Equality and Comparisons for Strings

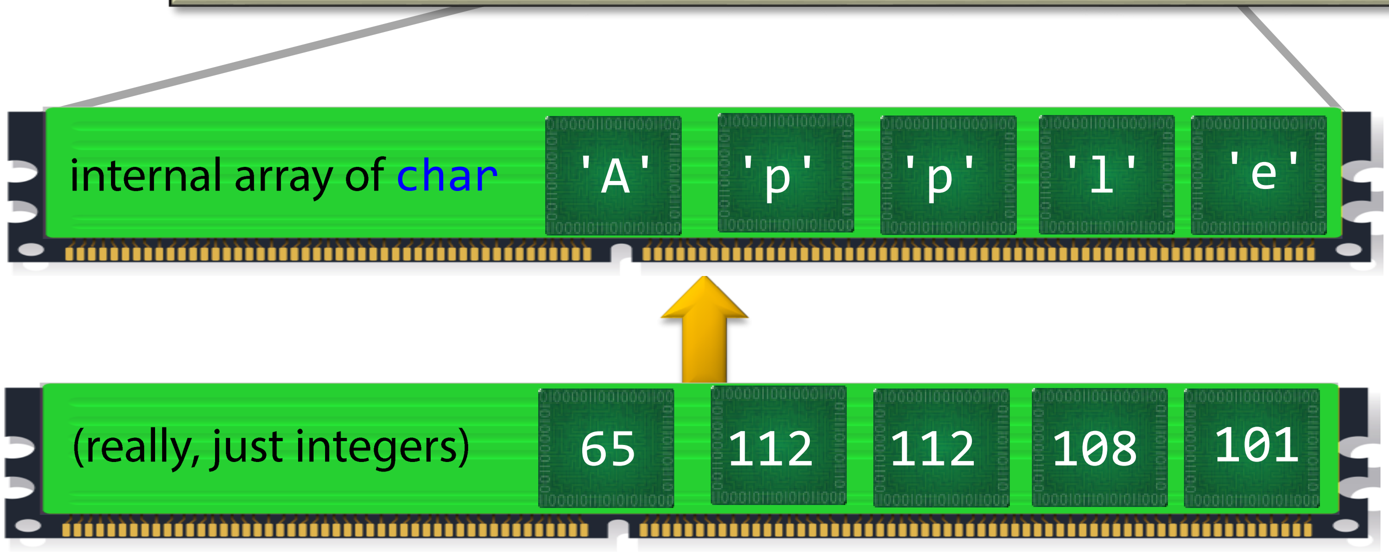
Simon Robinson
http://TechieSimon.com
@TechieSimon

hardcore dev and IT training

- What is in a `char` (not always one character)?
- Issues when comparing strings.
  - Case.
  - Culture.
- Code demos of different types of *string comparisons.*
  - When to use each comparison type.
- What do standard techniques do?
  - `object.Equals()`, `==`, `IComparable<T>`, etc.
- String pooling.

Strings are Composed of Characters

```
string apple = "Apple";
```



16 bit integer, unsigned: Range is 0 to 65535

Mapping Char Values

Many char values represent Unicode characters

Not all!

65

112

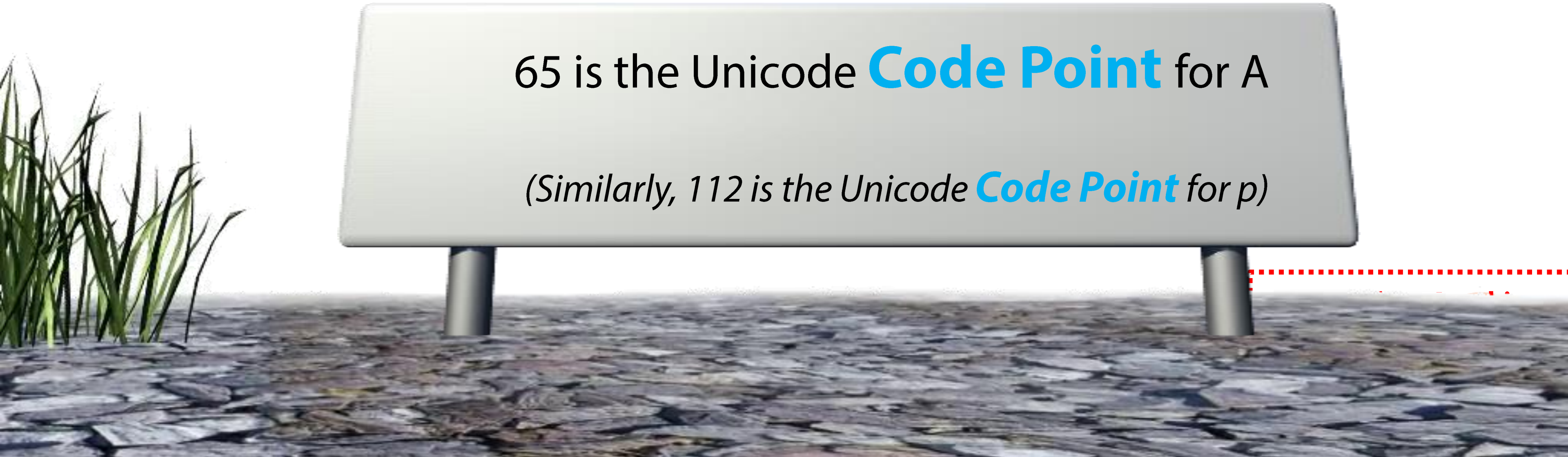
65 is the Unicode **Code Point** for A

*(Similarly, 112 is the Unicode **Code Point** for p)*

Mapping Char Values

(0x41 is hex for 65) U+0041 → A

(0x70 is hex for 112) U+0070 → p


65 is the Unicode **Code Point** for A

(Similarly, 112 is the Unicode ***Code Point*** *for p)*

Useful Code Point Ranges

(65 is U+0041)

65	66	67	68	69
A	B	C	D	E
70	71	72	73	74
F	G	H	I	J
75	76	77	78	79
K	L	M	N	O
80	81	82	83	84
P	Q	R	S	T
85	86	87	88	89
U	V	W	X	Y
90	91	92	93	94
Z	[	\	]	^

(97 is U+0061)

95	96	97	98	99
_	`	a	b	c
100	101	102	103	104
d	e	f	g	h
105	106	107	108	109
i	j	k	l	m
110	111	112	113	114
n	o	p	q	r
115	116	117	118	119
s	t	u	v	w
120	121	122	123	124
x	y	z	{	|

The ß Character

(0xDF is hex for 223)

U+00DF → ß

German eszett (=ss)

Straße = street

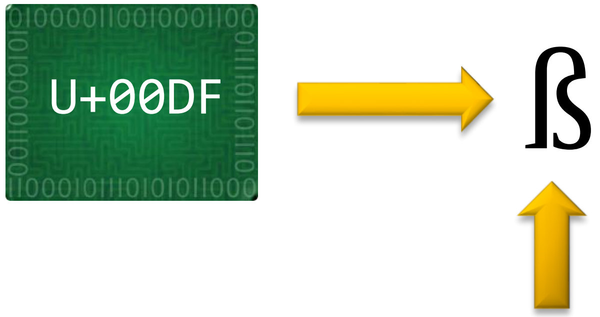
(Strasse)

Fußball = soccer

(Fussball)

The ß Character

(0xDF is hex for 223)



German eszett (=ss)

Character Expansions

When comparing strings…

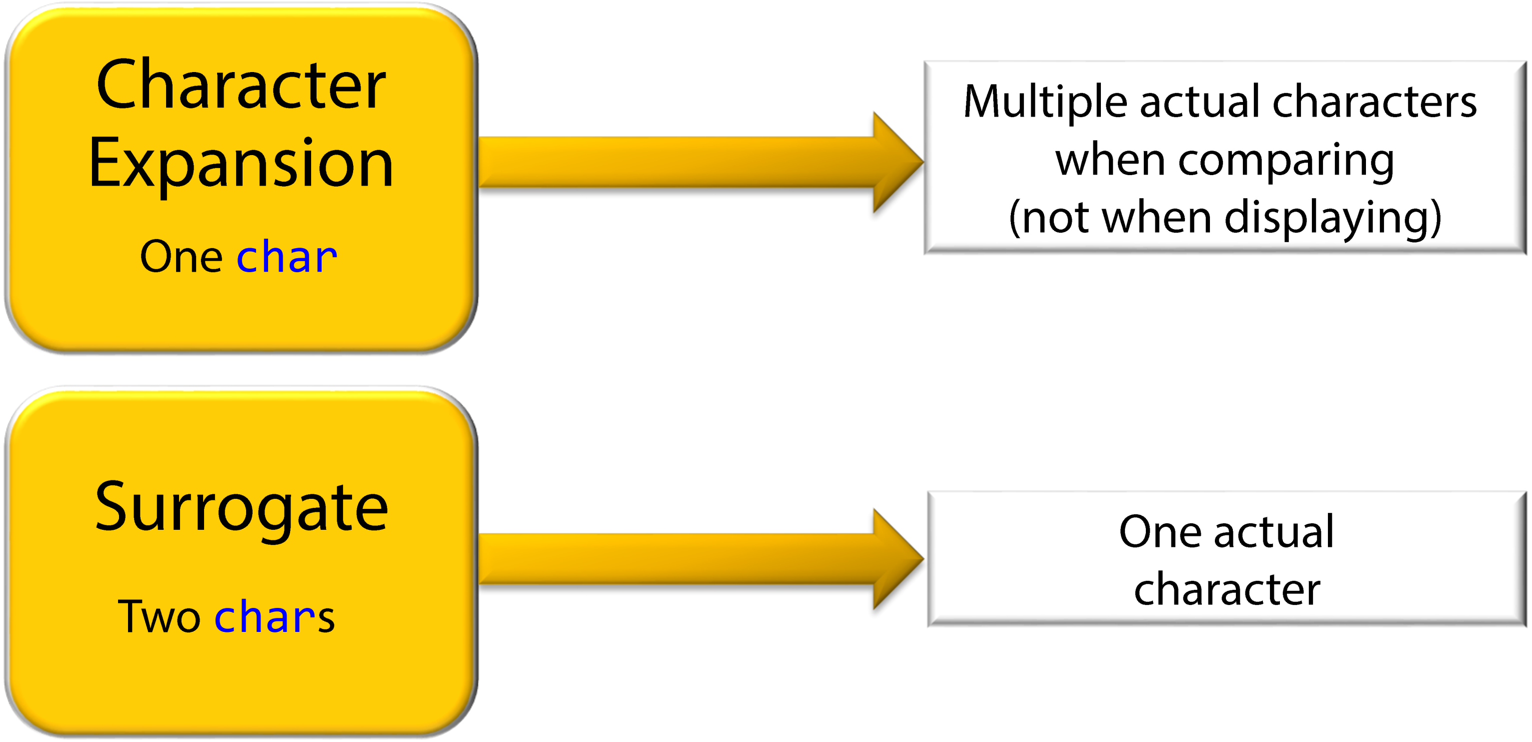
… expand character expansions first!

