**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. 7381 | Минуллин М. А. |
| 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.

**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:

1. Условие задания;
2. UML диаграмму разработанных классов;
3. Текстовое обоснование проектных решений;
4. Реализацию классов на языке С++.

**Индивидуализация.**

Вариант 13 – реализовать систему классов для фигур:

1. Треугольник;
2. Эллипс;
3. Прямоугольный треугольник.

**Обоснование проектных решений.**

Для представления цвета написан отдельный класс ColorRGBA, представляющий из себя класс для представления 4-байтного цвета палитры RGBA (красная, зелёная, синяя компоненты и прозрачность – альфа-канал).

Базовым классом для всех фигур является класс Shape2D, используемый для представления плоских геометрических фигур. Хранит в себе координаты центра фигуры (для разных классов центр может определяться по-разному) – абсциссу и ординату. Перемещение фигуры представляет из себя смену координат центра фигуры, поэтому метод перемещения фигуры объявлен в этом классе, необходимости переопределять этот метод в классах-наследниках нет, поэтому он не виртуальный.

Так же этот класс содержит в себе угол поворота (в радианах, по умолчанию ) от оси . Для поворота фигуры достаточно прибавить к этому углу необходимую величину поворота, поэтому поворот так же не виртуальный метод.

Здесь же хранится цвет фигуры, к которому так же есть доступ, потребность в виртуализации отсутствует.

Метод масштабирования сделан чисто виртуальным, поскольку его реализация уже зависит от способа организации фигуры в памяти (при ленивых вычислениях можно было бы использовать переменную для хранения масштаба объекта (по умолчанию ), но любые манипуляции связанные с вычислением каких-то метрик фигуры будут приводить к постоянному домножению на масштабный коэффициент, что может, вероятно, привести к ошибкам).

Поскольку одной виртуальной функции мне показалось мало, добавлены чисто виртуальные функции вычисления периметра и площадь фигуры, поскольку данные метрики свойственны любой плоской геометрической фигуры. Виртуальны эти функции, поскольку требуют перегрузки в классах наследниках в целях поддержки парадигмы полиморфизма, а чисто виртуальны, так же как операция масштабирования, поскольку их реализация в базовом классе не представляется возможной.

Для однозначной идентификации объекта базовый класс содержит две переменные: первая статическая (по умолчанию ), увеличивающаяся при каждом вызове конструктора базового класса на единицу, находящаяся в приватной зоне класса, чтобы классы наследники не могли изменить это значение, вторая – константная, поскольку её значение однозначно определяется при создании объекта (по значению счётчика объектов). Такой метод организации идентификации используется для того, чтобы точно избежать коллизий (вследствие умышленного вмешательства в значения счётчика или идентификатора объекта).

Для перегрузки оператора вывода фигуры в поток оператор объявлен во всех классах дружественной функцией, чтобы можно было вывести значения защищённых и приватных полей.

Класс Triangle, используемый для представления разностороннего треугольника, наследуется от Shape2D, содержит в себе защищённые (чтобы иметь к ним доступ из классов-наследников) поля для хранения длин сторон треугольника. Центр разностороннего треугольника при этом считается центром описанной окружности.

Для нахождения периметра треугольника используется тривиальная формула . Для нахождения площади используется формула Герона: , где – полупериметр треугольника. Для масштабирования длины сторон треугольника умножаются на масштабный коэффициент. Для нахождения координат углов треугольника используются формулы:

, где – координаты соответствующего угла треугольника (), – координаты центра треугольника, – радиус описанной окружности, – угол поворота соответствующего угла вокруг центра треугольника, определяемый следующим образом:

Класс RightTriangle является наследником класса Triangle, используется для представления прямоугольного треугольника. Поскольку все формулы, в общем-то справедливы и для прямоугольного треугольника, то изменена только функция нахождения площади, поскольку площадь прямоугольного треугольника можно вычислить быстрее по формуле , где и – катеты. Так же данный класс отличается от класса-родителя конструктором: данный класс принимает всего 2 аргумента – катеты (гипотенуза вычисляется очевидным образом ).

