**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 |  | Дрозд А.С. |
| 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 №10. Фигуры : круг, эллипс, сектор эллипса.

**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 resize(double k) – масштабирование фигуры на заданный коэффициент.

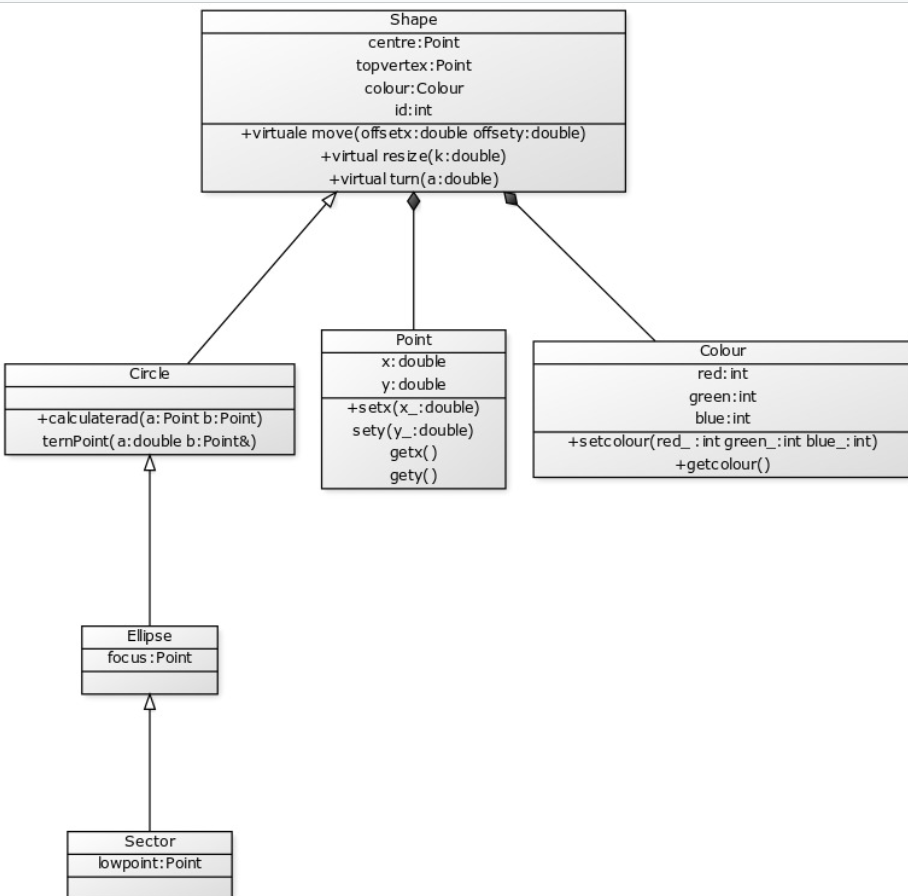
Классы Circle, Ellipse и Sector – классы наследуемые от класса Shape.

**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>

#define STANDART\_CENTRE 0

#define STANDART\_TOP 1

class Colour

{

private:

int red;

int green;

int blue;

public:

Colour()

{

red=0;

green=0;

blue=0;

}

void setcolour(int red\_,int green\_,int blue\_)

{

red=red\_%255;

green=green\_%255;

blue=blue\_%255;

}

int getcolour()

{

return red\*255\*255+green\*255+blue;

}

};

//одна точка

class Point

{

private:

double x, y;

public:

Point()

{

x = 0;

y = 0;

}

Point(double \_x, double \_y)

{

x = \_x;

y = \_y;

}

void setx(double \_x)

{

x = \_x;

}

void sety(double \_y)

{

y = \_y;

}

double getx()

{

return x;

}

double gety()

{

return y;

}

};

//круг

//элипс

//сектор элипса

class Shape

{

protected:

static int general\_id;

public:

Shape()

{

id = general\_id;

general\_id++;

}

int id;

Point centre;

Point topvertex;

Colour colour;

virtual void move(double, double) = 0;

virtual void resize(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:" << object.id << std::endl

<< "Color:red - " << object.colour.getcolour()/(255\*255) << ",green - " << (object.colour.getcolour()%(255\*255))/255 << ",blue - " << object.colour.getcolour()%255 << std::endl

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

<< "toppoint: (" << object.topvertex.getx() << ", " << object.topvertex.gety() << ")" << std::endl;

return stream;

}

class Circle : public Shape

{

public:

double calculaterad(Point a, Point b)

{

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

}

void nana()

{

centre.setx(10);

}

Circle()