U+00DF → U+0073 + U+0073

ß → s + s

(Only for comparisons – not when displaying text)

Surrogates vs Expansions



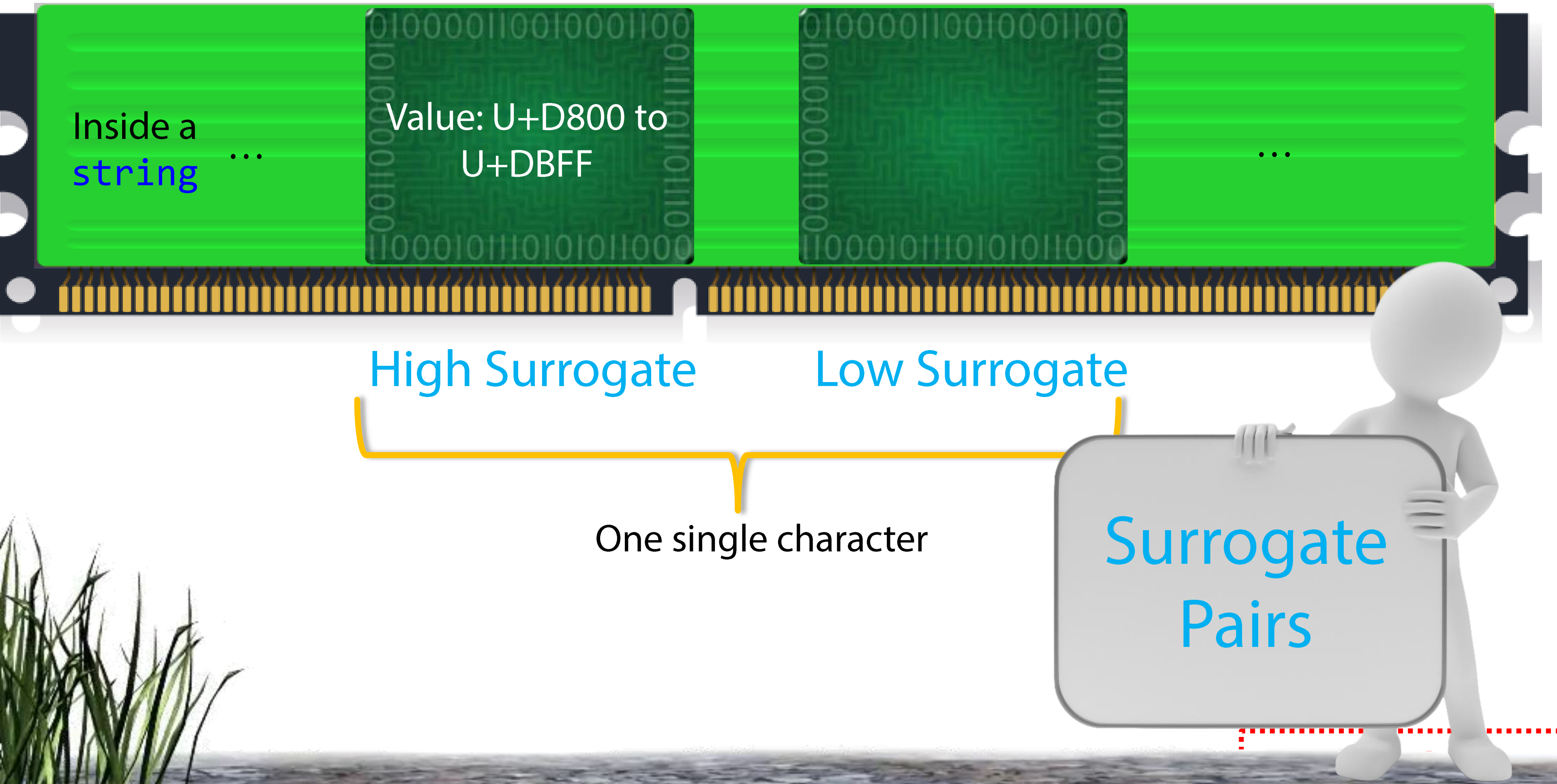
Not Enough Values

char : 16 bits	All Characters	UNICODE Code Points
0 to 65 535 (=0xFFFF)	Perhaps - millions	0 to 1 114 111 (=0x10FFFF)

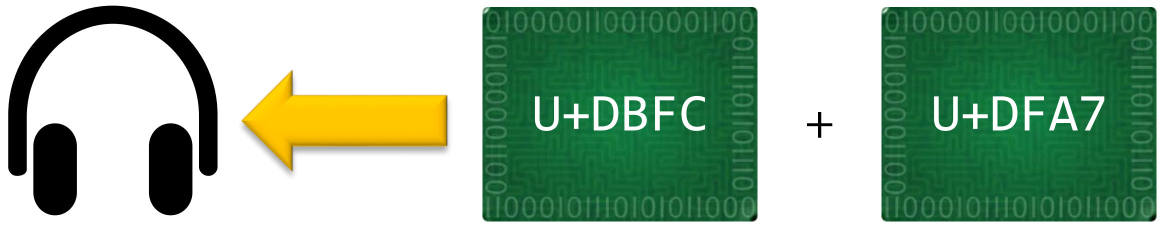
16 bits
don’t have
enough range
for all characters

Surrogates

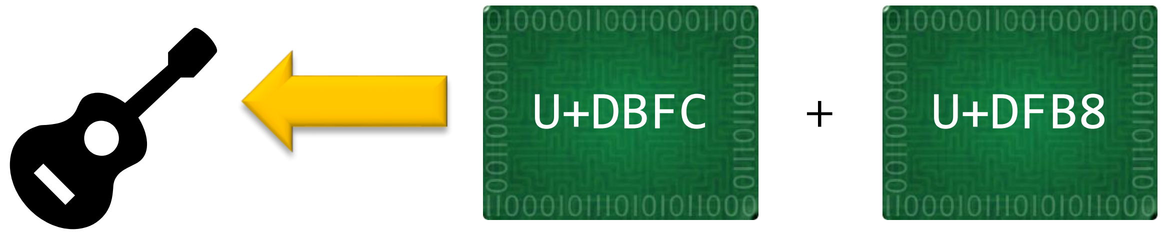
Some characters are represented by two char instances:



Surrogates: Examples



(U+1F3A7)



(U+1F3B8)

