Chapter 16 – Strings, Chars and Reg Expressions

Outline

Many slides modified by Prof. L. Lilien (even if only without an explicit message).

16.1 Introduction

- String and character processing capabilities – useful in:
  - Text editors
  - Word processors
  - …
- Simple string uses shown in previous chapters
- Now: expanding on string and character processing -- capabilities of Framework Class Library
  - Class String and type char
  - Class StringBuilder
  - Classes Regex and Match

16.2 Fundamentals of Characters and Strings

- Importance of characters:
  - Are fundamental building blocks of C# source code
- Characters – represented as:

  1) Normal character (a b c … 0 1 2 3 … ! @ # …)

  2) Character constants = character represented by character code
     - Character code is an integer value
     - E.g., (in ASCII):

       character constant ‘r’ represented by character code 114
       character constant ‘R’ represented by character code 82
       character constant ‘$’ represented by character code 36

16.2 Fundamentals of Characters and Strings

- Character sets
  - ASCII character set
    - Represents English characters on many computers
    - See Appendix F
  - Unicode character set
    - International character set
      - Includes ASCII character set
    - See Appendix G (on textbook book CD-ROM only)

16.2 Fundamentals of Characters and Strings

- String
  - Object of class String in System namespace
  - A series of characters treated as a single unit
    - E.g. "this is a string"
- String constants = string literals = literal strings
    - E.g., "this is a string constant" is a literal string
  - Assigning string literal to string reference
    - E.g., `string color = "blue";`

16.2 Fundamentals of Characters and Strings

- Escape sequences and verbatim string syntax
  - Equivalent expressions:

```
string file = "C:\\MyFolder\\MySubolder\\MyFile.txt"
```

    - uses escape sequences

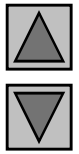
```
string file = @"C:\MyFolder\MySubolder\MyFile.txt"
```

    - uses verbatim string syntax
      - Also allows strings to span multiple lines

        (preserves newlines, spaces, tabs)

16.3 String Constructors

- Class String
  - Provides eight constructors
    - For initialing strings in various ways
      - 3 constructors shown in the next slide



Outline

StringConstructor.cs

```
1    // Fig. 15.1: StringConstructor.cs
2    // Demonstrating String class constructors.
3
4    using System;
5    using System.Windows.Forms;
6
7    // test several String class constructors
8    class StringConstructor
9    {
10      // The main entry point for the application.
11      [STAThread]
12      static void Main( string[] args )
13      {
14         string output;
15         string originalString, string1, string2,
16            string3, string4;
17                  // char array - stores a string
18         char[] characterArray =
19            { 'b', 'i', 'r', 't
20
21         // string initialization
22         originalString = "Welcome to C# pro
23         string1 = originalString; // string1 references
24         string2 = new string( characterArray );
   // char array as argument; new string contains array c
25         string3 = new string( characterArray, 6, 3 );
   // 6-starting position from which to copy; 3
26         string4 = new string( 'C', 5 );
   // 'C' char to copy; 5 - how many times to c
27
28         output = "string1 = " +
29            "string2 = " + "\"" +
30            "string3 = " + "\"" +
31            "string4 = " + "\"" + string4 + "\"\n";
32
```

String declarations

Allocate **char** array **characterArray**

Assign literal **string** to **string**

aracters

String2 assign

String constructor takes a character array as argument

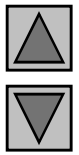
Set **string1** to reference

Starting index and count

Using **string** constructed with a character and an **int** specifying number of times to repeat character in the **string**

ArgumentOutOfRangeException thrown if element outside the

Output the strings



Outline

StringConstructor.cs

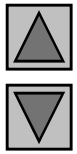
```
33          MessageBox.Show( output, "String Class Constructors",
34             MessageBoxButtons.OK, MessageBoxIcon.Information );
35
36       } // end method Main
37
38    } // end class StringConstructor
```

Shows the output

Program Output

16.4 String Indexer, Length Property and CopyTo Method

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



Outline

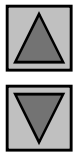
StringMethods.cs

```
1   // Fig. 15.2: StringMethods.cs
2   // Using the indexer, property Length and method CopyTo
3   // of class String.
4
5   using System;
6   using System.Windows.Forms;
7
8   // creates string objects and displays results of using
9   // indexer and methods Length and CopyTo
10  class StringMethods
11  {
12     // The main entry point for the application.
13     [STAThread]
14     static void Main( string[] args )
15     {
16        string string1, output;
17        char[] characterArray;
18
19        string1 = "hello there";
20        characterArray = new char[ 5 ];
21
22        // output string
23        output =
24           "string1: \"" + string1 + "\"";
25
26        // test Length property
27        output += "\nLength of string1: " + string1.Length;
28
29        // loop through character in string1 and display
30        // reversed
31        output += "\nThe string reversed is: ";
32
33        for ( int i = string1.Length - 1; i >= 0; i-- )
34           output += string1[ i ];
35
```

String declarations

String1 to store **string** literal “hello there”

Append to output of **string1** in reverse order



Outline

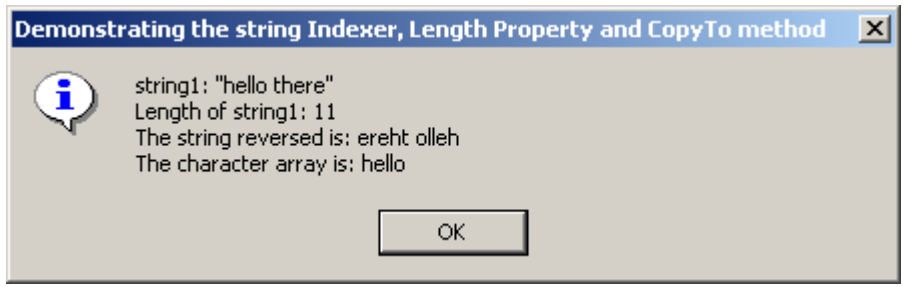
StringMethods.cs

```
36          // copy characters from pos. 0 of string1 into characterArray
37          string1.CopyTo( 0, characterArray, 0, 5 );
38          output += "\nThe character array is: ";
39
40          f
41
42
43                                    ut, "Demonstrating the string " +
44                                    Property and CopyTo method",
45                                    OK, MessageBoxIcon.Information );
46
47       } // end method Main
48
49    } // end class StringMethods
```

Method **Copyto** called by **string1**

...ber of characters into to copy from **string**

Append the **char** array contents to **string output**



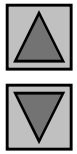
Program Output

16.5 Comparing Strings

- String comparisons
  - Based on char representation by char codes (Unicode in C#)
  - Greater than
    - E.g., Barb > Alice
  - Less than

16.5 Comparing Strings

- Class String provides several ways to compare strings
  - Method Equals and equality operator **==**
    - Test objects for equality
      - Use lexicographical comparison (Unicode values compared)
    - Return a Boolean
      - E.g., if string = "HELLO":
        `string.Equals("hello");` returns false
        `string == "HELLO";` returns true
  - Method CompareTo
    - Test objects for equality *and* inequality (tells if < or >)
      - Use lexicographical comparison (Unicode values compared)
    - Returns:
      - -1 if invoking string < argument
      - 0 if equal
      - -1 invoking string > argument
    - E.g., if string = "hello":
      `string.CompareTo("jello");` returns -1 (bec. "hello" is < "jello")



Outline

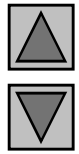
StringCompare.cs

```
1    // Fig. 15.3: StringCompare.cs
2    // Comparing strings.
3
4    using System;
5    using System.Windows.Forms;
6
7    // compare a number of strings
8    class StringCompare
9    {
10      // The main entry point for the application.
11      [STAThread]
12      static void Main( string[] args )
13      {
14         string string1 = "hello";
15         string string2 = "good bye";
16         string string3 = "Happy Birthday";
17         string string4 = "happy birthday";
18         string output;
19
20         // output values of four strings
21         output = "string1 = \"" + string1 + "\"" +
22            "\nstring2 = \"" + string2 + "\"" +
23            "\nstring3 = \"" + string3 + "\"" +
24            "\nstring4 = \""
25
26         // test for equality using Equals method
27         if ( string1.Equals( "hello" ) )
28            output += "string1 equals \"hello\"\n";
29         else
30            outpu                    \"hello\"\n";
31
32         // test
33         if ( string1 == "hello" )
34            output += "string1 equals \"hello\"\n";
```

Instance method **Equals**

Compares values in each string

Equality operator



Outline

StringCompare.cs

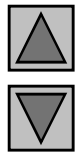
```
35          else
36             output += "string1 does not equal \"hello\"\n";
37
38          // test for equality comparing case
39          if ( String.Equals( string3, string4 ) )
40             output += "string3 equals string4\n";
41          else
42             outp                                  t e
43
44          // test CompareTo
45          output += "\nstring1.CompareTo( string2 ) is " +
46             string1.CompareTo( string2 ) + "\n" +
47             "string2.CompareTo( string1 ) is " +
48             string2.CompareTo( string1 ) + "\n" +
49             "string1.CompareTo( string1 ) is " +
50             string1.CompareTo( string1 ) + "\n" +
51             "string3.CompareTo( string4 ) is " +
52             string3.CompareTo( string4 ) + "\n" +
53             "string4.CompareTo( string3 ) is " +
54             string4.CompareTo( string3 ) + "\n\n";
55
56          MessageBox.Show( output, "Demonstrating string " +
57             "comparisons", MessageBoxButtons.OK,
58             MessageBoxIcon.Information );
59
60       } // end method Main
61
62    } // end class StringCompare
```

Static method **Equals**

Output the appropriate message

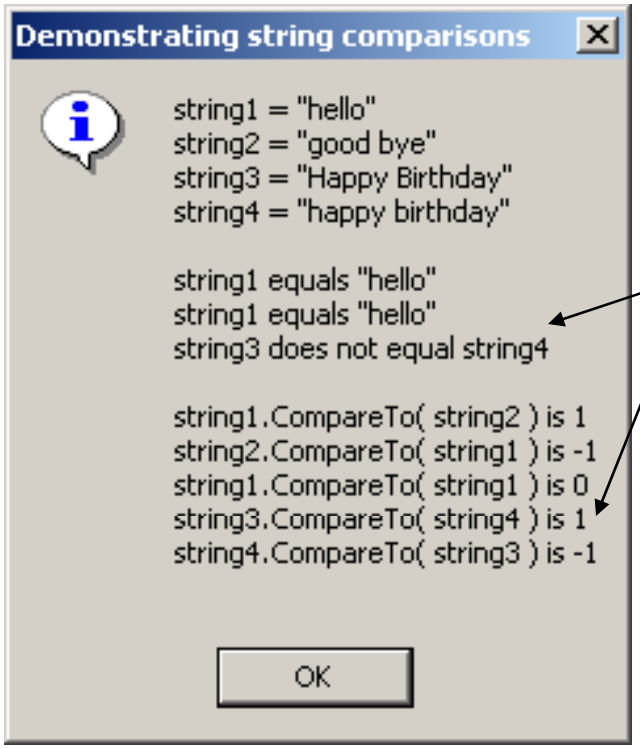
Test equality between **string3** and **string4**

