

# C#.Net

## Eventos

# Repaso Delegados - Ejercicio

- Vamos a codificar una clase **Cuenta5** que utilizaremos para contar en forma regresiva 5 segundos.
- Vamos a escribir los segundos que faltan en la consola cada vez que transcurra un segundo.
- En principio todo el trabajo lo va a realizar la clase **Cuenta5**
- Luego iremos transformando el código hasta implementar un mecanismo de eventos. La clase **Cuenta5** lanzará el evento **SegCumplido** al que podrán suscribirse otras clases para manejarlo en forma conveniente, en este caso simplemente imprimiendo en la consola los segundos que faltan.

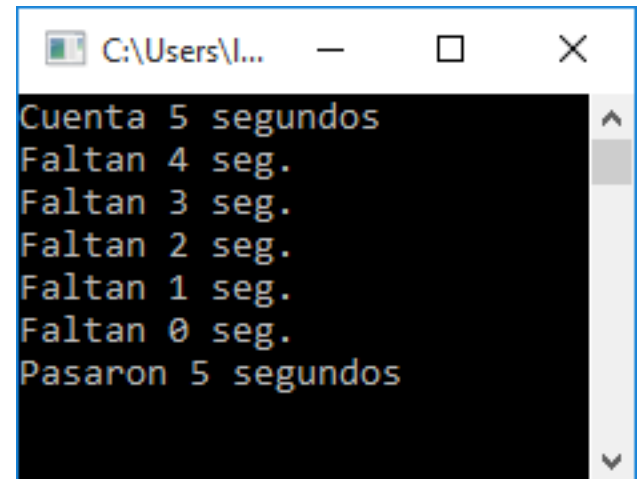
# Repaso Delegados - Ejercicio

```
using System;
class Cuenta5
{
    public void Run()
    {
        int contador=5;
        while (contador-- > 0){
            System.Threading.Thread.Sleep(1000);
            Console.WriteLine("Faltan {0} seg.",contador);
        }
    }
}
```

# Repaso Delegados - Ejercicio

```
class Program
{
    static void Main()
    {
        Cuenta5 cont=new Cuenta5();
        Console.WriteLine("Cuenta 5 segundos");
        cont.Run();
        Console.WriteLine("Pasaron 5 segundos");
        Console.ReadKey();
    }
}
```

Ejecute y compruebe  
su funcionamiento



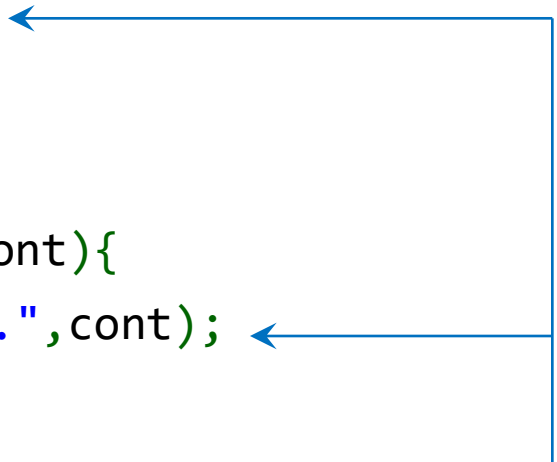
```
C:\Users\I...
Cuenta 5 segundos
Faltan 4 seg.
Faltan 3 seg.
Faltan 2 seg.
Faltan 1 seg.
Faltan 0 seg.
Pasaron 5 segundos
```

# Repaso Delegados - Ejercicio

- Vamos a modificar la clase **Cuenta5** para separar del método **Run** el código que se refiere al manejador del evento, es decir aquello que se debe realizar cada vez que se cumple un segundo

# Repaso Delegados - Ejercicio

```
class Cuenta5 {  
    public void Run(){  
        int contador=5;  
        while (contador-- > 0){  
            System.Threading.Thread.Sleep(1000);  
            manejadorDelEvento(contador);  
        }  
    }  
    private void manejadorDelEvento(int cont){  
        Console.WriteLine("Faltan {0} seg.",cont);  
    }  
}
```



The diagram consists of two blue arrows. The first arrow originates from the `manejadorDelEvento(contador);` line within the `Run()` method and points to the `manejadorDelEvento(int cont)` method signature. The second arrow originates from the `Console.WriteLine("Faltan {0} seg.",cont);` line within the `manejadorDelEvento` method and points to the same `manejadorDelEvento(int cont)` signature, illustrating the encapsulation of event handling logic.

