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. 6304 _____ Васильев А.А.

Преподаватель _____ Терентьев А.О.

Санкт-Петербург

2018

Цель работы.

Исследование механизма наследования в языке C++.

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 C++.

Вариант 3 – Равнобедренный треугольник, квадрат, прямоугольный треугольник

Ход работы.

1) Реализуем структуру Point для хранения координат и их удобного сложения:

```
struct Point
{
  Point(double xx, double yy)
          :x{xx},y{yy}{}

  Point& operator+=(const Point& b)
  {
          x+=b.x;
          y+=b.y;
          return *this;
```

```
  }
  double x, y;
};

Point operator +(const Point& a, const Point& b)
{
  Point k = a;
  k+=b;
  return k;
}
```

2) Реализуем структуру Color для хранения цвета:

```
  struct Color
{
  Color(int rr,int gg,int bb)
          :r{rr},g{gg},b{bb}
  {
          if(r <0 || r>255||g <0 || g>255||b <0 || b>255)
                  throw invalid_argument("invalid color!");
  }

  int r, g, b;

};
```

3) Создаем абстрактный класс Shape. Каждый объект данного типа содержит абстрактную информацию о фигуре – размер ограничивающего фигуру прямоугольника. Производные классы используют эту информацию в зависимости от их формы.

   При такой организации виртуальными остаются только два метода – print – для вывода и метод updateCoord, который обновляет координаты всех вершин фигуры в соответствии с обновленными данными

```
class Shape
{

public:
  Shape(double w, double h, Color col = Color(255,255,255))
          :angle(0),color(col), width(w),height(h)
  {
          if(width <=0 || height<=0)
                  throw invalid_argument("negative width is forbidden!");
  }

  void setColor(Color col){color = col;}
  void setColor(int r, int g, int b){color = Color(r,g,b);}

  Shape& move(Point m)
```

```
  {
          points[0] = m;
          updateCoord();
          return *this;
  }
  Shape& move(double x, double y){ return move(Point(x,y));}

  Shape& rotate(double ang)
  {
          angle+=toRad(ang);
          updateCoord();
          return *this;
  }

  Shape& scale(double sc)
  {
          width *= sc;
          height *= sc;
          updateCoord();
          return *this;
  }

  Color getColor(){return color;}

  virtual void printFigure(ostream & out)=0;

  virtual ~Shape(){}
protected:
  virtual void updateCoord()=0;

  vector<Point> points;
  double angle;
  Color color;
  double width, height;


};

ostream& operator<<(ostream& out, Shape & s)
{
  s.printFigure(out);
  return out;
          }
```

Вектор points содержит в себе координаты вершин фигуры. Первая координата – основная точка, вокруг которой будет осуществляться вращение и относительно которой будут высчитываться остальные координаты.

Метод move перемещает основную точку и обновляет координаты, тем самым перемещается фигура.

Метод rotate вращает фигуру вокруг основной точки.

Метод scale увеличивает\уменьшает ограничивающий прямоугольник и затем обновляет координаты фигуры.

Метод paintFigure выводит информацию о типе фигуры, ее цвет и ее координаты.

4) Производные классы должны определять методы updateCoord и printFigure. Конструкторы данных классов заполняют массив с координатами

```
class Square : public Shape
{
public:
	Square(Point BottomLeft,double width,Color col = Color(255,255,255))
		:Shape(width,width,col)
	{
		points.push_back(BottomLeft);
		points.push_back(BottomLeft + Point(width,0));
		points.push_back(BottomLeft + Point(width,width));
		points.push_back(BottomLeft + Point(0,width));
	}

	Square(double x, double y, double w ,Color col = Color(255,255,255))
		:Square(Point(x,y), w,col){}

	void printFigure(ostream& out) override
	{
		out << "This figure is square.\n\tCoordinates are: ";
		for(auto i:points)
			out << i << " ";
		out << "\n\tColor is " <<color <<"\n";

	}

protected:
	void updateCoord() override
	{
		points[1] = points[0] + Point(cos(angle)*width,sin(angle)*width);
		double len = sqrt(2)*width;
		points[2] = points[0] +
Point(cos(angle+M_PI/4)*len,sin(angle+M_PI/4)*len);
		points[3] = points[0] +
Point(cos(angle+M_PI/2)*width,sin(angle+M_PI/2)*width);
	}
};

class RightTriangle : public Shape
{
public:
	RightTriangle(Point LeftBottom, double w, double h,Color col =
Color(255,255,255))
		:Shape(w,h,col)
	{
		points.push_back(LeftBottom);
		points.push_back(LeftBottom + Point(w,0));
		points.push_back(LeftBottom + Point(0,h));
	}
	RightTriangle(double x, double y, double w, double h,Color col =
Color(255,255,255))
```

```
                :RightTriangle(Point(x,y), w,h,col){}

        void printFigure(ostream & out) override
        {
                out << "This figure is right triangle.\n\tCoordinates are: ";
                for(auto i:points)
                        out << i << " ";
                out << "\n\tColor is " <<color<<"\n";
        }

protected:
        void updateCoord() override
        {
                points[1] = points[0] + Point(cos(angle)*width,sin(angle)*width);
                points[2] = points[0] +
Point(cos(angle+M_PI/2)*height,sin(angle+M_PI/2)*height);
        }
};

class IsosTriangle : public Shape
{
public:
        IsosTriangle(Point LeftBottom, double w, double h,Color col =
Color(255,255,255))
                :Shape(w,h,col)
        {
                points.push_back(LeftBottom);
                points.push_back(LeftBottom + Point(w,0));
                points.push_back(LeftBottom + Point(w/2,h));

        }
        IsosTriangle(double x, double y, double w, double h,Color col =
Color(255,255,255))
                :IsosTriangle(Point(x,y), w,h,col){}

        void printFigure(ostream & out) override
        {
                out << "This figure is isosceles triangle.\n\tCoordinates are: ";
                for(auto i:points)
                        out << i << " ";
                out << "\n\tColor is " <<color<<"\n";
        }

protected:
        void updateCoord() override
        {
                double edge_len = sqrt(pow(width/2,2) + pow(height,2));
                double add_angle = asin(height/edge_len);
                points[1] = points[0] + Point(cos(angle)*width,sin(angle)*width);
                points[2] = points[0] +
Point(cos(angle+add_angle)*edge_len,sin(angle+add_angle)*edge_len);
        }
       };
```