Method **CompareTo** called to compare **string**s



Outline

`StringCompare.cs`
Program Output



String3 contains two uppercase letters, thus it is bigger than string4

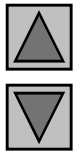
16.5 Comparing Strings

- StartsWith
  - `string.StartsWith( <string> )`
    - `E.g.` if string5 = “hello”:

      `string5.StartsWith( “he” );` returns true

- EndsWith
  - `string.EndsWith( <string> )`
    - `E.g.` if string5 = “hello”:

      `string5.EndsWith( “glo” );` return false



Outline

StringStartEnd.cs

```
1  // Fig. 15.4: StringStartEnd.cs
2  // Demonstrating StartsWith and EndsWith methods.
3
4  using System;
5  using System.Windows.Forms;
6
7  // testing StartsWith and EndsWith
8  class StringStartEnd
9  {
10    // The main entry point for the application.
11    [STAThread]
12    static void Main( string[] args )
13    {
14       string[] strings =
15             { "started", "starting",
16       string output = "";
17
18       //test every string to see if it starts with
19       for ( int i = 0; i < strings.Length; i++ )
20
21          if ( strings[ i ].StartsWith( "st" ) )
22             output += "\"" + strings[ i ] + "\"" +
23                " starts with \"st\"\n";
24
25
26
27       // test every string to see if it ends
28       for ( int i = 0; i < strings.Length; i++ )
29
30          if ( strings[ i ].EndsWith( "ed" ) )
31             output += "\"" + strings[ i ] + "\"" +
32                " ends with \"ed\"\n";
33
```

Contents defined at time of declaration

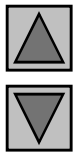
Method **StartsWith** takes a **String** argument

Return **true** if **test** statement matched

If structure determines if **string** at index i starts with “st”

With determines if a **string** instance ends with the **string** text passed to it

If structure determines if **string** at index i starts with “ed”

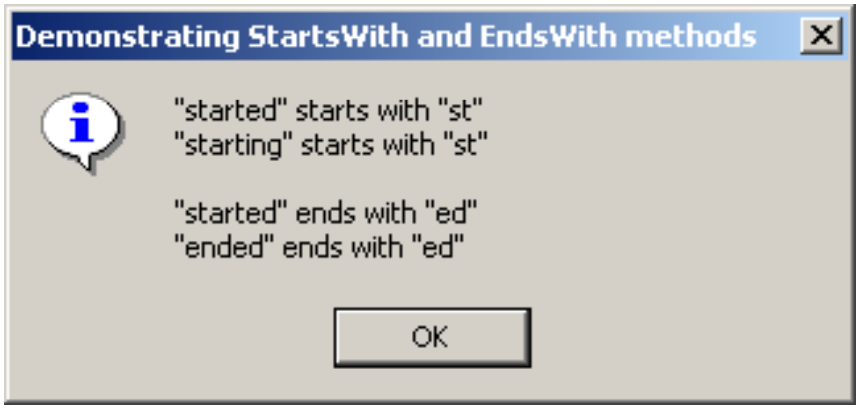


Outline

```
34         MessageBox.Show( output, "Demonstrating StartsWith and " +
35            "EndsWith methods", MessageBoxButtons.OK,
36            MessageBoxIcon.Information );
37
38      } // end method Main
39
40   } // end class StringStartEnd
```

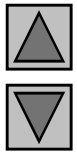
StringStartEnd.cs

Program Output



SKIP--15.6 String Method GetHashCode

- Often need to find needed info quickly
  - E.g., Bob's phone number in a big file
  - Hash table – one of the best ways to do it
- Hash table
  - Of class Object
  - Make information easily/quickly accessible
- Storing object
  - Takes object as input and calculates its hash code
  - Hash code determines where (in which location) to store the object in hash table
- Finding object
  - Takes object as input and calculates its hash code
  - Hash code determines where in hash table to look for the object
- GetHashCode calculates hash code



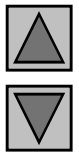
Outline

StringHashCode.cs

```
1   // Fig. 15.5: StringHashCode.cs
2   // Demonstrating method GetHashCode of class String.
3
4   using System;
5   using System.Windows.Forms;
6
7   // testing the GetHashCode method
8   class StringHashCode
9   {
10     // The main entry point for the application.
11     [STAThread]
12     static void Main( string[] args )
13     {
14
15        string string1 = "hello";
16        string string2 = "Hello";
17        string output;
18
19        output = "The hash code for \"" + string1 +
20           "\" is " + string1.GetHashCode() + "\n";
21
22        output += "The hash code for \"" + string2 +
23           "\" is " + string2.GetHashCode() + "\n";
24
25        MessageBox.Show( output, "Demonstrating String " +
26           "method GetHashCode", MessageBoxButtons.OK,
27           MessageBoxIcon.Information );
28
29     } // end method Main
30
31  } // end class StringHashCode
```

Define two **string**s

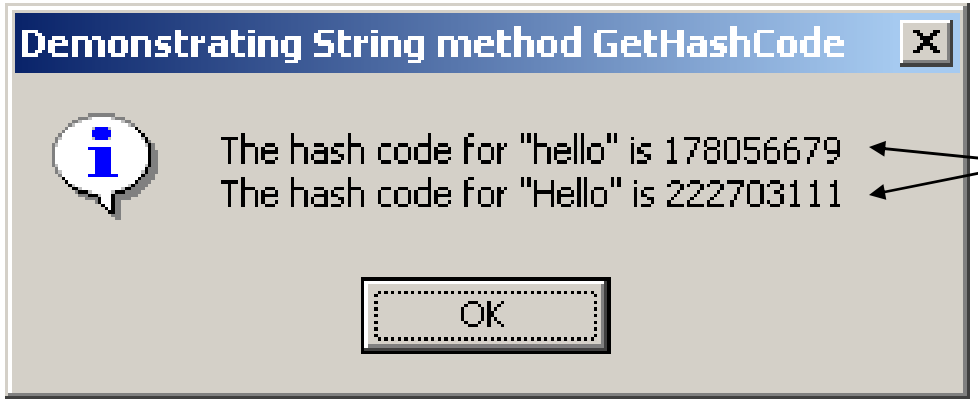
Method **GetHashCode** is called to calculate for **string1** and **string2**



Outline

`StringHashCode.cs`

Program Output

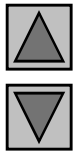


Hash code value for **string**s "**hello**" and "**Hello**"

16.6 Locating Characters and Substrings in Strings

- Search for a character or substring in a string
  - E.g., find “efi” in ‘this is my definition”
  - E.g., search for a word in a DOC report

- **String** methods doing this:
  - `IndexOf` – 1st ocurrence of character/substring
  - `LastIndexOf IndexOfAny` - last ocurrence of character/substring
  - `IndexOfAny` - 1st ocurrence of character/substring
  - `LastIndexOfAny`



Outline

StringIndexMethods.cs

```
1  // Fig. 15.6: StringIndexMethods.cs
2  // Using String searching methods.
3
4  using System;
5  using System.Windows.Forms;
6
7  // testing indexing capabilities of strings
8  class StringIndexMethods
9  {
10    // The main entry point for the application.
11    [STAThread]
12    static void Main( string[] args )
13    {
14       string letters = "abcdefghijklmabcdefghijklm";
15       string output = "";
16       char[] searchLetters = { 'c', 'a', '$' };
17
18       // test IndexOf to locate a character in a string
19       output += "'c' is located at index " +
20          letters.IndexOf( 'c' );
21
22       output += "\n'a' is located at index " +
23          letters.IndexOf( 'a', 1 ); // begin search
24                                     // remember that first cha
25       output += "\n'$' is located at index " +
26          letters.IndexOf( '$', 3, 5 ); // beg. at 3, sea
27
28       // test LastIndexOf to find a character in a st
29       output += "\n\nLast 'c' is located at " +
30          "index " + letters.LastIndexOf( 'c' );
31
32       output += "\nLast 'a' is located at index " +
33          letters.LastIndexOf( 'a', 25 ); // beg. b
34
```

IndexOf takes two arguments, the character to search for and the init

IndexOf takes three arguments, the character to search for, the earch and the s to search

Takes the character as an argument to search

Takes two argument, the character to search for and highest index to begin backward search

Outline

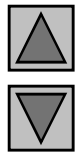
```
35          output += "\nLast '$' is located at index " +
36             letters.LastIndexOf( '$', 15, 5 ); // start
37                               // 15, search 5 positions
38          // test IndexOf to locate a substring in a st
39          output += "\n\n\"def\" is located at" +
40             " index " + letters.IndexOf( "def" );
41
42          output += "\n\"def\" is located at index " +
43             letters.IndexOf( "def", 7 );
44
45          output += "\n\"hello\" is located at index " +
46             letters.IndexOf( "hello", 5, 15 );
47
48          // test LastIndexOf to find a substring in a string
49          output += "\n\nLast \"def\" is located at index " +
50             letters.LastIndexOf( "def" );
51
52          output += "\nLast \"def\" is located at " +
53             letters.LastIndexOf( "def", 25 );
54
55          output += "\nLast \"hello\" is located at index " +
56             letters.LastIndexOf( "hello", 20, 15 );
57
58          // test IndexOfAny to find first occurrence of character
59          // from a set in array
60          output += "\n\nFirst occurrence of 'c', 'a', '$' is " +
61             "located at " + letters.IndexOfAny( searchLetters );
62                   // searchLetters = { 'c', 'a', '$' } - see l.16
63          output += "\nFirst occurrence of 'c, 'a' or '$' is " +
64             "located at " + letters.IndexOfAny( searchLetters, 7 );
65
66          output += "\nFirst occurrence of 'c', 'a' or '$' is " +
67             "located at " + letters.IndexOfAny( searchLetters, 20, 5 );
68
```

Argument takes the character to search for, the starting index to begin searching backward, and portion of string to search

IndexOf and **LastIndexOf** perform similarly as **LastIndexOf** and **IndexOf**

Instead of sending character arguments, these two methods search substring argument

returns the index of first occurrence of any characters specified in the character array argument



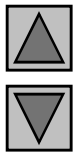
Outline

StringIndexMethods.cs

```
69          // test LastIndexOfAny to find last occurrence of character
70          // in array
71          output += "\n\nLast occurrence of 'c', 'a' or '$' is " +
72             "located at " + letters.LastIndexOfAny( searchLetters );
73
74          output += "\nLast occurrence of 'c', 'a' or '$' is " +
75             "located at " + letters.LastIndexOfAny( searchLetters, 1 );
76                                    // beg. backw search @ char 1 ('b')
77          output += "\nLast occurrence of 'c', 'a' or '$' is " +
78             "located at " + letters.LastIndexOfAny(
79             searchLetters, 25, 5 );
80
81          MessageBox.Show( output,
82             "Demonstrating class index methods",
83             MessageBoxButtons.OK, MessageBoxIcon.Informati
84
85       } // end method Main
86
87    } // end class StringIndexMethods
```

