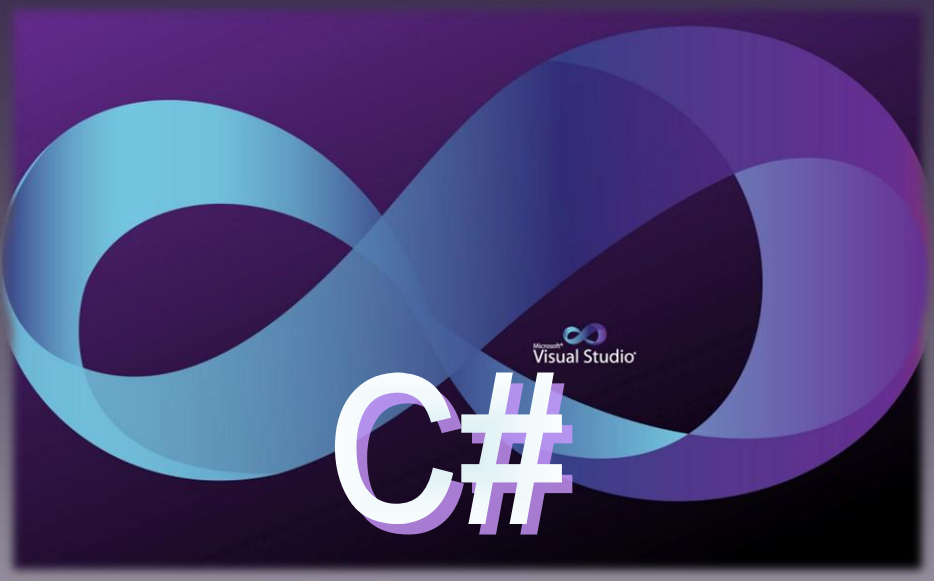
ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI



CORSO DI PROGRAMMAZIONE

METODI DELEGATI

DISPENSA 17.02

17-02_Delegati_[09]

Versione del: 07/11/2015
Revisione numero: 09

Prof. Andrea Zoccheddu
Dipartimento di Informatica

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

METODI DELEGATI

METODI DELEGATI

DELEGATI ED EVENTI IN C#

CENNI PRELIMINARI SUI DELEGATI

Programmando spesso ci si può trovare in situazioni in cui è necessario eseguire una particolare azione ma non si conosce in anticipo quali metodi o quali oggetti saranno coinvolti dalla stessa.

Ad esempio un controllo di un windows form potrebbe trovarsi nella condizione di notificare qualcosa ma potrebbe non conoscere a quale oggetto farlo. In questi casi, come vedremo, invece di collegare il controllo ad un oggetto specifico conviene collegarlo ad un delegato e lasciare a quest'ultimo la gestione dei metodi o degli oggetti coinvolti.

Agli albori della programmazione un programma iniziava la sua esecuzione e procedeva passo dopo passo fino alla sua conclusione con un coinvolgimento degli utenti limitato, nella maggior parte dei casi, alla compilazione di pochi campi.

Oggi i modelli di programmazione si basano su un approccio differente, denominato event-driven programming (programmazione guidata dagli eventi). In base a tale approccio i programmi moderni tipicamente interagiscono con gli utenti, tramite opportune interfacce, e rispondono alle loro azioni.

Gli utenti possono effettuare diverse azioni (ad esempio cliccare un pulsante o scrivere in una casella di testo) ed ognuna di esse genera un evento. Altri eventi possono invece essere generati indipendentemente dalle azioni degli utenti (ad esempio la scadenza di un timer).

Eventi e delegati sono concetti strettamente legati poiché in C# la gestione degli eventi viene implementata, come vedremo, tramite i delegati, ossia **oggetti utilizzati per incapsulare metodi con specifici parametri e tipo di ritorno**.

