C# Applicazioni Windows

Capitolo 3 – Elementi base di un'interfaccia grafica

Classe «Control»: classe «base» di tutti i controlli

- Proprietà di base del tipo «Control»
- Posizione e dimensioni dei controlli
- Tipi «Point» e «Size»
- «Fuoco» e «controllo selezionato»

Eventi di base della classe «Control»

Visualizzare messaggi: classe «MessageBox»

Controlli

- «Form»
- «Label»
- «TextBox»
- «Button»

«Consistenza» di un'interfaccia

3.1 LA CLASSE «Control»

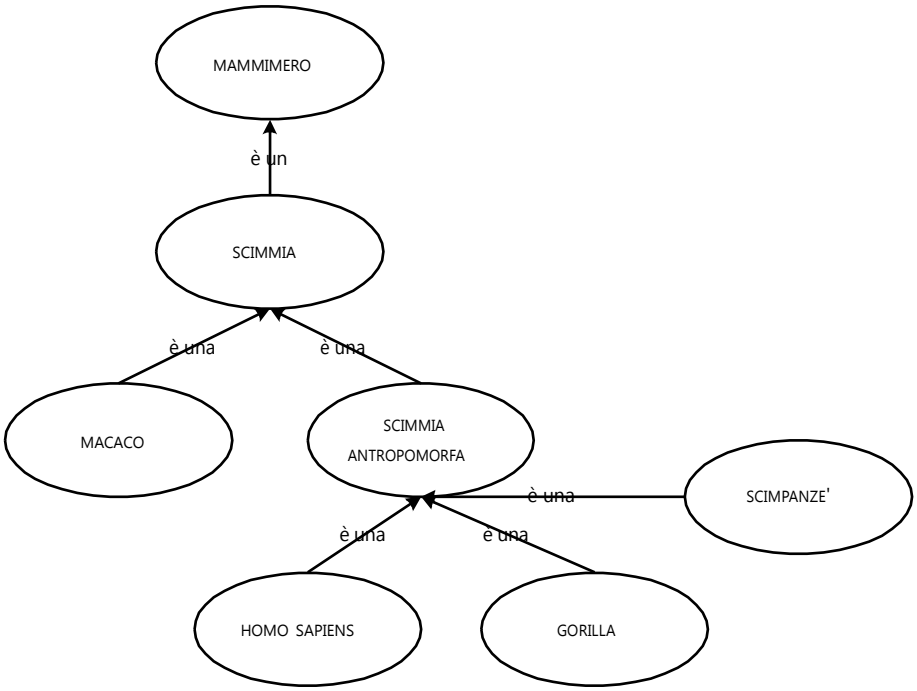
L'elemento centrale dell'interfaccia di una «Applicazione Windows» è la classe Control; ciò perché tutti i controlli – TextBox, Button, Label, Form, eccetera – prendono la maggior parte delle loro caratteristiche (derivano) dalla classe Control.

Conoscere e comprendere le caratteristiche della classe Control significa dunque conoscere e comprendere le basi di funzionamento di tutti i controlli.

3.1.1 «Derivazione» e «gerarchia di classi»

Il concetto di «derivazione» è già stato incontrato nel capitolo precedente; ora sarà approfondito, almeno per quanto riguarda l'ambito della realizzazione di interfacce grafiche. Per introdurlo useremo un'analogia. Si consideri la tassonomia del mondo animale, più precisamente quella parte che ci interessa da vicino in quanto appartenenti alla specie umana:[8]

Figura 3-1. Rappresentazione schematica delle relazioni tra alcune specie e classi animali.



I soggetti che costituiscono lo schema sono legati da una speciale «relazione di parentela» designata dal termine «è un» (o «è una»). Questa relazione, come si vede dallo schema, possiede un verso che determina il modo in cui dev'essere letta. E dunque, ad esempio: una scimmia «è un» mammifero, ma non vale il contrario, un mammifero non è una scimmia.

Nello schema ci sono inoltre animali che non possiedono alcuna relazione di parentela tra loro: il macaco non è una scimmia antropomorfa; l'homo sapiens non è un gorilla o uno scimpanzé, eccetera.

[8] Per gli interessati: lo schema presentato farebbe inorridire qualsiasi zoologo. Il suo scopo è solo quello di introdurre un'idea basandosi su un modello intuitivamente comprensibile.

Il significato della relazione di parentela «è un» è abbastanza intuitivo. Affermando che una scimmia «è un» mammifero si sostiene che tutte le scimmie condividono le caratteristiche biologiche e comportamentali di qualsiasi mammifero – quattro arti, sangue caldo, respirazione mediante polmoni, gestazione dei piccoli, eccetera – e ne introducono di nuove, che fanno parte dell'essere scimmia.

E ancora, affermare che una scimmia antropomorfa «è una» scimmia significa sostenere che essa condivide tutte le caratteristiche delle scimmie, e quindi anche tutte le caratteristiche dei mammiferi. Infine, un uomo sapiens «è una» scimmia antropomorfa e dunque condivide tutte le caratteristiche delle scimmie antropomorfe, e quindi anche delle scimmie e dei mammiferi.

Designando i termini mammifero, scimmia, homo sapiens, eccetera, come «tipi di animali», si può sostenere quindi che il tipo homo sapiens deriva («è una») dal tipo scimmia antropomorfa, che deriva dal tipo scimmia, che deriva dal tipo mammifero. In termini invertiti, si può sostenere anche che mammifero è il «tipo base» (da cui deriva) di scimmia, che è il tipo base di scimmia antropomorfa, che il tipo base di homo sapiens.

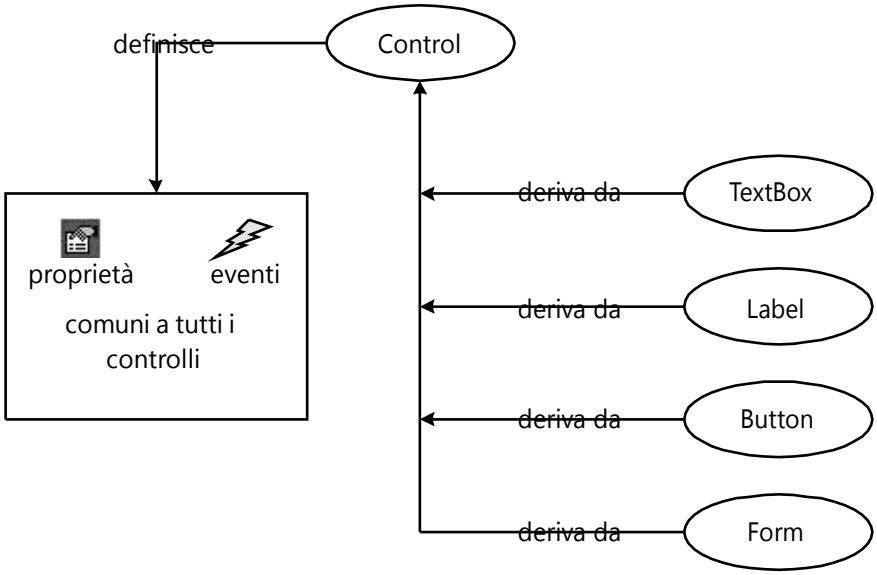
Dunque, esiste una «derivazione» – una relazione di parentela più o meno diretta – se esiste un percorso che, seguendo il verso delle frecce, collega un tipo di animale a un altro.

Lo schema mostra una gerarchia di tipi di animali. Alla base della gerarchia sta il mammifero, e cioè il tipo di animale da cui tutti gli altri derivano; esso definisce delle caratteristiche biologiche di base (che caratterizzano appunto l'essere un mammifero) le quali sono condivise da tutti gli altri tipi di animali. Ogni tipo di animale introduce nuove caratteristiche rispetto al tipo di animale da cui deriva direttamente, ma mantiene ovviamente le caratteristiche del tipo base.

3.1.2 Gerarchia dei controlli di una interfaccia grafica

I tipi di controlli di un'interfaccia grafica sono organizzati in una gerarchia analoga a quella schematizzata nel paragrafo precedente. Alla base della gerarchia sta il tipo Control.

Figura 3-2. Schema delle relazioni tra i controlli di un'interfaccia grafica.



