C# Applicazioni Windows

Sommario

PREMESSA

Il testo ha lo scopo di introdurre alla realizzazione di programmi dotati di interfaccia grafica, definiti «Applicazioni Windows» poiché usufruiscono pienamente delle funzionalità avanzate – grafiche e non – del sistema operativo Windows.

Programmi simili presentano una struttura e presuppongono un modello di interazione con l'utente completamente diversi rispetto alle «Applicazioni Console», e cioè i tipi di programmi presi in considerazione finora.

Nel primo capitolo, il testo introduce il termine «interfaccia utente», definendo successivamente le caratteristiche che differenziano un tipo di interfaccia da un altro. Quindi presenta due tipici modelli di interfaccia utente, gli unici presi in considerazione in tutto il corso di studi (e, occorre sottolineare, attualmente gli unici utilizzati nella programmazione in generale). Infine, esamina il modello di interfaccia usato dalle «Applicazioni Console», in modo da poterlo usare come termine di paragone nella successiva analisi del modello grafico impiegato dalle «Applicazioni Windows».

Tale analisi comincia nel secondo capitolo. Vengono prima introdotti gli aspetti che caratterizzano una «Applicazione Windows», sia nella gestione del video che nella comunicazione con l'utente, nonché i costituenti fondamentali di un'interfaccia grafica: i «controlli». Quindi, prendendo come riferimento un semplice programma dotato di interfaccia grafica, viene affrontato il compito di costruirne uno che produca lo stesso funzionamento.

Nel terzo capitolo vengono presentati gli elementi di base necessari per la realizzazione di una interfaccia grafica, e cioè l'introduzione alla classe `Control` e la presentazione di quei controlli che fanno parte di in qualsiasi interfaccia, per quanto minimale: `TextBox`, `Label`, `Button`, e `Form`. Nel quarto capitolo vengono presi in considerazione quei controlli, come `ListBox`, `ComboBox`, `RadioButton`, eccetera, che consentono di arricchire il livello dell'interazione con l'utente.

Infine, nel quinto capitolo viene presentata un esempio di realizzazione di un'interfaccia grafica completa. In questo contesto viene inoltre affrontato il problema di garantire la validità delle azioni e dei valori inseriti dall'utente.

Il testo è a carattere introduttivo e dunque viene esaminato soltanto un piccolo sotto insieme dei controlli disponibili, e per ognuno di essi sono presi in considerazione soltanto le proprietà e gli eventi più comuni.

Capitolo 1 – Interfaccia utente: concetti di base

Caratteristiche di un «interfaccia utente»

- Gestione del video
- Modello di comunicazione con l'utente

Modelli di «interfacce utente»

- «A caratteri/input-output»
- «Grafica/guidata dagli eventi»

Interfaccia utente delle «Applicazioni Console»

1.1 «INTERFACCIA UTENTE»

Con il termine «interfaccia utente» viene designata quella parte del programma che gestisce la comunicazione con l'utente, consentendo a questo di immettere le informazioni da elaborare e di ricevere il risultato di tale elaborazione. In maggior dettaglio, il termine interfaccia denota due significati distinti, anche se collegati, in base alla prospettiva dalla quale lo si considera:

dal punto di vista dell'utente, l'interfaccia è rappresentata dalla modalità di rappresentazione su video dei dati e dalla modalità di acquisizione degli stessi;

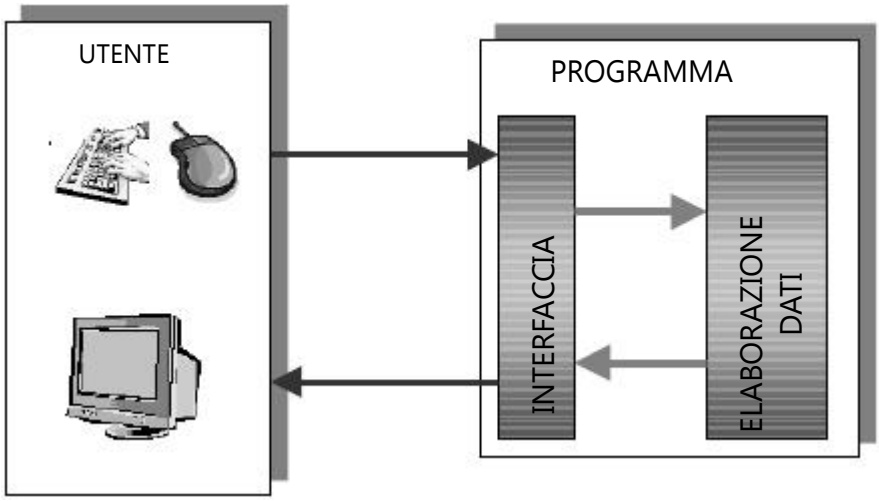
dal punto di vista del programmatore, l'interfaccia è rappresentata della parte di codice che svolge la funzione di ricevere le azioni dell'utente e di rispondere ad esse, visualizzando su video.

Detto questo, ogni programma può dunque intendersi logicamente suddiviso in due parti:

a) la parte «interfaccia», che svolge la funzione di comunicare con l'utente, e dunque di ricevere i dati da elaborare e di comunicare i risultati della elaborazione;

b) la parte «elaborazione» (business logic), con la quale si identifica il codice demandato alla vera e propria elaborazione dei dati.

Figura 1-1. Rappresentazione schematica generale di un programma.



Va sottolineato che una simile suddivisione dev'essere intesa in senso concettuale; in altre parole, perlomeno in programmi realistici, non esiste un criterio preciso per stabilire quali istruzioni appartengano alla parte dedicata all'interfaccia e quali alla parte elaborazione, e nemmeno è utile una cosa del genere. L'importante è comprendere che ogni programma deve soddisfare due tipi di compiti logicamente distinti:

1) ricevere e rispondere alle azioni dell'utente: e ciò è carico della parte chiamata «interfaccia»;

2) elaborare i dati: e ciò è a carico della parte chiamata «elaborazione».

1.1.1 Caratteristiche di una interfaccia utente

Sono dunque due gli aspetti che caratterizzano l'interfaccia di un programma:

1) le gestione del video, e dunque la forma con la quale le informazioni vengono visualizzate;

2) il modello di comunicazione con l'utente, e dunque il modo in cui il programma riceve le azioni – nella fattispecie i dati – dall'utente.

Per ognuno di questi aspetti esistono fondamentalmente due possibilità.

Gestione del video

La gestione del video, e cioè l'apparenza dell'interfaccia, può essere di tipo:

a) «a caratteri», o «testuale»;

b) «grafico».

Nel primo caso, il video viene trattato come una matrice di caratteri, il cui numero di righe e di colonne dipende da alcune impostazioni del sistema operativo. Il singolo elemento visualizzabile è il carattere, e dunque l'interfaccia può rappresentare solo informazioni testuali; sono escluse forme geometriche, bordi, immagini, ombreggiature, animazioni, eccetera[1].

Nel secondo caso, il video viene visto come una matrice di punti, chiamati «pixel», idealmente molto simile a un grafico cartesiano, dotato degli assi delle ascisse e delle ordinate. Poiché il numero di pixel è molto superiore a quello dei caratteri disponibili in un interfaccia a caratteri (minimo, 480 pixel in ordinata e 640 in ascissa), un'interfaccia grafica consente la visualizzazione di elementi grafici, ombreggiature, animazioni, eccetera.

Modello di comunicazione con l'utente

La comunicazione con l'utente può essere di tipo:

a) «ingresso/uscita», o «sequenziale»

b) «guidata dagli eventi» (event-driven).

La forma di comunicazione «ingresso/uscita» presuppone innanzitutto l'esistenza di metodi specializzati per l'acquisizione (ingresso) e per la visualizzazione (uscita) dei dati. Una seconda caratteristica è data dal fatto che esiste una chiara corrispondenza – una sequenzialità – tra l'ordine di esecuzione di tali metodi e l'effetto che essi producono dal punto di vista dell'utente. In altre parole, il comportamento dell'interfaccia dal punto di vista dell'utente – la sequenza di visualizzazione e di richiesta dei dati – è rigidamente codificato nelle istruzioni che implementano l'interfaccia stessa.

Infine, un modello di comunicazione «ingresso/uscita» non è in grado di gestire il mouse.