La creazione di un delegato in C# può essere effettuata tramite la parola chiave delegate seguita dal tipo restituito e dalla dichiarazione del metodo:

VC#

```
public delegate void DelegatoTest (string s)
```

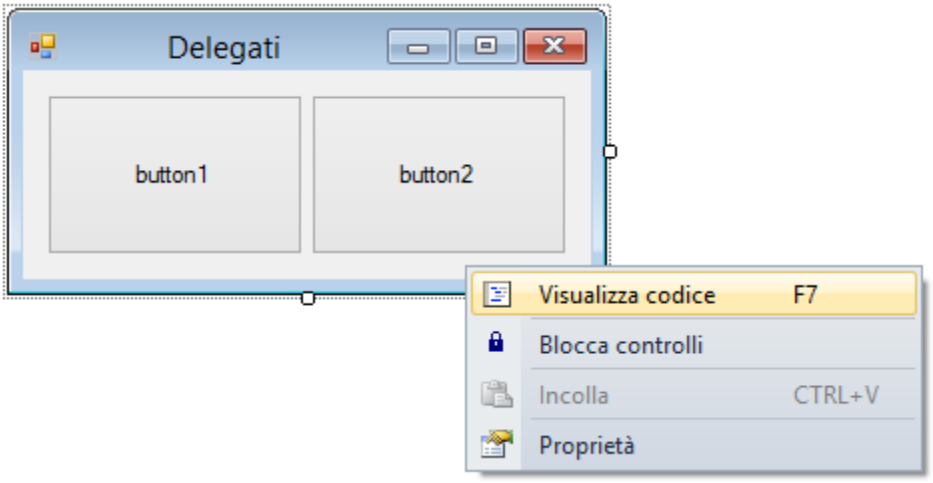
È da ricordare che l'incapsulamento è la proprietà per cui un oggetto contiene ('incapsula') al suo interno gli attributi (dati) e i metodi (procedure) che accedono ai dati stessi. Lo scopo principale dell'incapsulamento è dare accesso ai dati incapsulati solo attraverso metodi accessibili dall'esterno.

Dopo aver dichiarato il delegato DelegatoTest possiamo istanziarlo per incapsulare un metodo che abbia una firma (**signature**) corrispondente al delegato.

Vediamo un semplice esempio.

PROGETTO GUIDATO SU METODI DELEGATI

- Crea un nuovo progetto
- Disponi due pulsanti e prepara la finestra come nella figura



- Passa al codice e modificalo come nel seguente esempio:

VC#

```
namespace Delegati_Esempi
{
    public partial class Form1 : Form
    {
        static void InviaMessaggio(string s)
        {
            MessageBox.Show(s);
        }

        public delegate void DelegatoTest(string s);

        DelegatoTest dt = new DelegatoTest(InviaMessaggio);

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

- Dal form1 associa il gestore di evento button1_Click e scrivi il codice seguente:

VC#

