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 C++.

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>

struct Point
{
    double x;
    double y;
};

struct RGB
{
    int R;
    int G;
    int 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(int R, int G, int 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: (" << color.R << "; " << color.G << "; " <<
color.B << ")" << std::endl;
        std::cout << "Scale = " << Scale << std::endl << std::endl;
    }

    friend std::ostream& operator<<(std::ostream& stream, Shape& sh);
    virtual ~Shape() {}
};

int Shape:: NextCustomerId = 0;

class FivePointedStar : public Shape {
private:
    Point X1, X2, X3, X4, X5;
    double radius;
public:
    FivePointedStar(Point xy, double rad)
    : Shape(xy),  radius(rad),
    X1({xy.x + rad * cos(0), xy.y + rad * sin(0)}),
    X2({xy.x + rad * cos(1 * 2 * M_PI / 5), xy.y + rad * sin(1 * 2 * M_PI
/ 5)}),
    X3({xy.x + rad * cos(2 * 2 * M_PI / 5), xy.y + rad * sin(2 * 2 * M_PI
/ 5)}),
    X4({xy.x + rad * cos(3 * 2 * M_PI / 5), xy.y + rad * sin(3 * 2 * M_PI
/ 5)}),
    X5({xy.x + rad * cos(4 * 2 * M_PI / 5), xy.y + rad * sin(4 * 2 * M_PI
/ 5)})
    {}

    void MoveFigure(Point xy) override
    {
        X1.x += (xy.x - centre.x);
        X1.y += (xy.y - centre.y);
        X2.x += (xy.x - centre.x);
        X2.y += (xy.y - centre.y);
        X3.x += (xy.x - centre.x);
        X3.y += (xy.y - centre.y);
        X4.x += (xy.x - centre.x);
        X4.y += (xy.y - centre.y);
        X5.x += (xy.x - centre.x);
        X5.y += (xy.y - centre.y);
        centre.x = xy.x;
        centre.y = xy.y;

    }
```

```
    void SetTurnAngle(int new_angle) override
    {
        angle += new_angle;
        angle %= 360;
        double a_rad = angle * M_PI / 180;
        X1.x = centre.x + radius * cos(a_rad + 0);
        X1.y = centre.y + radius * sin(a_rad + 0);
        X2.x = centre.x + radius * cos(a_rad + 1 * 2 * M_PI / 5);
        X2.y = centre.y + radius * sin(a_rad + 1 * 2 * M_PI / 5);
        X3.x = centre.x + radius * cos(a_rad + 2 * 2 * M_PI / 5);
        X3.y = centre.y + radius * sin(a_rad + 2 * 2 * M_PI / 5);
        X4.x = centre.x + radius * cos(a_rad + 3 * 2 * M_PI / 5);
        X4.y = centre.y + radius * sin(a_rad + 3 * 2 * M_PI / 5);
        X5.x = centre.x + radius * cos(a_rad + 4 * 2 * M_PI / 5);
        X5.y = centre.y + radius * sin(a_rad + 4 * 2 * M_PI / 5);

    }

    void Scaling(double k) override
    {
        Scale *= k;
        centre.x *= k;
        centre.y *= k;
        X1.x *= k;
        X1.y *= k;
        X2.x *= k;
        X2.y *= k;
        X3.x *= k;
        X3.y *= k;
        X4.x *= k;
        X4.y *= k;
        X5.x *= k;
        X5.y *= k;
        MoveFigure({centre.x/k, centre.y/k});
    }

    friend std::ostream& operator<<(std::ostream& stream,
FivePointedStar& fpStar)
    {
        stream << "five-pointed star" << std::endl;
        fpStar.PrintShapeInfo();
        stream << "Points coordinates:\n";
        stream << "(" << fpStar.X1.x << "; " << fpStar.X1.y << ")"  <<
std::endl;
        stream << "(" << fpStar.X2.x << "; " << fpStar.X2.y << ")"  <<
std::endl;
        stream << "(" << fpStar.X3.x << "; " << fpStar.X3.y << ")"  <<
std::endl;
        stream << "(" << fpStar.X4.x << "; " << fpStar.X4.y << ")"  <<
std::endl;
```