Lo schema mostra un piccolo sotto insieme dei controlli disponibili, e cioè soltanto quelli introdotti nel capitolo precedente; soprattutto, ciò che lo schema non mostra sono i tipi di controlli intermedi. Infatti, dei controlli TextBox, Button, Form e Label, solo quest'ultimo direttamente da Control. Per il tipo di trattazione che seguirà, questo aspetto può essere ignorato.

La classe Control, esattamente come il tipo mammifero, è ciò che si definisce un «tipo astratto»; essa definisce delle proprietà e degli eventi comuni a tutti i controlli, ma in pratica nessuna interfaccia presenta mai un oggetto del tipo Control, ma soltanto oggetti il cui tipo deriva da esso, come appunto Label, Button, TextBox, eccetera.

Analogamente, non esiste un animale reale che sia semplicemente un mammifero, poiché il termine mammifero non designa una specie ma definisce un'insieme di caratteristiche comuni a varie specie.

3.1.3 Proprietà di base della classe «Control»

La classe Control definisce un notevole numero di proprietà; segue un elenco di quelle più comunemente utilizzate nei controlli che derivano da essa:

Tabella 3-1. Proprietà di base della classe Control. (continua)

PROPRIETÀ - TIPO	DESCRIZIONE
ForeColor, BackColor Color	Definiscono rispettivamente il colore di testo e dello sfondo. Per sfondo si intende l'area occupata dal controllo.
BorderStyle BorderStyle	Definisce l'aspetto del bordo che delimita l'area del controllo. I possibili valori sono: None: non viene visualizzato alcun bordo; FixedSingle: il bordo è rappresentato da un linea sottile nera; Fixed3D: il bordo è produce l'effetto visuale «bassorielievo».
BackgroundImage Image	Definisce l'immagine con la quale riempire l'area del controllo. Un modo per definire il contenuto della proprietà e quello di specificare il file che contiene l'immagine mediante il metodo FromFile della classe Image; ad esempio: controllo.BackgroundImage = Image.FromFile(@"icona.bmp");
Enabled bool	Indica se il controllo è abilitato (true) o non abilitato (false) a ricevere gli «eventi reali».
Left, Top int	Definiscono la posizione del controllo nel Form. Rispettivamente l'ascissa (Left) e l'ordinata (Top). I valori si intendono in pixel.

Tabella 3-1. Proprietà di base della classe Control.

PROPRIETÀ - TIPO	DESCRIZIONE
Width, Height int	Definiscono rispettivamente la dimensione orizzontale (Width) e verticale (Height) del controllo. I valori si intendono in pixel.
Location Point	Definisce la posizione del controllo. (Cioè le stesse informazioni delle proprietà Left e Top.)
Size Size	Definisce le dimensioni del controllo. (Cioè le stesse informazioni delle proprietà Width e Height.)
Text string	Definisce la stringa di testo visualizzata.
TabIndex int	Definisce l'ordine di tabulazione rispetto agli altri controlli; se non impostato riflette l'ordine di inserimento degli stessi nel form.
TabStop bool	Definisce se il controllo può essere raggiunto mediante il tasto di tabulazione (true) oppure no (false).
Visible bool	Indica se il controllo è visibile (true) o non visibile (false).

E' importante sottolineare fin d'ora che:

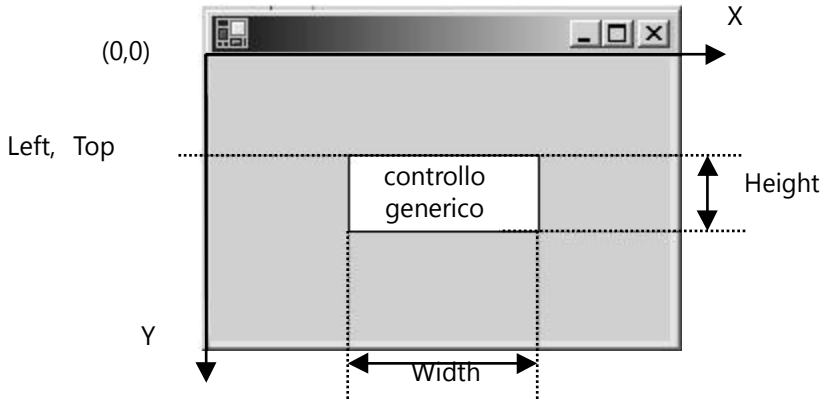
non tutte, tra le proprietà definite dalla classe Control sono effettivamente usate nelle classi derivate;

nelle classi derivate, alcune proprietà sono specializzate in base alla funzione svolta dal controllo.

3.1.4 Posizione e dimensioni dei controlli

La posizione e le dimensioni di qualsiasi controllo sono definite in relazione alla «client area» – area cliente – del form che li contiene. In figura, questa è circoscritta dal bordo giallo.

Figura 3-3. Rappresentazione della client area di un form.



La client area è rappresentata da un piano cartesiano la cui origine si trova nell'angolo in alto a sinistra. All'interno di essa ogni controllo occupa una posizione e una zona rettangolare – «area controllo» – di dimensioni ben precise, entrambe misurate in (numero di) pixel[9]. La parte di controllo che eventualmente eccede le dimensioni della client area non viene visualizzata.

La posizione di un controllo nella client area può essere espressa in due modi:

a) attraverso la coppia di proprietà Left (ascissa) e Top (ordinata);

b) attraverso la proprietà Location, che appartiene a un particolare tipo di dato: il Point.

In entrambi casi, la posizione si riferisce all'angolo in altro a sinistra dell'area controllo. Quando un controllo viene creato, sia la posizione che le dimensioni vengono automaticamente impostate a dei valori predefiniti: Left e Top sempre a zero; Width e Height in base al tipo di controllo. Di norma, il programmatore le reimposta in fase di costruzione dell'interfaccia (e cioè all'interno del costruttore del form), ma nulla impedisce di modificarle in qualsiasi punto del programma; dopo ogni modifica, il controllo viene automaticamente visualizzato in base alla nuova posizione e alle nuove dimensioni.

Ad esempio, il seguente codice (che presuppone il controllo già creato e aggiunto al form):

```
lblEsempio.Left -= lblEsempio.Width / 2;
lblEsempio.Top -= lblEsempio.Height / 2;
lblEsempio.Width += lblEsempio.Width *2;
lblEsempio.Height += lblEsempio.Height * 2;
```

raddoppia le dimensioni del controllo sia in orizzontale che in verticale, quadruplicando l'area occupata, ma senza spostare il punto centrale di tale area.

[9] Pixel sta per «picture element» e può essere tradotto in «punto».

Proprietà «Location» e tipo «Point»

La proprietà Location memorizza la posizione del controllo sotto forma di una coppia di coordinate – X e Y – e può essere manipolata come un unico valore, il cui tipo è Point. Il valore contenuto in Location riflette i valori di Top e Left, e viceversa.

Ad esempio, il seguente codice (il bottone b si suppone già creato):

```
b.Left = 100;
b.Top = 150;
```

equivale all'istruzione:

```
b.Location = new Point(100, 150);
```

che può essere tradotta in: «crea un oggetto di tipo Point e assegnalo alla proprietà Location».

E' possibile accedere, ma non modificare, alle singole coordinate memorizzate nella proprietà Location:

```
lblEsempio.Left = b.Location.X + 50; // ok: accesso all'ascissa di b
lbl.Location.Y = b.Location.Y; // errore!
```

Infine, è possibile utilizzare variabili di tipo Point.

```
Point coordinate = new Point(100, 100);
lblEsempio.Location = coordinate;
```

Proprietà «Size» e tipo «Size»

Analogamente alla proprietà Location, la proprietà Size memorizza le dimensioni del controllo sotto forma di una coppia di valori – Width e Height – e può essere manipolata come un unico oggetto, il cui tipo è Size. Il valore contenuto in Size riflette i valori delle proprietà Width e Height, e viceversa.

Ad esempio, il seguente codice (il bottone b si suppone già creato):

```
b.Width = 50;
b.Height = 25;
```

equivale all'istruzione:

```
b.Size = new Size(50, 25);
```

che può essere tradotta in: «crea un oggetto di tipo Size e assegnalo alla proprietà Size».

