What’s new in C# 8

Matteo Tumiati

Windows Dev MVP

Content Manager WinRT/DevOps @aspitalia

matteot@aspitalia.com | @xtumiox

aspitalia.com

Agenda

- C# 7 e C# 8
- Demo

.NET Framework 4.8

Controlli moderni per browser (Edge/IE) e media

Migliorie touch per WinForm e WPF

Miglioramenti per DPI con supporto a 4K e 8K

Jitter x64 di .NET Core 2.1

Supporto agli ultimi protocolli di rete e standard di sicurezza

Non c'è fretta per migrare

- Ma rinunciate a qualcosa
- Tutto ciò che è .NET Standard 2.0, funziona

.NET Standard 2.1

Inclusione di Span<T>

Aggiunte minori

- Reflection emit
- Compressione Brotli per HttpClient
- System.IO

98% completato

- https://github.com/dotnet/standard/milestone/3

E dopo .NET Standard? .NET 5!

C# 8

Nullable reference type

Default interface methods

Recursive pattern

Relax ordering ref e partial modifiers

Expression variables

Range e Index

Async stream e disposable

Platform dependency

C# 8 è supportato solo sulle piattaforme .NET Standard 2.1

- .NET Framework non lo supporterà
- .NET Core 3.0 (Xamarin, Unity e Mono) implementerà .NET Standard 2.1

C# 8 può funzionare su .NET Framework? Nì…

- Non c'è il supporto, ma non vuol dire che non si possa abilitare
- Microsoft lo disabilita dalla UI ma si può cambiare da csproj
- Solo alcune funzionalità sono portate, come ValueTuple o estensioni distribute tramite pacchetti di NuGet
- async streams, indexer, range e una parte dei nullable non ci sono…

Info: https://stackoverflow.com/questions/56651472/does-c-sharp-8-support-the-net-framework

Nullable reference types

```
string? name;

string s = null; // warning when nullable warning context is enabled
var txt = s.ToString(); // warning when nullable warnings context is safeonly, or enabled
```

Utile per lavorare e verificare tipi che potrebbero avere un valore null a runtime
Va abilitato esplicitamente
Per casi eccezionali può essere abbinato al null-forgiving operator «!»

https://docs.microsoft.com/en-us/dotnet/csharp/nullable-references

Default interface member

```
public interface ICustomer
{
  IEnumerable<IOrder> PreviousOrders { get; }

  DateTime DateJoined { get; }
  DateTime? LastOrder { get; }
  string Name { get; }
  IDictionary<DateTime, string> Reminders { get; }

  public decimal ComputeLoyaltyDiscount()
  {
    DateTime TwoYearsAgo = DateTime.Now.AddYears(-2);
    if ((DateJoined < TwoYearsAgo) && (PreviousOrders.Count() > 10))
      return 0.10m;

    return 0;
  }
}
```

Un’interfaccia può avere un’implementazione predefinita

Utile nei casi in cui ci sono molte implementazioni, ma non si vuole (o non si può) accedere a tutte

Readonly members

```
public struct Point
{
    public double X { get; set; }
    public double Y { get; set; }
    public readonly double Distance => Math.Sqrt(X * X + Y * Y);

    public readonly override string ToString() =>
        $"({X}, {Y}) is {Distance} from the origin";
}
```

Indica che il membro della struct non modifica lo stato

Un qualsiasi assegnamento non sarebbe permesso

Pattern matching (C# 7)

Pattern = elementi sintattici che testano un valore e lo estraggono qualora ci sia una corrispondenza

Tre tipologie di pattern in C# 7:

- Costanti, tipi, var

Pattern applicati a:

- Espressioni “*is*”
- Blocchi “*case*”

```
public void PrintStars(object o)
{
    if (o is null)
        return; // constant pattern "null"
    if (!(o is int i))
        return; // type pattern "int i"

    WriteLine(new string('*', i));
}
```

Pattern matching (C# 7)

Pattern = elementi sintattici che testano un valore e lo estraggono qualora ci sia una corrispondenza

Tre tipologie di pattern in C# 7:

- Costanti, tipi, var

Pattern applicati a:

- Espressioni “*is*”
- Blocchi “*case*”

```
switch(shape)
{
    case Circle c:
        WriteLine($"circle with radius {c.Radius}");
        break;
    case Rectangle s when (s.Length == s.Height):
        WriteLine($"{s.Length} x {s.Height} square");
        break;
    case Rectangle r:
        WriteLine($"{r.Length} x {r.Height} rectangle");
        break;
    default:
        WriteLine("<unknown shape>");
        break;
    case null:
        throw new ArgumentNullException(nameof(shape));
}
```

Pattern matching con switch in C# 8

```
public static RGBColor FromRainbow(Rainbow colorBand) =>
    colorBand switch
    {
        Rainbow.Red    => new RGBColor(0xFF, 0x00, 0x00),
        Rainbow.Orange => new RGBColor(0xFF, 0x7F, 0x00),
        Rainbow.Yellow => new RGBColor(0xFF, 0xFF, 0x00),
        Rainbow.Green  => new RGBColor(0x00, 0xFF, 0x00),
        Rainbow.Blue   => new RGBColor(0x00, 0x00, 0xFF),
        Rainbow.Indigo => new RGBColor(0x4B, 0x00, 0x82),
        Rainbow.Violet => new RGBColor(0x94, 0x00, 0xD3),
        _              => throw new ArgumentException("invalid enum value", nameof(colorBand))
    };
```

