APPROFONDIMENTO SUI CONTROLLI

Membri Comuni a più Classi di Controlli: Messa a Fuoco, Ordine di Tabulazione, proprietà TabIndex e metodo Focus

Molti *membri* (*proprietà, metodi o eventi*) sono disponibili in tutte o in molte Classi di Controlli e per questo rivestono particolare importanza.

Durante l'esecuzione di un Programma C#, sulla Form è possibile agire su *un solo controllo per volta*.

☞ Ad esempio, è impossibile far apparire il cursore lampeggiante in due caselle di testo contemporaneamente!

Il Controllo sul quale si agisce si dice che è **Messo a Fuoco** o che **Possiede il Fuoco**.

L'Utente può **Spostare la Messa a Fuoco** (o **Spostare il Fuoco**) utilizzando il *Mouse* oppure premendo il **tasto TAB** della tastiera: in quest'ultimo caso, ad ogni pressione del *Tasto TAB*, la *Messa a Fuoco si sposta ciclicamente da un controllo all'altro*, secondo un ordine chiamato **Ordine di Tabulazione**.

Ogni Controllo che può essere messo a fuoco, è dotato di un **proprietà TabIndex**: essa contiene un *numero d'ordine* che determina la *posizione* del controllo nella sequenza di spostamento del fuoco realizzabile con il tasto TAB.

Il programmatore, quindi, impostando il valore delle proprietà TabIndex dei vari controlli, può determinare l'Ordine di Tabulazione, ossia l'ordine in cui C# fa "scorrere" i controlli stessi con il tasto TAB.

✋ Quando il programma viene avviato, C# "*mette a fuoco*" automaticamente il controllo che ha in *TabIndex* il valore più piccolo. Generalmente si parte da 1 a salire ed è preferibile non inserire valori di TabIndex uguali in controlli diversi. Pur essendo possibile usare TabIndex da codice, essa però, viene quasi sempre impostata dal programmatore in modo interattivo nella Finestra Proprietà dell'ambiente di lavoro.

Il **metodo Focus( )** di un Controllo consente, da codice, di *spostare automaticamente il fuoco* sul controllo stesso.

☞ Se, da codice, vuoi "mettere a fuoco" una TextBox di nome *txtNumero*, basta eseguire l'istruzione: ***txtNumero.Focus( )*** Dopo l'esecuzione di questa istruzione, il "*cursore lampeggiante*" si posiziona automaticamente nella TextBox, proprio come se l'utente l'avesse selezionata manualmente con il mouse.

Membri Comuni a più Classi di Controlli: Visibilità, Attivazione, Posizionamento e Ridimensionamento

Tutti le Classi di Controlli sono dotate della **proprietà Visible** che consente di *rendere visibile o invisibile* il controllo, in fase di esecuzione. La *proprietà Visible* assume un valore di *tipo bool* (***true*** = *Visibile* / ***false*** = *Invisibile*).

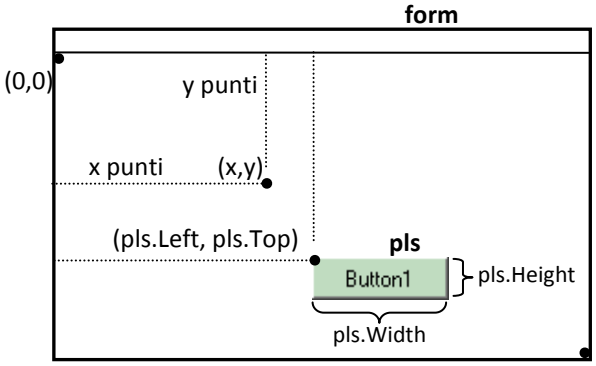
I Controlli che possono essere "*messe a fuoco*" sono tutti dotati della **proprietà Enabled**, anch'essa di *tipo bool*, che consente di *attivare/disattivare il controllo*: se disattivato, il controllo è visibile, ma l'utente *non può interagire con esso*.

Tutte la Classi di Controlli sono dotate di proprietà che determinano la **Posizione** del Controllo sulla Form.