```
        private void button1_Click(object sender, EventArgs e)
        {
            dt("pulsante button1");
        }
```

- Prova il progetto.

COMMENTO AL PROGETTO

Il progetto prevede l'uso di tre parti dichiarative e una parte esecutiva.

La prima delle parti da analizzare è la dichiarazione del metodo:

VC#
```
        static void InviaMessaggio(string s)
        {
            MessageBox.Show(s);
        }
```

Il metodo è un banale metodo statico del Form1: non rende nulla (**void**), ha un parametro di tipo stringa che conterrà il testo di un messaggio da mostrare, ma soprattutto è denominato **InviaMessaggio** che sarà il nome utilizzato nella istanziazione del delegato.

La seconda parte da analizzare è

VC#
```
        public delegate void DelegatoTest(string s);
```

che costituisce il vero nucleo della dichiarazione di un delegato. La riga di codice qui riportata è la dichiarazione del delegato: il descrittore public è opinabile (lo rende visibile anche fuori dalla unità di programma); la parola chiave **delegate** è un particolare tipo di dato che lo contraddistingue come un delegato, ovvero un metodo da associare a un corpo; il resto dei comandi specifica tipo nome e parametri del delegato, ma non presuppone alcun codice di corpo.

La terza parte da analizzare è

VC#
```
        DelegatoTest dt = new DelegatoTest(InviaMessaggio);
```

che costituisce la creazione (istanziazione) del metodo delegato. In questo caso si crea un oggetto (di tipo **DelegatoTest**) istanziato dal omonimo costruttore, il quale costruttore si aspetta come parametro il nome di un metodo (con corpo) da associare al nome del delegato.

I delegati dipendono soltanto dalla firma del metodo, non dalla classe o dall'oggetto che contiene il metodo. È, inoltre, opportuno osservare che nell'esempio precedente il metodo **InviaMessaggio** è stato dichiarato come statico: questo non è un requisito, ma se si usa dentro una classe Form1 è necessario.

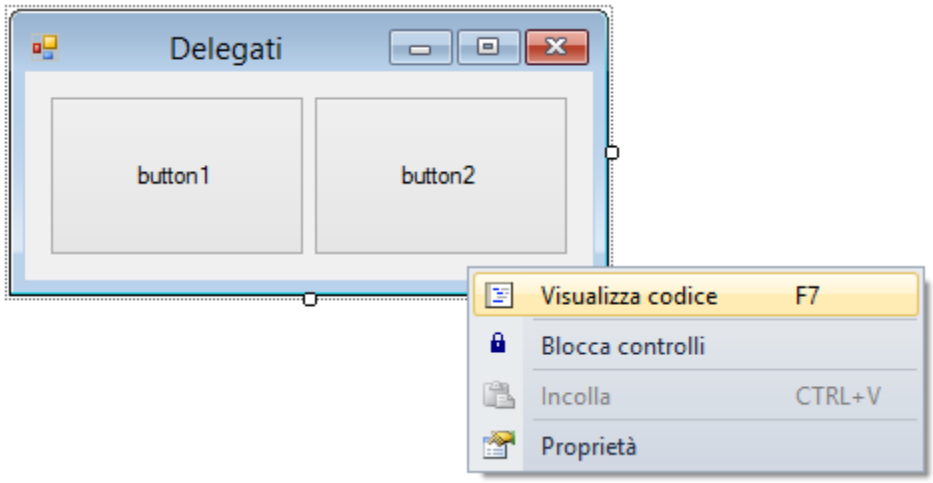
Infine al pulsante è stato associato l'invocazione del delegato, ovvero la sua "attivazione", il momento in cui il delegato è chiamato ad agire e a porre in azione l'effetto richiesto:

VC#
```
        {
            dt("pulsante button1");
        }
```

ove la parola **dt** è l'invocazione del delegato creato in precedenza.

PROGETTO GUIDATO

- Riapri il precedente progetto



- Modificalo il codice come segue:

VC#

```
namespace Delegati_Esempi
{
    public partial class Form1 : Form
    {
        static double CalcolaMedia(int a, int b)
        {
            return (a + b) / 2.0;
        }

        public delegate double DelegatoTest(int a, int b);

        DelegatoTest dt = new DelegatoTest(CalcolaMedia);

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

- Il gestore di evento button1_Click deve essere modificato come segue:

VC#

```
        private void button1_Click(object sender, EventArgs e)
        {
            double ris = dt(13, 17);
            MessageBox.Show("" + ris);
        }
```

- Prova il progetto.

COSA È UN DELEGATO?

DEFINIZIONE DI DELEGATO

LINK

Fonte: http://msdn.microsoft.com/it-it/library/ms173172.aspx

Un delegato è un tipo in cui è incapsulato un metodo in modo sicuro.

È simile a un puntatore a una funzione in C e C++ ma, diversamente da questo, è orientato ad oggetti, indipendente dai tipi e protetto. Il tipo di un delegato è definito dal nome del delegato

Come prima definizione, quindi, un delegato è un ulteriore tipo di dato. In realtà un delegato è un nuovo tipo di dato che il programmatore può costruire e associare a un procedimento.

La dichiarazione di un delegato segue la seguente sintassi:

VC#

```
descrittori delegate TipoReso NomeDelegato(elenco_parametri);
```

I descrittori del delegato sono quelli ordinari visti per altri tipi, come public, protected, private, static, ecc… in generale in questa dispensa vedremo il descrittore public come il più frequente.

Il frammento racchiuso nella cornice rossa indica la descrizione della firma del delegato, ovvero l'insieme del tipo restituito, il suo nome e l'elenco dei parametri; in definitiva la firma del delegato è analoga alla firma di un qualsiasi metodo già studiato in precedenza.

La parola chiave delegate che separa i descrittori dalla firma, è la keyword che dichiara un nuovo delegato, ovvero il definitore del nuovo tipo (analogo per esempio a class).

