

# Introduzione a Delegati ed Eventi

da <http://escher07.altervista.org>

## Generalità su puntatori a funzione in C++ e delegati/eventi in C#

Uno degli argomenti più incasinati del C++ sono i puntatori a funzione. Il concetto di per sé è semplice: al momento in cui si scrive il programma non si sa quale funzione chiamare, che dipenderà dai dati. Scriviamo un elementare programmino in C++ (di quelli classici su esempi matematici che non servono a nulla...) per cominciare a capire di cosa stiamo parlando.

```
#include "stdafx.h"
#include "iostream.h"

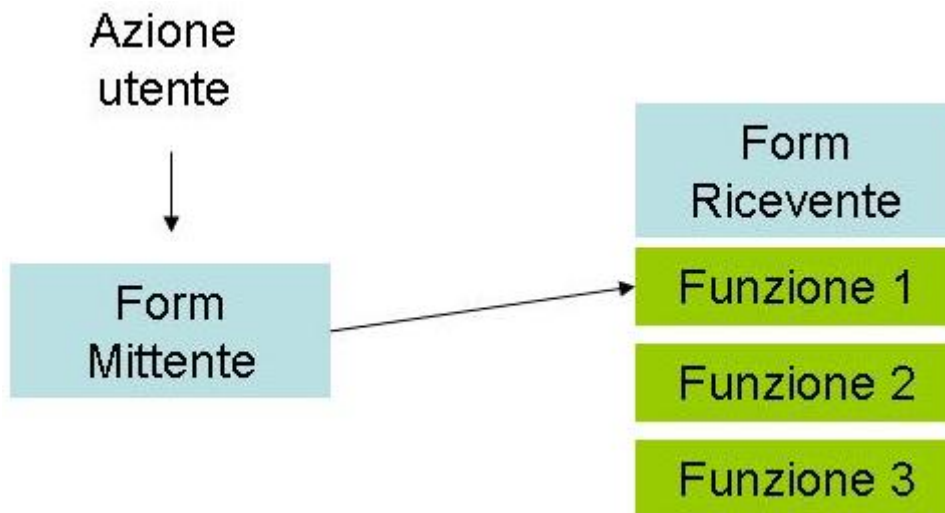
double (*CalcolaArea)(double Base, double Altezza);
double AreaQuadrato(double Base, double Altezza)
{
    return Base*Altezza;
}
double AreaTriangolo(double Base, double Altezza)
{
    return Base*Altezza/2;
}

int main(int argc, char* argv[])
{
    double b=0;
    double h=0;
    double A=0;
    char figura;
    cout << "Base : ";
    cin >> b;
    cout << "Altezza : ";
    cin >> h;
    cout << "(Q)uadrato o (T)riangolo : ";
    cin >> figura;
    if (figura=='T')
    {
        CalcolaArea = &AreaTriangolo;
    }
    else
    {
        CalcolaArea = &AreaQuadrato;
    }
    cout << "Area = ";
    cout << (*CalcolaArea)(b,h);
    cout << "\n";
    return 0;
}
```

Quella evidenziato in **rosso** è la dichiarazione del "delegato" ossia di un puntatore ad una funzione di un dato tipo e con certi argomenti (tipo=double e argomenti=2 double nell'esempio). A quale funzione? Il punto è proprio questo ..... non si dice! Si attacca al puntatore in questione l'indirizzo della funzione giusta per quell'input (quella che calcola l'area del quadrato se di essa si erano inseriti i dati o del triangolo nel caso opposto) e lo si fa col codice in **blu**. L'output finale è ottenuto semplicemente passando i giusti parametri di base ad altezza alla funzione puntata dal nostro delegato, che è quella giusta per i passaggi precedenti.

In questo esempio non ho introdotto per semplicità classi. Potevo però, in modo un po' più elegante in OOP, introdurre due classi ciascuna la prima (es. Front) contenente l'interfaccia con l'utente ed in puntatore a funzione, la seconda (es. Back) con le varie funzioni di calcolo dell'area per tutti i casi possibili, dei quali ne viene agganciata una runtime.

Dove sta l'analogia con la programmazione in windows? Semplice : le form sono classi e l'interazione fra due form si può schematizzare facilmente così:



Una form/classe è la main e ad un certo punto passa il controllo ad una figlia con la quale quindi interagisce direttamente l'utente. Le azioni raccolte dalla figlia devono poter influenzare la main.

