Programación Orientada a Objetos 2
Herencia

Contenido

Introducción

- Herencia:
  - Las clases son creadas absorbiendo los métodos y variables de una clase existente
  - Luego esta clase agrega sus propios métodos para ampliar sus capacidades
  - Esta clase es llamada una clase derivada por que esta hereda los métodos y variables de la clase base
  - Los objetos de la clase derivada son objetos de la clase base, pero no viceversa
  - Los objetos de la clase derivada pueden ser tratados como objetos de la clase base
  - Una clase derivada solo puede acceder a los miembros no privados de la clase base a menos que esta herede las funciones de acceso

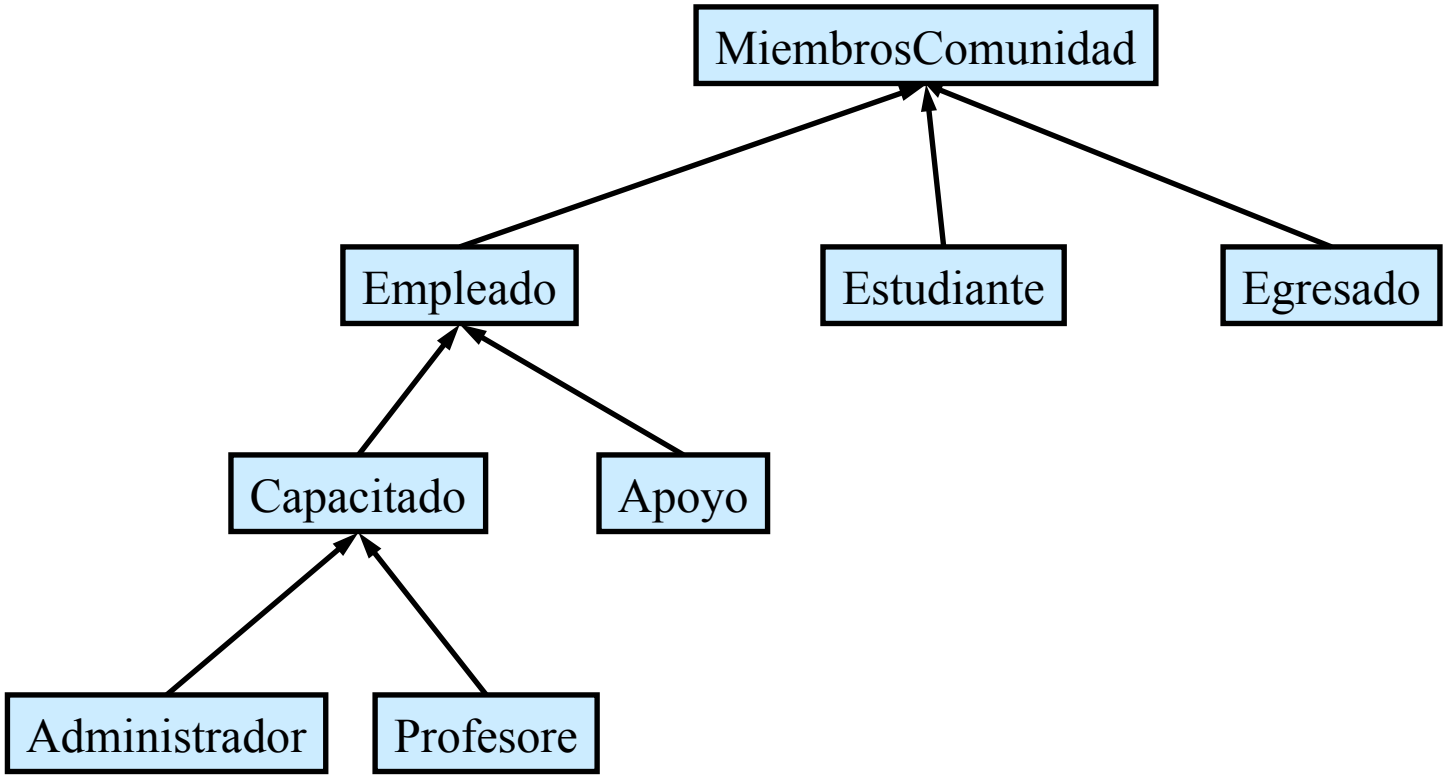
Clase Base y Clase Derivada 1

- Un objeto con frecuencia es un objeto de otra clase
- Cada clase derivada es un objeto de su clase base
- La herencia forma una jerarquía similar a la de un árbol
- Para especificar que una la clase uno es derivada de la clase dos
  - class uno : dos
- Los constructores no son heredados

