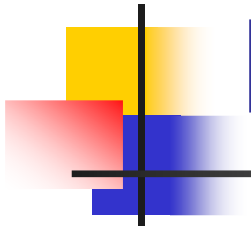
Tecniche di Programmazione avanzata

Corso di Laurea Specialistica in Ingegneria Telematica
Università Kore – Enna – A.A. 2009-2010

Alessandro Longheu
http://www.diit.unict.it/users/alongheu
alessandro.longheu@diit.unict.it

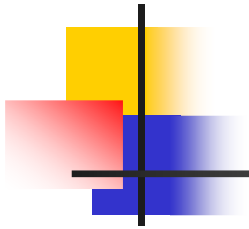
Il linguaggio C#
Eventi ed eccezioni



Eventi

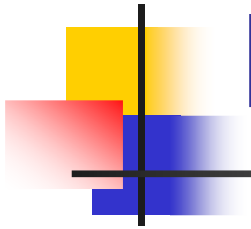
- C# offre un supporto nativo al concetto di EVENTO: un evento è qualcosa che un oggetto vuole notificare a qualche altro oggetto al verificarsi di una data attività

- **Event sender (autore)** l'oggetto che scatena (raise, trigger) l'evento, detto anche la sorgente dell'evento.
- **Event receiver (sottoscrittore)** l'oggetto per il quale l'evento è determinante e che quindi desidera essere avvisato quando l'evento si verifica.
- **Event handler** – il metodo (dell'event receiver) che viene eseguito all'atto della notifica.



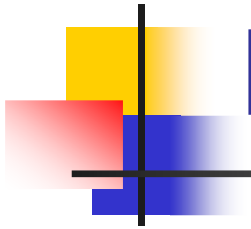
Eventi

- L'autore determina quando un evento viene generato; i sottoscrittori determinano quale azione viene eseguita in risposta all'evento.
- Un evento può avere più sottoscrittori. Uno sottoscrittore può gestire più eventi di più autori.
- Gli eventi senza sottoscrittore non sono mai chiamati.
- Gli eventi vengono comunemente utilizzati per segnalare azioni utente quali clic sui pulsanti o selezioni di opzioni di menu nell'interfaccia grafica.
- Quando un evento ha più sottoscrittori, i gestori eventi vengono richiamati in modo sincrono quando viene generato un evento. E' possibile anche richiamare gli eventi in modo asincrono.
- Gli eventi possono essere utilizzati per sincronizzare i thread.
- Nella libreria di classi di .NET Framework, gli eventi sono basati sul delegato EventHandler e sulla classe base EventArgs.



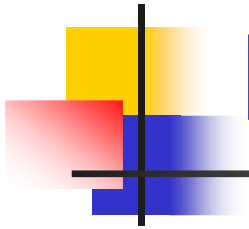
Eventi

- Quando si verifica l'evento, il sender "notifica" in qualche modo tutti i receiver
  - per farlo, invoca gli event handler dei vari receiver
  - in genere, il sender non conosce né i receiver, né gli handler
- Per collegare sender e receiver/handler si sfrutta il meccanismo dei delegati
  - per questo motivo, in C# un evento incapsula un delegato
  - è quindi necessario dichiarare un tipo di delegato prima di poter dichiarare un evento.
  - Gli eventi quindi utilizzano i delegati per fornire l'incapsulamento indipendente dai tipi dei metodi che verranno chiamati quando vengono generati



Eventi

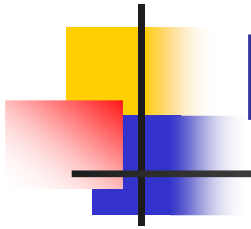
- Un evento consente a una classe di notificare agli oggetti la necessità di eseguire un'azione
- Gli eventi vengono in genere utilizzati nelle interfacce utente grafiche
- Gli eventi vengono, in genere, dichiarati tramite i gestori eventi dei delegati
- Gli eventi possono chiamare metodi anonimi in sostituzione dei delegati



Dichiarazione

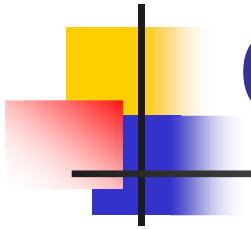
- Gli eventi dispongono di una firma che comprende un nome e un elenco di parametri, definita da un tipo di delegato
  - il primo parametro è un oggetto che fa riferimento all'origine dell'evento
  - il secondo è una classe che contiene i dati relativi all'evento.
  - Comunque la firma di un evento può essere uguale a qualsiasi firma di delegato valida, purché restituisca void.

```
public delegate void EventDelegate
(object sender, System.EventArgs e);
```



