C# Applicazioni Windows

Capitolo 4 – Migliorare la comunicazione con l'utente

Gestire collezioni di dati

- Controlli «ListBox» e «ComboBox»

Bottoni per la selezione tra più opzioni

- «RadioButton»
- «CheckBox»

Raggruppare i controlli

- «GroupBox» e «Panel»

Gestire le immagini: «PictureBox»

PREMESSA

Nonostante la potenza – nella visualizzazione e nella comunicazione – delle interfacce grafiche, l'impiego dei soli controlli Label e TextBox per la visualizzazione e l'acquisizione dei dati non consente di implementare un modello di comunicazione particolarmente più sofisticato rispetto a quello che si può ottenere attraverso l'interfaccia di una «Applicazione Console». Ciò dipende dalla natura dei controlli in questione, ognuno dei quali consente fondamentalmente di visualizzare e acquisire un solo valore per volta, in entrambi i casi di natura testuale.

> Parziale eccezione è rappresentata dai TextBox multilinea, i quali consentono all'utente di inserire più righe di testo e dunque più valori. Resta il fatto che un TextBox multilinea non rappresenta il tipo di controllo più appropriato per gestire una collezione di dati.

Ciò che è necessario per aumentare la qualità della comunicazione con l'utente è la possibilità di:

- gestire collezioni di dati, offrendo all'utente la possibilità di scegliere uno tra un insieme di valori e/o di aggiungere e togliere valori alla collezione;
- offrire all'utente la possibilità di selezionare una o più tra un insieme di alternative;
- visualizzare elementi grafici (immagini, disegni, sfondi) per migliorare l'estetica e la "comunicatività" dell'interfaccia.

4.1 GESTIRE COLLEZIONI DI DATI

Due tipi di controllo, ListBox e ComboBox, consentono di gestire collezioni di dati unidimensionali. (Un altro tipo di controllo, DataGrid, consente gestire dati in forma tabellare). Con il termine "gestire" si intende che tali controlli consentono di:

visualizzare una lista di valori di varia natura (numerica, testuale, eccetera);

aggiungere e togliere un valore o un insieme di valori;

cancellare tutti i valori della lista;

"popolare" la lista di valori mediante l'assegnazione di un vettore.

eseguire ricerche;

mantenere la lista ordinata;

Entrambi i tipi di controllo condividono queste e altre caratteristiche. Il tipo ComboBox, inoltre, aggiunge a queste le funzionalità di un TextBox, e dunque "combina" le caratteristiche di due tipi di controlli (ComboBox sta appunto per «Casella combinata»).

4.2 CLASSE «ListBox»

I ListBox hanno la caratteristica di mantenere internamente una collezione dinamica di elementi, ai quali si può accedere usando la stessa sintassi impiegata per i vettori. L'uso più comune di tali controlli prevede che l'utente selezioni con il mouse uno o più elementi dalla lista, determinando quindi le successive elaborazioni del programma.

Proprietà

Tabella 4-1. Proprietà della classe ListBox.

PROPRIETÀ - TIPO	DESCRIZIONE
DataSource object	Assegnando una oggetto vettore a questa proprietà è possibile, con una sola istruzione, popolare il ListBox con i valori contenuti nel vettore.
Items object[]	Items è la proprietà attraverso la quale si può accedere agli elementi della lista. Essa espone a sua volta proprietà e metodi attraverso i quali aggiungere e togliere elementi, conoscere il numero degli elementi, eccetera: Items.Count: memorizza il numero degli elementi della lista; Items.Add(object): aggiunge un elemento alla lista; Items.AddRange(object[]): aggiunge una sequenza di elementi; Items.RemoveAt(int): rimuove un elemento dalla lista; Items.Clear(): rimuove tutti gli elementi dalla lista.
SelectedIndex int	Indice dell'elemento correntemente selezionato. (Tale elemento viene di norma visualizzato in bianco su blu.) Se è non selezionato alcun elemento, SelectedIndex vale –1.
SelectedItem object	Riferimento all'elemento correntemente selezionato. Il riferimento è di tipo object, dunque, perché possa essere assegnato a una variabile, occorre che sia eseguita l'appropriata operazione di cast.
SelectionMode SelectionMode	Definisce il tipo di selezione ammissibile: None: non è possibile selezionare alcun elemento; One: è possibile selezionare un solo elemento per volta; MultiSimple: è possibile selezionare più elementi; MultiExtended: è possibile selezionare più elementi; l'utente può usare i tasti CTRL e SHIFT e tasti freccia per eseguire la selezione.
Sorted bool	Indica se la lista debba o meno essere mantenuta ordinata. Il valore di default è false (lista non ordinata).

Eventi

Del nutrito insieme di eventi pubblicato dalla classe ListBox vengono comunemente gestiti quelli che rispondono alla selezione di un elemento da parte dell'utente.

Tabella 4-2. Eventi della classe ListBox. (*)

EVENTO	DESCRIZIONE
Click e DoubleClick	Entrambi vengono sollevati dopo che l'utente ha cliccato (o eseguito un doppio clic) sull'area del ListBox. Se il clic (o il doppio clic) avviene su un elemento questo diventa selezionato prima che l'evento sia sollevato.
SelectedIndexChanged	Viene sollevato dopo che è stato selezionato un elemento diverso da quello corrente.

(*) Gli eventi sono di sola notifica e quindi sono caratterizzati dalla delega EventHandler e dalla classe informazioni EventArgs.

4.2.1 Popolare un «ListBox»

Con il verbo "popolare" si intende aggiungere gli elementi alla lista mantenuta dal controllo. Nel suo impiego più comune un ListBox viene inizialmente popolato con una lista di dati dei quali l'utente potrà in seguito selezionare quello desiderato in relazione a una determinata operazione.

E' possibile popolare un ListBox attraverso due modalità distinte:

assegnando un vettore alla proprietà DataSource;

aggiungendo elementi attraverso i metodi Items.Add() e Items.AddRange().

4.2.2 Popolare un «ListBox» attraverso l metodi «Add()» e «AddRange()»

Il metodo Add() della proprietà Items può essere invocato per aggiungere un elemento al ListBox. Ad esempio, dato come presupposto che sia stato già creato e aggiunto all'interfaccia il ListBox di nome lboNomiFamosi, il seguente codice aggiunge ad esso tre elementi:

```
lboNomiFamosi.Items.Add("Hitler");
lboNomiFamosi.Items.Add("Stalin");
lboNomiFamosi.Items.Add("Mussolini");
```

L'argomento del metodo Add() può essere di qualsiasi tipo e ciò implica che la lista può essere composta da elementi di tipo diverso:

```
lboNomiFamosi.Items.Add("Stalin");          // aggiunge una stringa
lboNomiFamosi.Items.Add("Mussolini");       // aggiunge una stringa
lboNomiFamosi.Items.Add(200.2);             // aggiunge un double
```

Ovviamente, il precedente non rappresenta un caso molto comune, né desiderabile.

Il funzionamento del metodo AddRange() è analogo a quello di Add() con la differenza che il primo è in grado di aggiungere più elementi contemporaneamente; l'argomento del metodo è infatti rappresentato da un vettore di object. Ad esempio:

```
string[] filosofi = {"Socrate", "Platone", "Kant", "Popper");
lboNomiFamosi.Items.AddRange(filosofi);
```

Un limite di questo metodo è dovuto al fatto che gli elementi del vettore devono essere di tipo riferimento. E' dunque sbagliato scrivere:

```
int[] altezze = {180, 190, 170);
lboAltezze.Items.AddRange(altezze);
```

> Questo requisito non riguarda il metodo AddRange() ma il linguaggio. Infatti un vettore di object è compatibile con qualsiasi altro vettore i cui elementi siano di tipo riferimento, (nella fattispecie string) mentre non è compatibile con un vettore di elementi di tipo valore (e quindi int, char, double, bool, eccetera).

Per questo motivo, se si vuole popolare il ListBox con un vettore di elementi di tipo valore occorre iterare attraverso il vettore e aggiungere ogni elemento con Add():

```
int[] altezze = {180, 190, 170);
foreach(altezza in altezze)
    lboAltezze.Items.Add(altezza);
```

4.2.3 Popolare un «ListBox» attraverso la proprietà «DataSource»

Si ottiene assegnando un vettore, di qualsiasi tipo, alla proprietà in questione; ciò fa sì che gli eventuali precedenti elementi siano rimossi. Ad esempio, presupposto che sia stato già creato e aggiunto all'interfaccia il ListBox di nome lboNomiFamosi, il seguente codice lo popola con un vettore di stringhe:

```
string[] nomiFisici = {"Einstein", "Fermi", "Hawking", "Bohr");
...
lboNomiFamosi.DataSource = nomiFisici;
```

Dopo la precedente assegnazione, comunque, è proibito aggiungere, togliere o modificare elementi del ListBox. Per rendere questo possibile è necessario prima assegnare null alla proprietà DataSource, come viene fatto nel seguente codice:

```
lboNomiFamosi.DataSource = null;                    // ora posso modificare la lista
...
lboNomiFamosi.Items[0] = "Plank";                   // modifica del primo elemento
```

E' importante comprendere che assegnare null a DataSource non determina la cancellazione degli elementi esistenti, ma soltanto la possibilità che questi possono essere modificati, aggiunti o eliminati.

