Tema: FUNDAMENTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS (P.O.O)

Objetivo:

- ✓ Conocer y aplicar correctamente los conceptos fundamentales de la programación Orientada a Objetos (P.O.O) en el lenguaje de Programación C#.

I. CONCEPTOS BASICOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

1. Clase: Una clase es el esquema o definición desde donde se obtienen objetos. Una clase es una representación abstracta de un elemento o entidad de la realidad que contiene la definición de las características y comportamientos esenciales del objeto que representa. Una clase consta de tres elementos básicos:

- ✓ **Nombre:** Está asociado o relacionado con el objeto que representa. Por lo general el nombre de una clase lleva la primera letra en mayúscula.

- ✓ **Atributos:** Son las características particulares del objeto representado. Los atributos constituyen los datos asociados al objeto, es decir, sus propiedades o su información más representativa.

- ✓ **Métodos:** Son funciones que modelan el comportamiento de un objeto. Los métodos permiten ejecutar funcionalidades o realizar cambios en la información almacenadas en los atributos.

2. Modificadores De Acceso (Visibilidad): En la P.O.O tanto métodos como atributos tienen una visibilidad, la cual se implementa usando uno de los siguientes modificadores de acceso:

- ✓ **Acceso o Visibilidad Pública (+):** representada por un signo “más”; los métodos y atributos pueden ser vistos por todos los objetos del sistema.

- ✓ **Acceso o Visibilidad Privada (-):** representada por el signo “menos”; los métodos y tributos solo pueden ser vistos por el objeto al cual pertenecen.

- ✓ **Acceso o Visibilidad Protegida (#):** representada por el signo “número”; los métodos y atributos pueden ser vistos por el objeto que los contiene y sus hijos (Cuando hay Herencia).

3. Constructor: Es un método cuya función principal es la de crear un nuevo objeto a partir de una clase. Este método puede estar definido de forma implícita o explícita y adicionalmente se puede programar para que simultáneamente al crear el objeto se le establezca un estado inicial. En la P.O.O los constructores tienen el mismo nombre que el de su clase.

4. Objeto: Un objeto se define como la unidad que en tiempo de ejecución tiene la capacidad de realizar tareas o ejecutar funcionalidades por medio del llamado a métodos. En P.O.O a los objetos se le conocen como “Instancias de una clase”. Para utilizar las funcionalidades definidas en una clase es obligatorio primero crear un objeto. Únicamente cuando el objeto es creado se pueden llamar y ejecutar métodos.

En la P.O.O los objetos poseen 3 características o propiedades importantes las cuales son:

- ✓ ***Identidad:*** La identidad es la propiedad que permite a un objeto diferenciarse de otros. La identidad se refiere al identificador único que debe tener cada objeto en un contexto a ámbito dado, esta propiedad se aplica por medio del nombre que se le da a los objetos.

- ✓ ***Estado****:* El estado de un objeto corresponde a la información almacenada en los atributos del objeto en un instante de tiempo dado. Los atributos y sus valores en un momento dado, determinan el estado de un objeto.

- ✓ ***Comportamiento:*** El comportamiento de un objeto se refiere al conjunto de tareas o funcionalidades que este es capaz de realizar. El comportamiento de un objeto está directamente relacionado con su funcionalidad y por ende, está definido por los métodos de su clase.

II. PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

Un software orientado a objetos se basa en modelos o representaciones abstractas de un problema o sistema real. Un software orientado a objetos es aquel que se construye basado en los conceptos y principios del paradigma de la programación orientada a objetos. Los pilares fundamentales de la programación orientada a objetos son:

- ✓ ***Abstracción:*** La abstracción es la propiedad que permite representar las características (atributos) y comportamientos (métodos) esenciales de un objeto. La abstracción en la P.O.O permite realizar un modelo computacional, en el cual por medio de una abstracción de datos se crea un T.D.A (Tipo De Datos Abstracto) que contiene un conjunto de atributos y métodos que representan computacionalmente un objeto real.

- ✓ ***Encapsulamiento:*** El *Encapsulamiento* o *encapsulación* es un principio de la P.O.O que permite asegurar que el contenido de la información almacenada en los atributos de un objeto permanezca oculta. En un objeto cada atributo solo le pertenece a él y solo puede ser manejado por él a través de sus métodos. Cuando se presenta encapsulamiento los atributos solo pueden ser accedidos a través de métodos públicos definidos dentro de la misma clase.