Clase Base y Clase Derivada 2

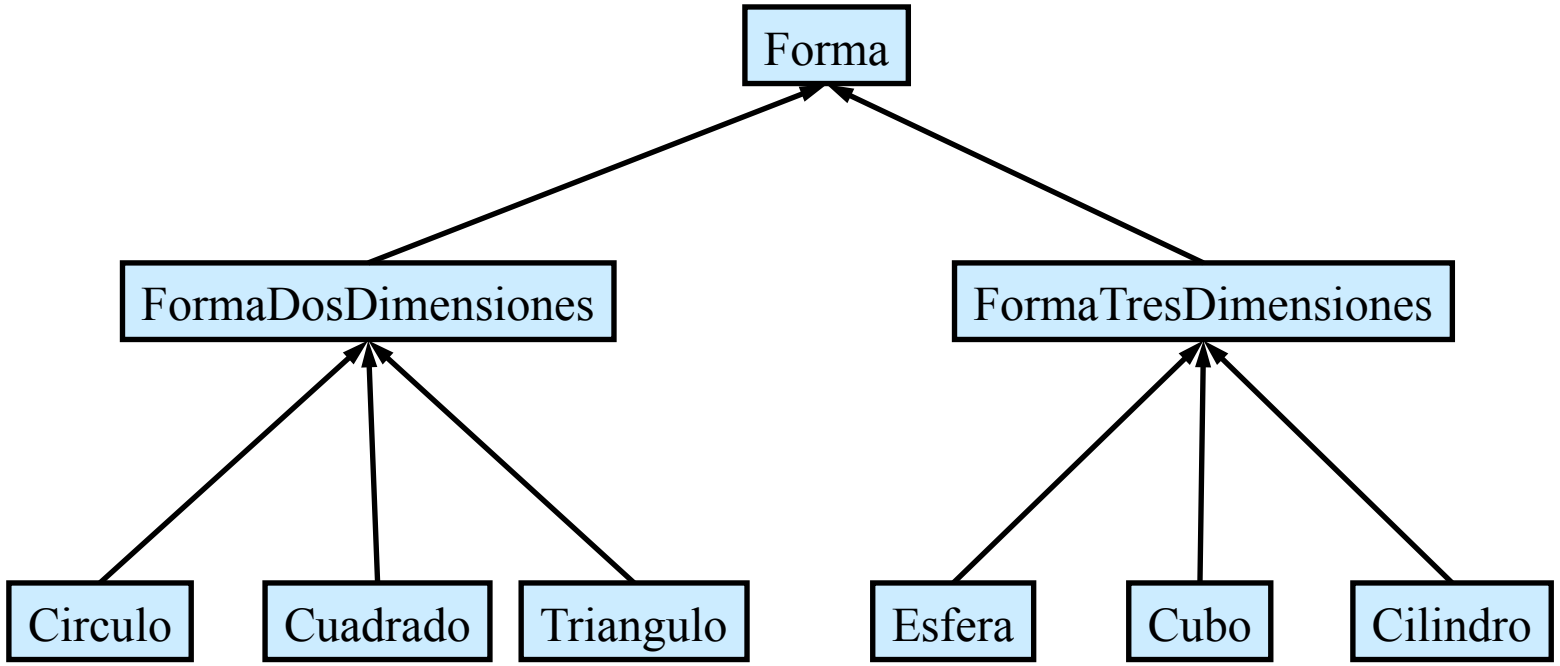
Clase Base	Clase Derivada
Estudiante	**EstudianteGraduado** **EstudianteSinGraduar**
Forma	**Circulo** **Triangulo** **Rectangulo**
Prestamo	**PrestamoCarro** **PrestamoMejoramientoCasa** **PrestamoHipotecario**
Empleado	**EmpleadoCapacitado** **EmpleadoApoto**
Cuenta	**CuentaBancaria** **CuentaAhorros**
Ejemplos de Herencia	