Un altro aspetto importante riguarda la possibilità di modificare gli elementi del ListBox agendo sul vettore usato per popolarlo. Semplicemente, ci ò non produce alcun effetto:

```
string[] nomiFisici = {"Einstein", "Fermi", "Hawking", " Bohr ");
...
lboNomiFamosi.DataSource = nomiFisici;
...
nomiFisici[0] = "Plank";                    // Il ListBox resta invariato;
```

E nemmeno il seguente codice produce effetti sul ListBox:

```
string[] nomiFisici = {"Einstein", "Fermi", "Hawking", "Bohr");
...
lboNomiFamosi.DataSource = nomiFisici;
...
nomiFisici[0] = "Plank";                    // modifica del primo elemento del vettore
lboNomiFamosi.DataSource = nomiFisici;              // riassegnazione del vettore
```

Diverso è il caso in cui viene assegnato un altro vettore alla proprietà DataSource:

```
...
string[] nomiMatematici = {"Gauss", "Poincarè", "Hilbert"};
lboNomiFamosi.DataSource = nomiMatematici;                  // ListBox ripopolato
```

In questo caso, infatti, il ListBox viene ripopolato con gli elementi del nuovo vettore. D'altra parte è possibile ripopolare un ListBox usando lo stesso vettore (modificato o meno che sia) purché prima di farlo venga assegnato null a DataSource:

```
string[] nomiFisici = {"Einstein", "Fermi", "Hawking", "Bohr");
...
lboNomiFamosi.DataSource = nomiFisici;
...
nomiFisici[0] = "Plank";                    // modifica del primo elemento del vettore
lboNomiFamosi.DataSource = null;
lboNomiFamosi.DataSource = nomiFisici;              // ListBox ripopolato
```

Infine, diversamente dal metodo AddRange(), attraverso la proprietà DataSource è possibile popolare un ListBox anche con vettori i cui elementi siano di tipo valore:

```
int[] altezze = {180, 190, 170);
lboAltezze.DataSource = altezze;                    // ok
```

4.2.4 Accesso agli elementi di un «ListBox»

Gli elementi della lista mantenuta da un ListBox sono di tipo object; ciò fa si che tale controllo possa essere popolato con valori di qualsiasi natura, addirittura anche con valori di tipo diverso tra loro. D'altra parte, quando si accede a tali elementi, ad esempio in risposta alla selezione da parte dell'utente, occorre di norma applicare l'operatore di cast per convertirli nel tipo appropriato.

Ad esempio, il seguente codice scandisce gli elementi di un ListBox e li copia in un vettore di stringhe (il ListBox si suppone essere già stato popolato in precedenza):

```
string[] grandiPoeti = new string[lboNomiFamosi.Items.Count];
for (int i = 0; i < lboNomiFamosi.Items.Count; i++)
```

```
    grandiPoeti[i] = (string) lboNomiFamosi.Items[i]; // uso del cast
```

Ovviamente, tutto ciò funziona se gli elementi del ListBox sono effettivamente delle stringhe. Non è questo il caso del seguente codice:

```
int[] altezze = {180, 190, 170);
lboNomiFamosi.DataSource = altezze;
...
string[] grandiPoeti = new string[lboNomiFamosi.Items.Count];
for (int i = 0; i < lboNomiFamosi.Items.Count; i++)
    grandiPoeti[i] = (string) lboNomiFamosi.Items[i]; // uso errato del cast
```

In questo caso il ListBox contiene una collezione di interi e dunque il tentativo di cast al tipo string produce un errore di esecuzione.

4.2.5 Uso della proprietà «Text»

Anche il controllo ListBox definisce la proprietà Text, della quale fa un uso non completamente convenzionale. Mediante essa è possibile ottenere una rappresentazione stringa dell'elemento attualmente selezionato. In alcune situazioni, dunque, può essere usata come sostituto della proprietà SelectedItem. Ad esempio, il seguente gestore di evento :

```
void lboNomiFamosi_SelectedIndexChanged(object sender, EventArgs e)
{
    MessageBox.Show(lboNomiFamosi.Text);
}
```

mostra in una message dialog l'elemento selezionato di un ListBox che si suppone già creato e aggiunto all'interfaccia.

La proprietà Text può essere usata anche per modificare l'elemento attualmente selezionato. Ciò si ottiene assegnando alla proprietà il valore in formato stringa dell'elemento che si vuole selezionare. Ad esempio, si supponga di aver popolato il ListBox con il seguente codice:

```
string[] nomiFisici = {"Einstein", "Fermi", "Hawking", "Bohr");
lboNomiFamosi.DataSource = nomiFisici;
```

E si supponga inoltre che "Einstein" sia l'elemento selezionato. Assegnando alla proprietà Text il valore "Fermi":

```
lboNomiFamosi.Text = "Fermi";
```

si ottiene il risultato di selezionare il secondo elemento del ListBox.

Nel caso in cui nessun elemento del ListBox abbia una rappresentazione stringa equivalente a quella assegnata alla proprietà Text, l'elemento selezionato resta quello attuale. Ad esempio, la seguente istruzione:

```
lboNomiFamosi.Text = "Plank";
```

lascia la situazione invariata, poiché non esiste un elemento di valore "Plank" nel ListBox.

4.2.6 Uso del controllo «ListBox»

Viene proposto un esempio d'uso molto banale; il programma si limita, dopo aver creato e popolato un ListBox, a gestire l'evento SelectedIndexChanged, impostando il contenuto di una Label con il valore dell'elemento correntemente selezionato.

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

[Applicazioni\Windows\exe\ProvaListBox.exe] 

class MainForm «ProvaListBox» 4.1

```
{
    Label lblPoetaSelezionato;
    ListBox lboGrandiPoeti;
    string[] grandiPoeti = {"ShakeSpeare", "Dante", "Keats", "Leopardi"}

    public MainForm()
    {
        Text = "ProvaListBox";
        StartPosition = FormStartPosition.CenterScreen;

        // ListBox
        lboGrandiPoeti = new ListBox();
        lboGrandiPoeti.Location = new Point(20, 20);
        lboGrandiPoeti.Size = new Size(150, 150);
        lboGrandiPoeti.DataSource = grandiPoeti;
        lboGrandiPoeti.SelectedIndexChanged += new
                                    EventHandler(lboGrandiPoeti_SelectedIndexChanged);
        Controls.Add(lboGrandiPoeti);

        // Label
        lblPoetaSelezionato = new Label();
        lblPoetaSelezionato.Location = new Point(20, 180);
        Controls.Add(lblPoetaSelezionato);
    }
    void lboGrandiPoeti_SelectedIndexChanged(object sender, EventArgs e)
    {
        lblPoetaSelezionato.Text = (string) lboGrandiPoeti.SelectedItem;
    }

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

L'esecuzione del programma produce il seguente output:

Figura 4-1. Output prodotto dal programma «ProvaListBox».

4.3 CLASSE «ComboBox»

I ComboBox combinano molte caratteristiche di un ListBox con quelle di un TextBox. Il controllo può dunque comunicare con l'utente sia attraverso il mouse (caratteristica tipica del ListBox) sia attraverso la tastiera (caratteristica tipica del TextBox). Tipicamente, l'utente può:

cliccare sulla freccia per visualizzare «l'elenco a discesa» (la lista degli elementi) e quindi selezionare un elemento; l'elemento selezionato viene visualizzato nella casella;

digitare all'interno della casella.

Proprietà

Seguono le proprietà che si aggiungono a quelle che la classe ComboBox condivide con la classe ListBox.

Tabella 4-3. Proprietà della classe ComboBox.

PROPRIETÀ - TIPO	DESCRIZIONE
DropDownStyle ComboBoxStyle	Questa proprietà caratterizza il comportamento del ComboBox e dunque la particolare funzione per il quale viene inserito nell'interfaccia. Possibili valori sono: Simple: L'utente può digitare nella casella oppure selezionare un elemento dall'elenco a discesa, il quale è sempre visibile. (Dunque non è presente la freccia per visualizzarlo). DropDown: L'utente può digitare nella casella oppure selezionare un elemento dall'elenco a discesa, il quale viene visualizzato cliccando sulla freccia. DropDownList: L'utente non può digitare nella casella e dunque può soltanto selezionare un elemento dall'elenco a discesa, il quale viene visualizzato cliccando sulla freccia.
MaxDropDownItems int	Massimo numero di elementi visualizzati contemporaneamente nell'elenco a discesa.
MaxLength e Text	Entrambe svolgono la stessa funzione delle proprietà omologhe esposte dalla classe TextBox.

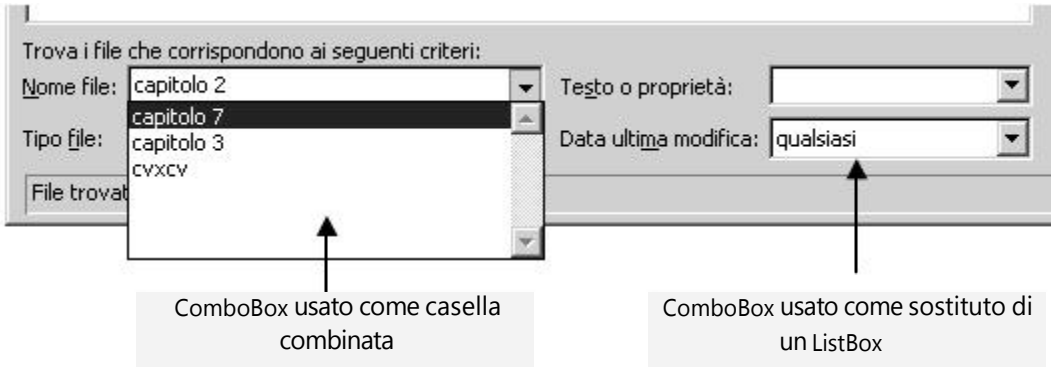
Eventi

Gli eventi pubblicati dalla classe ComboBox sono sostanzialmente gli stessi delle classi ListBox e TextBox. Quelli gestiti più comunemente sono: TextChanged, Click, DoubleClick, SelectedIndexChanged.

4.3.1 Uso del controllo «ComboBox»

Un primo impiego dei ComboBox è quello di sostituto dei ListBox laddove è richiesta la possibilità di selezionare un elemento da una lista ma lo spazio nel form non è sufficiente perché la lista sia costantemente visibile. L'impiego classico prevede invece che l'utente possa inserire un valore da tastiera oppure selezionarne uno tra quelli disponibili nella lista. In ogni caso, si accede all'elemento selezionato, o al testo digitato, attraverso la proprietà Text, analogamente a quanto avviene con i controlli TextBox.

Figura 4-2. Esempio d'impiego di controlli ComboBox nella dialog di apertura file in Word 97.



Di seguito viene riproposto il programma precedente, nel quale il ComboBox è impiegato per consentire la sola selezione di uno degli elementi della lista.

[Applicazioni\Windows\exe\ProvaComboBox.exe]

class MainForm «ProvaComboBox» 4.2