Combining Characters

Some code points modify the previous character…

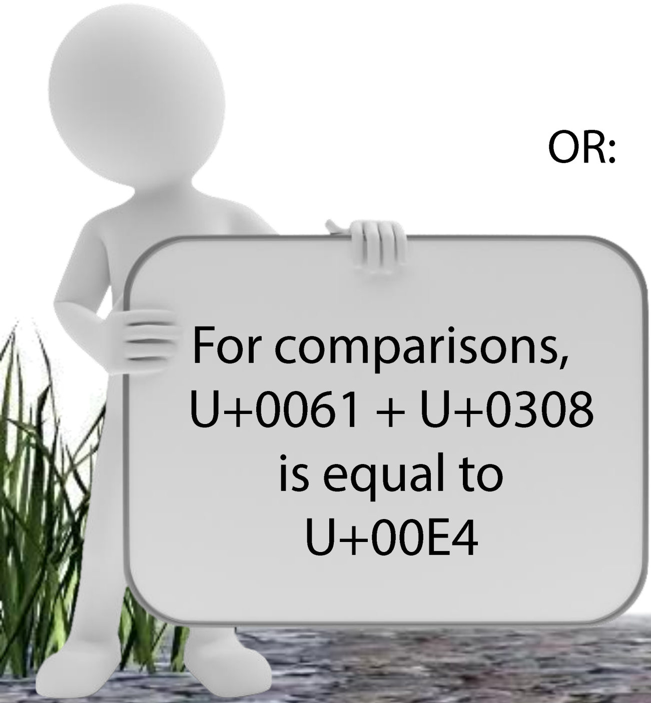
Example - the umlaut: ¨

U+00E4 = ä

OR: U+0061 + U+0308

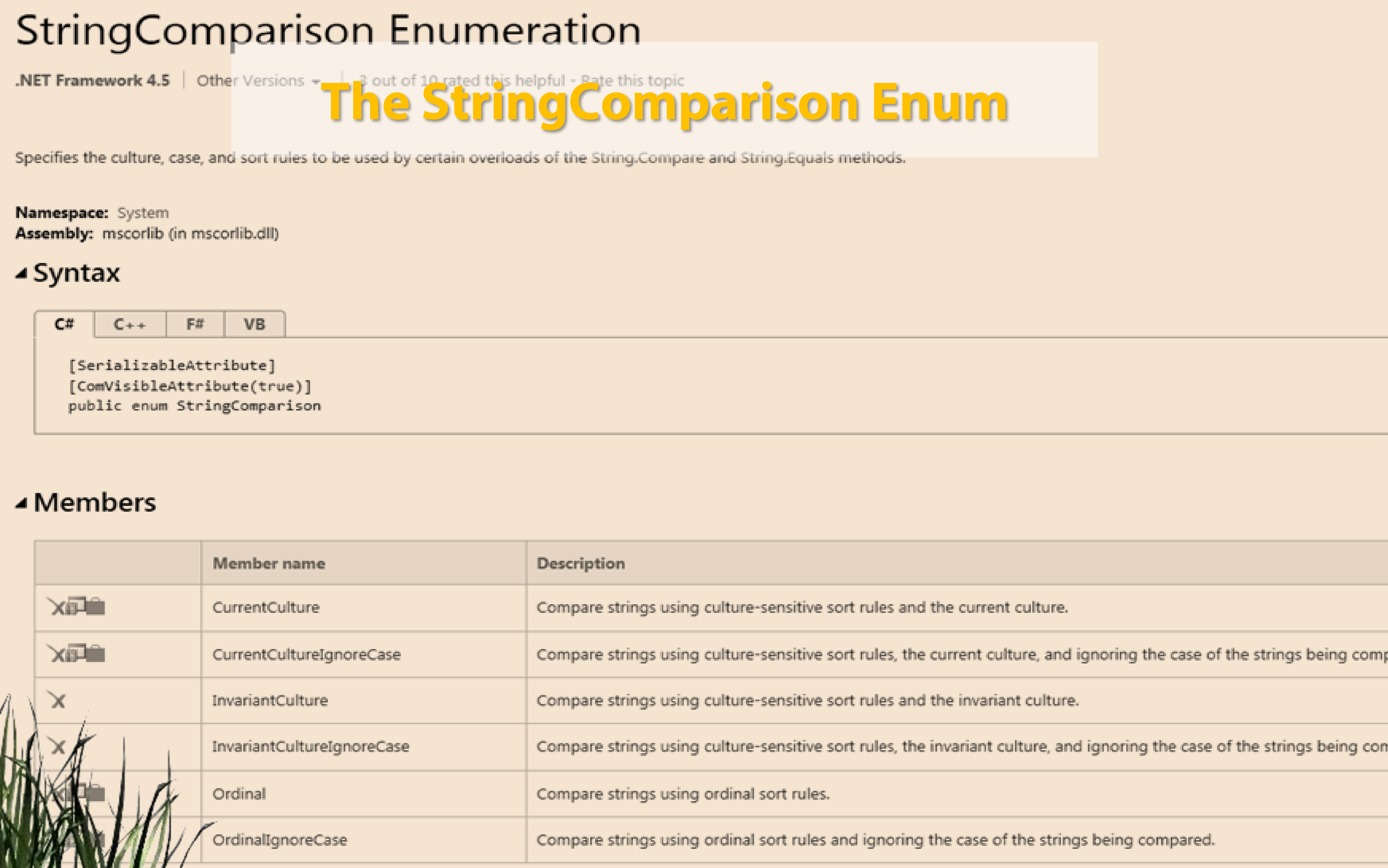
a ¨

Combining diaeresis
(an example of a
combining character)



For comparisons,
U+0061 + U+0308
is equal to
U+00E4

The StringComparison Enum



StringComparison Enumeration

.NET Framework 4.5 | Other Versions | 3 out of 10 rated this helpful - Rate this topic

Specifies the culture, case, and sort rules to be used by certain overloads of the String.Compare and String.Equals methods.

Namespace: System
Assembly: mscorlib (in mscorlib.dll)

Syntax

C# | C++ | F# | VB

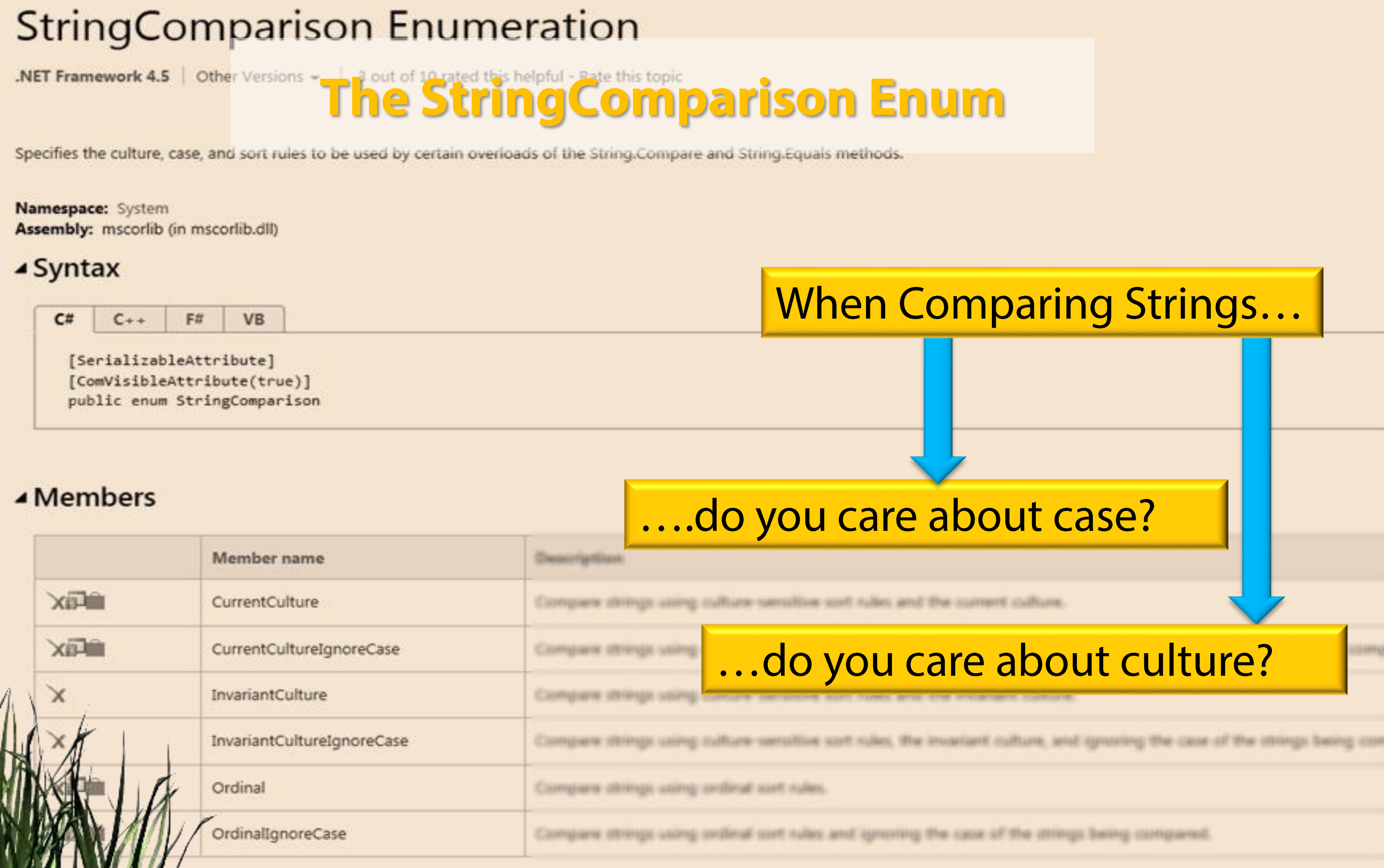
```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public enum StringComparison
```

Members

	Member name	Description
	CurrentCulture	Compare strings using culture-sensitive sort rules and the current culture.
	CurrentCultureIgnoreCase	Compare strings using culture-sensitive sort rules, the current culture, and ignoring the case of the strings being comp
	InvariantCulture	Compare strings using culture-sensitive sort rules and the invariant culture.
	InvariantCultureIgnoreCase	Compare strings using culture-sensitive sort rules, the invariant culture, and ignoring the case of the strings being com
	Ordinal	Compare strings using ordinal sort rules.
	OrdinalIgnoreCase	Compare strings using ordinal sort rules and ignoring the case of the strings being compared.

(From MSDN docs)

The StringComparison Enum



StringComparison Enumeration

.NET Framework 4.5 | Other Versions | 3 out of 10 rated this helpful - Rate this topic

Specifies the culture, case, and sort rules to be used by certain overloads of the String.Compare and String.Equals methods.

