ГОУ ВПО

«Дальневосточный государственный университет

путей сообщения»

Лабораторная работа №1  
Наследование и полиморфизм

Выполнил: Точилкин М.А 230 группа

Проверил: Латышевич Е.О.

Хабаровск, 2016

Цель: изучить наследование и полеморфизм.

Задание: Определить абстрактный класс Shape, обозначающий фигуру, который имеет:

* защищенное поле color типа string, обозначающее цвет фигуры;
* защищенное поле filled типа bool, обозначающее, что фигура закрашена;
* открытый конструктор без параметров, инициализирующий все поля значениями по умолчанию;
* открытый конструктор с параметрами, инициализирующий все поля указанными значениями;
* открытое свойство Color, возвращающее или задающее значение поля color;
* открытое свойство IsFilled, возвращающее или задающее значение поля filled;
* открытый абстрактный метод GetArea для получения площади фигуры;
* открытый абстрактный метод GetPerimeter для получения площади фигуры;
* открытый переопределенный метод ToString, возвращающий строку с описанием фигуры.

Определить класс Circle, обозначающий окружность и наследующий Shape, который имеет:

* защищенное поле radius типа double, обозначающее радиус окружности;
* открытый конструктор без параметров, инициализирующий все поля значениями по умолчанию;
* открытый конструктор с параметром, инициализирующий поле radius указанным значением, а остальные поля значениями по умолчанию;
* открытый конструктор с параметрами, инициализирующий все поля значениями по умолчанию;
* открытое свойство Radius, возвращающее или задающее значение поля radius;
* открытый переопределенный метод GetArea, возвращающий площадь круга, описываемого окружностью;
* открытый переопределенный метод GetPerimeter, возвращающий длину окружности;
* открытый переопределенный метод ToString, возвращающий строку с описанием окружности.

Определить класс Rectangle, обозначающий прямоугольник и наследующий Shape, который имеет:

* защищенное поле width типа double, обозначающее ширину прямоугольника;
* защищенное поле height типа double, обозначающее высоту прямоугольника;
* открытый конструктор без параметров, инициализирующий все поля значениями по умолчанию;
* открытый конструктор с двумя параметрами, инициализирующий поля width и height указанными значениями, а остальные поля значениями по умолчанию;
* открытый конструктор с параметрами, инициализирующий все поля значениями по умолчанию;
* открытое свойство Width, возвращающее или задающее значение поля width;
* открытое свойство Height, возвращающее или задающее значение поля height;
* открытый переопределенный метод GetArea, возвращающий площадь прямоугольника;
* открытый переопределенный метод GetPerimeter, возвращающий периметр прямоугольника;
* открытый переопределенный метод ToString, возвращающий строку с описанием прямоугольника.

Определить класс Square, обозначающий квадрат и наследующий Rectangle, который имеет:

* открытый конструктор без параметров, инициализирующий все поля значениями по умолчанию;
* открытый конструктор с одним параметром, инициализирующий поля width и height указанными значениями, а остальные поля значениями по умолчанию;
* открытый конструктор с параметрами, инициализирующий все поля значениями по умолчанию;
* открытое свойство Side, возвращающее или задающее размер стороны квадрата;
* открытый переопределенный метод GetArea, возвращающий площадь квадрата;
* открытый переопределенный метод GetPerimeter, возвращающий периметр квадрата;
* открытый переопределенный метод ToString, возвращающий строку с описанием квадрата.

Определить интерфейс:

public interface IMovable

{

void MoveUp();

void MoveDown();

void MoveLeft();

void MoveRight();

}

Определить класс MovablePoint, обозначающий точку и реализующий интерфейс IMovable, который имеет:

* защищенное поле x типа int, обозначающее координату точки на оси Х'Х;
* защищенное поле y типа int, обозначающее координату точки на оси Y'Y;
* защищенное поле xSpeed типа int, обозначающее скорость движения точки по оси Х'Х;
* защищенное поле ySpeed типа int, обозначающее скорость движения точки по оси Y'Y;
* конструктор с параметрами, инициализирующий все поля класса;
* переопределенный (override) метод ToString возвращающий строку с описанием точки.
* Определить класс MovableCircle, обозначающий окружность и реализующий интерфейс IMovable, который имеет:
* закрытое поле radius типа int, обозначающее радиус окружности;
* закрытое поле center типа MovablePoint, обозначающее центр окружности;
* конструктор с параметрами, инициализирующий все поля класса;
* переопределенный (override) метод ToString возвращающий строку с описанием окружности.