```
{
    Label lblPoetaSelezionato;
    ComboBox cboGrandiPoeti;
    string[] grandiPoeti = {"ShakeSpeare", "Dante", "Keats", "Leopardi"}
    public MainForm()
    {
        Text = "ProvaComboBox";
        StartPosition = FormStartPosition.CenterScreen;

        // ComboBox
        cboGrandiPoeti = new ComboBox();
        cboGrandiPoeti.Location = new Point(20, 20);
        cboGrandiPoeti.DataSource = grandiPoeti;
```

```
        cboGrandiPoeti.DropDownStyle = ComboBoxStyle. DropDownList;
        cboGrandiPoeti.SelectedIndexChanged += new
                                    EventHandler(cboGrandiPoeti_SelectedIndexChanged);
        Controls.Add(cboGrandiPoeti);

        // Label
        lblPoetaSelezionato = new Label();
        lblPoetaSelezionato.Location = new Point(150, 20);
        Controls.Add(lblPoetaSelezionato);
    }


    void cboGrandiPoeti_SelectedIndexChanged(object sender, EventArgs e)
    {
        lblPoetaSelezionato.Text = cboGrandiPoeti.Text;
    }

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

L'esecuzione del programma produce il seguente output:

Figura 4-3. Output prodotto dal programma «ProvaComboBox».

4.4 «RadioButton» E «CheckBox»

I controlli ListBox e ComboBox migliorano il livello di comunicazione con l'utente, fornendo a quest'ultimo un nuovo modello per l'inserimento dei dati. I controlli di tipo RadioButton e CheckBox arricchiscono ulteriormente questo modello, consentendo all'utente di selezionare una o più tra un insieme di opzioni disponibili.

4.4.1 Classe «RadioButton»

Un RadioButton svolge in pratica la funzione di un interruttore, il quale può trovarsi soltanto nello stato «accesso» oppure nello stato «spento»; analogamente, un RadioButton può assumere due soli stati possibili: cheked o uncheked («marcato» o «non marcato»).

Un RadioButton funziona sempre in collaborazione con altri RadioButton, insieme ai quali definisce la lista dei possibili stati che può assumere un determinato valore; in ogni momento uno solo tra i RadioButton può essere checked, determinando dunque lo stato del valore in questione.

Per questo motivo, bench é non rappresenti in sé un errore formale, inserire nel form un solo RadioButton è semplicemente privo di senso.

Proprietà

La proprietà più importante di un RadioButton è la proprietà Checked, la quale ne memorizza lo stato: «marcato» (true), «non marcato» (false).

Eventi

Di norma non è necessario gestire gli eventi sollevati dai RadioButton, poiché essi elaborano autonomamente le azioni dell'utente, impostando in modo appropriato il proprio stato, «marcato» o «non marcato», in relazione allo stato degli altri RadioButton. E' invece necessario gestire gli eventi di sola notifica Click o CheckedChanged se si desidera che lo stato di uno o più controlli venga aggiornato immediatamente in relazione al cambiamento dello stato del RadioButton in questione.

4.4.2 Uso del controllo «RadioButton»

Un insieme di RadioButton consente di definire la lista dei possibili valori di un'opzione, dei quali soltanto uno può essere selezionato in un dato momento. Diversamente dalla lista di elementi mantenuta da un ListBox o un ComboBox, il numero dei valori è fisso e corrisponde al numero di RadioButton.[12]

Il programma che segue ipotizza la richiesta di dati per la prenotazione di un volo aereo. Nella fattispecie viene chiesto all'utente di specificare il nome, mediante un TextBox, e la classe – prima classe o classe economica –, mediante due RadioButton.

[12] Ciò non è propriamente esatto, poiché nulla impedisce di aggiungere (o togliere) RadioButton in risposta a una qualche elaborazione o alle azioni dell'utente. Di fatto, questo rappresenta un impiego del tutto inappropriato dei RadioButton, che si prestano invece alla gestione di un elenco fisso di valori.

[Applicazioni\Windows\exe\ProvaRadioButton.exe] 

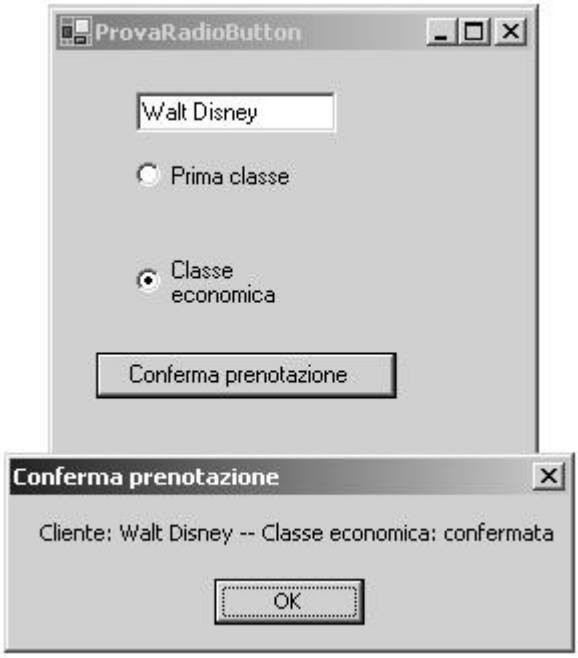
class MainForm «ProvaRadioButton» 4.3

```
{
    TextBox txtNome;
    RadioButton rbuPrimaClasse;
    RadioButton rbuClasseEconomica;
    Button btnConferma;
    public MainForm()
    {
        Text = "ProvaRadioButton";
        StartPosition = FormStartPosition.CenterScreen;

        // Prima classe
        rbuPrimaClasse = new RadioButton();
        rbuPrimaClasse.Location = new Point(40, 50);
        rbuPrimaClasse.Text = "Prima classe";
        Controls.Add(rbuPrimaClasse);

        // Classe economica - opzione impostata inizialmente
        rbuClasseEconomica = new RadioButton();
        rbuClasseEconomica.Location = new Point(40, 80);
        rbuClasseEconomica.AutoSize = true;
        rbuClasseEconomica.Size = new Size(100, 30);
        rbuClasseEconomica.Text = "Classe economica";
        rbuClasseEconomica.Checked = true;
        Controls.Add(rbuClasseEconomica);

        txtNome = new TextBox();
        txtNome.Location = new Point(40, 20);
        Controls.Add(txtNome);

        btnConferma = new Button ();
        btnConferma.Location = new Point(20, 120);
        btnConferma.Text = "Conferma prenotazione";
        btnConferma.Width = 150;
        btnConferma.Click += new EventHandler(btnConferma_Click);
        Controls.Add(btnConferma);
    }

    void btnConferma_Click (object sender, EventArgs e)
    {
        string s;
        if (rbuPrimaClasse.Checked == true)
```

```
            s = " -- Prima classe: confermata";
        else
            s = " -- Classe economica: confermata";
        MessageBox.Show("Cliente: " + txtNome.Text + s, "Conferma prenotazione");

    }

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

L'esecuzione del programma e produce il seguente output:

Figura 4-4. Output prodotto dal programma «ProvaRadioButton».



4.4.3 Controllo «CheckBox»

Anche un CheckBox, in modo simile a un RadioButton, svolge la funzione di un interruttore, il quale può trovarsi soltanto nello stato «accesso» oppure nello stato «spento» e cioè «marcato» o «non marcato».

> In realtà , per i CheckBox è previsto un terzo stato, «indeterminato», il quale non viene qui preso in considerazione.

Diversamente dai RadioButton, un CheckBox non agisce in collaborazione con altri CheckBox, (anche se nulla impedisce al programmatore di far sì che ciò accada); esso, infatti, definisce lo stato,

checked o unchecked, di una opzione che è indipendente da eventuali altre opzioni rappresentate nell'interfaccia.

Proprietà

Analogamente al controllo RadioButton, la classe CheckBox definisce la proprietà Checked, attraverso la quale è possibile conoscere e impostare il suo stato: «marcato» (true), «non marcato» (false).

Eventi

Analogamente a quanto accade per i RadioButton, di norma non è necessario gestire gli eventi sollevati da un CheckBox. E' comunque appropriato gestire gli eventi Click o CheckedChanged se si desidera che lo stato di uno o più controlli venga aggiornato immediatamente in relazione al cambiamento dello stato del CheckBox in questione.

4.4.4 Uso del controllo «CheckBox»

Laddove un insieme di RadioButton definisce i possibili stati ammissibili per un determinato valore, ogni CheckBox definisce lo stato di un valore a sé stante, che può essere «marcato» o «non marcato». Il programma che segue consente all'utente di modificare il valore di due proprietà di un TextBox – Visible e ReadOnly – impostando lo stato dei due CheckBox corrispondenti. Perché l'aggiornamento delle proprietà in questione sia collegato al cambiamento dello stato dei due CheckBox è necessario gestire l'evento CheckedChanged di entrambi.

[Applicazioni\Windows\exe\ProvaCheckBox.exe] 

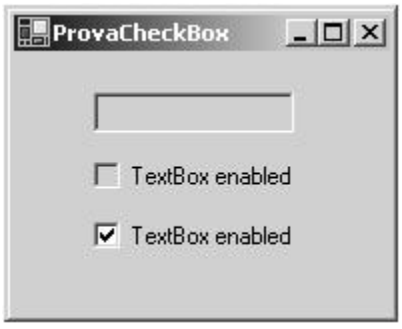
class MainForm «ProvaCheckBox» 4.4