E' possibile accedere, ma non modificare, alle singole coordinate memorizzate nella proprietà Size:

```
lblEsempio.Width = b.Size.Width + 50; // ok: accesso alla dimensione oriz. di b
lbl.Size.Height = b.Size.Height; // errore!
```

Infine, è possibile utilizzare variabili di tipo Size.

```
Size dimensioni = new Size(50, 40);
lblEsempio.Size = dimensioni;
```

3.1.5 Concetto di «fuoco» e di «controllo selezionato»

L'idea di «fuoco» e di «controllo selezionato» sono collegate, anche se il concetto di «selezione» in generale riveste un ruolo più ampio. Il tutto si riduce a rispondere dalla seguente domanda: in un dato momento, quale controllo, tra tutti quelli inseriti nel form, riceve gli «eventi reali» provenienti dalla tastiera? O in altre parole: a quale controllo vengono inviati i codici dei caratteri digitati dall'utente? La risposta è: al controllo attualmente selezionato, e cioè al controllo che in quel momento detiene il fuoco[10].

Si immagini il fuoco come un mirino, che può essere puntato su un solo controllo per volta tra tutti quelli inseriti nel form. Quel controllo riceverà gli «eventi reali» provenienti dalla tastiera.

Controlli che possono detenere il «fuoco»

Non tutti i controlli possono detenere il fuoco; non tutti, dunque, possono ricevere eventi dalla tastiera. I controlli Label, ad esempio, non possono. I TextBox ovviamente sì, ma anche i Button. Un Button, infatti, quando è selezionato è in grado di rispondere alla pressione del tasto BARRA SPAZIATRICE, traducendo questo evento in un clic del mouse (e quindi sollevando anche l'evento Click).

Acquisire il «fuoco» e perdere il «fuoco»

Perché l'interfaccia sia effettivamente funzionale, dev'essere ovviamente possibile spostare il fuoco da un controllo all'altro. Ciò può avvenire sia come risposta predefinita ad alcune azioni dell'utente, sia come effetto dell'esecuzione del metodo Select().

L'utente può spostare il fuoco da un controllo all'altro:

c) premendo il tasto di tabulazione (TAB e SHIFT TAB): il fuoco si sposta al controllo successivo (o precedente, nel caso di SHIFT TAB) nell'ordine di tabulazione, il quale dipende, per ogni controllo, dal valore della proprietà TabIndex.

d) cliccando sul controllo con il mouse.

Questo, ovviamente, se il controllo è del tipo che può ricevere il fuoco. Nel primo caso, c'è inoltre una ulteriore condizione: la proprietà TabStop del controllo dev'essere impostata a true, altrimenti esso potrà ricevere il fuoco soltanto attraverso il mouse. (Il valore predefinito di tale proprietà è per l'appunto true.)

L'esecuzione del metodo Select(), secondo la forma:

```
nome-controllo.Select();
```

fa sì che il controllo in questione riceva il fuoco. Dunque il programmatore può far sì che un controllo sia selezionato in risposta a un determinato evento o in base al risultato di una qualche elaborazione.

[10] In realtà un controllo può essere nello stato «selezionato» pur senza avere il fuoco, ma ciò soltanto quando il form che contiene il controllo in questione non è attivo.

Ogni controllo, sia quando riceve il fuoco, sia quando lo perde, solleva due eventi di sola notifica: GotFoucs ed Enter dopo che ha ricevuto il fuoco; LostFocus e Leave, dopo che lo ha perso. Attaccando ad essi dei gestori di eventi è possibile eseguire determinati compiti dopo che il controllo ha ricevuto, o perduto, il fuoco.

«Apparenza» di un controllo selezionato

Poiché l'utente deve sapere quale controllo è selezionato in un dato momento, di norma ogni controllo modifica il proprio aspetto in base al fatto che possieda il fuoco o meno. Un Button selezionato mostra ad esempio un bordo tratteggiato che circoscrive il testo. Il TextBox, invece, mostra il cursore testo.

3.2 EVENTI DI BASE DELLA CLASSE «Control»

Segue una lista di alcuni degli eventi definiti dalla classe Control.

Tabella 3-2. Eventi prodotti dal mouse (*).

EVENTO - CLASSE INFORMAZIONI - NOME DELEGA	DESCRIZIONE («EVENTO REALE» CHE DETERMINA L'INVOCAZIONE DELL'EVENTO»
Click	Clic con il pulsante sinistro del mouse. (Oppure quello destro in base alle impostazione del mouse.)
DoubleClick	Doppio clic con il pulsante sinistro del mouse. (Oppure quello destro in base alle impostazione del mouse.)
MouseHover	Il puntatore del mouse si trova sopra il controllo. L'evento viene sollevato in continuazione ad intervalli regolari finché il puntatore non esce dal controllo.
MouseEnter	Il puntatore del mouse "entra" nell'area del controllo.
MouseLeave	Il puntatore del mouse "lascia" l'area del controllo.
MouseDown - MouseEventArgs - MouseEventHandler	Abbassamento di un pulsante qualsiasi del mouse.
MouseMove - MouseEventArgs - MouseEventHandler	Spostamento del puntatore del mouse sopra l'area del controllo.
MouseUp - MouseEventArgs - MouseEventHandler	Rilascio di un pulsante qualsiasi del mouse. Nel caso del pulsante sinistro, il rilascio viene rilevato anche se il pulsante non si trova più sopra l'area del controllo quando ciò avviene. Non è così nel caso del pulsante destro.

(*) Per gli eventi di sola notifica non sono specificate la classe informazioni e il nome della delega; la prima è sempre EventArgs, la seconda è sempre EventHandler.

Tabella 3-3. Eventi prodotti dalla tastiera.

EVENTO - CLASSE INFORMAZIONI - NOME DELEGA	DESCRIZIONE («EVENTO REALE» CHE DETERMINA L'INVOCAZIONE DELL'EVENTO»
KeyDown - KeyEventArgs - KeyEventHandler	Un tasto qualsiasi viene premuto. (Compresi i tasti SHIFT, CAPS, CTRL e CTRL, ALT GR).
KeyUp - KeyEventArgs - KeyEventHandler	Un tasto qualsiasi viene rilasciato. (Compresi i tasti SHIFT, CAPS, CTRL e CTRL, ALT GR)
KeyPress - KeyPressEventArgs - KeyPressEventHandler	Un tasto viene premuto e rilasciato. (Questo evento rileva soltanto la pressione di un sotto insieme dei tasti presenti sulla tastiera, dal quale sono esclusi ad esempio i tasti precedentemente elencati, i tasti funzione (F1, F2, ect), i tasti freccia, i tasti INS e CANC, eccetera.

3.2.1 Classe «MouseEventArgs»

Il tipo di delega degli eventi MouseDown, MouseUp, MouseMove, richiede che il secondo parametro del gestore di evento sia di tipo MouseEventArgs. Questa classe espone delle proprietà che memorizzano informazioni sullo stato del mouse nel momento in cui viene sollevato l'evento.

Tabella 3-4. Proprietà della classe MouseEventArgs (*).

PROPRIETÀ - TIPO	DESCRIZIONE
Button MouseButton	Indica quale pulsante del mouse è stato premuto o rilasciato. I pulsanti sono definiti mediante le costanti: Left, Middle, Right.
X, Y int	Definiscono le coordinate (ascissa e ordinata) del puntatore del mouse. Le coordinate sono relative all'angolo in alto a sinistra dell'area controllo.

(*) Sono elencate solo le proprietà d'uso più comune.

Normalmente non serve gestire gli eventi MouseDown, MouseMove, MouseUp, se non in programmi che richiedono una sofisticata interazione con l'utente, come programmi di disegno e videoscrittura.

3.2.2 Classi «KeyEventArgs» e «KeyPressEventArgs»

Il tipo di delega degli eventi KeyDown, KeyUp richiede che il secondo parametro del gestore di evento sia di tipo KeyEventArgs. Questa classe espone delle proprietà che memorizzano informazioni sullo stato della tastiera nel momento in cui viene sollevato l'evento.

Tabella 3-5. Proprietà della classe KeyEventArgs (*).

PROPRIETÀ - TIPO	DESCRIZIONE
Alt, Control, Shift bool	Indicano se i tasti omonimi (ALT, CTRL, SHIFT) sono premuti (true) oppure no (false).
Handled bool	Se impostata al valore true, comunica al controllo che l'evento è stato gestito e che dunque non è più necessaria alcuna forma di elaborazione sul tasto premuto. (In pratica, inibisce la risposta predefinita.)
KeyCode Keys	Contiene il codice del tasto premuto. I codici dei tasti sono definiti mediante delle costanti appartenenti al tipo Keys. Ad esempio: Keys.Enter, Keys.Del, Keys.End rappresentano rispettivamente i tasti ENTER, DEL, END.