Класс Ellipse используется для представления эллипса, наследуется от Shape2D, хранит в себе 2 размера полуосей: мажорную и минорную. Периметр определяется по формуле , площадь .

**UML диаграмма разработанных классов.**

UML диаграмма разработанных классов представлена в приложении А и в соседнем документе (UML.png).

**Реализация классов на языке С++.**

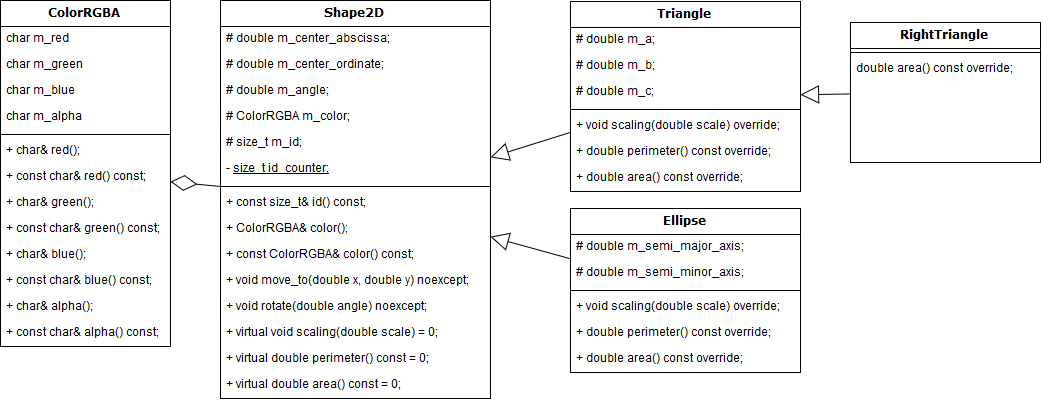
Реализация классов представлена в приложении Б.

**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 КЛАССОВ**

****

**ПРИЛОЖЕНИЕ Б**

**РЕАЛИЗАЦИЯ КЛАССОВ НА ЯЗЫКЕ C++**

#include <iostream>

#include <cmath>

class ColorRGBA

{

public:

ColorRGBA();

ColorRGBA(char red, char green, char blue, char alpha);

ColorRGBA(const ColorRGBA& color);

char& red() noexcept;

const char& red() const noexcept;

char& green() noexcept;

const char& green() const noexcept;

char& blue() noexcept;

const char& blue() const noexcept;

char& alpha() noexcept;

const char& alpha() const noexcept;

friend std::ostream& operator<<(std::ostream& stream, const ColorRGBA& color);

private:

char m\_red;

char m\_green;

char m\_blue;

char m\_alpha;

};

ColorRGBA::ColorRGBA()

: m\_red(0), m\_green(0), m\_blue(0), m\_alpha(0)

{

}

ColorRGBA::ColorRGBA(char red, char green, char blue, char alpha)

: m\_red(red), m\_green(green), m\_blue(blue), m\_alpha(alpha)

{

}

ColorRGBA::ColorRGBA(const ColorRGBA& color)

: m\_red(color.m\_red), m\_green(color.m\_green), m\_blue(color.m\_blue), m\_alpha(color.m\_alpha)

{

}

char& ColorRGBA::red() noexcept

{

return m\_red;

}

const char& ColorRGBA::red() const noexcept

{

return m\_red;

}

char& ColorRGBA::green() noexcept

{

return m\_green;

}

const char& ColorRGBA::green() const noexcept

{

return m\_green;

}

char& ColorRGBA::blue() noexcept

{

return m\_blue;

}

const char& ColorRGBA::blue() const noexcept

{

return m\_blue;

}

char& ColorRGBA::alpha() noexcept

{

return m\_alpha;

}

const char& ColorRGBA::alpha() const noexcept

{

return m\_alpha;

}

std::ostream& operator<<(std::ostream& stream, const ColorRGBA& color)

{

std::cout << "[" << size\_t(color.m\_red) << ", " << size\_t(color.m\_green) << ", " << size\_t(color.m\_blue) << ", " << size\_t(color.m\_alpha) << "]" << std::endl;

return stream;

