Tutorial de C#
Delegados y Eventos

Por: Óscar López, M.Sc.

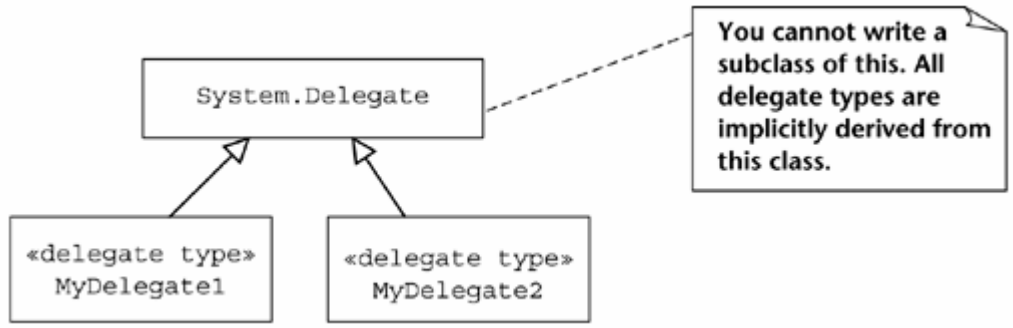
olopez@uniandino.com.co

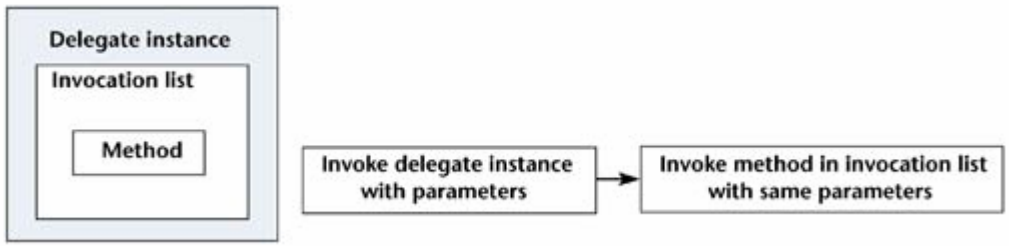
DELEGADOS

Delegados

- Son unos objetos que implícitamente extienden de `System.Delegate` y definen un tipo referencia soportado directamente por el lenguaje
- Su labor es encapsular uno o más métodos definidos en otros objetos. Tradicionalmente, esta capacidad es implementada por medio de punteros a funciones
- Técnicamente, un delegado describe la firma de un método como un tipo, y cualquier método con la misma firma puede ser añadido a la lista de invocación del delegado
- Creados para las situaciones en las que se necesita llevar a cabo una acción pero no se sabe de antemano qué método llamar o sobre cuál objeto invocarla. Se crea un delegado y se deja que los detalles particulares sean establecidos más adelante

Delegados - Estructura





Delegados - Declaración

```
[[atributos]] [modificador de acceso] delegate
tipo identificador (lista de parámetros,)
```

- Un tipo delegado se debe declarar antes de poder usarlo
- La declaración no lleva { } y se puede declarar dentro de otro tipo (clase, interfaz) o por fuera de una clase
- Sólo los métodos que tengan las mismas características que la declaración del tipo delegado (es decir, que sean compatibles con éste) le pueden ser añadidos:
  - El mismo tipo de retorno
  - El mismo número de parámetros
  - El mismo tipo de parámetros en el mismo orden

Delegados - Instanciación

- Un delegado se puede “instanciar” usando la palabra clave `new` y pasándole a su constructor alguno de los posibles parámetros:
  - Un método estático compatible con su tipo
  - Un método de instancia compatible con su tipo
  - Otro delegado
- Después de ser instanciado, se le pueden añadir nuevos métodos
- Una instancia de un delegado puede verse como la representación de el o los métodos que contiene
- Internamente, una instancia de un delegado guarda una lista de invocación con todos los métodos que representa, organizados en el mismo orden en el que fueron añadidos

Delegados - Invocación

- Cuando se invoca un delegado, se le pasan los parámetros con los que fue declarado
- El delegado se encarga de pasarle dichos parámetros a la lista de invocación con los métodos que encapsula, y los invoca uno detrás de otro
- Es posible declarar un delegado para que reciba parámetros por referencia. Puesto que a cada método en la lista de invocación se le pasan los mismos parámetros, los cambios en alguno de éstos serán pasados a los siguientes métodos

