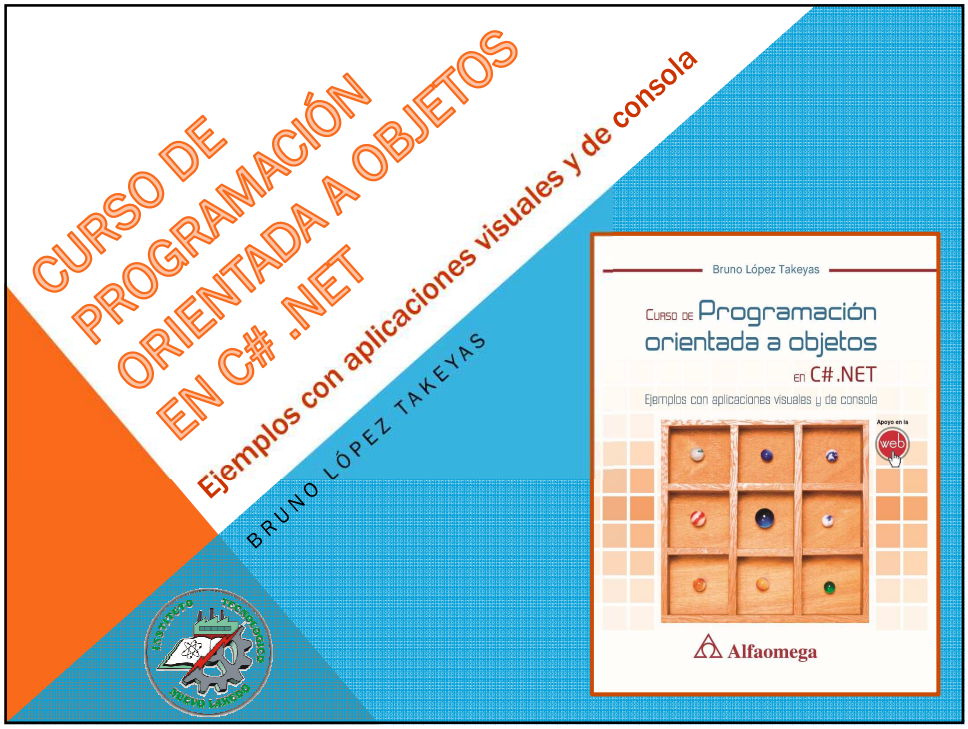

CURSO DE PROGRAMACIÓN ORIENTADA A OBJETOS EN C# .NET
Ejemplos con aplicaciones visuales y de consola
BRUNO LÓPEZ TAKEYAS
Bruno López Takeyas
Curso de Programación orientada a objetos en C#.NET
Ejemplos con aplicaciones visuales y de consola
Apoyo en la web
Alfaomega


CAPÍTULO 8
DELEGADOS
2

Preguntas detonadoras

- ¿Qué es un delegado? ¿Para qué sirve?
- ¿En qué circunstancias se recomienda implementar delegados?
- ¿Qué es un método anónimo? ¿Para qué sirve?
- ¿Qué es una expresión lambda?
- ¿Cuándo se recomienda implementar una expresión lambda?

3

DELEGADOS

- Es un nuevo **tipo** que hace referencia a un método
- Es muy semejante a un apuntador de C++
- Permiten pasar los métodos como parámetros

Sintaxis

```
modificador  delegate tipo nombre(parámetros);
```

Modificadores:

- `private`
- `protected`
- `public`
- `internal`

¿Para qué sirve un delegado?

- Si un delegado es un tipo, entonces se pueden declarar variables de dicho tipo.
- Un delegado es una referencia a un método.
- Una variable creada de un tipo delegado representa a un método determinado.

Los delegados se utilizan para enviar métodos como parámetros a otros métodos.

Ejemplo de delegados

```
public delegate bool TipoOrdenamiento(double x, double y);
```

- Cualquier método puede asignarse a un delegado.
- Deben coincidir...
  - El prototipo del delegado
  - Parámetros
  - Tipo de dato del valor devuelto

Cuando utilizar delegados en lugar de interfaces

- Tanto los delegados como las interfaces permiten a un diseñador de clases separar las declaraciones y la implementación de tipos.
- Cualquier objeto puede utilizar una referencia de interfase o un delegado sin tener conocimiento alguno sobre la clase que implementa el método de interfase o delegado.

Utilice un delegado cuando …

- Se utilice un modelo de diseño de eventos.
- Se prefiere a la hora de encapsular un método estático.
- El autor de las llamadas no tiene ninguna necesidad de obtener acceso a otras propiedades, métodos o interfaces en el objeto que implementa el método.
- Se desea conseguir una composición sencilla.
- Una clase puede necesitar más de una implementación del método.

