Chapter 16 – Strings and Characters: A Deeper look

16.1 Introduction

- String and character processing capabilities
  - Text editors
  - Word processors
  - Text mining

16.2 Fundamentals of Characters and Strings

- Unicode character set
- String
  - Object of class String in System namespace
  - Consist of characters
  - Provides eight constructors for initialing strings

```
// Fig. 16.1: StringConstructor.cs
// Demonstrating String class constructors.

using System;
using System.Windows.Forms;

// test several String class constructors
class StringConstructor
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string output;
      string originalString, string1, string2,
         string3, string4;

      char[] characterArray =
         { 'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y' };

      // string initialization
      originalString = "Welcome to C# programming!";
      string1 = originalString;
      string2 = new string( characterArray );
      string3 = new string( characterArray, 6, 3 );
      string4 = new string( 'C', 5 );

      output = "string1 = " + "\"" + string1 + "\"\n" +
         "string2 = " + "\"" + string2 + "\"\n" +
         "string3 = " + "\"" + string3 + "\"\n" +
         "string4 = " + "\"" + string4 + "\"\n";

```

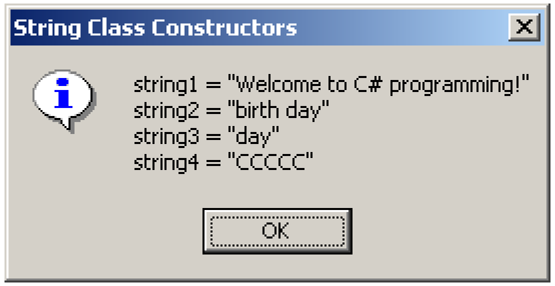
StringConstructor.cs

```
         MessageBox.Show( output, "String Class Constructors",
            MessageBoxButtons.OK, MessageBoxIcon.Information );

      } // end method Main

   } // end class StringConstructor
```

Outline

StringConstructor.cs

Program Output



```
string1 = "Welcome to C# programming!"
string2 = "birth day"
string3 = "day"
string4 = "CCCCC"
```

16.4 String Indexer, Length Property and CopyTo Method

- String indexer
  - Retrieval of any character in the string
- Length property
  - Returns the length of the string
- CopyTo
  - Copies specified number of characters into a char array

Outline

StringMethods.cs

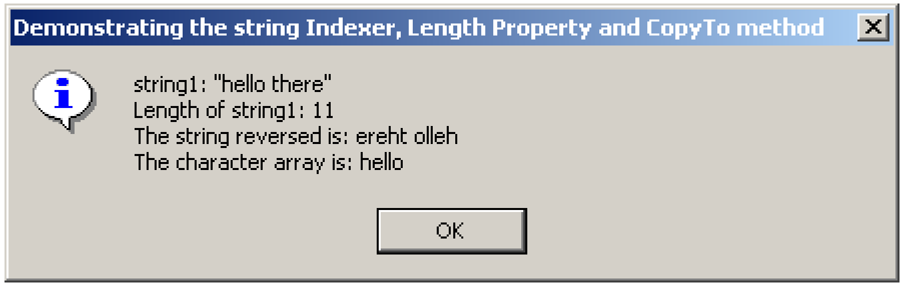
```
// Fig. 16.2: StringMethods.cs
// Using the indexer, property Length and method CopyTo
// of class String.

using System;
using System.Windows.Forms;

// creates string objects and displays results of using
// indexer and methods Length and CopyTo
class StringMethods
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string string1, output;
      char[] characterArray;

      string1 = "hello there";
      characterArray = new char[ 5 ];

      // output string
      output =
         "string1: \"" + string1 + "\"";

      // test Length property
      output += "\nLength of string1: " + string1.Length;

      // loop through character in string1 and display
      // reversed
      output += "\nThe string reversed is: ";

      for ( int i = string1.Length - 1; i >= 0; i-- )
         output += string1[ i ];

```

Outline

StringMethods.cs

```
         // copy characters from string1 into characterArray
         string1.CopyTo( 0, characterArray, 0, 5 );
         output += "\nThe character array is: ";

         for ( int i = 0 ; i < characterArray.Length; i++ )
            output += characterArray[ i ];

         MessageBox.Show( output, "Demonstrating the string " +
            "Indexer, Length Property and CopyTo method",
            MessageBoxButtons.OK, MessageBoxIcon.Information );

      } // end method Main

   } // end class StringMethods
```

Program Output


Demonstrating the string Indexer, Length Property and CopyTo method

```
string1: "hello there"
Length of string1: 11
The string reversed is: ereht olleh
The character array is: hello
```

16.5 Comparing Strings

- String comparison
  - Greater than
  - Less than
- Method Equals
  - Test objects for equality
  - Return a Boolean
  - Uses lexicographical comparison

Outline

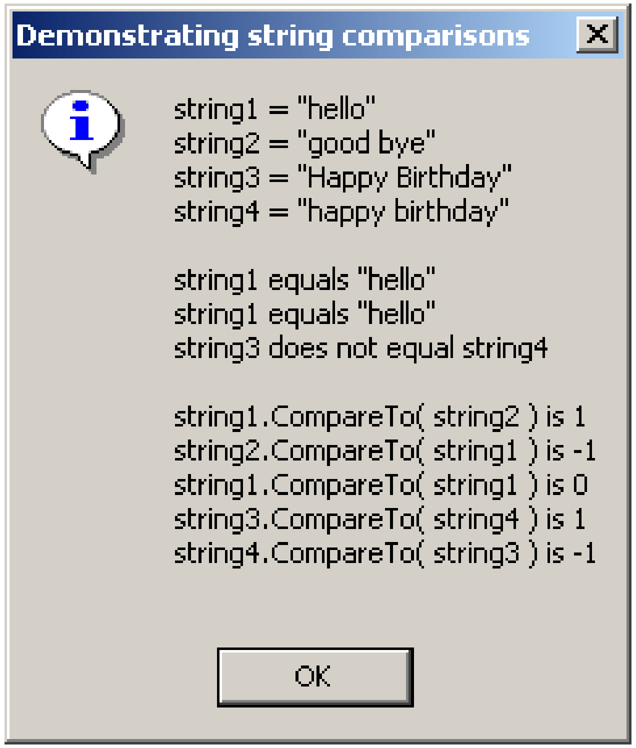
StringCompare.cs

```
// Fig. 16.3: StringCompare.cs
// Comparing strings.

using System;
using System.Windows.Forms;

// compare a number of strings
class StringCompare
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string string1 = "hello";
      string string2 = "good bye";
      string string3 = "Happy Birthday";
      string string4 = "happy birthday";
      string output;

      // output values of four strings
      output = "string1 = \"" + string1 + "\"" +
         "\nstring2 = \"" + string2 + "\"" +
         "\nstring3 = \"" + string3 + "\"" +
         "\nstring4 = \"" + string4 + "\"\n\n";

      // test for equality using Equals method
      if ( string1.Equals( "hello" ) )
         output += "string1 equals \"hello\"\n";
      else
         output += "string1 does not equal \"hello\"\n";

      // test for equality with ==
      if ( string1 == "hello" )
         output += "string1 equals \"hello\"\n";
```

StringCompare.cs

```
         else
            output += "string1 does not equal \"hello\"\n";

         // test for equality comparing case
         if ( String.Equals( string3, string4 ) )
            output += "string3 equals string4\n";
         else
            output += "string3 does not equal string4\n";

         // test CompareTo
         output += "\nstring1.CompareTo( string2 ) is " +
            string1.CompareTo( string2 ) + "\n" +
            "string2.CompareTo( string1 ) is " +
            string2.CompareTo( string1 ) + "\n" +
            "string1.CompareTo( string1 ) is " +
            string1.CompareTo( string1 ) + "\n" +
            "string3.CompareTo( string4 ) is " +
            string3.CompareTo( string4 ) + "\n" +
            "string4.CompareTo( string3 ) is " +
            string4.CompareTo( string3 ) + "\n\n";

         MessageBox.Show( output, "Demonstrating string " +
            "comparisons", MessageBoxButtons.OK,
            MessageBoxIcon.Information );

      } // end method Main

   } // end class StringCompare
```

