



# Corso di C#

## Le classi: delegate

(Parte 9)

DI MATTEO BACCAN

Implementare un sistema di callback, efficiente ed elegante, non è mai un'impresa semplice da realizzare...

Durante la stesura di un programma può capitare che, un semplice passaggio di parametri, possa limitare la potenza di un algoritmo o possa rendere complicata una personalizzazione di una certa parte di codice. Per tale ragione, quando si progetta un linguaggio di programmazione, si tiene sempre conto della possibilità di poter passare come parametro, un riferimento ad una particolare funzione o ad un metodo. In questo modo, un ipotetico algoritmo, ha possibilità di integrarsi con del codice esterno. Un esempio famoso di applicazione di tale tecnica è dato dalle API di Windows, nelle quali è molto diffusa la possibilità di passare un puntatore a funzione.

### UN PO' DI PRATICA

Per poter capire come funzionano queste funzioni, chiamate *callback function* e come sono implementate all'interno di C#, tramite i metodi *delegate*, è interessante fare riferimento ad un esempio che illustri un'esigenza reale, come l'ordinamento di due vettori di oggetti di diverso tipo.

Dal punto di vista concettuale, un algoritmo di ordinamento, è perfettamente identico in tutti i casi in cui venga applicato. L'aspetto che varia è dato dal tipo di elemento che si vuole ordinare. Conseguentemente varia anche il controllo che si effettua per capire se un tale elemento è precedente o successivo ad un altro. Supponendo di aver scritto un'efficiente classe di sort, specifica per il primo vettore, ci si trova ora di fronte al problema di applicare lo stesso algoritmo anche al secondo caso. Come raggiungere questo scopo? Vi sono diverse tecniche utilizzabili, alcune molto eleganti ed altre assolutamente sconsigliabili.

Un primo approccio potrebbe suggerire la duplicazione della classe, con la modifica di tutti i riferimenti ad oggetti, dal primo al secondo tipo. Questo modo di operare risulta però estremamente dispendioso in fase di correzione di errori di programmazione e produce una quantità considerevole di codice inutile. Si potrebbe invece pensare di creare una classe base che dispone dell'algoritmo di ordinamento e di un metodo di confronto che può essere sostituito da una classe derivata. Questo tipo di approccio è sicuramente meglio del precedente, anche se obbliga a creare una classe, ogni volta che si vuole ordinare un nuovo vettore.

Esiste però un'ulteriore possibilità: chiamare un metodo di confronto, contestuale al tipo di dato che si vuole ordinare.

Per fare in modo che quest'operazione sia possibile, l'algoritmo di ordinamento dovrà effettuare una chiamata al programma che lo ha invocato, che fornirà l'indirizzo di un metodo in grado di effettuare il confronto. Questo tipo di approccio si chiama appunto *callback*.

In alcuni linguaggi, come C, il concetto di callback è ottenuto tramite il passaggio dell'indirizzo della funzione che deve

essere eseguita. Ovviamente questa tecnica è brutale, anche se estremamente efficiente e permette di ottenere un risultato che è scarsamente controllabile e quindi fonte di possibili problemi. Vediamo ora come C# abbia definito, in modo rigoroso, i meccanismi con i quali è possibile creare un metodo di callback, per evitare errori in fase di definizione e d'uso.

### DELEGATE

Prima di tutto occorre capire come poter indicare un metodo di callback. Esiste un costrutto, chiamato *delegate*, in grado di specificare che il metodo che si sta scrivendo, non è un metodo vero e proprio, ma la struttura di un metodo che verrà definito a runtime. Vediamo ora, tramite un esempio, come si crea un metodo *delegate*:

```
public delegate string testMethod( string cNome );
```

In questo caso vi sono due importanti differenze, rispetto alla classica implementazione di un metodo.

La prima è data dalla parola *delegate*, posta prima del tipo di ritorno. La seconda è l'assenza del corpo.

L'implementazione del corpo di *testMethod* sarà infatti a carico del programma chiamante che vorrà utilizzare la classe nella quale è contenuto. Vediamo ora come inserire questo metodo all'interno di una classe:

```
class testDelegate {  
    public delegate string testMethod( string cNome );  
    public void visualizzaNome( testMethod metodo ){  
        Console.WriteLine( metodo("maTTEo") );  
    }  
}
```

*testMethod* è ora parte di una classe chiamata *testDelegate*. All'interno di tale classe vi è un secondo metodo, chiamato *visualizzaNome*, che riceve come parametro una variabile di nome *metodo* e di tipo *testMethod*.

In questo modo il compilatore accetterà come parametro solamente metodi che hanno la stessa struttura di *testMethod*. La fase successiva è quella di costruire un metodo che abbia tali requisiti.

### UTILIZZO A RUNTIME

Il metodo che dobbiamo definire riceve e restituisce un variabile di tipo *string*. Per verificare che il meccanismo di callback funzioni correttamente possiamo limitarci a scrivere un metodo che, preso il parametro in ingresso, lo metta in minuscolo prima di restituirlo. Questa operazione si può ottenere nel seguente modo:

```
public static string minuscolo( string cNome ){  
    return cNome.ToLower();  
}
```

Il metodo *minuscolo*, riceve il parametro *cNome* e restituisce il risultato di *ToLower()*.