- ✓ ***Modularidad:*** La Modularidad es la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las otras partes. En P.O.O una clase o un conjunto de clases pueden considerarse como un módulo de una aplicación.

- ✓ ***Agregación/Composición:*** Son relaciones entre clases de tipo jerárquica que se utilizan para expresar que un objeto presenta dependencia hacia otros objetos para llevar a cabo su función, de modo que uno de los objetos involucrados está *"formado"* por los otros. Aunque son conceptos similares la agregación y la composición poseen algunas diferencias de tipo conceptual.

  La *agregación* es una relación de tipo dinámica donde el tiempo de vida de los objetos que se comportan como "partes" es independiente del tiempo de vida del objeto conformado, es decir, se puede crear el objeto y luego agregarle o asignarle las partes. Por ejemplo un auto tiene llantas, motor, caja de cambios, puertas, etc.

  La *composición* en cambio es una relación de tipo estática donde el tiempo de vida de los objetos que se comportan como "partes" dependen del tiempo de vida del objeto conformado, es decir, si desaparece el objeto conformado también desaparecen las partes, el objeto componente no podría existir solo sin el objeto compuesto.. Por ejemplo una Ventana (I.GU) de un programa tiene botones, cuadro de textos, etiquetas, menús, etc.

- ✓ ***Herencia:*** Es una de las propiedades más importantes de la P.O.O. La herencia se puede definir como la capacidad que tiene una clase (llamada SubClase) de utilizar las características y comportamientos de otra clase (llamada clase padre o SuperClase. La herencia permite a los

objetos ser construidos extendiendo y/o reutilizando las funcionalidades ya definidas en otros, esto es posible gracias a que en un programa orientado a objetos cuando se usa la herencia una clase superior puede transferir sus atributos y métodos a otras clases de jerarquía inferior.

- ✓ ***Polimorfismo:*** Es la propiedad que indica la posibilidad de que un objeto tome "*muchas formas"*. En términos prácticos, el polimorfismo permite referirse a objetos y realizar la misma operación de diferentes maneras, es decir, llamar un mismo método y que este se comporte de formas diferentes de acuerdo al contexto o ámbito donde se utilice. El polimorfismo se implementa en la P.O.O usando recursos tales como la sobrecarga y la sobreescritura de métodos.

- ✓ ***Asociación:*** La asociación se podría definir como el momento en que dos objetos se unen para trabajar juntos y así alcanzar una meta o lograr un objetivo. En una asociación los objetos involucrados son independientes entre sí.

III. PROGRAMACIÓN ORIENTADA A OBJETOS EN LENGUAJE C#

Para declarar una clase en lenguaje C# se escribe primero la palabra *public*, seguida de la palabra reservada *class* y luego el nombre de la clase. Los Nombres de clases, atributos y métodos deben ser nombres nemotécnicos, es decir, fácil de recordar, sin espacios ni signos de puntuación y donde no se usen palabras reservadas por el lenguaje (int, string, if, for, while, etc),.

Ejemplo:

```
public class nombreclase
{ }
```

- ✓ Para declarar los atributos se debe anteponer primero el tipo de visibilidad (público, privado, protegido), luego el tipo de dato que representa el atributo y después el nombre.

Ejemplo:

```
private string nombreatributo;
```

- ✓ Para declarar un método se debe primero colocar el tipo de visibilidad, luego el tipo de retorno que hará el método y después el nombre, luego paréntesis y dentro de ellos los parámetros que deba llevar. El código del método se establece dentro de llaves.

Ejemplo:

```
public void nombremetodo(string atributo1, double atributo2)
{
        Codigo: cuerpo del método;
}
```

La estructura completa de una clase en términos generales seria por ejemplo así:

```
public class Ejemplo
{
        private string atributo1;
        private double atributo2;
        private int atributo3;
        public void metodo1(int parametro1, double parámetro2)
        {
                Cuerpo_del_metodo1;
```

```
        }

        public string metodo2( )
        {
                Cuerpo_del_metodo2;
                return "tipo String";
        }
}
```

Los atributos se pueden declarar de cualquier tipo de dato primitivo (int, double, string) o tipos complejos como arreglos o estructuras. Los métodos pueden presentarse de cuatro tipos: Con o sin parámetros de entrada, de retorno o de no retorno. Los métodos de no retorno deben llevar la palabra *void*. En los métodos con retorno o los que reciben parámetros se deben especificar el tipo de dato que retorna o los que recibe; estos pueden ser tipos de datos primitivos, tipos complejos como arreglos o estructuras o inclusive otros objetos.