```
{
    TextBox txtNome;
    CheckBox chkEnabled;
    CheckBox chkVisible;
    public MainForm()
    {
        Text = "ProvaCheckBox";
        StartPosition = FormStartPosition.CenterScreen;

        // CheckBox Proprietà Enabled
        chkEnabled = new CheckBox();
        chkEnabled.Location = new Point(40, 50);
        chkEnabled.Width = 200;
        chkEnabled.Text = "TextBox enabled";
        chkEnabled.Checked = true;
        chkEnabled.CheckedChanged += new EventHandler(chkEnabled_CheckedChanged);
        Controls.Add(chkEnabled);
```

```
        // CheckBox Proprietà Visible
        chkVisible = new CheckBox();
        chkVisible.Location = new Point(40, 80);
        chkVisible.Width = 200;
        chkVisible.Text = "TextBox enabled";
        chkVisible.Checked = true;
        chkVisible.CheckedChanged += new EventHandler(chkVisible_CheckedChanged);
        Controls.Add(chkVisible);

        txtNome = new TextBox();
        txtNome.Location = new Point(40, 20);
        Controls.Add(txtNome);

    }

    void chkEnabled_CheckedChanged (object sender, EventArgs e)
    {
        txtNome.Enabled = chkEnabled.Checked;
    }

    void chkVisible_CheckedChanged (object sender, EventArgs e)
    {
        txtNome.Visible = chkVisible.Checked;
    }
    public static void Main()
    {
        Application.Run(new MainForm());
    }
}
```

Sotto è mostrato l'output del programma.

Figura 4-5. Output prodotto dal programma «ProvaCheckBox».



4.5 RAGGRUPPARE I CONTROLLI

Esistono due controlli, Panel e GroupBox, il cui ruolo non è quello di "dialogare" con l'utente ma di fungere da «contenitori» (container) per gli altri controlli, TextBox, Label, RadioButton, eccetera. Come «contenitori» entrambi svolgono una funzione analoga a quella del form, cioè "ospitano" altri controlli e possono influenzare il loro aspetto e il loro comportamento.

Esistono svariate ragione per raggruppare dei controlli inserendoli in un Panel o un GroupBox, tra le quali:

- creare, ad esempio mediante un riquadro o un particolare colore di sfondo, delle aree visivamente separate dal resto dell'interfaccia, dedicate ad ospitare dei controlli logicamente correlati tra loro;
- facilitare l'aggiornamento dello stato di alcune proprietà appartenenti a controlli logicamente correlati tra loro. Ad esempio, impostando a false la proprietà Enabled di un Panel vengono automaticamente disabilitati tutti i controlli in esso contenuti;
- creare insiemi di RadioButton funzionalmente indipendenti tra loro.

In questo senso, le classi Panel e GroupBox hanno modalità di impiego del tutto analoghe, anche se presentano alcune differenze nell'aspetto e nel comportamento.

4.5.1 Classe «GroupBox»

Il controllo GroupBox viene comunemente impiegato come contenitore per RadioButton, consentendo così di gestire insiemi di RadioButton indipendenti tra loro. Infatti, all'interno di un controllo contenitore, di norma il form, un solo RadioButton per volta può trovarsi nello stato «marcato»; dunque, se è necessario gestire due valori distinti, i cui stati sono rappresentati attraverso due insiemi di RadioButton, è necessario collocare tali insiemi in contenitori a loro volta distinti.

Il programma che segue, attraverso due GroupBox, gestisce due insiemi di RadioButton che determinano il «case» e l'allineamento del testo di un TextBox.

[Applicazioni\Windows\exe\ProvaGroupBox.exe] 

class MainForm «ProvaGroupBox» 4.5

```
{
    GroupBox grpCase, grpAlign;
    RadioButton rbuMinusc, rbuMaiusc;
    RadioButton rbuLeft, rbuRight;
    TextBox txtEsempio;
    public MainForm()
    {
        Text = "ProvaCheckBox";
        StartPosition = FormStartPosition.CenterScreen;
```

```
// TextBox          <! dev'essere creato per primo !>
txtEsempio = new TextBox();
txtEsempio.Location = new Point(80, 24);
Controls.Add(txtEsempio);

// GroupBox «case» ----------------------------------
grpCase = new GroupBox ();
grpCase.Location = new Point(32, 64);
grpCase.Size = new Size (96, 80);
grpCase.Text = "Case";
Controls.Add(grpCase);

rbuMaiusc = new RadioButton();
rbuMaiusc.Text = "Maiuscolo";
rbuMaiusc.Location = new Point(8, 24);
rbuMaiusc.Width = 80;
rbuMaiusc.CheckedChanged += new EventHandler(Case_CheckedChanged);
rbuMaiusc.Checked = true;
grpCase.Controls.Add(rboMaiusc);

rbuMinusc = new RadioButton();
rbuMinusc.Text = "Minuscolo";
rbuMinusc.Location = new Point(8, 48);
rbuMinusc.Width = 80;
rbuMinusc.CheckedChanged += new EventHandler(Case_CheckedChanged);
grpCase.Controls.Add(rboMinusc);

// GroupBox «bordo» ----------------------------------
grpAlign = new GroupBox ();
grpAlign.Location = new Point(160, 64);
grpAlign.Size = new Size (96, 80);
grpAlign.Text = "Allineamento";
Controls.Add(grpAlign);

rbuLeft = new RadioButton();
rbuLeft.Text = "Sinistra";
rbuLeft.Location = new Point(8, 24);
rbuLeft.Width = 80;
rbuLeft.CheckedChanged += new EventHandler(Align_CheckedChanged);
grpAlign.Controls.Add(rboLeft);

rbuRight = new RadioButton();
rbuRight.Text = "Destra";
rbuRight.Location = new Point(8, 48);
rbuRight.Width = 80;
```

```
        rbuRight.CheckedChanged += new EventHandler(Align_CheckedChanged);
        rbuRight.Checked = true;
        grpAlign.Controls.Add(rboRight);

    }

    void Case_CheckedChanged (object sender, EventArgs e)
    {
        if (rbuMaiusc.Checked == true)
            txtEsempio.CharacterCasing = CharacterCasing.Upper;
        else
            txtEsempio.CharacterCasing = CharacterCasing.Lower;
    }

    void Align_CheckedChanged (object sender, EventArgs e)
    {
        if (rbuLeft.Checked == true)
            txtEsempio.TextAlign = HorizontalAlignment.Left;
        else
            txtEsempio.CharacterCasing = HorizontalAlignment.Left;
    }

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

Sotto è mostrato l'output del programma.

Figura 4-6. Output del programma «ProvaGroupBox»

In neretto sono evidenziate le istruzioni che aggiungono i vari RadioButton ai rispettivi GroupBox. Da notare che nel programma i due GroupBox non svolgono altra funzione che quella di contenitori, consentendo alle due coppie di RadioButton di funzionare in modo indipendente l'una dall'altra.

Infine altre due osservazioni, che riguardano il programma in sé:

la proprietà Checked dei RadioButton rbuMaiusc e rbuLeft dev 'essere impostata dopo che sono stati attaccati i rispettivi gestori di evento, altrimenti, all'inizio, lo stato del TextBox non viene aggiornato;

il TextBox dev 'essere creato prima dei RadioButton, poiché l'impostazione iniziale, all'interno del costruttore, delle proprietà sopra menzionate determina l'esecuzione dei gestori di evento Case_CheckedChanged() e Align_CheckedChanged(), i quali accedono al TextBox, che ovviamente deve già essere stato creato.