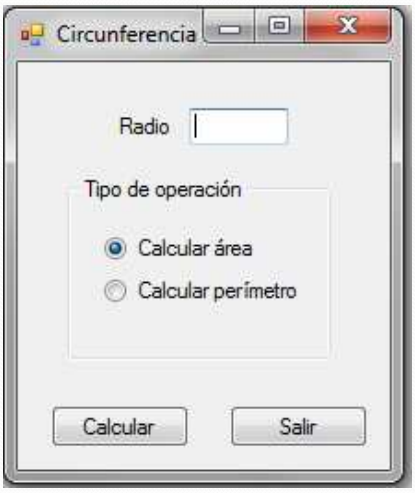
Utilice una interfase cuando …

- Haya un grupo de métodos relacionados a los que se pueda llamar.
- Una clase sólo necesita una implementación del método.
- La clase que utiliza la interfase deseará convertir esa interfase en otra interfase o tipos de clase.
- El método que se va a implementar está vinculado al tipo o identidad de la clase; por ejemplo, métodos de comparación.

¿Cómo se declara una variable de un tipo delegado?

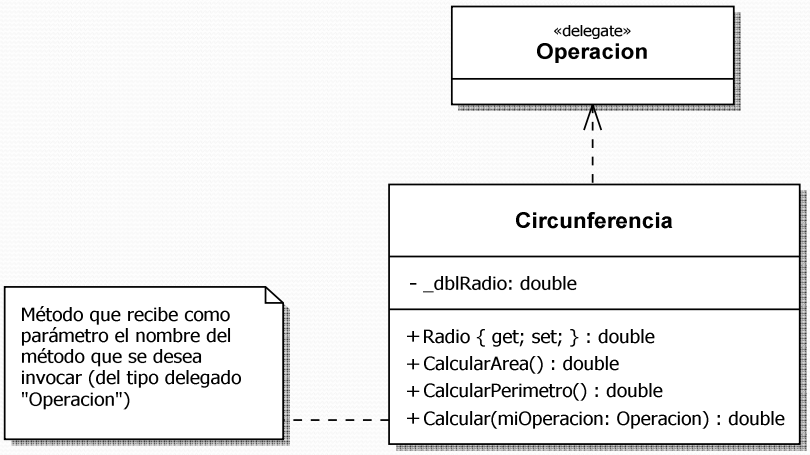
- Los delegados se declaran como cualquier otro objeto en .NET
- Para asignar un delegado, no se asigna un "valor", sino un método.