Separamos del método **Run** el código que se refiere al manejo del evento

# Repaso Delegados - Ejercicio

- Ahora modificamos la clase **Cuenta5** para llamar al manejador del evento por medio de una variable de tipo delegado. En esta variable se encola el método que usamos de manejador.
- Llamaremos a la variable con un nombre representativo del evento que representa:  
`SegCumplido`
- Llamaremos al delegado con un nombre representativo de los métodos que identifica:  
`SegCumplidoEventHandler`

# Repaso Delegados - Ejercicio

```
delegate void SegCumplidoEventHandler(int cont);
```

```
class Cuenta5 {  
    public SegCumplidoEventHandler SegCumplido;  
    public void Run(){  
        int contador=5;  
        SegCumplido = manejadorDelEvento;  
        while (contador-- > 0){  
            System.Threading.Thread.Sleep(1000);  
            SegCumplido(contador);  
        }  
    }  
    private void manejadorDelEvento(int cont){  
        Console.WriteLine("Faltan {0} seg.",cont);  
    }  
}
```



# Repaso Delegados - Ejercicio

- Ya estamos en condiciones de sacar el manejador de la clase **Cuenta5** para que sea otro objeto el que se suscriba al evento con su propio manejador.

```
class Cuenta5 {  
    public SegCumplidoEventHandler SegCumplido;  
    public void Run(){  
        int contador=5;  
        while (contador-- > 0){  
            System.Threading.Thread.Sleep(1000);  
            if (SegCumplido != null) ←  
                SegCumplido(contador);  
        }  
    }  
}
```

Ahora es necesario asegurarse que SegCumplido posee algún método antes de invocarlo

# Repaso Delegados - Ejercicio

- Pasamos el manejador a la clase **Program**

```
class Program{  
    static void Main(){  
        Cuenta5 cont=new Cuenta5();  
        cont.SegCumplido=contSegCumplido;  
        Console.WriteLine("Cuenta 5 segundos");  
        cont.Run();  
        Console.WriteLine("Pasaron 5 segundos");  
        Console.ReadKey();  
    }  
    static void contSegCumplido(int cont){  
        Console.WriteLine("Faltan {0} seg.",cont);  
    }  
}
```

**Program** se suscribe con el manejador **contSegCumplido** al evento que lanza el objeto **cont**

# Repaso Delegados - Ejercicio

- Finalmente para cumplir con la convención sobre la nomenclatura y los parámetros involucrados en el mecanismo de eventos se agrega la clase **SegCumplidoEventArgs** y se modifica el programa de la siguiente forma:

# Repaso Delegados - Ejercicio

```
delegate void SegCumplidoEventHandler(object sender,  
                                     SegCumplidoEventArgs e);
```

```
class SegCumplidoEventArgs:EventArgs{  
    public int Valor{get;set;}  
}
```

```
class Cuenta5 {  
    public SegCumplidoEventHandler SegCumplido;  
    public void Run(){  
        int contador=5;  
        while (contador-- > 0){  
            System.Threading.Thread.Sleep(1000);  
            if (SegCumplido != null)  
                SegCumplido(this,new SegCumplidoEventArgs(){Valor=contador});  
        }  
    }  
}
```

