Drawing&Painting

Mini tutorial sull'uso della classe Graphics per il tracciamento di figure

Applicazioni Windows Forms

Anno 2018/2019

Indice generale

1 Drawing

In questo tutorial fornisco un'introduzione alle operazioni di *drawing* nelle applicazioni Windows Forms. Con questo termine si intende la facoltà di disegnare figure geometriche e immagini in un controllo. Le funzione mostrate sono definite nel *namespace* `System.Drawing`.

1.1 Oggetto Graphics

Per poter disegnare sull'area di un controllo occorre innanzitutto creare un oggetto `Graphics` mediante l'invocazione di `CreateGraphics()`. Quest'oggetto fornisce i metodi per disegnare linee, figure geometriche, poligoni, immagini, consentendo di stabilire lo stile e il colore.

Supponi di avere il pannello `pnlArea` e di voler disegnare un cerchio su di esso:

```
private void btnDraw_Click(object sender, EventArgs e)
{
    Graphics gr = pnlArea.CreateGraphics();
    gr.DrawEllipse(Pens.Red, 0, 0, 100, 100);  //-> diametro: 100px
}
```

Nota bene:

- L'oggetto `Graphics` viene creato attraverso `pnlArea`; dunque, ogni operazione eseguita sarà sempre relativa alla superficie del pannello.
- Il metodo `DrawEllipse()` disegna un'ellisse alle coordinate 0,0 e di dimensioni 100, 100. Il tratto utilizzato è una linea continua rossa, stabilita dalla penna predefinita `Pens.Red`.

`DrawEllipse()` esiste in più versioni, le quali consentono di specificare in vari modi i parametri – stile e coordinate – usati per disegnare.

1.2 Disegnare una curva dati i punti che la definiscono

`Graphics` fornisce un metodo, `DrawLines()`, che consente di unire i punti che rappresentano un grafico, producendo una linea continua. Ogni punto è memorizzato in un oggetto di tipo `Point`.

Il seguente codice disegna una "spezzata" formata da 4 punti.

```
Graphics gr = pnlArea.CreateGraphics();
Point[] punti = new Point[]
{
    new Point(0, 0),
    new Point(30, 30),
    new Point(50, 70),
    new Point(160, 120),
};
Pen penna = new Pen(Color.Red, 3);
gr.DrawLines(penna, punti);
```

Nota bene: il tratto è stabilito da una penna personalizzata, di colore rosso e spessa tre pixel.

2 *Painting*: rendere persistente il *drawing*

I disegni ottenuti mediante l'oggetto `Graphics` non sono persistenti. Se eseguissi il codice precedente, riducessi a icona il form e lo visualizzassi di nuovo, ecco che la linea sarebbe scomparsa.

Per rendere un disegno persistente è necessario eseguirlo ogni qual volta la superficie del controllo viene cancellata o modificata da un evento esterno o interno all'applicazione. È windows a comunicare automaticamente questa necessità, "invalidando" il controllo e indicando così che deve essere ridisegnato. Si parla in questo caso di *painting* o *rendering* del controllo.

Si può rispondere in due modi all'invalidazione di un controllo:

- Dall'esterno, gestendo l'evento `Paint` del controllo.
- Dall'interno, nella classe che implementa il controllo, collocando il codice di *rendering* nel metodo virtuale `OnPaint()`.

2.1 Rendere persistente il disegno sul pannello

Riprendendo l'esempio precedente, ecco come rendere persistente il disegno della linea spezzata, gestendo l'evento `Paint` del pannello:

```
private void pnlArea_Paint(object sender, PaintEventArgs e)
{
    Graphics gr = e.Graphics;
    ...
    gr.DrawLines(penna, punti);
}
```

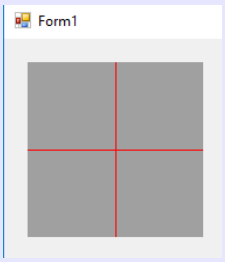
Nota bene: in questo caso l'oggetto `Graphics` non viene creato, ma ottenuto direttamente dal parametro dell'evento. Dopo questa modifica, il pannello mostrerà sempre la linea spezzata, fin dall'avvio del programma.

2.2 Implementare un controllo che implementa il *rendering*

Considera adesso l'idea di realizzare un *user control* che mostra due linee ortogonali rosse che dividono l'area in quattro quadranti uguali. Per ottenere questo risultato è necessario scrivere il codice nel metodo virtuale `OnPaint()`.

```
public partial class DrawControl : UserControl
{
    public DrawControl()
    {
        InitializeComponent();
    }

    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);           //-> chiama versione ereditata da UserControl
```



```
        Graphics gr = e.Graphics;
        int yascissa = Height / 2;
        gr.DrawLine(Pens.Red, 0, yascissa, Width, yascissa);     //-> orizzonale

        int xordinata = Width / 2;
        gr.DrawLine(Pens.Red, xordinata, 0, xordinata, Height);  //-> verticale
    }
}
```

Nota bene: il metodo non viene chiamato esplicitamente, poiché sarà eseguito automaticamente quando il controllo necessita di essere ridisegnato.

2.3 Forzare il *rendering* del controllo: Invalidate()

Molti controlli implementano delle proprietà che ne stabiliscono l'aspetto; evidentemente una loro modifica richiede il *rendering* del controllo. Ciò si ottiene eseguendo il metodo `Invalidate()`, che avvia il processo che culmina nell'esecuzione del metodo `OnPaint()`.

Supponi che `DrawControl` debba consentire la modifica del colore delle linee attraverso una proprietà. Quando alla proprietà viene assegnato un nuovo valore, il controllo deve essere ridisegnato:

```
public partial class DrawControl : UserControl
{
    public DrawControl()
    {
        InitializeComponent();
    }

    private Color _lineColor = Color.Black;  // colore predefinito
    public Color LineColor
    {
        get => _lineColor;
        set
        {
            if (value == _lineColor)
                return;             // il nuovo valore è uguale al precedente
            _lineColor = value;
            Invalidate();           // avvia il processo di risidegno (chiama OnPaint())
        }
    }

    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        var pen = new Pen(LineColor);   // crea un penna basata sull'attuale colore
        Graphics gr = e.Graphics;
        int yascissa = Height / 2;
        gr.DrawLine(pen, 0, yascissa, Width, yascissa);  //->orizzonale
        int xordinata = Width / 2;
```

```
        gr.DrawLine(pen, xordinata, 0, xordinata, Height);
    }
}
```

Nota bene: il *set accessor* della proprietà `LineColor` verifica innanzitutto che il nuovo valore sia diverso dal vecchio (in caso contrario, inutile ridisegnare il controllo). In caso positivo chiama `Invalidate()`, che, indirettamente, esegue `OnPaint()`. Quest'ultimo crea una penna del colore definito dalla proprietà e traccia le due linee.