Clase Base y Clase Derivada 3



Herencia jerárquica para **`MiembrosComunidad`**. universitaria

Clase Base y Clase Derivada 4



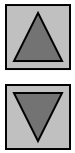
Parte de una clase jerárquica **`Forma`**

Miembros `protected` e `internal`

- miembros `protected`
  - Pueden ser accedidos por la clase base o cualquier clase derivada de esta clase base
- miembros `internal`
  - Pueden solo ser accedidos por la clase declarada en el mismo assembly
- Los miembros sobre escritos de la clase base puede ser accedidos mediante
  - base.miembro

Relación entre Clase Base y Clase Derivada

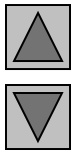
- La primer cosa que hace una clase derivada es llamar al constructor de su clase base, ya sea de forma explicita o implícita
- Se requiere usar la palabra clave `override` si una clase derivada anula o sobreescribe un método de la clase base
- Si un método de la clase base está siendo anulado deberá ser declarado como `virtual`



Outline

Clase: Punto

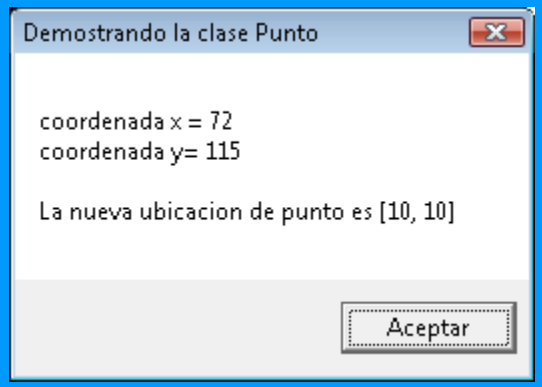
```
// Clase Punto representa un par de coordenadas x-y
using System;

namespace Herencia
{
   // la clase punto herda implicitamente de la clase Object
   public class Punto
   {
       // coordenadas del punto, no pueden ser accedidas
       // directamente por otras clases que hereden
       private int x, y;

      // constructor por defecto (sin parametros)
      public Punto()
      {
          // aqui ocurre una llamada implicita al constructor del objeto
      }

      // constructor
      public Punto( int xValue, int yValue )
      {
         // aqui ocurre una llamada implicita al constructor del objeto
         X = xValue;
         Y = yValue;
      }

      // propiedad X
      public int X
      {
         get { return x; }
         set { x = value; }
      }

      // propiedad Y
      public int Y
      {
         get { return y;}
         set { y = value; }
      }

      // regresa la representacion de cadena de Punto
      // sobre escribe el método ToString de la clase base
      public override string ToString()
      {
         return "[" + x + ", " + y + "]";
      }
   }
}
```

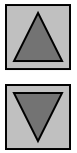


Outline

Clase: PruebaPunto

```
// Clase PruebaPunto
using System;
using System.Windows.Forms;

namespace Herencia
{
   class PruebaPunto
   {
      static void Main( string[] args )
      {
         // instgancia el objeto Punto
         Punto punto = new Punto( 72, 115 );

         // Despliega coordenadas de punto usando props. X y Y
         string salida = "coordenada x = " + punto.X +
            "\n" + "coordenada y= " + punto.Y;

         punto.X = 10; // establece la coordenada x via propiedad X
         punto.Y = 10; // establece la coordenada y via propiedad Y

         // despliega el nuevo valor de punto
         salida  += "\n\nLa nueva ubicacion de punto es " + punto;
         MessageBox.Show( salida , "Demostrando la clase Punto" );
      }
   }
}
```