---

[1] In realtà, mediante l'uso di particolari caratteri e sofisticati metodi di visualizzazione è possibile simulare una "quasi apparenza" grafica anche per le interfacce a caratteri. Ciò era fatto in passato, prima della definitiva affermazione delle funzionalità grafiche offerte dagli attuali sistemi operativi.

Il modello di comunicazione «guidato dagli eventi» assume una forma completamente diversa. Benché esistano ovviamente dei metodi per l'acquisizione dei dati e la loro visualizzazione, la comunicazione con l'utente non avviene direttamente mediante essi ma attraverso degli «oggetti visuali» e gli «eventi» che tali oggetti sono in grado di gestire.

In effetti, non è più nemmeno appropriato parlare di acquisizione o inserimento dati, poiché l'interazione con l'utente avviene attraverso singole «azioni» – la pressione di un tasto, il clic o lo spostamento del mouse – prodotte dall'utente; azioni dirette sempre verso un determinato oggetto visuale, che può essere o meno è in grado di riceverle.

Infine, gli oggetti visuali che costituiscono l'interfaccia determinano il tipo di azioni che l'utente può dirigere verso di essi, ma non il loro ordine di esecuzione, come avviene per un modello di comunicazione «ingresso/uscita».

1.1.2 Modelli di interfaccia utente

Considerato che un'interfaccia utente è caratterizzata da due aspetti: gestione del video e modello di comunicazione con l'utente; e che per ognuno di essi esistono due forme distinte, ne consegue che sono possibili quattro modelli di interfaccia utente:

1) «a caratteri» | «ingresso/uscita»;
2) «a caratteri» | «guidata dagli eventi»;
3) «grafica» | «ingresso/uscita»;
4) «grafica» | «guidata dagli eventi».

Storicamente parlando, ognuno di questi quattro modelli ha avuto il suo periodo di grande diffusione. Attualmente (e senz'altro anche per il prossimo futuro), il modello dominante è l'ultimo, utilizzato dalle «Applicazioni Windows». Il primo modello viene comunque tuttora impiegato, soprattutto in programmi che non richiedono una vera e propria interazione con l'utente se non l'acquisizione iniziale di alcuni dati.

E' questo il modello utilizzato dalle «Applicazioni Console».

1.2 INTERFACCIA UTENTE DELLE «APPLICAZIONI CONSOLE»

Tutte le «Applicazioni Console» implementano una versione "povera" del modello di interfaccia «a caratteri» | «ingresso/uscita». L'aggettivo "povera" implica che esistono implementazioni dello stesso modello che consentono una gestione più sofisticata sia del video che della comunicazione con l'utente.

Il modello offerto dalle «Applicazioni Console» è quello della "telescrivente". In esso il video è considerato alla stessa stregua di un modulo continuo cartaceo, del tutto simile a quelli utilizzati nelle tradizionali stampanti ad aghi o a margherita. Il programma, esattamente come il carrello di una stampante o di una macchina da scrivere, scrive sullo schermo da sinistra a destra e dall'alto verso il basso. In questa analogia, il ruolo del carrello è svolto dal «cursore testo», rappresentato normalmente da una linea orizzontale lampeggiante.

Durante la visualizzazione, i caratteri vengono visualizzati sullo schermo a partire dalla posizione attuale del «cursore testo», che si sposta verso destra dopo ogni carattere visualizzato. Se il «cursore testo» si trova sul limite destro dello schermo (o della finestra), il successivo carattere viene visualizzato a partire dall'inizio della riga successiva. (avviene cioè un «ritorno carrello», o «nuova linea», automatico). Se il «cursore testo» si trova nell'ultima riga visibile dello schermo (o della finestra), il successivo ritorno a capo determinerà lo scorrimento verso l'alto di tutte le righe precedentemente visualizzate, per fare posto alla visualizzazione della nuova riga.

Esistono due metodi di visualizzazione e due di acquisizione dati, appartenenti entrambi alla classe Console:

WriteLine() e Write() per la visualizzazione;

ReadLine() e Read() per l'acquisizione, di cui sarà trattato soltanto il primo.

Tutti trattano i dati dal punto di vista testuale, e cioè come sequenze di caratteri[2].

1.2.1 Modello di comunicazione delle «Applicazioni Console»

Com'è già stato sottolineato, il modello di comunicazione «ingresso/uscita» implementato nelle «Applicazioni Console» fa sì che la comunicazione con l'utente, e quindi l'ordine con il quale vengono visualizzati e richiesti i dati, equivalga all'ordine in cui sono eseguiti i metodi Write(), WriteLine() e ReadLine(). Ed entro certi limiti, l'ordine in cui tali metodi sono eseguiti dipende dall'ordine in cui sono scritti nel codice sorgente.

Un esempio aiuterà a comprendere meglio il concetto. Segue il programma che risolve il problema di calcolare la soluzione di un'equazione di 1° grado, dati i suoi coefficienti A e B.


[«Applicazioni\Console\exe\PrimoGradoCon.exe»]

static void Main() «PrimoGradoCon» 1.1

```
{
    double a, b, x;
```

[2] Ciò non è esattamente vero per il metodo Read(), il quale comunque non viene trattato.

```
    string tmp;
    Console.Write("Immetti il valore del coefficiente A:");
    tmp = Console.ReadLine();
    a = Convert.ToDouble(tmp);

    Console.Write("Immetti il valore del coefficiente B:");
    tmp = Console.ReadLine();
    b = Convert.ToDouble(tmp);

    if (a != 0)
    {
        x = -b / a;
        Console.WriteLine("La soluzione è: {0}", x);
    }
    else            // a è uguale a zero
    {
        if (b != 0)
            Console.WriteLine("Non esiste nessuna soluzione");
        else
            Console.WriteLine("Esistono infinite soluzioni");
    }

}
```

In grigio sono evidenziate le istruzioni che appartengono alla parte interfaccia del programma[3]. Il loro ordine all'interno del codice sorgente determina anche l'ordine nel quale sono richiesti i dati all'utente, nonché l'ordine nel quale sono visualizzati i messaggi informativi e i risultati. L'esecuzione del programma produce dunque la sequenza predefinita di eventi:

1) viene richiesto il coefficiente a;
2) viene richiesto il coefficiente b;
3) se esiste viene calcolata e visualizzata la soluzione dell'equazione;
4) altrimenti viene visualizzato un appropriato messaggio informativo.

Tale sequenza è strutturata nel programma; ad esempio, non esiste alcun modo per l'utente di inserire prima il valore di b e poi quello di a; oppure di ricevere i risultati prima di aver inserito i valori dei coefficienti.

Segue l'output prodotto dal programma, ipotizzando il valore 2 per il coefficiente a e -1 per il coefficiente b:

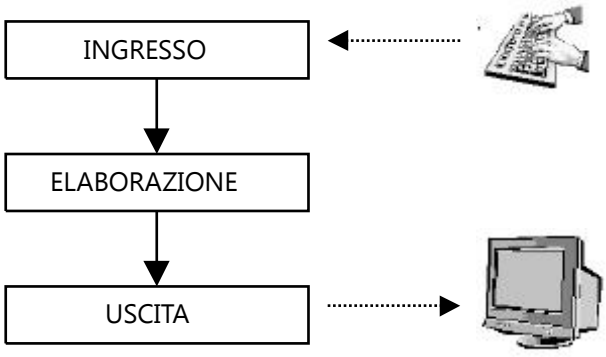
[3] Benché non gestiscano la comunicazione con l'utente, vengono considerate facenti della parte interfaccia anche le due invocazioni ai metodi ToDouble(), poiché rientrano nell'insieme di istruzioni il cui scopo è di acquisire i coefficienti a e b.

Figura 1-2. Output prodotto dall'esecuzione di «PrimoGradoCon».

La "sequenzialità" nella richiesta dei dati è anche garantita dal fatto che l'invocazione del metodo ReadLine() determina la sospensione dell'esecuzione del programma, esecuzione che riprende soltanto dopo che l'utente ha premuto il tasto INVIO. In altre parole, dunque, finché l'utente non ha inserito entrambi i coefficienti, il programma non può procedere al calcolo della soluzione dell'equazione, e non può semplicemente perché, durante l'acquisizione dati, esso non è in esecuzione.

