Module 16

”Manipulating Text”

TEKNOLOGISK
INSTITUT

Agenda

- **Building Strings**
- Regular Expressions
- Encodings

Dynamically Building Strings

- Strings are immutable in .NET
  - They are interned
- Hence building strings dynamically is expensive

- Instead .NET provides the `StringBuilder` class
  - Has a number of formatting methods for building strings
  - Retrieve end result as a conventional string with `ToString()`

StringBuilder

- StringBuilder supplies a number of methods
  - Append()
  - AppendFormat()
  - Replace()
  - Insert()
  - …

```
StringBuilder sb = new StringBuilder();
foreach( DriveInfo di in DriveInfo.GetDrives() )
{
   sb.AppendFormat( "{0} {1} ", di.Name, di.VolumeLabel );
}
sb.Insert( 0, header );
Console.WriteLine( sb.ToString() );
```

Agenda

- Building Strings
- **Regular Expressions**
- Encodings

Introducing Regular Expressions

- Well-established formalism for patterns of text
  - Validating
  - Matching
  - Replacing

- Concise syntax stemming from automata theory
  - Compatible with Perl regular expressions

- Regular expressions functionality in .NET
  - **Regex** class

Regex Quantifiers

- Quantifiers

`*`	Means	0 or more occurrences
`+`	Means	1 or more occurrences
`?`	Means	0 or 1 occurrences (optional)

- Range Quantifiers

`{n}`	Means	Exactly *n* occurrences
`{n,}`	Means	At least *n* occurrences
`{n,m}`	Means	Between *n* and *m* occurrences

Regex Positional Assertions

- Multiline-aware

`^`	Means	First position
`$`	Means	Last position

- Multiline-unaware

`\A`	Means	First position of string
`\Z`	Means	Last position of string (or before last newline)
`\z`	Means	Last position of string

- Other

`\G`	Means	End of last match
`\b`	Means	At word boundary (`\w` or `\W` enclosing)
`\B`	Means	The converse of `\b`

Regex Character Classes

- Custom groups

`[abc]`	Means	Any of the characters
`[^abc]`	Means	All characters not including
`[A-Z0-9]`	Means	Characters in the specified ranges

- Built-in groups

`.`	Means	Any character but newline
`\w`	Means	Any word character (alpha-numeric)
`\W`	Means	The opposite of `\w`
`\s`	Means	Any white-space character
`\S`	Means	The opposite of `\s`
`\d`	Means	Any digit
`\D`	Means	The opposite of `\d`

Matching with `Regex.IsMatch()`

- Positional assertions
- Quantifiers
- Character Groups
- Literals

```
Regex regex = new Regex( @"^-?\d+(\,\d{1,2})?$" );

Console.WriteLine( regex.IsMatch( "-87,0" ) );      // ???
Console.WriteLine( regex.IsMatch( "42,000" ) );     // ???
Console.WriteLine( regex.IsMatch( "1111,22" ) );    // ???
Console.WriteLine( regex.IsMatch( "9999,88$" ) );   // ???
Console.WriteLine( regex.IsMatch( "9.999,88" ) );   // ???
```

Extract Matched Data

- The data matched can be retrieved by using the static **`Match()`** method

```
string input = "Company Name: Contoso, Inc.";
Match m = Regex.Match(input, @"Company Name: (.*$)");

Console.WriteLine( m.Groups[1] );
```

- `RegExOptions` enumeration can be supplied
  - `None`
  - `IgnoreCase`
  - `Multiline`
  - `Compiled`
  - `Singleline`
  - `CultureInvariant`
  - …

Capture Groups

- Capture groups

(?<*name*>…)	Named capture
(…)	Unnamed implicit capture
(?:…)	Explicit noncapture group

- Backreferences

\n	Matches last capture group
\k<*name*>	Matches last named capture group
\k'*name*'	Matches last named capture group

Substitutions

- Matches can be substituted through replacement patterns
  - Not identical to regular expression patterns. Include e.g.