Определить класс MovableRectangle, обозначающий прямоугольник и реализующий интерфейс IMovable, который имеет:

* закрытое поле topLeft типа MovablePoint, обозначающее верхнюю левую точку прямоугольника;
* закрытое поле bottomRight типа MovablePoint, обозначающее нижнюю правую точку прямоугольника;
* конструктор с параметрами, инициализирующий все поля класса;
* переопределенный (override) метод ToString возвращающий строку с описанием окружности.

using System;

namespace Laba\_1

{

public class Circle : Shape

{

protected double radius;

public double Radius {

get { return radius; }

set { radius = value; }

}

public Circle()

{

color = "BLACK";

filled = false;

radius = 1.0;

}

public Circle(double Radius)

{

radius = Radius;

}

public Circle(string Color, bool Filled, double Radius)

{

color = Color;

filled = Filled;

radius = Radius;

}

public override double GetArea()

{

double s = Math.PI \* Math.Pow(radius, 2);

return s;

}

public override double GetPerimeter()

{

double p = 2 \* Math.PI \* radius;

return p;

}

public override string ToString()

{

return $"Цвет: {color}, Закрашена: {filled}, Радиус: {radius}, Площадь: {GetArea()}, Периметр {GetPerimeter()}";

}

}

}

namespace Laba\_1

{

public interface IMovable

{

void MoveUp();

void MoveDown();

void MoveLeft();

void MoveRight();

}

}

namespace Laba\_1

{

public class MovableCircle : Circle, IMovable

{

private MovablePoint center;

public MovableCircle(int Radius, MovablePoint Center)

{

radius = Radius;

center = Center;

}

public void MoveUp()

{

center.MoveUp();

}

public void MoveDown()

{

center.MoveDown();

}

public void MoveLeft()

{

center.MoveLeft();

}

public void MoveRight()

{

center.MoveRight();

}

public override string ToString()

{

return $"Радиус {radius}, Центр {center}";

}

}

}

namespace Laba\_1

{

public class MovablePoint : Point, IMovable

{

protected int xSpeed;

protected int ySpeed;

public MovablePoint(int X, int Y, int XSpeed, int YSpeed)

{

x = X;

y = Y;

xSpeed = XSpeed;

ySpeed = YSpeed;

}

public void MoveUp()

{

y+= 1;

}

public void MoveDown()

{

y-= 1;

}

public void MoveLeft()

{

x-= 1;

}

public void MoveRight()

{

x+= 1;

}

public override string ToString()

{

return $"Координата точки на оси Х'Х и Y'Y соответственно: {x}:{y}, Cкорость движения точки по оси Х'Х и Y'Y соответственно: {xSpeed}:{ySpeed}";

}

}

}

namespace Laba\_1

{

public class MovableRectangle : Rectangle, IMovable

{

private MovablePoint topLeft;

private MovablePoint bottomRight;

public MovableRectangle(MovablePoint TopLeft, MovablePoint BottomRight)

{

topLeft = TopLeft;

bottomRight = BottomRight;

}

public void MoveUp()

{

topLeft.MoveUp();

bottomRight.MoveUp();

}

public void MoveDown()

{

topLeft.MoveDown();

bottomRight.MoveDown();

}

public void MoveLeft()

{

topLeft.MoveLeft();

bottomRight.MoveLeft();

}

public void MoveRight()

{

topLeft.MoveRight();

bottomRight.MoveRight();

}

public override string ToString()

{

return $"Верхняя левая точка {topLeft}, нижняя правая точка {bottomRight}";

}

}

}

namespace Laba\_1

{

public class Point

{

protected int x;

protected int y;

public Point()

{

x = 0;

y = 0;

}

public Point(int X, int Y)

{

x = X;

y = Y;

}

public override string ToString()

{

return $"X: {x}, Y: {y}";