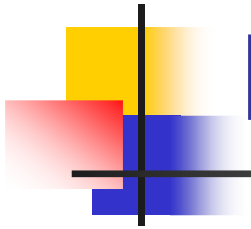
Dichiarazione

- Per aggiungere un evento a una classe, è necessario utilizzare la parola chiave event e fornire un tipo di delegato e un nome per l'evento
  - Gli eventi possono essere contrassegnati come public, private, protected, internal o protected internal
  - Un evento può essere dichiarato statico mediante la parola chiave static.
  - Un evento può essere contrassegnato come virtuale



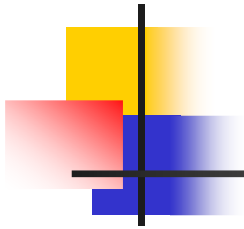
Generazione di eventi

- Per generare un evento, la classe può chiamare il delegato, passando gli eventuali parametri correlati all'evento
- Gli eventi per i quali non sono disponibili gestori sono null
- le origini devono inoltre creare una copia dell'evento prima di eseguire tale controllo e generare l'evento



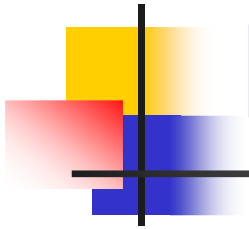
Registrazione

- creare un metodo per la ricezione di tale evento e quindi aggiungere un delegato per tale metodo all'evento stesso della classe.
  - È innanzitutto necessario che la classe ricevente contenga un metodo con la stessa firma dell'evento, ad esempio la firma del delegato.
  - Questo metodo, denominato gestore eventi, può quindi eseguire l'azione appropriata in risposta all'evento.



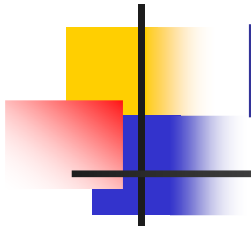
Eventi in C#: Sintassi

- **Sintassi**:
  [public] event TipoDiDelegato NomeEvento ;
- Esempio:
  public event EventHandler Changed;
- dove
- Changed è il nome del nuovo evento, che incapsula e controlla un delegato di tipo EventHandler.
- In pratica, Changed è un delegato, ma la keyword event ne limita la visibilità e le possibilità di utilizzo a quelle consone al concetto di evento.



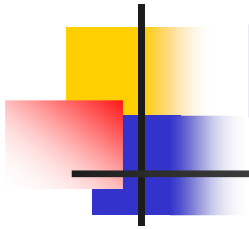
Eventi in C#: Motivazioni

- per modellare gli eventi basterebbero i delegati
- MA così facendo chiunque potrebbe chiamare il delegato, ossia "scatenare" l'evento.
- Poiché ciò è inopportuno, C# introduce una specifica nozione di EVENTO
  - un evento incapsula un delegato per disciplinarne l'uso
  - un evento non può essere scatenato da chiunque perché l'accesso al delegato interno è protetto
  - si introducono appositi operatori per aggiungere / togliere event receivers dalla "lista di notifica" dell'evento.



Eventi: Modello d'uso

- A seguito della definizione:
  public event TipoDiDelegato nomeEvento ;
- quando si verifica l'evento nomeEvento, viene chiamato il delegato TipoDiDelegato.
- Una volta dichiarato, l'evento può essere trattato come un delegato di tipo speciale:
  - può essere null se nessun event receiver si è registrato
  - può essere associato a uno o più metodi da invocare.
- Limitazione rispetto ai delegati:
  - benché l'evento sia public, l'invocazione dell'evento può avvenire solo all'interno della classe nella quale l'evento è stato dichiarato.