4.5.2 Classe «Panel»

Oltre che come generico contenitore di controlli, impiegato per motivi estetici o per facilitare l'aggiornamento dello stato di controlli logicamente correlati, l'uso del Panel è appropriato in quei casi nei quali in una certa area del form è troppo piccola per il numero e/o le dimensioni dei controlli che deve ospitare[13]. Il Panel, infatti, mediante due «barre di scorrimento», verticale e orizzontale, è in grado di rappresentare un'area più grande di quella effettivamente visualizzabile, consentendo di rendere visibile, e quindi accessibile all'utente, soltanto una parte di tale area in un dato momento, insieme ai controlli che essa contiene. Per abilitare il Panel all'impiego delle barre di scorrimento è necessario impostare a true la proprietà AutoScroll.

Il programma di esempio che segue non sfrutta comunque questa caratteristica; esso mostra come utilizzare un Panel per abilitare o disabilitare un gruppo di controlli in relazione allo stato di un CheckBox.

 [Applicazioni\Windows\exe\ProvaPanel.exe]

class MainForm «ProvaPanel» 4.6

```
{
    Label lblNome, lblFacolta, lblAnno;
    TextBox txtNome, txtFacolta, txtAnno;
    Panel pnlLaurea;
    CheckBox chkLaureato;
    public MainForm()
    {
        Text = "ProvaPanel";
        StartPosition = FormStartPosition.CenterScreen;

        // Label lblNome
        lblNome = new Label();
        lblNome.Location = new Point(8, 16);
```

```
lblNome.AutoSize = true;
lblNome.Text = "Nome e cognome";
Controls.Add(lblNome);

// TextBox txtNome
txtNome = new TextBox();
txtNome.Location = new Point(8, 40);
txtNome.Text = "";
Controls.Add(txtNome);

// CheckBox laureato
chkLaureato = new CheckBox();
chkLaureato.Location = new Point(144, 40);
chkLaureato.Text = "Laureato";
chkLaureato.Checked = false;
chkLaureato.CheckedChanged += new
                                    EventHandler(chkLaureato_CheckedChanged);
Controls.Add(chkLaureato);

// Panel
pnlLaurea = new Panel();
pnlLaurea.Location = new Point(8, 72);
pnlLaurea.Size = new Size(232, 72);
pnlLaurea.BackColor = Color.SteelBlue;
pnlLaurea.Enabled = false;
Controls.Add(pnlLaurea);

// Label lblfacolta
lblFacolta = new Label();
lblFacolta.Location = new Point(8, 8);
lblFacolta.AutoSize = true;
lblFacolta.Text = "Facolta";
pnlLaurea.Controls.Add(lblFacolta);

// TextBox txtFacolta
txtFacolta = new TextBox();
txtFacolta.Location = new Point(8, 32);
pnlLaurea.Controls.Add(txtNome);

// Label lblAnno
lblAnno = new Label();
lblAnno.Location = new Point(120, 8);
lblAnno.AutoSize = true;
```

[13] Situazioni che comunque dovrebbero essere evitate nella progettazione dell'interfaccia.

```
        lblAnno.Text = "Anno di laurea";
        pnlLaurea.Controls.Add(lblAnno);

        // TextBox txtAnno
        txtAnno = new TextBox();
        txtAnno.Location = new Point(120, 32);
        pnlLaurea.Controls.Add(txtAnno);
    }

    void chkLaureato_CheckedChanged (object sender, EventArgs e)
    {
        pnlLaurea.Enabled = chkLaureato.Checked;
    }

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

Sotto è mostrato l'output del programma:

Figura 4-7. Output del programma «ProvaPanel».

In sostanza, il programma abilita l'inserimento dei dati universitari soltanto se l'utente si dichiara laureato. L'uso del Panel semplifica notevolmente l'operazione, riducendola a una sola istruzione, invece delle quattro che sarebbero necessarie per abilitare/disabilitare i quattro controlli.

4.6 GESTIRE LE IMMAGINI

Le immagini sono parte integrante delle interfacce moderne. Esse possono svolgere svariati ruoli:

come informazioni da rappresentare ed elaborare; si pensi a un archivio fotografico, alle foto dei dipendenti di un archivio aziendale, all'oggetto di elaborazione di un programma di fotoritocco, eccetera;

per caratterizzare i controlli dell'interfaccia: le icone che caratterizzano i bottoni, i menù, le message dialog, eccetera;

come elemento dell'interfaccia che risponde alle azioni dell'utente;

come elemento decorativo;

La modalità di rappresentazione e gestione di un'immagine dipende dal ruolo che riveste; un esempio è già stato incontrato nel paragrafo 3.4.4, in relazione alla proprietà Image della classe Button, che consente di caratterizzare un bottone con un'icona. In questo, come in molti altri casi, l'immagine è parte caratterizzante dell'aspetto di un controllo, ma non ne determina in modo fondamentale le funzionalità. Esiste invece un particolare controllo, il PictureBox, il cui solo scopo è appunto quello di gestire immagini.

4.6.1 Classe «PictureBox»

Il controllo PictureBox è fondamentalmente caratterizzato da due elementi, rappresentati da altrettante proprietà:

l'immagine da visualizzare;

la modalità di visualizzazione dell'immagine.

Proprietà

Tabella 4-4. Proprietà della classe PictureBox. (continua)

PROPRIETÀ - TIPO	DESCRIZIONE
Image Image	Definisce l'immagine da visualizzare. Un modo per impostare il valore di questa proprietà e quello di specificare il file che contiene l'immagine mediante il metodo FromFile della classe Image: picEsempio.Image = Image.FromFile(@"nomeicona.bmp");

Tabella 4-4. Proprietà della classe PictureBox.

PROPRIETÀ - TIPO	DESCRIZIONE
SizeMode PictureBoxSizeMode	Definisce il modo in cui l'immagine viene visualizzata: Normal: l'immagine è visualizzata a partire dall'angolo in alto a sinistra dell'area del controllo. La parte che eccede tale area viene tagliata; StretchImage: l'immagine viene ridimensionata per coprire per intero l'area del controllo; AutoSize: il controllo modifica le proprie dimensioni in relazione a quelle dell'immagine; CenterImage: l'immagine è centrata rispetto all'area del controllo. Le parti di immagine che eventualmente eccedono tale area vengono tagliate.

Eventi

Di norma non è necessario gestire gli eventi sollevati da questo controllo. Comunque, se si desidera che risponda alle azioni dell'utente può essere appropriato gestire l'evento Click.

4.6.2 Uso del controllo «PictureBox»

Nel programma che segue, due PictureBox visualizzano la stessa immagine, usando diverse modalità di rappresentazione.

 [Applicazioni\Windows\exe\ProvaPictureBox.exe]

class MainForm «ProvaPictureBox» 4.7

```
{
    PictureoBox picA, picB;
    Button btnScambia;
    public MainForm()
    {
        Text = "ProvaPictureBox";
        StartPosition = FormStartPosition.CenterScreen;

        picA = new PictureBox();
        picA.Location = new Point(16, 40);
        picA.Size = new Size(98, 82);
        picA.Image = Image.FromFile(@"immagini\atleta.jpg");
        picA.SizeMode = PictureBoxSizeMode.StretchImage;

        Controls.Add(picA);
```

```
        picB = new PictureBox();
        picB.Location = new Point(160, 40);
        picB.Size = new Size(98, 82);
        picB.Image = Image.FromFile(@"immagini\atleta.jpg");
        picB.SizeMode = PictureBoxSizeMode.CenterImage;

        Controls.Add(picB);

        btnScambia = new Button();
        btnScambia.Location = new Point(78, 144);
        btnScambia.Size = new Size(116, 24);
        btnScambia.Text = "Scambia modalità";
        btnScambia.Click += new EventHandler(btnScambia_Click);
        Controls.Add(btnScambia);
    }

    void btnScambia_Click(object sender, EventArgs e)
    {
        if (picA.SizeMode == PictureBoxSizeMode.StretchImage)
            picA.SizeMode = PictureBoxSizeMode.CenterImage;
        else
            picA.SizeMode = PictureBoxSizeMode.StretchImage;

        if (picB.SizeMode == PictureBoxSizeMode.StretchImage)
            picB.SizeMode = PictureBoxSizeMode.CenterImage;
        else
            picB.SizeMode = PictureBoxSizeMode.StretchImage;
    }

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

Allo scopo di rendere ancor più evidente la differenza di rappresentazione dell'immagine in relazione al valore della proprietà SizeMode, il programma, mediante la gestione dell'evento Click del bottone btnScambia, consente all'utente di scambiare tra loro le modalità di visualizzazione impiegate dai due PictureBox.

Nella pagina successiva è mostrato l'output del programma prima e dopo che l'utente ha cliccato sul bottone:

Figura 4-8. Output del programma «ProvaPictureBox».