Ejemplo de aplicación

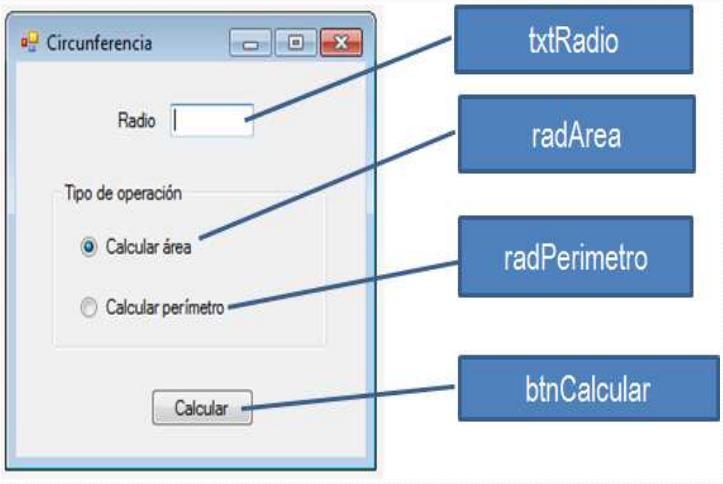


- Utilizar un delegado para seleccionar el tipo de cálculo a realizar

Diagrama de clase en UML

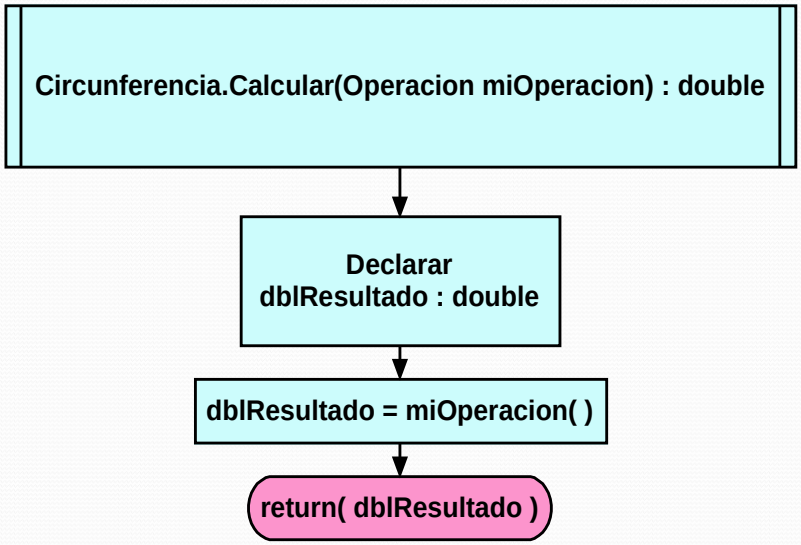


Diseño de la forma



13

Diagrama de flujo



14

Codificación de la clase

```
class Circunferencia
{
    //Atributo privado
    private double _dblRadio;

    // Propiedad pública
    public double Radio
    {
        get
        {
            return _dblRadio;
        }

        set
        {
            _dblRadio = value;
            if (_dblRadio < 0)
            {
                _dblRadio = 0;
                throw new Exception("No se
permite un valor negativo");
            }
        }
    }
```

```
 // Método para calcular el área
    public double CalcularArea()
    {
        return Math.PI * Math.Pow(Radio, 2);
    }

    // Método para calcular el Perímetro
    public double CalcularPerimetro()
    {
        return Math.PI * Radio * 2;
    }

    // Delegado público
    public delegate double Operacion();

    // Método que recibe como parámetro el
método que desea invocar (del tipo del
delegado)
    public double Calcular(Operacion
miOperacion)
    {
        double dblResultado = miOperacion();
        return dblResultado;
    }
}
```

Utilización del delegado para invocar al método

```
private void btnCalcular_Click(object sender, EventArgs e)
{
   Circunferencia miCircunferencia = new Circunferencia();

   try
   {
      miCircunferencia.Radio = double.Parse(txtRadio.Text);
   }
   catch (Exception x)
   {
      MessageBox.Show(x.Message);
      txtRadio.Text = "";
      txtRadio.Focus();
      return;
   }

   if (radCalcularArea.Checked)
    MessageBox.Show("Área = " + miCircunferencia.Calcular(miCircunferencia.CalcularArea));
   if (radCalcularPerimetro.Checked)
    MessageBox.Show("Perímetro = " + miCircunferencia.Calcular(miCircunferencia.CalcularPerimetro));
}
```

Métodos anónimos

- Hasta ahora, se han revisado los delegados generados a partir de métodos con un nombre
- Son estrategias para crear objetos de un delegado e inmediatamente definir el bloque de sentencias de código que ejecutará cuando se invoque

¿Para qué sirve un método anónimo?

- Pasa directamente un bloque de sentencias de código como parámetro a un delegado.
- Reduce con esto la sobrecarga de codificación al momento de crear objetos de delegados
- Evita definir un método independiente.

¿Cómo se invoca un método anónimo?

- El método anónimo no tiene nombre, entonces ¿cómo puede invocarse?
- La respuesta es a través de delegados.

Expresiones lambda

- Es un método anónimo que se utiliza para crear delegados
- Permite implementar funciones que se pueden enviar como parámetro o devolver un valor producto de su llamada.
- Puede ser un procedimiento o una función
- Puede o no tener parámetros

Sintaxis de expresiones lambda

```
(parámetros de entrada) => expresión | bloque de sentencias
```

```
() => Console.Write("Hola")  // Expresión lambda sin parámetros

x => x + 3  // Expresión lambda con un parámetro (no requiere paréntesis)

(Precio, Descuento) => Precio – (Precio * Descuento / 100.0)

(string Nombre, int Longitud) => Nombre.Length > Longitud
```

¿Cómo implementar expresiones lambda?

- *Con delegados*
  - El programador define explícitamente un delegado que coincida con los parámetros y tipo de valor devuelto de la expresión lambda
- *Con delegados genéricos integrados*
  - Se utilizan los delegados genéricos integrados en la librería de clase base (BCL) con la intención de obviar la escritura de código extra para un delegado y un método

¿Cómo implementar expresiones lambda?

```
// Delegado definido por el programador
delegate double Delegado(int x, double y);

// Delegado definido por el programador con expresión lambda
Delegado miDelegado = (x, y) => x * y;


// Delegado genérico con expresión lambda
System.Func<int, double, double> miDelegadoGenerico = (x, y) => x * y;
```

Ejemplo de aplicación con una expresión lambda

- *Calcular el descuento de un artículo*
- Variables
  - `dblPrecio`
  - `dblPorcentajeDescuento`