Un esempio di dichiarazione di delegato è la seguente:

VC#

```
        public delegate double DelegatoTest(int a, int b);
```

dove la parola chiave delegate indica che si tratta di una dichiarazione di un delegato, l'identificatore DelegatoTest è il nome del delegato costruito (è il nuovo tipo appena dichiarato), e con le parentesi tonde si descrive la firma del metodo atteso.

Vediamo adesso come costruire il delegato appena dichiarato:

LINK

Fonte: http://msdn.microsoft.com/it-it/library/ms173172.aspx

Per la costruzione di un oggetto delegato è in genere sufficiente fornire il nome del metodo di cui il delegato eseguirà il wrapping (**confezionamento**) oppure utilizzare un metodo anonimo.

Vediamo un esempio per preparare un metodo cui il delegato eseguirà il confezionamento:

VC#
```
static double CalcolaMedia(int a, int b)
{
    return (a + b) / 2.0;
}
```

E il confezionamento avverrà quando si eseguirà un'istruzione analoga alla seguente:

VC#
```
DelegatoTest dt = new DelegatoTest(CalcolaMedia);
```

Il wrapping può essere ottenuto anche indicando il solo identificatore di un metodo associabile, come nel seguente esempio:

VC#
```
DelegatoTest dt = CalcolaMedia;
```

L'alternativa a questo confezionamento è di utilizzare un metodo anonimo (senza nome) come nel seguente esempio:

VC#
```
DelegatoTest myDel = delegate (int x, int y) { return ((x + y)/2.0); };
```

Nel precedente esempio il delegato confezionato è anonimo, infatti è costruito al volo con la parola chiave delegate seguito dai parametri e dal corpo.

Occorre osservare che il delegato non ammette un metodo (da confezionamento o anonimo) la cui firma non corrisponda a quella della dichiarazione; rispetto al precedente esempio darebbe errore il seguente codice:

Errore
```
DelegatoTest myDel = delegate (string s) { return (s + "errore!"); };
```

Una volta creata un'istanza di un delegato, quest'ultimo passerà al metodo le chiamate ricevute. I parametri passati al delegato dal chiamante verranno passati al metodo e l'eventuale valore restituito dal metodo verrà restituito dal delegato al chiamante.

Vediamo un altro esempio con cui confezionare un delegato, invocarlo e usare il risultato:

VC#
```
DelegatoTest prova;          //dichiarazione di un nuovo oggetto delegato
prova = CalcolaMedia;        //wrapping dell'oggetto delegato
double ris = prova(12, 21);  //invocazione dell'oggetto delegato
MessageBox.Show("" + ris);   //
```

Per fare riferimento a questo processo si afferma normalmente che il delegato viene richiamato. È possibile richiamare un delegato di cui è stata creata un'istanza in modo analogo al metodo di cui è stato eseguito il **wrapping** dal delegato stesso.

DELEGATO COME TIPO

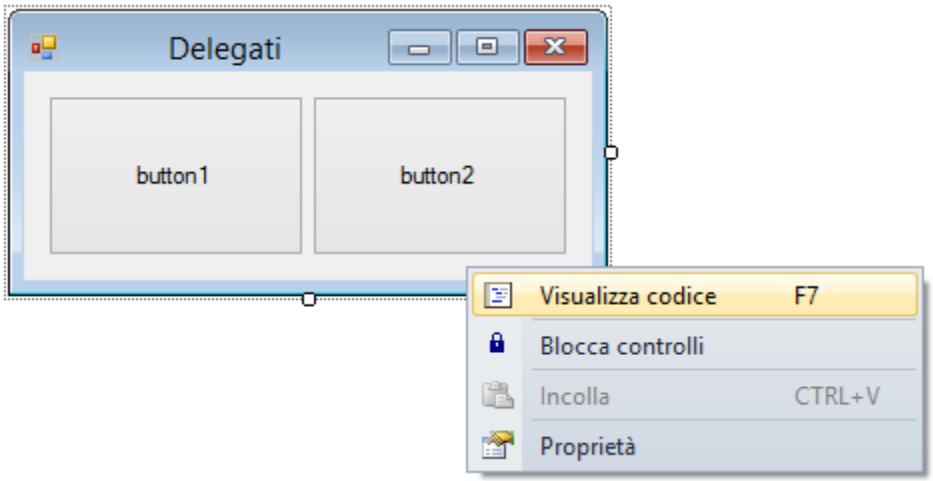
DELEGATO COME TIPO (CLASSE)

I tipi delegati vengono derivati dalla classe Delegate in .NET Framework e sono sealed (classi sigillate), ossia non possono essere utilizzati per dichiarare classi derivate da esso. Non è inoltre possibile derivare classi personalizzate da Delegate.