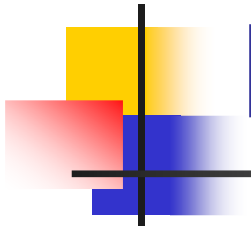
Eventi: Modello d'uso

- Per questo motivo, per **scatenare un evento** è bene:
  - definire un metodo protetto (virtuale) – l'unico che chiama direttamente l'evento
  - invocare sempre quello (mai l'evento direttamente)
- Esempio:
- definizione del metodo protetto:

```
protected virtual void OnChanged(EventArgs e){
    if (Changed != null) Changed(this, e);
}
```

- invocazione di questo da parte degli event senders

OnChanged(EventArgs.Empty); o OnChanged(...);

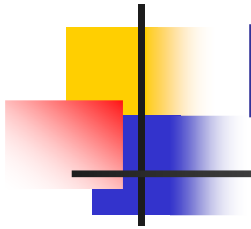


Eventi: Modello d'uso

- Per **agganciarsi a un evento** (cioè iniziare a ricevere le notifiche di un evento), un event receiver deve:
- definire il metodo (event handler) che verrà invocato all'atto della notifica dell'evento:
  void ListChanged(object sender, EventArgs e){...}

- creare un delegato dello stesso tipo dell'evento e farlo puntare al metodo event handler:
  EventHandler ListChangedHandler =new EventHandler(ListChanged);

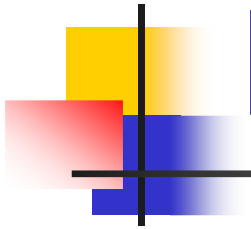
- aggiungere questo delegato alla lista dei delegati associati all'evento, utilizzando l'operatore +=:
  List.Changed += ListChangedHandler ;



Eventi: Modello d'uso

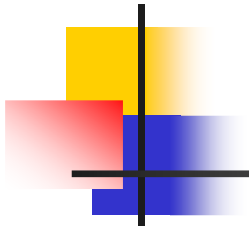
- Per **sganciarsi da un evento** (cioè smettere di ricevere le notifiche di un evento), un event receiver deve togliere il delegato dalla lista dei delegati associati all'evento, utilizzando l'operatore -=, esempio:
  List.Changed -= ListChangedHandler ;

- poiché += e -= sono le uniche operazioni permesse su un evento fuori dalla classe dove esso è definito, dall'esterno è possibile solo aggiungere e rimuovere handler per l'evento, ma NON RECUPERARE NÉ MODIFICARE la lista degli event handlers (protezione).



Delegati Per Eventi

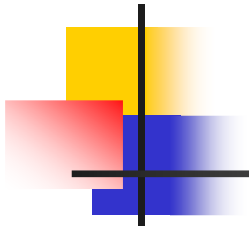
- In generale, i delegati alla gestione degli eventi hanno due parametri: la sorgente dell'evento e i dati dell'evento.
- La classe predefinita System.EventHandler è adatta a tutte quelle situazioni particolari in cui l'evento non genera dati, ossia in cui basta l'informazione "evento avvenuto" e la relativa sorgente. Di fatto, System.EventHandler è un delegato
- La classe System.EventArgs viene usata quando un evento non deve passare informazioni aggiuntive ai propri gestori. Se i gestori dell'evento hanno bisogno di informazioni aggiuntive, è necessario derivare una propria classe da EventArgs e aggiungere i dati necessari.



Eventi: Esempio Globale

- Definire la classe dell'event source
  - definisce l'evento e il relativo metodo OnEvent che lo scatena
  - definisce il metodo pubblico che esprime il "normale job" della sorgente, durante il quale, prima o poi, nascono eventi.

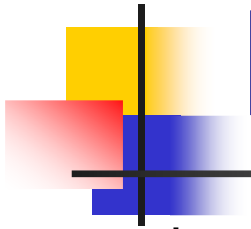
```
class MyEventSource {
    public event EventHandler evt;
    // evento è pubblico, ma non è scatenabile da fuori
    protected virtual void OnEvt(EventArgs args){
     if (evt!=null) evt(this,args); // attiva l'evento
    }
    public void DoJob(){ // il normal job della sorgente
          OnEvt(EventArgs.Empty); //la sorgente prima o poi genera
    eventi
    }
} }
```



Eventi: Esempio Globale

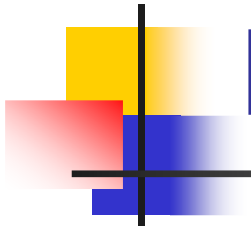
- Definire la classe dell'applicazione
  - definisce il metodo event handler che gestisce l'evento
  - crea la sorgente e associa l'event handler all'evento (pubblico)
  - attiva il normale job della sorgente (se necessario)

```
class Application {
    static void Main(string[] args){
     MyEventSource src = new MyEventSource();
     src.evt += new EventHandler(pippo);
     src.DoJob();
    }
    private static void pippo(Object sender, EventArgs x){
     Console.WriteLine("gestisco l'evento " + x +
      " generato da " + sender);
    }
}
```