Ora il punto è che non è preventivabile quale Funzione della classe ricevente sarà chiamata perché dipende dall'evento "raccolto" dalla classe mittente. Affinché ci possa essere la comunicazione il tutto si fa definendo una terza classe che è appunto il delegato e che implementa una "callback" perché è una retroazione della figlia nei confronti della "madre".

Un esempio concreto? Il task manager che gestisce i programmi in esecuzione. Esco da un programma e la lista dei processi in esecuzione si modifica.

Detto questo diamo un'occhiatina alla documentazione Microsoft. Allora a questo indirizzo:

[http://msdn2.microsoft.com/it-it/library/17sde2xt\(VS.80\).aspx](http://msdn2.microsoft.com/it-it/library/17sde2xt(VS.80).aspx)

trovate un esempio base di utilizzo di delegati ed eventi che si riferisce ad una applicazione console. Analizzandolo si può arrivare ad descrivere quali sono gli elementi fondamentali del processo, le loro caratteristiche e le relative azioni, ovvero in sostanza quali ad uno schema di questo tipo:

Nome	<b>AlarmEventArgs</b>
Ruolo	Contenitore Dati Evento
Tipo	Classe
Costruttore	Usuale con i parametri di input
Proprietà Input	Dati di input per l'evento
Proprietà Output	Dati di output calcolati come funzione dei precedenti

Nome	<b>AlarmEventHandler</b>
Ruolo	Delegato

Tipo	Dichiarazione (manca "class").
Argomento	sender (tipo <b>object</b> )
Argomento	e (tipo <b>AlarmEventArgs</b> )

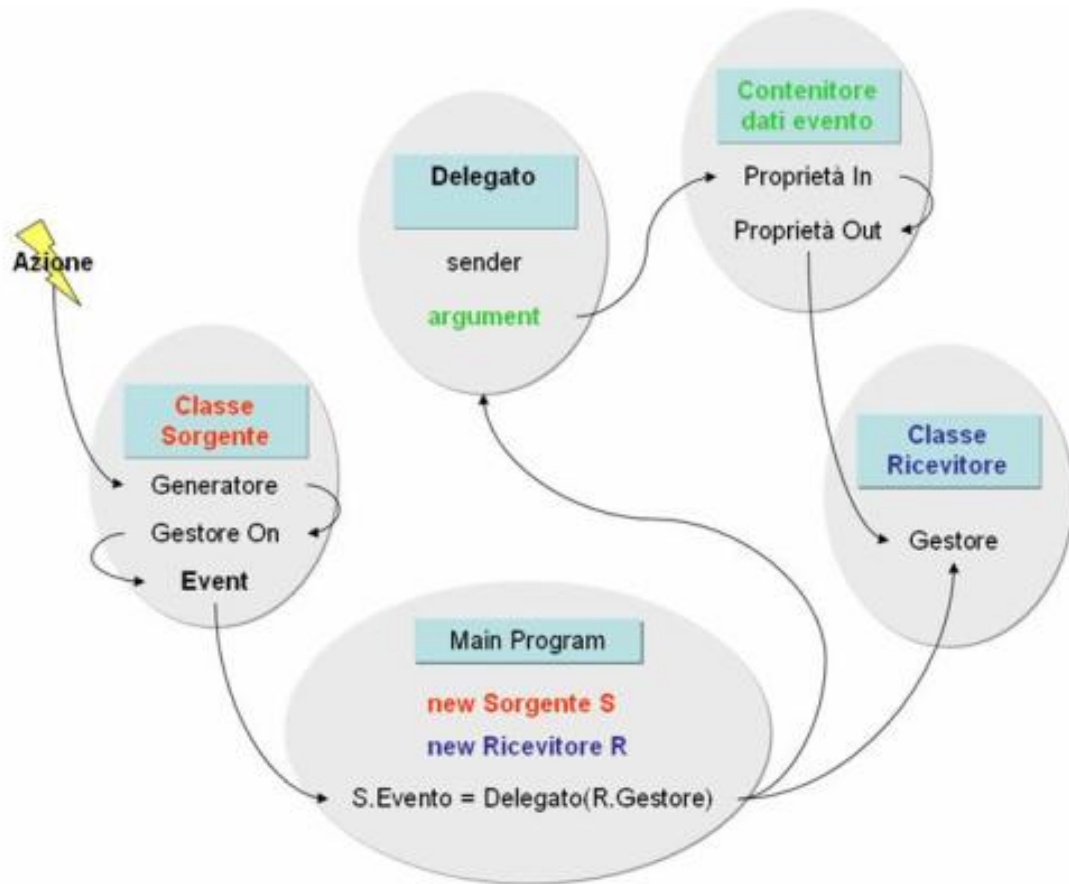
Nome	AlarmClock
Ruolo	Sorgente dell'evento
Tipo	Classe
Dichiarazione Membro Event	Tipo <b>AlarmEventHandler</b> nome <b>Alarm</b>
Membro On	Nome <b>OnAlarm</b> con argomento di tipo AlarmEventArgs. Istanza una nuova variabile di tipo evento (tipo AlarmEventHandler) e ci appiccica il membro precedentemente dichiarato (Alarm). Chiama il costruttore della variabile precedentemente definita con sender=this=istanza della classe corrente ed argomento quello che ha ricevuto in ingresso. Questo costruttore qui non c'è quindi va a prendere il delegato.
Membro Generatore Evento	Crea in corrispondenza della azione voluta (es. x>5 o pressione di bottone) una nuova istanza del Contenitore Dati Evento (AlarmEventArgs) con i corrispondenti parametri di input. Chiama il proprio Membro On (OnAlarm) con l'istanza definita come argomento.