5) Для тестирования создается функция main:

```
  #include <iostream>
#include "shape.h"
using namespace std;
```

```
int main()
{
          cout << "Square test!" << endl;
          Square s(Point(1,1),1,Color(1,2,3));
          cout << s;
          s.move(Point(2,2)).scale(2).rotate(180);
          cout << s;

          cout << "\nRight tri test!" << endl;
          RightTriangle t(Point(0,0),4,10);
          t.setColor(Color(23,23,23));
          cout << t;
          t.move(Point(5,0)).rotate(90).scale(0.5);
          cout << t;

          cout << "\nIso tri test!" << endl;
          IsosTriangle it(Point(0,0),8,20);
          cout << it;
          it.scale(0.25).rotate(-270).move(Point(5,-1));
          cout << it;


  return 0;
}
```

Результат теста:

```
Square test!
This figure is square.
        Coordinates are: (1, 1) (2, 1) (2, 2) (1, 2)
        Color is (1, 2, 3)
This figure is square.
        Coordinates are: (2, 2) (0, 2) (0, 0) (2, 0)
        Color is (1, 2, 3)

Right tri test!
This figure is right triangle.
        Coordinates are: (0, 0) (4, 0) (0, 10)
        Color is (23, 23, 23)
This figure is right triangle.
        Coordinates are: (5, 0) (5, 2) (0, 0)
        Color is (23, 23, 23)

Iso tri test!
This figure is isosceles triangle.
        Coordinates are: (0, 0) (8, 0) (4, 20)
        Color is (255, 255, 255)
This figure is isosceles triangle.
        Coordinates are: (5, -1) (5, 1) (0, 0)
        Color is (255, 255, 255)
```

Вывод.

В ходе данной лабораторной работы был исследован механизм наследования классов в языке C++

ПРИЛОЖЕНИЕ А

Код программы.

