

# Does Your Code Pass The Turkey Test?

Feb 16, 2008

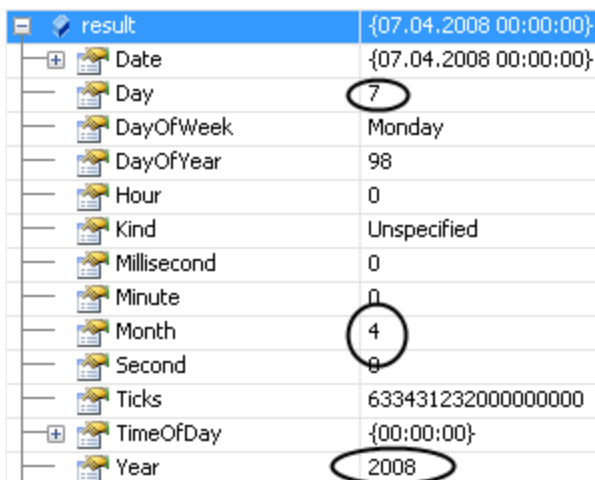
Over the past 6 years or so, I've failed each item on "The Turkey Test." It's very simple: will your code work properly on a person's machine in or around the country of Turkey? Take this simple test.

## 1. Parsing dates from a configuration file using `DateTime.Parse(string)`:

```
// Set the thread to act as if we're running in Turkey
Thread.CurrentThread.CurrentCulture = new CultureInfo("tr-TR");
string julyFourth = "07/04/2008";
DateTime result = DateTime.Parse(julyFourth);
```

Does it pass "The Turkey Test?"

Nope:



Property	Value
result	{07.04.2008 00:00:00}
Date	{07.04.2008 00:00:00}
Day	7
DayOfWeek	Monday
DayOfYear	98
Hour	0
Kind	Unspecified
Millisecond	0
Minute	0
Month	4
Second	0
Ticks	633431232000000000
TimeOfDay	{00:00:00}
Year	2008

**Reason:** Turkish people write July 4th, 2008 as "04.07.2008"

**Fix:** Always specify what format your date is in. In this case, we use a `DateTimeFormat.InvariantInfo` which just *happens* to be USA English's format (more or less):

```
// Set the thread to act as if we're running in Turkey
Thread.CurrentThread.CurrentCulture = new CultureInfo("tr-TR");
string julyFourth = "07/04/2008";
DateTime result = DateTime.Parse(julyFourth,
    DateTimeFormatInfo.InvariantInfo);
```

Which gives us what we were expecting:

Property	Value
Date	{04.07.2008 00:00:00}
Day	4
DayOfWeek	Friday
DayOfYear	186
Hour	0
Kind	Unspecified
Millisecond	0
Minute	0
Month	7
Second	0
Ticks	6335072640000000000
TimeOfDay	{00:00:00}
Year	2008

Scott Hanselman likes to talk about DateTimes. (Be sure to see his DateTime [interview question](#)).

2. Ok, ok. You knew about dates. I sort of did, but I still got it wrong the first few times. What about this seemingly simple piece of code:

```
string configurationValue = "4.5";
double discountMultiplier = 0.01 * double.Parse(configurationValue);
```

Does it pass “The Turkey Test?”

Nope:

configurationValue	"4.5"
discountMultiplier	0.45

**Reason:** Turkish people use a period to group digits (like people in the USA use a comma). Instead of getting a 4.5% discount like you intended, Turkish people will be getting at 45% discount.

