# **Exercise 1 – Shape**

using System;

using System.Collections.Generic;

namespace PRN292\_Lab1\_Exercise1

{

public class Point //Point object

{

private double x;

private double y;

public Point(double x, double y)

{

this.x = x;

this.y = y;

}

public double X { get => x; set => x = value; }

public double Y { get => y; set => y = value; }

public void movePoint(double moveX, double moveY)

{

x += moveX;

y += moveY;

}

public override string ToString()

{

return $"({x},{y})";

}

}

public abstract class Shape //base class

{

protected List<Point> points;

protected string name { get; set; }

public abstract void Show();

public void Move(double xAsis, double yAsis)

{

foreach (Point p in points)

{

p.movePoint(xAsis, yAsis);

}

}

public void Move(Point point, int xAsis, int yAsis)

{

if (points.Contains(point))

{

int i = points.IndexOf(point);

point.movePoint(xAsis, yAsis);

points[i] = point;

}

}

public string listPointToString()

{

String listPoint = "";

foreach (Point p in points)

{

listPoint += p.ToString() + ", ";

}

listPoint = listPoint.Substring(0, listPoint.Length - 2);

return listPoint;

}

public override string ToString()

{

return "Shape with" + points.Count + " point(s)";

}

}

class Line : Shape //Line

{

public Line(Point a, Point b)

{

points = new List<Point>();

points.Add(a);

points.Add(b);

name = "Line";

}

public override void Show()

{

Console.WriteLine("This is a " + name);

Console.WriteLine("\t - Points: " + base.listPointToString());

}

public override string ToString()

{

return "Line crosses two points: " + listPointToString();

}

}

public class Circle : Shape //circle

{

private double r { get; set; }

public Circle(Point o, double r)

{

points = new List<Point>();

points.Add(o);

this.r = r;

name = "Circle";

}

public override void Show()

{

Console.WriteLine("This is a {0} with center {1} and radius {2:F2}", name, listPointToString(), r);

}

public override string ToString()

{

return "Circle: r = " + r + " and " + "center = " + listPointToString();

}

}

public class Rectangle : Shape //rectangle

{

public Rectangle(Point p1, Point p2, Point p3)

{

points = new List<Point>();

points.Add(p1);

points.Add(p2);

points.Add(p3);

name = "Rectangle";

}

public override void Show()

{

Console.WriteLine("This is a " + name);

Console.WriteLine("\t - Points: " + base.listPointToString());

}

public override string ToString()

{

return "Rectangle: points: " + listPointToString();

}

}

public class PolyLine : Shape //polyline

{

public PolyLine(List<Point> list)

{

points = new List<Point>();

foreach (Point p in list)

{

base.points.Add(p);

}

name = "PolyLine";

}

public override void Show()

{

Console.WriteLine("This is a " + name);

Console.WriteLine("\t - Points: " + base.listPointToString());

}

public override string ToString()

{

return "PolyLine: " + base.listPointToString();

}

}

class shapeMain //class tester

{

static int inputInt(string s)

{

while (true)

{

try

{

Console.Write(s);

int rs = int.Parse(Console.ReadLine());

return rs;

}

catch (Exception ex)

{

Console.Write("Invalid input, try again! \n" + ex);

}

}

}

static double inputDouble(string s)

{

while (true)

{

try

{

Console.Write(s);

double rs = Double.Parse(Console.ReadLine());

return rs;

}

catch (Exception ex)

{

Console.Write("Invalid input, try again! \n" + ex);

}

}

}

static Point inputPoint(string s)

{

while (true)

{

try

{

Console.WriteLine(s);

double x = inputDouble("x = ");

double y = inputDouble("y = ");

return new Point(x, y);

}

catch (Exception ex)

{

Console.Write("Invalid input, try again!\n" + ex);

}

}

}

static void Test(Shape shape)

{

shape.Show();

shape.Move(2, 1);

Console.WriteLine("After move (2,1) ");

shape.Show();

}

static void Line()

{

Console.WriteLine("-------- Line --------");

Point p1 = inputPoint("Point 1: ");

Point p2 = inputPoint("Point 2: ");

Test(new Line(p1, p2));

}

static void Circle()

{

Console.WriteLine("-------- Cirlce --------");

Point o = inputPoint("Point center: ");

double r = inputDouble("r= ");

Test(new Circle(o, r));

}

static void Rectangle()

{

Console.WriteLine("-------- Rectangle --------");

Point p1 = inputPoint("Point 1: ");

Point p2 = inputPoint("Point 2: ");

Point p3 = inputPoint("Point 3: ");

Test(new Rectangle(p1, p2, p3));

}

static void PolyLine()

{

Console.WriteLine("-------- PolyLine --------");

Console.WriteLine("Number of points");

int n = inputInt("\tn = ");

if (n < 2)

{

Console.WriteLine("PolyLine must contain more than 1 point.");

return;

}

List<Point> list = new List<Point>();

for (int i = 0; i < n; i++)

{

Point p = inputPoint("Point " + (i+1) + ": ");

list.Add(p);

}

Test(new PolyLine(list));

}

static void Main(string[] args)

{

Line();

Circle();

Rectangle();

PolyLine();

Console.ReadLine();

}

}

}

# **Exercise 2 – Animal**

using System; //class main

namespace PRN292\_Lab1\_Exercise2

{

//animal

public abstract class Animal

{

protected string type;

protected string name;

protected Animal(string type, string name)

{

this.type = type;

this.name = name;

}

internal string Type { get => type; set => type = value; }

internal string Name { get => name; set => name = value; }

public abstract string getSound();

public abstract string getInformation();

public override string ToString()

{

return name + " - " + type;

}

}

//dog

public class Dog : Animal

{

private string breed; //one of values such as Spaniel, Chihuahua, and Collie,...

public Dog(string breed, string type, string name) : base(type, name)

{

this.breed = breed;

}

internal string Breed { get => breed; set => breed = value; }

public override string getInformation()

{

return "loyal to the owner.";

}

public override string getSound()

{

return "woow woow";

}

public override string ToString()

{

return "Dog [ type : " + type + " - name : " + name + " - breed : " + breed + "]\n=> says " + getSound() + " and " + getInformation();

}

}

//cat

public class Cat : Animal

{

public Cat(string type, string name) : base(type, name) { }

public string climb()

{

return "tree";

}

public override string getInformation()

{

return "very painful pawns";

}

public override string getSound()

{

return "meo meo";

}

public override string ToString()

{

return "Cat [ type : " + type + " - name : " + name + " - climb : " + climb() + "]\n=> says " + getSound() + " and " + getInformation();

}

}

//duck

public class Duck : Animal

{

public Duck(string type, string name) : base(type, name) { }

public string swim()

{

return "lake";

}

public override string getInformation()

{

return "have wings";

}

public override string getSound()

{

return "cap cap";

}

public override string ToString()

{

return "Duck [ type : " + type + " - name : " + name + " - swim : " + swim() + "]\n=> says " + getSound() + " and " + getInformation();

}

}

//main class

class animalMain

{

static void Main(string[] args)

{

Dog dog = new Dog("Chollie", "Mammal", "Kiki");

Cat cat = new Cat("Mammal", "Milu");

Duck duck = new Duck("Bird", "Donald");

Console.WriteLine(dog.ToString());

Console.WriteLine("\n" + cat.ToString());

Console.WriteLine("\n" + duck.ToString());

Console.ReadLine();

}

}

}