Namespace: System
Assembly: mscorlib (in mscorlib.dll)

Syntax

C# | C++ | F# | VB

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public enum StringComparison
```

Members

	Member name	Description
	CurrentCulture	Compare strings using culture-sensitive sort rules and the current culture.
	CurrentCultureIgnoreCase	Compare strings using
	InvariantCulture	Compare strings using culture-sensitive sort rules and the invariant culture.
	InvariantCultureIgnoreCase	Compare strings using culture-sensitive sort rules, the invariant culture, and ignoring the case of the strings being com
	Ordinal	Compare strings using ordinal sort rules.
	OrdinalIgnoreCase	Compare strings using ordinal sort rules and ignoring the case of the strings being compared.

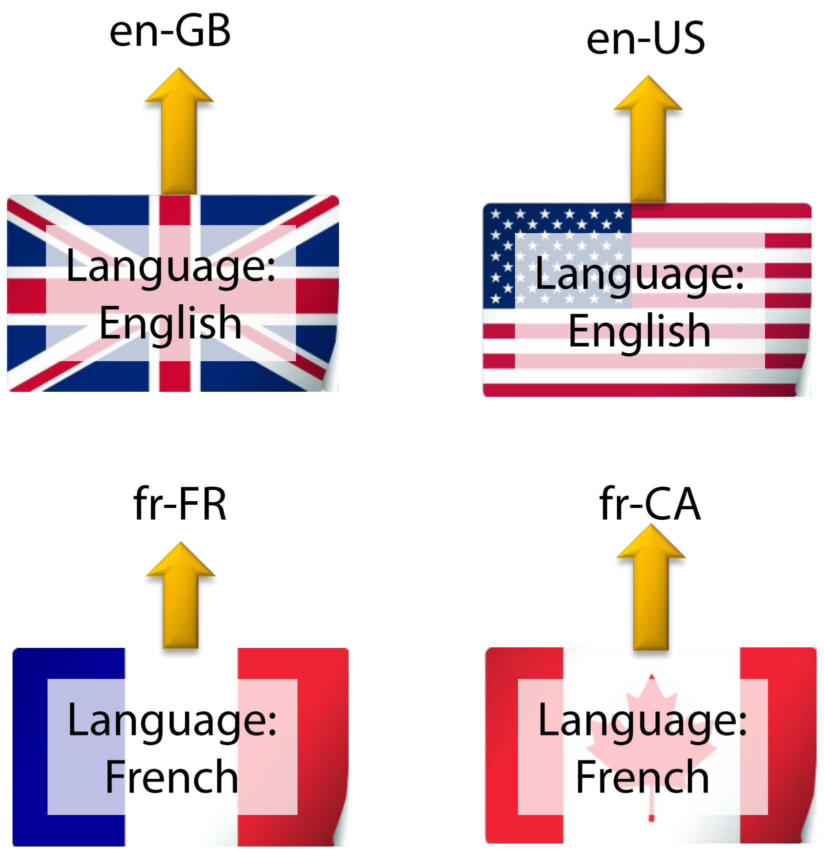
When Comparing Strings…

….do you care about case?

…do you care about culture?

(From MSDN docs)

What is Culture?

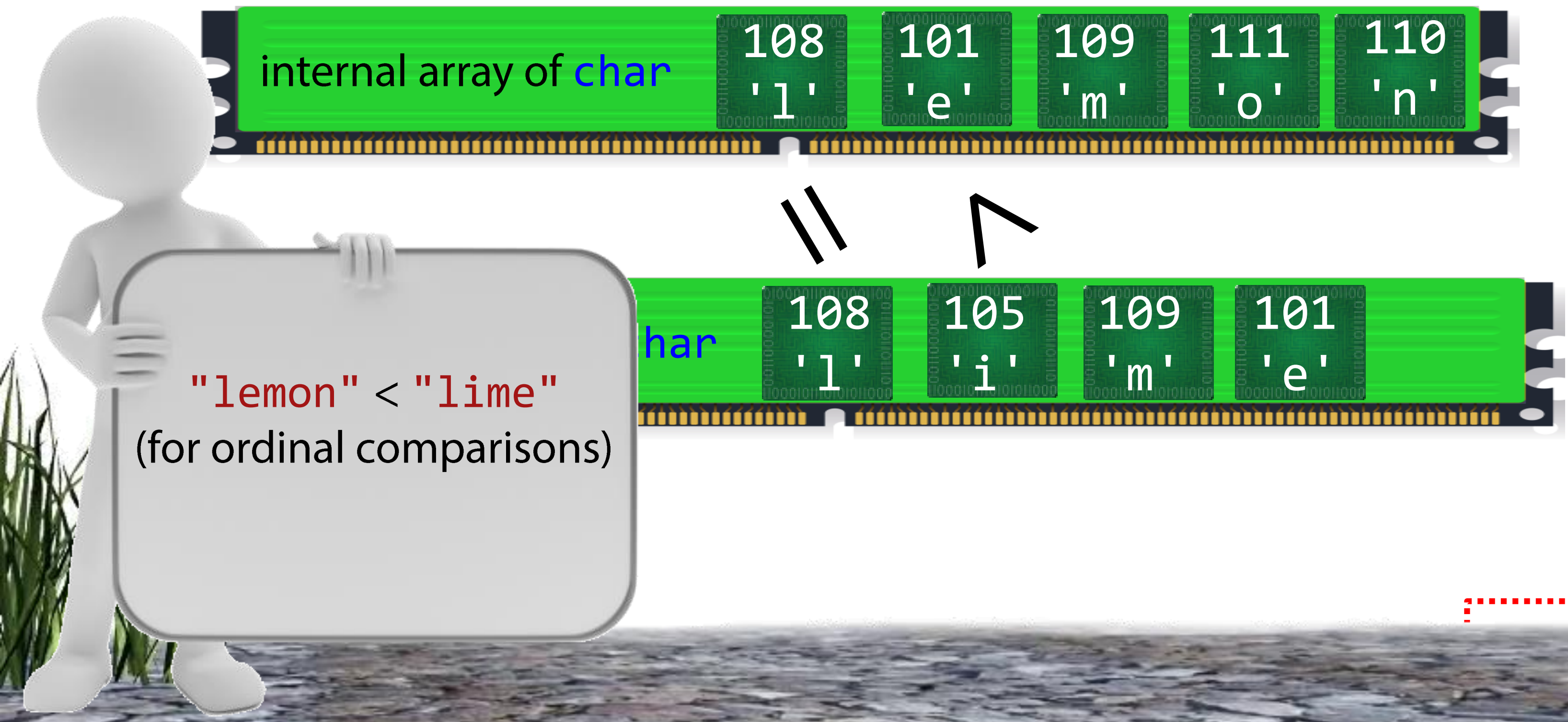


Ordinal Comparisons

Ordinal:

- Ignore the culture and Unicode issues
- Compare **numeric** values of chars only

```
string.Compare("lemon", "lime", StringComparison.Ordinal)
```

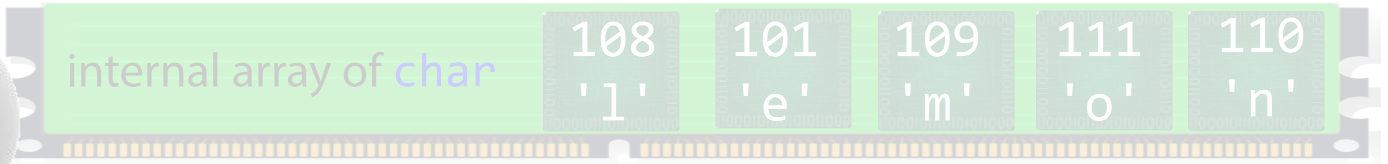


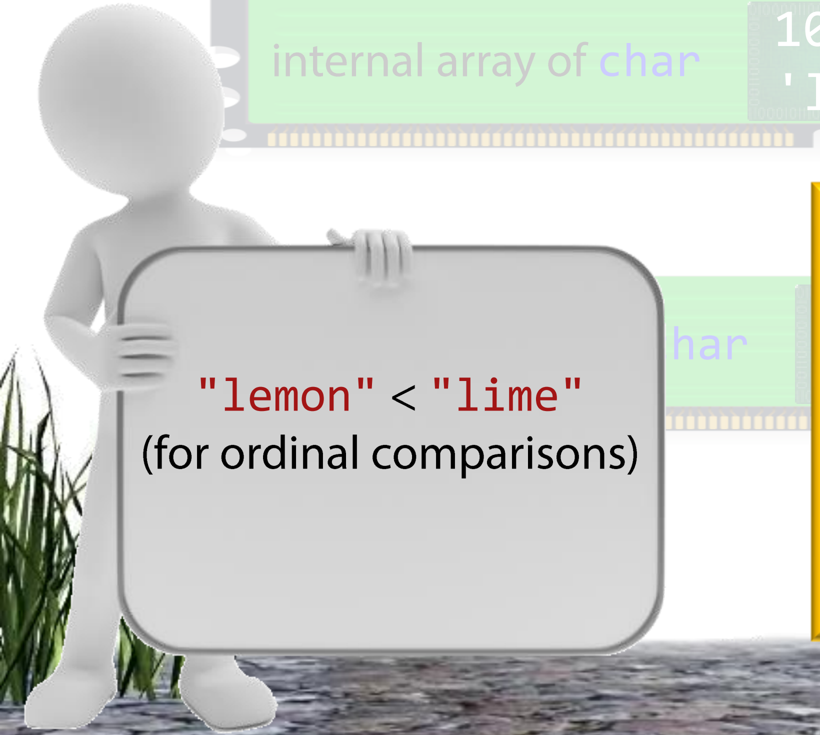
Ordinal Comparisons

Ordinal:

- Ignore the culture and Unicode issues
- Compare **numeric** values of chars only

```
string.Compare("lemon", "lime", StringComparison.Ordinal)
```





"lemon" < "lime"
(for ordinal comparisons)