LastIndexOfAny takes an array of

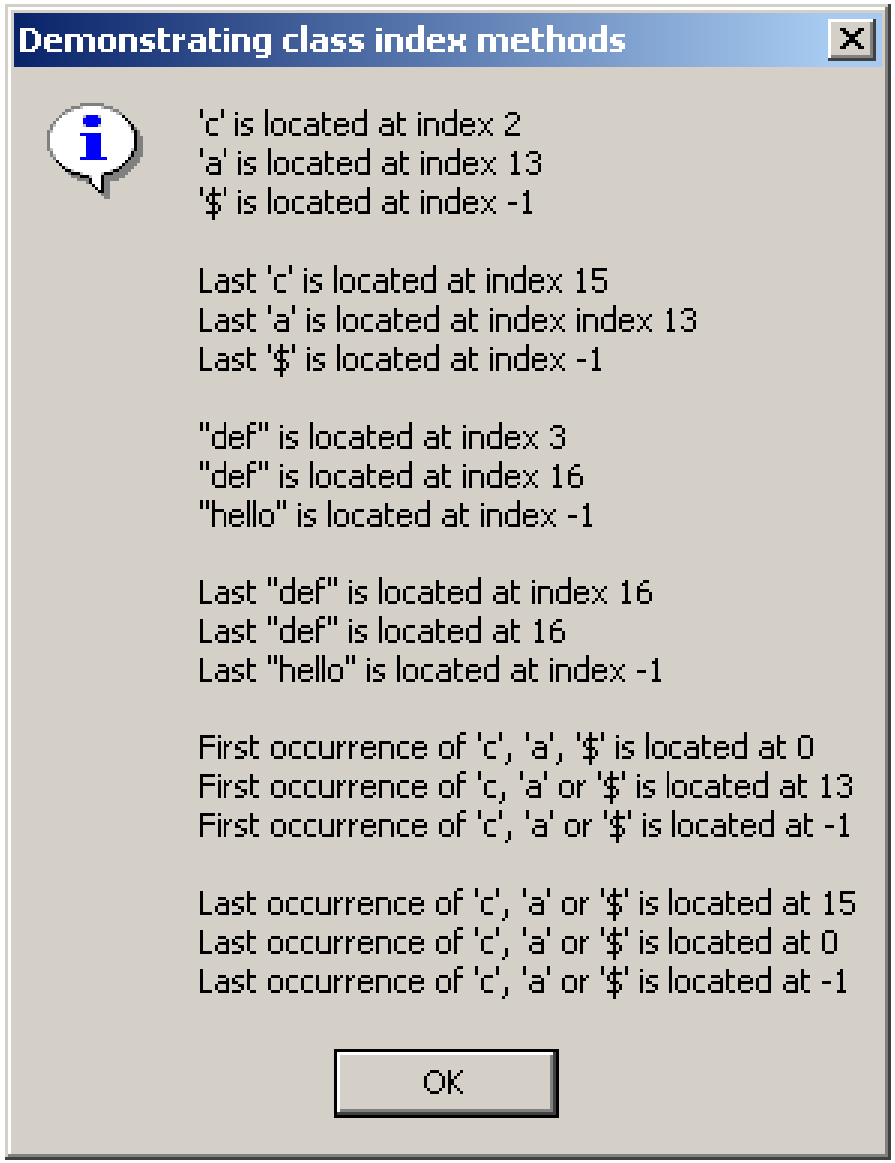
Method **LastIndexOfAny** return the index of first occurrence of any of the character from the argument



Outline

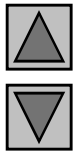
StringIndexMethods.cs

Program Output



16.7 Extracting Substrings from Strings

- Substring methods
  - Create a new string by copying a part of an existing string
  - Methods return new string



Outline

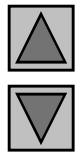
SubString.cs

```
1   // Fig. 15.7: SubString.cs
2   // Demonstrating the String Substring method.
3
4   using System;
5   using System.Windows.Forms;
6
7   // creating substrings
8   class SubString
9   {
10     // The main entry point for the application.
11     [STAThread]
12     static void Main( string[] args )
13     {
14        string letters = "abcdefghijklmabcdefghijklm";
15        string output = "";
16
17        // invoke Substring method and pass it one pa
18        output += "Substring from index 20 to end is
19           letters.Substring( 20 ) + "\"\n";
20
21        // invoke Substring method a
22        output += "Substring from in
23           letters.Substring( 0, 6 )
24
25        MessageBox.Show( output,
26           "Demonstrating String
27           MessageBoxButtons.OK,
28
29     } // end method Main
30
31  } // end class SubString
```

The returned substring contains a copy of the character from the specified index to the end

If index specified is not inside the bound, then **ArgumentOutOfRangeException** thrown

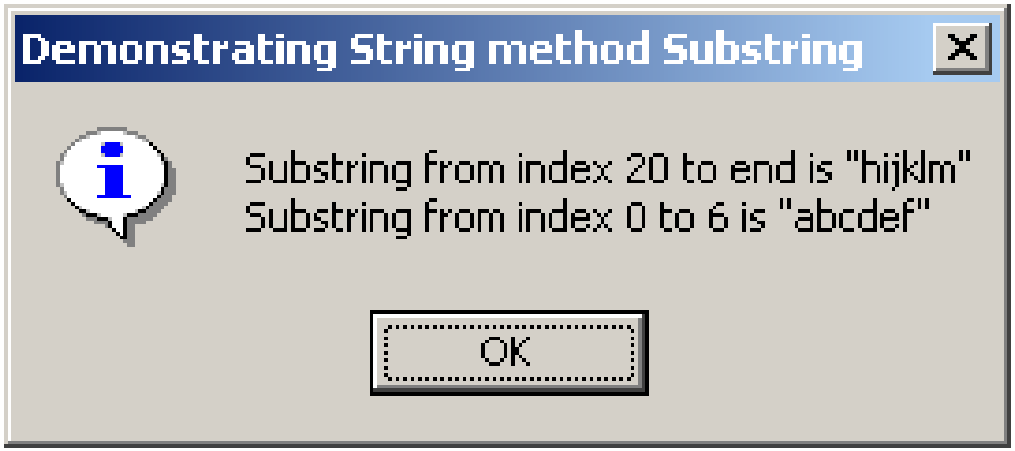
Substring return contains a copy of the specified characters from the original **string**



Outline

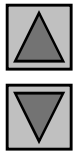
SubString.cs

Program Output



16.8 Concatenating Strings

- Static method Concat
  - Takes two string and return a new string
    - We have been using the + string operator for this



Outline

SubConcatination.cs

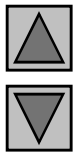
```
1   // Fig. 15.8: SubConcatination.cs
2   // Demonstrating String class Concat method.
3
4   using System;
5   using System.Windows.Forms;
6
7   // concatenates strings using String method Concat
8   class StringConcatenation
9   {
10     // The main entry point for the application.
11     [STAThread]
12     static void Main( string[] args )
13     {
14        string string1 = "Happy ";
15        string string2 = "Birthday";
16        string output;
17
18        output = "string1 = \"" + string1 + "\"\n" +
19           "string2 = \"" + string2 + "\"";
20
21        output +=
22           "\n\nResult of String.Concat( string1, string2 ) = " +
23           String.Concat( string1, string2 );
24
25        output += "\nstring1 after concatenation = " + string1;
26
27        MessageBox.Show( output,
28           "Demonstrating String method Concat",
29           MessageBoxButtons.OK, MessageBoxIcon.Inf
30
31     } // end method Main
32
33  } // end class StringConcatenation
```

Declare two new **string**s

Output the result from the **ConCat** method call

Append **string2** unto the end of **string1**

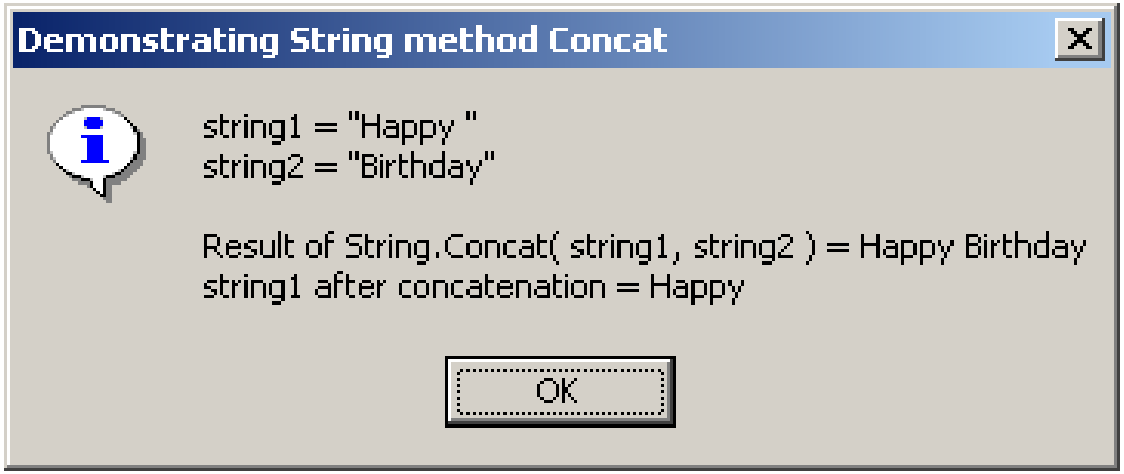
The original **string**, **string1** is not altered



Outline

SubConcatination.cs

Program Output

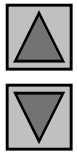


16.9 Miscellaneous String Methods

- Method Replace
  - Original string remains unchanged
  - Original string returned if no match
- Method ToUpper
  - Replace lower case letter
  - Original string remains unchanged
  - Original string returned if no match
- Method ToLower
  - Replace lower case letter
  - Original string remains unchanged
  - Original string returned if no match

16.9 Miscellaneous String Methods

- Method ToString
  - Obtain a string representation of any object
    - We know it very well!
- Method Trim
  - Remove whitespaces
  - Remove all characters that are in the array given as the argument



Outline

StringMiscellaneous2.cs

```
1   // Fig. 15.9: StringMiscellaneous2.cs
2   // Demonstrating String methods Replace, ToLower, ToUpper, Trim
3   // and ToString.
4
5   using System;
6   using System.Windows.Forms;
7
8   // creates strings using methods Replace, ToLower, ToUpper, Trim
9   class StringMethods2
10  {
11     // The main entry point for the application.
12     [STAThread]
13     static void Main( string[] args )
14     {
15        string string1 = "cheers!";
16        string string2 = "GOOD BYE ";     // 1 space after 'BYE'
17        string string3 = "   spaces   ";  // 3 spaces before & after
18        string output;
19
20        output = "string1 = \"" + string1 + "\"\n" +
21           "string2 = \"" + string2 + "\"\n" +
22           "string3 = \"" + string3 + "\"";
23
24        // call method Replace
25        output +=
26           "\n\nReplacing every \"e\" with \"E\" in string1: "
27           string1.Replace( 'e', 'E' ) + "\"";
28
29        // call ToLower and
30        output += "\n\nstring
31           string1.ToUpper() + "\"\nstring2.ToLower()
32           string2.ToLower() + "\"";
33
```

Method **Replace** return new string with correct revision based on the argument

Replace all instances of 'e' with 'E' in **string1**

String to replace with

Method **ToLower** return a new **string** from **string2** by lowercase equivalence