(*) Sono elencate solo le proprietà d'uso più comune.

Il tipo di delega dell'evento KeyPress richiede che il secondo parametro del gestore di evento sia di tipo KeyPressEventArgs. Questa classe espone le sole proprietà Handled e KeyChar; la seconda, di tipo char, contiene il codice Unicode del tasto premuto.

Di norma, si gestisce l'evento KeyDown (o KeyUp) quando è necessaria un controllo sofisticato della tastiera; più semplice e più comune è la gestione dell'evento KeyPress, che produce il codice Unicode del carattere digitato. Un esempio sulla gestione di tale evento sarà mostrato più avanti.

3.2.3 Commento sugli eventi MouseXXX e KeyXXX

Tali eventi sono definiti anche di "basso livello" poiché hanno una corrispondenza diretta con le azioni dell'utente (user generated events) e forniscono informazioni dettagliate sullo stato di tali azioni. Di solito non è necessario interagire con l'utente fino a questo livello. Ad esempio, nel gestire gli eventi relativi a un Button, poco importa conoscere le coordinate del puntatore del mouse quando l'utente clicca su di esso (informazioni fornite dagli eventi MouseDown e MouseUp), ciò che serve è semplicemente poter rispondere al clic dell'utente.

Analogamente, i TextBox dialogano in modo molto sofisticato con l'utente, gestendo automaticamente i tasti INS, DEL, HOME, END, UP, DOWN, eccetera, oltre al mouse naturalmente, senza che al programmatore sia richiesto di scrivere una sola riga di codice. E di norma non c'è alcun bisogno che esso intervenga in questa gestione, benché gli eventi KeyPress, KeyDown e KeyUp gli consentano di farlo.

3.2.4 Altri eventi della classe «Control»

Gli eventi sotto elencati non hanno una corrispondenza diretta con azioni dell'utente.

Tabella 3-6. Altri eventi della classe Control.

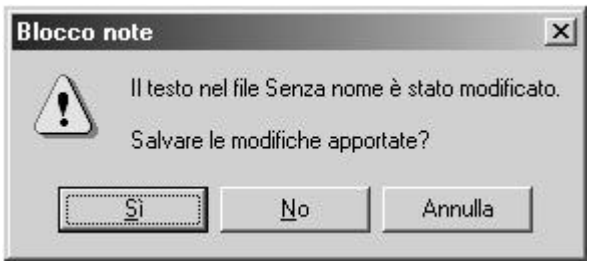
EVENTO (*) - CLASSE INFORMAZIONI - NOME DELEGA	DESCRIZIONE («EVENTO REALE» CHE DETERMINA L'INVOCAZIONE DELL'EVENTO»
Enter	Il controllo diventa selezionato.
Leave	Il controllo perde la selezione.
GotFocus	Il controllo riceve il fuoco.
LostFocus	Il controllo perde il fuoco.
Validating - CancelEventArgs - CancelEventHandler	Il controllo è in fase di "validazione". Gestendo questo evento è possibile impedire che il controllo, ad esempio un TextBox, perda il fuoco almeno fin quando non contiene un valore appropriato. (I tipi CancelEventArgs e CancelEventHandler sono definiti nel namespace: System.ComponentModel)
Validated	Questo evento viene di norma gestito in combinazione con l'evento Validating. Validated viene sollevato se la fase di "validazione" dà esito positivo.

Gli eventi GotFocus ed Enter da una parte, e LostFocus e Leave dall'altra non sono dei duplicati, poiché vengono sollevati in momenti diversi durante l'acquisizione, o perdita, del fuoco. Inoltre il loro funzionamento è diverso nel caso di interfacce composte da più form.

Per il momento, comunque, possono essere considerati eventi dal funzionamento equivalente.

3.3 VISUALIZZARE MESSAGGI: CLASSE «MessageBox»

In molte situazioni si presenta la necessità di comunicare all'utente lo stato di una certa elaborazione, oppure di chiedere una conferma prima di procedere o meno all'esecuzione di una determinata operazione. Un esempio, che riguarda il programma «Blocco note» (come qualsiasi programma di videoscrittura), è la richiesta di conferma per il salvataggio del documento, richiesta avanzata all'utente quando questo tenta di chiudere il programma senza aver prima salvato il documento:



L'esempio mostra un particolare tipo di form, chiamato «message dialog», che mediante tre Button chiede all'utente di selezionare una fra tre alternative possibili. La principale caratteristica di una message dialog è che assume il pieno controllo sulle azioni dell'utente; in altre, parole, finché la dialog non viene chiusa non è possibile accedere al resto dell'interfaccia.

Le message dialog sono fondamentalmente impiegate per servire i seguenti scopi:

1) visualizzare un messaggio informativo;
2) visualizzare un messaggio informativo di errore;
3) visualizzare un messaggio di avvertimento o di richiesta di conferma (vedi esempio).

Una message dialog viene visualizzata mediante l'esecuzione del metodo Show() della classe MessageBox. L'invocazione di tale metodo può assumere varie forme, in base al numero di elementi con i quali si intende caratterizzare la message dialog. Il suo prototipo è:

```
DialogResult Show(string testo,
                  string titolo_opz,
                  MessageBoxButtons buttons_opz,
                  MessageBoxIcon icon_opz);
```

dove:

testo	rappresenta il messaggio da visualizzare;
titolo	rappresenta il testo da visualizzare sulla barra del titolo della message dialog;
buttons	indica quali bottoni visualizzare;
icon	indica con quale icona caratterizzare la message dialog.

Come si vede dal prototipo, solo il parametro testo – il contenuto del messaggio – è obbligatorio; gli altri hanno lo scopo di caratterizzare l'apparenza e il comportamento della message dialog.

Il metodo Show() produce un valore di ritorno di tipo DialogResult che rappresenta la risposta dell'utente, e cioè il bottone cliccato; esso è definito dalle seguenti costanti:

Tabella 3-7. Elenco dei possibili valori prodotti del metodo Show().

RIPOSTA UTENTE - TIPO DialogResult	DESCRIZIONE
Abort	E' stato cliccato il bottone «Termina».
Cancel	E' stato cliccato il bottone «Annulla».
Ignore	E' stato cliccato il bottone «Ignora».
No	E' stato cliccato il bottone «No».
OK	E' stato cliccato il bottone «Ok».
Retry	E' stato cliccato il bottone «Riprova».
Yes	E' stato cliccato il bottone «Si».

3.3.1 Message dialogs informative

Una message dialog informativa si limita a comunicare un certo messaggio all'utente, senza chiedere a quest'ultimo di compiere alcuna scelta. Nella sua forma più semplice appare così:

Figura 3-4. Esempio di message dialog informativa.

Tale risultato si ottiene mediante l'invocazione:

```
MessageBox.Show("Questo è un messaggio informativo");
```

Come si nota, il valore ritornato dal metodo Show() viene ignorato ed è naturale che sia così, poiché, per definizione, l'unica azione concessa all'utente è quella di cliccare sul Button «OK».

La precedente dialog è piuttosto avara di contenuti; occorre senz'altro specificare un testo da visualizzare sulla barra del titolo, che può qualificare il tipo di messaggio, oppure semplicemente visualizzare il nome del programma:

Figura 3-5. Esempio di message dialog informativa con testo sulla barra del titolo.

Ciò si ottiene specificando come secondo argomento il testo da visualizzare sulla barra del titolo:

```
MessageBox.Show("Questo è un messaggio informativo", "Questo è il titolo");
```

Infine, anche se non strettamente necessario, può essere appropriato visualizzare un'icona che mostra la natura informativa della dialog:

Figura 3-6. Esempio di message dialog informativa con icona «Information».

Ciò richiede di specificare altri due parametri, poiché è impossibile visualizzare una message dialog con icona senza specificare anche quali bottoni visualizzare:

```
MessageBox.Show("Questo è un messaggio informativo",
                "Questo è il titolo",
                MessageBoxButtons.OK,
                MessageBoxIcon.Information);
```

Il testo in grassetto rappresenta il parametro che caratterizza l'icona visualizzata. I possibili valori del parametro icon sono:

Tabella 3-8. Elenco dei possibili valori del parametro icon.