**Fix:** Again, always specify your format explicitly:

```
string configurationValue = "4.5";
double discountMultiplier = 0.01 *
    double.Parse(configurationValue,
        NumberFormatInfo.InvariantInfo);
```

Which saves your company from having to go out of business from having too high of discounts:

configurationValue	"4.5"
discountMultiplier	0.045

3. Say your application reads in some command line parameter:

```
string configurationPrintOrientation = args[1]; // "PORTRAIT"

if (configurationPrintOrientation == "portrait")
{
    PrintInPortraitMode();
}
else
{
    PrintInLandscapeMode();
}
```

Forget about Turkey, this won't even pass in the USA. You need a case insensitive compare. So you try:

`String.Compare(string,string,bool ignoreCase):`

```
if (string.Compare(configurationPrintOrientation, "portrait", true) == 0)
{
    PrintInPortraitMode();
}
else
{
    PrintInLandscapeMode();
}
```

Or using `String.ToLower():`

```
switch (configurationPrintOrientation.ToLower())
{
    case "portrait":
        PrintInPortraitMode();
        break;
    default:
        PrintInLandscapeMode();
        break;
}
```

Or `String.Equals` with `CurrentCultureIgnoreCase:`

```
if (configurationPrintOrientation.Equals("portrait",
                                         StringComparison.CurrentCultureIgnoreCase))
{
    PrintInPortraitMode();
}
else
{
    PrintInLandscapeMode();
}
```

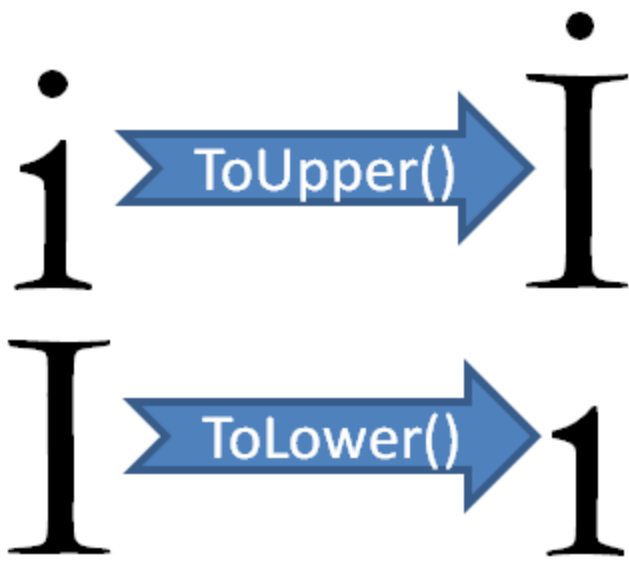
Or even a trusty `Regular Expression:`

```
Regex re = new Regex("port[rait]+",
                     RegexOptions.IgnoreCase);
if (re.IsMatch(configurationPrintOrientation))
{
    PrintInPortraitMode();
}
else
{
    PrintInLandscapeMode();
}
```

Do any of these pass "The Turkey Test?"

Not a chance!

**Reason:** You've been hit with the "Turkish I" problem.



As discussed by lots and lots of people, the “I” in Turkish behaves differently than in most languages. [Per the Unicode standard](#), our lowercase “i” becomes “İ” (U+0130 “Latin Capital Letter I With Dot Above”) when it moves to uppercase. Similarly, our uppercase “I” becomes “ı” (U+0131 “Latin Small Letter Dotless I”) when it moves to lowercase.

**Fix:** Again, use an [ordinal \(raw byte\) comparer](#), or [invariant culture](#) for comparisons unless you absolutely need culturally based linguistic comparisons (which give you uppercase I’s with dots in Turkey)

```
if (string.Compare(configurationPrintOrientation,
                    "portrait",
                    StringComparison.OrdinalIgnoreCase) == 0)
{
    PrintInPortraitMode();
}
else
{
    PrintInLandscapeMode();
}
```

Or

```
switch (configurationPrintOrientation.ToLowerInvariant())
{
    case "portrait":
        PrintInPortraitMode();
        break;
    default:
        PrintInLandscapeMode();
        break;
}
```

Or

```

if (configurationPrintOrientation.Equals("portrait",
                                         StringComparison.OrdinalIgnoreCase))
{
    PrintInPortraitMode();
}
else
{
    PrintInLandscapeMode();
}