Delegados - Ejemplo

```
1.      using System;

2.      public delegate int MyDelegate (int i);

3.      class TestClass {

4.        static int Double (int val) {
5.          Console.WriteLine("ejecutando Double");
6.          return val*2;
7.        }

8.        int Triple (int val) {
9.          Console.WriteLine("ejecutando Triple");
10.         return val*3;
11.       }

12.       public static void Main() {

13.         TestClass tc = new TestClass();
14.         MyDelegate d1, d2;

15.         d1 = new MyDelegate(TestClass.Double);
16.         d2 = new MyDelegate(tc.Triple);

17.         Console.WriteLine(d1(3));
18.         Console.WriteLine(" ---------- ");
19.         Console.WriteLine(d2(5));
20.         Console.ReadLine();

21.       }

22.     }
```

Delegados - Optimizaciones

Métodos: Instancia vs. Estáticos

Es conveniente declarar como estáticos los métodos que se pasan al delegado. Pero declararlos como de instancia puede ser ligeramente más eficiente

Delegados Estáticos

Es conveniente asignar la responsabilidad de instanciar el delegado a la clase que implementa el método compatible con éste, implementado como un campo estático

Delegados como Propiedades

Proporciona encapsulamiento, permite instanciar el delegado sólo cuando se necesita, sólo lectura

Delegados – Estableciendo el Orden de Ejecución

- Los delegados sirven para implementar un sistema en el que un usuario pueda decidir dinámicamente el orden en que se ejecuta un conjunto de operaciones
- Se crean delegados para cada operación y se los añade a una colección en el orden en el que quieren ejecutarse
- Simplemente, se itera sobre la colección invocando cada delegado
- Se tiene la flexibilidad de determinar dinámicamente qué métodos se llaman, en qué orden y con qué frecuencia