Outline

Clase: Circulo que hereda a la clase Punto

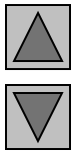
```
// Clase Circulo que herda la clase Punto
using System;
namespace Herencia
{
    class Circulo : Punto
    {
        private double radio;  // radio del Circulo
        // constructor por defecto
        public Circulo()
        {
            // Aqui ocurre la llamada implicita al consutrctor de la clase Punto
        }
        // constructor
        public Circulo(int xValue, int yValue, double radiusValue)
        {
            // Aqui ocurre la llamada implicita al con
            x = xValue;
            y = yValue;
            Radio = radiusValue;
        }
        // propiedad Radio
        public double Radio
        {
            get { return radio; }
            set { if (value >= 0) radio = value; }
        }
        // calcula el diamentro del Circulo
        public double Diametro()
        {
            return radio * 2;
        }
        // calcula la Circunferencia del Circulo
        public double Circunferencia()
        {
            return Math.PI * Diametro();
        }
        // calcula el area del Circulo
        public virtual double area()
        {
            return Math.PI * Math.Pow(radio, 2);
        }
        // regresa la cadena que representa al Circulo
        public override string ToString()
        {
            return "Centro = [" + x + ", " + y + "]" +
               "; Radio = " + radio;
        }
    }
}
```

El intento de modificar los miembros private de la clase base genera un error

Error List

4 Errors | 0 Warnings | 0 Messages

	Description	File	Line	Column	Project
1	'PruebaPunto.Punto.x' is inaccessible due to its protection level	Circulo.cs	21	13	PruebaPunto
2	'PruebaPunto.Punto.y' is inaccessible due to its protection level	Circulo.cs	22	13	PruebaPunto
3	'PruebaPunto.Punto.x' is inaccessible due to its protection level	Circulo.cs	55	35	PruebaPunto
4	'PruebaPunto.Punto.y' is inaccessible due to its protection level	Circulo.cs	55	46	PruebaPunto

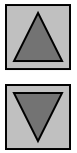


Outline

Clase: Punto2

```
// Clase Punto representa un par de coordenadas x-y
using System;
namespace Herencia
{
   // la clase punto herda implicitamente de la clase Object
   public class Punto2
   {
      // coordenadas del punto protected
      protected  int x, y;

      // constructor por defecto
      public Punto2()
      {

      }

      // constructor
      public Punto2( int xValue, int yValue )
      {
         X = xValue;
         Y = yValue;
      }

      // propiedad X
      public int X
      {
         get { return x; }
         set { x = value; }
      }

     // propiedad Y
      public int Y
      {
         get { return y;}
         set { y = value; }
      }

      // regresa la representacion de cadena de Punto
      public override string ToString()
      {
         return "[" + x + ", " + y + "]";
      }
   }
}
```

Declarar coordenadas como **protected**, las clases que la hereden podrán manipularlas directamente

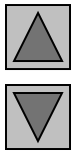


Outline

```
// Clase Circulo que herda la clase Punto
using System;
namespace Herencia
{
    class Circulo2 : Punto2
    {
        private double radio;  // radio del Circulo
        // constructor por defecto
        public Circulo2()
        {
            // Aquí ocurre la llamada implícita al constructor de la clase Punto
        }
        // constructor
        public Circulo2(int xValue, int yValue, double radiusValue)
        {
            // Aqui ocurre la llamada implícita al constructor de la clase Punto
            x = xValue;
            y = yValue;
            Radio = radiusValue;
        }
        // propiedad Radio
        public double Radio
        {
            get { return radio; }
            set { if (value >= 0) radio = value; }
        }
        // calcula el diamentro del Circulo
        public double Diametro()
        {
            return radio * 2;
        }
        // calcula la Circunferencia del Circulo
        public double Circunferencia()
        {
            return Math.PI * Diametro();
        }
        // calcula el area del Circulo
        public virtual double Area()
        {
            return Math.PI * Math.Pow(radio, 2);
        }
        // regresa la cadena que representa al Circulo
        public override string ToString()
        {
            return "Centro = [" + x + ", " + y + "]" +
               "; Radio = " + radio;
        }
    }
}
```