shape.h

```
#ifndef SHAPE_H
#define SHAPE_H

#include <iostream>
#include <cmath>
#include <vector>
using namespace std;

double roundTo(double chislo)
{
	double result;
	long long iChislo;
	int k = (chislo>=0)?1:-1;
	iChislo = (long long) chislo;
	result=((long long) ((chislo - iChislo)*pow(10,3)+k*0.5))/pow(10,3);
	result += iChislo;
	return result;
}

inline double toRad(double deg)
{
	return deg*M_PI/180;
}

struct Color
{
	Color(int rr,int gg,int bb)
		:r{rr},g{gg},b{bb}
	{
		if(r <0 || r>255||g <0 || g>255||b <0 || b>255)
			throw invalid_argument("invalid color!");
	}

	int r, g, b;

};

ostream& operator<<(ostream& out, Color & p)
{
	out << "(" << p.r << ", " << p.g <<", "<<  p.b<< ")";
	return out;
}

struct Point
{
	Point(double xx, double yy)
		:x{xx},y{yy}{}

	Point& operator+=(const Point& b)
	{
		x+=b.x;
		y+=b.y;
		return *this;
```

```
        }

        double x, y;
};

Point operator +(const Point& a, const Point& b)
{
        Point k = a;
        k+=b;
        return k;
}

ostream& operator<<(ostream& out, Point & p)
{
        out << "(" << roundTo(p.x) << ", " << roundTo( p.y) << ")";
        return out;
}

class Shape
{

public:
        Shape(double w, double h, Color col = Color(255,255,255))
                :angle(0),color(col), width(w),height(h)
        {
                if(width <=0 || height<=0)
                        throw invalid_argument("negative width is forbidden!");
        }

        void setColor(Color col){color = col;}
        void setColor(int r, int g, int b){color = Color(r,g,b);}

        Shape& move(Point m)
        {
                points[0] = m;
                updateCoord();
                return *this;
        }
        Shape& move(double x, double y){ return move(Point(x,y));}

        Shape& rotate(double ang)
        {
                angle+=toRad(ang);
                updateCoord();
                return *this;
        }

        Shape& scale(double sc)
        {
                width *= sc;
                height *= sc;
                updateCoord();
                return *this;
        }

        Color getColor(){return color;}

        virtual void printFigure(ostream & out)=0;

        virtual ~Shape(){}
protected:
```

```
        virtual void updateCoord()=0;

        vector<Point> points;
        double angle;
        Color color;
        double width, height;


};

ostream& operator<<(ostream& out, Shape & s)
{
        s.printFigure(out);
        return out;
}


class Square : public Shape
{
public:
        Square(Point BottomLeft,double width,Color col = Color(255,255,255))
                :Shape(width,width,col)
        {
                points.push_back(BottomLeft);
                points.push_back(BottomLeft + Point(width,0));
                points.push_back(BottomLeft + Point(width,width));
                points.push_back(BottomLeft + Point(0,width));
        }

        Square(double x, double y, double w ,Color col = Color(255,255,255))
                :Square(Point(x,y), w,col){}

        void printFigure(ostream& out) override
        {
                out << "This figure is square.\n\tCoordinates are: ";
                for(auto i:points)
                        out << i << " ";
                out << "\n\tColor is " <<color <<"\n";

        }

protected:
        void updateCoord() override
        {
                points[1] = points[0] + Point(cos(angle)*width,sin(angle)*width);
                double len = sqrt(2)*width;
                points[2] = points[0] +
Point(cos(angle+M_PI/4)*len,sin(angle+M_PI/4)*len);
                points[3] = points[0] +
Point(cos(angle+M_PI/2)*width,sin(angle+M_PI/2)*width);
        }
};

class RightTriangle : public Shape
{
public:
        RightTriangle(Point LeftBottom, double w, double h,Color col =
Color(255,255,255))
                :Shape(w,h,col)
        {
                points.push_back(LeftBottom);
```

```
                points.push_back(LeftBottom + Point(w,0));
                points.push_back(LeftBottom + Point(0,h));
        }
        RightTriangle(double x, double y, double w, double h,Color col =
Color(255,255,255))
                :RightTriangle(Point(x,y), w,h,col){}

        void printFigure(ostream & out) override
        {
                out << "This figure is right triangle.\n\tCoordinates are: ";
                for(auto i:points)
                        out << i << " ";
                out << "\n\tColor is " <<color<<"\n";
        }

protected:
        void updateCoord() override
        {
                points[1] = points[0] + Point(cos(angle)*width,sin(angle)*width);
                points[2] = points[0] +
Point(cos(angle+M_PI/2)*height,sin(angle+M_PI/2)*height);
        }
};

class IsosTriangle : public Shape
{
public:
        IsosTriangle(Point LeftBottom, double w, double h,Color col =
Color(255,255,255))
                :Shape(w,h,col)
        {
                points.push_back(LeftBottom);
                points.push_back(LeftBottom + Point(w,0));
                points.push_back(LeftBottom + Point(w/2,h));

        }
        IsosTriangle(double x, double y, double w, double h,Color col =
Color(255,255,255))
                :IsosTriangle(Point(x,y), w,h,col){}

        void printFigure(ostream & out) override
        {
                out << "This figure is isosceles triangle.\n\tCoordinates are: ";
                for(auto i:points)
                        out << i << " ";
                out << "\n\tColor is " <<color<<"\n";
        }

protected:
        void updateCoord() override
        {
                double edge_len = sqrt(pow(width/2,2) + pow(height,2));
                double add_angle = asin(height/edge_len);
                points[1] = points[0] + Point(cos(angle)*width,sin(angle)*width);
                points[2] = points[0] +
Point(cos(angle+add_angle)*edge_len,sin(angle+add_angle)*edge_len);
        }
};

#endif // SHAPE_H
```

main.cpp

```
#include <iostream>
#include "shape.h"
using namespace std;

int main()
{
                cout << "Square test!" << endl;
                Square s(Point(1,1),1,Color(1,2,3));
                cout << s;
                s.move(Point(2,2)).scale(2).rotate(180);
                cout << s;

                cout << "\nRight tri test!" << endl;
                RightTriangle t(Point(0,0),4,10);
                t.setColor(Color(23,23,23));
                cout << t;
                t.move(Point(5,0))
                                .rotate(90)
                                .scale(0.5);
                cout << t;

                cout << "\nIso tri test!" << endl;
                IsosTriangle it(Point(0,0),8,20);
                cout << it;
                it.scale(0.25).rotate(-270).move(Point(5,-1));
                cout << it;


        return 0;
}
```