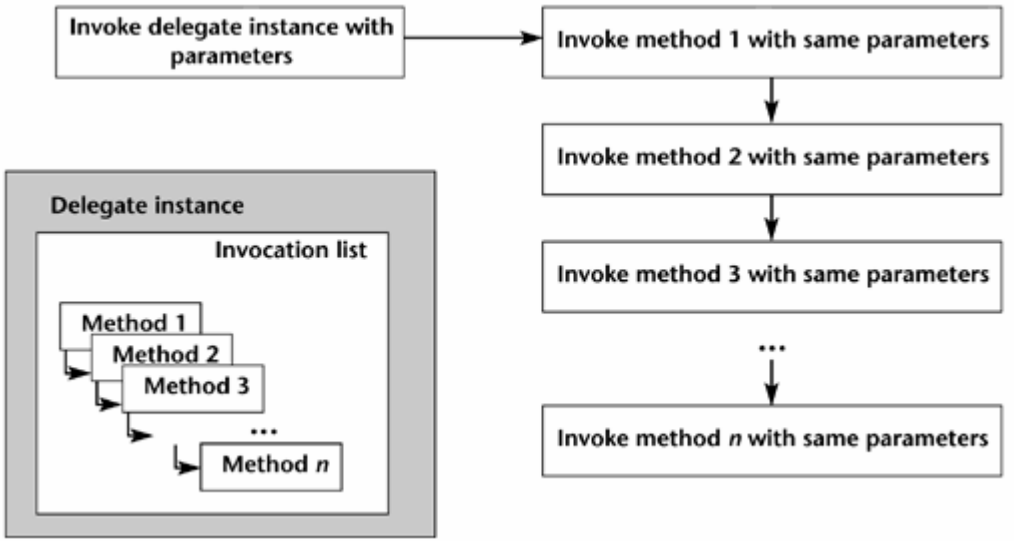
Multicasting

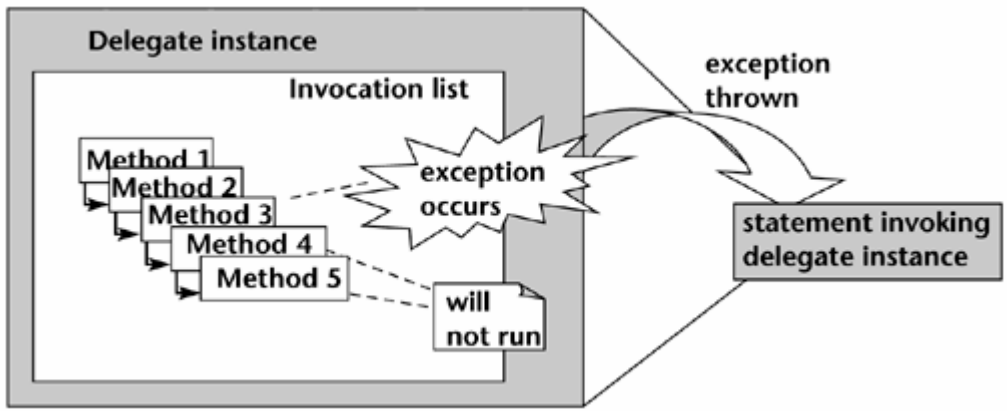
- En ocasiones, es necesario poder llamar dos o más métodos simultáneamente a través de un mismo delegado
- Un mismo delegado puede encapsular varios métodos -sean estáticos o dinámicos- con tal de que todos sean compatibles con su definición de tipo, sin importar que estén en distintas clases
- Se usan los operadores +, −, += y −= para añadir y quitar métodos a un delegado
- Los métodos en la lista de invocación se insertan y se ejecutan en el mismo orden en el que fueron añadidos

Multicasting

- Es posible añadir un mismo método más de una vez
- Si se quita un método que aparece más de una vez, se remueve la última ocurrencia de éste
- No pasa nada si se quita un método que no estaba en la lista de invocación
- Un delegado sin métodos tiene valor `null`
- Si el delegado retorna un valor, sólo se retorna el resultado de invocar el último método
- Si ocurre una excepción durante la invocación de un método, los demás métodos de la lista no son invocados

Multicasting - Invocación





Multicasting - Ejemplo

```
1.      using System;

2.      public class MulticastTest {

3.        delegate int MyDelegate (int i);

4.        static int Double (int val){
5.          Console.WriteLine("running Double");
6.          return val*2;
7.        }

8.        int Triple (int val){
9.          Console.WriteLine("running Triple");
10.         return val*3;
11.       }

12.       static public void Main() {

13.         MulticastTest tc = new MulticastTest();
14.         MyDelegate d1, d2, compositeDelegate;

15.         d1 = new MyDelegate(MulticastTest.Double);
16.         d2 = new MyDelegate(tc.Triple);
17.         compositeDelegate = d1 + d2;

18.         int retVal = compositeDelegate(3);
19.         Console.WriteLine(retVal);
20.         Console.ReadLine();

21.         }

22.     }
```

Delegados - Usos Avanzados

Valores de Delegados Multicast

Para recuperar todos los valores retornados por un delegado *multicast*, iterar sobre el resultado de llamar `GetInvocationList()`

Invocando Delegados Asíncronos

Si un método delegado se tarda mucho tiempo en retornar, se puede invocar de manera asíncrona usando `BeginInvoke()`. Pero, ¿Cómo saber cuándo retorna?

Delegados - Usos Avanzados

Métodos Call Back

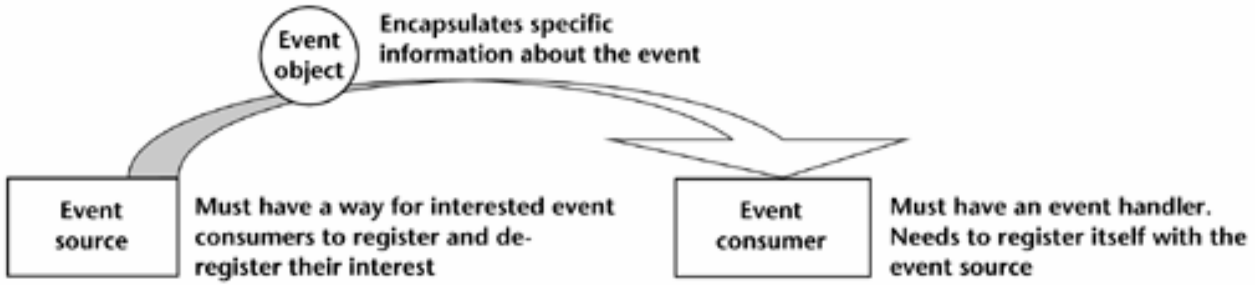
Pueden implementarse usando delegados; resuelven el problema de los delegados asíncronos. Basta con proporcionar un método que implemente este delegado:

```
[Serializable] public delegate void AsyncCallback(IAsyncResult ar)
```