StringCompare.cs
Program Output



```
string1 = "hello"
string2 = "good bye"
string3 = "Happy Birthday"
string4 = "happy birthday"

string1 equals "hello"
string1 equals "hello"
string3 does not equal string4

string1.CompareTo( string2 ) is 1
string2.CompareTo( string1 ) is -1
string1.CompareTo( string1 ) is 0
string3.CompareTo( string4 ) is 1
string4.CompareTo( string3 ) is -1
```

Outline

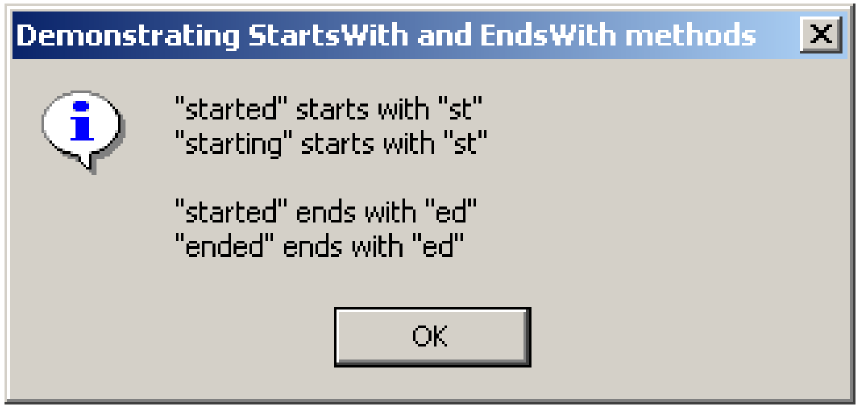
StringStartEnd.cs

```
// Fig. 16.4: StringStartEnd.cs
// Demonstrating StartsWith and EndsWith methods.

using System;
using System.Windows.Forms;

// testing StartsWith and EndsWith
class StringStartEnd
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string[] strings =
            { "started", "starting", "ended", "ending" };
      string output = "";

      //test every string to see if it starts with "st"
      for ( int i = 0; i < strings.Length; i++ )

         if ( strings[ i ].StartsWith( "st" ) )
            output += "\"" + strings[ i ] + "\"" +
               " starts with \"st\"\n";

      output += "\n";

      // test every string to see if it ends with "ed"
      for ( int i = 0; i < strings.Length; i ++ )

         if ( strings[ i ].EndsWith( "ed" ) )
            output += "\"" + strings[ i ] + "\"" +
               " ends with \"ed\"\n";

      MessageBox.Show( output, "Demonstrating StartsWith and " +
         "EndsWith methods", MessageBoxButtons.OK,
         MessageBoxIcon.Information );

   } // end method Main

} // end class StringStartEnd
```

Outline

StringStartEnd.cs

Program Output


Demonstrating StartsWith and EndsWith methods

```
"started" starts with "st"
"starting" starts with "st"

"started" ends with "ed"
"ended" ends with "ed"
```

16.7 Locating Characters and Substrings in Strings

- Application of **String** methods:
  - IndexOf
  - IndexOfAny
  - LastIndexOf
  - LastIndexOfAny

Outline

StringIndexMethods.cs

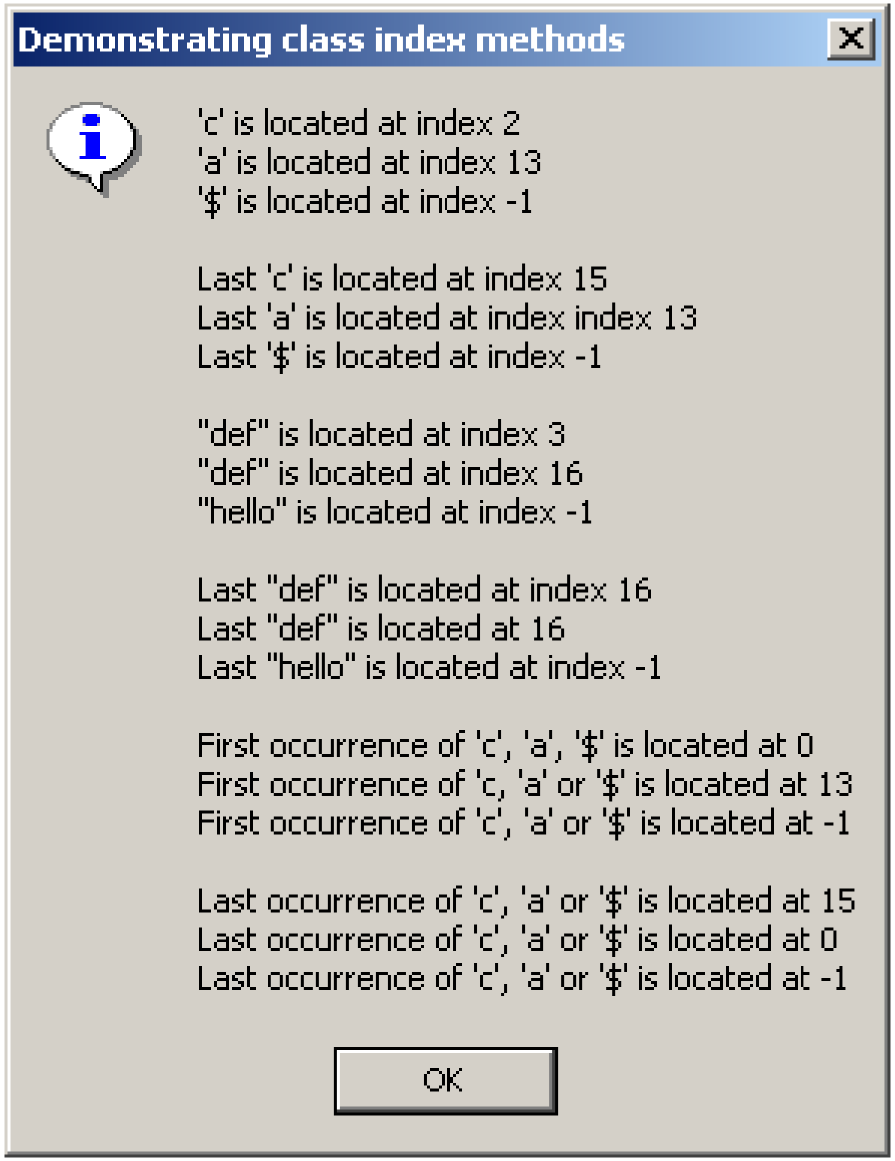
```
// Fig. 16.6: StringIndexMethods.cs
// Using String searching methods.

using System;
using System.Windows.Forms;

// testing indexing capabilities of strings
class StringIndexMethods
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string letters = "abcdefghijklmabcdefghijklm";
      string output = "";
      char[] searchLetters = { 'c', 'a', '$' };

      // test IndexOf to locate a character in a string
      output += "'c' is located at index " +
         letters.IndexOf( 'c' );

      output += "\n'a' is located at index " +
         letters.IndexOf( 'a', 1 );

      output += "\n'$' is located at index " +
         letters.IndexOf( '$', 3, 5 );

      // test LastIndexOf to find a character in a string
      output += "\n\nLast 'c' is located at " +
         "index " + letters.LastIndexOf( 'c' );

      output += "\nLast 'a' is located at index " +
         letters.LastIndexOf( 'a', 25 );

     output += "\nLast '$' is located at index " +
        letters.LastIndexOf( '$', 15, 5 );

     // test IndexOf to locate a substring in a string
     output += "\n\n\"def\" is located at" +
        " index " + letters.IndexOf( "def" );

     output += "\n\"def\" is located at index " +
        letters.IndexOf( "def", 7 );

     output += "\n\"hello\" is located at index " +
        letters.IndexOf( "hello", 5, 15 );

     // test LastIndexOf to find a substring in a string
     output += "\n\nLast \"def\" is located at index " +
        letters.LastIndexOf( "def" );

     output += "\nLast \"def\" is located at " +
        letters.LastIndexOf( "def", 25 );

     output += "\nLast \"hello\" is located at index " +
        letters.LastIndexOf( "hello", 20, 15 );

     // test IndexOfAny to find first occurrence of character
     // in array
     output += "\n\nFirst occurrence of 'c', 'a', '$' is " +
        "located at " + letters.IndexOfAny( searchLetters );

     output += "\nFirst occurrence of 'c, 'a' or '$' is " +
        "located at " + letters.IndexOfAny( searchLetters, 7 );

     output += "\nFirst occurrence of 'c', 'a' or '$' is " +
        "located at " + letters.IndexOfAny( searchLetters, 20, 5 );