Clase Circulo2 hereda de Punto2

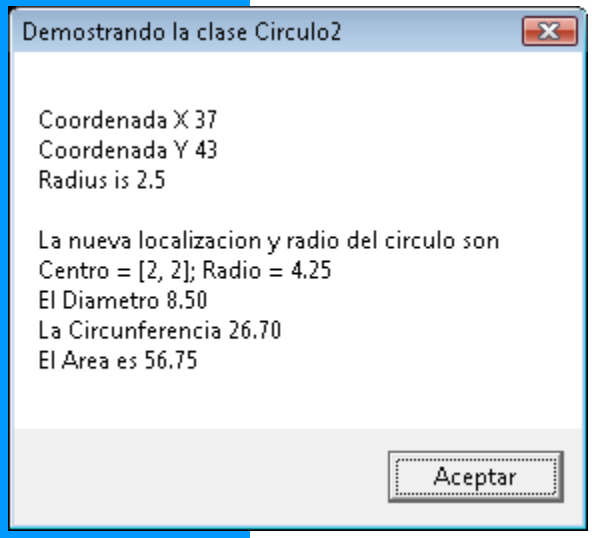
El cambio directo de los miembros protected de la clase base no genera error

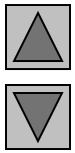


Outline

Clase: PruebaCirculo

```
// Probando clase Circulo2
using System;
using System.Windows.Forms;
namespace Herencia
{
    class PruebaCirculo
    {
        static void Main(string[] args)
        {
            // instania de Circulo2
            Circulo2 circulo = new Circulo2(37, 43, 2.5);
            // obtiene las coordenadas x-y iniciales y radio de Circulo2
            string salida = "Coordenada X " + circulo.X + "\n" +
               "Coordenada Y " + circulo.Y + "\nRadio  " +
               circulo.Radio;
            // establece las coordenadas x'y de Circulo2
            // y el radio a los nuevos valores
            circulo.X = 2;
            circulo.Y = 2;
            circulo.Radio = 4.25;
            // despliega la representacion de cadena de Circulo2
            salida += "\n\n" +
               "La nueva localizacion y radio del circulo son " +
               "\n" + circulo + "\n";
            // despliega el diámetro de Circulo2
            salida  += "El Diametro " +
               String.Format("{0:F}", circulo.Diametro()) + "\n";
            // despliega la Cirncunferencia de Ciculo2
            salida  += "La Circunferencia " +
               String.Format("{0:F}", circulo.Circunferencia()) +
               "\n";
            // despliega el Area del Circulo2
            salida += "El Area es " +
               String.Format("{0:F}", circulo.Area());
            MessageBox.Show(salida, "Demostrando la clase Circulo2");
        }
    }
}
```



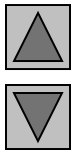


Outline

Clase: Punto3

```
// Clase Punto representa un par de coordenadas x-y
using System;
namespace Herencia
{
   // la clase punto herda implicitamente de la clase Object
   public class Punto3
   {
      // coordenadas del punto
      private int x, y;
      // constructor por defecto
      public Punto3()
      {
         // aqui ocurre una llamada implicita al constructor de Object
      }
      // constructor
      public Punto3( int xValue, int yValue )
      {
         // aqui ocurre una llamada implicita al constructor de Object
         X = xValue; // usa propiedad X
         Y = yValue; // usa propiedad Y
      }
      // propiedad X
      public int X
      {
         get { return x; }
         set { x = value; }
      }
      // propiedad Y
      public int Y
      {
         get { return y;}
         set { y = value; }
      }
      // regresa la representacion de cadena de Punto
      public override string ToString()
      {
         return "[" + x + ", " + y + "]";
      }
   }
}
```

Declarar coordenadas como privadas



Outline

Clase: Circulo3

```
// Clase Circulo que herda la clase Punto
using System;
namespace Herencia
{
    class Circulo3 : Punto3
    {
        private double radio;  // radio del Circulo
        // constructor por defecto
        public Circulo3()
        {
            // Aquí ocurre la llamada implicita al consutrctor de la clase Punto
        }
        // constructor
        public Circulo3(int xValue, int yValue, double radiusValue)
            : base(xValue, yield)
        {
            Radio = radiusValue;
        }
        // propiedad Radio
        public double Radio
        {
            get { return radio; }
            set { if (value >= 0) radio = value; }
        }
        // calcula el diamentro del Circulo
        public double Diametro()
        {
            return radio * 2;
        }
        // calcula la Circunferencia del Circulo
        public double Circunferencia()
        {
            return Math.PI * Diametro();
        }
        // calcula el area del Circulo
        public virtual double Area()
        {
            return Math.PI * Math.Pow(radio, 2);
        }
        // regresa la cadena que representa al Circulo
        public override string ToString()
        {
            return "Centro = " + base.ToString() +
               "; Radio = " + radio;
        }
    }
}
```