```
string.Compare(x,y) …
      =0  if  x=y
      <0  if  x<y
      >0  if  x>y
```

Code Demo

Ignoring Case Often Takes Longer

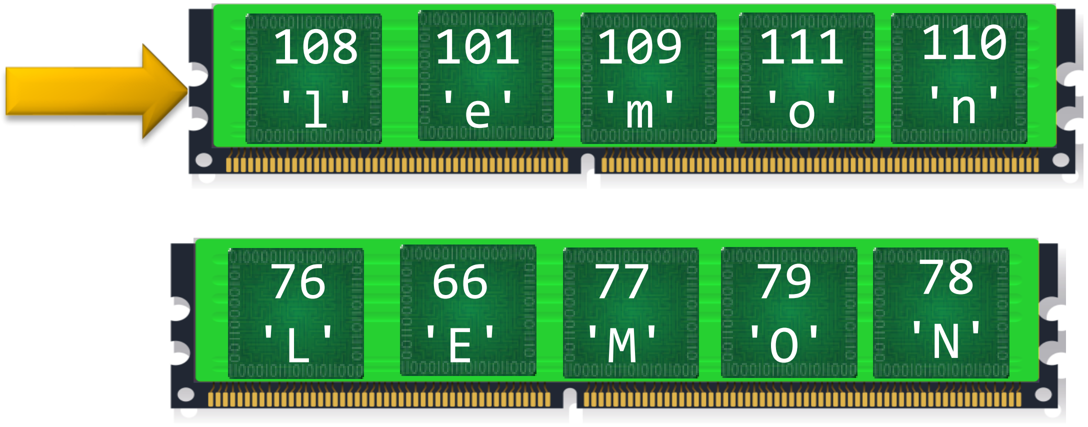
Additional work:

Try to match letters $'L' \overset{?}{==} 'l'$?

May need to examine more characters

OrdinalIgnoreCase:
No letters are different

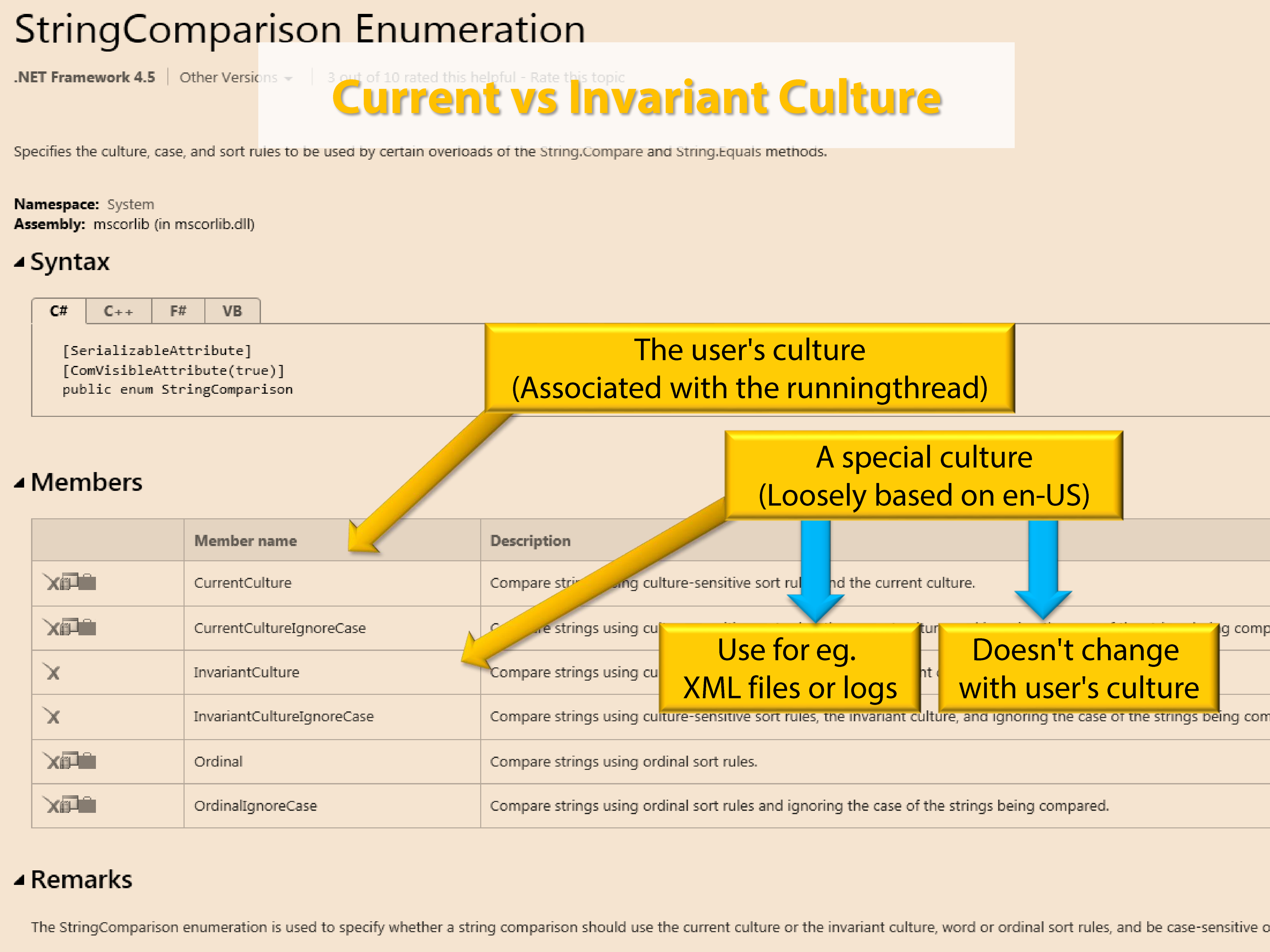
Ordinal:
First different letter
is here



Ordinal vs Culture-Sensitive

For each char…	Ordinal	Culture-Sensitive
	Consider numeric value only	Consider 'meaning' of numeric value
		eg. ß $\longrightarrow$ ss
		Consider any ordering rules for the culture
		Eg. Rules for accented chars

Current vs Invariant Culture



StringComparison Enumeration

.NET Framework 4.5 | Other Versions | 3 out of 10 rated this helpful - Rate this topic

Specifies the culture, case, and sort rules to be used by certain overloads of the String.Compare and String.Equals methods.

Namespace: System
Assembly: mscorlib (in mscorlib.dll)

Syntax

C# | C++ | F# | VB

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public enum StringComparison
```

Members

	Member name	Description
	CurrentCulture	Compare stri… …ing culture-sensitive sort rul… …nd the current culture.
	CurrentCultureIgnoreCase	C… …e strings using cu… …tur… …g comp
	InvariantCulture	Compare strings using cu… …nt
	InvariantCultureIgnoreCase	Compare strings using culture-sensitive sort rules, the invariant culture, and ignoring the case of the strings being com
	Ordinal	Compare strings using ordinal sort rules.
	OrdinalIgnoreCase	Compare strings using ordinal sort rules and ignoring the case of the strings being compared.

Remarks

The StringComparison enumeration is used to specify whether a string comparison should use the current culture or the invariant culture, word or ordinal sort rules, and be case-sensitive o

The user's culture
(Associated with the runningthread)

A special culture
(Loosely based on en-US)