```
        stream << "(" << fpStar.X5.x << "; " << fpStar.X5.y << ")"  <<
std::endl << std::endl;
        return stream;
    }
};

class SixPointedStar : public Shape
{
private:
    double radius;
    Point X1, X2, X3, X4, X5, X6;
public:
    SixPointedStar(Point xy, double radius)
    : Shape(xy), radius(radius),
    X1({xy.x + radius * cos(0), xy.y + radius * sin(0)}),
    X2({xy.x + radius * cos(1 * 2 * M_PI / 6), xy.y + radius * sin(1 * 2
* M_PI / 6)}),
    X3({xy.x + radius * cos(2 * 2 * M_PI / 6), xy.y + radius * sin(2 * 2
* M_PI / 6)}),
    X4({xy.x + radius * cos(3 * 2 * M_PI / 6), xy.y + radius * sin(3 * 2
* M_PI / 6)}),
    X5({xy.x + radius * cos(4 * 2 * M_PI / 6), xy.y + radius * sin(4 * 2
* M_PI / 6)}),
    X6({xy.x + radius * cos(5 * 2 * M_PI / 6), xy.y + radius * sin(5 * 2
* M_PI / 6)})
    {}

    void MoveFigure(Point xy) override
    {
        X1.x += (xy.x - centre.x);
        X1.y += (xy.y - centre.y);
        X2.x += (xy.x - centre.x);
        X2.y += (xy.y - centre.y);
        X3.x += (xy.x - centre.x);
        X3.y += (xy.y - centre.y);
        X4.x += (xy.x - centre.x);
        X4.y += (xy.y - centre.y);
        X5.x += (xy.x - centre.x);
        X5.y += (xy.y - centre.y);
        X6.x += (xy.x - centre.x);
        X6.y += (xy.y - centre.y);
        centre.x = xy.x;
        centre.y = xy.y;
    }

    void Scaling(double k) override
    {
        Scale *= 2;
        radius *= k;
        centre.x *= k;
        centre.y *= k;
```

```
        X1.x *= k;
        X1.y *= k;
        X2.x *= k;
        X2.y *= k;
        X3.x *= k;
        X3.y *= k;
        X4.x *= k;
        X4.y *= k;
        X5.x *= k;
        X5.y *= k;
        X6.x *= k;
        X6.y *= k;
        MoveFigure({centre.x/k, centre.y/k});
    }

    void SetTurnAngle(int rotation_angle) override
    {
        angle += rotation_angle;
        angle %= 360;
        double a_rad = angle * M_PI / 180;

        X1.x = centre.x + radius * cos(a_rad + 0);
        X1.y = centre.y + radius * sin(a_rad + 0);
        X2.x = centre.x + radius * cos(a_rad + 1 * 2 * M_PI / 6);
        X2.y = centre.y + radius * sin(a_rad + 1 * 2 * M_PI / 6);
        X3.x = centre.x + radius * cos(a_rad + 2 * 2 * M_PI / 6);
        X3.y = centre.y + radius * sin(a_rad + 2 * 2 * M_PI / 6);
        X4.x = centre.x + radius * cos(a_rad + 3 * 2 * M_PI / 6);
        X4.y = centre.y + radius * sin(a_rad + 3 * 2 * M_PI / 6);
        X5.x = centre.x + radius * cos(a_rad + 4 * 2 * M_PI / 6);
        X5.y = centre.y + radius * sin(a_rad + 4 * 2 * M_PI / 6);
        X6.x = centre.x + radius * cos(a_rad + 5 * 2 * M_PI / 6);
        X6.y = centre.y + radius * sin(a_rad + 5 * 2 * M_PI / 6);
    }

    friend std::ostream &operator<<(std::ostream &stream, SixPointedStar
&SStar) {
        stream << "SixPointedStar" << std::endl;
        SStar.PrintShapeInfo();
        stream << "Radius: " << SStar.radius << std::endl;
        stream << "Point coordinates:\n";
        stream << "(" << SStar.X1.x << "; " << SStar.X1.y << ")" <<
std::endl;
        stream << "(" << SStar.X2.x << "; " << SStar.X2.y << ")" <<
std::endl;
        stream << "(" << SStar.X3.x << "; " << SStar.X3.y << ")" <<
std::endl;
        stream << "(" << SStar.X4.x << "; " << SStar.X4.y << ")" <<
std::endl;
        stream << "(" << SStar.X5.x << "; " << SStar.X5.y << ")" <<
std::endl;
```

```
        stream << "(" << SStar.X6.x << "; " << SStar.X6.y << ")" <<
std::endl << std::endl;

        return stream;
    }

};

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)
{
        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;
    }

};

int main() {

    std::cout.precision(1);
    std::cout.setf(std::ios::fixed);

    FivePointedStar Pent({0, 0}, 10);
    std::cout << Pent;
```

```
    Pent.MoveFigure({5, 5});
    Pent.SetTurnAngle(30);
    Pent.Scaling(2);
    std::cout << Pent;

    SixPointedStar SDavid({0, 0}, 10);
    std::cout << SDavid;
    SDavid.Scaling(2);
    SDavid.MoveFigure({5, 5});
    SDavid.SetTurnAngle(30);
    SDavid.SetTurnAngle(30);
    std::cout << SDavid;

    Circle circl({0, 10}, 5);
    std::cout << circl;
    circl.SetColor(256, 256, 256);
    circl.Scaling(2);
    std::cout << circl;

    return 0;
}
```

Приложение Б

UML диаграмма