Medesima funzionalità, ma in versione più compatta

Property pattern

```
public static decimal ComputeSalesTax(Address location, decimal salePrice) =>
    location switch
    {
        { State: "WA" } => salePrice * 0.06M,
        { State: "MN" } => salePrice * 0.75M,
        { State: "MI" } => salePrice * 0.05M,
        // other cases removed for brevity...
        _ => 0M
    };
```

L’obiettivo è sempre scrivere codice compatto

Viene valutata la proprietà *State* della classe *Address*, ma possono essere combinate più proprietà con il Tuple pattern

Tuple Pattern

```
public static string RockPaperScissors(string first, string second)
    => (first, second) switch
    {
        ("rock", "paper") => "rock is covered by paper. Paper wins.",
        ("rock", "scissors") => "rock breaks scissors. Rock wins.",
        ("paper", "rock") => "paper covers rock. Paper wins.",
        ("paper", "scissors") => "paper is cut by scissors. Scissors wins.",
        ("scissors", "rock") => "scissors is broken by rock. Rock wins.",
        ("scissors", "paper") => "scissors cuts paper. Scissors wins.",
        (_, _) => "tie"
    };
```

Utile per gestire combinazioni
In questo caso, il vincitore è associato ai due stati della tupla

Il discard _ è utilizzato per gestire gli altri casi (es: testo differente, pareggi, etc)

Positional Pattern

```
public class Point
{
    public int X { get; }
    public int Y { get; }

    public Point(int x, int y) => (X, Y) = (x, y);
    public void Deconstruct(out int x, out int y) => (x, y) = (X, Y);
}
```

```
static Quadrant GetQuadrant(Point point) => point switch
{
    (0, 0) => Quadrant.Origin,
    var (x, y) when x > 0 && y > 0 => Quadrant.One,
    var (x, y) when x < 0 && y > 0 => Quadrant.Two,
    var (x, y) when x < 0 && y < 0 => Quadrant.Three,
    var (x, y) when x > 0 && y < 0 => Quadrant.Four,
    var (_, _) => Quadrant.OnBorder,
    _ => Quadrant.Unknown
};
```

Abbinato al Deconstructor, consente di scomporre una classe in valori primitivi, più facili da gestire

Using declaration

```
static void WriteLinesToFile(IEnumerable<string> lines)
{
    using var file = new System.IO.StreamWriter("WriteLines2.txt");
    foreach (string line in lines)
    {
        // If the line doesn't contain the word 'Second', write the line to the file.
        if (!line.Contains("Second"))
        {
            file.WriteLine(line);
        }
    }
    // file is disposed here
}
```

Versione semplificata del costrutto using

Static local functions

```
int M()
{
    int y = 5;
    int x = 7;
    return Add(x, y);

    static int Add(int left, int right) => left + right;
}
```

Utile per evitare che (inavvertitamente) si catturi una variabile locale dentro la local function

In caso di riferimento a *x* o *y*, ci sarà un errore di compilazione

Stream async

```
public static async System.Collections.Generic.IAsyncEnumerable<int> GenerateSequence()
{
    for (int i = 0; i < 20; i++)
    {
        await Task.Delay(100);
        yield return i;
    }
}
```

```
await foreach (var number in GenerateSequence())
{
    Console.WriteLine(number);
}
```

Si possono consumare e produrre stream async grazie a *IAsyncEnumerable<T>*

Non si aspetta l'arrivo di tutta la sequenza per continuare l'esecuzione

Index e Range

```
var words = new [] {"A", "B", "C", "D" };

var x = words[^1];      // l'ultimo

var y = words[^2..^0]; // dal terz'ultimo all'utimo (non inclusi)

var k0 = words[1..3];  // dal secondo al quarto
var k1 = words[..];    // tutti
var k2 = words[..2];   // fino al terzo
var k3 = words[2..];   // dal terzo in poi

// come variabile
Range phrase = 1..4;

// passata poi come range
var text = word[phrase];
```

^ indica un indice relativo alla fine (esclusivo)

.. indica un intervallo

Demo

E’ tutto qui?

Principalmente il focus è sulla produttività, ma…

Null coalescing assignment

Unmanaged constraint

Interpolated verbatim strings

E dopo C# 8? C# 9!

Proposal https://github.com/dotnet/csharplang/milestone/15

Target-typed new

Record

Static delegate

defer

```
Person p = new ();
Person p1 = new ("Mario", "Rossi")
```

```
class Point {
public readonly int X;
public readonly int Y;

public Point With(int x = this.X, int y = this.Y)
          => new Point(x, y);
}
```

```
var resource = AcquireResource();
// operations here…
defer resource.Dispose();
```

```
Point p = new Point(3, 4); p = p.With(x: 1);
```

C# 10?

Proposal https://github.com/dotnet/csharplang/milestone/16

Exponential operator

Type Classes (o Generic Constraints)

```
public static long operator ** (int value1, int value2) =>
       Math.Pow(value1, value2);
```

Grazie!

@xtumiox
matteot@aspitalia.com

Materiale su
http://aspit.co/devday-19