1. Se saca la lista de invocación del delegado
2. Se llama `BeginInvoke()` sobre cada elemento
3. El primer parámetro será un nuevo delegado de tipo `AsyncCallback`, el segundo nuestro propio delegado
4. Cuando el *call back* sea llamado, pasará un objeto `IAsyncResult` que tiene nuestro delegado en la propiedad `AsyncState`
5. Se convierte el objeto retornado y se llama `EndInvoke()` sobre éste; se recupera su valor

EVENTOS

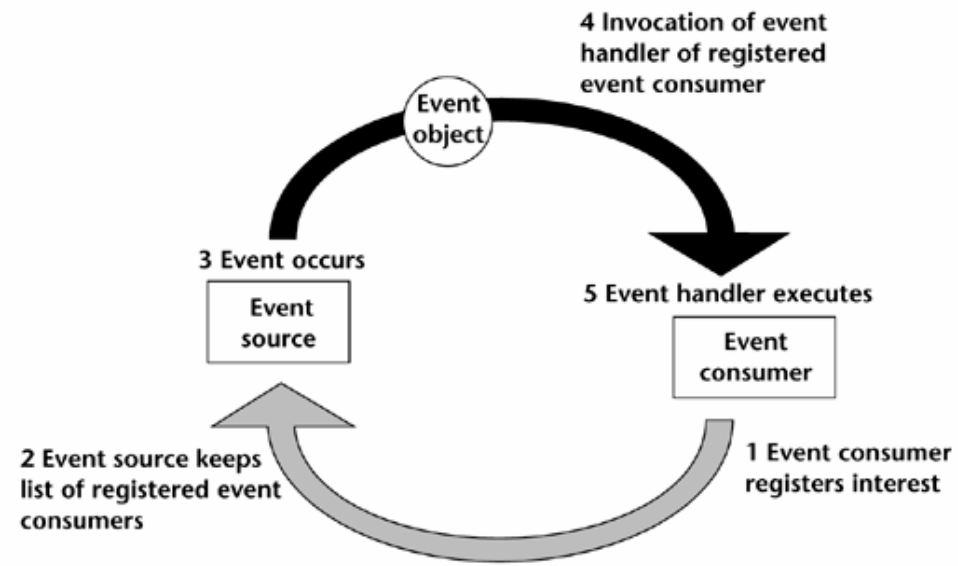
Modelo Genérico de Eventos



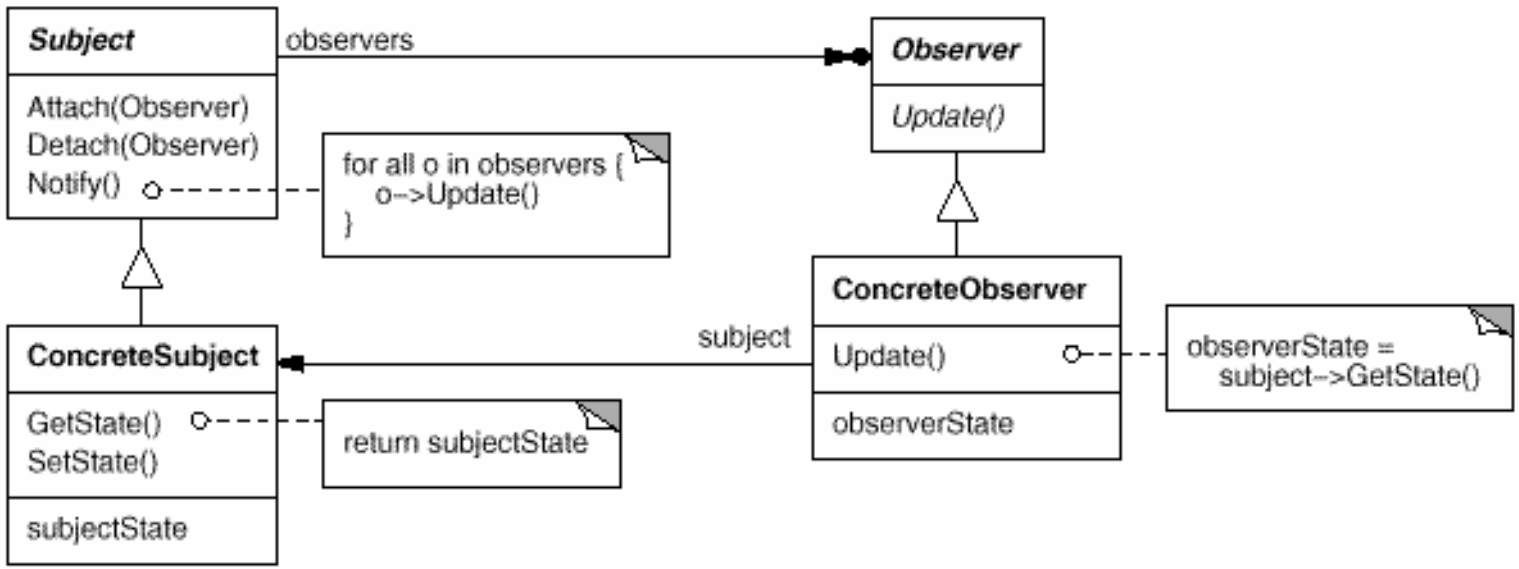
Rol	Descripción
Fuente de Eventos	El objeto que (potencialmente) causa un evento. Proporciona un mecanismo para que los consumidores se registren, y los mantiene en una lista de forma tal que, cuando ocurre un evento, todos son notificados
Consumidor de Eventos	El objeto interesado en escuchar un evento en particular. Tiene un método especial, conocido como *handler* de eventos que recibe como parámetro un objeto evento. Fuentes y consumidores pueden tener una relación muchos-a-muchos
Objeto Evento	Se crea un nuevo objeto evento cuando un evento ocurre en la fuente. Este objeto se pasa al método *handler* del consumidor de eventos como un parámetro. Encapsula información específica al evento, la cual puede ser extraída por el consumidor