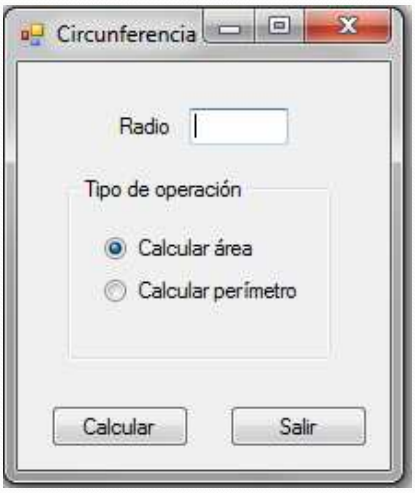
Expresión lambda que calcula el descuento con delegado creado por el programador

```
// Definición de un delegado
delegate double Calcular(double _dblPrecio, double
_dblPorcentajeDescuento);
 // Delegado con expresión lambda
Calcular CalcularDescuento = (Precio, Descuento) => Precio - (Precio *
Descuento / 100.0);
 // Ejecución de la expresión lambda
Console.WriteLine("El precio del artículo (con el descuento aplicado)
es "+CalcularDescuento(dblPrecio,
dblPorcentajeDescuento).ToString("C"));
```

Expresión lambda que calcula el descuento con delegado genérico integrado

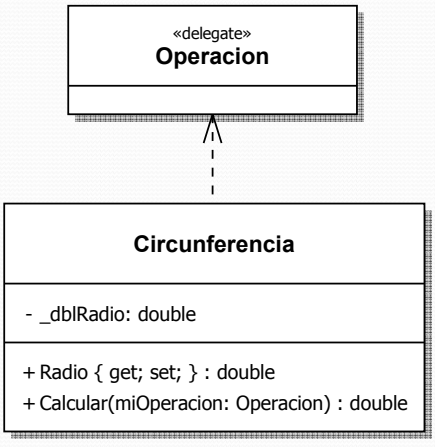
```
// Delegado genérico con expresión lambda
System.Func<double, double, double>
CalcularDescuentoGenerico = (Precio, Descuento) =>
Precio - (Precio * Descuento / 100.0);


Console.WriteLine("El precio del artículo (con el
descuento aplicado) es " +
CalcularDescuentoGenerico(dblPrecio,
dblDescuento).ToString("C"));
```

Otro ejemplo de aplicación de una expresión lambda



- Utilizar una expresión lambda para determinar el tipo de cálculo a realizar

Diagrama de clase en UML



- Nótese que no se implementan los métodos para calcular el área ni el perímetro
- El método `Calcular()` utiliza un delegado para realizar ambos cálculos

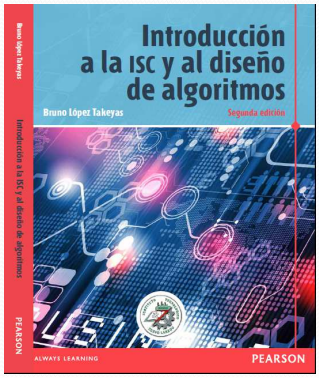
Codificación de la clase

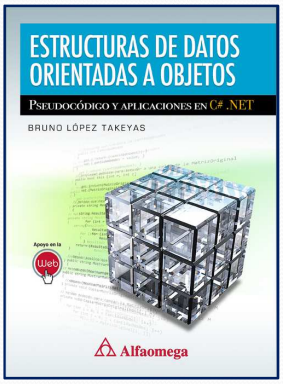
```
class Circunferencia
    {
        private double _dblRadio;

        public double Radio
        {
            get { return _dblRadio; }
            set { _dblRadio = value; }
        }

        // Delegado
        public delegate double Operacion();

        // Método que recibe como parámetro el código
        // a ejecutar enviado a través de una
        // expresión lambda
        public double Calcular(Operacion miOperacion)
        {
            return miOperacion();
        }
    }
```

```
private void btnCalcular_Click(object sender, EventArgs e)
{
   Circunferencia miCircunferencia = new Circunferencia();

   try {
          miCircunferencia.Radio = double.Parse(txtRadio.Text);
   }
   catch(Exception ex)  {
          MessageBox.Show(ex.Message);
          return;
   }
   finally {
          txtRadio.Clear();
          txtRadio.Focus();
   }

   if (radCalcularArea.Checked)
   {
      double dblArea = miCircunferencia.Calcular(() => Math.PI * Math.Pow(miCircunferencia.Radio, 2));
      MessageBox.Show("Área = " + dblArea);
   }

   if (radCalcularPerimetro.Checked)
   {
      double dblPerimetro = miCircunferencia.Calcular(() => Math.PI * miCircunferencia.Radio * 2);
      MessageBox.Show("Perímetro = " + dblPerimetro);
   }
}
```

Otros títulos del autor

http://www.itnuevolaredo.edu.mx/Takeyas/Libro





takeyas@itnuevolaredo.edu.mx

Bruno López Takeyas