`$n`	replace last substring matched by group
`${`*name*`}`	replace last substring matched by named group
`$&`	replace entire match
`$+`	replace last group captured
`$_`	replace entire input string

Substitution Example

- Substitute with the **`Regex.Replace()`** method

```
string input = "03/24/2007";

string s = Regex.Replace(input,
   @"\b(?<month>\d{1,2})/(?<day>\d{1,2})/(?<year>\d{2,4})\b",
   "${day}-${month}-${year}"
);
```

Quiz: Backreferencing

TEKNOLOGISK INSTITUT

- What do these capture?

```
Regex r = new Regex(@"href\s*=\s*(?:""(?<1>[^""]*)""|(?<1>\S+))",
   RegexOptions.IgnoreCase|RegexOptions.Compiled);
for( Match m = r.Match(inputString); m.Success; m = m.NextMatch() )
{
   Console.WriteLine("Found href " + m.Groups[1] + " at "
                  + m.Groups[1].Index);
}
```

- What is matched by the following examples?

```
(?<char>\w)\k<char>
```

```
(?<1>a)(?<1>\1b)*
```

Agenda

- Building Strings
- Regular Expressions
- **Encodings**

Encodings and Code Pages

- Characters need to be represented by byte values

- Code pages
  - Mappings between characters and byte values
  - Can be interchanged to overcome problem that many different characters need to be represented

- ASCII and ANSI are encodings based upon code pages
  - ASCII maps 0-127 and 128-255 to characters
  - ANSI/ISO

Encodings

- Other encodings are not based upon code pages
  - Unicode is basically a table of ”all” characters

- Encodings
  - Unicode
    - `UTF7` `UTF7Encoding`
    - `UTF8` `UTF8Encoding`
    - `UTF32` `UTF32Encoding`
    - …
  - `ASCII` `ASCIIEncoding`
  - …

Using the **Encoding** Class

- Encode back and forth using a specific encoding

```
byte[] encodedText = Encoding.Unicode.GetBytes( "Hello world" );

Console.WriteLine( Encoding.UTF7.GetString( encodedText ) );
```

- Use `Encoding.GetEncodings()` to retrieve supported encodings

```
EncodingInfo[] ei = Encoding.GetEncodings();
foreach( EncodingInfo e in ei )
{
   Console.WriteLine( "{0}: {1}, {2}",
      e.CodePage, e.Name, e.DisplayName );
}
```

Encoding Files

TEKNOLOGISK INSTITUT

- You can specify the encoding when reading or writing files

```
string filename = @"C:\Tmp\utf7.txt";
using( StreamWriter sw =
   new StreamWriter( filename, false, Encoding.UTF7 ) )
{
   sw.WriteLine( "Hello, World!" );
}
```

```
string filename = @"C:\Tmp\utf7.txt";
using( StreamReader sr =
   new StreamReader( filename, Encoding.UTF7 ) )
{
   Console.WriteLine( sr.ReadToEnd() );
}
```

Summary

- Building Strings
- Regular Expressions
- Encodings

Question

You are creating an application with a method using regular expressions to validate inputs as follows:

```
01 bool ContainsEmail( string input )
02 {
03     string pattern = @"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}\b";
04     
05     return r.IsMatch( input );
06 }
```

You need to ensure that the regular expression syntax is only evaluated when the **`Regex`** object is initially instantiated. Which code segment should be added to line 04?

a) `Regex r = new Regex( pattern, RegexOptions.None );`

b) `Regex r = new Regex( pattern, RegexOptions.CultureInvariant );`

c) [✓ `Regex r = new Regex( pattern, RegexOptions.Compiled );`]

d)
```
var info = new RegexCompilationInfo( input, new AssemblyName( "regex" ) );
Regex.CompileToAssembly( new []{ info } );
Regex r = new Regex( pattern, RegexOptions.ECMAScript );
```

TEKNOLOGISK
INSTITUT