

## Programmare con C# 8 - Errata Corrigere ☹️

In questa sezione sono riportate delle **correzioni** sulle varie parti del libro o **precisazioni** ove si rendessero necessarie. Le pagine indicate si riferiscono alla versione cartacea del libro.

Ringrazio tutti i lettori che hanno segnalato gli errori, che mi hanno posto domande o che mi hanno inviato i loro suggerimenti e critiche, via mail (info at antoniopelleriti.it) o sulla pagina facebook dedicata al libro <https://www.facebook.com/programmare.con.csharp>.

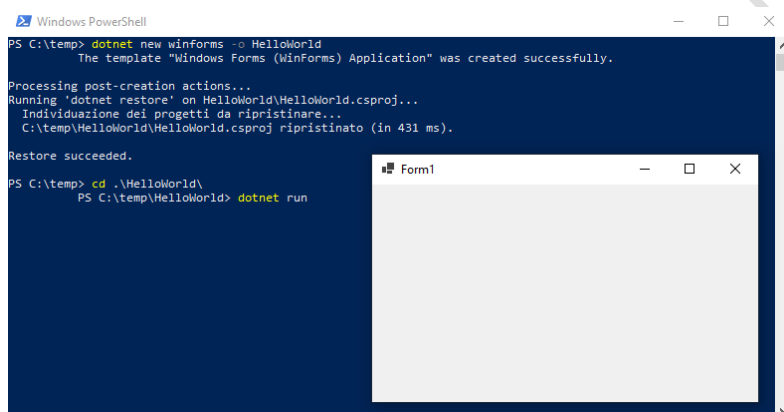
### Capitolo 2

#### Pag. 34 - creazione di un'applicazione Windows Forms

Il template `windows` utilizzato con il comando `dotnet` non è più supportato (lo era nella preview di .NET Core 3.0). È ora necessario utilizzare il template `winforms`.

```
C:\> dotnet new winforms -o HelloWorld
```

Anche la figura 2.4 mostra un risultato obsoleto. Il codice generato con la versione definitiva crea una finestra vuota, come in quella seguente:



### Capitolo 3

#### Pag. 92 - paragrafo Tipi a virgola mobile

Il nome corretto del tipo è **float** e non **flat**:

**Tabella 3.2** - Tipi a virgola mobile predefiniti di C#.

Nome	Tipo CTS	Descrizione	Cifre decimali
float	System.Single	32 bit a singola precisione	7 cifre
double	System.Double	64 bit a doppia precisione	circa 15 cifre

Il tipo **float** è un tipo a singola precisione e permette di rappresentare numeri con circa 7 cifre decimali, mentre il numero di tipo **double** può rappresentare circa 15 cifre decimali. Per assegnare un valore numerico con la virgola si usa una rappresentazione con

## Programmare con C# 8 - Errata Corrigere ☹️

### Capitolo 4

#### *Pag. 137 - MinValue e MaxValue*

`MinValue` e `MaxValue` sono in realtà dei campi costanti e non proprietà. Campi costanti però implica che siano anche statici, quindi non è necessario una istanza per leggerne i valori, ma deve essere utilizzato il nome della classe.

#### *Pag. 155 - paragrafo Flag di bit*

L'esempio di enum denominata `GiorniSettimana` per il membro `Domenica` riporta il valore 128, mentre quello corretto è 64.

Il bit pari a 1 è infatti il settimo e non l'ottavo.

Il codice corretto è quindi:

```
[Flags]
enum GiorniSettimana
{
    Lunedì = 1,      //00000001
    Martedì = 2,      //00000010
    Mercoledì = 4,    //00000100
    ...
    Domenica = 64, //01000000
}
```

### Capitolo 5

#### *Pag. 193 - esempio operatore XOR ^*

Nell'esempio di utilizzo dell'operatore di OR esclusivo ^ viene utilizzato l'operatore OR |.

```
z = (byte)(x|y); // 0000 1100
Console.WriteLine(z); // = 12
```

L'esempio corretto è:

```
z = (byte)(x^y); // 0000 1100
```

Il risultato 12 è invece quello esatto.

### Capitolo 8

#### *Pag. 356 - nome del metodo*

In fondo alla pagina ci si riferisce a un metodo `Start`, mentre il nome corretto è `Print`.