}

// base abstact class for flat figures

class Shape2D

{

public:

Shape2D();

Shape2D(double x, double y, const ColorRGBA& color);

const size\_t& id() const noexcept;

// get-set color of shape

ColorRGBA& color();

const ColorRGBA& color() const;

// move shape to coordinates (x; y)

void move\_to(double x, double y) noexcept;

// rotate the figure at angle

void rotate(double angle) noexcept;

virtual void scaling(double scale) = 0;

virtual double perimeter() const = 0;

virtual double area() const = 0;

protected:

// center coordinates

double m\_center\_abscissa;

double m\_center\_ordinate;

// angel of rotation from the axis

double m\_angle;

// shape color

ColorRGBA m\_color;

const size\_t m\_id;

private:

static size\_t id\_counter;

};

size\_t Shape2D::id\_counter = 0;

Shape2D::Shape2D()

: m\_center\_abscissa(0.0), m\_center\_ordinate(0.0), m\_angle(0.0), m\_color(), m\_id(id\_counter)

{

++id\_counter;

}

Shape2D::Shape2D(double x, double y, const ColorRGBA& color)

: m\_center\_abscissa(x), m\_center\_ordinate(y), m\_angle(0.0), m\_color(color), m\_id(id\_counter)

{

++id\_counter;

}

const size\_t& Shape2D::id() const noexcept

{

return m\_id;

}

ColorRGBA& Shape2D::color()

{

return m\_color;

}

const ColorRGBA& Shape2D::color() const

{

return m\_color;

}

void Shape2D::move\_to(double x, double y) noexcept

{

m\_center\_abscissa = x;

m\_center\_ordinate = y;

}

void Shape2D::rotate(double angle) noexcept

{

m\_angle += angle;

}

class Triangle : public Shape2D

{

public:

Triangle();

Triangle(double x, double y, const ColorRGBA& color, double a, double b, double c);

void scaling(double scale) override;

double perimeter() const override;

double area() const override;

friend std::ostream& operator<<(std::ostream& stream, const Triangle& triangle);

protected:

// three sides of triangle

double m\_a;

double m\_b;

double m\_c;

};

Triangle::Triangle()

: Shape2D(), m\_a(0.0), m\_b(0.0), m\_c(0.0)

{

}

Triangle::Triangle(double x, double y, const ColorRGBA& color, double a, double b, double c)

: Shape2D(x, y, color), m\_a(a), m\_b(b), m\_c(c)

{

}

void Triangle::scaling(double scale)

{

m\_a \*= scale;

m\_b \*= scale;

m\_c \*= scale;

}

double Triangle::perimeter() const

{

return (m\_a + m\_b + m\_c);

}

double Triangle::area() const

{

double p = perimeter() / 2.0;

return (std::sqrt(p \* (p - m\_a) \* (p - m\_b) \* (p - m\_c)));

}

std::ostream& operator<<(std::ostream& stream, const Triangle& triangle)

{

stream << "triangle output" << std::endl;

stream << "perimeter: " << triangle.perimeter() << std::endl;

stream << "area: " << triangle.area() << std::endl;

stream << "center abscissa: " << triangle.m\_center\_abscissa << std::endl;

stream << "center ordinate: " << triangle.m\_center\_ordinate << std::endl;

stream << "rotate angle: " << triangle.m\_angle << std::endl;

stream << "side 1: " << triangle.m\_a << std::endl;

stream << "side 2: " << triangle.m\_b << std::endl;

stream << "side 3: " << triangle.m\_c << std::endl;

stream << "triangle points: " << std::endl;

double a = triangle.m\_a;

double b = triangle.m\_b;

double c = triangle.m\_c;

double P = triangle.perimeter();

double S = triangle.area();

double angle1 = triangle.m\_angle;

double angle2 = angle1 + 2.0 \* M\_PI \* a / P;

double angle3 = angle2 + 2.0 \* M\_PI \* b / P;

double xc = triangle.m\_center\_abscissa;

double yc = triangle.m\_center\_ordinate;

double R = a \* b \* c / 4.0 / S;

double x1 = xc + R \* std::cos(angle1);

double y1 = xc + R \* std::sin(angle1);

double x2 = xc + R \* std::cos(angle2);

double y2 = xc + R \* std::sin(angle2);

double x3 = xc + R \* std::cos(angle3);

double y3 = xc + R \* std::sin(angle3);

stream << "1 point: " << "(" << x1 << ", " << y1 << ")" << std::endl;

stream << "2 point: " << "(" << x2 << ", " << y2 << ")" << std::endl;

stream << "3 point: " << "(" << x3 << ", " << y3 << ")" << std::endl;

return stream;