L'aspetto della sequenzialità nella richiesta e nella visualizzazione dei dati ha importanti implicazioni (e rappresenta l'elemento caratterizzante, nonché limitante, di tale modello di comunicazione): il programmatore è in grado, semplicemente agendo sulla struttura del programma, di imporre un preciso ordine all'interazione con l'utente. Ciò consente ad esempio di implementare con estrema facilità il tipico modello di funzionamento di una «Applicazione Console», impiegato nell'esempio precedente e schematizzato nella figura sottostante:

Figura 1-3. Schema di funzionamento di una tipica «Applicazione Console».



Il modello è semplice perché è basato sulla struttura del programma:

prima le istruzioni che richiedono i dati;

poi le istruzioni che li elaborano;

infine le istruzioni che visualizzano i risultati.

Una simile architettura garantisce ad esempio che le variabili che memorizzano i dati non saranno elaborate finché l'utente non ha inserito i dati stessi; e ancora, che i risultati non saranno visualizzati finché non è stata eseguita la parte elaborazione. Entrambe le garanzie dipendono implicitamente dall'organizzazione del programma, e l'unico modo perché siano violate (ad esempio che siano visualizzati dei dati prima che siano stati eseguiti i calcoli appropriati) è che il programma sia stato strutturato in modo errato.

Capitolo 2 – Introduzione alle Applicazioni Windows

Struttura di una interfaccia grafica

- Introduzioni ai controlli
- Il controllo principale: «Form»
- «Button», «TextBox», «Label»

Struttura del codice di una «Applicazione Windows»

- Creare e aggiungere i controlli all'interfaccia
- Impostare le «proprietà» dei controlli

«Eventi» e «gestori di eventi»

- «eventi reali»: risposta di .NET alle azioni dell'utente
- «eventi C#»: risposta dei controlli agli «eventi reali»
- Gestori di eventi: risposta del programma agli «eventi C#».

2.1 STRUTTURA DI UNA INTERFACCIA GRAFICA

La Figura 1-1, se pur grossolanamente, schematizza il funzionamento di un programma che implementa il modello di interfaccia «a caratteri | «ingresso/uscita», impiegato nelle «Applicazioni Console»; ebbene, il modello di interfaccia «grafica | guidata dagli eventi» (d'ora in avanti semplicemente: interfaccia grafica) presenta un'architettura molto diversa.

Alla base di una interfaccia grafica vi sono quattro elementi fondamentali:

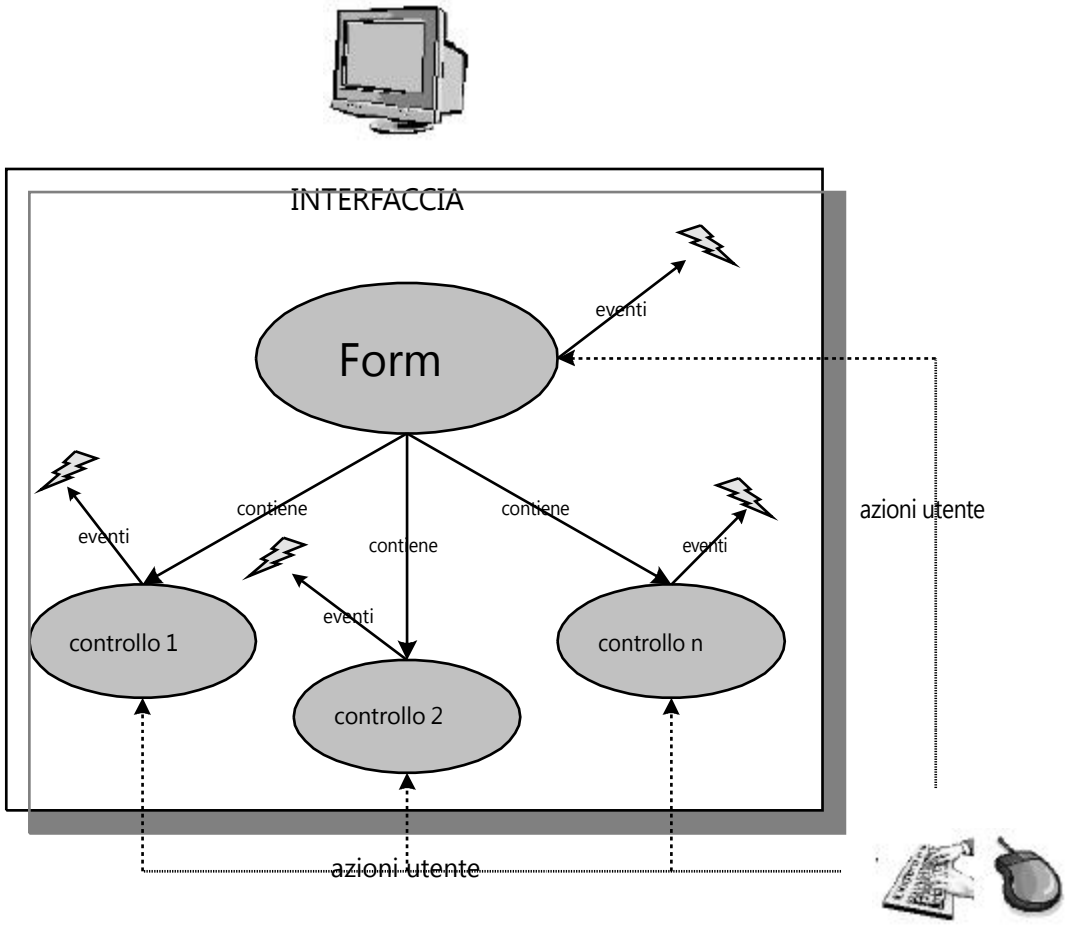
la finestra grafica – «form» – associata all'applicazione;

gli «oggetti visuali», definiti «controlli», contenuti nella finestra grafica;

gli «eventi» generati dai controlli;

i metodi che gestiscono gli eventi, o «gestori di eventi».

Figura 2-1. Rappresentazione schematica di una interfaccia grafica.



In questo modello, il dialogo tra l'utente e il programma non avviene mediante metodi di ingresso e di uscita dei dati, ma attraverso i controlli che costituiscono l'interfaccia. Essi sono di vari tipi, in base alla natura della comunicazione che svolgono con l'utente: acquisire una sequenza di caratteri, rispondere al clic del mouse, visualizzare un immagine, un'animazione, una serie di valori in forma di lista o di tabella, eccetera.

I controlli rispondono alle azioni dell'utente dirette verso di essi mettendo in atto dei comportamenti predefiniti e contemporaneamente generando – sollevando – uno o più eventi. Ad esempio, se l'utente clicca su un bottone questo si abbassa e si alza, producendo l'effetto visuale appropriato, e contemporaneamente genera l'evento Click. Come vedremo più avanti, a un evento può essere associato un metodo definito dal programmatore, chiamato «gestore dell'evento». Nel momento in cui il controllo genera un evento, il metodo – il gestore di evento – ad esso associato viene eseguito automaticamente.

2.1.1 Introduzioni ai «controlli»

I controlli sono oggetti visuali, oggetti, cioè, in grado di "visualizzare se stessi" sullo schermo. In effetti, non esistono metodi specifici per la visualizzazione dei controlli, essi lo fanno da sé; tutto ciò che deve fare il programmatore è:

a) creare il controllo;

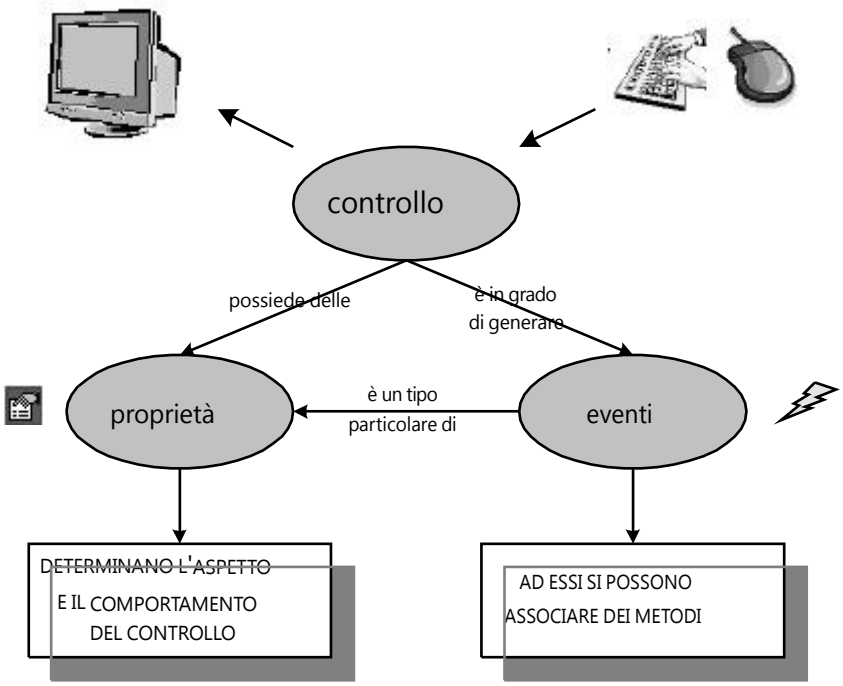
b) definirne l'aspetto, impostando opportunamente le sue «proprietà»

c) definire e associare dei metodi a gli eventi del controllo che si desidera gestire.

d) aggiungere il controllo all'interfaccia.