}

}

}

using System;

namespace Laba\_1

{

class Program

{

static void Main(string[] args)

{

Circle c1 = new Circle("RED", false, 2);

Console.WriteLine(c1);

Console.WriteLine(c1.GetArea());

Console.WriteLine(c1.GetPerimeter());

Console.WriteLine(c1.Color);

Console.WriteLine(c1.IsFilled);

Console.WriteLine(c1.Radius);

Console.WriteLine();

Rectangle r1 = new Rectangle("RED", false, 2, 1);

Console.WriteLine(r1);

Console.WriteLine(r1.GetArea());

Console.WriteLine(r1.GetPerimeter());

Console.WriteLine(r1.Color);

Console.WriteLine(r1.IsFilled);

Console.WriteLine(r1.Width);

Console.WriteLine(r1.Height);

Console.WriteLine();

Square s1 = new Square(6.6);

Console.WriteLine(s1);

Console.WriteLine(s1.GetArea());

Console.WriteLine(s1.GetPerimeter());

Console.WriteLine(s1.Color);

Console.WriteLine(s1.IsFilled);

Console.WriteLine(s1.Side);

Console.WriteLine();

MovablePoint m1 = new MovablePoint(5, 6, 10, 10);

Console.WriteLine(m1);

m1.MoveLeft();

Console.WriteLine(m1);

Console.WriteLine();

MovableCircle m2 = new MovableCircle(2, new MovablePoint(5, 6, 10, 10));

Console.WriteLine(m2);

m2.MoveRight();

Console.WriteLine(m2);

Console.WriteLine();

MovableRectangle m3 = new MovableRectangle(new MovablePoint(10, 8, 10, 10), new MovablePoint(5, 6, 10, 10));

Console.WriteLine(m3);

m3.MoveRight();

Console.WriteLine(m3);

Console.ReadLine();

}

}

}

namespace Laba\_1

{

public class Rectangle : Shape

{

protected double width;

protected double height;

public double Width

{

get { return width; }

set { width = value; }

}

public double Height

{

get { return height; }

set { height = value; }

}

public Rectangle()

{

color = "BLACK";

filled = false;

width = 2.0;

height = 1.0;

}

public Rectangle(double Width, double Height)

{

width = Width;

height = Height;

}

public Rectangle(string Color, bool Filled, double Width, double Height)

{

color = Color;

filled = Filled;

width = Width;

height = Height;

}

public override double GetArea()

{

double s = width \* height;

return s;

}

public override double GetPerimeter()

{

double p = 2 \* (width + height);

return p;

}

public override string ToString()

{

return $"Цвет: {color}, Закрашена: {filled}, Высота: {height}, Ширина: {width}, Площадь: {GetArea()}, Периметр {GetPerimeter()}";

}

}

}

namespace Laba\_1

{

public abstract class Shape

{

protected string color;

protected bool filled;

public abstract double GetArea();

public abstract double GetPerimeter();

public string Color

{

get { return color; }

set { color = value; }

}

public bool IsFilled

{

get { return filled; }

set { IsFilled = filled; }

}

public Shape()

{

color = "BLACK";

filled = false;

}

public Shape(string Color, bool Filled)

{

color = Color;

filled = Filled;

}

public override string ToString()

{

return $"Цвет: {color}, Закрашена: {filled}";

}

}

}

using System;

namespace Laba\_1

{

public class Square : Rectangle

{

public double Side

{

get { return width; }

set { width = height = value; }

}

public Square()

{

color = "BLACK";

filled = false;

width = 1.0;

height = 1.0;

}

public Square(double WidthHeight)

{

color = "BLACK";

filled = false;

width = WidthHeight;

height = WidthHeight;

}

public Square(string Color, bool Filled, double WidthHeight)

{

color = Color;

filled = Filled;

width = WidthHeight;

height = WidthHeight;

}

public override double GetArea()

{

double s = Math.Pow(width, 2);

return s;

}

public override double GetPerimeter()

{

double p = 4 \* width;

return p;

}

public override string ToString()

{

return $"Цвет: {color}, Закрашена: {filled}, Сторона: {height}, Площадь: {GetArea()}, Периметр {GetPerimeter()}";

}

}

}