```

StringIndexMethods.cs

StringIndexMethods.cs

```
         // test LastIndexOfAny to find last occurrence of character
         // in array
         output += "\n\nLast occurrence of 'c', 'a' or '$' is " +
            "located at " + letters.LastIndexOfAny( searchLetters );

         output += "\nLast occurrence of 'c', 'a' or '$' is " +
            "located at " + letters.LastIndexOfAny( searchLetters, 1 );

         output += "\nLast occurrence of 'c', 'a' or '$' is " +
            "located at " + letters.LastIndexOfAny(
            searchLetters, 25, 5 );

         MessageBox.Show( output,
            "Demonstrating class index methods",
            MessageBoxButtons.OK, MessageBoxIcon.Information );

      } // end method Main

   } // end class StringIndexMethods
```

StringIndexMethods.cs
Program Output



```
'c' is located at index 2
'a' is located at index 13
'$' is located at index -1

Last 'c' is located at index 15
Last 'a' is located at index index 13
Last '$' is located at index -1

"def" is located at index 3
"def" is located at index 16
"hello" is located at index -1

Last "def" is located at index 16
Last "def" is located at 16
Last "hello" is located at index -1

First occurrence of 'c', 'a', '$' is located at 0
First occurrence of 'c, 'a' or '$' is located at 13
First occurrence of 'c', 'a' or '$' is located at -1

Last occurrence of 'c', 'a' or '$' is located at 15
Last occurrence of 'c', 'a' or '$' is located at 0
Last occurrence of 'c', 'a' or '$' is located at -1
```

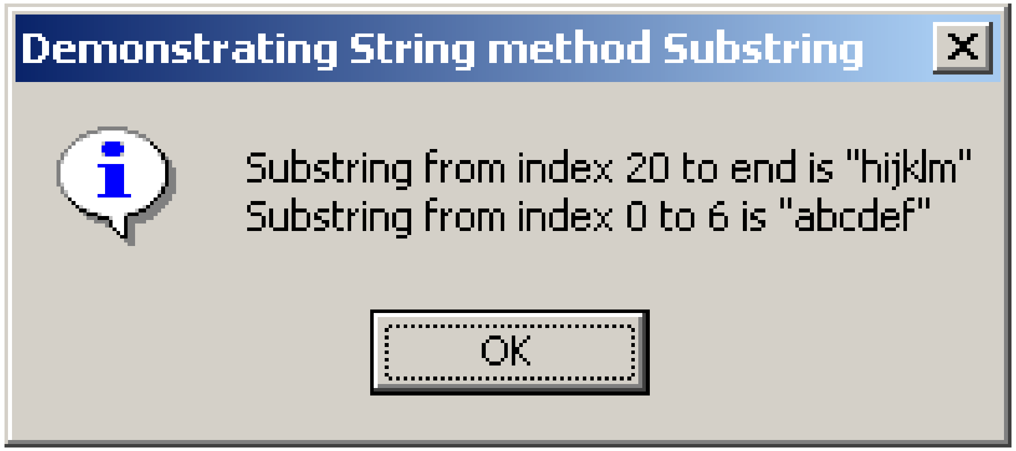
16.8 Extracting Substrings from Strings

- Substring methods
  - Create new string
  - Method return new string

SubString.cs

```
// Fig. 16.7: SubString.cs
// Demonstrating the String Substring method.

using System;
using System.Windows.Forms;

// creating substrings
class SubString
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string letters = "abcdefghijklmabcdefghijklm";
      string output = "";

      // invoke Substring method and pass it one parameter
      output += "Substring from index 20 to end is \"" +
         letters.Substring( 20 ) + "\"\n";

      // invoke Substring method and pass it two parameters
      output += "Substring from index 0 to 6 is \"" +
         letters.Substring( 0, 6 ) + "\"";

      MessageBox.Show( output,
         "Demonstrating String method Substring",
         MessageBoxButtons.OK, MessageBoxIcon.Information );

   } // end method Main

} // end class SubString
```

SubString.cs
Program Output



```
Substring from index 20 to end is "hijklm"
Substring from index 0 to 6 is "abcdef"
```

16.9 Concatenating Strings

- Static method Concat
  - Takes two string and return a new string

Outline

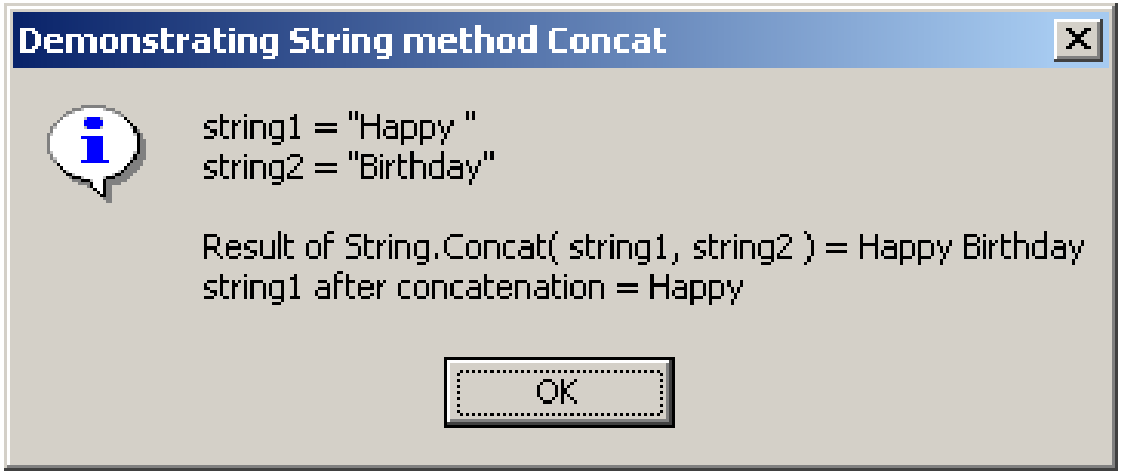
SubConcatination.cs

```
// Fig. 16.8: SubConcatination.cs
// Demonstrating String class Concat method.

using System;
using System.Windows.Forms;

// concatenates strings using String method Concat
class StringConcatenation
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string string1 = "Happy ";
      string string2 = "Birthday";
      string output;

      output = "string1 = \"" + string1 + "\"\n" +
         "string2 = \"" + string2 + "\"";

      output +=
         "\n\nResult of String.Concat( string1, string2 ) = " +
         String.Concat( string1, string2 );

      output += "\nstring1 after concatenation = " + string1;

      MessageBox.Show( output,
         "Demonstrating String method Concat",
         MessageBoxButtons.OK, MessageBoxIcon.Information );

   } // end method Main

} // end class StringConcatenation
```

Outline

`SubConcatination.cs`

Program Output



```
string1 = "Happy "
string2 = "Birthday"

Result of String.Concat( string1, string2 ) = Happy Birthday
string1 after concatenation = Happy
```

16.10 Miscellaneous String Methods

- Method Replace
- Method ToUpper
  - Replace lower case letter
  - Original string remain unchanged
  - Original string return if no occurrence matched
- Method ToLower
  - Replace lower case letter
  - Original string remain unchanged
  - Original string return if no occurrence matched

16.10 Miscellaneous String Methods

- Method ToString
  - Can be called to obtain a string representation of any object
- Method Trim
  - Remove whitespaces
  - Remove characters in the array argument