ICONA - TIPO MessageBoxIcon	DESCRIZIONE
Asterisk	Viene visualizzata l'icona «informazione».
Error	Viene visualizzata l'icona «errore»
Esclamation	Viene visualizzata l'icona «punto esclamativo».
Hand	Viene visualizzata l'icona «mano».
Information	Viene visualizzata l'icona «informazione».
Question	Viene visualizzata l'icona «punto di domanda».
Stop	Viene visualizzata l'icona «cartello di stop».
Warning	Viene visualizzata l'icona «punto esclamativo».

Message dialogs informative di errore

Una message dialog di errore comunica uno stato errato del programma; di norma, ciò avviene aggiungendo l'icona appropriata al messaggio informativo.

Ciò si ottiene mediante un'invocazione del tipo:

```
MessageBox.Show("Messaggio di errore!",
                "Questo è il titolo",
                MessageBoxButtons.OK,
                MessageBoxIcon.Error);
```

3.3.2 Elaborare la risposta dell'utente

Mediante le message dialog puramente informative la comunicazione avviene sempre e solo dal programma verso l'utente. Quest'ultimo può solo confermare di aver "ricevuto" il messaggio. Diverso è il caso delle message dialog di avvertimento o conferma. Con esse si chiede all'utente di selezionare una tra più scelte possibili, rappresentate dai bottoni visualizzati. In questo caso, ovviamente, il programma dovrà elaborare in qualche modo la risposta dell'utente.

> Ci ò non rappresenta affatto un requisito del linguaggio. D'altra parte è illogico visualizzare ad esempio una message dialog che presenta i bottoni «OK » e «Annulla » e poi non verificare quale tra le due scelte ha effettuato l'utente.

Segue un frammento di codice che visualizza una message dialog di conferma ed elabora successivamente il valore ritornato dal metodo `Show()`, che rappresenta appunto la risposta dell'utente:

```
DialogResult scelta;
scelta = MessageBox.Show("Confermi l'operazione?","Questo è il titolo",
                         MessageBoxButtons.YesNo,
                         MessageBoxIcon.Warning);
if (scelta == DialogResult.Yes)
{
    ...// effettua l'operazione di cui è stata richiesta la conferma
}
```

La message dialog visualizzata è:

Figura 3-7. Esempio di message dialog di richiesta conferma.

In questo caso, il metodo Show() ritorna un codice che rappresenta la risposta fornita dall'utente; codice che nell'istruzione precedente viene assegnato alla variabile scelta. Ovviamente, il tipo e il numero dei bottoni visualizzati determina il tipo di risposta che può fornire l'utente.

Segue l'elenco dei valori che può assumere il parametro buttons:

Tabella 3-9. Elenco dei possibili valori del parametro buttons.

BOTTONI - TIPO MessageBoxButtons	DESCRIZIONE
AbortRetryIgnore	Sono visualizzati i bottoni «Termina», «Riprova», «Ignora».
OK	E' visualizzato il bottone «OK».
OKCancel	Sono visualizzati i bottoni «OK», «Annulla».
RetryCancel	Sono visualizzati i bottoni «Riprova», «Annulla».
YesNo	Sono visualizzati i bottoni «Si», «No».
YesNoCancel	Sono visualizzati i bottoni «Si», «No», «Annulla».

Come si vede, il tipo MessageBoxButtons definisce delle costanti per tutte le combinazioni possibili, tra quelle logicamente ammissibili.

3.3.3 Conclusioni

La classe MessagBox, con i tipi correlati – MessageBoxButtons, DialogResult, MessageBoxIcon – fornisce al programmatore la massima flessibilità per affrontare situazioni nelle quali è richiesta una comunicazione molto semplice con l'utente. Ovviamente sta al programmatore garantire la coerenza tra tutti gli elementi che caratterizzano una message dialog:

messaggio visualizzato;

testo visualizzato sulla barra del titolo;

bottoni visualizzati;

eventuale icona visualizzata.

verifica della risposta dell'utente.

3.4 CONTROLLI «Form», «Label», «TextBox» , «Button»

I controlli di tipo Form, Label, TextBox, come qualsiasi altro tipo di controllo, ereditano le caratteristiche – proprietà ed eventi – dalla classe Control, aggiungendone di nuove. Ognuno di essi, essendo stato realizzato per svolgere particolari funzioni all'interno dell'interfaccia, specializza alcune delle proprietà ereditate, oppure le nasconde addirittura.

Segue una loro breve caratterizzazione.

3.4.1 Classe «Form»

Rispetto agli altri tipi di controllo, la classe Form rientra in una categoria a parte, poiché di fatto non esiste un'interfaccia grafica senza almeno un form. La classe espone un notevole numero di proprietà ed eventi; di entrambi ne sarà esaminato soltanto un piccolo sotto insieme.

Proprietà

Tabella 3-10. Proprietà della classe Form.

PROPRIETÀ - TIPO	DESCRIZIONE
ClientSize Size	Memorizza le dimensioni della client area del form.
ForeColor (*) Color	Impostando questa proprietà viene modificato il colore di ForeColor di tutti controlli contenuti nel form per i quali non è stata impostata la proprietà omologa.
FormBorderStyle FormBorderStyle	Stile del bordo. Questa proprietà influenza sia l'apparenza del bordo che la possibilità di poter dimensionare il form. I possibili valori sono: None, FixedSingle, Fixed3D, FixedDialog, Sizable, FixedToolWindow, SizableToolWindow.
KeyPreview bool	se true, fa sì che il form riceva gli eventi di tastiera prima del controllo che possiede il fuoco.
MinimumSize MaximumSize Size	Dimensioni minime e massime che può assumere il form.
StartPosition FormStartPosition	Posizione iniziale del form sullo schermo. I possibili valori sono: Manual, CenterScreen, WindowsDefaultLocation, WindowsDefaultBounds, CenterParent

(*) Entrambe le proprietà non influenzano affatto l'apparenza del testo contenuto nella proprietà Text del form.

Segue un programma di esempio che illustra l'uso di alcune delle proprietà elencate.

[«Applicazioni\Windows\exe\PrimoForm1.exe»] 

ProvaForm1 3.1

```
class MainForm
{
    Label lblProva1, lblProva2;
    public MainForm()
    {
        Text = "ProvaForm1";
        ForeColor = Color.Red;
        MinimumSize = new Size(50, 50);
        MaximumSize = new Size(250, 150);
        StartPosition = FormStartPosition.CenterScreen;

        // prima etichetta
        lblProva1 = new Label();
        lblProva1.Location = new Point(50, 50);
        lblProva1.Text = "testo di prova 1";
        lblProva1.ForeColor = Color.Green;
        Controls.Add(lblProva1);

        // seconda etichetta
        lblProva2 = new Label();
        lblProva2.Location = new Point(50, 80);
        lblProva2.Text = "testo di prova 2";
        Controls.Add(lblProva2);
    }

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

L'esecuzione del programma produce il seguente output:

Figura 3-8. Output prodotto dal programma «ProvaForm1».

Eventi

Tabella 3-11. Eventi della classe Form. (*)

EVENTO - CLASSE INFORMAZIONI - NOME DELEGA	DESCRIZIONE
Activated	Viene sollevato dopo che il form è diventato attivo. Un form si dice attivo quando riceve il controllo sulle azioni dell'utente. Un form attivo mostra una barra del titolo colorata (la barra del titolo di un form non-attivo è grigia).
Deactivate	Viene sollevato prima che il form diventi non-attivo.
Closing - CancelEventArgs - CancelEventHandler	Viene sollevato mentre è in atto un tentativo di chiusura del form. Impostando a true la proprietà Cancel del parametro informazioni evento il tentativo di chiusura viene abortito.
Closed	Viene sollevato dopo che il form è stato chiuso. (Ma prima che scompaia dallo schermo)
Load	Viene sollevato dopo che il form è stato costruito, ma prima che sia visualizzato.

(*) Per gli eventi di sola notifica non sono specificate la classe informazioni e il nome della delega; la prima è sempre EventArgs, la seconda è sempre EventHandler.

Questi eventi sono contestuali ai vari cambiamenti di stato di un form. Per quanto riguarda il form principale di una applicazione:

- viene creato una sola volta, contestualmente all'esecuzione del programma (evento Load);
- viene chiuso una sola volta, contestualmente alla fine del programma (eventi Closing e Closed);
- come ogni altro form, può diventare attivo e non attivo un numero arbitrario di volte (eventi Activated e Deactivate);

In relazione al livello introduttivo che stiamo considerando, raramente si rende necessario gestire uno o più degli eventi elencati. Viene comunque fornito un esempio di gestione dell'evento Closing, attraverso il quale il programmatore può intervenire sulla sequenza di chiusura di un form. (Si immagini che il codice sia aggiunto al precedente programma).


[Applicazioni\Windows\exe\ProvaForm2.exe]

public MainForm() ProvaForm2 3.2

