SOLUCIÓN EXERCIZOS TEMA5-A01

1.1 Tarefas

1.1.1 Tarefa 1. Identificación de elementos da programación orientada a obxectos (POO)

Nos seguintes exemplos de código identifica todos os conceptos e termos que **podas** dos vistos nos apuntamentos: clase, obxecto, construtor, atributos, métodos, encapsulado de campos, abstracción, encapsulamento, herdanza, polimorfismo, sobrecarga, envío de mensaxes, agregación/composición, xeneralización/especialización, asociación.

Por cada un dos que atopes copia e pega algún anaco de código ou ben explica o concepto utilizando os anacos de código involucrados.

- **Código 1**

```
using System;
using System.Windows.Forms;
namespace ClasesAbstractas
{
    abstract class FiguraGeometrica
    {
        public abstract double Area();
    }
   class Circulo : FiguraGeometrica
    {
        private int radio;
        public int Radio
        {
            get
            {
                return radio;
            }
            set
            {
                radio = value;
            }
        }
        public Circulo(int r)
        {
            Radio = r;
        }
        public override double Area()
```

```
    {
        return (double)(Radio * Radio * 3.14);
    }
}
class Triangulo : FiguraGeometrica
{
    private int ladA;
    private int ladB;
    private int ladC;
    public int LadoA
    {
        get
        {
            return ladA;
        }
        set
        {
            ladA = value;
        }
    }
    public int LadoB
    {
        get
        {
            return ladB;
        }
        set
        {
            ladB = value;
        }
    }
    public int LadoC
    {
        get
        {
            return ladC;
        }
        set
        {
            ladC = value;
        }
    }
    public Triangulo(int a, int b, int c)
    {
        LadoA = a;
        LadoB = b;
```

```
            LadoC = c;
        }
        public override double Area()
        {
            double num, mPer;
            mPer = (double)(LadoA + LadoB + LadoC) / 2;
            num = mPer * (mPer - LadoA) * (mPer - LadoB) * (mPer
- LadoC);
            if (num <= 0)
            {
                MessageBox.Show("posiblemente el triangulo no
exista, intentelo nuevamente");
                return 0;
            }
            else
                return Math.Pow(num, 0.5);
        }
    }
    class Cuadrado : FiguraGeometrica
    {
        private int lado;
        public int Lado
        {
            get
            {
                return lado;
            }
            set
            {
                lado = value;
            }
        }
        public Cuadrado(int l)
        {
            Lado = l;
        }
        public override double Area()
        {
            return Lado * Lado;
        }
    }
}
```

- **Código 2**

```
class Clase1 {
```

```
public void prueba()
{
        System.out.println("\n Metodo sin Argumentos:()");
}
public void prueba(int x)
{
        System.out.print("\n Metodo con un Argumento:");
        System.out.println(" x= " +x);
}
public void prueba(double x, double y, double z)
{
        System.out.print("\n Metodo con tres Argumentos:");
        System.out.println(" x= " +x+ "    y= " + y + "    z= " +
z + "\n");
}
}
class UsarClase1 {
        public static void main (String [] var)
        {
                Clase1 objeto= new Clase1();
                objeto.prueba();
                objeto.prueba(30);
                objeto.prueba(-3.5,20.0,18.6);
        }
}
```

Solución

- **Clases**.- Serían todas aquelas liñas de código nas que se definen as clases.
  - Para o código 1:

```
abstract class FiguraGeometrica
class Circulo : FiguraGeometrica
class Triangulo : FiguraGeometrica
class Cuadrado : FiguraGeometrica
```

  - Para o código 2:

```
class Clase1 {
class UsarClase1 {
```

- **Obxectos**.- Un *obxecto* é a instancia dunha *clase* polo que deberemos de buscar as liñas onde se instancien novos obxectos:
  - Para o código 1: *Non existen.*
  - Para o código 2:

```
Clase1 objeto= new Clase1();
```

- **Construtores**.- O *construtor* dunha clase é un método estándar para instanciar os obxectos desa clase polo que deberemos de buscar no código os métodos para crear novos obxectos:
  - Para o código 1:

```
public Circulo(int r)
{
    Radio = r;
}
public Triangulo(int a, int b, int c)
{
    LadoA = a;
    LadoB = b;
    LadoC = c;
}
public Cuadrado(int l)
{
    Lado = l;
}
```

  - Para o código 2: *Non Existen.*
- **Atributos**.- Os *atributos* se implementan mediante variables que almacenarán os datos específicos do obxecto:
  - Para o código 1:

```
private int radio;
private int ladA;
private int ladB;
private int ladC;
private int lado;
```

  - Para o código 2: *Non existen.*
- **Métodos**.- Os *métodos* se implementan mediante funcións ou procedementos que poderemos invocar para realizar operacións específicas cos obxectos:
  - Para o código 1:

```
public override double Area()
{
    double num, mPer;
    mPer = (double)(LadoA + LadoB + LadoC) / 2;
    num = mPer * (mPer - LadoA) * (mPer - LadoB) * (mPer
- LadoC);
    if (num <= 0)
    {
        MessageBox.Show("posiblemente el triangulo no
exista, intentelo nuevamente");
```

```
            return 0;
        }
        else
            return Math.Pow(num, 0.5);
    }
    public override double Area()
    {
        return (double)(Radio * Radio * 3.14);
    }
    public override double Area()
    {
        return Lado * Lado;
    }
```