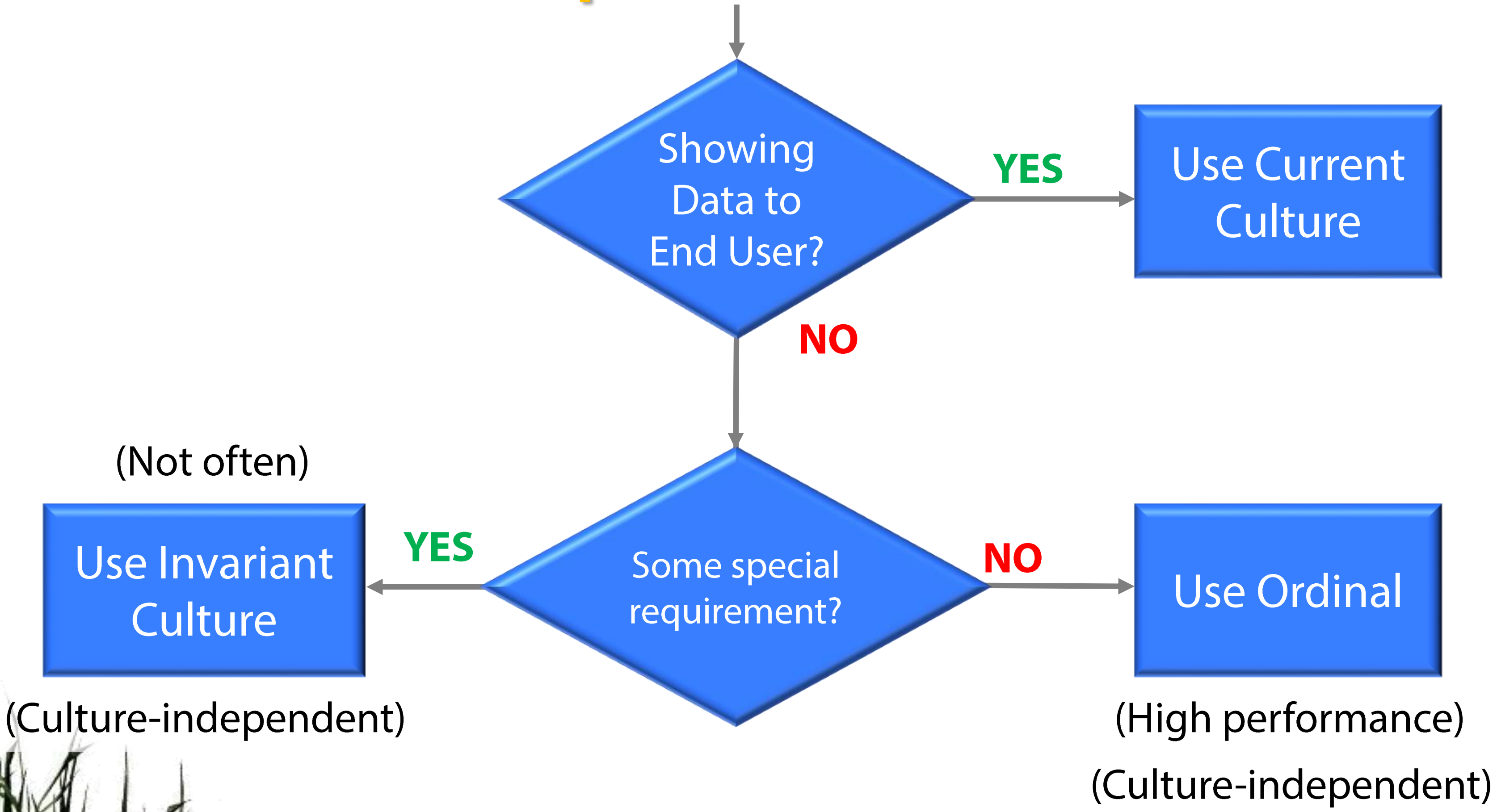
Use for eg.
XML files or logs

Doesn't change
with user's culture

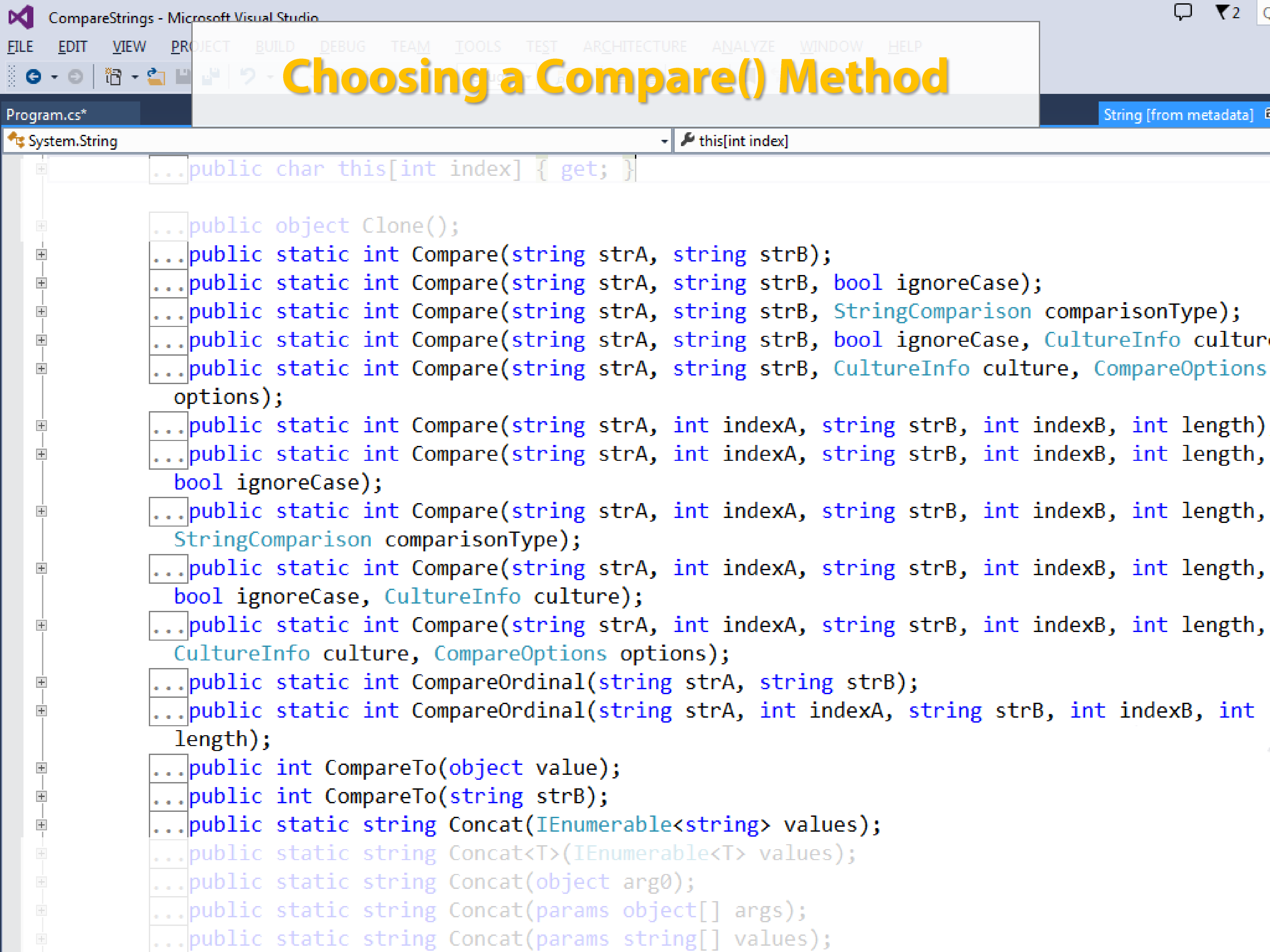
Letter Orders

Cultural	a < A < b < B < c < C < …. < z < Z
Ordinal	A < B < … < Z < a < b < … z
Cultural/Ignore-Case Ordinal/Ignore-Case	(a = A) < (b = B) < (c = C) < …. < (z = Z)

Which Comparison Should You Choose?

