HOME WORK 25/08

Full Name: Lê Hữu Hoàn

Student ID: 20IT684  
Class : C# development (3)

Exercise 1:

 Define an abstract base class Shape that includes protected data members for the (x, y) position of a shape, a public method to move a shape, and a public abstract method Show () to output information

of a shape.

 Derive subclasses for lines, circles, and rectangles. You can represent

a line as two points, a circle as a center and a radius, and a rectangle as three points.

 Also, define the class PolyLine with Shape as its base class.

 Implement the ToString() method for each class.

 Test the classes by creating objects of the derived classes, and then invoking methods for each.

Console:   
A screenshot of a computer

Description automatically generated

Code:

|  |
| --- |
| using System;  namespace OOP\_02  {  internal class Program  {  static void Main(string[] args)  {  Line line = new Line(0, 0, 5, 5);  line.Show();  Circle circle = new Circle(3, 3, 5);  circle.Show();  Rectangle rect = new Rectangle(0, 0, 4, 0, 4, 2);  rect.Show();  PolyLine polyline = new PolyLine(1, 1);  polyline.Show();  }  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02  {  public abstract class Shape  {  protected int x;  protected int y;  public Shape(int x, int y)  {  this.x = x;  this.y = y;  }  public void Move(int dx, int dy)  {  x += dx;  y += dy;  }  public abstract void Show();  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02  {  public class Rectangle : Shape  {  private int x2;  private int y2;  private int x3;  private int y3;  public Rectangle(int x1, int y1, int x2, int y2, int x3, int y3) : base(x1, y1)  {  this.x2 = x2;  this.y2 = y2;  this.x3 = x3;  this.y3 = y3;  }  public override void Show()  {  Console.WriteLine($"Rectangle: Points ({x}, {y}), ({x2}, {y2}), ({x3}, {y3})");  }  public override string ToString()  {  return $"Rectangle: Points ({x}, {y}), ({x2}, {y2}), ({x3}, {y3})";  }  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02  {  public class PolyLine : Shape  {  public PolyLine(int x, int y) : base(x, y)  {  }  public override void Show()  {  Console.WriteLine($"PolyLine: ({x}, {y})");  }  public override string ToString()  {  return $"PolyLine: ({x}, {y})";  }  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02  {  public class Line : Shape  {  private int x2;  private int y2;  public Line(int x1, int y1, int x2, int y2) : base(x1, y1)  {  this.x2 = x2;  this.y2 = y2;  }  public override void Show()  {  Console.WriteLine($"Line: ({x}, {y}) to ({x2}, {y2})");  }  public override string ToString()  {  return $"Line: ({x}, {y}) to ({x2}, {y2})";  }  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02  {  public class Circle : Shape  {  private int radius;  public Circle(int x, int y, int radius) : base(x, y)  {  this.radius = radius;  }  public override void Show()  {  Console.WriteLine($"Circle: Center ({x}, {y}), Radius: {radius}");  }  public override string ToString()  {  return $"Circle: Center ({x}, {y}), Radius: {radius}";  }  }  } |

Exercise 2:

 Define an abstract class Animal which has the following members:

- Type is used to identify the type of animal (mammal, bird), and its value is set by the constructor.

- A method returns sound made by animal.

- A method returns information of animal.

 Define classes, which derive from Animal, for Dog, Cat and Duck. These classes have Name attribute. Besides, each class has more some other members such as:

- Dog class has Breed attribute. This attribute can be one of values such as Spaniel, Chihuahua, and Collie.

- Cat has Climb method which indicates that cat is climbing something (tree, wall, roof, etc).

 Duck has Swim method which indicates that duck is swimming something (pond, pool, etc).

- Test the classes by creating objects of the derived classes, and then invoking methods for each.

A computer screen shot of a black screen

Description automatically generated

Code:

|  |
| --- |
| using System;  namespace OOP\_02\_02  {  internal class Program  {  static void Main(string[] args)  {  Dog dog = new Dog("Max", "Spaniel");  Console.WriteLine(dog.MakeSound());  Console.WriteLine(dog.GetInfo());  Console.WriteLine();  Cat cat = new Cat("Whiskers");  Console.WriteLine(cat.MakeSound());  Console.WriteLine(cat.GetInfo());  cat.Climb("a tree");  Console.WriteLine();  Duck duck = new Duck("Donald");  Console.WriteLine(duck.MakeSound());  Console.WriteLine(duck.GetInfo());  duck.Swim("a pond");  }  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02\_02  {  abstract class Animal  {  public string Type { get; }  public Animal(string type)  {  Type = type;  }  public abstract string MakeSound();  public abstract string GetInfo();  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02\_02  {  class Dog : Animal  {  public string Name { get; }  public string Breed { get; }  public Dog(string name, string breed) : base("Mammal")  {  Name = name;  Breed = breed;  }  public override string MakeSound()  {  return "Woof!";  }  public override string GetInfo()  {  return $"Type: {Type}, Name: {Name}, Breed: {Breed}";  }  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02\_02  {  class Cat : Animal  {  public string Name { get; }  public Cat(string name) : base("Mammal")  {  Name = name;  }  public override string MakeSound()  {  return "Meow!";  }  public override string GetInfo()  {  return $"Type: {Type}, Name: {Name}";  }  public void Climb(string objectToClimb)  {  Console.WriteLine($"The cat named {Name} is climbing {objectToClimb}.");  }  }  } |

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace OOP\_02\_02  {  class Duck : Animal  {  public string Name { get; }  public Duck(string name) : base("Bird")  {  Name = name;  }  public override string MakeSound()  {  return "Quack!";  }  public override string GetInfo()  {  return $"Type: {Type}, Name: {Name}";  }  public void Swim(string placeToSwim)  {  Console.WriteLine($"The duck named {Name} is swimming in {placeToSwim}.");  }  }  } |