Dopodiché esso sarà sempre visibile e in grado di ricevere le azioni dell'utente, almeno fin quando non sarà eliminato dall'interfaccia oppure temporaneamente reso invisibile o disabilitato.

Figura 2-2. Rappresentazione schematica del rapporto tra controllo, proprietà ed eventi.



Dichiarare e creare il controllo

I controlli sono oggetti, come gli array, e dunque è possibile accedere ad essi attraverso delle variabili, ma soltanto dopo che sono stati creati. Ogni controllo, come qualsiasi altro oggetto, appartiene a un determinato tipo, o classe. La dichiarazione della variabile controllo assume l'identica sintassi di qualsiasi altra dichiarazione:

```
tipo nome-controllo;
```

Ad esempio, per dichiarare un bottone, e cioè un controllo della classe (tipo) Button, si scrive:

```
Button b;
```

dove b è il nome del il bottone. La creazione del controllo avviene, esattamente come per gli array, mediante l'operatore new:

```
nome-controllo = new tipo();
```

Ad esempio, per creare un bottone si scrive:

```
b = new Button();
```

Definire l'aspetto del controllo

Per aspetto di un controllo si intendono le sue caratteristiche visuali, come ad esempio: posizione, altezza, larghezza, colore, eccetera. Le caratteristiche, o attributi, di un controllo vengono definite assegnando gli opportuni valori alle sue «proprietà». Le proprietà sono analoghe alle variabili campi di classe e determinano appunto l'aspetto del controllo. Per modificare il valore di una proprietà si usa la sintassi:

```
nome-controllo.nome-proprietà = valore;
```

Ad esempio, per definire il testo di un bottone (posto che questo sia già stato creato) si imposta il valore della proprietà Text, che è ovviamente di tipo stringa:

```
b.Text = "Clicca qui!";
```

Per impostare invece la posizione verticale del controllo al valore 100, occorre modificare la proprietà Top:

```
b.Top = 100;
```

Associare dei metodi agli eventi del controllo

Ogni controllo definisce – il termine tecnico è «pubblica» – un certo numero di eventi di varia natura, molti dei quali sono generati in risposta alle azioni dell'utente. Se ad esempio si desidera che in risposta al clic del mouse su un bottone il programma svolga un determinato compito, è necessario scrivere un metodo che implementi il compito e associarlo all'evento Click del bottone in questione. Il metodo diventa dunque il gestore dell ' evento Click di quel bottone.

L'associazione di un metodo a un determinato evento sarà descritta più avanti.

Aggiungere il controllo all'interfaccia

Creare un controllo e definirne l'aspetto non lo rende operativo, né visibile sullo schermo: occorre prima aggiungerlo all'interfaccia. In questo caso, con la parola «interfaccia» viene designata la finestra all'interno della quale viene eseguito il programma. La finestra stessa è a sua volta un controllo, appartenente al tipo Form.

2.1.2 «Form»: il controllo principale

Ogni «Applicazione Windows» presenta almeno un controllo di tipo Form; esso è l'elemento principale dell'interfaccia, poiché è il primo ad essere creato, rappresenta la finestra associata all'applicazione e infine fa da contenitore a tutti gli altri controlli. Aggiungere un controllo all'interfaccia significa dunque aggiungerlo al form. Questo mantiene internamente una collezione di controlli chiamata Controls, la quale espone un metodo chiamato Add() per l'inserimento dei controlli. Dunque, per aggiungere un controllo al form occorre invocare il metodo Add() secondo la forma:

```
Controls.Add(nome-controllo);
```

Ad esempio, per aggiungere il bottone b al form si scrive:

```
Controls.Add(b);
```

Solo dopo l'invocazione di Add(), il bottone appartiene realmente all'interfaccia ed è in grado di ricevere le azioni dell'utente.

Di norma si aggiunge il controllo all'interfaccia solo dopo averne definito tutte le proprietà (e associato gli eventuali gestori di eventi), ma ciò non rappresenta un requisito. Il seguente codice, ad esempio, è perfettamente funzionante:

```
b = new Button();
Controls.Add(b);        // qui aggiunge il botton all'interfaccia
b.Top = 100;            // qui definisce alcune sue proprietà
b.Left = 150;
```

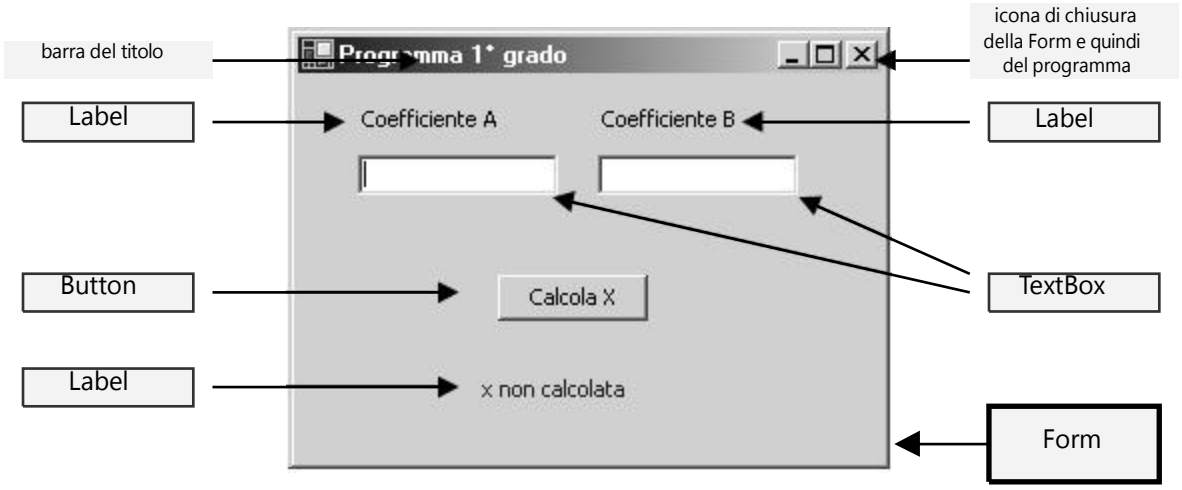
anche se non rappresenta un buon esempio di programmazione. Infatti, una volta che è stato inserito nell'interfaccia, qualsiasi modifica alle proprietà di un controllo viene immediatamente riflettuta nella sua visualizzazione. In altre parole, con un codice simile al precedente, si corre il rischio di vedere il controllo cambiare posizione all'interno del form.

2.1.3 Controlli di base di un'interfaccia: «Button», «TextBox», «Label»

Per introdurre i controlli d'uso più comune di un interfaccia grafica si farà riferimento a un programma di esempio, che risolve l'equazione di 1° grado, la cui interfaccia è rappresentata nella figura sottostante.

[«Applicazioni\Windows\exe\PrimoGradoWin.exe»]

Figura 2-3. Interfaccia del programma «PrimoGradoWin».



L'interfaccia del programma, analogo a «PrimoGradoCon» presentato nel Capitolo 1, contiene sei controlli, oltre naturalmente al form.

Controllo «TextBox»

I controlli di tipo TextBox (Casella di testo) svolgono l'analoga funzione del metodo ReadLine() e cioè l'acquisizione di stringhe di caratteri. L'utente può digitare caratteri in un TextBox soltanto se è selezionato (dentro di esso lampeggia il «cursore testo»); soltanto, cioè, se il TextBox possiede il «fuoco». Un modo molto semplice per selezionare un TextBox è cliccare al suo interno.