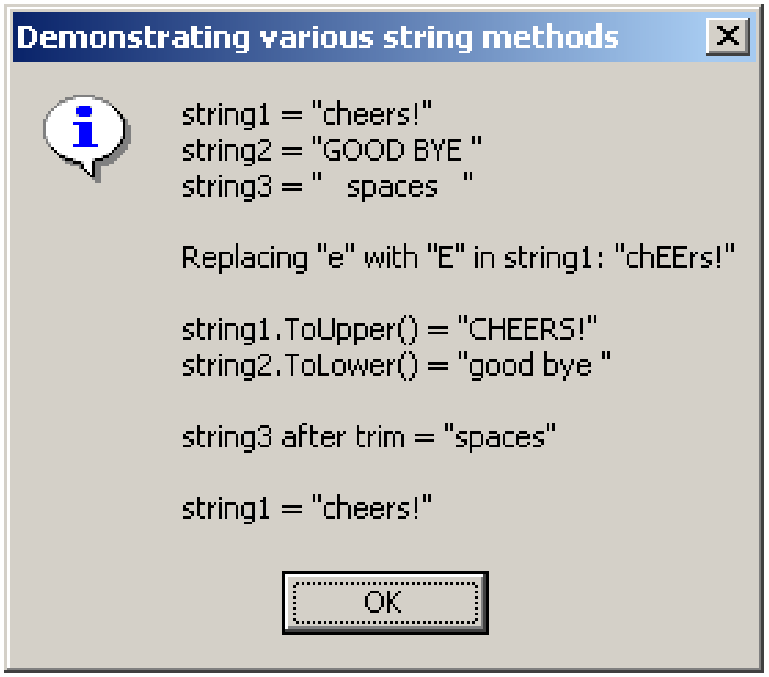
Outline

StringMiscellaneous2.cs

```
// Fig. 16.9: StringMiscellaneous2.cs
// Demonstrating String methods Replace, ToLower, ToUpper, Trim
// and ToString.

using System;
using System.Windows.Forms;

// creates strings using methods Replace, ToLower, ToUpper, Trim
class StringMethods2
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string string1 = "cheers!";
      string string2 = "GOOD BYE ";
      string string3 = "   spaces   ";
      string output;

      output = "string1 = \"" + string1 + "\"\n" +
         "string2 = \"" + string2 + "\"\n" +
         "string3 = \"" + string3 + "\"";

      // call method Replace
      output +=
         "\n\nReplacing \"e\" with \"E\" in string1: \"" +
         string1.Replace( 'e', 'E' ) + "\"";

      // call ToLower and ToUpper
      output += "\n\nstring1.ToUpper() = \"" +
         string1.ToUpper() + "\"\nstring2.ToLower() = \"" +
         string2.ToLower() + "\"";

      // call Trim method
      output += "\n\nstring3 after trim = \"" +
         string3.Trim() + "\"";

      // call ToString method
      output += "\n\nstring1 = \"" + string1.ToString() + "\"";

      MessageBox.Show( output,
         "Demonstrating various string methods",
         MessageBoxButtons.OK, MessageBoxIcon.Information );

   } // end method Main

} // end class StringMethods2
```

StringMiscellaneous2.cs

Program Output


Demonstrating various string methods

```
string1 = "cheers!"
string2 = "GOOD BYE "
string3 = "   spaces   "

Replacing "e" with "E" in string1: "chEErs!"

string1.ToUpper() = "CHEERS!"
string2.ToLower() = "good bye "

string3 after trim = "spaces"

string1 = "cheers!"
```

16.10 Class StringBuilder

- Class StringBuilder
  - Create and manipulate dynamic string information
  - Capable of resizing

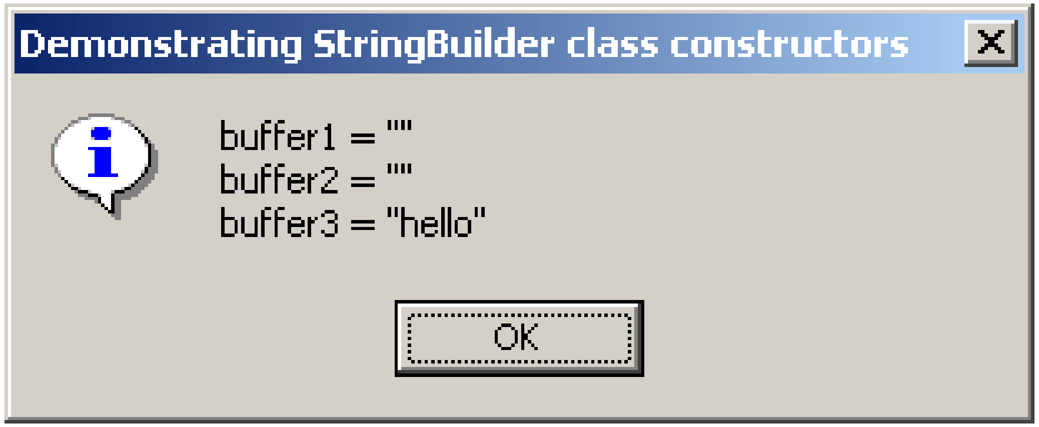
Outline

StringBuilderConstructor.cs

```
// Fig. 16.10: StringBuilderConstructor.cs
// Demonstrating StringBuilder class constructors.

using System;
using System.Windows.Forms;
using System.Text;

// creates three StringBuilder with three constructors
class StringBuilderConstructor
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      StringBuilder buffer1, buffer2, buffer3;
      string output;

      buffer1 = new StringBuilder();
      buffer2 = new StringBuilder( 10 );
      buffer3 = new StringBuilder( "hello" );

      output = "buffer1 = \"" + buffer1.ToString() + "\"\n";

      output += "buffer2 = \"" + buffer2.ToString() + "\"\n";

      output += "buffer3 = \"" + buffer3.ToString() + "\"\n";

      MessageBox.Show( output,
         "Demonstrating StringBuilder class constructors",
         MessageBoxButtons.OK, MessageBoxIcon.Information );

   } // end method Main

} // end class StringBuilderConstructor
```

StringBuilderConstructor.cs

Program Output



```
buffer1 = ""
buffer2 = ""
buffer3 = "hello"
```

16.11 StringBuilder Indexer, Length and Capacity Properties, and EnsureCapacity Method

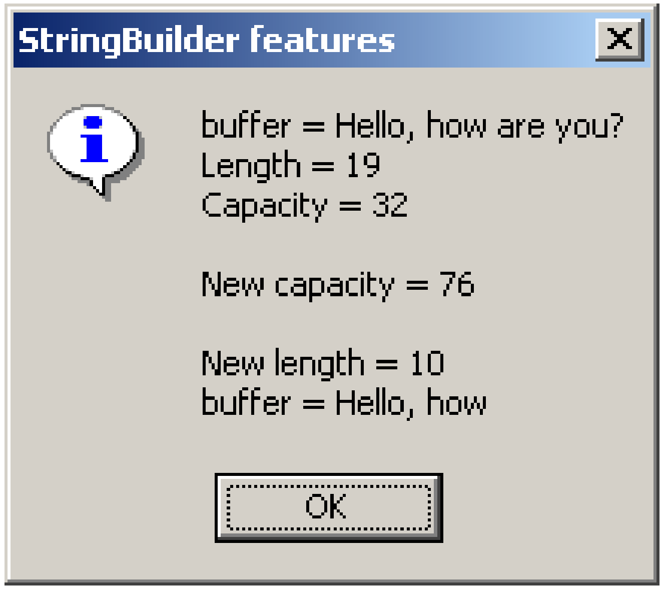
- Method EnsureCapacity
  - Allow programmers to guarantee StringBuilder has capacity that reduces the number of times capacity must be increased
  - Length property return number of character in StringBuilder
  - Capacity property return number StringBuilder can store without allocating memory

```
// Fig. 16.11: StringBuilderFeatures.cs
// Demonstrating some features of class StringBuilder.

using System;
using System.Windows.Forms;
using System.Text;

// uses some of class StringBuilder's methods
class StringBuilderFeatures
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      StringBuilder buffer =
         new StringBuilder( "Hello, how are you?" );

      // use Length and Capacity properties
      string output = "buffer = " + buffer.ToString() +
         "\nLength = " + buffer.Length +
         "\nCapacity = " + buffer.Capacity;

      // use EnsureCapacity method
      buffer.EnsureCapacity( 76 );

      output += "\n\nNew capacity = " +
         buffer.Capacity;

      // truncate StringBuilder by setting Length property
      buffer.Length = 10;

      output += "\n\nNew length = " +
         buffer.Length + "\nbuffer = ";