Poiché il delegato di cui è stata creata un'istanza è un oggetto, può essere passato come parametro o assegnato a una proprietà. **In questo modo un metodo può accettare un delegato come parametro e chiamarlo in un secondo momento.** Questa operazione è nota come **callback asincrono** e rappresenta uno dei modi più comuni per notificare a un chiamante il completamento di un processo prolungato. Quando si utilizza un delegato in questo modo, nel codice che include il delegato non deve essere necessariamente specificata l'implementazione del metodo. La funzionalità è simile all'incapsulamento fornito dalle interfacce.

PROGETTO GUIDATO

- Riapri il precedente progetto (oppure si crei uno nuovo, con la dichiarazione del delegato **DelegatoTest** :



- Aggiungi al codice la seguente dichiarazione di metodo:

VC#

```
public void MetodoConUnCallback(int param1, int param2, DelegatoTest param3)
{
      double ris;
      ris = param3(param1, param2);
      MessageBox.Show("Il risultato è: " + ris);
}
```

- Adesso associa al secondo pulsante button2 il seguente gestore di evento:

VC#

```
private void button2_Click(object sender, EventArgs e)
{
     MetodoConUnCallback(12, 13, CalcolaMedia);
}
```

- Prova il progetto ed usa il button2: funziona?
- È possibile associare al button1 un callback che invoca un metodo diverso da CalcolaMedia, per esempio un metodo che calcola il massimo, la media pesata, o la distanza dall'origine del punto le cui coordinate sono passate coi parametri?

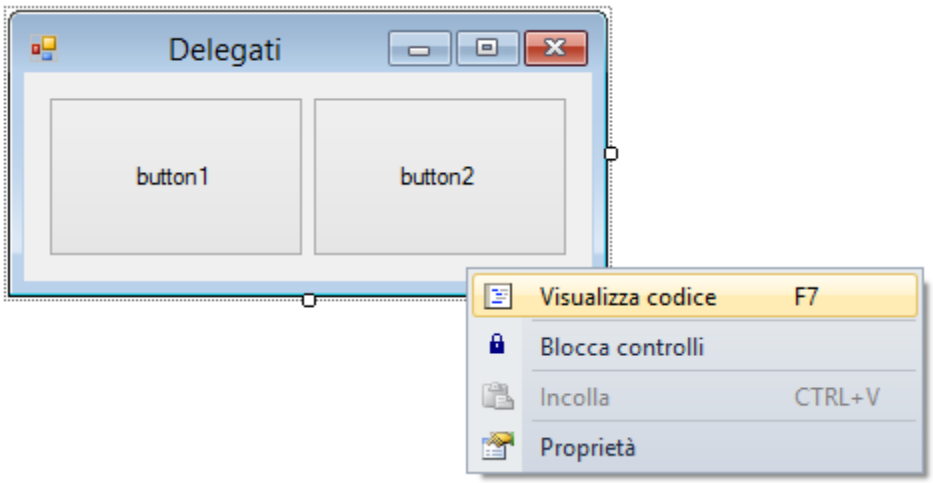
COLLEZIONE DI METODI DELEGATI

LISTA DI METODI DELEGATI

I delegati possono contenere una lista interna di delegati (chiamata **invocation list**) che contiene delegati aventi la stessa firma, si parla in questi casi di **delegati multicast**. Quando un delegato di questo tipo viene invocato esso a sua volta invoca tutti i delegati presenti nella sua invocation list. E' possibile aggiungere e rimuovere delegati da una invocation list utilizzando gli operatori di overload **+=** e **-=** rispettivamente.

