**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практической работе №2**

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

**Тема**: **Наследование**

|  |  |  |
| --- | --- | --- |
| Студент гр. 7382 |  | Бахеров Д.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++.

Вариант №1. Фигуры : квадрат, эллипс, правильный пятиугольник.

**Текстовое обоснование разработанных классов.**

struct Point – структура для хранения значений точки.

class Colour – класс. Храненит значения цвета фигуры и методы для его задания и считывания.

class Shape – базовый класс. Он содержит основные методы для геометрических фигур:

* void move(double x,double y)- перемещение фигуры на указанное расстояние.
* void turn(double a) – поворот фигуры на заданный угол.
* void scalling(double k) – масштабирование фигуры на заданный коэффициент.

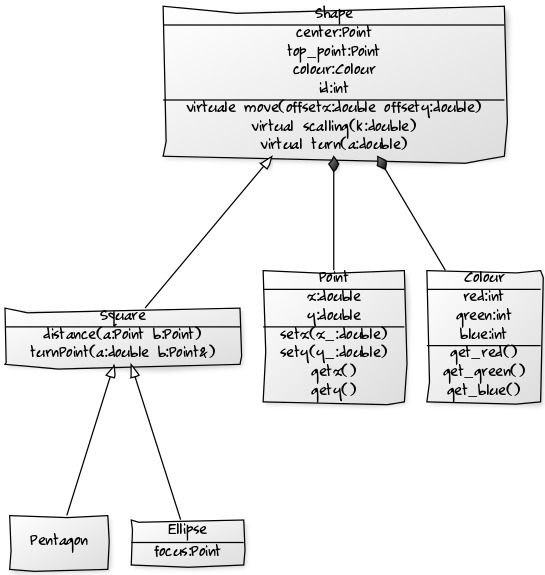
Классы Square, Ellipse и Pentagon – классы наследуемые от класса Shape.

**Выводы.**

В ходе выполнения лабораторной работы была спроектирована система классов для работы с геометрическими фигурами в соответствии с индивидуальным заданием. В иерархии наследования были использованы виртуальные функции, базовый класс при этом является виртуальным (класс называется виртуальным, если содержит хотя бы одну виртуальную функцию). Были реализованы методы перемещения фигуры в заданные координаты, поворота на заданный угол, масштабирования на заданный коэффициент, была реализована однозначная идентификация объекта.

**ПРИЛОЖЕНИЕ А**

UML диаграмма разработанных классов

****

**ПРИЛОЖЕНИЕ Б**

Исходный код программы

#include <iostream>

#include <cmath>

#include <string>

class Colour

{

private:

int red;

int green;

int blue;

public:

Colour(int \_red=0, int \_green=0, int \_blue=0) : red(\_red), green(\_green), blue(\_blue)

{}

int get\_red()

{

return red;

}

int get\_green()

{

return green;

}

int get\_blue()

{

return blue;

}

};

class Point

{

private:

double x, y;

public:

Point(double \_x=0, double \_y=0) : x(\_x), y(\_y)

{}

void setx(double \_x)

{

x=\_x;

}

void sety(double \_y)

{

y=\_y;

}

double getx()

{

return x;

}

double gety()

{

return y;

}

};

class Shape

{

protected:

static int \_id;

Point center;

Point top\_point;

Colour colour;

public:

Shape(Colour clr) : colour(clr)

{

id = \_id;

\_id++;

}

int id;

virtual void move(double, double) = 0;

virtual void scalling(double) = 0;

virtual void turn(double) = 0;

friend std::ostream& operator<<(std::ostream& stream, Shape& object);

};

std::ostream& operator<<(std::ostream& stream, Shape& object)

{

stream << "Figure id(Square):" << object.id << std::endl

<< "Colour: R - " << object.colour.get\_red() << ", G - " << object.colour.get\_green() << ", B - " << object.colour.get\_blue() << std::endl

<< "Center: (" << object.center.getx() << ", "<<object.center.gety() << ")" << std::endl

<< "Top1: (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*2);

stream << "Top2: (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*2);

stream << "Top3: (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*2);

stream << "Top4: (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*2);

return stream;

}

class Square : public Shape

{

protected:

double Square\_sqr;

public:

double distance(Point a, Point b)

{

return sqrt((a.getx() - b.getx()) \* (a.getx() - b.getx()) + (a.gety() - b.gety()) \* (a.gety() - b.gety()));