Modelo Genérico de Eventos

1. El consumidor de eventos registra su interés con una fuente de eventos y proporciona el nombre del *handler* que debe ser invocado cuando ocurra un evento
2. La fuente de eventos mantiene referencias a todos los consumidores registrados en una lista interna
3. Cuando ocurre un evento, la fuente crea una nueva instancia de un objeto evento y le pasa información específica al evento
4. La fuente recorre la lista de consumidores registrados e invoca cada *handler* de eventos, pasándoles el objeto evento como parámetro
5. Se ejecuta el *handler* de eventos en el consumidor. La información del evento puede ser obtenida del objeto evento



Patrón *Observer*



Eventos en C#

Fuente de Eventos

Una clase que declara un miembro delegado y su correspondiente miembro evento e invoca este último cuando corresponda

Consumidor de Eventos

Proporciona un método *handler* de eventos compatible con el delegado de la fuente y lo registra con el miembro evento de la fuente usando el operador `+=`

Objeto Evento

General, pero no necesariamente, es una instancia de la clase `System.EventArgs` o de una de sus sub-clases, a la que se le han añadido campos con información adicional sobre el evento

Eventos en C#

```
[[atributos]] [modificadores]
event tipo identificador
```

- Un miembro evento es un tipo especial de delegado, declarado con la palabra clave `event`
- El tipo es el delegado al cual se va a asociar este evento: a cada evento corresponde un delegado, pero varios eventos pueden tener el mismo tipo de delegado
- Por convención, el identificador del delegado termina con la palabra `Handler` y el del evento empieza con la palabra `On`
- Los métodos que encapsula (compatibles con el delegado), no son otros que los *handlers* de eventos
- Lo que diferencia a un miembro evento de un miembro delegado, es la presencia de algunas restricciones de acceso adicionales: sólo puede ser invocado por la clase que lo define y sólo pueden usarse los operadores `+=` y `-=` para añadir y quitar métodos

Eventos en C#

- Generalmente, los delegados asociados a un evento son declarados con la siguiente firma:

```
public delegate void
AlgunHandler(object fuente, EventArgs objEvento)
```

- Es decir: retornan `void`, el primer parámetro es la fuente del evento y el segundo una instancia de `EventArgs` o de alguna de sus sub-clases
- Dado que los delegados con esta firma son muy frecuentes, se puede utilizar `EventHandler`, un delegado proporcionado por el *framework* que tiene la firma necesaria y no necesita ser declarado explícitamente
- Cualquier objeto puede ofrecer un conjunto de eventos a los cuales se pueden suscribir unos consumidores. Cuando la fuente señala la ocurrencia de un evento, los consumidores son notificados
- Se trata de una implementación del patrón *Observer*, pero sin tener que preocuparse por definir e implementar interfaces o clases abstractas
- La fuente queda desacoplada de sus consumidores, y pueden variar independientemente