```
{
    ...
    Closing += new CancelEventHandler(MainForm_Closing);
```

```
    ...
}
void MainForm_Closing (object sender, CancelEventArgs e)
{
    DialogResult s;
    s = MessageBox.Show("Confermi chiusura del programma?", "ProvaForm2",
                        MessageBoxButtons.YesNoCancel);
    if (s != DialogResult.Yes)                    // chiusura non confermata
        e.Cancel = true;
}
```

Mediante una message dialog viene chiesto all'utente se intende confermare la chiusura del form (e quindi del programma). Se l'utente non conferma (e dunque non clicca sul bottone «Si») viene impostata a true la proprietà Cancel del parametro e: la procedura di chiusura viene abortita.

3.4.2 Classe «Label»

L'uso più comune delle Label è quello di svolgere il ruolo di testo informativo, con lo scopo di qualificare gli altri controlli, soprattutto TextBox, specificando la natura delle informazioni che essi visualizzano e/o consentono di inserire. D'altra parte nulla impedisce di impiegarle direttamente per l'output dei dati, come è stato fatto nel programma «MioPrimoGrado».

Proprietà

Tabella 3-12. Proprietà della classe Label.

PROPRIETÀ - TIPO	DESCRIZIONE
AutoSize bool	Impostando a true questa proprietà le dimensioni del controllo variano in modo automatico in relazione allo spazio occupato dal testo da visualizzare. Tale spazio dipende oltre che dalla "quantità" di testo anche dalle caratteristiche del Font impostato.
Text string	Contiene il testo da visualizzare. Se all'interno del testo è presente il carattere & («e» commerciale), il carattere che lo segue viene interpretato come «tasto di accesso» o «acceleratore», e cioè un tasto che premuto in combinazione con ALT determina la selezione del controllo associato alla Label.
TextAlign ContentAlignment	Consente di impostare l'allineamento del testo rispetto all'area del controllo. (Funziona solo se AutoSize è false). I possibili valori sono: BottomCenter, BottomLeft, BottomRight, MiddleCenter, MiddleLeft, MiddleRight, TopCenter, TopLeft, TopRight.

Una Label può essere "associata" al controllo che la segue nell'ordine di tabulazione (proprietà TabIndex); se l'utente digita la combinazione ALT + «acceleratore», il controllo in questione viene automaticamente selezionato.

Segue un programma di esempio che illustra l'uso di alcune delle proprietà elencate.

[Applicazioni\Windows\exe\ProvaLabel.exe]

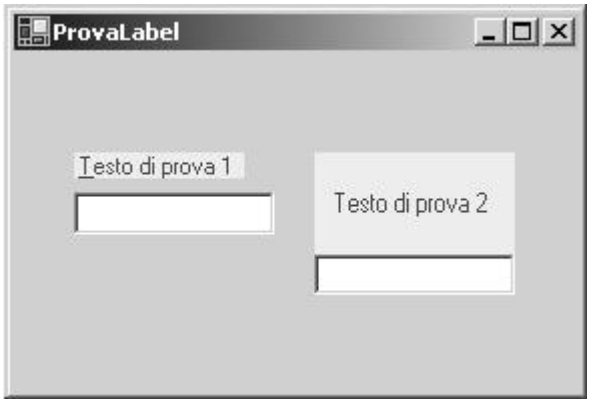
class MainForm ProvaLabel 3.3

```
{
    Label lblProva1, lblProva2;
    TextBox txtProva1, txtProva2;
    public MainForm()
    {
        Text = "ProvaLabel";
        StartPosition = FormStartPosition.CenterScreen;

        // prima etichetta
        lblProva1 = new Label();
        lblProva1.Location = new Point(30, 50);
        lblProva1.Text = "&Testo di prova 1";
        lblProva1.ForeColor = Color.Green;
        lblProva1.BackColor = Color.Blue;
        lblProva1.AutoSize = true;
        Controls.Add(lblProva1);

        // TextBox associato a lblProva1
        txtProva1 = new TextBox();
        txtProva1.Location = new Point(30, 70);
        Controls.Add(txtProva1);

        // seconda etichetta
        lblProva2 = new Label();
        lblProva2.Location = new Point(150, 50);
        lblProva2.Size = new Size(100, 50);
        lblProva2.Text = "Testo di prova 2";
        lblProva2.TextAlign = ContentAlignment.MiddleCenter;
        lblProva2.ForeColor = Color.Yellow;
        lblProva2.BackColor = Color.Blue;
        Controls.Add(lblProva2);

        txtProva2 = new TextBox();
        txtProva2.Location = new Point(150, 100);
        Controls.Add(txtProva2);
    }

    public static void Main()
    {
```

```
            Application.Run(new   MainForm());
        }
}
```

L'esecuzione del programma produce:

Figura 3-9. Output prodotto dal programma «ProvaLabel».



Poiché il controllo txtProva1 è stato aggiunto al form immediatamente dopo lblProva1, il primo viene associato al secondo; di conseguenza, la combinazione di tasti ALT + T (dove T è il tasto di accesso definito nel testo dell'etichetta) determina la selezione di txtProva1, ovunque si trovi il fuoco in quel momento.

Eventi

Di norma non c'è alcun bisogno di gestire gli eventi sollevati da una etichetta.

3.4.3 Classe «TextBox»

Di norma, il ruolo svolto dai controlli TextBox è quello consentire l'inserimento dei dati, benché nulla impedisca di utilizzarli a semplice scopo di visualizzazione.

Proprietà

Tabella 3-13. Proprietà della classe TextBox. (continua)

PROPRIETÀ - TIPO	DESCRIZIONE
CharacterCasing CharacterCasing	Impostando questa proprietà è possibile forzare il «case» dei caratteri digitati dall'utente, trasformandoli in maiuscolo, minuscolo oppure lasciandoli inalterati. I possibili valori sono: Normal, Upper, Lower.
MaxLength int	Mediante questa proprietà è possibile specificare il numero massimo di caratteri che l'utente può inserire.

Tabella 3-13. Proprietà della classe TextBox.

PROPRIETÀ - TIPO	DESCRIZIONE
MultiLine bool	Definisce la natura – «singola linea» o «multilinea» – del TextBox. Un TextBox multilinea consente la visualizzazione e l'inserimento di più righe di testo.
Lines string[]	Consente, nel caso di un TextBox multilinea (vedi prop. MultiLine) di accedere al testo contenuto nel controllo come ad un array di stringhe.
ReadOnly bool	Impostando a true questa proprietà si rende il TextBox di «sola lettura», impedendo all'utente di modificarne il contenuto.
TextAlign HorizontalAlignment	Consente di specificare l'allineamento orizzontale del testo. I possibili valori sono: Center, Left, Right.
WordWrap bool	Questa proprietà funziona solo per i TextBox multilinea (vedi proprietà MultiLine). Impostata a true fa sì che il controllo esegua un ritorno a capo automatico quando il testo digitato dall'utente raggiunge il limite destro.

Segue un programma di esempio che illustra l'uso di alcune delle proprietà elencate.

[Applicazioni\Windows\exe\ProvaTextBox.exe]

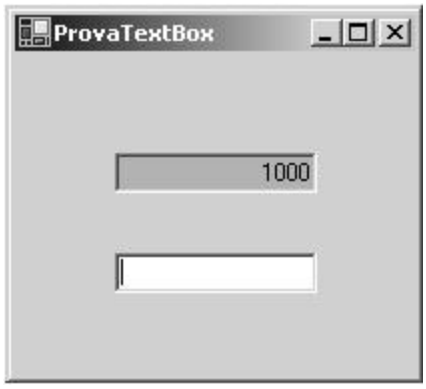
class MainForm ProvaTextBox 3.4