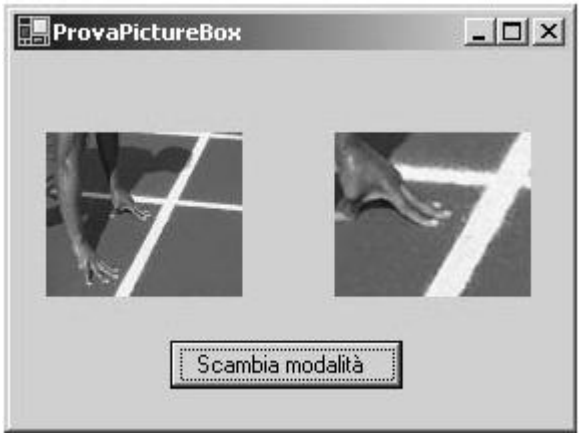
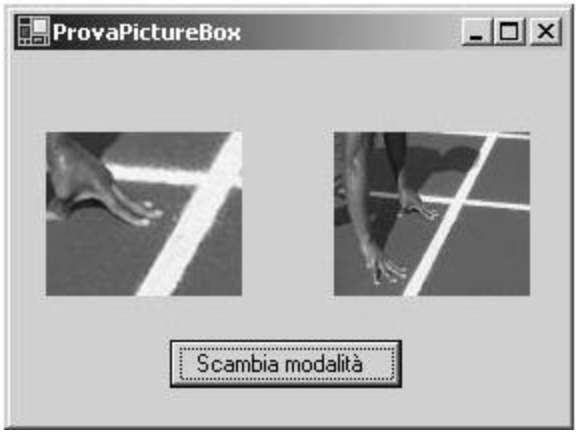


Figura 4-9. Output del programma «ProvaPictureBox».



Capitolo 5 – Esempio di una interfaccia completa

5.1 DEFINIZIONE DEL PROBLEMA

5.1.1 Premessa

L'obiettivo di questo capitolo è di applicare a un caso (quasi) realistico alcune delle conoscenze apprese nei capitoli precedenti. A questo scopo viene mostrata la realizzazione di un programma che presenta la sola parte interfaccia, la quale implementa alcune verifiche di consistenza.

5.1.2 Testo del problema

«Si richiede la realizzazione di un programma che consenta la compilazione di un curriculum personale. Il curriculum è composto dai seguenti dati:

1) nome
2) cognome;
3) residenza;
4) sesso;
5) titolo di studio conseguito:
   "Nessuno", "Licenza elementare", "Licenza media", "Diploma superiore", "Laurea";
6) corso frequentato (se il titolo di studio conseguito è "Laurea");
7) esperienze lavorative precedenti.

Il programma, in sostanza, dovrà implementare un'interfaccia che consenta all'utente di inserire i dati richiesti e di confermare l'avvenuto inserimento mediante il clic su un bottone. Mediante un secondo bottone l'utente potrà azzerare tutti i dati inseriti, riportando il contenuto e lo stato dei controlli alle loro condizioni iniziali.

5.1.3 Files del progetto

Progetto(realizzato con «Sharp Develop»):

[Applicazioni\Windows\progetti\Curricolo\Curricolo.cmbx] 

File codice sorgente:

[Applicazioni\Windows\progetti\Curricolo\MainForm.cs] 

Programma eseguibile:

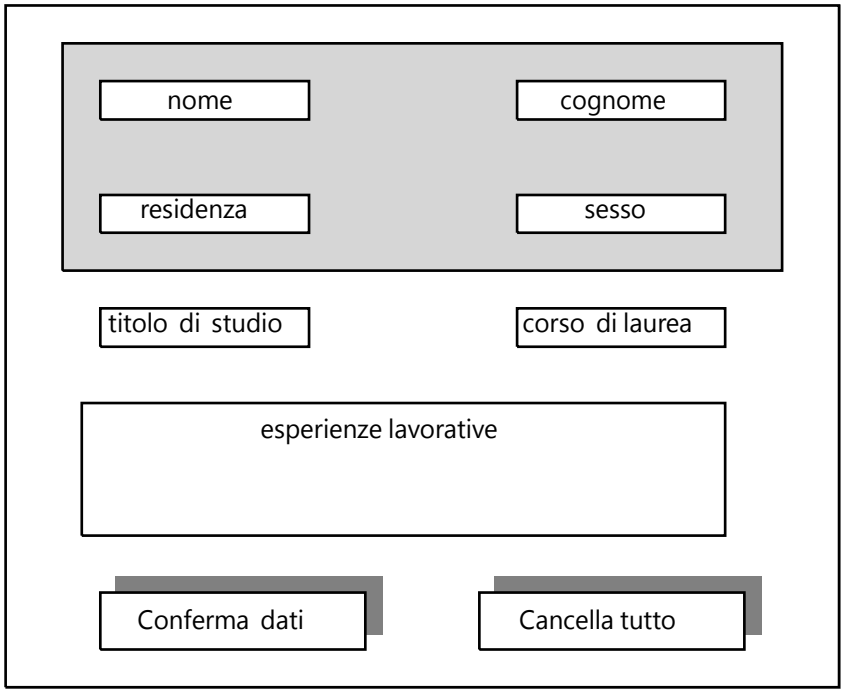
[Applicazioni\Windows\exe\Curricolo.exe 

5.2 PROGETTAZIONE DELL'INTERFACCIA

5.2.1 Progettazione del «layout»

Si decide di suddividere l'interfaccia in due aree distinte; la prima riservata ai dati personali: nome e cognome, residenza, sesso; la seconda riservata a tutti gli altri dati. Segue uno schema che mostra il layout generale (disposizione) dell'interfaccia:

Figura 5-1. Layout generale dell'interfaccia.



Nota bene: lo schema mostra una visione generale di quella che dovrà essere l'interfaccia, non specifica affatto l'esatta natura dei controlli e le eventuali verifiche di consistenza sui dati.

5.2.2 Scelta dei controlli

In questa fase si decide quali controlli utilizzare per l'acquisizione dei dati. Alcune scelte sono del tutto ovvie, altre sono in parte arbitrarie.

Tabella 5-1. Lista dei controlli utilizzati per l'acquisizione dei dati. (continua)

DATO	CONTROLLO	NOME CONTROLLO
nome	TextBox	txtNome
cognome	TextBox	txtCognome
Residenza	TextBox	txtResidenza
Sesso	RadioButton	rbuMaschile rbuFemminile

Tabella 5-1. Lista dei controlli utilizzati per l'acquisizione dei dati.

DATO	CONTROLLO	NOME CONTROLLO
Titolo di studio	ComboBox	cboTitolo
Corso di laurea	TextBox	txtCorso
Esperienze lavorative	TextBox (multilinea)	txtEsperienze

I due bottoni, il primo per la conferma dei dati, il secondo l'azzeramento dei dati inseriti, sono chiamati rispettivamente: btnConferma e btnCancella.

Vi sono inoltre altri tre controlli:

pnlDatiPersonali: un Panel il cui scopo è quello di evidenziare mediante un appropriato colore di sfondo l'area del form dedicata ai dati personali;

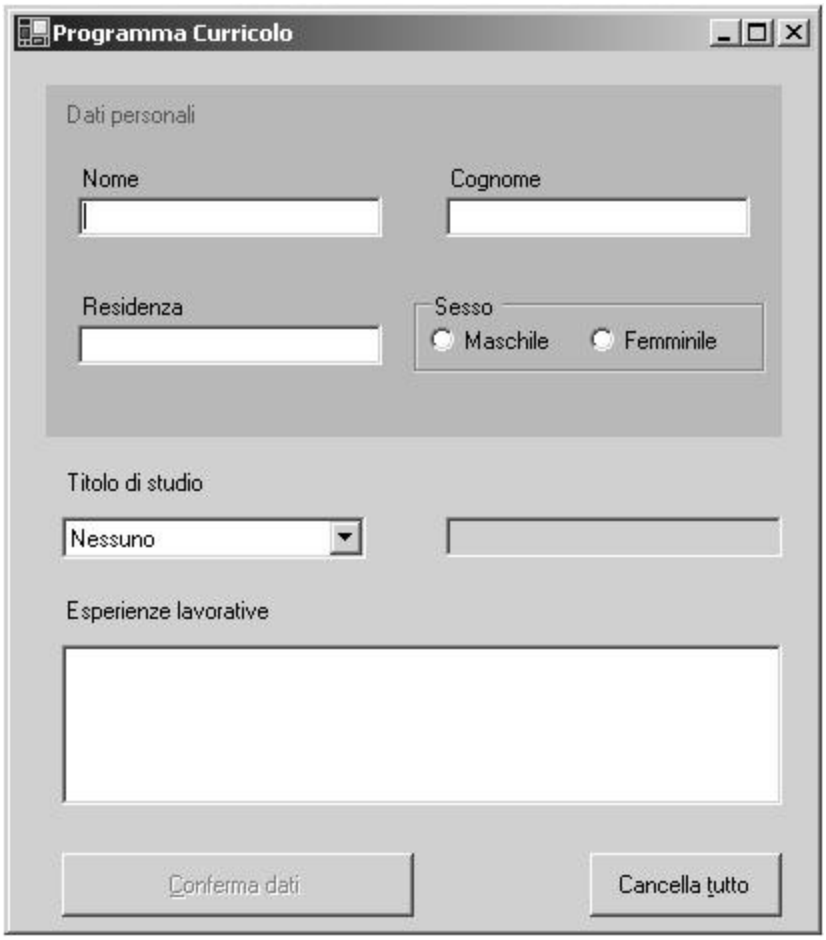
grpSesso: GroupBox che fa da contenitore ai due RadioButton per la selezione del sesso; esattamente come il controllo precedente, anche grpSesso riveste un ruolo puramente estetico e non partecipa in alcun modo al dialogo con l'utente;

lblCorso: Label il cui testo qualifica il controllo txtCorso. I possibili valori della proprietà Text dell'etichetta possono essere: "Laureato in", "Laureata in" o stringa vuota, in base al tipo di selezione fatta dall'utente in relazione al sesso.

