Strings, Regex, Web Response

Last Time

- We started talking about collections
  - Hash tables
  - ArrayLists

Example

- Let's look at the example in Figure 3.3 on pages 72-74 of the book

Strings

- Another way of finding the classes available through the .NET FCL is the Object Browser
  - View menu, then Object Browser
- You will find the classes under mscorlib
- String is under mscorlib->System
  - It's in the System namespace

Strings

- The methods by the String class include:
  - ToLower
  - ToUpper
  - Replace
  - Length
  - IndexOf

Example

```
string theString = "Programming Microsoft .NET";
Console.WriteLine("default: \t {0}", theString);
Console.WriteLine("lowercase: \t {0}", theString.ToLower());
Console.WriteLine("uppercase: \t {0}", theString.ToUpper());
Console.WriteLine("replace: \t {0}",
                  theString.Replace(".", "dot "));
Console.WriteLine("length: \t {0}", theString.Length);
Console.WriteLine("\"Microsoft\" occurs at character {0}",
                  theString.IndexOf("Microsoft"));
Console.WriteLine();
Console.WriteLine();
Console.WriteLine("Please press enter key to quit");
Console.ReadLine();
```

What is the Output?

- Strings can be combined using + or treated as character arrays

```
string e = "dog" + "bee";
e += "cat";
string f = e.Substring(1,7);
Console.WriteLine(f);

for (int i = 0; i < f.Length; i++)
{
    Console.Write("{0,-3}", f[i]);
}
```


Concatenation

- The items being concatenated don't have to be strings

```
string t = "item " + 12 + " sells at "
           + '\xA3' + 3.45;
Console.WriteLine(t);
```


Formatting Strings

- Another useful class is String.Format, which has a lot in common with Console.WriteLine

```
Console.WriteLine(
  "Hello {0} {1} {2} {3} {4} {5} {6} {7} {8}",
    123, 45.67, true, 'Q', 4, 5, 6, 7, '8');
string u = String.Format(
  "Hello {0} {1} {2} {3} {4} {5} {6} {7} {8}",
    123, 45.67, true, 'Q', 4, 5, 6, 7, '8');
Console.WriteLine(u);
```


Syntax for formatString

- "{ N [, M][: formatString]}", arg1, ... argN
  - N is a zero-based integer indicating the argument to be formatted
  - M is an optional integer indicating the width of the region to contain the formatted value, padded with spaces. If M is negative, the formatted value is left-justified; if M is positive, the value is right-justified
  - formatString is an optional string of formatting codes
  - argN is the expression to use at the equivalent position inside the quotes in the string

Example of Using Width Region

- What is the Output?

```
Console.WriteLine(
    "\n{0,-10}{1,-3}", "Name","Salary");
Console.WriteLine(
    "----------------");
Console.WriteLine(
    "{0,-10}{1,6}", "Bill", 123456);
Console.WriteLine(
    "{0,-10}{1,6}", "Polly", 7890);
```


Standard Format Specifiers

Character	Interpretation
C	Currency
D	Decimal integer
E	Exponent
F	Fixed point
G	General
N	Same as F with , every thousand
P	Percentage
R	Round trip
X	Hex

Example

```
int i = 123456;
Console.WriteLine("{0:C}", i); // $123,456.00
Console.WriteLine("{0:D}", i); // 123456
Console.WriteLine("{0:E}", i); // 1.234560E+005
Console.WriteLine("{0:F}", i); // 123456.00
Console.WriteLine("{0:G}", i); // 123456
Console.WriteLine("{0:N}", i); // 123,456.00
Console.WriteLine("{0:P}", i); // 12,345,600.00 %
Console.WriteLine("{0:X}", i); // 1E240
```


Custom Format Specifiers

0	Zero placeholder	Results in a nonsignificant zero if a number has fewer digits than there are zeros in the format
#	Digit Placeholder	Replaces the pound symbol (#) with only significant digits
.	Decimal point	Displays a period
,	Group Separator	Separates number groups, as in 1,000
%	Percent Notation	Display a percent
E+0	Exponent notation	Formats the output of exponents
\	Literal	Used as escape characters
“abc”	Literal string	Displays what’s within quotes exactly
;	Section separator	Specifies different output if the numeric value to be formatted is positive, negative, or zero

Example

- `Console.WriteLine("{0:#0;(#0);<zero>}", i);`
- `Console.WriteLine("{0:#%}", i);`
- What is the output if:
  - i = 12345
  - i = -12345
  - i = 0

Regular Expressions

- System.Text provides a number of classes for regular expression processing
- They are a powerful, flexible, and efficient strategy for processing text