La Form, come tutte le immagini digitali, è composta da una matrice di punti (pixels) e, ciascun punto è individuato tramite un **Coppia di Coordinate (x,y)**.

Nella Form, le Coordinate del punto **Origine**, ossia quello in alto a sinistra, sono **(0,0)**.

✋ La *cooridnata X aumenta* man mano che si considerano punti più *spostati a destra* e la *coordinata Y aumenta* man mano che si considerano punti più *spostati verso il basso*.



Le **proprietà Left e Top** di un Controllo, indicano rispettivamente la *Coordinata X (Left)* e la *Coordinata Y (Top)* del punto in alto a sinistra del controllo.

☞ Modificando il valore della *proprietà Left e Top* di un controllo, si *cambia la posizione* dell'oggetto sulla Form, e quindi se ne può provocare lo spostamento, anche da codice. Ad esempio: ***pls.Left = pls.Left + 20*** sposta *pls* di *20 pixels* a destra.

Le **proprietà Width e Height** di un Controllo, indicano rispettivamente *la Larghezza e l'Altezza*, espressa in punti (pixels) del controllo stesso.

☞ Anche le *proprietà Width e Height* possono essere modificate da codice (oltre che, in fase di progettazione, dalla finestra proprietà o ridimensionando manualmente il controllo) e, di conseguenza, si può ridimensionare il controllo in fase di esecuzione. Ad esempio, l'istruzione ***pls.Width = 300*** fa diventare *pls* largo esattamente *300 pixels*.

Anche la *Form,* indicabile con la *parola riservata* **this**, è dotata delle *proprietà Width e Height,* che ne indicano larghezza e altezza, *inclusi i bordi.*

Le **proprietà Left e Top della Form** (*this.Left* e *this.Top*) indicano *la posizione della Form rispetto all'intero schermo.*

Più precisamente le proprietà della Form *this.Left* e *this.Top* indicano la posizione dell'angolo in alto a sinistra della Form rispetto all'angolo in alto a sinistra dello schermo. Sono quindi utili per determinare la posizione della Form sullo schermo.

Membri Comuni a più Classi di Controlli: Testo e Colori

Il valore della **proprietà Text** di un Controllo determina il *Testo Visibile* nel controllo stesso.

Come sai bene, la *proprietà Text* viene spesso utilizzata (in lettura) per recuperare, da codice, il Dato che l'utente digita in una TextBox, ma può essere usata anche (in scrittura) per far apparire un risultato in una Label o per modificare il testo presente su un Button.

La **proprietà TextAlign** di un Controllo determina il modo in cui il Testo viene *Allineato* all'interno del controllo stesso. I possibili tipi di allineamento possibili sono *Sinistra, Destra, Centro*.

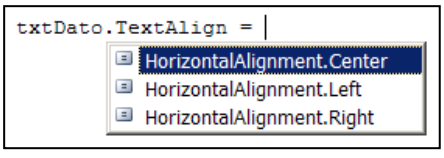
Spesso capita che una proprietà, anziché un valore *numerico* o *string*, possa assumere un *insieme di particolari valori "speciali"*. In questo caso si ricorre alle *Costanti Enumerate*…

Le **Costanti Enumerate** sono un Gruppo di Costanti (detto **Enumerazione**), ciascuna delle quali rappresenta uno dei possibili valori "speciali" che è possibile attribuire a una determinata proprietà. Ogni Enumerazione ha un *nome*, ed è possibile richiamare il valore desiderato con la solita notazione: *<enumerazione>.<valore>*

La *proprietà TextAlign* può assumere solo valori appartenenti all'*Enumerazione* chiamata **HorizontalAlignment**. Questa enumerazione (gruppo di costanti) include le costanti **Left, Right, Center**.