```

StringBuilderFeatures.cs

```
         // use StringBuilder indexer
         for ( int i = 0; i < buffer.Length; i++ )
            output += buffer[ i ];

         MessageBox.Show( output, "StringBuilder features",
            MessageBoxButtons.OK, MessageBoxIcon.Information );

      } // end method Main

   } // end class StringBuilderFeatures
```

StringBuilderFeatures.cs



```
buffer = Hello, how are you?
Length = 19
Capacity = 32

New capacity = 76

New length = 10
buffer = Hello, how
```

Program Output

16.12 StringBuilder Append and AppendFormat Methods

- Append method
  - Allow various data-type values to append to the end of a StringBuilder
  - Convert argument into string
- AppendFormat method
  - Convert string to a specifiable format

Outline

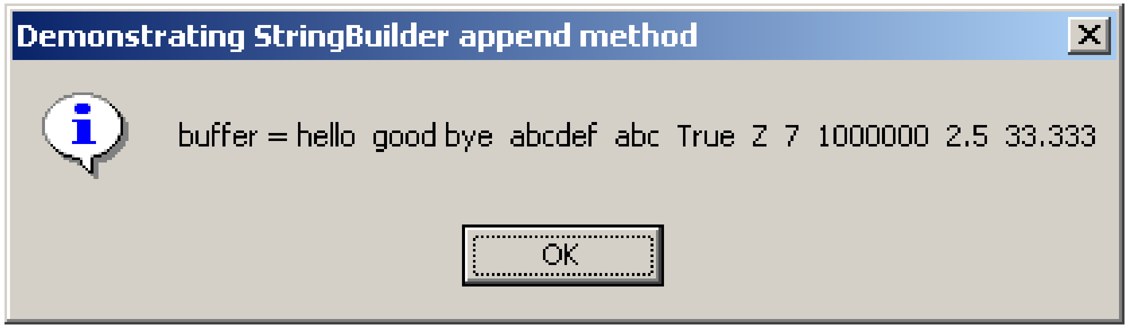
StringBuilderAppend.cs

```
// Fig. 16.12: StringBuilderAppend.cs
// Demonstrating StringBuilder Append methods.

using System;
using System.Windows.Forms;
using System.Text;

// testing the Append method
class StringBuilderAppend
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      object objectValue = "hello";
      string stringValue = "good bye";
      char[] characterArray = { 'a', 'b', 'c', 'd',
                                'e', 'f' };

      bool booleanValue = true;
      char characterValue = 'Z';
      int integerValue = 7;
      long longValue = 1000000;
      float floatValue = 2.5F;
      double doubleValue = 33.333;
      StringBuilder buffer = new StringBuilder();

      // use method Append to append values to buffer
      buffer.Append( objectValue );
      buffer.Append( "  " );
      buffer.Append( stringValue );
      buffer.Append( "  " );
      buffer.Append( characterArray );
      buffer.Append( "  " );
```

Outline

StringBuilderAppend.cs

```
         buffer.Append( characterArray, 0, 3 );
         buffer.Append( "  " );
         buffer.Append( booleanValue );
         buffer.Append( "  " );
         buffer.Append( characterValue );
         buffer.Append( "  " );
         buffer.Append( integerValue );
         buffer.Append( "  " );
         buffer.Append( longValue );
         buffer.Append( "  " );
         buffer.Append( floatValue );
         buffer.Append( "  " );
         buffer.Append( doubleValue );

         MessageBox.Show( "buffer = " + buffer.ToString(),
            "Demonstrating StringBuilder append method",
            MessageBoxButtons.OK, MessageBoxIcon.Information );

      } // end method Main

   } // end class StringBuilderAppend
```


Demonstrating StringBuilder append method

```
buffer = hello  good bye  abcdef  abc  True  Z  7  1000000  2.5  33.333
```

Program Output

Outline

StringBuilderAppendFormat.cs

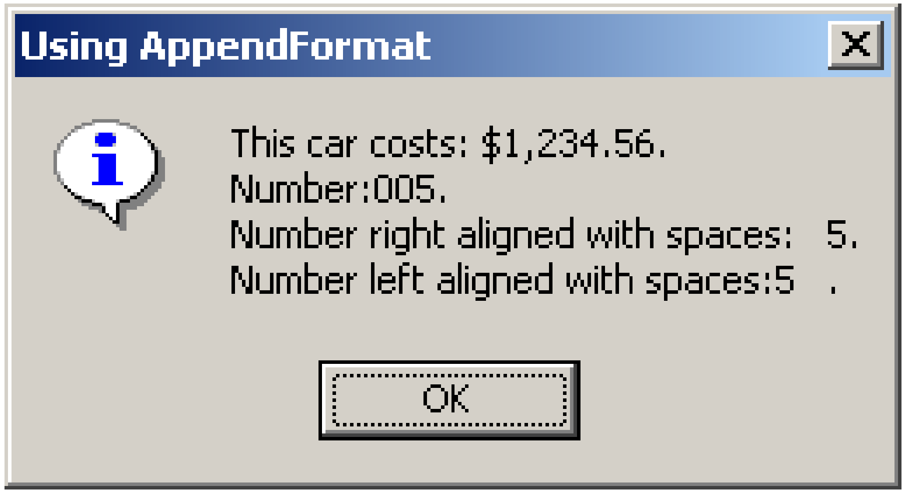
```
// Fig. 16.13: StringBuilderAppendFormat.cs
// Demonstrating method AppendFormat.

using System;
using System.Windows.Forms;
using System.Text;

// use the AppendFormat method
class StringBuilderAppendFormat
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      StringBuilder buffer = new StringBuilder();
      string string1, string2;

      // formatted string
      string1 = "This {0} costs: {1:C}.\n";

      // string1 argument array
      object[] objectArray = new object[ 2 ];

      objectArray[ 0 ] = "car";
      objectArray[ 1 ] = 1234.56;

      // append to buffer formatted string with argument
      buffer.AppendFormat( string1, objectArray );

      // formatted string
      string2 = "Number:{0:d3}.\n" +
         "Number right aligned with spaces:{0, 4}.\n" +
         "Number left aligned with spaces:{0, -4}.";

```

Outline

StringBuilderAppendFormat.cs

```
         // append to buffer formatted string with argument
         buffer.AppendFormat( string2, 5 );

         // display formatted strings
         MessageBox.Show( buffer.ToString(), "Using AppendFormat",
            MessageBoxButtons.OK, MessageBoxIcon.Information );

      } // end method Main

   } // end class StringBuilderAppendFormat
```

Program Output



```
This car costs: $1,234.56.
Number:005.
Number right aligned with spaces:   5.
Number left aligned with spaces:5   .
```

16.13 StringBuilder Insert, Remove and Replace Methods

- Insert method
  - Insert into at any position
- Remove method
  - Takes two argument
- Replace method
  - Substitute specified string

Outline

StringBuilderInsertRemove.cs

```
// Fig. 16.14: StringBuilderInsertRemove.cs
// Demonstrating methods Insert and Remove of the
// StringBuilder class.

using System;
using System.Windows.Forms;
using System.Text;

// test the Insert and Remove methods
class StringBuilderInsertRemove
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      object objectValue = "hello";
      string stringValue = "good bye";
      char[] characterArray = { 'a', 'b', 'c',
                                'd', 'e', 'f' };

      bool booleanValue = true;
      char characterValue = 'K';
      int integerValue = 7;
      long longValue = 10000000;
      float floatValue = 2.5F;
      double doubleValue = 33.333;
      StringBuilder buffer = new StringBuilder();
      string output;

      // insert values into buffer
      buffer.Insert(0, objectValue);
      buffer.Insert(0, "  ");
      buffer.Insert(0, stringValue);
      buffer.Insert(0, "  ");
```