# Repaso Delegados - Ejercicio

```
using System;
```

```
class Program{
```

```
    static void Main(){
```

```
        Cuenta5 cont=new Cuenta5();
```

```
        cont.SegCumplido=contSegCumplido;
```

```
        Console.WriteLine("Cuenta 5 segundos");
```

```
        cont.Run();
```

```
        Console.WriteLine("Pasaron 5 segundos");
```

```
        Console.ReadKey();
```

```
    }
```

```
    static void contSegCumplido(object sender, SegCumplidoEventArgs e)
```

```
    {
```

```
        Console.WriteLine("Faltan {0} seg.", e.Valor);
```

```
    }
```

```
}
```

# Eventos

- Un evento es similar a una propiedad donde el campo asociado es un delegado.
- Permiten controlar la forma en que se accede a los campos delegados y dan la posibilidad de asociar código a ejecutar cada vez que se añada o elimine un método de un campo delegado.
- A diferencia de los delegados, a los eventos sólo se le pueden aplicar dos operaciones: **+=** y **-=**.

# Eventos

- **Sintaxis:**

```
<modificadores> event <tipoDelegado> <nombreEvento>
{
    add {
        <códigoAdd>
    }
    remove {
        <códigoRevome>
    }
}
```

# Eventos - Ejercicio

- Vamos a modificar la clase **Cuenta5** para que en lugar de publicar una variable de tipo Delegado publique un evento. Luego vamos a conseguir que en el momento de suscribirse al evento **SegCumplido** en un objeto **Cuenta5** se dispare automáticamente el método **Run**, que además dejará de ser público.
- Comenzamos definiendo un evento en la clase **Cuenta5** asociado a la variable **SegCumplido**



# Eventos - Ejercicio

```
class Cuenta5 {  
    private SegCumplidoEventHandler segCumplido;  
    public event SegCumplidoEventHandler SegCumplido{  
        add{segCumplido += value;}  
        remove{segCumplido -= value; }  
    }  
    public void Run(){  
        int contador=5;  
        while (contador-- > 0){  
            System.Threading.Thread.Sleep(1000);  
            if (segCumplido != null)  
                segCumplido(this,new SegCumplidoEventArgs(){Valor=contador});  
        }  
    }  
}
```

La variable de tipo delegado se hace privada (utilizamos **segCumplido** en minúscula)

# Eventos - Ejercicio

```
class Cuenta5 {  
    private SegCumplidoEventHandler segCumplido;  
    public event SegCumplidoEventHandler SegCumplido{  
        add{segCumplido += value;}  
        remove{segCumplido -= value; }  
    }  
}
```

```
public void Run(){  
    int contador=5;  
    while (contador-- > 0){  
        System.Threading.Thread.Sleep(1000);  
        if (segCumplido != null)  
            segCumplido(this,new SegCumplidoEventArgs(){Valor=contador});  
    }  
}  
}
```

Se agrega el evento **SegCumplido** que controlará el acceso al delegado **segCumplido**.

# Eventos - Ejercicio

```
class Cuenta5 {  
    private SegCumplidoEventHandler segCumplido;  
    public event SegCumplidoEventHandler SegCumplido{  
        add{segCumplido += value;}  
        remove{segCumplido -= value; }  
    }  
    public void Run(){  
        int contador=5;  
        while (contador-- > 0){  
            System.Threading.Thread.Sleep(1000);  
            if (segCumplido != null)  
                segCumplido(this,new SegCumplidoEventArgs(){Valor=contador});  
        }  
    }  
}
```

A diferencia de las propiedades, donde se puede omitir alguno de los descriptores (**get** o **set**) con los eventos es obligatorio codificar los dos descriptores de acceso (**add** y **remove**)

# Eventos - Ejercicio

```
class Program{
    static void Main(){
        Cuenta5 cont=new Cuenta5();
        cont.SegCumplido += contSegCumplido;
        Console.WriteLine("Cuenta 5 segundos");
        cont.Run();
        Console.WriteLine("Pasaron 5 segundos");
        Console.ReadKey();
    }
    static void contSegCumplido(object sender, SegCumplidoEventArgs e)
    {
        Console.WriteLine("Faltan {0} seg.",e.Valor);
    }
}
```

En el método **Main()** cambiar el operador = por +=

# Eventos - Ejercicio

- Ahora se pretende que en el momento de suscribirse al evento **SegCumplido** de un objeto **Cuenta5** se dispare automáticamente el método **Run**. Además **Run** debe hacerse privado.
- Para resolverlo, simplemente se realiza una invocación al método **Run** desde la sección **add** en la definición del evento **SegCumplido**. **Run** se hace privado y se modifica **Program** que ya no debe invocar a dicho método.