Nome	WakeMeUp (non influisce)
Ruolo	Ricevitore dell'evento
Tipo	Classe
Membro Gestore Evento	Nome <b>AlarmRang</b> (non legato ai precedenti ma proprio quel nome deve essere chiamato dal Main) Stesso tipo (void, int etc..) del delegato e stessi argomenti. Può chiamare i metodi del Contenitore Dati Evento e accedere alle proprietà tramite la sintassi ((sorgente)sender).proprietà.

Nome	AlarmDriver (non influisce)
Ruolo	Programma Principale
Tipo	Classe
Main	Istanza la classe sorgente. Istanza la classe ricevente. Collega il membro event della classe sorgente (Alarm) con una nuova istanza del delegato (AlarmEventHandler) – sono dello stesso tipo – con argomento il gestore evento della classe ricevitore (AlarmRang), ovvero fa puntare Alarm (che ha accesso alle proprietà della classe sorgente) di AlarmClock ad AlarmRang (dove viene generato l'output) di WakeMeUp.

Nota : ho evidenziato in rosso i legami fra i nomi. In sostanza affinché tutto funzioni occorre che se nella classe ricevente c'è un evento che si chiama "Alarm" il suo gestore si deve chiamare "**OnAlarm**" e così via.

Schematizzando succede questo:



Per semplicità abbiamo legato l'azione alla classe sorgente mentre in realtà sarebbe legata all'istanza. Il delegato in sostanza è un puntatore ad una funzione definita nel suo argomento. La classe sorgente non sa quale sarà il gestore dell'evento nella classe ricevitore : per questo al suo interno contiene un "delegato" che può puntare ad una classe di funzioni senza specificare precisamente a quale punterà, cosa che sarà definita al momento dell'istanziamento. Il delegato più il suo argomento recuperano la funzione gestore che potrà usare proprietà e metodi definiti nel contenitore dati, recuperati parimenti dall'istanziamento del delegato.

Bene, visto tutto questo poniamoci il seguente:

### **Problema**

Supponiamo di avere un'auto dotata di tachimetro che fa vedere il valore della velocità corrente e lo aggiorna tutte le volte che cambia. L'auto parte da ferma. La velocità della macchina è decisa dal sistema di controllo che può fare le seguenti azioni:

- Accelerare (+10 Km/h alla pressione, nessun effetto se  $v = 150\text{Km/h}$ )
- Frenare (-10 Km/h alla pressione, nessun effetto se  $v = 0\text{ Km/h}$ )

Il problema (che potrebbe essere risolto facendo aggiornare la velocità a seguito della pressione di un bottone in un'unica form) deve essere risolto con due maschere in cui la prima genera un evento nella seconda.