Para crear un objeto partir de una determinada clase se llama el método constructor haciendo uso de la palabra clase "new". La sintaxis general para crear un nuevo objeto en lenguaje C# es:

<NombreClase> <NombreObjeto> = `new` <NombreClase ( )>;

Ejemplos:

1. Crear un nuevo objeto en una sola línea de código:

```
Estudiante ObjEstudiante = new Estudiante( );
```

2. Crear un nuevo objeto usando dos líneas de código:

```
Operaciones ObjOperacion;
ObjOperacion = new Operaciones( );
```

Para llamar o ejecutar métodos es obligatorio primero crear un objeto. En el lenguaje de programación C#, el operador "punto" (.) es usado para referirse o "llamar" a un método particular de un objeto.

Ejemplos:

1. Llamado a un método sin parámetros de entrada y no retorno:
   ObjEstudiante.Matricular( );

2. Llamado a un método con parámetros de entrada y no retorno:
   ObjEstudiante.SetNombre("Pedro");

3. Llamado a método sin parámetros de entrada y retorno:
   string nombre = ObjEstudiante.GetNombre( );

4. Llamado a método con parámetros de entrada y retorno:
   int suma = ObjOperacion.Sumar(15,35);

EJERCICIO P.O.O N°1
(ABSTRACCIÓN Y ENCAPSULAMIENTO)

1. Cree un nuevo proyecto tipo “Windows Forms” C#, agregue un formulario nuevo y colóquele “Principal” como nombre.

2. En tiempo de diseño, diseñe su formulario usando los controles MenuStrip, ToolStrip y StatusStrip.

3. Configure el “MenuStrip para que muestre el menú de opciones con la siguiente distribución:

1. Menú Archivo
   - ✓ Salir
2. Menú Ventana
   - ✓ Cascada
   - ✓ Mosaico Vertical
   - ✓ Mosaico Horizontal
   - ✓ Cerrar Todo
3. Menú Ejercicios
   - ✓ Ejercicio POO 1
   - ✓ Ejercicio POO 2
4. Menú Ayuda
   - ✓ Acerca De

4. Programe las opciones del menú tal y como se vio en la guía de “Menus y Submenus (C#)”.

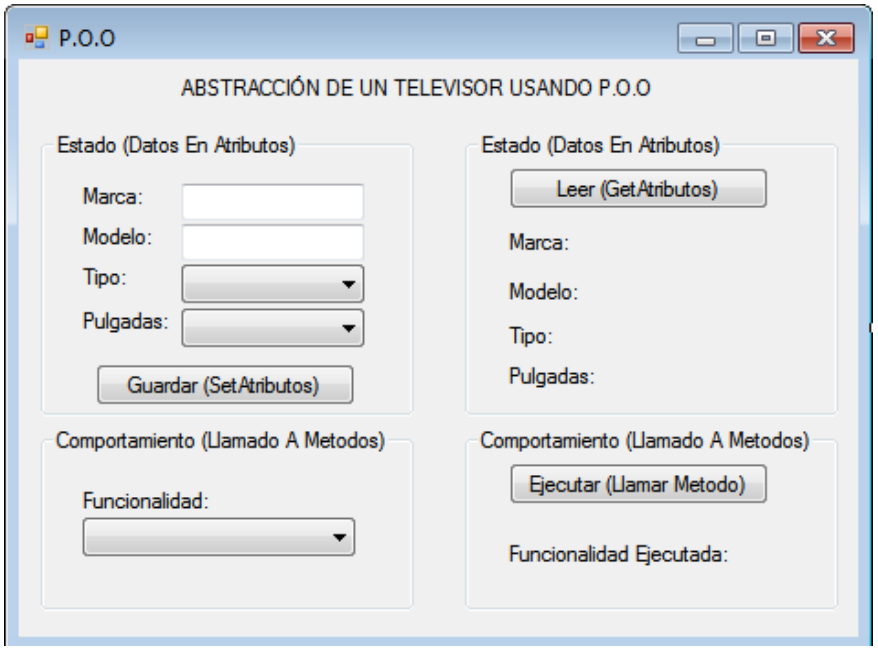
5. Cambie el valor la propiedad del formulario Principal llamada “IsMdiContainer” de false a true