Ora non ci resta che creare un oggetto della classe *testDelegate* e richiamare il metodo *visualizzaNome* creando un parametro di tipo *testMethod*:

```
public static void Main() {
    testDelegate test = new testDelegate();
    test.visualizzaNome( new testDelegate.testMethod
                        ( minuscolo ) );
}
```

La creazione di tale metodo può essere fatta tramite una semplice *new*, specificando il tipo di dato che si intende creare, che è *testDelegate.testMethod*. La semplicità di questo approccio è sicuramente estrema e permette al programmatore di poter creare, in modo molto lineare, delle integrazioni ai propri algoritmi con del codice esterno.

Un altro vantaggio è dato dal controllo che si riesce ad avere con una definizione di questo genere. Se provate a variare i parametri del metodo *minuscolo* e a compilare questo programma, noterete subito un errore, che vi avviserà che, i parametri definiti non sono uguali a quelli del metodo che intendete utilizzare.

Se avessimo provato a costruire un simile programma in C, il compilatore non se ne sarebbe accorto e avrebbe generato un errore di runtime, dato che in C il passaggio di parametri si limita ad un semplice indirizzo di memoria.

## MULTICASTING

Un risvolto interessante dei metodi delegate è dato dal fatto che possono essere sommati fra di loro, in modo da formare una lista, richiamabile come se si trattasse di un singolo metodo. Questo aspetto risulta particolarmente utile in alcune applicazioni pratiche come la gestione degli eventi in cui, in un particolare momento, più oggetti devono essere avvisati del verificarsi di una certa condizione.

Proviamo allora a modificare l'esempio sul quale abbiamo appena lavorato, in modo da gestire una lista di metodi, al posto di uno solo. Inizieremo col modificare il metodo *delegate*.

### RIQUADRO 1 Esempio d'uso di delegate

```
using System;
public class testMain {
    public static string maiuscolo( string cNome ) {
        return cNome.ToUpper();
    }
    public static string minuscolo( string cNome ) {
        return cNome.ToLower();
    }
    public static void Main() {
        testDelegate test = new testDelegate();
        test.visualizzaNome( new
            testDelegate.testMethod( maiuscolo ) );
        test.visualizzaNome( new
            testDelegate.testMethod( minuscolo ) );
    }
}
class testDelegate {
    public delegate string testMethod( string cNome );
    public void visualizzaNome( testMethod metodo ){
        Console.WriteLine( metodo("maTTEo") );
    }
}
```

Dato che il nostro obiettivo è invocare una lista di valori e non più un singolo elemento, modificheremo il ritorno di *testMethod*, da *string* a *void*. Inoltre, faremo il modo che sia il metodo stesso a visualizzare il dato:

```
class testDelegate {
    public delegate void testMethod( string cNome );
    public void visualizzaNome( testMethod metodo ){
        metodo("maTTEo");
    }
}
```

A questo punto varieremo anche il programma chiamante, andando a definire due metodi di uguale sintassi, ma in grado di compiere due operazioni completamente diverse, come la visualizzazione del dato in maiuscolo e in minuscolo:

```
public static void maiuscolo( string cNome ) {
    Console.WriteLine( cNome.ToUpper() );
}
public static void minuscolo( string cNome ) {
    Console.WriteLine(cNome.ToLower() );
}
```

Una volta che tutto è stato definito, siamo pronti a creare due variabili di tipo *testMethod* e a passarne la loro somma al metodo *visualizzaNome*. Il compilatore creerà un solo oggetto e lo passerà al metodo richiamato:

```
public static void Main() {
    testDelegate test = new testDelegate();
    testDelegate.testMethod primo, secondo;
    primo = new testDelegate.testMethod( maiuscolo );
    secondo = new testDelegate.testMethod( minuscolo );
    test.visualizzaNome( primo + secondo );
}
```

Essendo però 2 i metodi, il singolo richiamo all'interno di *visualizzaNome*, andrà ad invocarli entrambi.

## CONCLUSIONI

Come si è potuto osservare, la definizione di un metodo *delegate* offre, in modo semplice e strutturato, la possibilità di chiamare del codice esterno alla classe nella quale viene utilizzato. Il tutto viene fatto con quel pizzico di formalismo che evita la creazione di codice ambiguo, fonte di probabili errori. La possibilità di combinare fra loro una serie di *delegate* arricchisce il codice di interessanti prospettive, come ad esempio l'integrazione di *delegate* provenienti da linguaggi diversi.

## BIBLIOGRAFIA

- [1] Eric Gunnerson, "A Programmer's Introduction to C#", ISBN 1-893115-86-0
- [2] Andrew Troelsen, "C# and the .NET Platform", ISBN 1-893115-59-3
- [3] <http://www.infomedia.it/artic/Baccan>, sito del corso C#

**Matteo Baccan** è uno specialista di progettazione e sviluppo in C++, C#, Java e Lotus Notes. Attualmente si occupa dello studio di tecniche avanzate di programmazione in ambito Internet/Intranet tramite Lotus Notes e Java, presso un'importante azienda italiana. Si possono consultare alcune sue pubblicazioni presso <http://www.infomedia.it/artic/Baccan>. Può essere contattato all'indirizzo [baccan@infomedia.it](mailto:baccan@infomedia.it)