Regular Expressions

.	Matches any character except \n
[characters]	Matches a single character in a list
[^characters]	Matches a single character not in a list
[charX-charY]	Matches a single character in the specified range
\w	Matches a word character
\W	Matches a nonword character
\s	Matches a whitespace character
\S	Matches a nonwhitespace character
\d	Matches a decimal digit
\D	Matches a nondigit character

Regular Expressions

^	Bginning of the line
$	End of the line
\b	On a word boundary
\B	Not on a word boundary
*	Zero or more matches
+	One or more matches
?	Zero or one matches
{n}	Exactly n matches
{n,}	At least n matches
{n,m}	At least n but no more than m matches

Regular Expressions

()	Capture matched substring
(?<name>)	Capture matched substring into group name
|	Logical OR

Example

- What is the output?

```
string s = "Once Upon A Time In America";
Regex r = new Regex(" ");
foreach (string ss in r.Split(s))
{
     Console.WriteLine(ss);
}
```


Multiple Delimiters

```
string t =
  "Once,Upon:A/Time\\In\'America";
Regex q = new Regex(@" |,|:|/|\\|\'");
foreach (string ss in q.Split(t))
{
     Console.WriteLine(ss);
}
```


What's the Output?

```
string u = "Once   Upon A Time In
  America";
Regex p = new Regex(" ");
foreach (string ss in p.Split(u))
{
     Console.WriteLine(ss);
}
```


A Better Output

```
string v = "Once   Upon A Time In   America";
Regex o = new Regex(@"[\s]+");
foreach (string ss in o.Split(v))
{
    Console.WriteLine(ss);
}
```

- What if we are concerned with only spaces and not other whitespace characters?

What's Happening Here?

```
string x = "Once Upon A Time In
  America";
Regex m = new Regex("( )");
foreach (string ss in m.Split(x))
{
     Console.WriteLine(ss);
}
```

Matching

- Other classes in System.Text are Match and MatchCollection

```
Regex r = new Regex("in");
Match m = r.Match("Matching");
if (m.Success)
{
   Console.WriteLine(
           "Found '{0}' at position {1}",
            m.Value, m.Index);
}
```


Multiple Expressions

```
Regex p = new Regex("((an)|(in)|(on))");
MatchCollection mn = p.Matches(
    "The King Kong Band Wins Again");
for (int i = 0; i < mn.Count; i++)
{
    Console.WriteLine(
        "Found '{0}' at position {1}",
        mn[i].Value, mn[i].Index);
}
```


Backtracking

```
Regex n = new Regex("Gr(a|e)y");
MatchCollection mp = n.Matches(
    "Green, Grey, Granite, Gray");
for (int i = 0; i < mp.Count; i++)
{
    Console.WriteLine(
        "Found '{0}' at position {1}",
        mp[i].Value, mp[i].Index);
}
```


Groups

```
Regex q = new Regex(
    "(?<something>\\w+):(?<another>\\w+)");
Match n = q.Match("Salary:123456");
Console.WriteLine(
    "{0} = {1}",
    n.Groups["something"].Value,
    n.Groups["another"].Value);
```

- Output is:

```
Salary = 123456
```


What is the Output?

```
Console.WriteLine();
s = @"<html>
    <a href=""first.htm"">first text</a>
    <br>loads of other stuff
    <a href=""second.htm"">second text</a>
    <p>more<a href=""third.htm"">third text</a>
    </html>";
e = @"<a[^>]*href\s*=\s*[""']?([^'"">]+)['""]?>";
MatchCollection mc = Regex.Matches(s, e);
foreach (Match mm in mc)
    Console.WriteLine("HTML links: {0}", mm);
```


Breakdown of Expression

<a[^>]*href	Match the character substring "<a" followed by zero or more instances of any characters except the ">" character, followed by the string "href",
\s*=\s*	followed by zero or more instances of whitespace, followed by the "=" character, followed by zero or more instances of whitespace,
['"]?	followed by zero or one instance of single or double quotes,
([^'">]+)	followed by one or more instances of any characters except the single or double quotes or the closing ">",
['"]?>	followed by zero or one instance of either the single or double quotes.

Your Turn

- Write a program that will convert a string to proper case
  - Initial caps on each word in the string
- Sample output:
  - `Initial String: the qUEEn wAs in HER parLOr`
  - `ProperCase:     The Queen Was In Her Parlor`

Internet Classes

- Let's examine figure 3.5 on p. 82

Summary

- Collections
  - Strings
  - Regex
  - Internet Classes
- Completed pages 71 - 83