Esempi

- Vedere esempi allegati
- Altri esempi:
  - http://msdn.microsoft.com/en-us/library/5z57dxz2.aspx
  - http://msdn.microsoft.com/en-us/library/9aackb16.aspx
- Esercitazione:
  - Realizzare un programma che permetta l'inserimento di una serie di valori da tastiera, segnalando quando il valore medio corrente scende al di sotto di una certa soglia (utilizzare gli eventi)
  - Implementare il modello del produttore-consumatore tramite eventi
  - Un'applicazione industriale deve monitorare la sicurezza di un impianto. Sono in tal senso presenti un sensore di temperatura ed uno di presenza esalazioni tossiche, che inviano opportuni eventi qualora vengono superati valori prefissati di temperatura o di concentrazione in ppm, rispettivamente (generare casualmente il valore istantaneo della grandezza). Un unico sistema di monitoraggio deve produrre messaggi di avviso al personale, indicando anche quante volte in totale sono scattati ciascuno dei due allarmi



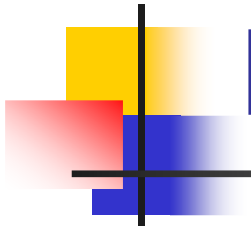
Eccezioni in C#

- Le eccezioni in C# funzionano come in Java
  - stessi costrutti try, catch, finally, throw
  - stessa gerarchia delle eccezioni fondamentali
- Minime varianti
  - è possibile evitare di specificare il nome dell'oggetto eccezione in un catch se non lo si usa
- catch (IndexOutOfBoundException) { … }
  - è possibile usare una clausola catch senza parametri per catturare tutte le eccezioni di qualunque tipo
- catch { … }
  - non esiste più la clausola throws per dichiarare che un metodo può generare una eccezione
- Un nuovo concetto: istruzioni checked e unchecked per controllare o ignorare un eventuale overflow nei calcoli di una espressione o un blocco di istruzioni

Eccezioni in C#

```
FileStream s = null;
try {
    s = new FileStream(curName, FileMode.Open);
    ...
} catch (FileNotFoundException e) {
    Console.WriteLine("file {0} not found", e.FileName);
} catch (IOException) {
    Console.WriteLine("some IO exception occurred");
} catch {
    Console.WriteLine("some unknown error occurred");
} finally {
    if (s != null) s.Close();
}
```

- finally è sempre eseguita
- Il parametro Exception può essere omesso.In questo caso si assume System.Exception.
- Exception derivato da System.Exception.
- throw lancia l'eccezione



Eccezioni in C#

Proprietà

e.Message stringa;
inizializzato da *new Exception(msg);*

e.StackTrace stringa

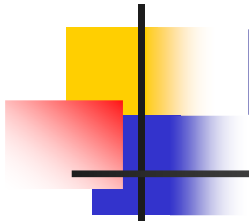
e.Source sorgente dell'eccezione

e.TargetSite nome del metodo che invia l'eccezione

Metodi

e.ToString() nome dell'eccezione e traccia dello stack

...



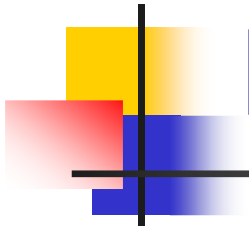
Eccezioni in C#

Cause di sollevamento di un'eccezione:

- Operazione invalida (implicita):
  Divisione per 0, overflow degli indici, uso di un riferimento null…

- usando throw (esplicita):
  throw new FunnyException(10);
  class FunnyException : ApplicationException {
  public int errorCode;
  public FunnyException(int x) { errorCode = x; }
  }

- usando un metodo che può emettere un'eccezione, ma non la cattura:
  s = new FileStream(...);



Eccezioni in C#

Firma dei metodi in caso di eccezioni:

- Java

  void myMethod() throws IOException {
  ... throw new IOException(); ...
  }

- Il metodo invocante o
  - cattura l'eccezione IOException o
  - specifica IOException nella propria firma

- C#

  void myMethod() {
  ... throw new IOException(); ...
  }

- il metodo invocante può non gestire l'eccezione.
  - + semplice
  - - meno robusto