6. Presione doble click en los submenús Ejercicio POO 1 y Ejercicio POO 2 y escriba el siguiente código:

```
private void ejercicioPOO1ToolStripMenuItem_Click(object…)
      {
          Form1 FormPOO1 = new Form1();
          FormPOO1.MdiParent = this;
          FormPOO1.Show();
      }
private void ejercicioPOO2ToolStripMenuItem_Click (object…  )
      {   Form2 FormPOO2 = new Form2();
          FormPOO2.MdiParent = this;
          FormPOO2.Show();
      }
```

7. Verifique en la clase “program.cs” que el formulario inicial corresponda al Principal. ***Ejemplo:***
`Application.Run(new Principal());`

8. En tiempo de diseño desarrolle el formulario1 tal y como se muestra a continuación:



Nota: A cada comboxBox cámbiele la propiedad DropDownStyle a DropDownList

9. Agregue una carpeta llamada "Clases" desde el explorador de soluciones. Dentro de esta carpeta agregue una nueva Clase y llámela "Televisor". Ubique cada bloque de código para escribir las instrucciones según se muestra a continuación:

```
namespace Clases
{
    public class Televisor
    {
     #region Atributos
        private string marca;
        private string modelo;
        private string tipo;
        private int pulgadas;
     #endregion

     #region Metodos
        public void SetMarca(string  _marca)
        { this.marca = _marca; }

        public string GetMarca()
        { return this.marca; }

        public void SetModelo(string  _modelo)
        { this.modelo = _modelo;}

        public string GetModelo()
        { return this.modelo; }
```

```
        public void SetTipo(string _tipo)
        { this.tipo = _tipo; }

        public string GetTipo()
        { return this.tipo; }

        public void SetPulgadas(int _pulgadas)
        { this.pulgadas = _pulgadas; }

        public int GetPulgadas()
        { return this.pulgadas; }

        public string Encender()
        { return "El Televisor ha sido encendido"; }

        public string Apagar()
        { return "El Televisor ha sido apagado"; }

        public string CambiarCanal()
        { return "El canal sintonizado ha sido cambiado"; }

        public string SubirVolumen()
        { return "El volumen se ha subido"; }

        public string BajarVolumen()
        { return "El volumen se ha bajado"; }

     #endregion
    }
}
```

10. En el Formulario1 Ubique cada bloque de código (Controles y Eventos) para escribir las instrucciones según se muestra a continuación:

```
using Clases; //Agregue el espacio de nombres que corresponde

    public partial class Form1 : Form
    {
        Televisor ObjTV = new Televisor();

        public Form1()
        { InitializeComponent();}

        private void Form1_Load(object sender, EventArgs e)
        {
            comboBox1.Items.Add("CRT");
            comboBox1.Items.Add("LCD");
            comboBox1.Items.Add("LED");

            comboBox2.Items.Add("14");
            comboBox2.Items.Add("15");
            comboBox2.Items.Add("22");
            comboBox2.Items.Add("26");
```

```
        comboBox2.Items.Add("32");
        comboBox2.Items.Add("40");
        comboBox2.Items.Add("50");

        comboBox3.Items.Add("Encender");
        comboBox3.Items.Add("Apagar");
        comboBox3.Items.Add("CambiarCanal");
        comboBox3.Items.Add("SubirVolumen");
        comboBox3.Items.Add("BajarVolumen");
    }

    private void button1_Click(object sender, EventArgs e)
    {
        ObjTV.SetMarca(textBox1.Text);
        ObjTV.SetModelo(textBox2.Text);
        ObjTV.SetTipo(comboBox1.Text);
        ObjTV.SetPulgadas(int.Parse(comboBox2.Text));
    }

    private void button3_Click(object sender, EventArgs e)
    {
        label7.Text = "Marca: " + ObjTV.GetMarca();
        label8.Text = "Modelo: " + ObjTV.GetModelo();
        label9.Text = "Tipo: " + ObjTV.GetTipo();
        label10.Text = "Pulgadas: " + ObjTV.GetPulgadas();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        switch (comboBox3.Text)
        {
            case "Encender":
                {
                    label11.Text = ObjTV.Encender();
                    break;
                }
            case "Apagar":
                {
                    label11.Text = ObjTV.Apagar();
                    break;
                }
            case "CambiarCanal":
                {
                    label11.Text = ObjTV.CambiarCanal();
                    break;
                }
            case "SubirVolumen":
                {
                    label11.Text = ObjTV.SubirVolumen();
                    break;
                }
```

```
                case "BajarVolumen":
                    {
                        label11.Text = ObjTV.BajarVolumen();
                        break;
                    }
                default:
                    {
                        label11.Text = "Seleccione Una Funcionalidad";
                        break;
                    }
            }
        }
    }
```