Controllo «Label»

I controlli di tipo Label (Etichetta) svolgono l'analoga funzione del metodo WriteLine(), e cioè la visualizzazione di stringhe di caratteri. Di norma, la loro funzione è soltanto informativa e quindi non devono ricevere le azioni dell'utente. Un uso tipico delle etichette è quello di qualificare i controlli TextBox, informando così l'utente cosa gli si richiede di inserire.

Nel programma «PrimoGradoWin», un'etichetta viene utilizzata per visualizzare la soluzione dell'equazione (che in Figura 2-3 si presuppone non ancora calcolata).

Controllo «Button»

I controlli di tipo Button (Bottone) svolgono una funzione che non ha un evidente analogo nelle «Applicazioni Console». Essi vengono di norma impiegati per dare il via a qualche forma di elaborazione, risultato che viene ottenuto associando un metodo all'evento Click del bottone.

Nel programma «PrimoGradoWin», il calcolo della soluzione dell'equazione, che rappresenta la parte «elaborazione» del programma, viene eseguito in risposta al clic dell'utente sul bottone. Tale calcolo è svolto appunto da un metodo che è associato all'evento Click.

2.1.4 Analisi del funzionamento del programma «PrimoGradoWin»

«PrimoGradoWin» risolve lo stesso problema del suo omologo «PrimoGradoCon» e possiede la stessa parte «elaborazione», ma si interfaccia con l'utente in modo completamente diverso. Il modello di comunicazione impiegato non presuppone un sequenza predefinita di azioni da parte dell'utente e di elaborazioni da parte del programma, come avviene per «PrimoGradoCon».

Infatti, l'utente è libero di:

- chiudere il programma, cliccando sull'icona di chiusura del form, che determina la terminazione immediata del programma;
- cliccare sul bottone «Calcola X», determinando così il calcolo della soluzione, senza aver inserito alcun valore in uno dei due TextBox o in entrambi;
- inserire prima il valore del coefficiente B e poi quello del coefficiente A, o viceversa;
- inserire i due valori dei coefficienti, ma non eseguire il calcolo della soluzione, cliccando invece sull'icona di chiusura del form;
- calcolare la soluzione di quante equazioni desidera, modificando ogni volta i valori dei due coefficienti e cliccando nuovamente sul bottone «Calcola X»;

Dal punto di vista del programma, invece, la parte «elaborazione» viene eseguita se e quando l'utente clicca sul bottone «Calcola X», in risposta all'evento Click generato dal bottone stesso. In «PrimoGradoCon», d'altra parte, l'esecuzione della parte «elaborazione» è implicitamente garantita dalla struttura del programma: infatti, il flusso di esecuzione, per raggiungere l'ultima istruzione del metodo Main() deve per forza "passare" per la parte «elaborazione ».

Anche in una «Applicazione Windows», dunque, l'elaborazione avviene in risposta alle azioni dell'utente, ma queste ultime non sono predeterminate dalla struttura del programma. Ciò, come vedremo, produce conseguenze notevoli, che impongono alcuni accorgimenti nella progettazione dell'interfaccia.

2.2 STRUTTURA DEL CODICE DI UNA «APPLICAZIONE WINDOWS»

Prendendo come riferimento il programma «PrimoGradoWin» sarà costruita una «Applicazione Windows» che presenti una interfaccia del tutto analoga, partendo da uno "scheletro" minimale, ma funzionante, e aggiungendo via via i vari elementi dell'interfaccia. Il nome dell'applicazione è «MioPrimoGrado».

2.2.1 "Scheletro" di una «Applicazione Windows»

Eccetto due requisiti, né .NET né C# impongono particolari prescrizioni sulla struttura del codice di una «Applicazione Windows». D'altra parte, gli ambienti di sviluppo Visual Studio.NET e Sharp Develop generano automaticamente uno "scheletro" che ricalca più o meno fedelmente la seguente struttura di base:

Scheletro di base del programma 2.1

```
using System;
using System.Windows.Forms;
using System.Drawing;

class MainForm: Form
{
    public MainForm()
    {
    }

    public static void Main()
    {
        Application.Run( new MainForm() );
    }
}
```

Tale programma, se eseguito produce un form vuoto:

Figura 2-4. Output prodotto dall'esecuzione dello scheletro di una «Applicazione Windows».

Segue una breve introduzione ai singoli elementi del programma.

- using System.Windows.Forms;

Il namespace System.Windows.Forms contiene le classi alle quali appartengono i controlli: Form, TextBox, Label, Button, eccetera.

- using System.Drawing;

L'uso del namespace System.Drawing non è in realtà strettamente necessario; d'altra parte esso contiene tipi di dati, classi e metodi spesso impiegati nelle «Applicazioni Windows».

- class MainForm: Form { }

Questo rappresenta il primo dei tre elementi davvero innovativi rispetto allo scheletro di una «Applicazione Console». MainForm è il nome della classe principale del programma (classe che contiene il metodo Main()). La notazione «: Form» (simbolo due-punti seguito dal nome della classe Form) indica che la classe MainForm «deriva» dalla classe Form, la quale, come sappiamo, designa il tipo del controllo principale di una «interfaccia Windows». Il concetto di «derivazione» di una classe rappresenta un concetto centrale dei moderni linguaggi di programmazione e per quanto attiene le «Applicazioni Windows» sarà affrontato nel prossimo capitolo; per il momento ne sarà fornita una breve spiegazione. Derivare una classe da un'altra significa:

definire un nuovo tipo di dato[4] che mantiene, ed eventualmente estende, le caratteristiche del tipo originale da cui deriva.

Derivare una classe dalla classe Form significa dunque definire un nuovo tipo di form che presenta tutte le caratteristiche del tipo originale, più eventualmente altre che è il programmatore a definire.

> In realtà non sarebbe necessario derivare una classe per realizzare l 'interfaccia di una «Applicazione Windows». In pratica ciò viene sempre fatto, per motivi che in questa fase sarebbe piuttosto difficile spiegare.

- public MainForm() { }

MainForm() rappresenta il «costruttore» della classe MainForm. Un costruttore è un metodo molto speciale, che ha lo stesso nome della classe alla quale appartiene, e:

viene invocato automaticamente nel momento in cui viene creato un oggetto di quella classe.

Le istruzioni contenute in MainForm() vengono dunque eseguite automaticamente ogni qual volta viene creato un oggetto della classe.

> Si sorvoli per il momento sulla parola chiave public, la quale non è comunque una prerogativa del costruttore MainForm() in quanto può essere applicata a qualsiasi metodo e campo di una classe, come dimostra la sua applicazione al metodo Main().

· Application.Run(new MainForm());

Innanzitutto – per renderla meglio comprensibile – questa istruzione può essere scomposta in due istruzioni distinte che producono lo stesso effetto:

```
MainForm formPrincipale = new MainForm();
Application.Run(formPrincipale);
```

che è quello di creare un oggetto della classe MainForm, e dunque un controllo form, e di passarlo come argomento al metodo Run() della classe Application. Il metodo Run() svolge la funzione di dare l'avvio vero e proprio al programma; esso visualizza infatti il form principale sullo schermo, consentendo dall'utente di cominciare a interagire con il programma. Prima di questo, però, il form principale dev'essere creato. Ciò viene fatto, come per ogni oggetto, attraverso l'operatore new. Dunque:

```
new MainForm()
```

determina la creazione un form, invocando automaticamente il costruttore della classe; l'oggetto così creato può essere subito passato come argomento al metodo Run(), come viene fatto nello scheletro, oppure prima assegnato a una variabile e quindi passato al metodo in questione. Le due modalità sono perfettamente equivalenti.

La chiusura del form principale, che avviene in risposta al clic sull'icona di chiusura, determina anche la fine dell'esecuzione del metodo Run() e dunque dell'intero programma, poiché l'invocazione di tale metodo è l'unica istruzione contenuta nel metodo Main().

Conclusioni

Lo scheletro di una «Applicazione Windows» presenta tre sostanziali novità rispetto alle ormai consuete «Applicazioni Console»:

- la classe principale deriva dalla classe Form;
- la classe principale definisce un metodo speciale, chiamato «costruttore», il quale viene invocato automaticamente quando viene creato il form;
- il metodo Main() contiene una sola istruzione, nella quale viene creato un form, il cui riferimento viene passato come argomento al metodo Run() della classe Application. Di fatto, è tale metodo che determina l'esecuzione vera e propria del programma.