### **Implementazione**

Definiamo tre form (classi):

- Main
- Auto

- Controllo

Dalla Main si può decidere di istanziare una nuova coppia di Auto/Controllo o eliminare la corrente.

Per poterlo fare il Main ha una struttura di questo tipo:

```
public partial class Main : Form
{
    public Main()
    {
        InitializeComponent();
    }

    Auto auto = new Auto();
    Controllo controllo = new Controllo();

    private void button1_Click(object sender, EventArgs e)
    {
        controllo.Show();
        auto.Show();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        auto.Close();
        controllo.Close();
    }
}
```

Nel sistema di controllo mettiamo due bottoni:

- Accelera
- Frena

Ciascuno dei quali determina una variazione della velocità (proprietà esposta) nell'auto. In questa form esiste una progress bar che fa vedere la velocità corrente.

La prima difficoltà è : come faccio a dire quando viene premuto il bottone Accelera si aggiorna la velocità dell'istanza di un'altra classe ? Se fosse una proprietà di sé stessa (ovvero nell'ambito della stessa maschera/classe) non ci sarebbero problemi ma qui per accederci direttamente dovrebbe essere una proprietà static e quindi non potrebbe controllare il `this.progressBar1` di `auto`.

La soluzione è che la pressione del tasto in controllo scatena un'evento "Aggiorna Velocità" in `auto`. L'evento aggiorna la velocità e chiama un refresh della `this`.

Per fare questo occorrono i seguenti passi:

- Definire la classe contenitore dati evento;
- Dichiarare il delegato;
- Nella classe in cui l'evento si genererà (ovvero "controllo"):
  - Dichiarare un membro "Event" dello stesso tipo del delegato;
  - Definire un membro "On" con argomento dello stesso tipo del contenitore;
  - Definire un membro che lancia l'evento
- Nella classe in cui l'evento verrà gestito (ovvero "auto")
- Definire la routine di gestione dell'evento

Nota: abbiamo usato "Definire" per indicare i casi in cui si esplica una certa routine in tutti i dettagli, "Dichiarare" per i casi in cui si indica solo la presenza di una routine, il tipo (void / int / string etc...) ed i relativi argomenti. Ricorda che in C/C++ è comune dichiarare prima una funzione e poi definire in seguito il corpo delle istruzioni in cui si esplica.

Operiamo così:

Per il punto 1. aggiungiamo un file Common.cs da utilizzare per le funzioni comuni sul progetto e qui dichiariamo la classe CambiaVelocitàEventArgs in questo modo:

```
public class CambiaVelocitàEventArgs : EventArgs
{
    private int _DeltaVelocità;

    public CambiaVelocitàEventArgs(int DeltaVelocitàIniziale)
    {
        this._DeltaVelocità = DeltaVelocitàIniziale;
    }

    public int DeltaVelocità
    {
        get { return this._DeltaVelocità; }
        set { this._DeltaVelocità = value; }
    }
}
```

Questa classe ha un'unica variabile ovvero il valore della velocità di incremento (se positiva) o di decremento (se negativa) al solito dichiarata come privata con accesso tramite la funzione di accesso `DeltaVelocità` come da teoria della programmazione ad oggetti. Notare che in C# get e set sono nella stessa funzione, mentre in genere in C avremmo avuto due funzioni `GetDeltaVelocità` per la lettura e `SetDeltaVelocità` per la scrittura. In più ha un **costruttore** in cui sulla base del parametro in input si genera una nuova istanza della classe in questione.

Per il punto 2. nello stesso file dichiariamo il delegato così:

```
public delegate void CambiaVelocitàEventHandler(object sender, CambiaVelocitàEventArgs e);
```

Per i punti 3 aggiungiamo codice in Controllo.cs ed in particolare per il punto 3.1

```
public event CambiaVelocitàEventHandler CambiaVelocità;
```

.. per il punto 3.2 ..

```
protected virtual void OnCambiaVelocità(CambiaVelocitàEventArgs e)
{
    CambiaVelocitàEventHandler evCambiaVelocità = CambiaVelocità;
    if (evCambiaVelocità != null)
    {
        evCambiaVelocità(this, e);
    }
}
```