```

And finally, a fix to our Regex friend:

```

Regex re = new Regex("port[rait]+",
                    RegexOptions.IgnoreCase
                    |
                    RegexOptions.CultureInvariant);
if (re.IsMatch(configurationPrintOrientation))
{
    PrintInPortraitMode();
}
else
{
    PrintInLandscapeMode();
}

```

4. My final example is especially embarrassing. I was actually smug when I wrote something like this (note the comment):

```

string input = Console.ReadLine();
Regex zipCodeChecker = new Regex(@"\d{5}");
Match m = zipCodeChecker.Match(input);
if (m.Success)
{
    // Parse is safe since the Regex
    // guarantees it's a digit
    int zipCode = int.Parse(m.Value);
    Console.WriteLine("Next zip code is {0}", zipCode + 1);
}
else
{
    Console.WriteLine("That wasn't a zip code!");
}

```

Does this simple program pass “The Turkey Test?”

You’re probably hesitant to say “yes” ... and rightly so. Because this too fails the test.

**Reason:** As [Raymond Chen points out](#), there are more than 10 digits out there. Here, I use real Arabic digits (see page 4 of [this code table](#)):

```
string input = "٤١٠٣٨";
Regex zipCodeChecker = new Regex(@"\d{5}");
Match m = zipCodeChecker.Match(input);
if (m.Success)
{
    // Parse is safe since the Regex
    // guarantees it's a digit
    int zipCode = int.Parse(m.Value);
    Console.WriteLine("Next zip code is {0}", zipCode);
}
else
{
    // FormatException was unhandled
    // Input string was not in a correct format.
    // Troubleshooting tips:
    // Make sure your method arguments are in the right format.
    // When converting a string to DateTime, parse the string to take the date and time.
    // Get general help for this exception.
}
```

**Fix:** A `CultureInfo` won't help you here. The only option is to explicitly specify the character range you mean:

```
Regex zipCodeChecker = new Regex(@"[0-9]{5}");
```

Or use the `RegexOptions.ECMAScript` option. In `JavaScript`, "d" means [0-9] which gives us:

```
Regex zipCodeChecker = new Regex(@"\d{5}",
    RegexOptions.ECMAScript);
```

"The Turkey Test" poses a very simple question, but yet is full of surprises for guys like me who didn't realize all the little details. Turkey, as we saw above, is sort of like "New York, New York" in the classic Frank Sinatra song:

*"These little town blues, are melting away  
I'll make a brand new start of it - in old New York  
**If I can make it there, I'll make it anywhere**  
Its up to you - New York, New York"*

If your code properly runs in Turkey, it'll probably work anywhere.

This brings us to the logo program:



### “Turkey Test” Logo Program Requirements:

1. Read Joel Spolsky’s [basic introduction to Unicode](#) to understand the absolute minimum about it.
2. Read Microsoft’s [“New Recommendations for Using Strings in Microsoft .NET 2.0”](#) article and [this post](#) by the BCL team.
3. Always specify culture and number formatter for all string, parsing, and regular expression you use.
4. If you read data from the user and want to process it in a language sensitive matter (e.g. sorting), use the [CurrentCulture](#). If none of that matters, really try to use use [Ordinal](#) comparisons.
5. Run [FxCop](#) on your code and make sure you have no [CA1304](#) (SpecifyCultureInfo) or [CA1305](#) (SpecifyFormatProvider) warnings.
6. Unit test string comparing operations in the [“tr-TR”](#) culture as well as your local culture (unless you actually live in Turkey, then use a culture like [“en-US”](#)).

Having successfully passed the above requirements, your software will finally be able to wear “Passed ‘The Turkey Test’” logo with pride.

**Note:** Special thanks to my coworker, Evan, for calling the 3rd type of error “Turkeys.” Also, thanks to [Chip and Dan Heath](#) for the “Sinatra Test” idea.