod(){

return sqrt(x\*x+y\*y);

}

};

class Shape{

public:

void set\_color(const RGB col){

color.red=col.red;

color.green=col.green;

color.blue=col.blue;

type=SHAPE;

}

RGB get\_color(){

return color;

}

type\_sh get\_type(){

return type;

}

virtual void move(Coord offs)=0;

virtual void rotate(double x)=0;

virtual void scale(double k)=0;

private:

RGB color;

protected:

type\_sh type;

};

class Square: public Shape{

private:

Coord center;

Coord point;

public:

Square(){}

Square(Coord cent, Coord pnt):center(cent),point(pnt){

type=SQUARE;

}

void move(Coord offs) override{

point+=offs;

center+=offs;

}

void scale(double k) override{

point=(point-center)\*k+center;

}

void rotate(double x){

Coord vec, new\_vec;

vec=point-center;

new\_vec.x=vec.x\*cos(x)-vec.y\*sin(x)+center.x;

new\_vec.y=vec.x\*sin(x)+vec.y\*cos(x)+center.y;

point.x=round(new\_vec.x\*100000)/100000;

point.y=round(new\_vec.y\*100000)/100000;

}

friend std::ostream& operator <<(std::ostream& stream,Square& sq){

stream<< "Coordinates of square points:"<<std::endl;

for(int i=1;i<=4;i++){

stream<<i<<") coordinate x: "<<sq.point.x<<", coordinate y: "<<sq.point.y<<std::endl;

sq.rotate(M\_PI/2);

}

}

};

class Trapezium: public Shape{

protected:

std::vector<Coord> points;

public:

Trapezium(){

points.push\_back(Coord(0,0));

points.push\_back(Coord(3,0));

points.push\_back(Coord(1,1));

points.push\_back(Coord(2,1));

type=TRAPEZIUM;

}

Trapezium(Coord lmbp, Coord rmbp, Coord lsbp, Coord rsbp){

try{

Coord fst\_vec=rmbp-lmbp;

Coord sec\_vec=rsbp-lsbp;

if(fabs(fst\_vec\*sec\_vec/(fst\_vec.mod()\*sec\_vec.mod())-1)>0.1)

throw 1;

}

catch(int err){

std::cout<<"Данные координаты точек не соответствуют трапеции!"<<std::endl;

throw;

}

points.push\_back(lmbp);

points.push\_back(rmbp);

points.push\_back(lsbp);

points.push\_back(rsbp);

type=TRAPEZIUM;

}

void rotate(double x)override{

Coord center;

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

center+=points[i];}

center=center\*(0.25);

for(int i=0; i<points.size(); i++){

Coord vec, new\_vec;

vec=points[i]-center;

new\_vec.x=vec.x\*cos(x)-vec.y\*sin(x)+center.x;

new\_vec.y=vec.x\*sin(x)+vec.y\*cos(x)+center.y;

points[i].x=round(new\_vec.x\*100000)/100000;

points[i].y=round(new\_vec.y\*100000)/100000;

}

}

void scale(double k) override{

Coord center;

for(int i=0;i<points.size();i++){

center+=points[i];

}

center=center\*(0.25);

for(int i=0; i<points.size(); i++){

points[i]=(points[i]-center)\*k+center;

}

}

void move(Coord offs) override{

for(int i=0; i<points.size(); i++)

points[i]+=offs;

}

friend std::ostream& operator <<(std::ostream& stream,const Trapezium& tr){

stream <<"Coordinates of trapezium points:"<<std::endl;

for(int i=0;i<tr.points.size();i++){

stream<<i+1<<") coordinate x: " << tr.points[i].x <<", coordinate y: "<<tr.points[i].y<<std::endl;

}

return stream;

}

};

class IsoxcTrapezium: public Trapezium

{

public:

IsoxcTrapezium(){

points.push\_back(Coord(0,0));

points.push\_back(Coord(3,0));

points.push\_back(Coord(1,1));

type=ISOXC\_TRAPEZIUM;