Outline

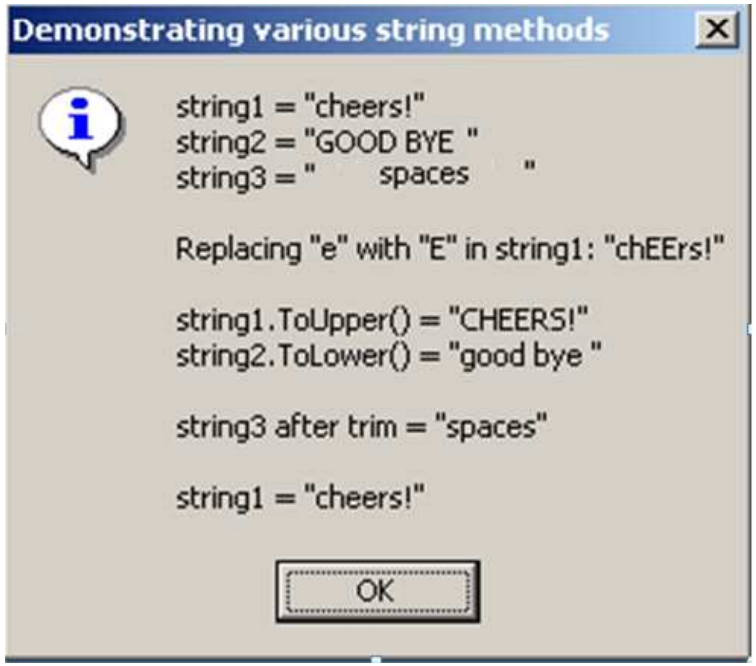
```
34         // call Trim method
35         output += "\n\nstring3 after trim
36            string3.Trim() + "\"";
37
38         // call ToString method
39         output += "\n\nstring1 = \"" + string1.ToString() + "\"";
40
41         MessageBox.Show( output,
42            "Demonstrating various string methods",
43            MessageBoxButtons.OK, MessageBoxIcon.Information );
44
45      } // end method Main
46
47   } // end class StringMethods2
```

Return new **string** omitting leading or trailing whitespace character

Method **ToString** to show **string1** have not been modified

StringMiscellaneous2.cs

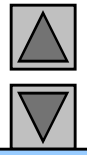
Program Output



16.10 Class StringBuilder

- So far – string processing methods for object class String
  - Strings were immutable
  - The methods did not affect original strings
  - Returned new strings rather than changing old ones

- Now other class: StringBuilder
  - To create and manipulate dynamic string information
    - Mutable strings
  - Capable of dynamic resizing
    - Exceeding capacity of StringBuilder causes the capacity to increase
  - Has 6 constructors (3 shown below)



Outline

```
1   // Fig. 15.10: StringBuilderConstructor.cs
2   // Demonstrating StringBuilder class constructors.
3
4   using System;
5   using System.Windows.Forms;
6   using System.Text;
7
8   // creates three StringBuilder with three constructors
9   class StringBuilderConstructor
10  {
11     // The main entry point for the application.
12     [STAThread]
13     static void Main( string[] args )
14     {
15        StringBuilder buffer1, buffer2, buffer3;
16        string output;
17          // buffer1 & buffer2 – get empty strings –
18        buffer1 = new StringBuilder();// default init. capacity:
19        buffer2 = new StringBuilder( 10 ); // initial capacity:
20        buffer3 = new StringBuilder( "hello" ); //initial capacit
21                        // is 2**3 = 8 (since 8 > length of "hello")
22        output = "buffer1 = \"" + buffer1.ToString() + "\"\n";
23
24        output += "buffer2 = \"" + buffer2.ToString() + "\"\n";
25
26        output += "buffer3 = \"" + buffer3.ToString() + "\"\n";
27
28        MessageBox.Show( output,
29           "Demonstrating StringBuilder class constructors",
30           MessageBoxButtons.OK, MessageBoxIcon.Information );
31
32     } // end method Main
33
34  } // end class StringBuilderConstructor
```

No-argument **StringBuilder** constructor with default initial capacity at 16

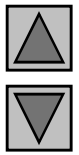
Initial capacity is the smallest power of two greater than the number of characters in the **string** argument

Stringbuilder with no characters and initial size at 10

Stringbuilder with string argument

Method **ToString** to obtain **string** representation of the **StringBuilders'** content

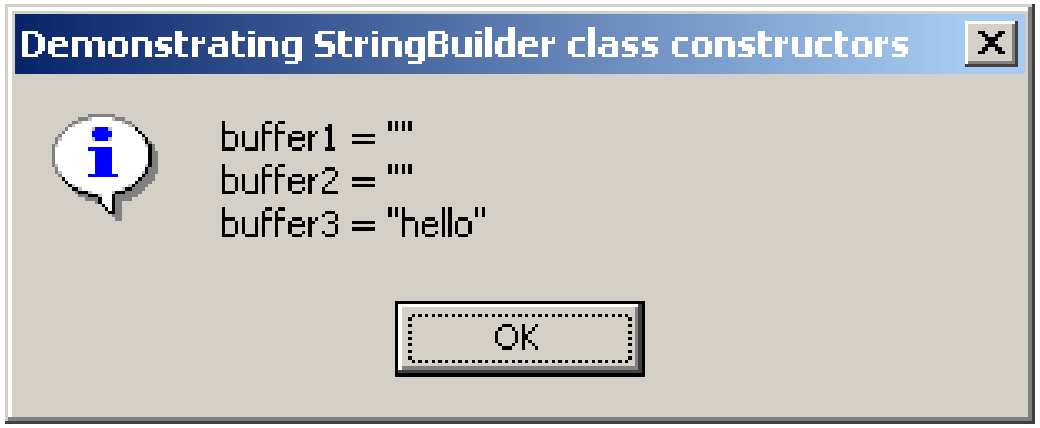
Method returns the **StringBuilders'** underlying string



Outline

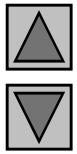
StringBuilderConstructor.cs

Program Output



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

- Method EnsureCapacity
  - Allows programmers to guarantee StringBuilder has a certain capacity
    - Reduces the number of times capacity must be increased
  - EnsureCapacity *approximately* doubles the current capacity of a StringBuilder instance
    - Details: p. 781
    - Examples of capacity provided by instantiation of the StringBuilder class:
      - For explicitly demanded initial capacity, it is as demanded (cf. line 19 in Slide 40)
      - For empty string argument : default initial capacity is 16 (cf. line 18 in Slide 40)
      - For non-empty string argument , default initial capacity is the s\mallest power of 2 larger than string length (cf. line 20 in Slide 40)
- Properties of StringBuilder
  - Length property
    - returns number of characters currently in StringBuilder
  - Capacity property
    - returns capacity
      - i.e., the number of characters that StringBuilder can store without allocating memory



Outline

StringBuilderFeatures.cs

```
1  // Fig. 15.11: StringBuilderFeatures.cs
2  // Demonstrating some features of class StringBuilder.
3
4  using System;
5  using System.Windows.Forms;
6  using System.Text;
7
8  // uses some of class StringBuilder's methods
9  class StringBuilderFeatures
10 {
11    // The main entry point for the application.
12    [STAThread]
13    static void Main( string[] args )
14    {
15       StringBuilder buffer =
16          new StringBuilder( "Hello, how are you?" );
17
18       // use Length and Capacity properties
19       string output = "buffer = " + buffer.ToString() +
20          "\nLength = " + buffer.Length +
21          "\nCapacity = " + buffer.Capacity; // 32 = 2**
22
23       // use EnsureCapacity method
24       buffer.EnsureCapacity( 75 ); // Ask for
25                                    // p. 781
26       output += "\n\nNew capacity = " +
27          buffer.Capacity;
28
29       // truncate StringBuilder by sett
30       buffer.Length = 10;
31
32       output += "\n\nNew length = " +
33          buffer.Length + "\nbuffer = ";
34
```

Declare a **StringBuilder** name

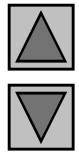
Take **string** argument to initialize its value to the actual string

Append to **output** the

Append to **output** the capacity of **StringBuilder**

Expands the capacity to a minimum of 75 characters

Would append null characters to the end of the **StringBuilder** if Length exceed space needed

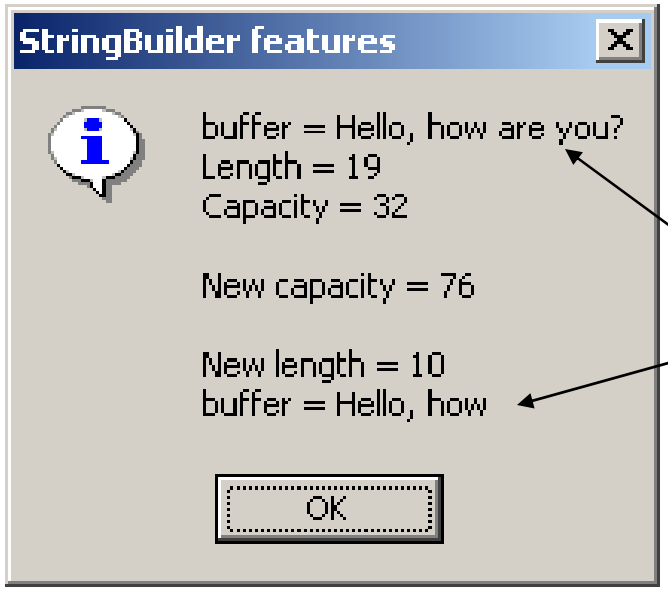


Outline

StringBuilderFeatures.cs

```
35          // use StringBuilder indexer
36          for ( int i = 0; i < buffer.Length; i++ )
37             output += buffer[ i ];
38
39          MessageBox.Show( output, "StringBuilder features",
40             MessageBoxButtons.OK, MessageBoxIcon.Information );
41
42       } // end method Main
43
44    } // end class StringBuilderFeatures
```

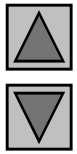
Program Output



New length for the **StringBuilder** is only 10, truncates any preceding characters

16.12 Append and AppendFormat Methods of Class StringBuilder

- Append method
  - Allows various data-type values to be appended to the end of a StringBuilder object instance
  - Converts its argument into string
- AppendFormat method
  - Like Append + converts string to a specifiable format



Outline

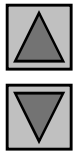
StringBuilderAppend.cs

```
1    // Fig. 15.12: StringBuilderAppend.cs
2    // Demonstrating StringBuilder Append methods.
3
4    using System;
5    using System.Windows.Forms;
6    using System.Text;
7
8    // testing the Append method
9    class StringBuilderAppend
10   {
11      // The main entry point for the application.
12      [STAThread]
13      static void Main( string[] args )
14      {
15         object objectValue = "hello";
16         string stringValue = "good bye";
17         char[] characterArray = { 'a', 'b', 'c', 'd',
18                                   'e', 'f' };
19
20         bool booleanValue = true;
21         char characterValue = 'Z';
22         int integerValue = 7;
23         long longValue = 1000000;
24         float floatValue = 2.5F;
25         double doubleValue = 33.333;
26         StringBuilder buffer = new StringBuilder();
27                 // 10 different overloaded Append methods below
28   // use method Append to append values of various classes to
29         buffer.Append( objectValue );
30         buffer.Append( "  " );
31         buffer.Append( stringValue );
32         buffer.Append( "  " );
33         buffer.Append( characterArray );
34         buffer.Append( "  " );
```