}

Square(Colour color,Point centr={0,0}, Point anypoint={0,1}) : Shape(color)

{

center = centr;

top\_point = anypoint;

}

void move(double offsetx, double offsety)

{

center.setx(center.getx() + offsetx);

center.sety(center.gety() + offsety);

top\_point.setx(top\_point.getx() + offsetx);

top\_point.sety(top\_point.gety() + offsety);

}

double Sqr()

{

return distance(center, top\_point)\*distance(center, top\_point)\*2;

}

void scalling(double k)

{

top\_point.setx(center.getx()+k\*(top\_point.getx()-center.getx()));

top\_point.sety(center.gety()+k\*(top\_point.gety()-center.gety()));

}

void turnPoint(double a,Point &b)

{

double radian;

if(b.gety()==center.gety())

radian=atan(1)\*2\*(b.getx()-center.getx()>0 ? 1 : -1);

else

{

radian=atan((b.getx()-center.getx())/(b.gety()-center.gety()));

if(b.gety()-center.gety()<0)

radian=radian+4\*atan(1);

}

double line=distance(b,center);

b.setx(center.getx()+line\*sin(radian-a));

b.sety(center.gety()+line\*cos(radian-a));

}

void turn(double a)

{

turnPoint(a,top\_point);

}

};

class Ellipse : public Square

{

protected:

Point focus;

double Ellipse\_sqr;

public:

friend std::ostream& operator<<(std::ostream& stream, Ellipse& object);

Ellipse(Colour clrr, Point cen, Point foc, Point top): Square(clrr,cen,top)

{

Point secfoc(center.getx()-focus.getx(),center.gety()-focus.gety());

if(distance(foc,top)+distance(secfoc,top)>distance(foc,secfoc))

focus=foc;

else

{

center={0,0};

top\_point={0,1};

focus.setx(0);

focus.sety(0);

}

}

void scalling(double k)

{

Square::scalling(k);

focus.setx(center.getx()+k\*(focus.getx()-center.getx()));

focus.sety(center.gety()+k\*(focus.gety()-center.gety()));

}

void move(double offsetx,double offsety)

{

Square::move(offsetx,offsety);

focus.setx(focus.getx() + offsetx);

focus.sety(focus.gety() + offsety);

}

void turn(double a)

{

Square::turn(a);

turnPoint(a,focus);

}

};

std::ostream& operator<<(std::ostream& stream, Ellipse& object)

{

stream << "Figure id(Ellipse):" << object.id << std::endl

<< "Colour: R - " << object.colour.get\_red() << ", G - " << object.colour.get\_green() << ", B - " << object.colour.get\_blue() << std::endl

<< "Centre: (" << object.center.getx() << ", "<<object.center.gety() << ")" << std::endl

<< "Top: (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl

<< "Focus: (" << object.focus.getx() << "," << object.focus.gety() << ")" << std::endl;

return stream;

}

class Pentagon : public Square

{

protected:

double Pentagon\_sqr;

public:

Pentagon(Colour clor,Point cent, Point anypoin) : Square(clor, cent, anypoin)

{}

double Sqr()

{

Point a=top\_point;

Square::turn(atan(1)\*8/5);

double b=distance(a, top\_point);

return 5\*b\*b/4\*tan(atan(1)\*8/10);

}

friend std::ostream& operator<<(std::ostream& stream, Pentagon& object);

};

std::ostream& operator<<(std::ostream& stream, Pentagon& object)

{

stream << "Figure id(Pentagon):" << object.id << std::endl

<< "Colour: R - " << object.colour.get\_red() << ", G - " << object.colour.get\_green() << ", B - " << object.colour.get\_blue() << std::endl

<< "Center: (" << object.center.getx() << ", "<<object.center.gety() << ")" << std::endl

<< "Top1 : (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*8/5);

stream << "Top2 : (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*8/5);

stream << "Top3 : (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*8/5);

stream << "Top4 : (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*8/5);

stream << "Top5 : (" << object.top\_point.getx() << ", " << object.top\_point.gety() << ")" << std::endl;

object.turn(atan(1)\*8/5);

return stream;

}

int Shape::\_id=0;

int main(){

Square a({1,1,1},{0,0}, {1,2});

std::cout << a.Sqr() << std::endl;

Ellipse b({1,1,2},{0,0}, {0,1},{0,2});

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

std::cout << c.Sqr() << std::endl;

std::cout << a << b << c << std::endl;

return 0;

}