# Eventos - Ejercicio

```
class Cuenta5 {  
    private SegCumplidoEventHandler segCumplido;  
    public event SegCumplidoEventHandler SegCumplido{  
        add{segCumplido += value; this.run();}  
        remove{segCumplido -= value; }  
    }  
    private void run(){  
        int contador=5;  
        while (contador-- > 0){  
            System.Threading.Thread.Sleep(1000);  
            if (segCumplido != null)  
                segCumplido(this,new SegCumplidoEventArgs(){Valor=contador});  
        }  
    }  
}
```

Debido a que ahora el método **Run** es privado, por convención lo renombramos con minúscula

# Eventos - Ejercicio

La suscripción al evento  
provocará el inicio de la cuenta  
regresiva

```
class Program{
    static void Main(){
        Cuenta5 cont=new Cuenta5();
        Console.WriteLine("Cuenta 5 segundos");
        cont.SegCumplido+=contSegCumplido;
        Console.WriteLine("Pasaron 5 segundos");
        Console.ReadKey();
    }
    static void contSegCumplido(object sender,
                                SegCumplidoEventArgs e){
        Console.WriteLine("Faltan {0} seg.",e.Valor);
    }
}
```

# Métodos anónimos

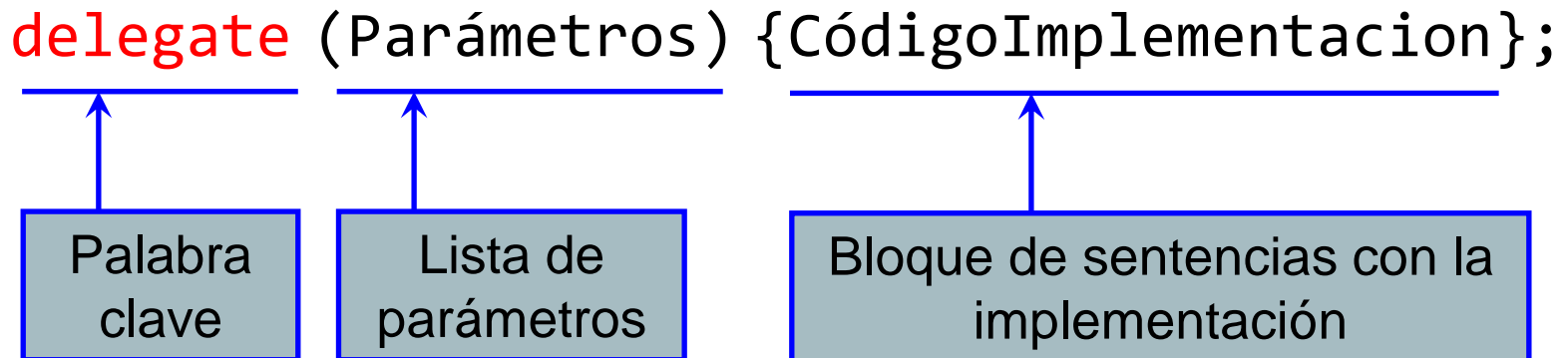
- **¿Qué sucede si un método sólo se utiliza para crear una instancia del delegado?** En ese caso, no existe una necesidad real de usar un **método con nombre** definido separadamente.
- Los **métodos anónimos** permiten prescindir del método con nombre definido por separado.
- Un **método anónimo** es un método que se declara **en línea**, en el momento de crear una instancia de un delegado