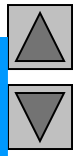
Constructor con llamada implícita al constructor de la clase base

Constructor con llamada implícita al constructor de la clase base

Método área declarado virtual para que pueda ser sobre escrito

Método ToString de Circulo3 sobreescribe método ToString de Punto3

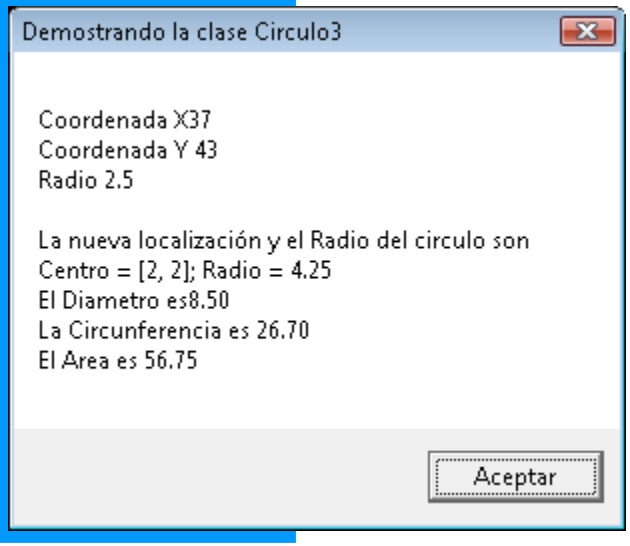
Llama al método ToString de Punto3 para desplegar las coordenadas



Outline

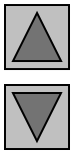
Clase: PruebaCirculo4

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
namespace Herencia
{
    class PruebaCirculo3
    {
        static void Main(string[] args)
        {
            // instancia Circulo3
            Circulo3 circulo = new Circulo3(37, 43, 2.5);
            // obtiene coordneadas x-y iniciales de Circulo3 y el radio
            string salida = "Coordenada X" + circulo.X + "\n" +
               "Coordenada Y " + circulo.Y + "\n" +
               "Radio " + circulo.Radio;

            // establece coordenadas x-y de Circulo3 y Radio a los nuevos valores
            circulo.X = 2;
            circulo.Y = 2;
            circulo.Radio = 4.25;


            // despliega la representación de cadena de Circulo3
            salida += "\n\n" +
               "La nueva localización y el Radio del circulo son  " +
               "\n" + circulo + "\n";

            // despliega el diametro de Circulo3
            salida += "El Diametro es" +
               String.Format("{0:F}", circulo.Diametro()) + "\n";

            // despliega la circunferencia de Circulo3
            salida += "La Circunferencia es " +
               String.Format("{0:F}", circulo.Circunferencia()) + "\n";
            // despliega el área de Circulo3
            salida += "El Area es " +
               String.Format("{0:F}", circulo.Area());
            MessageBox.Show(salida, "Demostrando la clase Circulo3");
        }
    }
}
```



Caso de estudio, tres niveles de Herencia

- Ejemplo de tres niveles de herencia
  - La clase **`Cilindro`** hereda de la clase **`Circulo3`**
  - La clase **`Circulo3`** hereda de la calase **`Punto3`**



Outline

Clase: Cilindro

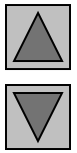
```
using System;
using System.Collections.Generic;
using System.Text;
namespace Herencia
{
    class Cilindro : Circulo3
    {
        private double alto;

      // construcror por defecto
      public Cilindro()
      {
         // aquí ocurre llamada implícita al constructor de Circulo3
      }

      // constructor con 4 parámetros
      public Cilindro( int xValue, int yValue, double radiusValue,
         double altoValue ) : base( xValue, yValue, radiusValue )
      {
         alto = altoValue; // establece el alto del Cilindro
      }

      // propiedad alto
      public double Alto
      {
         get { return alto; }
         set { if ( value >= 0 ) alto = value; }
      }

      // sobre escribe el método Area de Circulo3 para calcular el Area del Cilindro
      public override double Area()
      {
         return 2 * base.Area() + base.Circunferencia() * alto;
      }