Non è necessario – e sarebbe impossibile – comprendere a fondo gli elementi presentati; ciò che importa è comprendere che una «Applicazione Windows», al di là della sua complessità, presenta comunque una struttura di fondo equivalente a quella appena introdotta.

2.2.2 Aggiungere i controlli all'interfaccia

Il precedente programma funziona, ma non è in grado di svolgere alcun compito, poiché presenta un'interfaccia completamente vuota. Occorre "popolarla" dei controlli appropriati.

[4] Si ricorda che i termini «classe» e «tipo» sono per molti versi equivalenti.

Dichiarare i controlli

Perché un controllo faccia parte dell'interfaccia occorre che sia dichiarata una variabile del tipo appropriato, sia creato il controllo e sia aggiunto al form. L'interfaccia del programma «PrimoGradoWin» presenta sei controlli e quindi richiede la dichiarazione di sei variabili. Benché in tal senso né .NET né il C# impongano alcun particolare requisito, i controlli vengono di norma dichiarati come campi di classe e non come variabili locali, in modo che sia possibile accedere ad essi in qualunque metodo della classe. Segue lo scheletro del programma con l'aggiunta delle necessarie dichiarazioni (d'ora in avanti saranno omesse le direttive using):

MioPrimoGrado1 2.2

```
class MainForm: Form
{
    Button btnCalcolaX; // bottone che determina il calcolo della soluzione X
    Label lblA;         // etichetta informativa associata al TextBox txtA
    Label lblB;         // etichetta informativa associata al TextBox txtA
    Label lblX;         // etichetta che visualizza la soluzione X
    TextBox txtA;       // TextBox per l'acquisizione del coefficiente A
    TextBox txtB;       // TextBox per l'acquisizione del coefficiente B

    public MainForm()
    {
    }

    public static void Main()
    {
        Application.Run( new MainForm() );
    }
}
```

Nota bene.

1) I nomi dei controlli presentano un suffisso che ne indica il tipo: «lbl» per Label; «btn» per Button; eccetera. E' una pratica molto comune quella di utilizzare dei suffissi nei nomi degli oggetti visuali (e non solo) messi a disposizione da .NET.

2) Le dichiarazioni non sono precedute dalla parola chiave static. Brevemente, la sua presenza qualifica il campo, o il metodo, come «statico»; la sua assenza lo qualifica come campo, o metodo, «di istanza». Nel secondo caso, il campo, o il metodo, è accessibile soltanto attraverso un oggetto della classe, che dev'essere stato precedentemente creato con l'operatore new.

Creazione, impostazione e inserimento dei controlli all'interfaccia

Poiché nel momento stesso in cui il form viene visualizzato l'interfaccia dev'essere già completa, il luogo più ovvio nel quale collocare le istruzioni che creano, impostano e inseriscono i controlli nell'interfaccia è sicuramente il costruttore. Sempre nel costruttore viene impostato il testo che fa da titolo al programma, testo che compare appunto nella «barra del titolo» del form.

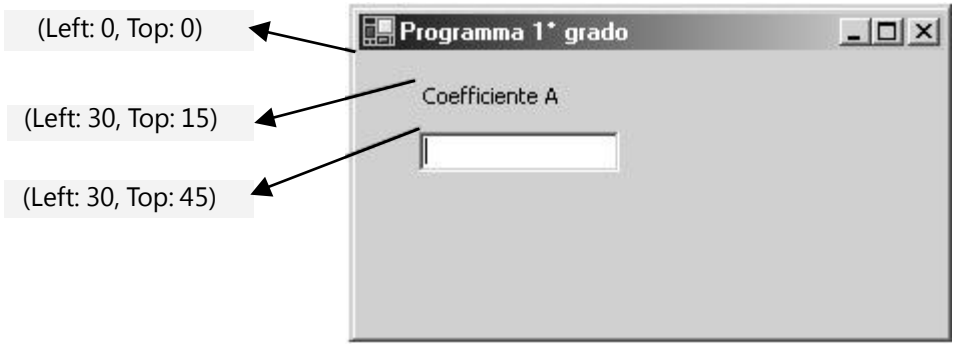
Allo scopo di procedere in modo graduale, segue l'implementazione del solo codice che definisce il testo della barra del titolo e crea e imposta la Label informativa e il TextBox relativi al coefficiente A.

MioPrimoGrado1 2.3

```
public MainForm()
{
    Text = "Programma 1° grado";          // imposta il testo della «barra del titolo»

    // crea e imposta le proprietà dell'etichetta lblA
    lblA = new Label();
    lblA.Text = "Coefficiente A";
    lblA.Top = 15;
    lblA.Left = 30;

    Controls.Add(lblA);                   // aggiunge lblA al form

    // crea e imposta le proprietà del TextBox txtA
    txtA = new TextBox();
    txtA.Top = 40;
    txtA.Left = 30;

    Controls.Add(txtA);                   // aggiunge txtA al form

...
}
```

Prima di commentare il codice, vale la pena di verificare che la sua esecuzione produce il seguente output:

Figura 2-5. Output prodotto dall'esecuzione di «MioPrimoGrado1 2.3».



La barra del titolo mostra il testo assegnato alla proprietà Text del form. I due controlli, lblA e txtA, vengono creati e inseriti nel form in posizioni dalle coordinate ben precise, espresse in pixel. Esistono vari modi per impostare le coordinate di un controllo; quello usato nel codice imposta le proprietà Left e Top, rispettivamente l'ascissa e l'ordinata in un ipotetico sistema cartesiano che ha la propria origine nell'angolo in alto a sinistra dell'area grigia del form. Infine, alla proprietà Text del controllo lblA viene assegnato l'appropriato testo informativo.

Fin d'ora è importante notare che:

alcune proprietà, come ad esempio Left, Top e Text, sono possedute da più tipi di controlli;

l'accesso alle proprietà di un controllo si ottiene sempre attraverso la notazione:

nome-controllo.nome-proprietà

quando il nome del controllo non viene specificato, la proprietà in questione si intende appartenere al form. Infatti, l'istruzione:

```
Text = "Programma 1° grado";
```

determina l'assegnazione del costante stringa "Programma 1° grado" alla proprietà Text del form, il che equivale a impostare il testo della barra del titolo.

Completamento dell'interfaccia di «MioPrimoGrado»

I passi descritti devono essere ripetuti anche per gli altri controlli. Segue il frammento di codice da aggiungere al costruttore per il completamento dell'interfaccia.

```
...
// crea e imposta le proprietà dell'etichetta lblB
lblB = new Label();
lblB.Text = "Coefficiente B";
lblB.Top = 15;
lblB.Left = 150;
Controls.Add(lblB);

// crea e imposta le proprietà del TextBox txtB
txtB = new TextBox();
txtB.Top = 40;
txtB.Left = 150;
Controls.Add(txtB);

// crea e imposta le proprietà del bottone btnCalcolaX
btnCalcola = new Button();
btnCalcola.Text = "Calcola X";
btnCalcola.Top = 100;
btnCalcola.Left = 100;
Controls.Add(btnCalcolaX);
```

```
// crea e imposta le proprietà dell'etichetta lblX
lblX = new Label();
lblX.Text = "x non calcolata";
lblX.Top = 150;
lblX.Left = 90;
lblX.ForeColor = Color.Blue;
lblX.AutoSize = true;
Controls.Add(lblX);
...
```

Non c'è molto da commentare, se non le due istruzioni evidenziate in grigio, nelle quali vengono introdotte due nuove proprietà: ForeColor e AutoSize. La prima (che sta per Foreground color) definisce il colore del testo di un controllo – in questo caso l'etichetta lblX. Esiste un elenco di identificatori predefiniti per i colori, appartenenti al tipo Color.

> In questa fase non è importante approfondire il discorso sul tipo Color. Benché esso non sia una classe, svolge una funzione analoga, e cio è quella di fungere da tipo di dato e da contenitore di metodi. Nella fattispecie, .NET definisce un insieme di proprietà , appartenenti al tipo Color, che identificano alcuni colori di base. Per accedere ad esse è dunque necessario premettere il nome Color, usando la stessa notazione impiegata per accedere a qualsiasi proprietà di un controllo.

