C# 8 Feature Cheat Sheet

Default Interface Methods

Allows you to add new functionality to your interfaces of your libraries and ensure the backward compatibility with code written for older versions of those interfaces.

```
interface IWriteLine
 {
  public void WriteLine()
   {
    Console.WriteLine("Wow C# 8!");
   }
 }
```

Nullable Reference Types

Allows you will get a compiler error or warning, if a variable that may not be null is assigned null.

```
 string? nullableString = null;
  // WARNING: may be null! Take care!
Console.WriteLine(nullableString.Length)
```

Pattern Matching

Provides the ability to deconstruct matched objects, giving you access to parts of their data structures. C# offers a rich set of patterns that can be used for matching:

- Switch expressions
- Property patterns
- Tuple patterns
- Positional patterns

```
static bool Positive(Point p) => p switch
{
    (0, 0) => true,
   (var x, var y) when x > 0 && y > 0 => true,
    _ => false
};
```

Asynchronous Streams

Allows to have enumerators that support async operations.

```
 await foreach (var x in enumerable)
 {
   Console.WriteLine(x);
 }
```

Indices and Ranges

Allows you to use more natural syntax for specifying subranges in an array or a collection.

```
int[] a = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

Index: Used to obtain the collection from the beginning or from the end.

```
  // number 4 from end of the collection
 Index i2 = ^4;
  Console.WriteLine($"{a[i2]}"); // "6"
```

Range: Access a sub-collection(slice) from a collection.

```
 var slice = a[i1..i2];
  // { 3, 4, 5 }
```

Caller Expression Attribute

Allows callees to 'stringify' the expressions passed in at a call

```
 Verify.InRange(index, 0, array.Length - 1);
    // Error message by wrong Index:
   // "index (-1) cannot be less than 0 (0).", or
   // "index (6) cannot be greater than - 1 (5)."
```

Static Local Functions

Allows you to add the 'static' modifier to the local functions.

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

 static int Add(int f, int s) => f + s;
}
```

Using Declarations

Enhances the 'using' operator to use with Patterns and make it more natural.

```
using var repository = new Repository();
Console.WriteLine(repository.First());
// repository is disposed here!
```

Generic Attributes

Allows the generic type in the C# 'Attributes'.

```
class GenericAttribute<T> : Attribute{}
```

Default in Deconstruction

Allows the following syntax:

```
(int x, string y) = (default, default);//C# 7
(int x, string y) = default;        //C# 8
```

Relax Ordering of ref and partial Modifiers

Allows the partial keyword before ref in the class definition.

```
public ref partial class { } // C# 7
public partial ref class { } // C# 8
```

Null Coalescing Assignment

Simplifies a common coding pattern where a variable is assigned a value if it is null.
It is common to see the code of the form:

```
if (variable == null)
{
  variable = expression;  // C#  1..7
}
variable ??= expression; // C# 8
```

Alternative Interpolated Verbatim Strings

Allows `@$""` as a verbatim interpolated string,

```
var file = @$"c:\temp\{filename}";//C# 8
```

Disposable ref structs

Allows you to use the 'using' pattern with `ref struct` or `readonly ref struct`'.

```
 // Pattern-based using for ref struct
ref struct Test {
  public void Dispose() {}
}

using var local = new Test();
// local is disposed here!
```

About me

Bassam Alugili
Senior Software Specialist/Database Expert at STRATEC Biomedical AG
alugili@gmail.com
github.com/alugili

www.infoq.com/profile/Bassam-Alugili

02.04.2019