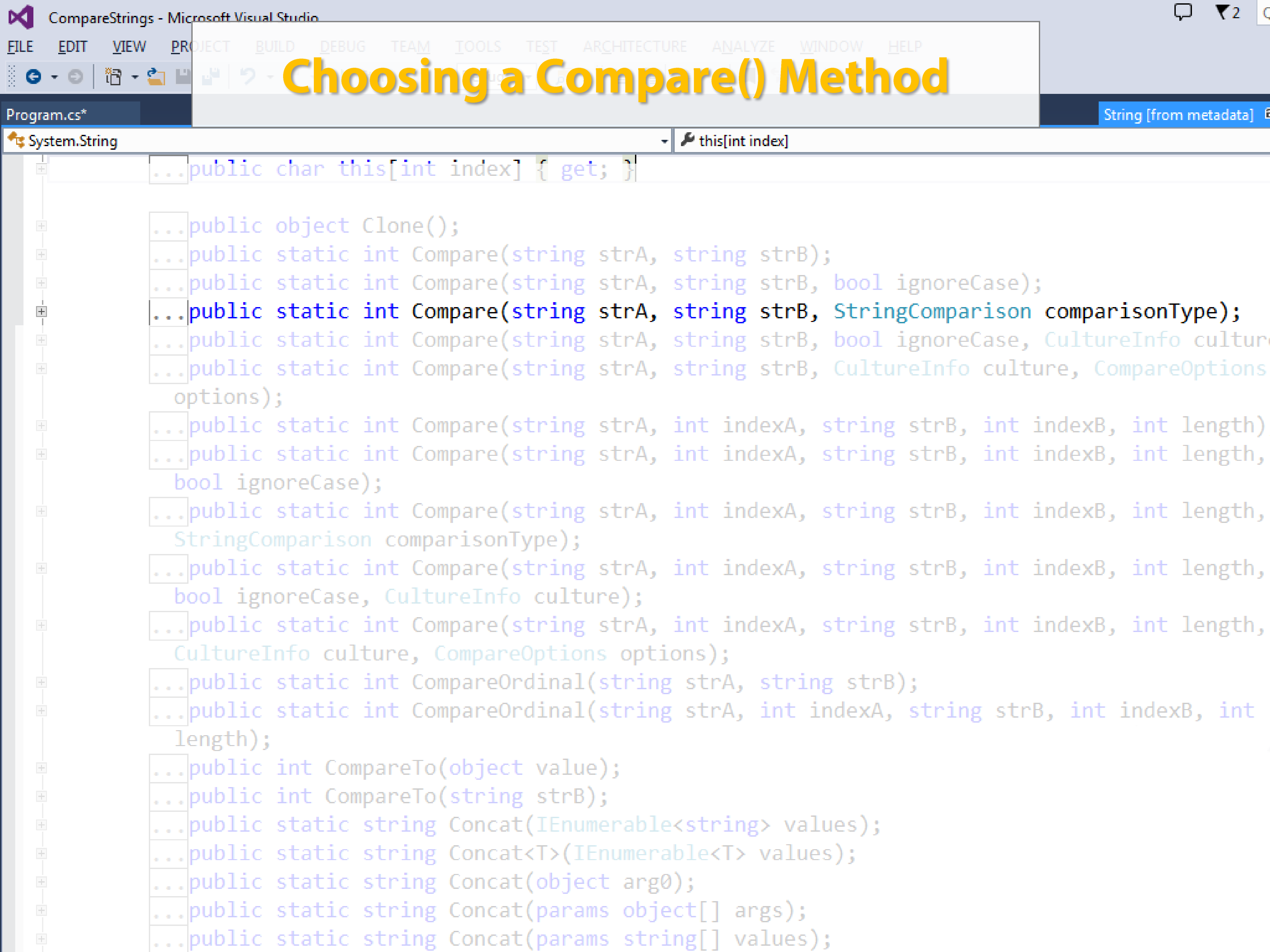


Choosing a Compare() Method



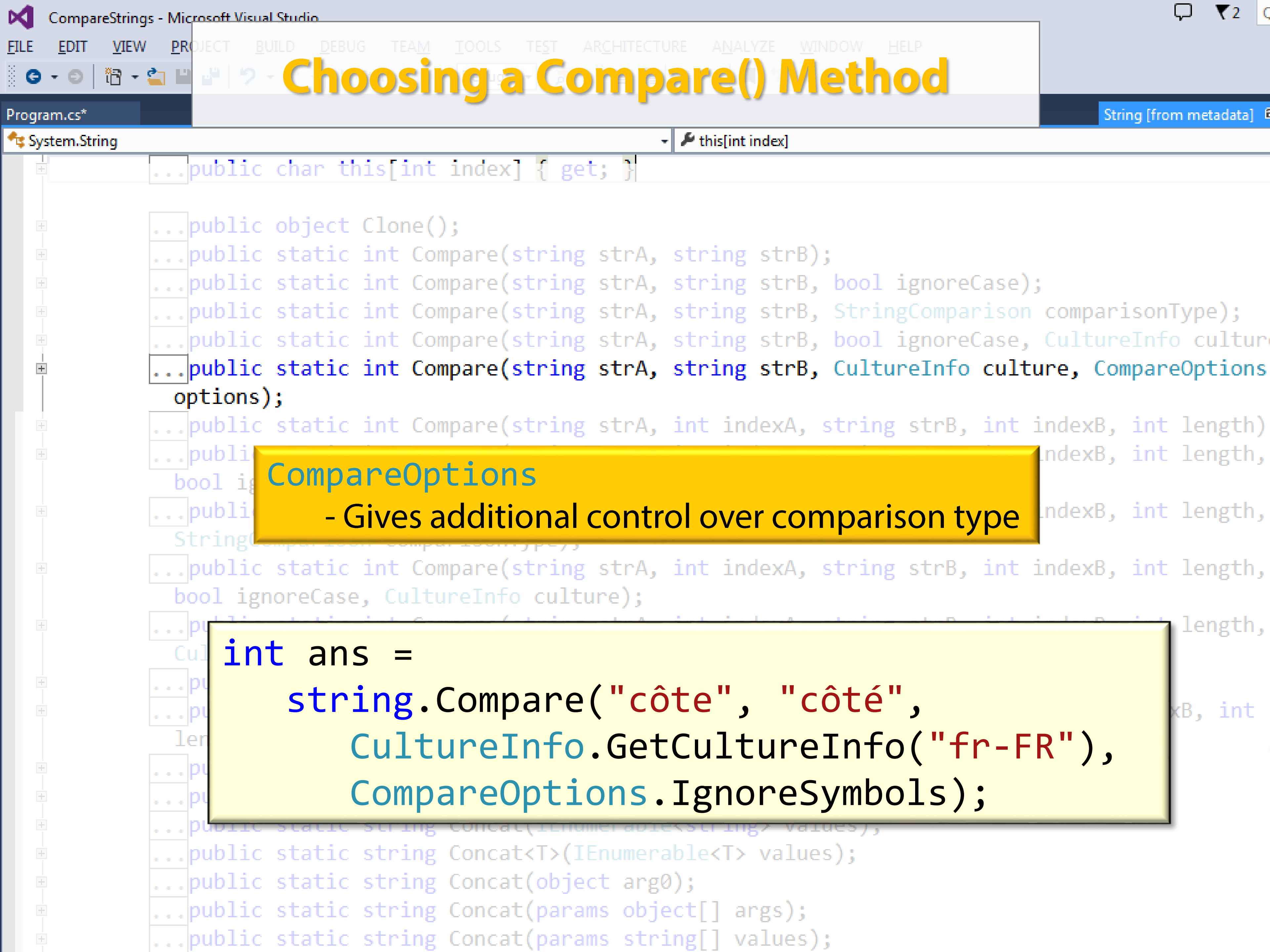
```
...public char this[int index] { get; }

...public object Clone();
...public static int Compare(string strA, string strB);
...public static int Compare(string strA, string strB, bool ignoreCase);
...public static int Compare(string strA, string strB, StringComparison comparisonType);
...public static int Compare(string strA, string strB, bool ignoreCase, CultureInfo cultur
...public static int Compare(string strA, string strB, CultureInfo culture, CompareOptions
  options);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length)
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
  bool ignoreCase);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
  StringComparison comparisonType);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
  bool ignoreCase, CultureInfo culture);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
  CultureInfo culture, CompareOptions options);
...public static int CompareOrdinal(string strA, string strB);
...public static int CompareOrdinal(string strA, int indexA, string strB, int indexB, int
  length);
...public int CompareTo(object value);
...public int CompareTo(string strB);
...public static string Concat(IEnumerable<string> values);
...public static string Concat<T>(IEnumerable<T> values);
...public static string Concat(object arg0);
...public static string Concat(params object[] args);
...public static string Concat(params string[] values);
```

Choosing a Compare() Method



```
...public char this[int index] { get; }

...public object Clone();
...public static int Compare(string strA, string strB);
...public static int Compare(string strA, string strB, bool ignoreCase);
...public static int Compare(string strA, string strB, StringComparison comparisonType);
...public static int Compare(string strA, string strB, bool ignoreCase, CultureInfo cultur
...public static int Compare(string strA, string strB, CultureInfo culture, CompareOptions
  options);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length)
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
  bool ignoreCase);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
  StringComparison comparisonType);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
  bool ignoreCase, CultureInfo culture);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
  CultureInfo culture, CompareOptions options);
...public static int CompareOrdinal(string strA, string strB);
...public static int CompareOrdinal(string strA, int indexA, string strB, int indexB, int
  length);
...public int CompareTo(object value);
...public int CompareTo(string strB);
...public static string Concat(IEnumerable<string> values);
...public static string Concat<T>(IEnumerable<T> values);
...public static string Concat(object arg0);
...public static string Concat(params object[] args);
...public static string Concat(params string[] values);
```

Choosing a Compare() Method



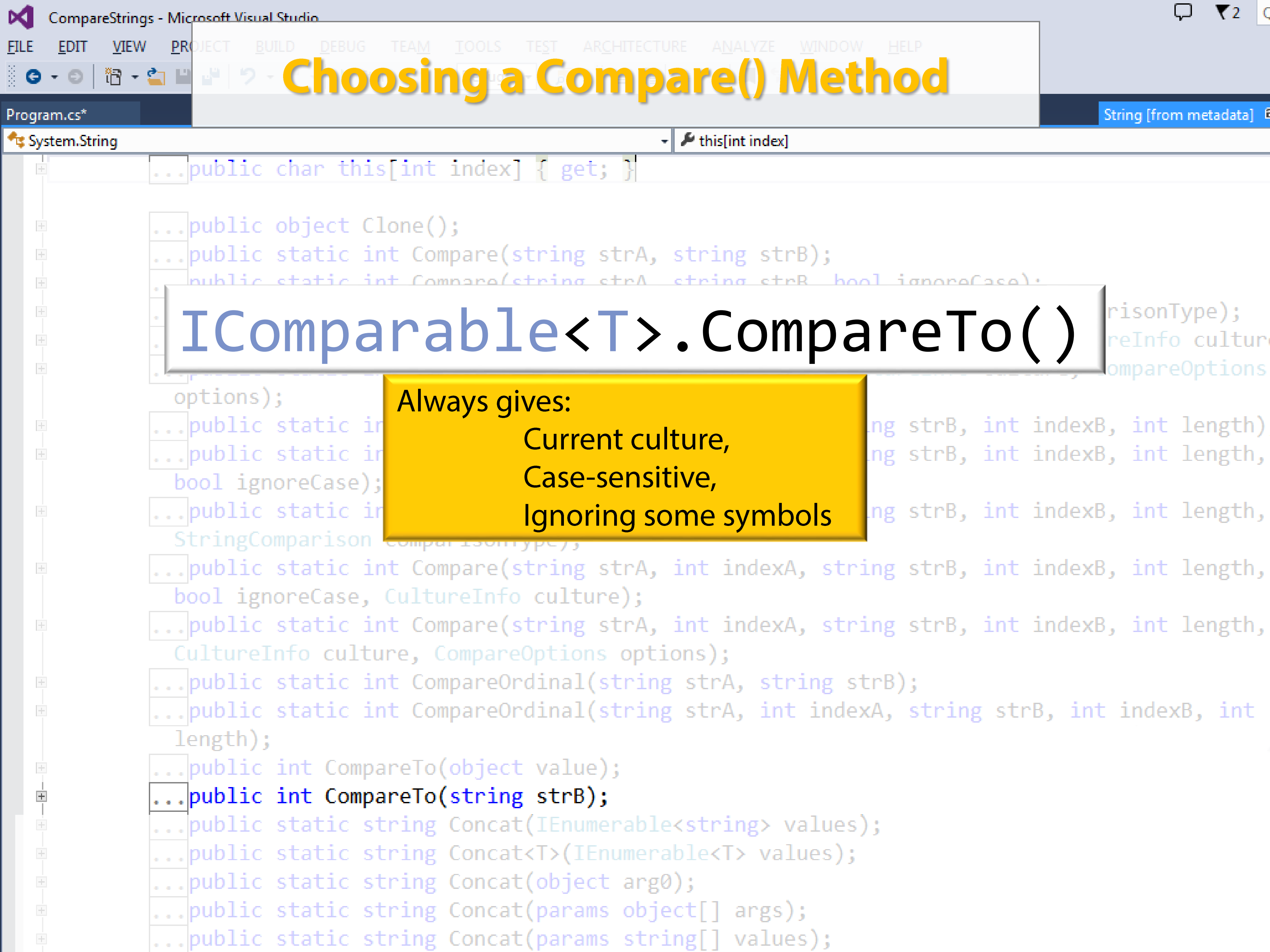
```
public char this[int index] { get; }

public object Clone();
public static int Compare(string strA, string strB);
public static int Compare(string strA, string strB, bool ignoreCase);
public static int Compare(string strA, string strB, StringComparison comparisonType);
public static int Compare(string strA, string strB, bool ignoreCase, CultureInfo cultur
public static int Compare(string strA, string strB, CultureInfo culture, CompareOptions
options);
public static int Compare(string strA, int indexA, string strB, int indexB, int length)
public static int Compare(string strA, int indexA, string strB, int indexB, int length,
bool ignoreCase, CultureInfo culture);
public static string Concat<T>(IEnumerable<T> values);
public static string Concat(object arg0);
public static string Concat(params object[] args);
public static string Concat(params string[] values);
```