La proprietà AutoSize fa sì che le dimensioni dell'etichetta varino automaticamente in relazione alla grandezza del testo da visualizzare.

Con l'aggiunta del precedente codice, l'esecuzione del programma produce un form del tutto identico a quello rappresentato in Figura 2-3. L'interfaccia è dunque completa e funzionante, in grado di ricevere le azioni dell'utente. Resta il fatto che non è stata ancora programmata alcuna risposta a tali azioni, nella fattispecie all'unica azione che il programma deve elaborare e cioè il clic sul bottone btnCalcolaX.

Occorre dunque "collegare" la parte interfaccia con la parte elaborazione del programma, ancora da scrivere. Ciò lo si ottiene scrivendo un gestore dell'evento Click sollevato dal bottone btnCalcolaX.

[«Applicazioni\Windows\exe\MioPrimoGrado1.exe»] 

2.3 «EVENTI» E «GESTORI DI EVENTI»

Il concetto di evento è piuttosto sfuggente, anche perché acquista significati distinti, anche se collegati , in base alla prospettiva dalla quale si considera tale termine. Esiste infatti l'evento inteso come "qualcosa che accade", ed è il significato comune e generico che viene attribuito ad esso normalmente. Esiste poi l'evento inteso come elemento del linguaggio e di .NET, e si riferisce a "una notifica di qualcosa che è accaduto".

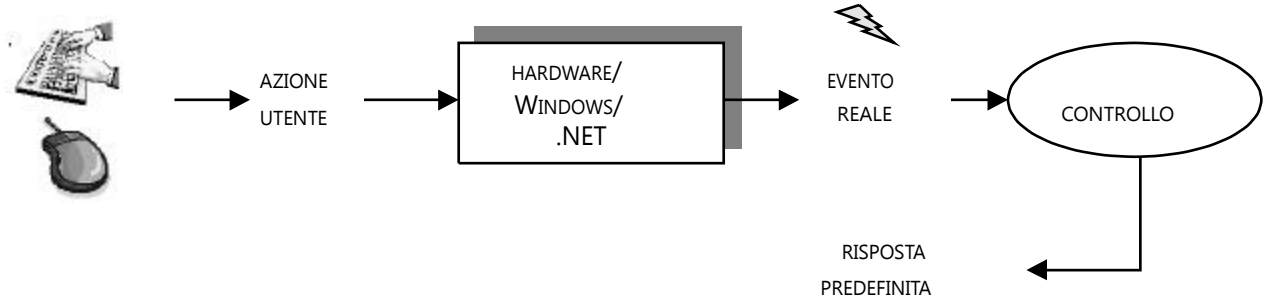
> La precedente definizione è approssimativa, poiché i controlli sono in grado di sollevare eventi anche in relazione a qualcosa "che sta accadendo " o addirittura "che deve ancora accadere ". In questa fase, come in altre parti del testo, si è scelto di rinunciare a una formulazione rigorosa, allo scopo di favorire la comprensione dei concetti e dei meccanismi presentati.

Nelle prossime righe faremo riferimento alla prima definizione di evento con il termine «evento reale», mentre faremo riferimento alla seconda con il termine «evento C#».

Per «evento reale» si intende qualcosa che accade ed è potenzialmente in grado di influenzare lo stato di esecuzione del programma. Ad esempio, la pressione di un tasto, della tastiera o del mouse, è un «evento reale», come lo è lo spostamento del mouse. Ma gli «eventi reali» non corrispondono necessariamente in modo diretto alle azioni dell'utente. La visualizzazione del form (apertura) corrisponde a un «evento reale» che si verifica dopo che l'utente ha eseguito il programma (altro «evento reale»). Analogamente, la chiusura del form, e quindi del programma, è un «evento reale» che segue il clic dell'utente sull'icona di chiusura (altro «evento reale»).

Un «evento reale» è dunque qualcosa che viene rilevato dall ' interfaccia, nella fattispecie da uno o più controlli. Se si considerano solo gli «eventi reali» che corrispondono direttamente alle azioni dell'utente, essi possono essere così schematizzati:

Figura 2-6. Schematizzazione di un «evento reale».



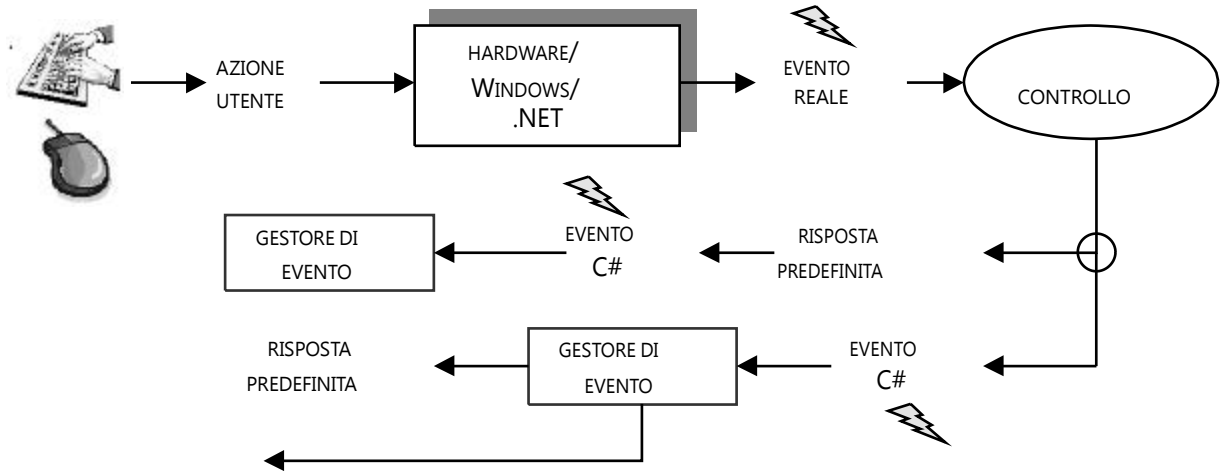
Lo schema mostra che l'azione dell'utente viene elaborata dall'hardware, dal sistema operativo Windows e da .NET per essere tradotta in un «evento reale», che il controllo verso il quale è diretta l'azione è in grado di ricevere. I controlli mettono in atto dei comportamenti predefiniti in risposta agli «eventi reali», o semplicemente non rispondono affatto. Se ad esempio si clicca in un punto qualsiasi del form non si ottiene nessuna forma di risposta; la stessa azione diretta verso un bottone produce l'effetto ottico di abbassamento e successivo innalzamento del bottone. Se si preme una

lettera mentre il «cursore testo» si trova in un TextBox, la lettera viene aggiunta o inserita nel testo. Se si clicca sull'icona di chiusura del form, questo risponde "autochiudendosi"; la stessa risposta viene ottenuta se l'utente digita la combinazione ALT + F4.

In conclusione, ad ogni «evento reale» corrisponde una risposta – anche nulla – da parte dei controlli che lo ricevono; tale risposta è chiamata predefinita poiché i controlli sono in grado di metterla in atto senza che il programmatore debba scrivere alcuna riga di codice.

Mentre un «evento reale» è ricevuto, un «evento C#» è generato dai controlli; il termine tecnico è «sollevato»[5], o «notificato». Lo scopo degli «eventi C#» è quello di consentire al programmatore di scrivere le risposte del programma agli «eventi reali», in aggiunta, o in alternativa, alle risposte predefinite messe in atto dai controlli.

Figura 2-7. Schematizzazione di un «evento reale» e conseguente «evento C#»



Lo schema mostra che dopo aver ricevuto l'«evento reale», il controllo può:

mettere comunque in atto la risposta predefinita e poi sollevare l'«evento C#»;

sollevare l'«evento C#» prima di mettere in atto la risposta predefinita;

E' importante comprendere che un «evento C#» è soltanto qualcosa di potenziale, di per sé, infatti, non produce alcunché. Un «evento C#» rappresenta un meccanismo che consente al programmatore di "agganciare" il proprio codice – il gestore di evento – agli «eventi reali», ma solo se lo desidera. Un «evento C#» rappresenta dunque una «delega»: il controllo delega il gestore di evento per rispondere all'«evento reale» che ha ricevuto. Come mostra lo schema, in base al tipo di «evento C#», il gestore di evento è in grado di influenzare, o addirittura cancellare, la risposta predefinita del controllo.