Various different type of objects created to append

StringBuilder buffer created for this example

Examples of overloaded **Append** methods



Outline

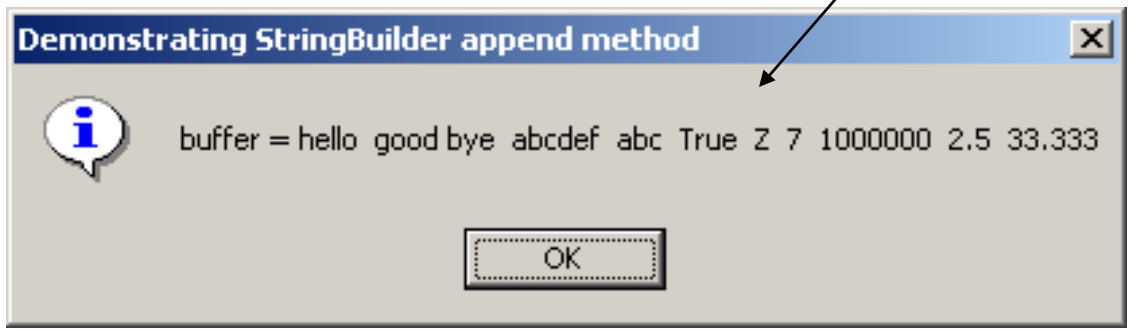
lderApp

```
35          buffer.Append( characterArray, 0, 3 ); // append 3, start at 0
36          buffer.Append( "  " );
37          buffer.Append( booleanValue );
38          buffer.Append( "  " );
39          buffer.Append( characterValue );
40          buffer.Append( "  " );
41          buffer.Append( integerValue );
42          buffer.Append( "  " );
43          buffer.Append( longValue );
44          buffer.Append( "  " );
45          buffer.Append( floatValue );
46          buffer.Append( "  " );
47          buffer.Append( doubleValue );
48
49          MessageBox.Show( "buffer = " + buffer.ToString(),
50             "Demonstrating StringBuilder append method",
51             MessageBoxButtons.OK, MessageBoxIcon.Information );
52
53       } // end method Main
54
55    } // end class StringBuilderAppend
```

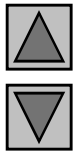
More overloaded **Append** method to attach object onto **buffer**

Append behave similarly to the + operator, used with **string**s

Appends all of the objects created from lines 15-26



Program Output



Outline

StringBuilderAppendFormat.cs

```
1   // Fig. 15.13: StringBuilderAppendFormat.cs
2   // Demonstrating method AppendFormat.
3
4   using System;
5   using System.Windows.Forms;
6   using System.Text;
7
8   // use the AppendFormat method
9   class StringBuilderAppendFormat
10  {
11     // The main entry point for the application.
12     [STAThread]
13     static void Main( string[] args )
14     {
15        StringBuilder buffer = new Strin
16        string string1, string2;
17
18        // formatted string
19        string1 = "This {0} costs: {1:C} \n";
20
21        // string1 argument array
22        object[] objectArray = new ob
23
24        objectArray[ 0 ] = "car";
25        objectArray[ 1 ] = 1234.5
26
27        // append to buffer forma
28        buffer.AppendFormat( stri
29
30        // formatted string
31        string2 = "Number:{0:d3}.\n" +
32           "Number right aligned with spaces:{0, 4}.\n" +
33           "Number left aligned with spaces:{0, -4}.";
34 // 4/-4 – length of output is 4 align to the left/right – se
```

Number of argument t… **string1** with …formation

Format as currency value

Argument referred by **“{0}”** is the object array at index **0**

…rve as …o the **Strin** format **string**

Specify that first argument will be formatted as a thr… digit decimal

Specify that argument should have four characters and be right aligned

Specify that the string should be left aligned

Outline

```
35          // append to buffer formatted string with arg
36          buffer.AppendFormat( string2, 5 ); // forma
37    // to append "5" to buffer 3 times, each time in a d
38                                             strings
39                                        fer.ToStrin                    t",
40             MessageBoxButtons.OK, Messag                              ;
41
42       } // end method Main
43
44    } // end class StringBuilderAppendFormat
```

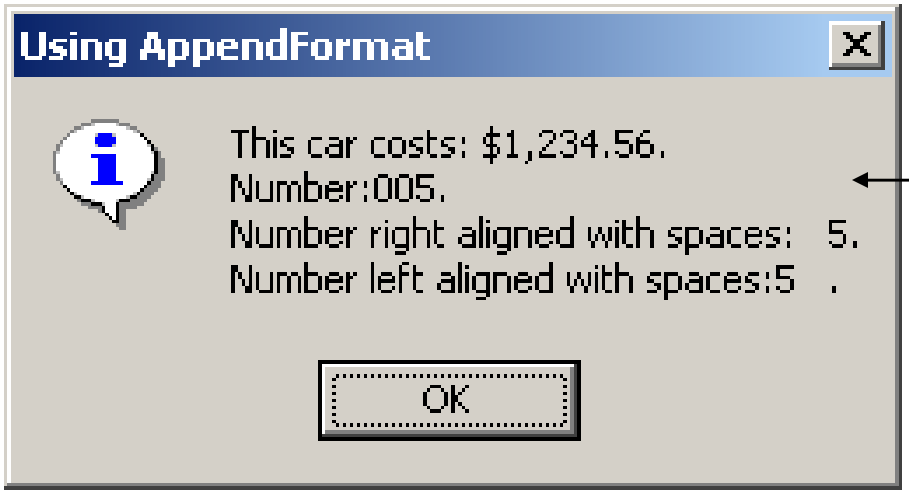
Version of **AppendFormat** that takes two parameters

String containing a format

An object to which the format is applied

StringBuilderAppendFormat.cs

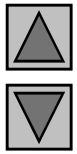
Program Output



Output shows the result of applying **AppendFormat** with the respective arguments

16.13 Insert, Remove and Replace Methods of Class StringBuilder

- Insert method
  - StringBuilder provides 18 overloaded methods
    - Insert at any position
  - Program may throw ArgumentOutOfRangeException
- Remove method
  - Takes two arguments
    - `buffer.Remove(x, y)`
      - starting from position x of 'buffer' remove y chars
  - Program may throw ArgumentOutOfRangeException
- Replace method
  - Substitute specified string



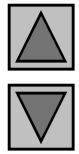
Outline

StringBuilderInsertRemove.cs

```
1    // Fig. 15.14: StringBuilderInsertRemove.cs
2    // Demonstrating methods Insert and Remove of the
3    // StringBuilder class.
4
5    using System;
6    using System.Windows.Forms;
7    using System.Text;
8
9    // test the Insert and Remove methods
10   class StringBuilderInsertRemove
11   {
12      // The main entry point for the application.
13      [STAThread]
14      static void Main( string[] args )
15      {
16         object objectValue = "hello";
17         string stringValue = "good bye";
18         char[] characterArray = { 'a', 'b', 'c',
19                                   'd', 'e', 'f' };
20
21         bool booleanValue = true;
22         char characterValue = 'K';
23         int integerValue = 7;
24         long longValue = 10000000;
25         float floatValue = 2.5F;
26         double doubleValue = 33.333;
27         StringBuilder buffer = new StringBuilder();
28         string output;
29
30         // insert values into buffer at position 0
31         buffer.Insert(0, objectValue);
32         buffer.Insert(0, "  ");
33         buffer.Insert(0, stringValue); // will precede objectValue
34         buffer.Insert(0, "  ");
```

Various different type of objects created to append

Uses the **Insert** method



Outline

StringBuilderInsertRemove.cs

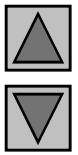
```
35          buffer.Insert(0, characterArray); // will precede stringValue
36          buffer.Insert(0, "  ");
37          buffer.Insert(0, booleanValue); // will precede characterArray
38          buffer.Insert(0, "  ");
39          buffer.Insert(0, characterValue); // ...
40          buffer.Insert(0, "  ");
41          buffer.Insert(0, integerValue);
42          buffer.Insert(0, "  ");
43          buffer.Insert(0, longValue);
44          buffer.Insert(0, "  ");
45          buffer.Insert(0, floatValue);
46          buffer.Insert(0, "  ");
47          buffer.Insert(0, doubleValue); // will precede
48          buffer.Insert(0, "  ");  // 2 spaces precede a
49
50          output = "buffer after inserts: \
51             buffer.ToString() + "\n\n";
52           // b.Remove(x, y) - remove y cha
53          buffer.Remove( 10, 1 ); // delete
54          buffer.Remove( 2, 4 );  // delete .333 in 33.333
55
56          output += "buffer after Removes:\n" +
57             buffer.ToString();
58
59          MessageBox.Show( output, "Demonstrating StringBuilder " +
60             "Insert and Remove methods", MessageBoxButtons.OK,
61             MessageBoxIcon.Information );
62
63       } // end method Main
64
65    } // end class StringBuilderInsertRemove
```

Output display in reverse order was inserted

Specify the extent of the removal

buffer to location of

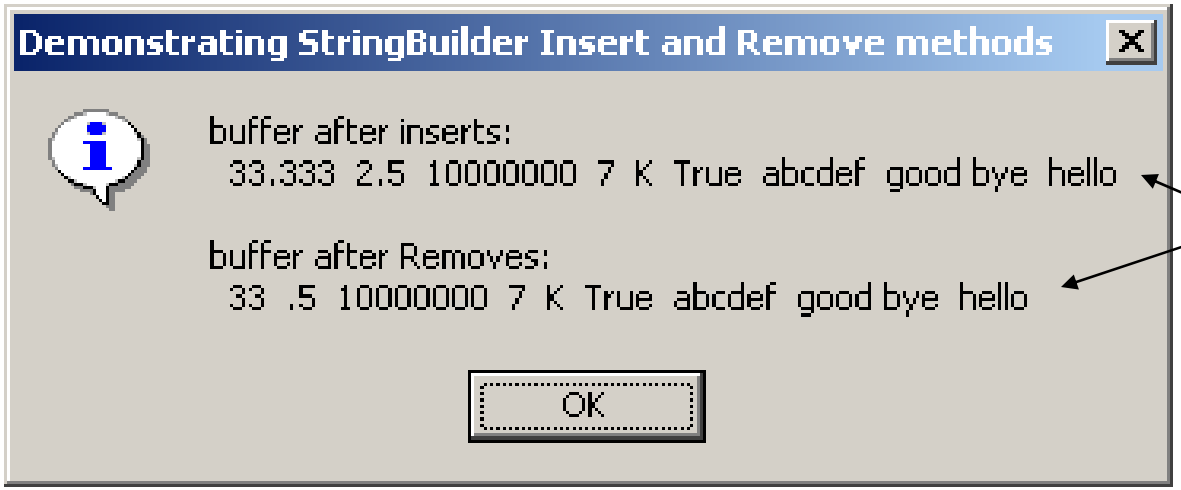
Output the new string from as the result from method **Remove**