CompareOptions

- Gives additional control over comparison type

```
int ans =
    string.Compare("côte", "côté",
        CultureInfo.GetCultureInfo("fr-FR"),
        CompareOptions.IgnoreSymbols);
```

Choosing a Compare() Method


CompareStrings - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Program.cs*
String [from metadata]
System.String
this[int index]

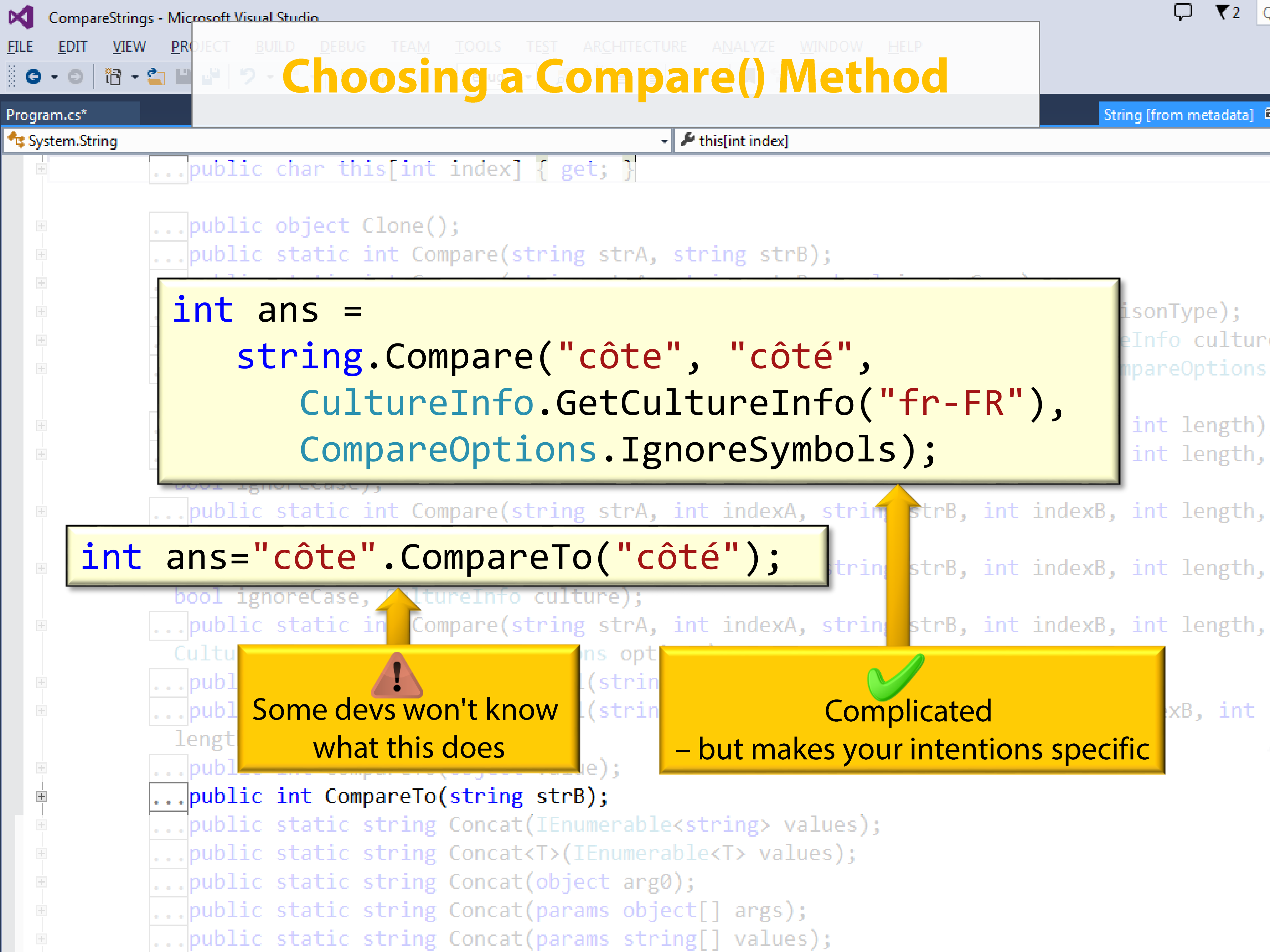
```
...public char this[int index] { get; }

...public object Clone();
...public static int Compare(string strA, string strB);
...public static int Compare(string strA, string strB, bool ignoreCase);
...                                                      risonType);
...                                                      reInfo cultur
...                                                      ompareOptions
   options);
...public static in                          ng strB, int indexB, int length)
...public static in                          ng strB, int indexB, int length,
   bool ignoreCase);
...public static in                          ng strB, int indexB, int length,
   StringComparison comparisonType);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
   bool ignoreCase, CultureInfo culture);
...public static int Compare(string strA, int indexA, string strB, int indexB, int length,
   CultureInfo culture, CompareOptions options);
...public static int CompareOrdinal(string strA, string strB);
...public static int CompareOrdinal(string strA, int indexA, string strB, int indexB, int
   length);
...public int CompareTo(object value);
...public int CompareTo(string strB);
...public static string Concat(IEnumerable<string> values);
...public static string Concat<T>(IEnumerable<T> values);
...public static string Concat(object arg0);
...public static string Concat(params object[] args);
...public static string Concat(params string[] values);
```

IComparable<T>.CompareTo()

Always gives:

- Current culture,
- Case-sensitive,
- Ignoring some symbols

Choosing a Compare() Method


CompareStrings - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Program.cs*
String [from metadata]
System.String
this[int index]

```
...public char this[int index] { get; }

...public object Clone();
...public static int Compare(string strA, string strB);
                                                    isonType);
                                                    eInfo cultur
                                                    mpareOptions

                                                    int length)
                                                    int length,
   bool ignoreCase);
...public static int Compare(string strA, int indexA, strin strB, int indexB, int length,
                                                    tring strB, int indexB, int length,
   bool ignoreCase, CultureInfo culture);
...public static int Compare(string strA, int indexA, strin strB, int indexB, int length,
   Cultu                    ns opt
...publ                   l(strin
...publ                   l(strin                                       xB, int
   lengt
...publ                         e);
...public int CompareTo(string strB);
...public static string Concat(IEnumerable<string> values);
...public static string Concat<T>(IEnumerable<T> values);
...public static string Concat(object arg0);
...public static string Concat(params object[] args);
...public static string Concat(params string[] values);
```

```
int ans =
    string.Compare("côte", "côté",
        CultureInfo.GetCultureInfo("fr-FR"),
        CompareOptions.IgnoreSymbols);
```

```
int ans="côte".CompareTo("côté");
```

Some devs won't know what this does

Complicated – but makes your intentions specific

String Pooling

```
string apple1 = "Apple";
string apple2 = "Ap" + "ple";
```

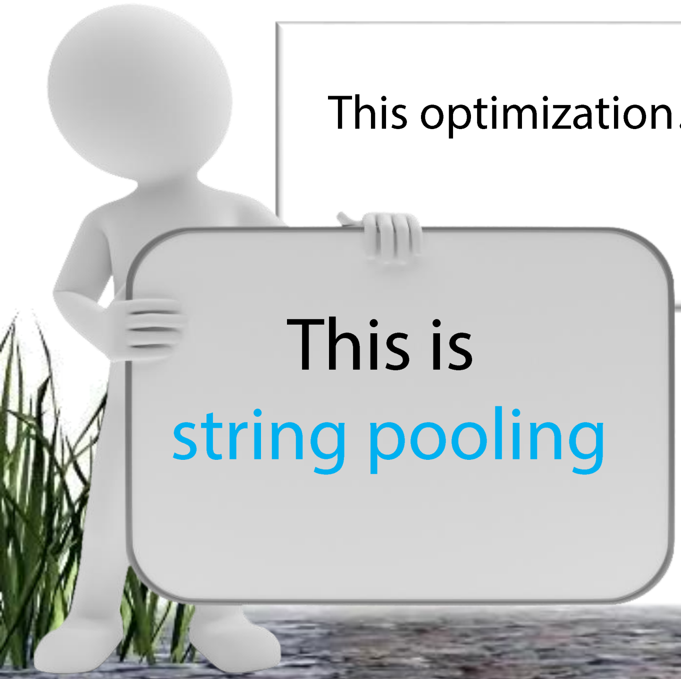
Hardcoded strings *might* compile to a single instance…

`ReferenceEquals()` *might* return `true`

This optimization…

✓ Saves memory

✓ Can speed string comparisons



Code Demo

Summary

- Ordinal comparison: Just check numerical value of each `char`.
- Culture-sensitive comparisons:
  - Perform character expansions etc.
  - Take account of specified culture.
  - Slower.
- `string` offers overloads of `Compare()` and `Equals()`.
- == gives an ordinal, case-sensitive equality comparison.
  - But `IComparable<T>.CompareTo()` is culture-sensitive.
- The compiler might do string pooling.