Ciò detto, d'ora in avanti, se non diversamente specificato, con il termine evento sarà designato un «evento C#».

[5] Con il termine «sollevare» non si intende qui «alzare», «portare in altro», ma «innescare», «scatenare», «dare l'avvio».

2.3.1 Gestire gli eventi di un programma

Per realizzare un programma equivalente a «PrimoGradoWin» l'unico evento da gestire è l'evento Click sollevato dal bottone btnCalcolaX. In risposta a tale evento occorre:

1) acquisire i coefficienti A e B, inseriti nei due TextBox;
2) se possibile, calcolare la soluzione X;
3) visualizzare la soluzione X, o l'appropriato testo informativo se X non è calcolabile, impostando il testo dell'etichetta lblX.

E' dunque necessario scrivere un metodo che implementi i compiti sopra elencati e associare – il termine tecnico è «attaccare» – tale metodo all'evento Click di btnCalcolaX. Dopo che il metodo è stato attaccato all'evento, il clic del mouse sopra il bottone ne determina automaticamente l'invocazione.

Prototipo di un «gestore di evento»

Un gestore di evento è un metodo che viene eseguito automaticamente in risposta all'evento al quale è attaccato; per questo motivo il suo prototipo deve rispettare la seguente forma:

```
void nome-metodo (object sender, Classe-informazioni-evento e)[6]
```

Il metodo deve cioè definire due parametri e non ritornare alcun valore.

Parametri di un gestore di evento

- object sender.

Il parametro sender (che sta per «mandante») rappresenta un riferimento al controllo che ha sollevato l'evento. Per il momento, tale parametro può essere tranquillamente ignorato.

- Classe-informazioni-evento e.

Sappiamo che un evento rappresenta la risposta di un controllo a un «evento reale» che ha ricevuto. In alcuni casi, la semplice notifica è sufficiente. E' questo il caso dell'evento Click: non importa conoscere altre informazioni se non quella che l'utente ha cliccato con il mouse sul controllo. In altre situazioni, comunque, all'evento si accompagnano necessariamente informazioni aggiuntive. Ad esempio, se, mentre il cursore si trova su un TextBox, l'utente preme un tasto, viene generato l'evento KeyPress. Tale evento non implica la sola notifica che un tasto è stato premuto, ma anche l'informazione di quale tasto è stato premuto.

Ebbene, le informazioni che qualificano ulteriormente l'evento sono contenute nel parametro e, il cui tipo dipende dal tipo dell'evento gestito. Nel caso l'evento sia di sola notifica, non necessiti

[6] I nomi «sender» ed «e» sono arbitrari (nomi diversi andrebbero altrettanto bene) ma non casuali. Se un gestore di evento viene creato attraverso i comandi disponibili in Visual Studio.NET, è con questi identificatori che vengono denominati i due parametri.

cioè di informazioni aggiuntive, il tipo in questione è: EventArgs. In questo caso, ovviamente, il parametro e può essere ignorato.

Gestore dell'evento «Click» di «btnCalcolaX»

Segue l'implementazione del metodo che gestisce l'evento Click sul bottone. Tale metodo rappresenta in pratica la parte «elaborazione» del programma.

MioPrimoGrado2 2.4

```
void btnCalcola_Click(object sender, EventArgs e)
{
    // acquisisce i coefficienti dai due TextBox
    double a = Convert.ToDouble(txtA.Text);
    double b = Convert.ToDouble(txtB.Text);
    if (a != 0)
    {
        x = -b / a;
        lblX.Text = "La soluzione è: " + x;
    }
    else            // a è uguale a zero
    {
        if (b != 0)
            lblX.Text = "Non esiste nessuna soluzione";
        else
            lblX.Text = "Esistono infinite soluzioni";
    }
}
```

La parte «elaborazione» è del tutto analoga a quella contenuta nel programma «PrimoGradoCon»; cambiano soltanto le istruzioni (evidenziate in grigio) attraverso le quali vengono acquisiti i dati e visualizzati i risultati.

> Nota bene. Il nome del gestore di evento btnCalcolaX_Click() è frutto di una scelta arbitraria ma non casuale. Esso informa che il metodo gestisce l'evento Click sollevato dal bottone btnCalcolaX. La forma utilizzata per nominare il metodo segue lo stile impiegato da Visual Studio.NET:
>
> nome-controllo_nome-evento

2.3.2 Attaccare un «gestore di evento» a un evento

L'interfaccia è completa, la parte «elaborazione» anche, ora non resta che collegarle insieme. Ciò si ottiene fornendo al metodo btnCalcolaX_Click() la "delega" per gestire l'evento Click sul bottone, esattamente mediante l'istruzione:

```
btnCalcolaX.Click += new EventHandler(btnCalcolaX_Click);
```

che produce il risultato di:

delegare il metodo btnCalcolaX_Click() a rispondere all'evento Click sollevato dal controllo btnCalcolaX.

Segue l'esame dell'istruzione, elemento per elemento.

- btnCalcolaX.Click.

Click rappresenta l'evento da gestire; si accede ad esso attraverso l'ormai consueta notazione:

nome-controllo.nome-evento

- new EventHandler()

EventHandler rappresenta il tipo di delega[7], la quale viene creata mediante l'operatore new e successivamente attaccata all'evento Click mediante l'operatore +=. Eventi di natura diversa richiedono deleghe di diverso tipo; EventHandler è una delega di tipo generico, appropriata per la gestione di eventi di sola notifica. Ad esempio, l'evento KeyPress richiederebbe la delega di tipo KeyPressEventHandler.

- btnCalcolaX_Click.

btnCalcolaX_Click è il nome del metodo che gestisce l'evento;

- +=

L'operatore di assegnazione composta += produce l'operazione di associazione all'evento Click della delega appena creata.

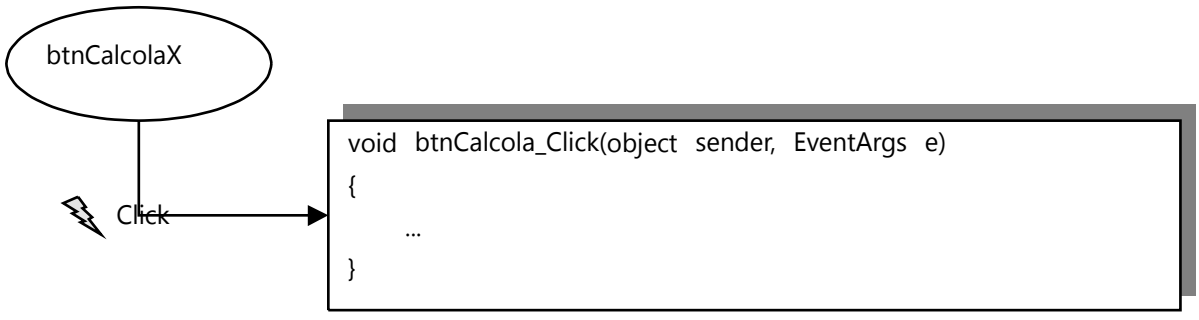
> Bench é il tipo di operazione prodotta dall'operatore += non sia una somma, essa possiede una semantica in qualche modo analoga. Infatti, mediante l 'operatore += è possibile attaccare più di un metodo allo stesso evento; procedura che per il momento non viene trattata.

In conclusione, l'esecuzione dell'istruzione:

```
btnCalcolaX.Click += new EventHandler(btnCalcolaX_Click);
```

produce un risultato che può essere così schematizzato:

Figura 2-8. Schema associazione tra metodo btnCalcolaX_Click e l'evento Click.



[7] Il termine tecnico del linguaggio è «delegate», che si può tradurre nel poco comprensibile «delegato».

E' importante comprendere che nell'istruzione:

```
controllo.evento += new tipo-delega (nome-metodo);
```

vi dev'essere corrispondenza tra tipo di evento, tipo di delega e prototipo del metodo, altrimenti il compilatore segnalerà un errore formale. In questo senso, tutti gli eventi di sola notifica, a qualunque «evento reale» essi rispondano e da qualunque controllo siano sollevati, richiedono sempre la delega EventHandler. Inoltre, i gestori di tali eventi devono presentare sempre il seguente prototipo:

```
void nome-metodo (object sender, EventArgs e)
```