☞ Se vuoi centrare il testo all'interno di una textbox *txtDato*, devi agire sulla sua *proprietà TextAlign*, assegnandovi il "valore speciale" che indica l'allineamento centrato. A questo scopo userai l'enumerazione *HorizontalAlignment* e scriverai così:

txtDato.TextAlign = HorizontalAlignment .Center

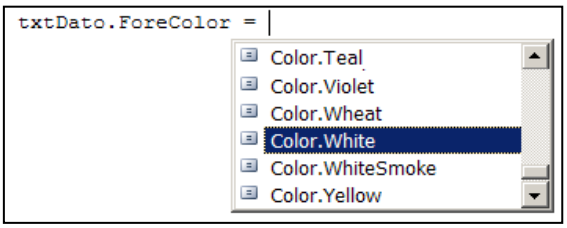


Nota come, mentre digiti il codice, la *funzione Intellisense* di VB ti aiuta, "elencandoti" automaticamente tutti le costanti previste dall'enumerazione HorizontalAlignment e quindi, tutti i possibili valori per la proprietà TextAlign.

Le **proprietà ForeColor e BackColor** permettono di impostare il *Colore di Scrittura del Testo* (ForeColor) e il *Colore del Fondo* (BackColor) di un Controllo. I colori sono disponibili con l'*enumerazione* **Color**.

☞ Se vuoi che il testo di *txtDato*, sia visualizzato in Bianco su fondo Nero, devi agire sulle *proprietà ForeColor e BackColor*, assegnandovi i "valori speciali" resi disponibili dall'enumerazione *Color …*

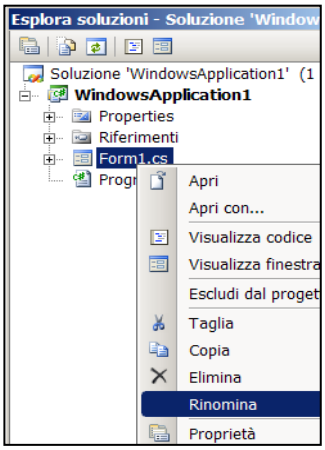
txtDato.ForeColor = Color.White
txtDato.BackColor = Color.Black



Label: altre Proprietà, Metodi ed Eventi utili

La **proprietà AutoSize della classe *Label*** è di *tipo bool* e permette di decidere se la *dimensione della Label è modificabile o automatica*.

Se impostata a *true*, la *proprietà AutoSize* di una Label, fa sì che la dimensione della Label stessa sia *automaticamente adattata al testo* e, quindi, "bloccata"; se impostata a *false,* invece, la sua dimensione diventa "*liberamente modificabile*", sia in fase di progettazione che tramite codice.



Form: altre Proprietà, Metodi ed Eventi utili

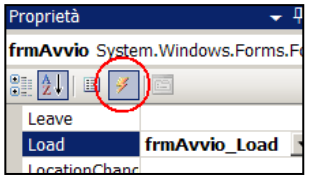
Se si desidera **modificare il Nome della Form** (ossia la *proprietà Name* della Form) è consigliabile agire sulla *finestra Esplora Soluzioni* e modificarne il *Nome del File* con il **comando *Rinomina***. Nel rinominare il File è necessario mantenere l'estensione **.cs** .

☞ Supponi di voler rinominare la tua Form, modificandone il nome da "***Form1***" a "***frmAvvio***": richiami la *finestra Esplora Soluzioni* dal *menù Visualizza*; fai click-destro su ***Form1.cs***; selezioni il *comando Rinomina*; digiti "***frmAvvio.cs***"; al quesito successivo, confermi.

In tal modo, C# modifica in sol colpo: (a) il **Nome del File** in cui la Form viene salvata su disco, che diventa ***frmAvvio.cs***; (b) la **proprietà Name** della Form, che assume il valore ***frmAvvio***; (c) tutte le **Righe di Codice** che contengono *Form1* si aggiornano con ***frmAvvio***.

La Form rende disponibile lo speciale **evento Load** che viene *eseguito solo una volta*, all'avvio del programma.

L'evento Load è il punto ideale in cui digitare codice per eseguire *azioni di inizializzazione o preparazione*, da realizzare "prima" che l'utente inizi ad operare con la Form stessa (es. posizionare il cursore su una specifica textbox, impostare qualche proprietà dei controlli, ecc.)