### Capitolo 9

#### *Pag. 396 - nota duplicata*

## Programmare con C# 8 - Errata Corrigi ☹️

Sono riportate due note identiche. La seconda nota corretta è:

NOTA: Il CLR consente di lanciare come eccezione un qualunque oggetto di una qualunque classe, anche un `Int32` o una `string`. Microsoft ha però deciso che all'interno di un linguaggio di programmazione che rispetti le regole CLS (Common Language Specification) di interoperabilità, ogni eccezione venga derivata da `System.Exception`.

### Capitolo 10

#### *Pag. 416 - codice esempio*

Nel secondo esempio della pagina, il tipo `T` nella riga seguente:

```
T temp = left;
```

Deve essere corretto con `U`:

```
U temp = left;
```

#### *Pag. 436 - interfaccia `IEnumerable<out T>`*

Parlando dell'Interfaccia `IEnumerable<out T>` si afferma che la parola chiave `out` indica che l'interfaccia è **controvariante**. L'affermazione corretta è invece che l'interfaccia è **covariante**.

#### *Pag. 444 - esempio `Array.Sort`*

L'esempio di utilizzo del metodo `Array.Sort` riportato sotto è errato:

```
var vettoreOrdinato=Array.Sort(vettore);
```

Il metodo non restituisce un array ordinato, ma agisce direttamente sul parametro. Quindi va usato come segue:

```
Array.Sort(vettore);
```

L'array `vettore` a questo punto sarà ordinato.

#### *Pag. 446 - esempio `Predicate<T>`*

L'esempio di utilizzo del delegate `Predicate<T>` riportato sotto è errato:

```
public static void IsPari(int x)
{
    return x%2==0;
}
```

In quanto deve restituire **bool** e non **void**.

### Capitolo 11

#### *Pag. 488 - esempi delegate generici*

## Programmare con C# 8 - Errata Corrigi ☹️

Gli esempi dei due delegate generici riportano i tipi nell'ordine invertito. Quelli corretti devono essere:

```
ConvertOriginToDest<string, int> isconvert = IntToString;
```

e il secondo

```
ConvertOriginToDest<int, string> siconvert = StringToInt;
```

Sempre a pag. 488, l'ultimo esempio di delegate con due vincoli deve essere scritto così:

```
public delegate TDest ConvertOriginToDest<TDest, UOrig>(UOrig val) where TDest:struct where UOrig:struct
```

### *Pag. 490 - esempio metodo generico errato*

La firma del seguente metodo `Evaluate` utilizzato come esempio è errata:

```
public T Evaluate(Func func<T,T>, T val)
```

Quella corretta è:

```
public T Evaluate<T>(Func<T, T> func, T val)
```

## Capitolo 12

### *Pag.525 - risultato esempio*

Nel secondo esempio della pagina il risultato è 6 e 36, non 4 e 16

## Capitolo 12

### *Pag.532 - nota*

Nel secondo esempio della pagina il risultato è 6 e 36, non 4 e 16

## Appendice A

### *Pag. 826 - paragrafo Costruzione di Stringhe*

La seguente affermazione:

Non esiste un costruttore a cui passare la stringa come argomento, quindi non è possibile utilizzare in tal caso l'operatore `new`:

```
string str = new string("hello world"); //ERRORE
```

non è più vera a partire da C# 7.2. Infatti con l'introduzione del tipo `ReadOnlySpan<T>` e alla conversione implicita di `string` in `ReadOnlySpan<char>`, è stato aggiunto anche il costruttore di `string` seguente:

```
string(ReadOnlySpan<char>)
```

per cui nell'esempio

```
string str = new string("hello world");
```

## Programmare con C# 8 - Errata Corrigere ☹️

La stringa "hello world" viene convertita implicitamente in `ReadOnlySpan<char>` e poi utilizzato il costruttore suddetto.

### *Pag. 826 - paragrafo Costruzione di Stringhe*

Nell'esempio seguente, l'istruzione non ha i doppi apici finali e non è chiusa dal punto e virgola:

```
string str=@"Questa stringa è fra doppi apici" // "Questa stringa è fra doppi apici"
```

L'esempio corretto è

```
string str=@"Questa stringa è fra doppi apici""; // "Questa stringa è fra doppi apici"
```

### *Pag. 832 - paragrafo Confronto di Stringhe*

Nell'esempio finale del paragrafo, i metodi `CompareOrdinal` e `Compare` non restituiscono un valore booleano, ma un intero.

Per cui l'esempio corretto è il seguente:

```
int i = String.CompareOrdinal("Strass", "Straß");//restituisce un numero minore di zero  
i = String.Compare("Strass", "Straß");//restituisce 0, le stringhe sono equivalenti
```