– Para o código 2:

```
public void prueba()
{
        System.out.println("\n Metodo sin Argumentos:()");
}
public void prueba(int x)
{
        System.out.print("\n Metodo con un Argumento:");
        System.out.println(" x= " +x);
}
public void prueba(double x, double y, double z)
{
        System.out.print("\n Metodo con tres Argumentos:");
        System.out.println(" x= " +x+ "    y= " + y + "    z= " +
z + "\n");
}
        public static void main (String [] var)
        {
            Clase1 objeto= new Clase1();
            objeto.prueba();
            objeto.prueba(30);
            objeto.prueba(-3.5,20.0,18.6);
        }
```

- **Encapsulado de campos**.- Métodos especiais que nos permitirán controlar o acceso aos atributos e realizar comprobacións adicionais.
  – Para o código 1:

```
    public int Radio
    {
        get
        {
```

```
            return radio;
        }
        set
        {
            radio = value;
        }
    }
    public int LadoA
    {
        get
        {
            return ladA;
        }
        set
        {
            ladA = value;
        }
    }
    public int LadoB
    {
        get
        {
            return ladB;
        }
        set
        {
            ladB = value;
        }
    }
    public int LadoC
    {
        get
        {
            return ladC;
        }
        set
        {
            ladC = value;
        }
    }
    public int Lado
    {
        get
        {
            return lado;
        }
```

```
            set
            {
                lado = value;
            }
        }
```

  - Para o código 2: *Non Existen.*

- **Abstracción** (e encapsulamento). O concepto de *abstracción* a nivel de código conséguese mediante a utilización de *interfaces* ou clases *abstractas*:
  - Para o código 1:

```
abstract class FiguraGeometrica
{
    public abstract double Area();
}
```

  - Para o código 2: *Non Existen.*

- **Herdanza.** Cando facemos que una clase sea filla "doutra" :
  - Para o código 1:

```
class Circulo : FiguraGeometrica
class Triangulo : FiguraGeometrica
class Cuadrado : FiguraGeometrica
```

  - Para o código 2: *Non Existen.*

- **Polimorfismo.** O *polimorfismo* refírese á capacidade para que varias clases derivadas dunha antecesora implementen un mesmo método de forma diferente:
  - Para o código 1:

```
        public override double Area()
        {
            return (double)(Radio * Radio * 3.14);
        }
        public override double Area()
        {
            double num, mPer;
            mPer = (double)(LadoA + LadoB + LadoC) / 2;
            num = mPer * (mPer - LadoA) * (mPer - LadoB) * (mPer
- LadoC);
            if (num <= 0)
            {
                MessageBox.Show("posiblemente el triangulo no
exista, intentelo nuevamente");
                return 0;
            }
```

```
            else
                return Math.Pow(num, 0.5);
        }
        public override double Area()
        {
            return Lado * Lado;
        }
```

– Para o código 2: *Non Existen.*

- **Sobrecarga de métodos.** Permite definir dous ou máis métodos co mesmo nome, pero que difieren en cantidade ou tipo de parámetros:
  – Para o código 1: *Non Existen.*
  – Para o código 2:

```
public void prueba()
{
       System.out.println("\n Metodo sin Argumentos:()");
}
public void prueba(int x)
{
       System.out.print("\n Metodo con un Argumento:");
       System.out.println(" x= " +x);
}
public void prueba(double x, double y, double z)
{
       System.out.print("\n Metodo con tres Argumentos:");
       System.out.println(" x= " +x+ "    y= " + y + "    z= " +
z + "\n");
}
```

- **Envío de mensaxes.** O xeito de enviar unha mensaxe a un obxecto desde outro é mediante a invocación dun dos seus métodos.
  – Para o código 1:

```
                MessageBox.Show("posiblemente el triangulo no
exista, intentelo nuevamente");
                return Math.Pow(num, 0.5);
```

– Para o código 2:

```
       System.out.println("\n Metodo sin Argumentos:()");
       System.out.print("\n Metodo con un Argumento:");
       System.out.println(" x= " +x);
       System.out.print("\n Metodo con tres Argumentos:");
```

```
        System.out.println(" x= " +x+ "    y= " + y + "    z= " +
z + "\n");
                objeto.prueba();
                objeto.prueba(30);
                objeto.prueba(-3.5,20.0,18.6);
```

- **Agregación/composición.** Neste tipo de relacións un obxecto compoñente intégrase nun obxecto composto.
  - Para o código 1:*Non existen.*
  - Para o código 2: *Non existen.*
- **Xeneralización/Especialización**. A xeneralización e a especialización son diferentes perspectivas do mesmo concepto, a xeneralización é unha perspectiva ascendente, mentres que a especialización é unha perspectiva descendente: unha superclase representa unha xeneralización das subclases e unha subclase representa unha especialización da clase superior.
  - *Véxase Herdanza*
- **Asociación**. Serían relacións xerais, nas que un obxecto realiza chamadas aos métodos doutro, interactuando con el.
  - Para o código 1:*Non existen.*
  - Para o código 2: *Non existen.*