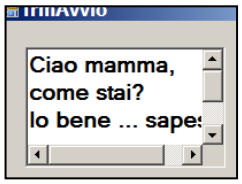


Nel codice, **per generare l'evento Load** della Form, è necessario selezionare la Form, accedere all'elenco dei suoi eventi e fare doppio-click sulla voce Load.

TextBox: altre Proprietà, Metodi ed Eventi utili

La **proprietà MaxLength della classe TextBox** consente di stabilire *la quantità massima di caratteri* che la TextBox può accettare. I valori ammessi sono *numeri interi* da 1 a salire.

La limitazione *non ha effetto* per le modifiche apportate alla proprietà Text *da codice*.

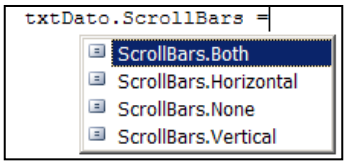


Un controllo di *classe TextBox* può gestire anche un input costituito da **Più Righe di Testo**.

Per attivare questa funzionalità basta *impostare a TRUE* la **proprietà booleana MultiLine della classe TextBox**: fatto questo diventa possibile "*tornare a capo*" nella casella e digitare testo composto da più righe. Diventa possibile anche regolare la *dimensione verticale* della TextBox, oltre a quella orizzontale.

E' possibile accedere alle *singole righe* del testo presente nella TextBox multilinea, con la **proprietà indicizzata Lines**, da usare con la solita notazione: <*nome-textbox*>**.Lines[**<*indice*>**]** dove <*indice*> è il *numero della riga* desiderata (zero indica la prima). Con la proprietà <*nome-textbox*>**.Lines.Lenght** si ottiene il numero totale di righe presenti nel testo.

La **proprietà ScrollBars della classe TextBox** consente scegliere quali **Barre di Scorrimento** far apparire: i valori consentiti sono quelli dell'*enumerazione* ***ScrollBars*****:** *None, Horizontal, Vertical, Both (both significa "entrambe")*.



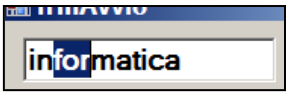
La **proprietà booleana WordWrap della TextBox** consente di attivare/disattivare il *Ritorno a Capo Automatico* del testo nella TextBox.

Il **metodo SelectAll** *della classe TextBox* consente di *Selezionare automaticamente Tutto il Testo* nella TextBox stessa.

Il **metodo Select** *della classe TextBox* consente di *Selezionare automaticamente una Parte del Testo* contenuto nella TextBox stessa. L'utilizzo è il seguente:

<*nome-textbox*>**.Select (** <*posizione-primo-carattere*>**,** <*numero-di-caratteri*> **)**

☞ Se in una textbox *txtDato* c'è la stringa "*informatica*", l'istruzione **txtDato.Select(2, 3)** provoca l'evidenziazione dei caratteri "*for*" (ossia dalla posizione 2, evidenzia 3 caratteri). La posizione è, come al solito, in base-zero.



Attenzione: la selezione sarà attuabile e sarà visibile solo mentre l'oggetto è "a fuoco" !

L'**evento Leave** su un Controllo si verifica ogni volta che *la messa a fuoco "abbandona"* il Controllo stesso.

L'*evento Leave* è utile per **verificare se il dato digitato in una TextBox è *accettabile***, prima di acquisirlo in una variabile. Se l'utente "*prova ad abbandonare*" la TextBox, viene eseguito il codice dell'*evento Leave* che ne può verificare la validità.