{

centre.setx(STANDART\_CENTRE);

centre.sety(STANDART\_CENTRE);

topvertex.setx(STANDART\_CENTRE);

topvertex.sety(STANDART\_TOP);

}

Circle(Point centr, Point anypoint)

{

centre = centr;

topvertex = anypoint;

}

void move(double offsetx, double offsety) override

{

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

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

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

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

}

void resize(double k) override

{

topvertex.setx(centre.getx()+k\*(topvertex.getx()-centre.getx()));

topvertex.sety(centre.gety()+k\*(topvertex.gety()-centre.gety()));

}

void turnPoint(double a,Point &b)

{

double radian;

if(b.gety()==centre.gety())

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

else

{

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

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

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

}

double radius=calculaterad(b,centre);

b.setx(centre.getx()+radius\*sin(radian-a));

b.sety(centre.gety()+radius\*cos(radian-a));

}

void turn(double a) override

{

turnPoint(a,topvertex);

}

};

class Ellipse : public Circle

{

protected:

Point focus;

public:

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

Ellipse()

{

Circle();

focus.setx(STANDART\_CENTRE);

focus.sety(STANDART\_CENTRE);

}

void nana()

{

centre.setx(20);

}

Ellipse(Point cen, Point foc, Point top): Circle(cen,top)

{

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

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

focus=foc;

else

{

centre={STANDART\_CENTRE,STANDART\_CENTRE};

topvertex={STANDART\_CENTRE,STANDART\_TOP};

}

}

void resize(double k) override

{

Circle::resize(k);

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

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

}

void move(double offsetx,double offsety) override

{

Circle::move(offsetx,offsety);

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

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

}

void turn(double a) override

{

Circle::turn(a);

turnPoint(a,focus);

}

};

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

{

stream << "figure id:" << object.id << std::endl

<< "Color:red - " << object.colour.getcolour()/(255\*255) << ",green - " << (object.colour.getcolour()%(255\*255))/255 << ",blue - " << object.colour.getcolour()%255 << std::endl

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

<< "toppoint: (" << object.topvertex.getx() << ", " << object.topvertex.gety() << ")" << std::endl

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

return stream;

}

class Sector : public Ellipse

{

public:

Sector()

{

Ellipse();

lowpoint.setx(STANDART\_CENTRE);

lowpoint.sety(STANDART\_TOP);

}

Sector(Point cen, Point foc, Point top,Point low):Ellipse(cen,foc,top)

{

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

if(calculaterad(foc,top)+calculaterad(secfoc,top)>calculaterad(foc,secfoc) &&

calculaterad(foc,top)+calculaterad(secfoc,top)==calculaterad(foc,low)+calculaterad(secfoc,low))

lowpoint=low;

else

{

cen={STANDART\_CENTRE,STANDART\_CENTRE};

focus={STANDART\_CENTRE,STANDART\_CENTRE};

topvertex={STANDART\_CENTRE,STANDART\_TOP};

}

}

void resize(double k) override

{

Ellipse::resize(k);

lowpoint.setx(centre.getx()+k\*(lowpoint.getx()-centre.getx()));

lowpoint.sety(centre.gety()+k\*(lowpoint.gety()-centre.gety()));

}

void move(double offsetx,double offsety) override

{

Ellipse::move(offsetx,offsety);

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

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

}

void turn(double a) override

{

Ellipse::turn(a);

turnPoint(a,lowpoint);

}

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

protected:

Point lowpoint;

};

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

{

stream << "figure id:" << object.id << std::endl

<< "Color:red - " << object.colour.getcolour()/(255\*255) << ",green - " << (object.colour.getcolour()%(255\*255))/255 << ",blue - " << object.colour.getcolour()%255 << std::endl

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

<< "toppoint: (" << object.topvertex.getx() << ", " << object.topvertex.gety() << ")" << std::endl

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

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

return stream;

}

int Shape::general\_id=0;

int main(int argc, char const \*argv[])

{

Circle circle({0,0}, {1,2});

std::cout << circle << std::endl;

circle.colour.setcolour(12,9,234);

circle.turn(atan(1)\*4);

std::cout << circle << std::endl << std::endl;

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

ellipse.colour.setcolour(0,254,255);

std::cout << ellipse << std::endl;

circle.nana();

std::cout << "!" << circle << std::endl << std::endl;

ellipse.Circle::nana();

std::cout << ellipse << std::endl;

Sector sixstar({0,0},{1,0},{0,1},{0,-1});

std::cout << sixstar << std::endl;

sixstar.colour.setcolour(0,0,254);

sixstar.move( -1, 6);

std::cout << sixstar << std::endl;

sixstar.turn(4\*atan(1));

std::cout << sixstar << std::endl;

return 0;

}