```
{
    TextBox txtProva1, txtProva2;
    public MainForm()
    {
        Text = "ProvaTextBox";
        StartPosition = FormStartPosition.CenterScreen;

        // primo TextBox: di sola visualizzazione
        txtProva1 = new TextBox();
        txtProva1.Location = new Point(50, 50);
        txtProva1.Text = "1000";
        txtProva1.TextAlign = HorizontalAlignment.Right;
        txtProva1.BackColor = Color.Aqua;
        txtProva1.ReadOnly = true;
        Controls.Add(txtProva1);

        // secondo TextBox
        txtProva2 = new TextBox();
        txtProva2.Location = new Point(50, 100);
```

```
            txtProva2.MaxLength= 5;
            txtProva2.CharacterCasing = CharacterCasing.Upper;
            Controls.Add(txtProva2);
        }
        public static void Main()
        {
            Application.Run(new MainForm());
        }
}
```

Sotto è mostrato l'output del programma

Figura 3-10. Output prodotto dal programma «ProvaTextBox».



Provando il programma, si nota che il valore true nella proprietà ReadOnly del primo TextBox ne impedisce la modifica, ma non la selezione; per questo motivo l'impiego di TextBox per la sola visualizzazione non è molto usuale, eccetto in quei casi nei quali si vuol dare all'utente la possibilità di eseguire operazioni di «copia&incolla» del contenuto del TextBox.

Eventi

L'unico evento della classe TextBox preso in considerazione è l'evento di sola notifica TextChanged, il quale viene sollevato in risposta a una qualsiasi modifica del testo contenuto nel TextBox (inserimento, cancellazione, modifica dei caratteri contenuti nella proprietà Text).

La gestione delle evento TextChanged è utile quando lo stato di alcuni controlli deve dipendere dal contenuto di uno o più TextBox. Un esempio di gestione di tale evento sarà presentato più avanti.

3.4.4 Classe «Button»

Il ruolo svolto dai controlli Button è del tutto evidente: essi consentono all'utente di confermare e/o dare l'avvio a determinate elaborazioni; ciò si ottiene gestendo l'evento Click sollevato dal controllo. Nella sua forma di base, l'aspetto di un Button è caratterizzato semplicemente da un testo che ne precisa la funzione.

Proprietà

Tabella 3-14. Proprietà della classe Button.

PROPRIETÀ - TIPO	DESCRIZIONE
FlatStyle FlatStyle	Definisce l'apparenza del bottone. I possibili valori sono: Flat: il bottone appare piatto (senza rilievo tridimensionale); Popup: il bottone appare piatto, ma diventa tridimensionale quando il mouse si trova sopra di esso; Standard: il bottone appare tridimensionale; System: l'apparenza è determinata dal sistema operativo.
Image Image	Consente di caratterizzare il bottone con un'immagine (generalmente un'icona). Un modo per definire il contenuto di tale proprietà e quello di specificare il file che contiene l'immagine mediante il metodo FromFile della classe Image: btnEsempio.Image = Image.FromFile(@"nomeicona.bmp");
ImageAlign ContentAlignment	Consente di impostare l'allineamento dell'immagine rispetto all'area del controllo. (Per i possibili valori, vedi proprietà TextAlign della classe Label.)
Text string	Contiene il testo da visualizzare. Se all'interno del testo è presente il carattere & («e» commerciale), il carattere che lo segue viene interpretato come «acceleratore» e cioè un tasto che premuto in combinazione con ALT determina il sollevamento dell'evento Click.
TextAlign ContentAlignment	Consente di impostare l'allineamento del testo rispetto all'area del controllo. (Per i possibili valori, vedi proprietà TextAlign della classe Label.)

Segue un programma di esempio che illustra l'uso di alcune delle proprietà elencate.

[Applicazioni\Windows\exe\ProvaButton.exe]

class MainForm ProvaButton 3.5

```
{
    Button btnEsempio;
    public MainForm()
    {
        Text = "ProvaButton";
        StartPosition = FormStartPosition.CenterScreen;

        btnEsempio = new Button();
        btnEsempio.Location = new Point(20, 20);
```

```
            btnEsempio.Location = new Size(100, 50);
            btnEsempio.Text = "&Clicca qui!";
            btnEsempio.FlatStyle = FlatStyle.Popup;
            btnEsempio.TextAlign = ContentAlignment.MiddleRight;
            btnEsempio.Image = Image.FromFile(@"immagini\mano.bmp");
            btnEsempio.ImageAlign = ContentAlignment.MiddleLeft;
            btnEsempio.Click += new EventHandler(btnEsempio_Click);
            Controls.Add(btnEsempio);
        }

        void btnEsempio_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Hai cliccato!");
        }

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

Figura 3-11. Output prodotto dal programma «ProvaButton».

Nota bene. Nell'invocazione del metodo FromFile della classe Image, il percorso del file contenente l 'immagine viene interpretato come relativo alla cartella nella quale si trova il programma eseguibile.

Eventi

Al di là di una implementazione particolarmente sofisticata della comunicazione con l'utente, l'unico evento da gestire è senz'altro l'evento Click.

3.5 «CONSISTENZA» DELL'INTERFACCIA

Il programma «MioPrimoGrado», il cui funzionamento ricalca quello di «PrimoGradoWin», presenta dei limiti nella gestione dell'interazione con l'utente; di questi saranno affrontati i seguenti due:

nulla impedisce all'utente di eseguire il calcolo della soluzione dell'equazione senza aver prima inserito i valori dei due coefficienti;

nulla impedisce all'utente di inserire caratteri qualsiasi, dunque anche lettere, nei TextBox relativi ai due coefficienti.

Entrambe le questioni riguardano il problema della «consistenza» dell'interfaccia. Brevemente[11]:

un'interfaccia è consistente quando è in grado di garantire (entro certi limiti) che le azioni e i dati inseriti dall'utente siano coerenti con la natura delle elaborazioni effettuate dal programma.

Di norma, un'interfaccia consistente la si progetta e non la si rende tale dopo averla realizzata, e impiegando strumenti – proprietà ed eventi – specifici, ma a questo livello ciò che ci interessa realmente è cominciare a mettere in pratica alcuni degli elementi acquisiti nei paragrafi precedenti allo scopo di rendere il funzionamento di «MioPrimoGrado» più vicino a quello di un programma realistico.

Data la precedente definizione, l'interfaccia di «MioPrimoGrado» non è consistente, poiché:

a) consente l'esecuzione del calcolo dell'equazione senza che siano stati inseriti i coefficienti;

b) consente l'esecuzione del calcolo dell'equazione a prescindere dal fatto che i valori inseriti siano effettivamente di natura numerica.

3.5.1 Verifica dell'esistenza dei dati

Un requisito fondamentale di qualsiasi interfaccia è quello di garantire che i dati siano stati effettivamente inseriti prima che il programma svolga qualsiasi elaborazione su di essi. Il problema può essere affrontato in due modi distinti, implementando cioè una forma di verifica «anticipata», oppure «posticipata». (In alcuni casi è appropriato implementarle entrambe).

Verifica posticipata dell'esistenza dei dati

Questa forma di verifica è implementata controllando, all'interno del metodo che gestisce l'elaborazione dei dati, se questi esistono davvero. Se la verifica dà esito negativo, l'elaborazione viene abortita e l'utente viene informato, in modo più o meno dettagliato, che i dati non rispettano i requisiti richiesti dal programma.

 [Applicazioni\Windows\exe\MioPrimoGrado3.exe]

[11] E' questa una definizione incompleta del concetto di «consistenza» di un interfaccia. Ma è utile per il tipo di problema che stiamo considerando.

«Implementazione verifica «posticipata» esistenza dei dati MioPrimoGrado3 3.6

```
void btnCalcola_Click(object sender, EventArgs e)
{
    // verifica dell'esistenza dei dati
    if (txtA.Text == "" || txtB.Text == "")
    {
        MessageBox.Show("Uno o entrambi i coefficienti non sono stati inseriti",
                        "MioPrimoGrado",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    // acquisisce i coefficienti dai due TextBox
    ...
}
```

L'implementazione si spiega da sé. Se uno o entrambi i TextBox sono vuoti viene visualizzata una message dialog di errore e successivamente viene terminato il metodo.