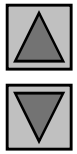
Outline

StringBuilderInsertRemove.cs

Program Output



Changes to the string from the Remove method call



Outline

StringBuilderReplace.cs

```
1   // Fig. 15.15: StringBuilderReplace.cs
2   // Demonstrating method Replace.
3
4   using System;
5   using System.Windows.Forms;
6   using System.Text;
7
8   // testing the Replace method
9   class StringBuilderReplace
10  {
11     // The main entry point for the application.
12     [STAThread]
13     static void Main( string[] args )
14     {
15        StringBuilder builder1 =
16           new StringBuilder( "Happy Birthday Jane" );
17
18        StringBuilder builder2 =
19           new StringBuilder( "good bye greg" );
20
21        string output = "Before replacements:\n" +
22           builder1.ToString() + "\n" + builder2.ToString();
23
24        builder1.Replace( "Jane", "Greg" ); // obvious
25        builder2.Replace( 'g', 'G', 0, 5 ); // replaceme
26             // to 5 positions starting at 0 (i.e., positions 0 - 4)
27        output += "
28           builder1
29
```

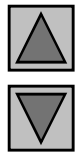
StringBuilder created with specified **string**s

Replace “Jane” with “Greg”

Method **Replace** called

Replaces all instances of the first character with the seco

Index specify begin point and continuation span

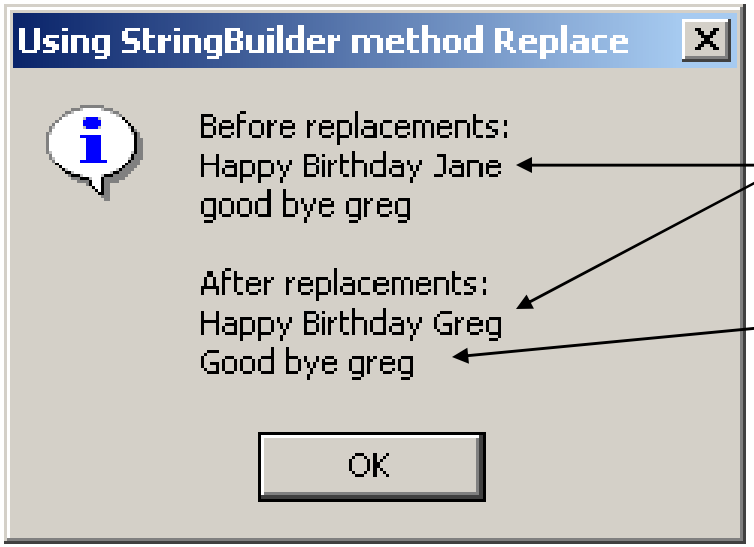


Outline

StringBuilderReplace.cs

```
30         MessageBox.Show( output,
31            "Using StringBuilder method Replace",
32            MessageBoxButtons.OK, MessageBoxIcon.Information );
33
34      } // end method Main
35
36   } // end class StringBuilderReplace
```

Program Output



Result from replacing “Jane” with “Greg”

The continuation span of five was insufficient for “greg” to be altered

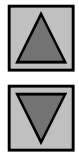
16.14 Char Methods

- Structure
  - A data type
  - Similar to a class but is a value type (not a class)
    - Includes methods and properties
    - Same modifiers: public, private, protected
    - Same member access operator '.'
  - Created with keyword struct

- Many primitive data types are actually aliases for different structures
  - E.g., int is defined by structure System.Int32
    long is defined by structure System.Int64

16.14 Char Methods

- Char is a structure: structure Char
  - A structure for characters
  - Most methods are static
  - Methods (examples):
    - IsLower
    - IsUpper
    - ToUpper
    - ToLower
    - IsPunctuation
    - IsSymbol
    - IsWhiteSpace

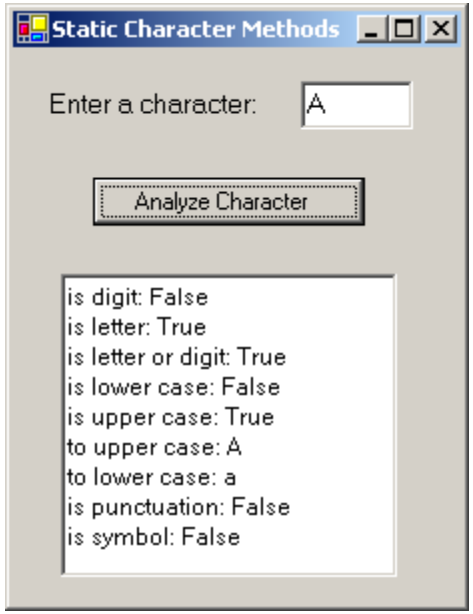


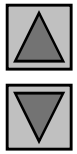
Outline

GUI for Next Program –

Testing Methods For Char Structure

CharMethods.cs





Outline

CharMethods.cs

```
1    // Fig. 15.16: CharMethods.cs
2    // Demonstrates static character testing methods
3    // from Char structure
4
5    using System;
6    using System.Drawing;
7    using System.Collections;
8    using System.ComponentModel;
9    using System.Windows.Forms;
10   using System.Data;
11
12   // Form displays information about specific characters.
13   public class StaticCharMethods : System.Windows.Forms.Form
14   {
15      private System.Windows.Forms.Label enterLabel;
16      private System.Windows.Forms.TextBox inputTextBox;
17      private System.Windows.Forms.Button analyzeButton;
18      private System.Windows.Forms.TextBox outputTextBox;
19
20      private System.ComponentModel.Container components = null;
21
22      // The main entry point for the application.
23      [STAThread]
24      static void Main()
25      {
26         Application.Run( new StaticCharMethods() );
27      }
28
29      // Visual Studio .NET generated code
30
```

TextBox for user to i

TextBox to displays the output of analysis

Outline

CharMethods.cs

```
31    // handle analyzeButton_Click
32    private void analyzeButton_Click(
33       object sender, System.EventArgs e )
34    {                          // convert string to char
35       char character = Convert.ToChar( inputTextBox.Text );
36       BuildOutput( character );
37    }
38
39    // display character information in outputTextBox
40    private void BuildOutput( char inputCharacter )
41    {
42       string output;
43
44       output = "is digit: " +
45          Char.IsDigit( inputCharacter ) + "\r\n";
46
47       output += "is letter: " +
48          Char.IsLetter( inputCharacter ) + "\r\n";
49
50       output += "is letter or digit: " +
51          Char.IsLetterOrDigit( inputCharacter ) +
52
53       output += "is lower case: " +
54          Char.IsLower( inputCharacter ) + "\r\n";
55
56       output += "is upper case: " +
57          Char.IsUpper( inputCharacter ) + "\r\n";
58
59       output += "to upper case: " +
60          Char.ToUpper( inputCharacter ) + "\r\n";
61
62       output += "to lower case: " +
63          Char.ToLower( inputCharacter ) + "\r\n";
64
```

Event handler invoke when user click **Analyze Character**

Method **BuildOutput**

Convert.ToChar data from a **string** to a **char**

Char method **IsDigit**, return **true** if **inputCharacter** is define as a digit

Char method **IsLetter** return **true** if **inputCharacter** is a letter

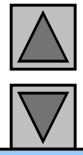
Char method **IsLetterOrDigit** return **true** if **inputCharacter** is a letter or digit

Char method **IsLower** return **true** if **inputCharacter** is a lowercase

Char method **IsUpper** return **true** if **inputCharacter** is an uppercase

Char method **ToUpper** convert **inputCharacter** to uppercase

Method will return original argument if no conversion made equivalent



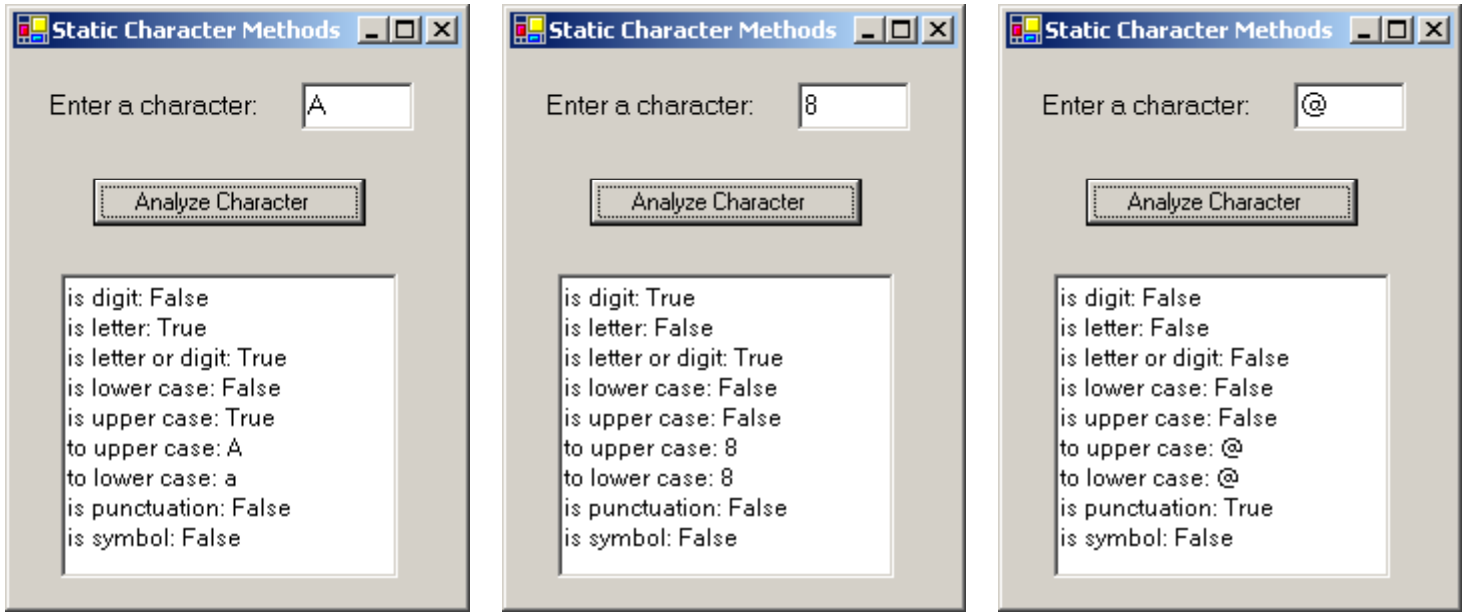
Outline

```
65          output += "is punctuation: " +
66             Char.IsPunctuation( inputCharacter ) + "\r\n";
67
68          output += "is symbol: " + Char.IsSymbol(
69
70          outputTextBox.Text = output;
71
72       } // end method BuildOutput
73
74    } // end class StaticCharMethods
```

Char method **IsPunctuation** return **true** if **inputCharacter** is a punctuation mark

Char method **IsSymbol** return **true** if **inputCharacter** is a symbol

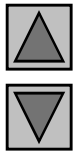
Program Output



SELF-STUDY:

16.15 Card Shuffling and Dealing Simulation

- Class Card
  - Two string instance variable
    - Face and suit
  - Method ToString