In una casella di testo *txtVoto* è necessario accettare solo valori compresi fra 1 e 10. L'*evento Leave* potrebbe essere:

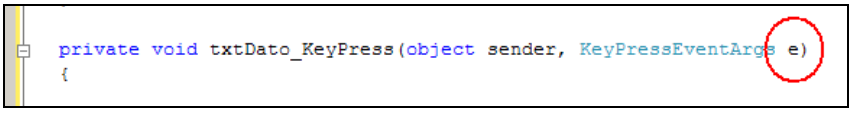
```
private void txtVoto_Leave(object sender, EventArgs e)
{
        int Voto = Convert.ToInt16(txtVoto.Text)

        if  ( ( Voto < 1 ) || ( Voto > 10 ) )          // se il voto non è compreso fra 1 e 10 …
        {       MessageBox.Show("Digita un voto valido.");   // visualizza un messaggio di errore …
                txtNum.Focus();                        // riposizionati sulla textbox …
                txtNum.SelectAll();                    // preseleziona tutto il testo, per la correzione …
        }
}
```

Eventi di Tastiera: KeyPress, KeyDown e oggetto "e"

L'**evento KeyPress** si verifica *sull'oggetto "a fuoco"* quando l'utente *preme un qualsiasi Tasto della Tastiera*.

Quando si verifica un evento, *C#* rende disponibile uno speciale **oggetto "e"**, che fornisce *informazioni aggiuntive sull'evento stesso*. L'*oggetto "e"* esiste solo durante l'esecuzione del codice di un evento.

Avrai probabilmente notato che l'oggetto "e" compare nella "testata" degli eventi generata automaticamente da C#:



La **proprietà e.KeyChar** (disponibile solo nell'evento KeyPress) fornisce *il carattere che l'utente ha digitato*.

Tale proprietà può essere usata per **riconoscere il singolo carattere digitato** e agire di conseguenza.

In più, assegnando alla **proprietà e.Handled** il valore TRUE, **si annulla la digitazione del carattere**.

Supponi di voler "*filtrare*" i caratteri che l'utente digita in una textbox *txtPorola*, limitandoli alle sole lettere maiuscole. Ovviamente accetterai anche il carattere 

(codice ASCII = 8), mentre tutti gli altri caratteri dovranno essere ignorati.
Devi programmare l'evento KeyPress del controllo txtParola: il codice deve accedere alla *proprietà e.KeyChar* per riconoscere il *carattere digitato* e, in caso non sia uno dei caratteri "ammessi", *annullare la digitazione* con *e.Handled = true.*

```
if ( ( (e.KeyChar < 'A') || (e.KeyChar > 'Z') )  &&     // … NON è lettera maiuscola   oppure
   (e.KeyChar != Convert.ToChar(8) ) )                  // … NON è il BACKSPACE
        e.Handled = true;                               // … allora Annulla la Digitazione
```

Anche l'**evento KeyDown** si verifica *sull'oggetto "a fuoco"* quando l'utente *preme un Tasto della Tastiera*, ma in esso, **l'Oggetto "e" restituisce il *"Tasto"* anziché il *"Carattere".***

Questo consente di rilevare anche la pressione di tasti che *non corrispondono a un carattere*, tasti come MAIUSCOLO, CTRL, ALT, Frecce, Stampa, Tasti Funzione, ecc.).

La **proprietà e.KeyCode** (disponibile solo nell'evento KeyDown) fornisce il *Codice di Tastiera del Tasto premuto* (qualunque sia il tasto premuto, anche quelli che non generano un carattere).

I possibili valori per *e.KeyCode* sono espressi con l'**enumerazione Keys**: oltre a tutti i tasti associati a caratteri, sono presenti anche le costanti per i tasti speciali: **Up**, **Down**, **Left**, **Right** (frecce), **Ctrl**, **Alt**, **Shift**, **F1**, **F2**, ecc.

L'*evento KeyDown* e la *proprietà e.KeyCode* si usano quando è richiesto un input da tastiera per *impartire comandi* più che per *digitare testo* ... ad esempio, in semplici videogames …

Normalmente il Tasto premuto dall'utente viene *"intercettato" dall'oggetto che è a fuoco* ed è su di esso che si verificano eventi quali *KeyPress* o *KeyDown*.

Nel caso in cui *non sia presente nessun controllo che possa essere "messo a fuoco"*, allora *è la Form a processare la pressione del tasto* e **gli eventi KeyPress o KeyDown si verificano per l'oggetto Form**.