Verifica anticipata dell'esistenza dei dati

Questa forma di verifica è più sofisticata e sfrutta la natura ad eventi delle interfacce grafiche. Lo scopo è quello di impedire all'utente di dare l'avvio all'elaborazione fintantoché tutti i dati non sono stati inseriti; ciò si ottiene abilitando o disabilitando il bottone btnCalcolaX. Viene dunque a crearsi una relazione tra lo stato del bottone – abilitato o non abilitato – e il contenuto dei due TextBox, che può essere espressa così: btnCalcolaX è abilitato solo se txtA e txtB contengono i dati; altrimenti è disabilitato.

Una simile relazione può essere tradotta in pratica gestendo l'evento TextChanged sollevato dai due TextBox. Tale evento viene sollevato ogni qual volta viene modificato il contenuto del TextBox: ebbene, ogni qual volta avviene una tale modifica, di qualunque natura essa sia, è sufficiente verificare se il contenuto di entrambi i TextBox sia o no diverso da stringa nulla; se sì, la proprietà Enabled di btnCalcolaX viene impostata a true, altrimenti viene impostata a false.

Segue l'implementazione del gestore di evento:

«Implementazione verifica «anticipata» esistenza dei dati MioPrimoGrado4 3.7

```
void txtAeB_TextChanged(object sender, EventArgs e)
{
    btnCalcolaX.Enabled = (txtA.Text != "" && txtB.Text != "");
}
```

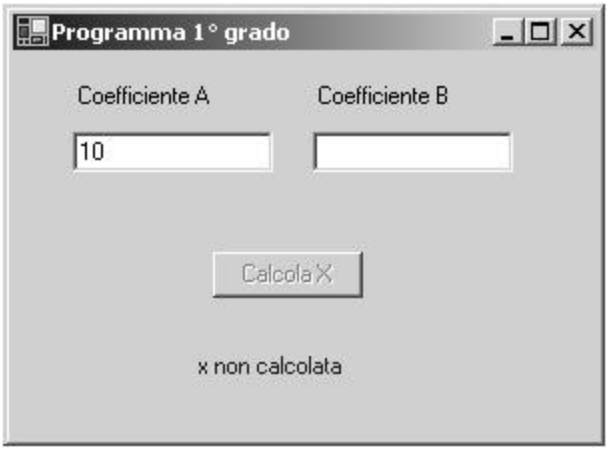
Naturalmente, perché il tutto funzioni come si deve, il gestore di evento dev'essere attaccato a entrambi i TextBox, in modo che qualsiasi modifica venga effettuata su uno qualsiasi di essi determini l'immediata e appropriata impostazione di btnCalcolaX.Enabled in relazione al loro contenuto. Segue il codice da aggiungere al costruttore della classe MainForm:

«Implementazione verifica «anticipata» esistenza dei dati MioPrimoGrado4 3.8

```
public MainForm()
{
    ...
    // creazione di txtA
    ...
    txtA.TextChanged += new EventHandler(txtAeB_TextChanged);
    ...
    // creazione di txtB
    ...
    txtB.TextChanged += new EventHandler(txtAeB_TextChanged);
    ...
}
```

In figura è rappresentata una fase dell'esecuzione del programma nella quale l'utente ha inserito il solo coefficiente A: il bottone è disabilitato.

Figura 3-12. Output del programma «MioPrimoGrado4».



La gestione della «verifica anticipata» non è ancora completa. Infatti, subito dopo il lancio del programma i due TextBox sono vuoti, ma il bottone è egualmente abilitato. Ciò avviene perché finché l'utente non modifica il contenuto dei TextBox, l'evento TextChanged non viene sollevato, il gestore dell'evento non viene eseguito e di conseguenza lo stato del bottone non viene aggiornato. Per risolvere questo problema è sufficiente impostare in modo appropriato lo stato iniziale del bottone:

```
public MainForm()
{
    ...
    btnCalcolaX.Enabled = false;
    ...
}
```

 [Applicazioni\Windows\exe\MioPrimoGrado4.exe]

3.5.2 Verifica della natura e del valore dei dati

Verificare la natura e il valore dei dati implica che l'avere inserito "qualcosa" da parte dell'utente non è un requisito sufficiente perché possa essere eseguita la parte elaborazione del programma; occorre anche che quel "qualcosa" sia coerente con la natura della elaborazione effettuata.

In questo senso, con il termine «verifica della natura» ci si riferisce a una verifica sul tipo dei dati inseriti: valori numerici, numeri telefonici, nominativi, codici fiscali, eccetera; con il termine «verifica del valore» ci si riferisce invece a una verifica sull'appartenenza dei dati a un insieme di valori ammissibili. Sarà presa in considerazione soltanto la prima modalità di verifica.

Verifica posticipata della natura dei dati

Verificare che il contenuto dei due TextBox sia effettivamente di natura numerica (senza che tale verifica produca un'interruzione del programma, come avviene eseguendo semplicemente i metodi di conversione della classe Convert) presuppone la conoscenza di elementi del linguaggio non ancora affrontati. Resta inteso che una verifica posticipata sulla natura dei dati implica anche, per definizione, una verifica sulla loro esistenza.

Verifica anticipata della natura dei dati

Questa forma di verifica può essere tradotta in pratica sfruttando gli eventi di tastiera sollevati dai controlli TextBox; essa ha lo scopo di impedire all'utente di inserire caratteri che non siano coerenti con la natura dei dati, nella fattispecie qualsiasi carattere non appartenente all'insieme «0-9, punto-decimale, segno-meno».

> In realtà , questa non rappresenta una garanzia molto "robusta", poiché nulla impedirebbe all'utente di inserire valori del tipo "0...10", "---100", "1-1-2.22", eccetera. Ma, lo ripetiamo, in questa fase lo scopo non è quello di raggiungere un alto grado di sofisticazione nella progettazione di interfacce, ma semplicemente di comprendere il meccanismo di gestione degli eventi sollevati dai controlli.

Tra gli eventi di tastiera, l'evento più appropriato da gestire è KeyPress. Segue l'implementazione del gestore di evento:

«Implementazione verifica «anticipata» natura dei dati MioPrimoGrado5 3.9

```
void txtAeB_KeyPress(object sender, KeyPressEventArgs e)
{
    char ch = e.KeyChar;
    if (ch == 8)            // tasto BACKSPACE
        return;

    if (char.IsDigit(ch) == false && ch != ',' && ch != '-')
        e.Handled = true;
```

Nel gestore di evento, dopo aver assegnato a una variabile ch il codice del tasto premuto (proprietà KeyChar del parametro informazioni evento), viene verificato se questo corrisponde al tasto BACKSPACE: in questo caso, infatti, il metodo deve terminare e lasciare che il controllo elabori il tasto. In tutti gli altri casi (tra i tasti che determinano il sollevamento dell'evento KeyPress) viene verificato se appartengono o meno all'insieme dei caratteri coerenti con la natura numerica dei dati. Se non è così, la proprietà Handled del parametro informazioni evento viene impostata a true, in pratica informando il controllo di non elaborare – e dunque di scartare – il carattere digitato.

Ci sono due osservazioni da fare:

per verificare se un carattere appartiene all'insieme '0' – '9' viene fatto uso del metodo IsDigit() esposto dal tipo char. Tale metodo ritorna il valore true se l'argomento rientra appunto nell'intervallo '0' – '9', altrimenti ritorna false.

il punto decimale, che nel codice sorgente è sempre rappresentato dal «carattere punto», nell'ambito dell'interfaccia è rappresentato dal «carattere virgola». Ciò dipende dalla «cultura» impostata a livello di sistema operativo. In Italia, per il punto decimale viene impiegata la virgola, mentre il punto viene utilizzato come separatore delle migliaia.

Anche in questo caso, occorre attaccare il gestore di evento ad entrambi i TextBox:

«Implementazione verifica «anticipata» esistenza dei dati MioPrimoGrado5 3.10

```
public MainForm()
{
    ...
    // creazione di txtA
    ...
    txtA.KeyPress += new KeyPressEventHandler(txtAeB_KeyPress);
    ...
    // creazione di txtB
    ...
    txtB.KeyPress += new KeyPressEventHandler(txtAeB_KeyPress);
    ...
}
```

[Applicazioni\Windows\exe\MioPrimoGrado5.exe]