Nota: Recuerde que .Net presenta la estructura principal del código. Tenga en cuenta solamente escribir el código que corresponde.

EJERCICIO P.O.O N° 2
(HERENCIA Y POLIMORFISMO)

1. Agregue en la carpeta llamada "Clases" (Desde el explorador de soluciones) una nueva Clase y llámela "Operaciones". Ubique cada bloque de código para escribir las instrucciones según se muestra a continuación:

```
namespace Clases
{
  class Operaciones
    {
      #region Atributos
        private double numero1;
        private double numero2;
        protected double resultado;
      #endregion

     #region Metodos
        public void SetNumero1(double _numero1)
        { numero1 = _numero1; }

        public double GetNumero1()
        { return numero1; }

        public void SetNumero2(double _numero2)
        {  numero2 = _numero2; }

        public double GetNumero2()
        { return numero2; }

        public void SetResultado(double _resultado)
        { resultado = _resultado; }

        public double GetResultado()
        { return resultado; }
```

```
        public void Sumar()
        { resultado = numero1 + numero2; }
        public double Sumar(double num1, double num2)
        { return (num1 + num2); }

        public double Restar()
        {  return (numero1 - numero2);}

        public void Restar(double num1, double num2)
        {resultado = numero1 - num2;}
        public double Multiplicar(double num1, double num2)
        { return (num1 * num2); }

        public double Multiplicar(double num1, double num2, double num3)
        { return (num1 * num2 * num3); }

        public double Dividir(double num1, double num2)
        { return num1/num2; }

        public double CalcularPotencia(double numero, double potencia)
        { return (Math.Pow(numero, potencia)); }
    #endregion
    }
}
```

2. Agregue en la carpeta llamada “Clases” (Desde el explorador de soluciones) una nueva Clase y llámela “Poligono”. Ubique cada bloque de código para escribir las instrucciones según se muestra a continuación:

```
namespace Clases
{
    class Poligono:Operaciones
    // Herencia: La clase Polígono Hereda atributos y métodos Protegidos y Públicos de la clase operaciones
    {
     #region Atributos
        protected double radio;
        protected double volumen;
     #endregion

    #region Metodos
        public void SetRadio(double r)
          {radio = r;}
        public double GetRadio()
          {return radio;}
        public double GetVolumen()
          {return  volumen;}

        public void CalcularVolumen(double h)//volumenCono= "Pi*r^2*h/3"
          {
            double numero1 = Dividir(Math.PI, 3); ;
            double numero2 = CalcularPotencia(radio,2);
            volumen = Multiplicar(numero1, numero2, h);
          }
```

```
        public double CalcularVolumen() // calculoEsfera = "4/3*Pi*r^3"
          {
            double numero1 = Multiplicar(4,Dividir(Math.PI, 3));
            double numero2 = CalcularPotencia(radio, 3);
            return Multiplicar(numero1,numero2);
          }
    #endregion
   }
}
```