5.3 LAYOUT DELL'INTERFACCIA

La figura che segue mostra l'apparenza dell'interfaccia dopo che sono stati inseriti tutti i controlli:

Figura 5-2. Apparenza dell'interfaccia.



Poiché composto da un notevole numero di righe, viene omesso il codice completo, collocato nel costruttore della classe MainForm, che determina la costruzione dell'interfaccia. (Vedi file «Curricolo.cs» nella cartella «Applicazioni\Windows\sorgenti\Curricolo» per il codice sorgente).

A titolo dimostrativo segue il codice che costruisce i controlli rbuMaschile, rbuFemminile, grpSesso, cboTitolo:

class MainForm Curricolo 5.1

```
{
    ...
    ComboBox cboTitolo;
    RadioButton rbuMaschile;
    RadioButton rbuFemminile;
    GroupBox grpSesso;
    ...
```

```
    public MainForm()
    {
        Text = "Programma Curricolo";

        // rboMaschile
        rbuMaschile = new RadioButton();
        rbuMaschile.Location = new Point(8, 16);
        rbuMaschile.Size = new Size(80, 16);
        rbuMaschile.Text = "Maschile";
        ...
        // rboFemminile
        rbuFemminile = new RadioButton();
        rbuFemminile.Location = new Point(88, 16);
        rbuFemminile.Size = new Size(80, 16);
        rbuFemminile.Text = "Femminile";
        ...
        // grpSesso
        grpSesso = new GroupBox();
        grpSesso.Location = new Point(184, 104);
        grpSesso.Size = new Size(176, 40);
        grpSesso.Text = "Sesso";
        grpSesso.Controls.Add(rbuMaschile);
        grpSesso.Controls.Add(rbuFemminile);
        ...
        // cboTitolo
        cboTitolo = new ComboBox();
        cboTitolo.DropDownStyle = ComboBoxStyle.DropDownList;
        cboTitolo.Location = new Point(24, 232);
        cboTitolo.Size = new Size(152, 21);
        cboTitolo.SelectedIndexChanged += new
                        EventHandler(cboTitolo_SelectedIndexChanged);
        ...
        Load += new EventHandler(MainForm_Load);
    }
}
```

5.4 CONTENUTO INIZIALE DEI CONTROLLI

Esaminando il precedente codice si nota che all'interno del costruttore non viene impostato il contenuto iniziale dei controlli. Allo scopo di semplificare il programma si è deciso di collocare nel costruttore il solo codice che ne definisce l'aspetto; il loro contenuto iniziale viene impostato contestualmente all'evento di caricamento del form, gestito dal metodo MainForm_Load():

Curricolo 5.2
```
void MainForm_Load (object sender, EventArgs e)
{
    cboTitolo.DataSource = titoliDiStudio;
    InizializzaCampi();
}
```

La riga di codice:

```
Load += new EventHandler(MainForm_Load);
```

posta nel costruttore attacca il gestore all'evento in questione.

La variabile titoliDiStudio, un vettore di stringhe, è dichiarata come campo di classe:

Curricolo 5.3
```
«dichirazione del vettore titoliDiStudio
string[] titoliDiStudio =
          {"Nessuno",
           "Licenza elementare",
           "Licenza media",
           "Diploma superiore",
           "Laurea"};
```

Il gestore di evento esegue due compiti:

popola cboTitolo;

esegue il metodo InizializzaCampi() il cui scopo è quello di impostare il contenuto iniziale dei controlli.

Segue la definizione del metodo:

Curricolo 5.4
```
void InizializzaCampi()
{
    txtNome.Text = "";
    txtCognome.Text = "";
    txtResidenza.Text = "";
    cboTitolo.Text = "Nessuno";
    txtCorso.Text = "";
    txtEsperienze.Text = "";
    rbuMaschile.Checked = false;
    rbuFemminile.Checked = false;
```

```
}
```

Collocare il codice di inizializzazione dei controlli in un metodo apposito presenta in questo caso due vantaggi:

uno, di carattere generale, è dato dal fatto che centralizzare tale codice in un unico posto invece di disperderlo all'interno del costruttore facilita la comprensione e la modifica del programma;

uno, relativo all'interfaccia in questione, risiede nel fatto che il programma prevede la possibilità per l'utente di azzerare i dati inseriti; l'azzeramento dei dati è appunto prodotto dal precedente codice, che viene dunque invocato sia all'inizio del programma, sia in risposta al clic dell'utente sul bottone btnCancella.

Questa strategia obbedisce alla regola generale:

mai scrivere due parti di codice distinte che svolgono lo stesso compito.

In questo caso è sempre opportuno collocare il codice in questione in un metodo, che sarà invocato dove necessario.

A questo proposito c'è una considerazione da fare. Mentre l'impostazione del contenuto iniziale di cboTitolo viene eseguita, come per tutti gli altri controlli, all'interno di InizializzaCampi(), il popolamento della lista avviene fuori dal metodo, direttamente nel gestore di evento. Ciò è appropriato, poiché mentre il contenuto – l'elemento selezionato – di cboTitolo può essere reinizializzato in risposta alle azioni dell'utente, l'elenco dei titoli di studio non varia mai durante l'esecuzione del programma e dunque può essere impostato una volta soltanto durante il caricamento del form.

5.5 RENDERE L'INTERFACCIA CONSISTENTE

Intervenire sulla consistenza dell'interfaccia rappresenta un'operazione soggetta alle scelte del programmatore, sia in relazione a quali verifiche si decide di implementare, sia in relazione al modo di implementarle. In questo senso, comunque, va la regole generale che un'interfaccia non dovrebbe soltanto garantire la coerenza dei dati in relazione alle elaborazioni effettuate su di essi, ma garantire anche:

- la coerenza dell'aspetto e dello stato dei controlli, laddove essi dipendono dal contenuto e/o dallo stato di altri controlli;
- che le azioni dell'utente siano dirette in modo appropriato, impedendo a quest'ultimo di agire in modo incoerente in relazione allo stato e/o al contenuto di determinati controlli.

Dunque, un'interfaccia può essere consistente per quanto riguarda la verifica dei dati, garantendo che questi siano comunque coerenti con le future elaborazioni, ma inconsistente per quanto riguarda l'aspetto e il comportamento, consentendo all'utente azioni inappropriate e comunicando con il medesimo in modo incoerente e ambiguo. Un'interfaccia davvero robusta deve curare entrambi gli aspetti.

Nel programma «Curricolo» si è deciso di considerare tre aspetti:

- garantire l'esistenza dei dati personali (nome, cognome, residenza, sesso);
- consentire l'accesso al controllo `txtCorso` soltanto se il titolo di studio selezionato è "Laurea";
- modificare il testo informativo (l'etichetta) associato del controllo `txtCorso` in relazione al fatto che l'utente si dichiari maschio o femmina;

5.5.1 Garantire l'esistenza dei dati personali

In questo caso, a scolpo dimostrativo, è stato deciso di implementare due forme diverse di verifica, una, anticipata, per i dati nome, cognome e residenza, l'altra, posticipata, per il sesso.

Verifica anticipata di esistenza dei dati nome, cognome e residenza

Tale verifica condiziona lo stato abilitato/disabilitato del bottone `btnConferma` all'effettiva esistenza di un contenuto nei controlli corrispondenti ai dati citati. Ciò si ottiene gestendo l'evento `TextChanged` dei controlli in questione.

Curricolo 5.5

```
void DatiPersonali_TextChanged (object sender, EventArgs e)
{
        btnConferma.Enabled =(txtNome.Text != "" &&
                              txtCognome.Text != "" &&
                              txtResidenza.Text != "");
}
```

Ovviamente, nel costruttore tale gestore di evento dev'essere attaccato a tutti e tre i controlli:

```
...Curricolo 5.6
txtNome.TextChanged += new EventHandler (DatiPersonali_TextChanged);
...
txtCognome.TextChanged += new EventHandler (DatiPersonali_TextChanged);
...
txtResidenza.TextChanged += new EventHandler (DatiPersonali_TextChanged);
...
```

Verifica posticipata di esistenza del sesso

In questo caso la verifica avviene dopo che è stato cliccato su `btnConferma` e quindi all'interno del gestore dell'evento `Click` di tale bottone:

```
void btnConferma_Click(object sender, EventArgs e)                    Curricolo 5.7
{
    bool sesso = (rbuMaschile.Checked == true || rbuFemminile.Checked == true);
    if (sesso == false)
    {
        MessageBox.Show("Dati incompleti: specificare il sesso",
                        "Programma Curricolo", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
        return;
    }

    MessageBox.Show("Dati confermati", "Programma Curricolo",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

La verifica si traduce nel testare se l'uno o l'altro dei due `RadioButton` è marcato. Se nessuno dei due lo è (variabile `sesso == false`) viene visualizzato un messaggio di errore che comunica all'utente di inserire il dato mancante.

Figura 5-3. Messaggio di errore che comunica all'utente di inserire il dato mancante.

> Nota bene. In un programma realistico non ci sarebbe alcun motivo per gestire il dato relativo al sesso in modo diverso dagli altri dati personali.

5.5.2 Condizionare l'accesso a «txtCorso» al valore "Laurea" di «cboTitolo»

Questa forma di verifica fa sì che il TextBox relativo al corso di laurea sia accessibile soltanto se l'utente dichiara, mediante il cboTitolo, di essere laureato. Ciò si ottiene gestendo l'evento SelectedIndexChanged di cboTitolo e impostando lo stato Enabled in relazione al contenuto della proprietà Text del ComboBox:

```
void cboTitolo_SelectedIndexChanged (object sender, EventArgs e)                Curricolo 5.8
{
    txtCorso.Enabled = (cboTitolo.Text == "Laurea");
}
```

Da notare che il gestore di evento si limita a modificare lo stato di abilitazione del controllo txtCorso e non il suo contenuto. In altre parole, l'utente potrebbe dichiarare di essere laureato, digitare il corso di laurea nel controllo txtCorso e successivamente modificare il titolo di studio, selezionando ad esempio "Scuola superiore". In questo caso txtCorso verrebbe disabilitato ma manterrebbe il proprio contenuto, precedentemente inserito.

5.5.3 Modificare l'etichetta associata al controllo «txtCorso»

Questa forma di verifica è puramente a scopo dimostrativo, poiché un'etichetta statica del tipo "Laureato/a in" risulterebbe senz'altro sufficiente.

Per far sì che il testo dell'etichetta lblCorso sia coerente con il sesso dichiarato dall'utente, occorre gestire l'evento CheckedChanged sollevato dai due controlli:

```
void Sesso_CheckedChanged (object sender, EventArgs e)                          Curricolo 5.9
{
    if (rbuMaschile.Checked == true)
        lblCorso.Text = "Laureato in:";
    else
    if (rbuFemminile.Checked == true)
        lblCorso.Text = "Laureata in:"
    else
        lblCorso.Text = ""
}
```

Ovviamente, all'interno del costruttore, il gestore di evento dev'essere attaccato ad entrambi i RadioButton:

```
...Curricolo 5.10
rbuMaschile.CheckedCahnged += new EventHandler (Sesso_CheckedChanged);
...
rbuFemminile.CheckedCahnged += new EventHandler (Sesso_CheckedChanged);
...
```

5.6 STATO INIZIALE DEL CONTROLLI

Il codice implementato finora garantisce un comportamento consistente dell'interfaccia ma trascura un aspetto molto importante: garantire la consistenza dello stato iniziale dei controlli. Il problema sta nel fatto che l'esecuzione dei gestori di eventi che gestiscono il cambiamento di stato e di contenuto dei controlli ad essi attaccati non è garantita se non in risposta alle azioni dell'utente. Ciò fa sì che all'atto della esecuzione del programma, e quindi del caricamento dell'interfaccia, alcuni controlli possano trovarsi in uno stato inconsistente poiché non sono ancora stati eseguiti i metodi che ne gestiscono il cambiamento di stato.

Ad esempio, il testo dell'etichetta lblCorso dovrebbe valere inizialmente stringa vuota, ma non è certo che sia così[14], poiché prima che l'utente compia alcuna azione non è garantita l'esecuzione del gestore di evento Sesso_CheckedChanged, il cui scopo è appunto quello di impostare in modo coerente il testo dell'etichetta. Quanto detto vale anche per altri controlli.

Di questo problema esistono due soluzioni: la prima del tutto ovvia, ma poco soddisfacente; la seconda meno intuitiva, ma che garantisce in qualsiasi momento la consistenza dell'interfaccia.

5.6.1 Impostare "manualmente" lo stato iniziale dei controlli

Ciò si traduce nello scrivere, all'interno del costruttore o del gestore di evento MainForm_Load, il codice che imposta lo stato iniziale dei controlli dell'interfaccia. Ad esempio

Curricolo 5.11

```
void MainForm_Load (object sender, EventArgs e)
{
    cboTitolo.DataSource = titoliDiStudio;
    InizializzaCampi();

    btnConferma.Enabled = false;
    lblCorso.Text = "";
    txtCorso.Enabled = false;
}
```

Il codice funziona, ma presenta dei problemi potenziali. Infatti, eventuali modifiche nelle regole di gestione della consistenza dell'interfaccia non si rifletterebbero nello stato iniziale dei controlli. Ad esempio, attualmente il testo iniziale dell'etichetta lblCorso è stringa vuota, e ciò è coerente con lo stato iniziale dei due RadioButton rbuMaschile e rbuFemminile, per entrambi falso. In altre parole, finché l'utente non dichiara il proprio sesso, il testo dell'etichetta non può che essere indefinito, e cioè vuoto.

Si supponga però di decidere che il valore default per il sesso sia "Maschile", e che dunque rbuMaschile sia marcato. Per coerenza, dunque, il valore iniziale di lblCorso dovrebbe diventare "Laureato in". Ma perché ciò sia vero, con l'attuale impostazione dello stato iniziale dei controlli, occorre intervenire direttamente sul codice di inizializzazione, modificandolo in modo appropriato.

[14] Non è certo poiché dipende dall'ordine in cui vengono eseguite le istruzioni che impostano lo stato dei controlli.

Ora, nel caso specifico l'obiezione appare abbastanza artificiosa, ma si immagini una situazione simile con una ventina di controlli, per molti dei quali esistano relazioni di dipendenza analoghe: qualsiasi modifica nelle regole di gestione di consistenza avrebbe il potere di rendere lo stato iniziale dell'interfaccia inconsistente.

5.6.2 Scrivere un metodo che aggiorni lo stato dei controlli

Questa soluzione non solo risolve il problema di impostare in modo consistente lo stato iniziale dei controlli, ma semplifica anche il codice di aggiornamento di tale stato presente nei gestori di eventi. L'obiettivo è il seguente: centralizzare in un solo metodo tutto il codice che svolge il compito di aggiornare lo stato dei controlli che hanno una relazione di dipendenza con lo stato e/o il contenuto di altri controlli. Quindi, ogni qual volta è necessario garantire uno stato consistente dell'intera interfaccia è sufficiente invocare il metodo in questione:

Curricolo 5.12

```
void AggiornaStatoControlli ()
{
    btnConferma.Enabled =(txtNome.Text != "" &&
                          txtCognome.Text != "" &&
                          txtResidenza.Text != "");

    txtCorso.Enabled = (cboTitolo.Text == "Laurea");

    if (rbuMaschile.Checked == true)
        lblCorso.Text = "Laureato in:";
    else
    if (rbuFemminile.Checked == true)
        lblCorso.Text = "Laureata in:"
    else
        lblCorso.Text = ""
}
```

Come si vede, il metodo AggiornaStatoControlli() contiene tutto il codice di gestione di consistenza dell'interfaccia scritto in precedenza e disperso nei vari gestori di eventi. Il metodo dev'essere invocato all'inizio del programma, dopo che è stato caricato il form, e in qualunque gestore di evento che risponda al cambiamento dello stato di uno dei controlli dell'interfaccia:

Curricolo 5.13

```
void DatiPersonali_TextChanged(object sender, EventArgs e)
{
    AggiornaStatoControlli();
}
...
void Sesso_CheckedChanged(object sender, EventArgs e)
{
    AggiornaStatoControlli();
}
```

```
void cboTitolo_SelectedIndexChanged(object sender, EventArgs e)
{
    AggiornaStatoControlli();
}
...
void MainForm_Load (object sender, EventArgs e)
{
    InizializzaCampi();
    cboTitolo.DataSource = titoliDiStudio;
    AggiornaStatoControlli();
}
```

Vantaggi e svantaggi di questo approccio

Ci sono i seguenti vantaggi:

- l'esecuzione del metodo AggiornaStatoControlli() garantisce la consistenza dell'interfaccia; la sua esecuzione all'inizio del programma garantisce dunque anche uno stato iniziale coerente d tutti i controlli;
- il codice di gestione della consistenza viene centralizzato in un unico punto e dunque può essere facilmente modificato o esteso;
- il codice dei gestori di eventi che rispondono ai cambiamenti di stato dei controlli si riduce alla semplice invocazione del metodo in questione.

Questo approccio comporta comunque un problema potenziale, di norma trascurabile, ma in alcuni casi (non questo) di una certa rilevanza:

- qualsiasi cambiamento in uno qualsiasi dei controlli determina l'esecuzione dell'intero codice che gestisce la consistenza dell'interfaccia, e dunque anche di quel codice che non ha nessuna relazione con il controllo che ha subito il cambiamento.

Esiste dunque un problema di inefficienza, trascurabile nella maggior parte dei casi, ma rilevante se il codice di gestione della consistenza è piuttosto corposo oppure deve compiere elaborazioni sofisticate e dunque potenzialmente rilevanti in termini di costo computazionale.

In realtà esiste un secondo potenziale problema, di natura più sottile. Esso riguarda il possibile instaurarsi di una condizione di ricorsione indiretta, provocato dal fatto che il cambiamento di un controllo provoca l'esecuzione del metodo di aggiornamento, il quale provoca a sua volta il cambiamento di stato in altri controlli, il quale provoca nuovamente l'esecuzione del metodo di aggiornamento, il quale provoca nuovamente l 'aggiornamento ... eccetera.

In generale è raro che possa verificarsi una situazione del genere. D'altra parte, per evitare di realizzare un'interfaccia potenzialmente malfunzionante occorre eseguire dei test accurati che ne verificano la consistenza indipendentemente dalle azioni dell'utente.