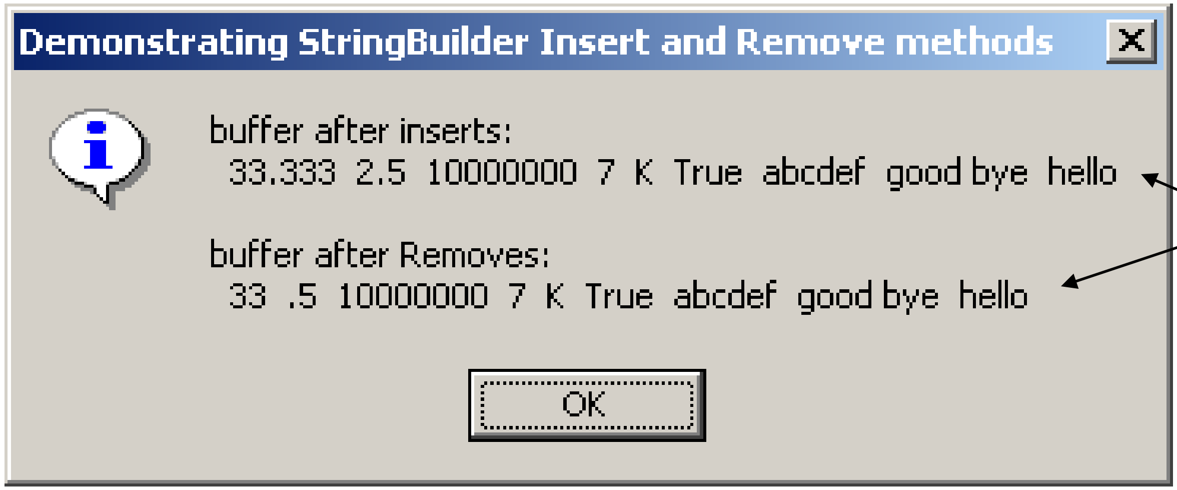
Outline

StringBuilderInsertRemove.cs

```
         buffer.Insert(0, characterArray);
         buffer.Insert(0, "  ");
         buffer.Insert(0, booleanValue);
         buffer.Insert(0, "  ");
         buffer.Insert(0, characterValue);
         buffer.Insert(0, "  ");
         buffer.Insert(0, integerValue);
         buffer.Insert(0, "  ");
         buffer.Insert(0, longValue);
         buffer.Insert(0, "  ");
         buffer.Insert(0, floatValue);
         buffer.Insert(0, "  ");
         buffer.Insert(0, doubleValue);
         buffer.Insert(0, "  ");

         output = "buffer after inserts: \n" +
            buffer.ToString() + "\n\n";

         buffer.Remove( 10, 1 ); // delete 2 in 2.5
         buffer.Remove( 2, 4 );  // delete .333 in 33.333

         output += "buffer after Removes:\n" +
            buffer.ToString();

         MessageBox.Show( output, "Demonstrating StringBuilder " +
            "Insert and Remove methods", MessageBoxButtons.OK,
            MessageBoxIcon.Information );

      } // end method Main

   } // end class StringBuilderInsertRemove
```

Outline

`StringBuilderInsertRemove.cs`
Program Output



```
buffer after inserts:
  33.333  2.5  10000000  7  K  True  abcdef  good bye  hello

buffer after Removes:
  33  .5  10000000  7  K  True  abcdef  good bye  hello
```

Changes to the string from the Remove method call

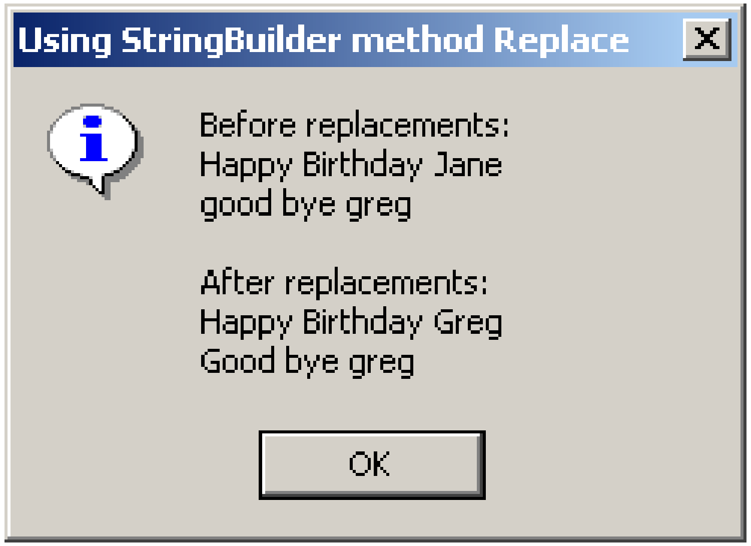
```
// Fig. 16.15: StringBuilderReplace.cs
// Demonstrating method Replace.

using System;
using System.Windows.Forms;
using System.Text;

// testing the Replace method
class StringBuilderReplace
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      StringBuilder builder1 =
         new StringBuilder( "Happy Birthday Jane" );

      StringBuilder builder2 =
         new StringBuilder( "good bye greg" );

      string output = "Before replacements:\n" +
         builder1.ToString() + "\n" + builder2.ToString();

      builder1.Replace( "Jane", "Greg" );
      builder2.Replace( 'g', 'G', 0, 5 );

      output += "\n\nAfter replacements:\n" +
         builder1.ToString() + "\n" + builder2.ToString();

```

StringBuilderReplace.cs

```
            MessageBox.Show( output,
               "Using StringBuilder method Replace",
               MessageBoxButtons.OK, MessageBoxIcon.Information );

         } // end method Main

      } // end class StringBuilderReplace
```

StringBuilderReplace.cs

Program Output



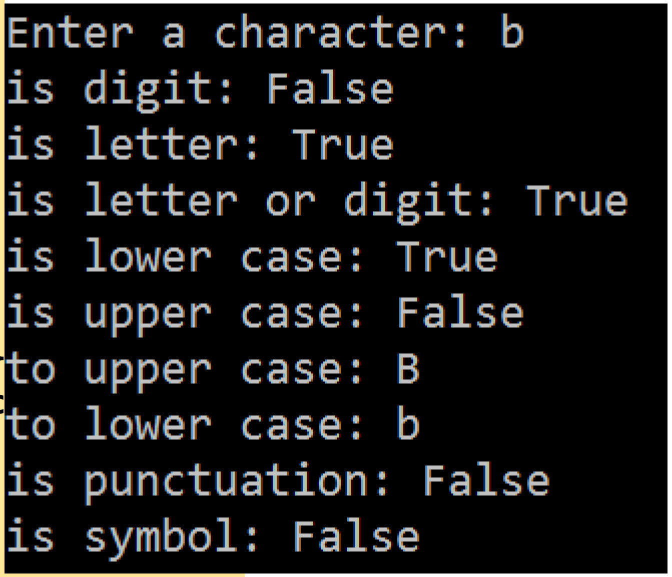
```
Before replacements:
Happy Birthday Jane
good bye greg

After replacements:
Happy Birthday Greg
Good bye greg
```

16.14 Char Methods

- Structure Char
  - For character usage
  - Most methods are static
  - Methods:
    - IsLower
    - IsUpper
    - ToUpper
    - ToLower
    - IsPunctuation
    - IsSymbol
    - IsWhiteSpace

CharMethods.cs

```
static void Main(string[] args)
{
      Console.Write("Enter a character: ");
      var character = char.Parse(Console.ReadLine());

      Console.WriteLine($"is digit: {char.IsDigit(character)}");
      Console.WriteLine($"is letter: {char.IsLetter(character)}");
      Console.WriteLine(
         $"is letter or digit: {char.IsLetterOrDigit(character)}");
      Console.WriteLine($"is lower case:
         {char.IsLower(character)}");
      Console.WriteLine($"is upper case:
         {char.IsUpper(character)}");
      Console.WriteLine($"to upper case:
         {char.ToUpper(character)}");
      Console.WriteLine($"to lower case:
         {char.ToLower(character)}");
      Console.WriteLine(
         $"is punctuation: {char.IsPunctuation(character)}
      Console.WriteLine($"is symbol: {char.IsSymbol(charac
}
```



```
Enter a character: b
is digit: False
is letter: True
is letter or digit: True
is lower case: True
is upper case: False
to upper case: B
to lower case: b
is punctuation: False
is symbol: False
```