}

IsoxcTrapezium(Coord lmbp, Coord rmbp, Coord lsbp, Coord rsbp){

Coord fst\_vec=rmbp-lmbp;

Coord sec\_vec=rsbp-lsbp;

try{

if(fabs(fst\_vec\*sec\_vec/(fst\_vec.mod()\*sec\_vec.mod())-1)>0.00001)

throw 1;

fst\_vec=lsbp-lmbp;

sec\_vec=rsbp-rmbp;

if(fst\_vec.mod()-sec\_vec.mod()>0.00001)

throw 2;

}

catch(int err){

std::cout<<"Данные координаты точек не соответствуют равнобедренной трапеции!"<<std::endl;

throw;

}

points.clear();

points.push\_back(lmbp);

points.push\_back(rmbp);

points.push\_back(lsbp);

}

void scale(double k) override{

Coord center,fst\_vec,sec\_vec;

for(int i=0;i<points.size();i++){

center+=points[i];

}

fst\_vec=points[2]-points[0];

sec\_vec=points[1]-points[0];

double angle\_cos=fst\_vec\*sec\_vec/(fst\_vec.mod()\*sec\_vec.mod());

double small\_base\_length=sec\_vec.mod()-2\*fst\_vec.mod()\*angle\_cos;

center+=points[2]+(sec\_vec\*(small\_base\_length/sec\_vec.mod()));

center=center\*(0.25);

for(int i=0; i<points.size(); i++){

points[i]=(points[i]-center)\*k+center;

}

}

void rotate(double x)override{

Coord center,fst\_vec,sec\_vec;

for(int i=0;i<points.size();i++){

center+=points[i];

}

fst\_vec=points[2]-points[0];

sec\_vec=points[1]-points[0];

double angle\_cos=fst\_vec\*sec\_vec/(fst\_vec.mod()\*sec\_vec.mod());

double small\_base\_length=sec\_vec.mod()-2\*fst\_vec.mod()\*angle\_cos;

center+=points[2]+(sec\_vec\*(small\_base\_length/sec\_vec.mod()));

center=center\*(0.25);

for(int i=0; i<points.size(); i++){

Coord vec, new\_vec;

vec=points[i]-center;

new\_vec.x=vec.x\*cos(x)-vec.y\*sin(x)+center.x;

new\_vec.y=vec.x\*sin(x)+vec.y\*cos(x)+center.y;

points[i].x=round(new\_vec.x\*100000)/100000;

points[i].y=round(new\_vec.y\*100000)/100000;

}

}

friend std::ostream& operator <<(std::ostream& stream,IsoxcTrapezium& tr){

Coord fst\_vec=tr.points[2]-tr.points[0];

Coord sec\_vec=tr.points[1]-tr.points[0];

double angle\_cos=fst\_vec\*sec\_vec/(fst\_vec.mod()\*sec\_vec.mod());

double small\_base\_length=sec\_vec.mod()-2\*fst\_vec.mod()\*angle\_cos;

stream <<"Coordinates of trapezium points:"<<std::endl;

for(int i=0;i<tr.points.size();i++){

stream<< i+1 <<") coordinate x: " << tr.points[i].x <<", coordinate y: "<<tr.points[i].y<<std::endl;

}

Coord th=tr.points[2]+(sec\_vec\*(small\_base\_length/sec\_vec.mod()));

stream<<"4) coordinate x: " << th.x <<", coordinate y: "<<th.y<<std::endl;

return stream;

}

};

int main(){

Square a({4,4},{5,2.5});

a.move({1,1});

a.scale(2);

a.rotate(M\_PI\*2);

std:: cout<<a;

Trapezium b(Coord(1,1),Coord(4,2),Coord(1,2),Coord(3,2));

b.move({1,1});

b.scale(2);

b.rotate(2\*M\_PI);

std::cout<<b;

IsoxcTrapezium c({0,0},{3,0},{1,1},{2,1});

c.move({1,1});

c.scale(2);

c.rotate(2\*M\_PI);

std::cout<<c;

return 0;

}