...e per il punto 3.3:

```
private void button1_Click(object sender, EventArgs e)
{
    int delta = +10;
    CambiaVelocitàEventArgs arCambiaVelocità = new CambiaVelocitàEventArgs(delta);
    OnCambiaVelocità(arCambiaVelocità);
}
```

In pratica quando viene premuto il button1 (Accelera) viene generato un nuovo set di argomenti arCambiaVelocità costruito sulla variabile di input variazione velocità = 10 Km/h. Viene poi lanciata la routine di gestione OnCambiaVelocità con questa istanza del set di argomenti che provvede a dichiarare un evento evCambiaVelocità per il quale non alloca nuova memoria (manca la keyword new) ma il cui puntatore punta all'area del delegato, ovvero che ha la stessa struttura del delegato definito nel punto 1. Questo permette di passare a questo delegato i puntatori alla form corrente (this) ed il set degli argomenti correnti (e) attraverso l'istruzione evCambiaVelocità(this, e).

A questo punto il delegato ha tutte le informazioni prelevate dalla form controllo, ovvero il puntatore alla sua istanza e il valore della velocità che deve variare. In tale form (punto 5.) aggiungiamo pertanto la routine che "raccolgerà" il delegato con questo codice:

```
public void CambiaVelocità(object sender, CambiaVelocitàEventArgs e)
{
    int VelocitàFutura;
    VelocitàFutura = this.progressBar1.Value + e.GetDeltaVelocità;
    if ((VelocitàFutura < VelocitàMinima) || (VelocitàFutura > VelocitàMassima))
    {
        // nessuna variazione (supererebbe max o min)
    }
    else
    {
        this.progressBar1.Value = VelocitàFutura;
    }
}
```

Notare l'uso di e.DeltaVelocità per recuperare il delta proveniente dall'altra maschera. Domanda : cosa permette di legare la void CambiaVelocità col delegato? Al momento niente : dobbiamo aggiungere nella button1\_click del main questo legame:

Istanza Sorgente -> Delegato -> Istanza Destinazione

Ovvero il seguente codice:

```
controllo.CambiaVelocità += new CambiaVelocitàEventHandler(auto.CambiaVelocità);
```

in cui diciamo che al delegato (puntatore) CambiaVelocità presente nella form controllo associamo una nuova istanza del delegato comune istanziato in modo da puntare alla routine CambiaVelocità della form auto. Questa come si vede riceverà fra i parametri di ingresso (sender ed e) il puntatore alla form controllo (sender) ed il set di argomenti in questa valorizzati (e).

## Risultato Finale

Il Main sarà siffatto:



Dopo la pressione del bottone "Nuova Corsa" (button1) nel main si ottengono le seguenti due form interagenti:



Premendo "Fine Corsa" (button2) le due form vengono chiuse.

Nota conclusiva :

Vedendo tutto questo che non è così lapalissiano, almeno al sottoscritto, sono venuti due dubbi, ossia:

- Se sappiamo già quale è la void da lanciare nella madre è lo stesso obbligatorio usare i delegati?
- E se semplicemente nella madre definisco una void statica e la richiamo dal gestore degli eventi della secondaria?

La risposta è no per entrambi. Detto in altri termini un programma di questo genere:

```
Main  
.Public (static) void FaQualcosa()
```

```
Form1
.OnClick()
..Call Main.FaQualcosa()
```

Non potrà mai funzionare. Cerchiamo di capire il perché rifacendoci ai due quesiti di prima.

Per il primo il punto è che la Main (o meglio un'istanza della Main IstMain) genera un'istanza della Form1 "IstForm1" a cui passa il proprio puntatore nel campo sender. Quando qui succede qualcosa (es. evento OnClick) il gestore di questo evento ha sì un campo sender che però ora punta a IstForm1. In sintesi da questo gestore non si ha modo di rintracciare IstMain.

Per il secondo il fatto è che c'è bisogno di accedere all'istanza, ovvero proprio a quella finestra che viene visualizzata e non alla classe. Potrebbe in altri termini essere passato un parametro statico tipo "Aggiorna=True" ma l'istanza della Main non si aggiornerebbe perché può farlo solo in risposta ad un evento.

In sintesi per gestire interazioni fra form che implicino ad esempio effetti visivi è obbligatorio utilizzare i delegati ... o almeno io non ho trovato un metodo altrettanto efficace.