16.16 Regular Expression and Class Regex

- Regular expression
  - Specially formatted strings used to find patterns in text.
  - They can be used to ensure that data is in a particular format.
- Class Regex
  - Method Match
    - Return object of class Match
  - Method Matches
    - Return a MatchCollection object

Outline

RegexMatches.cs

```
// Fig. 16.20: RegexMatches.cs
// Demonstrating Class Regex.

using System;
using System.Windows.Forms;
using System.Text.RegularExpressions;

// test out regular expressions
class RegexMatches
{
   // The main entry point for the application.

   static void Main( string[] args )
   {
      string output = "";

      // create regular expression
      Regex expression =
         new Regex( @"J.*\d[0-35-9]-\d\d-\d\d" );

      string string1 = "Jane's Birthday is 05-12-75\n" +
         "Dave's Birthday is 11-04-68\n" +
         "John's Birthday is 04-28-73\n" +
         "Joe's Birthday is 12-17-77";

      // match regular expression to string and
      // print out all matches
      foreach (var myMatch in expression.Matches( string1 ) )
         output += myMatch.ToString() + "\n";

```

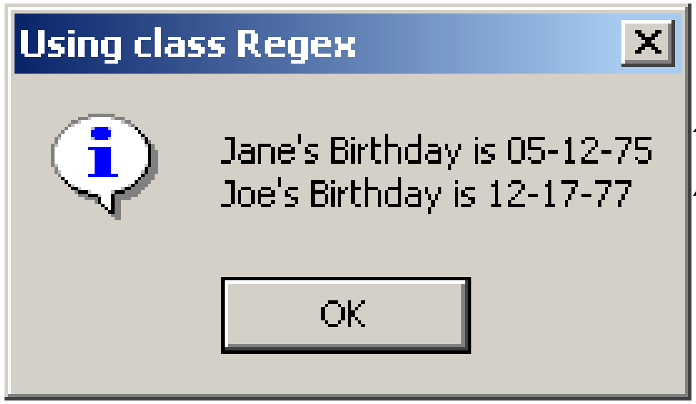
foreach loop iterates through each **Match**

RegexMatches.cs

```
            MessageBox.Show( output, "Using class Regex",
               MessageBoxButtons.OK, MessageBoxIcon.Information );

         } // end method Main

      } // end class RegexMatches
```

Output show the two matches from **string1** that have same pattern specified

Program Output

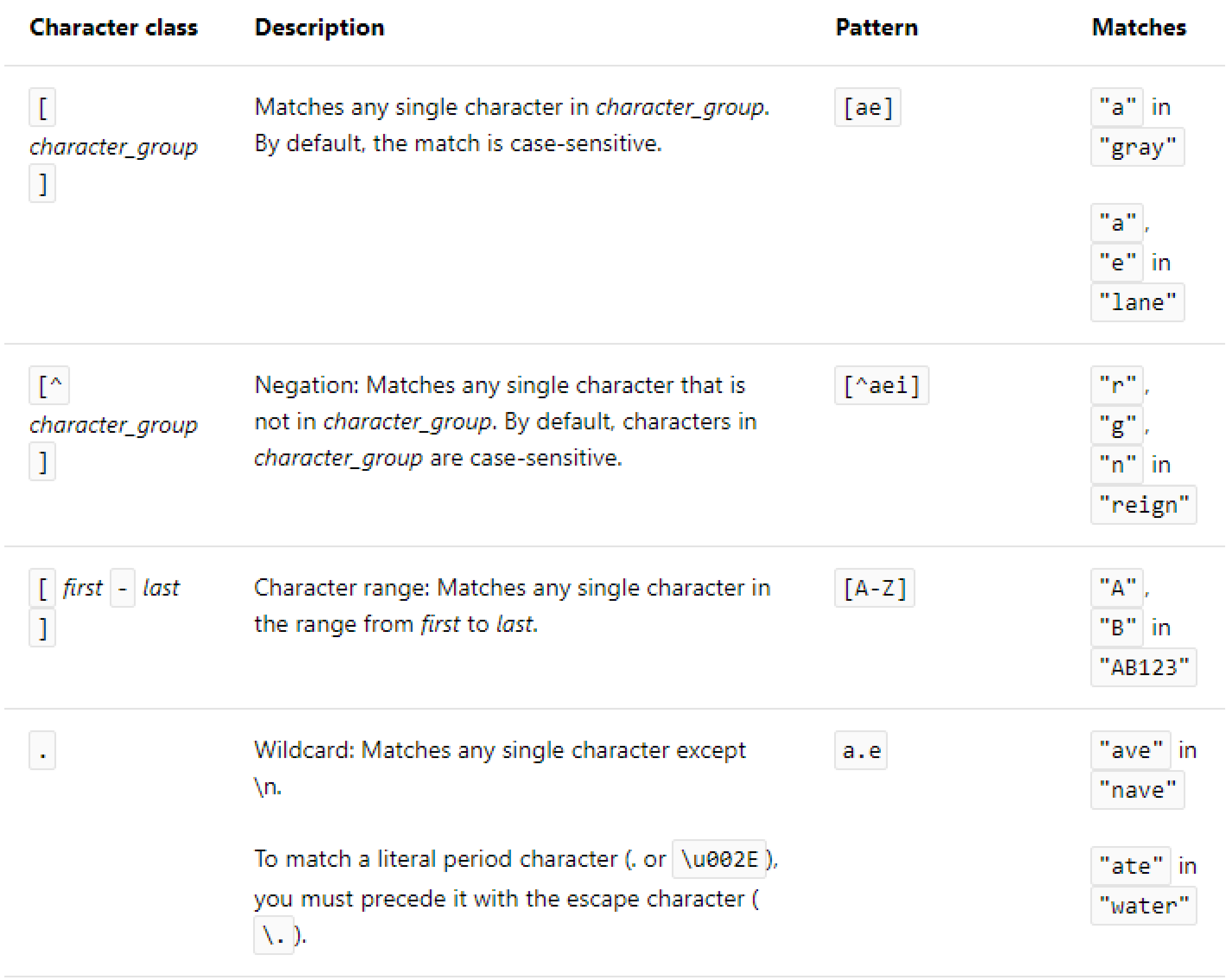


```
Jane's Birthday is 05-12-75
Joe's Birthday is 12-17-77
```

16.16 Regular Expressions and Class `Regex`

Quantifier	Description	Pattern	Matches
`*`	Matches the previous element zero or more times.	`\d*\.\d`	`".0"`, `"19.9"`, `"219.9"`
`+`	Matches the previous element one or more times.	`"be+"`	`"bee"` in `"been"`, `"be"` in `"bent"`
`?`	Matches the previous element zero or one time.	`"rai?n"`	`"ran"`, `"rain"`
`{` *n* `}`	Matches the previous element exactly *n* times.	`",\d{3}"`	`",043"` in `"1,043.6"`, `",876"`, `",543"`, and `",210"` in `"9,876,543,210"`
`{` *n* `,}`	Matches the previous element at least *n* times.	`"\d{2,}"`	`"166"`, `"29"`, `"1930"`
`{` *n* `,` *m* `}`	Matches the previous element at least *n* times, but no more than *m* times.	`"\d{3,5}"`	`"166"`, `"17668"` `"19302"` in `"193024"`