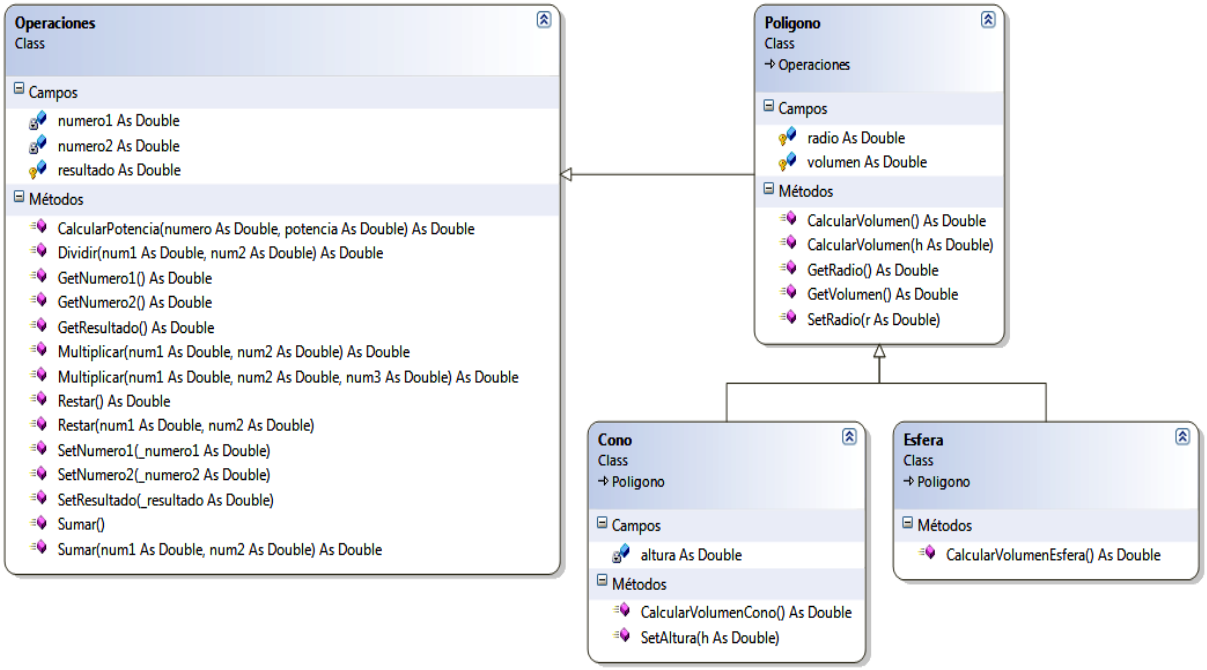
3. Agregue en la carpeta llamada "Clases" una nueva Clase y llámela "Cono". Ubique cada bloque de código para escribir las instrucciones según se muestra a continuación:

```
namespace Clases
{
    class Cono : Poligono
    // Herencia: La clase Cono Hereda atributos y métodos Protegidos y Públicos de la clase Poligono
    {
     private double altura;
     public void SetAltura(double h)
        {altura = h;}

     public double CalcularVolumenCono()
     {
       CalcularVolumen(altura);
       return GetVolumen();
     }
   }
 }
```

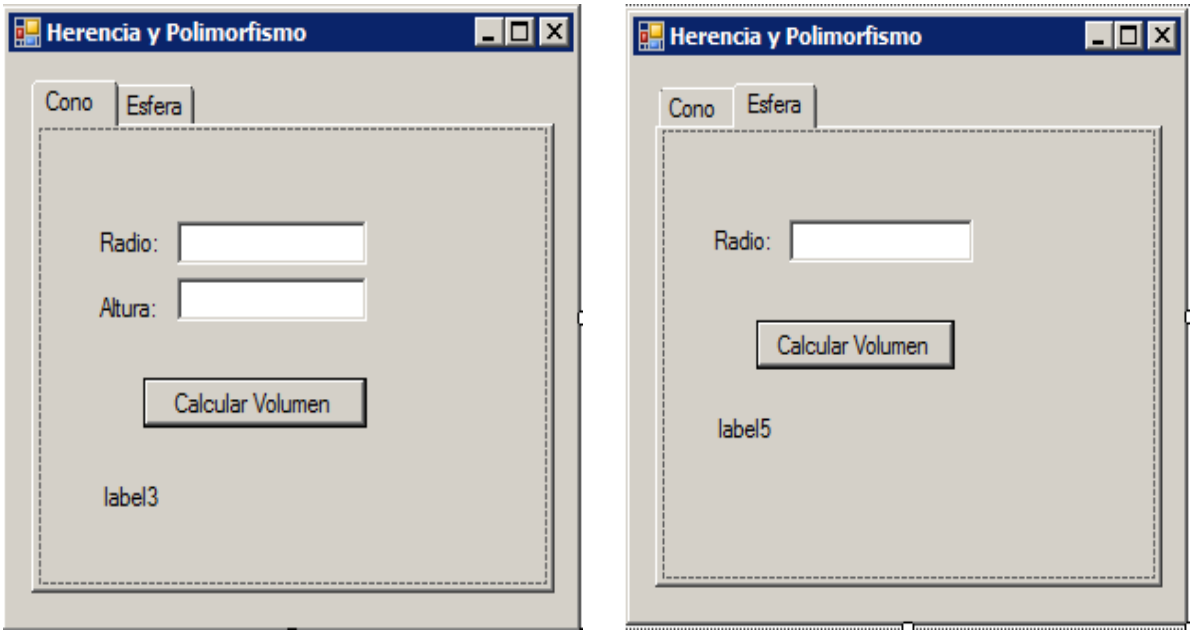
4. Agregue en la carpeta llamada "Clases" una nueva Clase y llámela "Esfera". Ubique cada bloque de código para escribir las instrucciones según se muestra a continuación:

```
namespace Clases
{

    class Esfera:Poligono
    // Herencia: La clase Esfera Hereda los atributos y métodos Protegidos y Públicos de la clase Polígono
       {
        public double CalcularVolumenEsfera()
        {
            return CalcularVolumen();
        }

       }
}
```