Outline

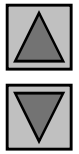
Card.cs

```
1    // Fig. 15.17: Card.cs
2    // Stores suit and face information on each card.
3
4    using System;
5
6    // the representation of a card
7    public class Card
8    {
9       private string face;
10      private string suit;
11
12      public Card( string faceValue,
13         string suitValue )
14      {
15         face = faceValue;
16         suit = suitValue;
17
18      } // end constructor
19
20      public override string ToString()
21      {
22         return face + " of " + suit;
23
24      } // end method ToString
25
26   } // end class Card
```

String instance variables, **face** and **suit**

Constructor receives two **string**s that it use to initialize **face** and **suit**

Method creates a **string** consisting of the **face** and **suit** of the card



Outline

DeckOfCards.cs

```
1   // Fig. 15.18: DeckOfCards.cs
2   // Simulating card drawing and shuffling.
3
4   using System;
5   using System.Drawing;
6   using System.Collections;
7   using System.ComponentModel;
8   using System.Windows.Forms;
9   using System.Data;
10
11  // provides the functionality for the form
12  public class DeckForm : System.Windows.Forms.Form
13  {
14     private System.Windows.Forms.Button dealButton;
15     private System.Windows.Forms.Button shuffleButton;
16
17     private System.Windows.Forms.Label displayLabel;
18     private System.Windows.Forms.Label statusLabel;
19
20     private System.ComponentModel.Container components = null;
21
22     private Card[] deck = new Card[ 52 ];
23     private int currentCard;
24
25     // main entry point for application
26     [STAThread]
27     static void Main()
28     {
29        Application.Run( new deckForm() );
30     }
31
32     // Visual Studio .NET generated code
33
```

Application **Deckform** to create a deck of 52 playing cards using **Card** objects

User can **Deal Card** by pressing its button

Each card dealt is displayed

Outline

DeckOfCards.cs

```
34      // handles form at load time
35      private void DeckForm_Load(
36         object sender, System.EventArgs e )
37      {
38         string[] faces = { "Ace", "Deuce", "Three", "Four",
39                            "Five", "Six", "Seven", "Eight",
40                            "Nine", "Ten", "Jack", "Queen",
41                            "King" };
42
43         string[] suits = { "Hearts", "Diamonds", "Clubs",
44                            "Spades" };
45
46         // no cards have been drawn
47         currentCard = -1;
48
49         // initialize deck
50         for ( int i = 0; i < deck.Length; i++ )
51            deck[ i ] = new Card( faces[ i % 13 ], suits[ i % 4 ] );
52
53      } // end method deckForm_Load
54
55      // handles dealButton Click
56      private void dealButton_Click(
57         object sender, System.EventArgs e )
58      {
59         Card dealt = DealCard();
60
61         // if dealt card is null, then no cards left
62         // player must shuffle cards
63         if ( dealt != null )
64         {
65            displayLabel.Text = dealt.ToString();
66            statusLabel.Text = "Card #: " + currentCard;
67         }
```

Enumeration for possible face types

Enumeration for all possible suits

For structure used to fill the **deck** array with **Card**s

Event handler **dealButton_Click**

If **deck** is not empty, method returns a **Card** object reference, otherwise it returns null

Display card in **displayLabel**

Card number display in the **statusLabel**

```
68          else
69          {
70             displayLabel.Text = "NO MORE CARDS TO DEAL";
71             statusLabel.Text = "Shuffle cards to continue";
72          }
73       }
74
75       // shuffle cards
76       private void Shuffle()
77       {
78          Random randomNumber = new Random()
79          Card temporaryValue;
80
81          currentCard = -1;
82
83          // swap each card with random card
84          for ( int i = 0; i < deck.Length; i++ )
85          {
86             int j = randomNumber.Next( 52 );
87
88             // swap cards
89             temporaryValue = deck[ i ];
90             deck[ i ] = deck[ j ];
91             deck[ j ] = temporaryValue;
92          }
93
94          dealButton.Enabled = true;
95
96       } // end method Shuffle
97
```

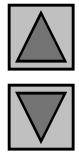
String "NO MORE CARDS TO DEAL" is displayed in **displayLabel** if **DealCard** is null

String "Shuffle cards to ...displayed in ...**f DealCard** is null

Method **Shuffle** makes a total of 52 swaps during single pass of the entire array

The for loops go through all 52 cards (0-51), randomly picks numbers between 0 and 51

Current **Card** object and randomly selected **Card** are swapped in the array



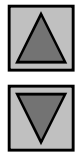
Outline

DeckOfCards.cs

```
98      private Card DealCard()
99      {
100        // if there is a card to deal then deal it
101        // otherwise signal that cards need to be shuffled by
102        // disabling dealButton and returning null
103        if ( currentCard + 1 < deck.Length )
104        {
105           currentCard++;
106           return deck[ currentCard ];
107        }
108        else
109        {
110           dealButton.Enabled = false;
111           return null;
112        }
113
114     } // end method DealCard
115
116     // handles shuffleButton Click
117     private void shuffleButton_Click(
118        object sender, System.EventArgs e )
119     {
120        displayLabel.Text = "SHUFFLING...";
121        Shuffle();
122        displayLabel.Text = "DECK IS SHUFFLED";
123
124     } // end method shuffleButton_Click
125
126  } // end class deckForm
```

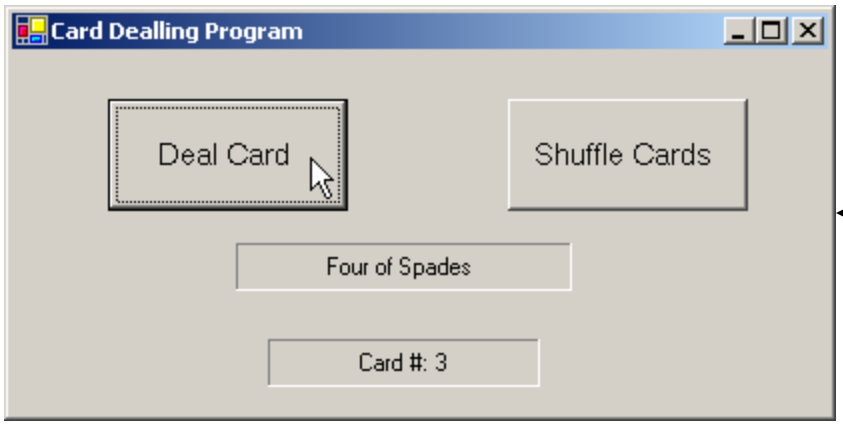
Method **shuffleButton_Click** invokes method **Shuffle** to shuffle the cards

When the call to **Shuffle** returns, **displayLabel** displays "**DECK IS SHUFFLED**"



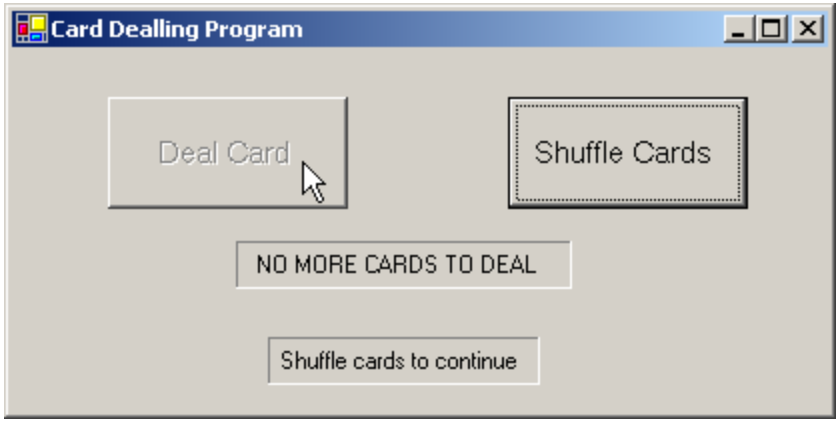
Outline

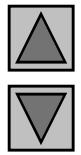
`DeckOfCards.cs`
Program Output



An example of a successful run

Prompt user to shuffle cards because **deck** is empty

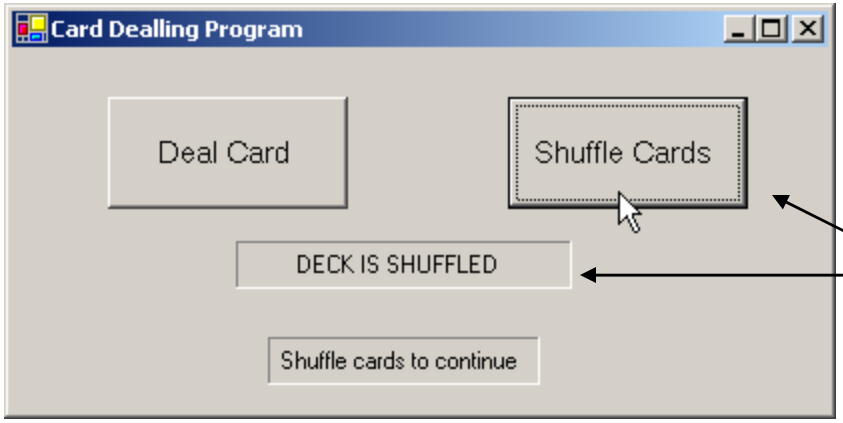




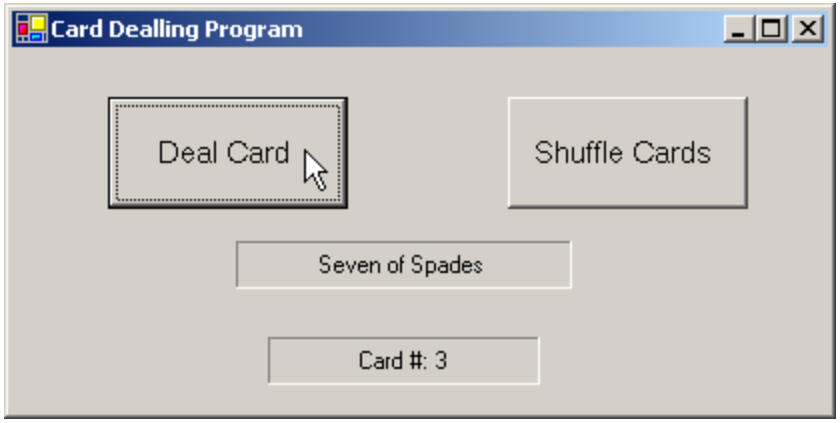
Outline

DeckOfCards.cs

Program Output



When shuffle is completed, message display to user



16.16 Regular Expression and Class Regex

- Regular expression
  - Specially formatted strings used to find patterns in text
  - Facilitates string search
    - E.g., very useful in compilers
      - Used to validate the syntax of a program
- Class Regex
  - An immutable regular expression
  - Method Match
    - Return an object of class Match
  - Method Matches
    - Return a MatchCollection object
- Class Match (different from Regex.Match above!)
  - A regular expression matching operation

16.16 Regular Expressions and Class Regex

Character	Matches	Character	Matches
`\d`	any digit	`\D`	any non-digit
`\w`	any word character	`\W`	any non-word character
`\s`	any whitespace	`\S`	any non-whitespace

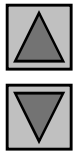
Fig. 15.19 Character classes.