PROGETTO GUIDATO

- Crea un progetto nuovo:



- Aggiungi al codice la seguente dichiarazione di metodo:

VC#
```
//-------------------------------------------------------------secondo esempio
public delegate double MioDelegato (double a, double b);

public double MediaDelegata (double a, double b)
           { MessageBox.Show("1");  return ((a + b) / 2.0); }

public double MassimoDelegato (double a, double b)
           { MessageBox.Show("2"); if (a > b) return a; else return b; }
public double DistanzaDelegata (double a, double b)
           { MessageBox.Show("3"); return Math.Sqrt(a * a + b * b); }

MioDelegato deleg_1;
```

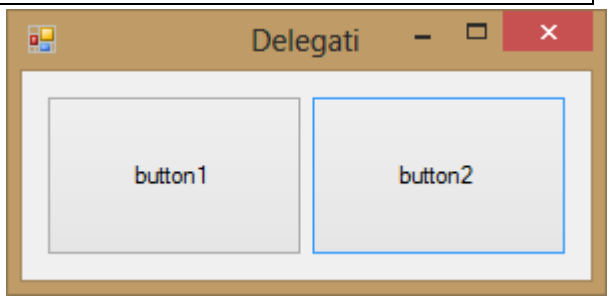
- Associa al pulsante **button1** il seguente codice:

VC#
```
//-------------------------------------------------------------secondo esempio
private void button1_Click(object sender, EventArgs e)
{
     deleg_1 += MediaDelegata;

     deleg_1 += MassimoDelegato;

     deleg_1 += DistanzaDelegata;
}
```

- Associa al pulsante **button2** il seguente codice:

VC#
```
//-------------------------------------------------------------secondo esempio
private void button2_Click(object sender, EventArgs e)
{
     double ris = deleg_1(12, 21);       //invocazione dell'oggetto delegato
     MessageBox.Show("Ma il risultato finale è: " + ris);

}
```

- Avvia il progetto e premi prima button1 e solo dopo button2
- Dopo premi nuovamente prima button1 e solo dopo button2
- Ripeti la precedente indicazione alcune volte
- Termina il progetto



COMMENTO AL PROGETTO

Abbiamo già affermato che i **delegati multicast** possono contenere una lista interna di delegati (chiamata **invocation list**) che contiene delegati tutti aventi la stessa firma.

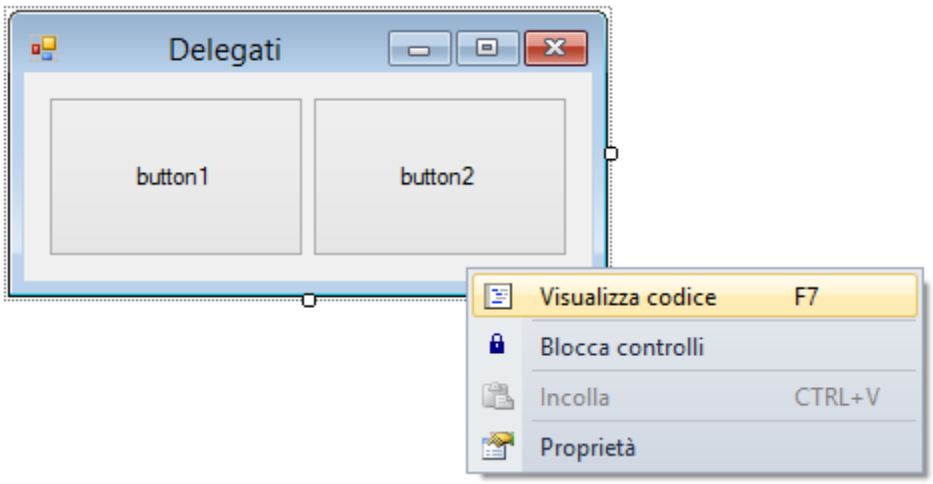
Questo significa che lo stesso delegato fa riferimento non ad un solo metodo ma ad un elenco; nell'esempio appena concluso il delegato deleg_1 fa riferimento inizialmente a nessun metodo; dopo il clic sul button1 ne vengono associati tre, nell'ordine di assegnazione; quando si preme ancora il button1 ne vengono associati altri tre, sempre nell'ordine di assegnazione. E così via.

Quando il delegato viene invocato, esso chiama l'invocazione di tutti i metodi in elenco. Se ci sono tre metodi, allora ne invoca tre. Se ci sono sei metodi, allora ne invoca sei. Se ci sono nove metodi, allora ne invoca nove. E così via.