# Métodos anónimos - sintaxis

La sintaxis de un método anónimo incluye los siguientes componentes

- La palabra clave **delegate**
- La **lista de parámetros** (si son necesarios)
- El **bloque de sentencias** con la implementación del método



# Métodos anónimos

```
class Program{  
    static void Main(){  
        Cuenta5 cont=new Cuenta5();  
        Console.WriteLine("Cuenta 5 segundos");  
        cont.SegCumplido+=contSegCumplido;  
        Console.WriteLine("Pasaron 5 segundos");  
        Console.ReadKey();  
    }  
  
    static void contSegCumplido(object sender,  
                                SegCumplidoEventArgs e){  
        Console.WriteLine("Faltan {0} seg.",e.Valor);  
    }  
}
```

Un método como el caso de **contSegCumplido** que no será invocado desde ninguna otra parte del programa podría ser reemplazado por un método anónimo

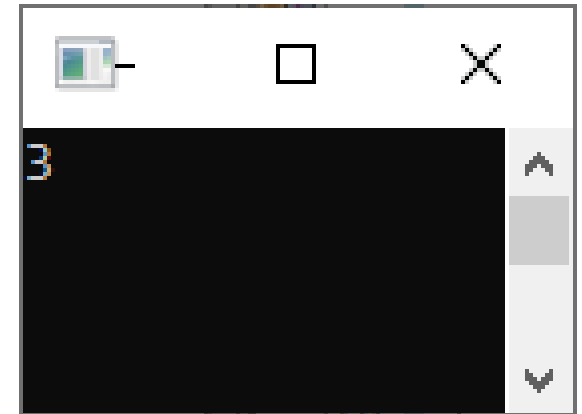
# Métodos anónimos

```
class Program{  
    static void Main(){  
        Cuenta5 cont=new Cuenta5();  
        Console.WriteLine("Cuenta 5 segundos");  
        cont.SegCumplido+=  
            delegate(object sender, SegCumplidoEventArgs e) {  
                Console.WriteLine("Faltan {0} seg.",e.Valor);  
            };  
        Console.WriteLine("Pasaron 5 segundos");  
        Console.ReadKey();  
    }  
}
```

# Métodos anónimos – variables externas

Los métodos anónimos pueden acceder a sus variables locales y a las definidas en el entorno que lo rodea (variables externas).

```
delegate void miDelegado();  
class Program{  
    static void Main(){  
        int x = 1;  
        miDelegado miDel;  
        miDel = delegate()  
        {  
            int y = 2;  
            Console.WriteLine(x + y);  
        };  
        miDel();  
        Console.ReadKey();  
    }  
}
```




Acceso a la variable  
externa x

# Expresiones Lambda

- Los **métodos anónimos** se introdujeron en **C# 2.0**. Las **expresiones Lambda** se introdujeron en **C# 3.0** con el mismo propósito pero con una **sintaxis más reducida**.
- Se puede **transformar** fácilmente un **método anónimo** en una expresión **lambda** haciendo lo siguiente:
  - **Eliminar** la palabra clave **delegado**.
  - **Colocar** el operador lambda, **=>**, entre la lista de parámetros y el cuerpo del método anónimo.

```
miDel = delegate(int x) {return x+1;};
```

Método  
anónimo



```
miDel = (int x) => {return x+1;};
```

Expresión  
Lambda



# Expresiones Lambda

Pero aún es posible otras simplificaciones sintácticas

- Si no existen parámetros **ref** o **out**, el tipo de los parámetros puede omitirse

```
miDel = (x) => {return x+1;};
```

- Si hay un único parámetro, pueden omitirse los paréntesis

```
miDel = x => {return x+1;};
```

- Si el bloque de instrucciones es sólo una expresión de retorno, puede reemplazarse todo el bloque por la expresión de retorno

```
miDel = x => x+1;
```

Nota: Si el delegado no tiene parámetros se deben usar paréntesis vacíos:

```
()=>{. . . }
```