String Interpolation in C#, concatenare stringhe

La "**String Interpolation**" è una modalità di inserimento di valori all'interno delle stringhe che rende il codice estremamente leggibile. È stata introdotta a partire da C# 6 proprio con l'intento di "semplificare", "condensare" ma soprattutto "rendere chiaro" l'intento del codice.
Prima di esaminare la String Interpolation, è utile ripassare i **metodi per concatenare stringhe** previsti da C# prima della versione 6:

- il più performante **StringBuilder**;
- la concatenazione mediante **operatore +**;
- **string.Format** che è un po' la sintesi dei due metodi. Per completezza ci sarebbe anche il metodo `AppendFormat` dello `StringBuilder`, ma lo possiamo considerare nello *string.Format*.

StringBuilder

Lo **StringBuilder** è il metodo più performante soprattutto in termini di memoria, tuttavia non è il massimo della semplificazione e della lettura.

```
using System;
using System.Text;
using Xunit;


namespace StringInterpolation
{
        public class Obj
        {
                public Obj(int id, string valore, DateTime creato)
                {
                        Id = id;
                        Valore = valore;
                        Creato = creato;
                }
                public int Id { get; set; }
                public string Valore { get; set; }
                public DateTime Creato { get; set; }
        }
        public class Test
    {
                private const string ValoreAtteso = "Ciao sono un oggetto con Id:1,
Valore:'Buono' e creato il 20160101";
```

```
                private const string ValoreAttesoCondizionale = "Ciao sono un oggetto con
Id:1, Valore:'Buono' e creato il 20160101 quindi attuale";
                private readonly Obj _obj = new Obj(1,"Buono", DateTime.Parse("2016 - 01 -
01 00: 00:00"));
            [Fact]
            public void StringBuilderTest()
            {
                    var sb = new StringBuilder();
                    sb.Append("Ciao sono un oggetto con Id:");
                    sb.Append(_obj.Id);
                    sb.Append(", Valore:'");
                    sb.Append(_obj.Valore);
                    sb.Append("' e creato il ");
                    sb.Append(_obj.Creato.ToString("yyyyMMdd"));
                    var stringa = sb.ToString();
                    Assert.Equal(ValoreAtteso, stringa);
            }
        }
}
```

Concatenazione con operatore “+”

Proprio la complessità nel far emergere l’intento porta molti programmatori ad usare la concatenazione tramite **operatore +**

```
[Fact]
public void ConcatenazioneTest()
{
        var stringa = "Ciao sono un oggetto con Id:" + _obj.Id + ", Valore:'" +
_obj.Valore + "' e creato il " + _obj.Creato.ToString("yyyyMMdd");
        Assert.Equal(ValoreAtteso, stringa);
}
```

È evidente come sia più leggibile, ma **la concatenazione non è performante** poiché ad ogni operazione di + vengono unite due oggetti copiandoli in un nuovo oggetto stringa, con la conseguente proliferazione di oggetti in memoria. Questo non è buono per la memoria e non performante.

string.Format

Lo **string.Format** unisce le due esigenze precedenti con qualche compromesso. Usa al suo interno lo `string.builder`, in particolare il metodo `appendformat` con un piccolo overhead, ma mantiene comunque ottime performance di memoria.

```
[Fact]
public void StringFormatTest()
{
        var stringa = string.Format(
                "Ciao sono un oggetto con Id:{0}, Valore:'{1}' e creato il {2:yyyyMMdd}",
                _obj.Id,
                _obj.Valore,
                _obj.Creato);
```

```
        Assert.Equal(ValoreAtteso, stringa);
}
```

Tuttavia già con tre parametri la lettura potrebbe essere non immediata, poiché con lo sguardo per ogni parametro trovato in stringa dobbiamo saltare al valore associato per capire cosa ci verrà scritto. Tale problema è ancora maggiore se l'ordine dei parametri non è lineare (causa evoluzione del codice).

String Interpolation

Per soddisfare l'esigenza della manutenzione nascono le **stringhe interpolate** (string interpolation) che evolvono quanto c'è di buono nello *string.Format* in un metodo ancora più semplice e condensato. Ecco un esempio:

```
var stringa = $"Ciao {nome_utente}, come stai?";
```

La sintassi è:

```
$" <testo> { <espressione> [,<larghezza-da-rispettare>] [:<formato>] } <testo> {... } "
```

Argomento	Descrizione
testo	È il testo che precede o segue i valori interpolati.
espressione	È l'espressione che ritornerà il valore da inserire nella stringa.
larghezza-da-rispettare	[opzionale] numero di caratteri. Serve se vogliamo che la porzione interpolata della stringa rispetti una certa dimensione, vengono aggiunti degli spazi quando non si raggiunge la dimensione desiderata.
formato	[opzionale] serve a introdurre un formato al valore calcolato. Ad es. per un avere solo i primi due decimali di un numero aggiungiamo `:F2`.

È da preferire perché riduce gli errori tipici che si commettono col metodo *string.Format*, come l'ordine sbagliato degli argomenti o l'assenza di uno o più parametri.

```
[Fact]
public void StringInterpolationTest()
{
        var stringa = $"Ciao sono un oggetto con Id:{_obj.Id}, Valore:'{_obj.Valore}' e
creato il {_obj.Creato:yyyyMMdd}";
        Assert.Equal(ValoreAtteso, stringa);
}
```

La string interpolation permette anche di condensare del **codice condizionale :**

```
[Fact]
public void StringInterpolationPlusTest()
{
        var stringa = $"Ciao sono un oggetto con Id:{_obj.Id}, Valore:'{_obj.Valore}' e creato il {_obj.Creato:yyyyMMdd}{(_obj.Creato.Year>=2016?" quindi attuale" : "")}";
        Assert.Equal(ValoreAttesoCondizionale, stringa);
}
```

Di fatto **la string interpolation viene modificata a compile time in una chiamata *string.Format*** equivalente, quindi non è una feature del framework, ma del linguaggio ed è per questo legato alla versione del IDE (Visual Studio 2015) e non di un particolare versione del framework. Quindi occorre attenzione nei team in cui alcuni hanno la versione 2015 e altri 2013, il rischio è non far buildare i colleghi.