**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**ОТЧЕТ**

**по лабораторной работе №2**

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

**Тема: «Наследование»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент гр. 7381 | |  | Вологдин М.Д. |
| Преподаватель |  |  | Жангиров Т. Р. |

Санкт-Петербург

2019

**Цель работы:**

Ознакомиться с наследованием, полиморфизмом, абстрактными классами и виртуальными функциями – принципами их работы и организацией в памяти в языке C++. В соответствии с индивидуальным заданием разработать систему классов для моделирования 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++.

**Вариант 4:**

1. Круг
2. Пятиконечная звезда
3. Шестиконечная звезда

**Обоснование проектных решений**

1. Для хранения точки и цвета были созданы структуры Point и RGB соответственно.
2. Для общего представления геометрических фигур был создан абстрактный класс Shape, который хранит в себе угол поворота, цвет, координаты центра, масштаб и id фигуры, а также общие методы для установки и получения цвета и вывода общей информации о фигуре. Функции перемещения, масштабирования и поворота чисто виртуальные, так как их реализация зависит от фигуры.
3. Звезда представлены классом PointedStar. Определяется координатами центра, радиусом и количеством лучей.
4. Пятиконечная и шестиконечная звезда представлены классами FivePointedStar и SixPointedStar соответственно. Такие звезды определяются координатами центра и радиусом.
5. Круг представлен классом Circle. Он определяется координатами центра и длиной радиуса. Метод поворота отсутствует за ненадобностью
6. Для идентификации объекта в базовом классе содержатся две переменные: статическая, которая увеличивается при создании фигуры на единицу, и константная, которая однозначно определяет id фигуры.
7. Перегруженный оператор “<<” объявлен во всех классах дружественной функцией, чтобы иметь возможно выводить значения защищённых и приватных полей.

UML диаграмма разработанных классов представлена в приложении Б. Код представлен в приложении А.

**Выводы:**

В результате работы было изучено наследование в C++, спроектирована система классов для работы с геометрическими фигурами.

**Приложение А**

**Исходный код**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

#include <iostream>

#include <cmath>

#include <vector>

struct Point

{

double x;

double y;

};

struct RGB

{

unsigned char R;

unsigned char G;

unsigned char B;

};

class Shape {

private:

static int NextCustomerId;

protected:

int angle;

RGB color;

Point centre;

double Scale;

const int id;

public:

Shape (Point xy)

: Scale(1), angle(0), color({0,0,0}), centre(xy), id(++NextCustomerId)

{}

void virtual MoveFigure(Point xy) = 0;

void virtual SetTurnAngle(int other\_angle) = 0;

void virtual Scaling(double k) = 0;

void SetColor(unsigned char R, unsigned char G, unsigned char B)

{

color = { R, G, B };

}

RGB GetColor()

{

return color;

}

void PrintShapeInfo()

{

std::cout << "Shape ID: " << id << std::endl;

std::cout << "Centre: (" << centre.x << "; " << centre.y << ")" << std::endl;

std::cout << "Angle = " << angle << std::endl;

std::cout << "Color: (" << static\_cast<int>(color.R) << "; " << static\_cast<int>(color.G) << "; " << static\_cast<int>(color.B) << ")" << std::endl;

std::cout << "Scale = " << Scale << std::endl << std::endl;

}

virtual ~Shape() {}

};

int Shape:: NextCustomerId = 0;

class PointedStar : public Shape

{

private:

double radius;

int count;

double m\_rad;

public:

PointedStar(Point xy, double rad, int count)

: Shape(xy), radius(rad), count (count), m\_rad((8\*rad)/21)

{}

void MoveFigure(Point xy) override

{

centre.x = xy.x;

centre.y = xy.y;

}

void SetTurnAngle(int new\_angle) override

{

angle += new\_angle;

angle %= 360;

}

void Scaling(double k) override

{

Scale \*= k;

radius\*= k;

}

friend std::ostream& operator<<(std::ostream& stream, PointedStar & Star);

};

std::ostream& operator<<(std::ostream& stream, PointedStar & Star)

{

stream << Star.count << "-pointed star" << std::endl;

Star.PrintShapeInfo();

stream << "Points coordinates:\n";

for(int i=0;i<Star.count;i++)

{

stream << "(" <<Star.centre.x + Star.radius \* cos(Star.angle + i \* 2 \* M\_PI / Star.count) << "; " << Star.centre.y + Star.radius \* sin(Star.angle + i \* 2 \* M\_PI / Star.count) << ")" << std::endl;

stream << "(" <<Star.centre.x + Star.m\_rad \* cos(Star.angle + M\_PI/Star.count + i \* 2 \* M\_PI / Star.count) << "; " << Star.centre.y + Star.m\_rad \* sin(Star.angle + M\_PI/Star.count + i \* 2 \* M\_PI / Star.count) << ")" << std::endl;

}

stream << std::endl;

return stream;

}

class FivePointedStar : public PointedStar

{

public:

FivePointedStar(Point xy, double rad)

: PointedStar(xy,rad, 5)

{

}

};

class SixPointedStar : public PointedStar

{

public:

SixPointedStar(Point xy, double rad)

: PointedStar(xy,rad, 6)

{

}

};

class Circle : public Shape

{

private:

double radius;

public:

Circle(Point xy, double radius) : Shape(xy),

radius(radius) {}

void Scaling(double k) override

{

Scale\*=k;

radius \*= k;

}

void MoveFigure(Point xy) override

{

centre.x = xy.x;

centre.y = xy.y;

}

void SetTurnAngle(int rotation\_angle) override

{}

friend std::ostream &operator<<(std::ostream &stream, Circle &circle);

};

std::ostream &operator<<(std::ostream &stream, Circle &circle)

{

stream.precision(1);

stream.setf(std::ios::fixed);

stream << "Circle" << std::endl;

circle.PrintShapeInfo();

stream << "Circle radius: " << circle.radius << std::endl << std::endl;

return stream;

}

**Приложение Б**

**UML диаграмма**