}

class RightTriangle : public Triangle

{

public:

RightTriangle();

RightTriangle(double x, double y, const ColorRGBA& color, double cathetus\_1, double cathetus\_2);

// overloading area() method, because we can compute area faster than Heron

double area() const override;

friend std::ostream& operator<<(std::ostream& stream, const RightTriangle& rightTriangle);

protected:

};

RightTriangle::RightTriangle()

: Triangle()

{

}

RightTriangle::RightTriangle(double x, double y, const ColorRGBA& color, double cathetus\_1, double cathetus\_2)

: Triangle(x, y, color, cathetus\_1, cathetus\_2, std::hypot(cathetus\_1, cathetus\_2))

{

}

double RightTriangle::area() const

{

return (m\_a \* m\_b / 2.0);

}

std::ostream& operator<<(std::ostream& stream, const RightTriangle& rightTriangle)

{

stream << "right triangle output: " << std::endl;

stream << "perimeter: " << rightTriangle.perimeter() << std::endl;

stream << "area: " << rightTriangle.area() << std::endl;

stream << "center abscissa: " << rightTriangle.m\_center\_abscissa << std::endl;

stream << "center ordinate: " << rightTriangle.m\_center\_ordinate << std::endl;

stream << "rotate angle: " << rightTriangle.m\_angle << std::endl;

stream << "catet 1: " << rightTriangle.m\_a << std::endl;

stream << "catet 2: " << rightTriangle.m\_b << std::endl;

stream << "hypotenuse: " << rightTriangle.m\_c << std::endl;

stream << "triangle points: " << std::endl;

double a = rightTriangle.m\_a;

double b = rightTriangle.m\_b;

double c = rightTriangle.m\_c;

double P = rightTriangle.perimeter();

double S = rightTriangle.area();

double angle1 = rightTriangle.m\_angle;

double angle2 = angle1 + 2.0 \* M\_PI \* a / P;

double angle3 = angle2 + 2.0 \* M\_PI \* b / P;

double xc = rightTriangle.m\_center\_abscissa;

double yc = rightTriangle.m\_center\_ordinate;

double R = a \* b \* c / 4.0 / S;

double x1 = xc + R \* std::cos(angle1);

double y1 = xc + R \* std::sin(angle1);

double x2 = xc + R \* std::cos(angle2);

double y2 = xc + R \* std::sin(angle2);

double x3 = xc + R \* std::cos(angle3);

double y3 = xc + R \* std::sin(angle3);

stream << "1 point: " << "(" << x1 << ", " << y1 << ")" << std::endl;

stream << "2 point: " << "(" << x2 << ", " << y2 << ")" << std::endl;

stream << "3 point: " << "(" << x3 << ", " << y3 << ")" << std::endl;

return stream;

}

class Ellipse : public Shape2D

{

public:

Ellipse();

Ellipse(double x, double y, const ColorRGBA& color, double a, double b);

void scaling(double scale) override;

double perimeter() const override;

double area() const override;

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

protected:

double m\_semi\_major\_axis;

double m\_semi\_minor\_axis;

};

Ellipse::Ellipse()

: Shape2D(), m\_semi\_major\_axis(0.0), m\_semi\_minor\_axis(0.0)

{

}

Ellipse::Ellipse(double x, double y, const ColorRGBA& color, double a, double b)

: Shape2D(x, y, color), m\_semi\_major\_axis(a), m\_semi\_minor\_axis(b)

{

}

void Ellipse::scaling(double scale)

