**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №2**

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

Тема: Наследование

|  |  |  |
| --- | --- | --- |
| Студент гр. 7381 |  | Дорох С.В. |
| Преподаватель |  | Жангиров Т. Р. |

Санкт-Петербург

2019

**Цель работы**

Изучение принципов наследования классов в С++.

**Задание**

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++.

**Условие задания**

Вариант 7: фигуры, которые необходимо реализовать: квадрат, параллелограмм, ромб.

**Обоснование проектных решений**

В абстрактном классе Shape, который является базовым, находятся общие для всех фигур поля: координаты центра фигуры, цвета, угол, между основанием и осью абсцисс, а также идентификатор объектов. В классе Shape были реализованы функции-методы для перемещения, установки и получения цветов, а также две виртуальные функции масштабирования и поворота фигуры, которые переопределялись в других классах по необходимости, была перегружена операция вывода в поток.

Производными классами от класса Shape являются Circle и Trapeze. В класс Circle было добавлено поле, хранящее значение радиуса круга, в класс Trapeze были добавлены следующие поля: длины оснований, высота трапеции, а также координаты медиан оснований трапеции. Класс CircleSeg, описывающий сектор круга был производным от Circle, так как сектор круга представляет из себя часть фигуры, для описания сектора в класс CircleSeg, были добавлены два поля, хранящие значения углов от оси абсцисс до границ сектора.

**UML-диаграмма**

Ниже представлена UML-диаграмма, для отображения организации структур и классов.

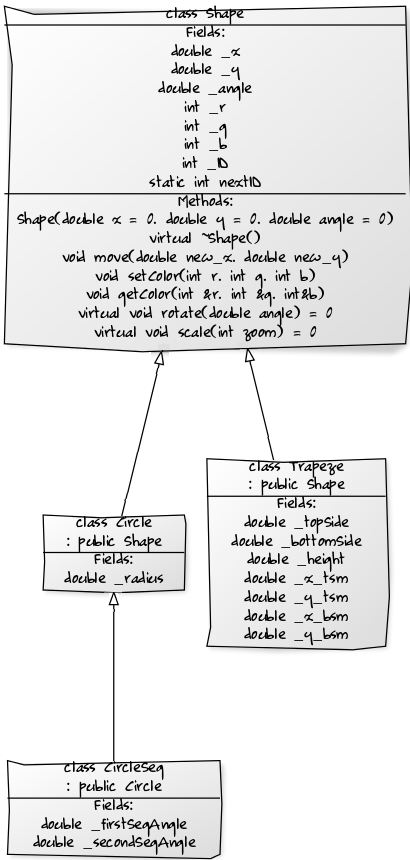


Рисунок 1 - UML-диаграмма данного проекта.

**Выводы.**

В ходе лабораторной работы была реализована система классов для описания геометрических фигур при помощи одной из главных особенностей объектно-ориентированного программирования – наследования.

Приложение А

код программы LAB2.cpp

#include <iostream>

class Shape {

protected:

double \_x;

double \_y;

double \_angle;

int \_r;

int \_g;

int \_b;

int \_ID;

static int nextID;

public:

Shape(double x = 0, double y = 0, double angle = 0) : \_x(x), \_y(y), \_angle(angle), \_ID(++nextID) {

\_r = \_g = \_b = 0;

}

virtual ~Shape() {}

void move(double new\_x, double new\_y) {

\_x = new\_x;

\_y = new\_y;

}

void setColor(int r, int g, int b) {

\_r = r;

\_g = g;

\_b = b;

}

void getColor(int &r, int &g, int&b) {

r = \_r;

g = \_g;

b = \_b;

}

virtual void rotate(double angle) = 0;

virtual void scale(int zoom) = 0;

friend std::ostream & operator<<(std::ostream &out, const Shape &figure) {

        out << "Coordinates of the center: (" << figure.\_x << ", " << figure.\_y << ")" << std::endl;

out << "Angle: " << figure.\_angle << std::endl;

out << "ID <" << figure.\_ID << ">" << std::endl;

out << "Color: red-" << figure.\_r << " green-" << figure.\_g << " blue-" << figure.\_b << std::endl;

        return out;

    }

};

int Shape::nextID = 0;

class Circle : public Shape {

protected:

double \_radius;

public:

Circle(double x = 0, double y = 0, double angle = 0, double radius = 0) : Shape(x, y, angle), \_radius(radius) {}

~Circle() {}

void scale(int zoom) override {

\_radius \*= zoom;

}

void rotate(double angle) override {

\_angle += angle;

}

friend std::ostream & operator<< (std::ostream &out, Circle &figure) {

out << "\tCIRCLE" << std::endl;

out << (Shape&) figure << std::endl;

out << "Radius: " << figure.\_radius;

         return out;

     }

};

class CircleSeg : public Circle {

protected:

double \_firstSegAngle;

double \_secondSegAngle;

public:

CircleSeg(double firstAng, double secondAng, double radius = 0, double x = 0, double y = 0, double angle = 0)

: Circle( x, y, angle, radius), \_firstSegAngle(firstAng), \_secondSegAngle(secondAng) {}

~CircleSeg() {}

friend std::ostream & operator << (std::ostream &out, const CircleSeg &figure) {

out << "\tSEGMENT" << std::endl;

out << (Circle&) figure << std::endl;

out << "Segment angle: " << figure.\_firstSegAngle - figure.\_secondSegAngle << std::endl;

return out;

     }

};

class Trapeze : public Shape {

protected:

double \_topSide;

double \_bottomSide;

double \_height;

double \_x\_tsm;

double \_y\_tsm;

double \_x\_bsm;

double \_y\_bsm;

public:

Trapeze(double top, double bottom, double x\_tsm, double y\_tsm, double x\_bsm, double y\_bsm, double x = 0, double y = 0, double angle = 0)

: Shape(x, y, angle),

\_topSide(top), \_bottomSide(bottom),

\_x\_tsm(x\_tsm), \_y\_tsm(y\_tsm),

\_x\_bsm(x\_bsm), \_y\_bsm(y\_bsm) {

\_height = \_y\_tsm - \_y\_bsm;

\_y = \_height / 2;

\_x = (\_topSide + \_bottomSide) / 4;

}

~Trapeze() {}

void scale(int zoom) override {

\_topSide \*= zoom;

\_bottomSide \*= zoom;

\_height \*= zoom;

\_x\_tsm \*= zoom;

\_y\_tsm \*= zoom;

\_x\_bsm \*= zoom;

\_y\_bsm \*= zoom;

}

void rotate(double angle) override {

\_angle += angle;

}

friend std::ostream & operator << (std::ostream &out, const Trapeze &figure) {

out << "\tTRAPEZE" << std::endl;

out << (Shape&) figure << std::endl;

out << "Side length: top-" << figure.\_topSide << " bottom-" << figure.\_bottomSide << std::endl;

out << "Height: " << figure.\_height << std::endl;

return out;

    }

};

int main() {

Circle A(5,5,20,15);

CircleSeg B(48, 20, 13);

Trapeze C(13, 27, 15, 7, 9, 3);

    std::cout << A << std::endl;

std::cout << B << std::endl;

std::cout << C << std::endl;

A.rotate(35);

A.scale(3);

A.setColor(255, 128, 0);

B.rotate(77);

B.scale(5);

B.setColor(15,15,15);

C.rotate(3);

C.scale(7);

C.setColor(33,5,7);

std::cout << A << std::endl;

std::cout << B << std::endl;

std::cout << C << std::endl;

}