Finalmente se debe obtener un diagrama de clases como el que se muestra a continuación:



5. En tiempo de diseño desarrolle el formulario (Herencia y Polimorfismo) tal y como se muestra a continuación:



Nota: Las imágenes corresponden al mismo formulario. El control que permite el uso de pestañas se llama *"tabControl"*

6. En Formulario (Herencia y Polimorfismo) Ubique cada bloque de código (Controles y Eventos) para escribir las instrucciones según se muestra a continuación:

```
using Clases; //Agregue el espacio de nombres correspondiente

public partial class Form3 : Form
    {
        public Form3()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Cono ObjCono = new Cono();
            ObjCono.SetRadio(double.Parse(textBox1.Text));
            ObjCono.SetAltura(double.Parse(textBox2.Text));
            label3.Text = "El area del cono es: " +
            ObjCono.CalcularVolumenCono();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Esfera ObjEsfera = new Esfera();
            ObjEsfera.SetRadio(double.Parse(textBox3.Text));
            label5.Text = "El Area de la esfera es: " +
            ObjEsfera.CalcularVolumenEsfera();
        }
```

TALLER:

I. Aplicar a través de un ejercicio práctico los conceptos fundamentales de la programación orientada a objetos tales como: Clase, Objeto, Constructor y Encapsulamiento. En dicho ejercicio deben diseñar e implementar una clase que represente a una "Persona"

- ✓ Se debe hacer una abstracción que le permita obtener el conjunto de atributos y métodos (mínimo tres) que permitan representar computacionalmente por medio de una clase a los elementos característicos y funcionales más representativos de una persona.

- ✓ En todas las clases que se proponga se debe aplicar el concepto de encapsulamiento, es decir, debe definir los atributos como privados y crear los métodos públicos necesarios para acceder a ellos (Métodos Set y Métodos Get).

- ✓ Se debe implementar un formulario donde se cree objetos, se maneje el estado (Asignar y Leer atributos) y se verifique el comportamiento (llamar o invocar métodos).

II. Desarrolle una aplicación donde aplique la Programación Orientada a Objetos (POO) para satisfacer los siguientes requerimientos:

a) Presentar un "ComboBox" con las siguientes opciones:

- ✓ Suma
- ✓ Resta
- ✓ Multiplicación
- ✓ División
- ✓ Potencia
- ✓ Raíz n-sima
- ✓ Seno
- ✓ Coseno
- ✓ Factorial
- ✓ Serie de Fibonacci.

b) Permitir ingresar uno o dos números por medio de cuadros de textos (TextBox) según la operación seleccionada

c) Cuando el usuario seleccione una de las opciones y presione click sobre algún botón, se debe mostrar en alguna etiqueta (Label) el resultado correcto de acuerdo a la operación seleccionada.

d) Debe aplicar herencia y/o polimorfismo (Una clase padre llamada "OperacionesBasicas" para suma, resta, multiplicación, división y una clase hija llamada "Matematica" para el resto de las operaciones.

(IMPORTANTE: SE DEBE APLICAR P.O.O, ES DECIR, CLASES Y OBJETOS)