Come è possibile aggiungere metodi al delegato è anche possibile rimuoverlo; per eliminare un metodo dalla lista occorre utilizzare l'operatore **-=**; vediamo un esempio:

PROGETTO GUIDATO

- Modifica il progetto precedente:



- Modifica il corpo del gestore di evento del button1 come segue:

VC#

```
        private void button3_Click(object sender, EventArgs e)
        {
            deleg_1 += MediaDelegata;       //1
            deleg_1 += MassimoDelegato;     //2
            deleg_1 += DistanzaDelegata;    //3
            deleg_1 += MediaDelegata;       //1
            deleg_1 += MassimoDelegato;     //2
            deleg_1 += DistanzaDelegata;    //3
            deleg_1 -= MassimoDelegato;     // elimino un 2
           //invocazione dell'oggetto delegato
            double ris = deleg_1(12, 21);
        }
```

- Avvia il progetto e premi button1 alcune volte
- Termina il progetto

COMMENTO AL PROGETTO

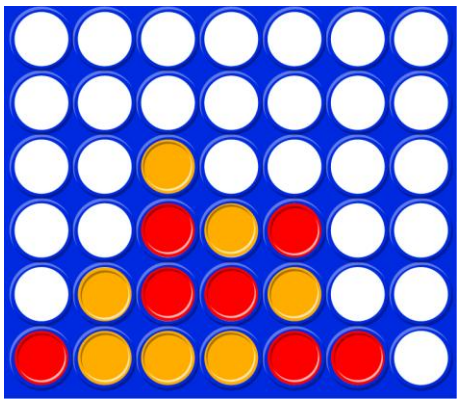
Abbiamo visto che un delegato fa riferimento non ad un solo metodo ma ad un elenco; nell'esempio appena concluso il delegato deleg_1 accoda 6 metodi ma poi ne rimuove uno; il metodo rimosso è l'ultimo dell'elenco con il nome specificato.

Quando il delegato viene invocato, esso richiama l'invocazione di tutti i metodi in elenco. La prima volta quindi invocherà 5 metodi (6 meno 1), la seconda invocherà 10 metodi (12 meno 2), alla terza invocherà 15 metodi (18 meno 3), e così via!

ESERCIZI

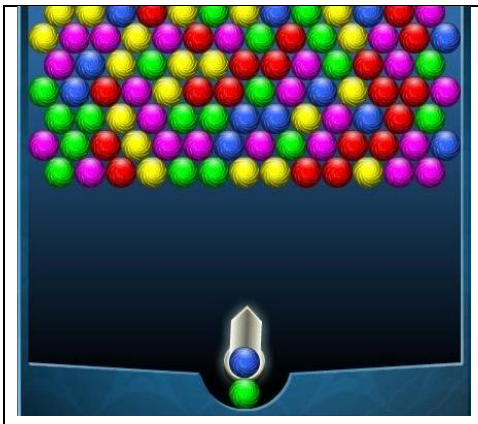
ESERCIZI

ESERCIZIO 1.



- Crea un progetto visuale
- All'avvio del programma si crea «al volo» una matrice di immagini come nella figura qui a lato (6 x 7)
- Definire un delegato che accetta due diversi metodi:
  1. Colora di rosso l'immagine cliccata
  2. Colora di giallo l'immagine cliccata
  3. Ad ogni clic il delegato rimuove il metodo associato e si aggancia all'altro

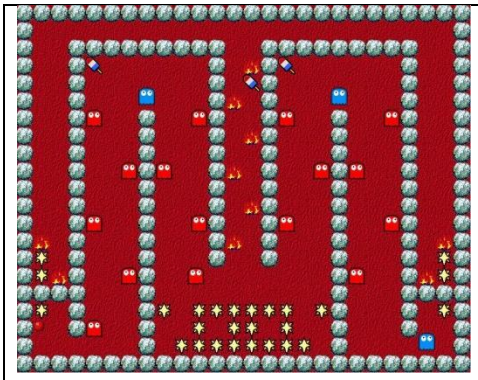
ESERCIZIO 2.



- Crea un progetto visuale
- All'avvio del programma si crea «al volo» una collezione di immagini come nella figura qui a lato

1. ?

ESERCIZIO 3.



- Crea un progetto visuale
- All'avvio del programma si crea «al volo» una collezione di immagini come nella figura qui a lato

?

SOMMARIO