16.16 Regular Expressions and Class `Regex`



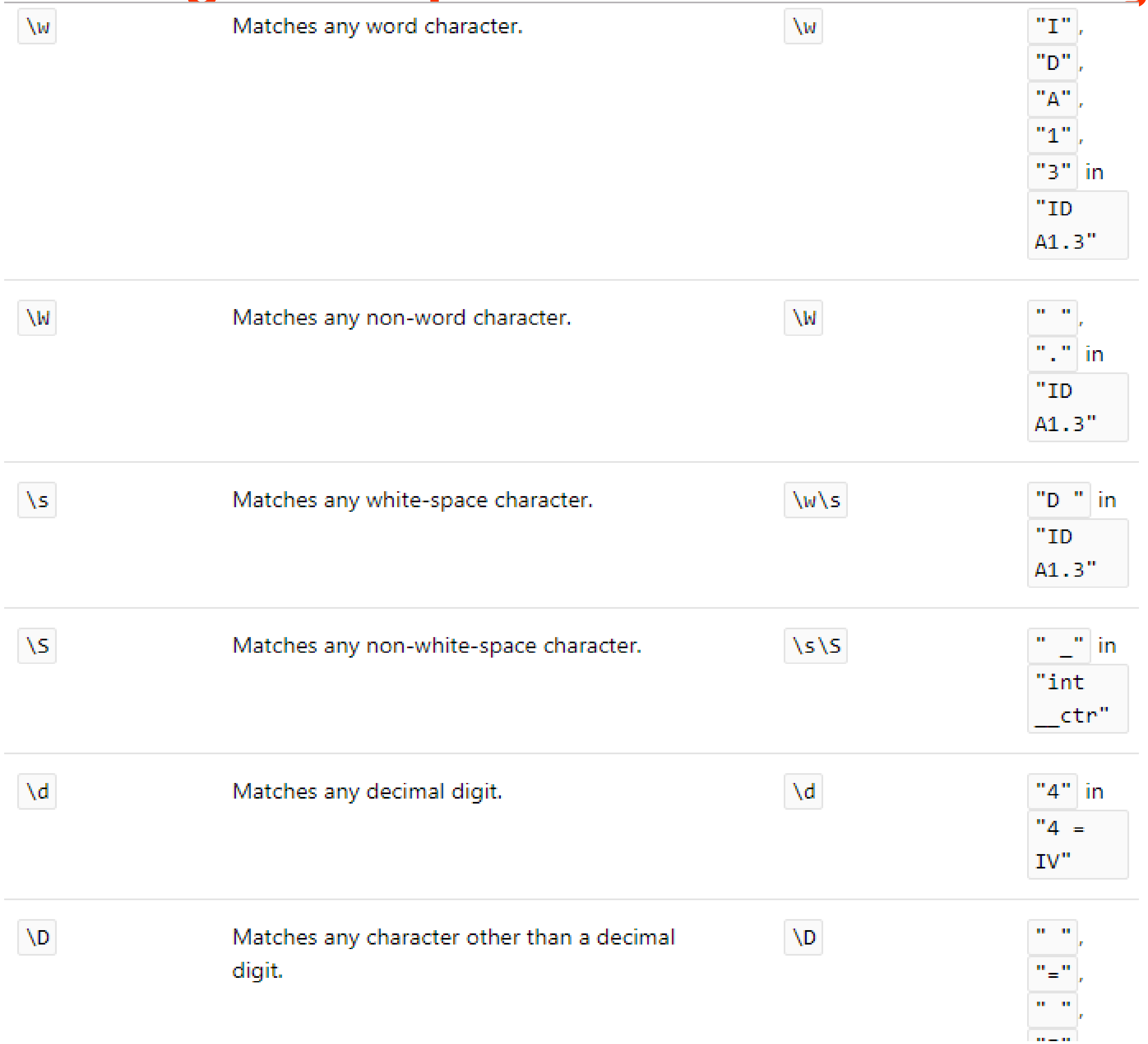
Character class	Description	Pattern	Matches

`[` *character_group* `]`	Matches any single character in *character_group*. By default, the match is case-sensitive.	`[ae]`	`"a"` in `"gray"`
			
			`"a"`, `"e"` in `"lane"`

`[^` *character_group* `]`	Negation: Matches any single character that is not in *character_group*. By default, characters in *character_group* are case-sensitive.	`[^aei]`	`"r"`, `"g"`, `"n"` in `"reign"`
`[` *first* `-` *last* `]`	Character range: Matches any single character in the range from *first* to *last*.	`[A-Z]`	`"A"`, `"B"` in `"AB123"`

`.`	Wildcard: Matches any single character except \n.
	
	To match a literal period character (. or `\u002E`), you must precede it with the escape character (`\.`).
		`a.e`	`"ave"` in `"nave"`
			
			`"ate"` in `"water"`

16.16 Regular Expressions and Class Regex



`\w`	Matches any word character.	`\w`	`"I"`, `"D"`, `"A"`, `"1"`, `"3"` in `"ID A1.3"`
`\W`	Matches any non-word character.	`\W`	`" "`, `"."` in `"ID A1.3"`
`\s`	Matches any white-space character.	`\w\s`	`"D "` in `"ID A1.3"`
`\S`	Matches any non-white-space character.	`\s\S`	`" _"` in `"int __ctr"`
`\d`	Matches any decimal digit.	`\d`	`"4"` in `"4 = IV"`
`\D`	Matches any character other than a decimal digit.	`\D`	`" "`, `"="`, `" "`,

Outline

RegexSubstitution.cs

```
// Fig. 16.23: RegexSubstitution.cs
// Using Regex method Replace.

using System;
using System.Text.RegularExpressions;
using System.Windows.Forms;

// Summary description for RegexSubstitution.
public class RegexSubstitution1
{

   // The main entry point for the application.
   static void Main( string[] args )
   {
      string testString1 =
         "This sentence ends in 5 stars *****";

      string testString2 = "1, 2, 3, 4, 5, 6, 7, 8";
      Regex testRegex1 = new Regex( "stars" );
      Regex testRegex2 = new Regex( @"\d" );
      string[] results;
      string output = "Original String 1\t\t\t" + testString1;

      testString1 = Regex.Replace( testString1, @"\*", "^" );

      output += "\n^ substituted for *\t\t\t" + testString1;

      testString1 = testRegex1.Replace( testString1, "carets" );

      output += "\n\"carets\" substituted for \"stars\"\t" +
         testString1;

      output += "\nEvery word replaced by \"word\"\t" +
         Regex.Replace( testString1, @"\w+", "word" );

```

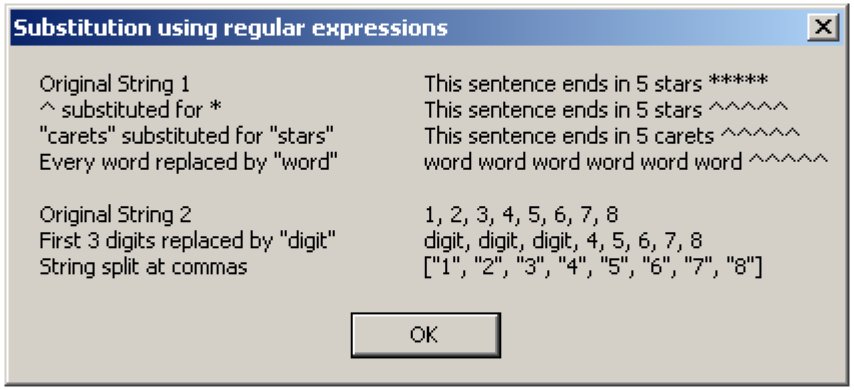
RegexSubstitutio

```
      output += "\n\nOriginal String 2\t\t\t" + testString2;
      results = Regex.Split( testString2, @",\s*" );

      foreach ( string result
      {
         output += "\"" + res
      }

      output = output.Substring( 0, output.Length - 2 ) + "]";

      MessageBox.Show( output,
         "Substitution using regular expressions" );

   } // end method Main

} // end class RegexSubstitution
```

M
in

Method **Split** returns array of substrings between matches for the regular expression

ken in any location
regular expression


Substitution using regular expressions

```
Original String 1                  This sentence ends in 5 stars *****
^ substituted for *                This sentence ends in 5 stars ^^^^^
"carets" substituted for "stars"   This sentence ends in 5 carets ^^^^^
Every word replaced by "word"      word word word word word word ^^^^^

Original String 2                  1, 2, 3, 4, 5, 6, 7, 8
First 3 digits replaced by "digit" digit, digit, digit, 4, 5, 6, 7, 8
String split at commas             ["1", "2", "3", "4", "5", "6", "7", "8"]
```