      // calcula el volumén del Cilindro
      public double Volumen()
      {
         return base.Area() * alto;
      }

      // convierte cilindro a cadena
      public override string ToString()
      {
         return base.ToString() + "; alto = " + alto;
      }
    }
}
```

Clase Cilindro hereda de clase Circulo3

Declara variable alto como **private**

Constructor que llama implícitamente al constructor de la clase base

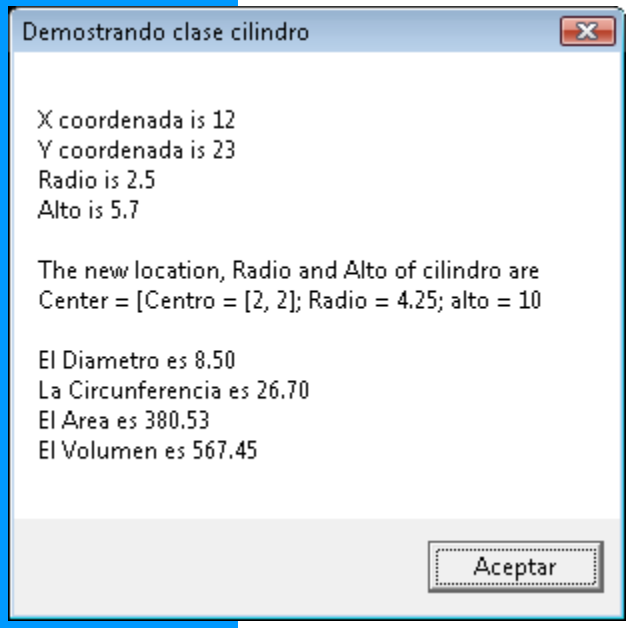
Constructor que llama explícitamente al constructor de la clase base



Outline

Clase: PruebaClilindro

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
namespace Herencia
{
    class PruebaCilindro
    {
        static void Main(string[] args)
        {
            // instancia un objeto de la clase Cilindro
            Cilindro cilindro = new Cilindro(12, 23, 2.5, 5.7);
            // obtiene coordenadas x, y, Radio y Alto de las propiedades
            string salida = "X coordenada is " + cilindro.X + "\n" +
               "Y coordenada is " + cilindro.Y + "\nRadio is " +
               cilindro.Radio + "\n" + "Alto is " + cilindro.Alto;
            // establece coordenadas x, y, Radio y Alto usando las propiedades
            cilindro.X = 2;
            cilindro.Y = 2;
            cilindro.Alto = 10;
            cilindro.Radio = 4.25;
            // obtiene las nuevas coordenadas x-y y el Radio
            salida += "\n\nThe new location, Radio and Alto of " +
               "cilindro are\nCenter = [" + cilindro + "\n\n";
            // Despliega el Diametro del Cilindro
            salida += "El Diametro es " +
               String.Format("{0:F}", cilindro.Diametro()) + "\n";
            // Despliega la Circunferencia del Cilindro
            salida += "La Circunferencia es " +
               String.Format("{0:F}", cilindro.Circunferencia()) + "\n";
            // Despliega el Area del Cilindro
            salida += "El Area es " +
               String.Format("{0:F}", cilindro.Area()) + "\n";
            // Despliega el Volumen del Cilindro
            salida += "El Volumen es " +
               String.Format("{0:F}", cilindro.Volumen());
            MessageBox.Show(salida, "Demostrando clase cilindro");
        }
    }
}
```



Constructores y destructores en clases derivadas

- Al instanciar una clase derivada, causa que el constructor de la clase base sea llamado, implicita o explícitamente
  - Puede causar una reacción en cadena cuando una clase base también contiene una clase derivada
- Cuando se llama a un destructor, este realiza su tarea y luego invoca el constructor de la clase base de la clase derivada

Ingeniería de Software usando Herencia

- Puedes personalizar las clases derivadas para que cumplan con tus necesidades, a través de
  - Crear nuevas variables miembro
  - Crear nuevos métodos
  - Sobre escribir los miembros de la clase base
- .NET Framework Class Library(FCL) permite el rehusó del software a través de la herencia