{

m\_semi\_major\_axis \*= scale;

m\_semi\_minor\_axis \*= scale;

}

double Ellipse::perimeter() const

{

return (std::sqrt(2.0) \* M\_PI \* std::hypot(m\_semi\_major\_axis, m\_semi\_minor\_axis));

}

double Ellipse::area() const

{

return (M\_PI \* m\_semi\_major\_axis \* m\_semi\_minor\_axis);

}

std::ostream& operator<<(std::ostream& stream, const Ellipse& ellipse)

{

stream << "ellipse output" << std::endl;

stream << "perimeter: " << ellipse.perimeter() << std::endl;

stream << "area: " << ellipse.area() << std::endl;

stream << "center abscissa: " << ellipse.m\_center\_abscissa << std::endl;

stream << "center ordinate: " << ellipse.m\_center\_ordinate << std::endl;

stream << "rotate angle: " << ellipse.m\_angle << std::endl;

stream << "ellipse points: " << std::endl;

double P = ellipse.perimeter();

double S = ellipse.area();

double R1 = ellipse.m\_semi\_minor\_axis;

double R2 = ellipse.m\_semi\_major\_axis;

double angle1 = ellipse.m\_angle;

double angle2 = angle1 + M\_PI / 2.0;

double angle3 = angle2 + M\_PI / 2.0;

double angle4 = angle3 + M\_PI / 2.0;

double xc = ellipse.m\_center\_abscissa;

double yc = ellipse.m\_center\_ordinate;

double x1 = xc + R1 \* std::cos(angle1);

double y1 = xc + R1 \* std::sin(angle1);

double x2 = xc + R2 \* std::cos(angle2);

double y2 = xc + R2 \* std::sin(angle2);

double x3 = xc + R1 \* std::cos(angle3);

double y3 = xc + R1 \* std::sin(angle3);

double x4 = xc + R2 \* std::cos(angle4);

double y4 = xc + R2 \* std::sin(angle4);

stream << "1 point: " << "(" << x1 << ", " << y1 << ")" << std::endl;

stream << "2 point: " << "(" << x2 << ", " << y2 << ")" << std::endl;

stream << "3 point: " << "(" << x3 << ", " << y3 << ")" << std::endl;

stream << "4 point: " << "(" << x4 << ", " << y4 << ")" << std::endl;

return stream;

}

int main()

{

// default Triangle constructor

Triangle triangle;

std::cout << triangle.id() << ": " << triangle;

// default RightTriangle constructor

RightTriangle rightTriangle;

std::cout << rightTriangle.id() << ": " << rightTriangle;

// default Ellipse constructor

Ellipse ellipse;

std::cout << ellipse.id() << ": " << ellipse;

Triangle triangle2(42.0, 42.0, ColorRGBA(), 3.0, 4.0, 5.0);

std::cout << "triangle perimeter: " << triangle2.perimeter() << std::endl;

std::cout << "triangle area: " << triangle2.area() << std::endl;

std::cout << triangle2;

RightTriangle rightTriangle2(42.0, 42.0, ColorRGBA(), 3.0, 4.0);

std::cout << "right triangle perimeter: " << rightTriangle2.perimeter() << std::endl;

std::cout << "right triangle area: " << rightTriangle2.area() << std::endl;

std::cout << "right triangle scaling (x5)..." << std::endl;

rightTriangle2.scaling(5.0);

std::cout << "right triangle perimeter: " << rightTriangle2.perimeter() << std::endl;

std::cout << "right triangle area: " << rightTriangle2.area() << std::endl;

Ellipse ellipse2(42.0, 42.0, ColorRGBA(4, 8, 15, 16), 4.0, 2.0);

std::cout << "ellipse color: " << ellipse2.color();

std::cout << "ellipse color modification..." << std::endl;

ellipse2.color() = ColorRGBA(15, 16, 23, 42);

std::cout << "ellipse color: " << ellipse2.color();

std::cout << triangle2 << std::endl;

std::cout << rightTriangle2 << std::endl;

std::cout << ellipse2 << std::endl;

return EXIT\_SUCCESS;

}