Example regular expression: `Jo.*\sin\s20\d[0-6].`

Matches a substring:

Starting with ‘Jo’
Followed by any number of any characters
Followed by a whitespace
Followed by ‘in’
Followed by a whitespace
Followed by ‘20’
Followed by any digit
Followed by a digit between 0 and 6 (inclusive)
Followed by ‘.’

E.g., matches “John was here in 2005.” or “Joan will repay in 2090.”



Outline

RegexMatches.cs

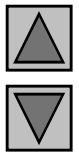
```
1    // Fig. 15.20: RegexMatches.cs
2    // Demonstrating Class Regex.
3
4    using System;
5    using System.Windows.Forms;
6    using System.Text.RegularExpressions;
7
8    // test out regular expressions
9    class RegexMatches
10   {
11      // The main entry point for the application.
12      [STAThread]
13      static void Main( string[] args )
14      {
15         string output = "";
16
17         // create regular expression
18         Regex expression =
19            new Regex( @"J.*\d[0-35-9]-\d\d-\d\d" );
20                     // @ - use string verbatim – cf. p.634
21         string string1 = "Jane's Birthday is 05-12-75\n" +
22            "Dave's Birthday is 11-04-68\n" +
23            "John's Birthday is 04-28-73\n" +
24            "Joe's Birthday is 12-17-77";
25
26         // match regular expression to string and
27         // print out all matches
28         foreach ( Match myMatch in expression.Matches( string1 ) )
29            output += myMatch.ToString() + "\n";
30
```

Represent a two digit number

The dot character matches

“\d” will match any numeric digit

foreach loop iterates through each **Match**



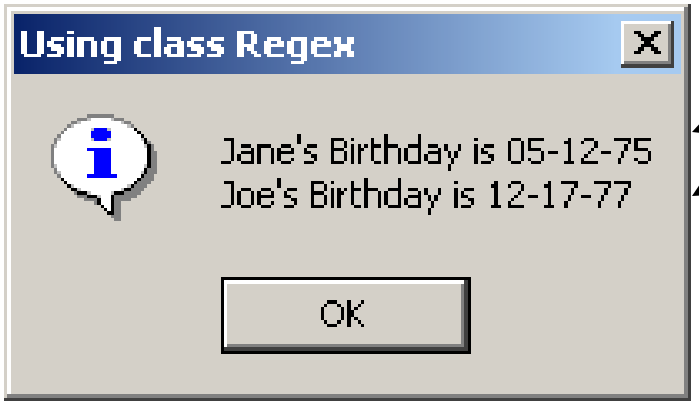
Outline

RegexMatches.cs

```
31          MessageBox.Show( output, "Using class Regex",
32             MessageBoxButtons.OK, MessageBoxIcon.Information );
33
34       } // end method Main
35
36    } // end class RegexMatches
```

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

Program Output



16.16 Regular Expressions and Class Regex

Quantifier	Matches
*	Matches zero or more occurrences of the pattern.
+	Matches one or more occurrences of the pattern.
?	Matches zero or one occurrences of the pattern.
{n}	Matches exactly n occurrences.
{n,}	Matches at least n occurrences.
{n,m}	Matches between n and m (inclusive) occurrences.

Fig. 15.21 Quantifiers used regular expressions.

^ - beginning of the string $ - end of the string a | b – matches a or b

Example regular expression - phone number format:

^[1-9]\d{2}-[1-9]\d{2}-\d{4}$

Matches a whole string (from the beginning ^ to the end $):

Beginning (^) with a digit between 1 and 9 (i.e., 0 excluded)
Followed by 2 digits / Followed by '-' /
Followed by a digit between 1 and 9 /
Followed by 2 digits / Followed by '-' / Ending ($) with 4 digits

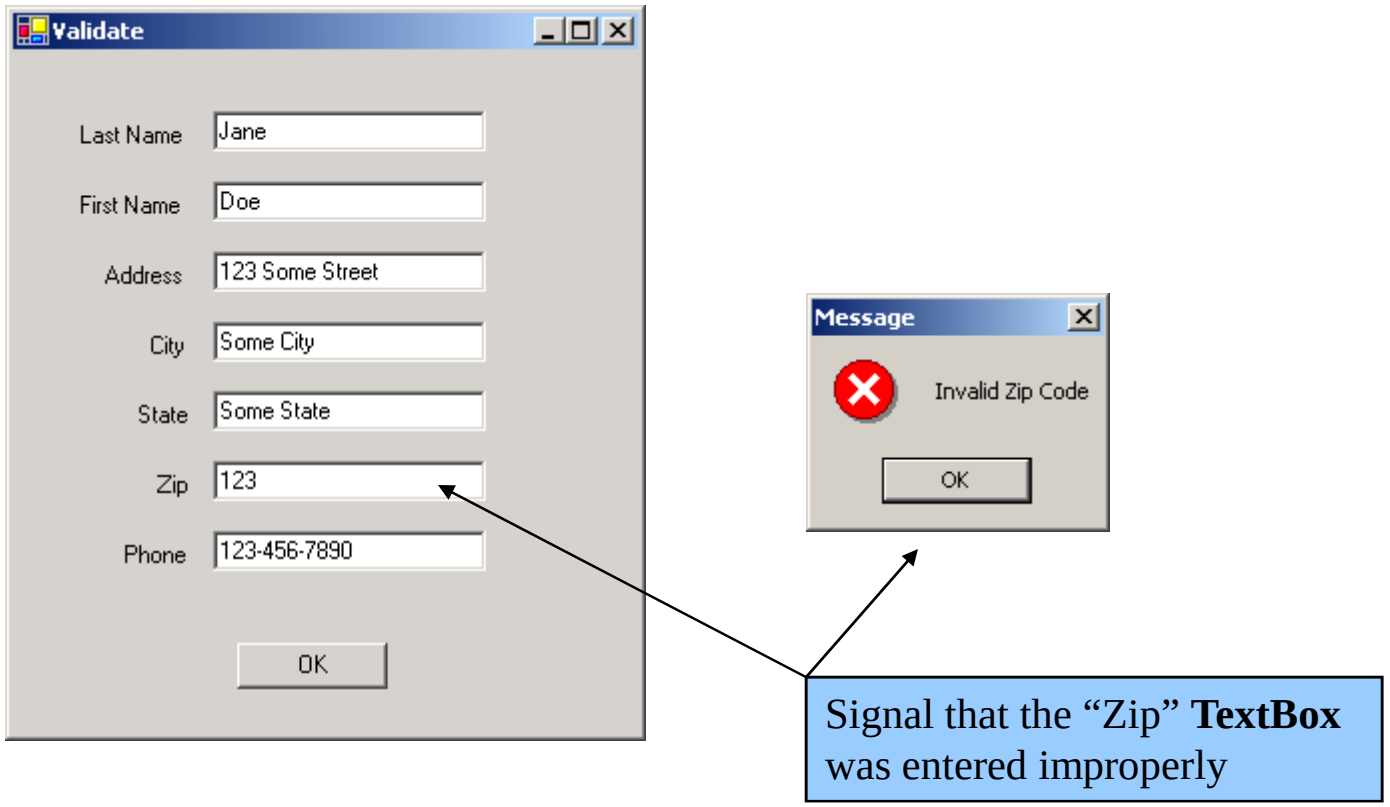
E.g., matches "123-456-7890"

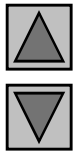
E.g., does not match "123-056-7890" (Why does not match?)

Basic GUI for the Next Program

Outline

`Validate.cs`
Program Output

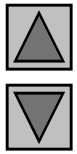




Outline

Validate.cs

```
1    // Fig. 15.22: Validate.cs
2    // Validate user information using regular expressions.
3
4    using System;
5    using System.Drawing;
6    using System.Collections;
7    using System.ComponentModel;
8    using System.Windows.Forms;
9    using System.Data;
10   using System.Text.RegularExpressions;
11
12   // use regular expressions to validate strings
13   public class ValidateForm : System.Windows.Forms.Form
14   {
15      private System.Windows.Forms.Label phoneLabel;
16      private System.Windows.Forms.Label zipLabel;
17      private System.Windows.Forms.Label stateLabel;
18      private System.Windows.Forms.Label cityLabel;
19      private System.Windows.Forms.Label addressLabel;
20      private System.Windows.Forms.Label firstLabel;
21      private System.Windows.Forms.Label lastLabel;
22
23      private System.Windows.Forms.Button okButton;
24
25      private System.Windows.Forms.TextBox phoneTextBox;
26      private System.Windows.Forms.TextBox zipTextBox;
27      private System.Windows.Forms.TextBox stateTextBox;
28      private System.Windows.Forms.TextBox cityTextBox;
29      private System.Windows.Forms.TextBox addressTextBox;
30      private System.Windows.Forms.TextBox firstTextBox;
31      private System.Windows.Forms.TextBox lastTextBox;
32
33      private System.ComponentModel.Container components = null;
34
```



Outline

Validate.cs

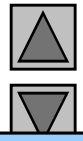
```
35    // The main entry point for the application.
36    [STAThread]
37    static void Main()
38    {
39       Application.Run( new validateForm() );
40    }
41
42    // Visual Studio .NET generated code
43
44    // handles okButton Click event
45    private void okButton_Click(
46       object sender, System.EventArgs e )
47    {
48       // ensures no textboxes are empty
49       if ( lastTextBox.Text == "" || firstTextBox.Text == "" ||
50          addressTextBox.Text == "" || cityTextBox.Text == "" ||
51          stateTextBox.Text == "" || zipTe
52          phoneTextBox.Text == "" )
53       {
54          // display popup box
55          MessageBox.Show( "Please fill in all fields", "Error",
56             MessageBoxButtons.OK, MessageBoxIcon.Error );
57
58          // set focus to lastTextBox
59          lastTextBox.Focus(); // places cursor
60
61          return;
62       }
63
```

Method **okButton_Click** to ensure that all data fields were filled in

Will inform if a certain field was left blank

Method **Focus** places the cursor within the **TextBox**

Program control then exits the event handler



Outline

```
64          // if last name format invalid show message
65          if ( !Regex.Match( lastTextBox.Text,
66             @"^[A-Z][a-zA-Z]*$" ).Success )
67          {                    // above: Success property
68             // last name was incorrect
69             MessageBox.Show( "Invalid Las
70                MessageBoxButtons.OK, Mess
71             lastTextBox.Focus();
72
73             return;
74          }
75
76          // if first name format invalid show message
77          if ( !Regex.Match( firstTextBox.Text,
78             @"^[A-Z][a-zA-Z]*$" ).Success )
79          {
80             // first name was incorrect
81             MessageBox.Show( "Invalid First Name", "Message",
82                MessageBoxButtons.OK, MessageBoxIcon.Error );
83             firstTextBox.Focus();
84
85             return;
86          }
87
88          // if address format invalid show message
89          if ( !Regex.Match( addressTextBox.Text,
90             @"^[0-9]+\s+([a-zA-Z]+|[a-zA-Z]+\s[a-zA-Z]+)$" ).Success )
91          {
92             // address was i
93             MessageBox.Show(                    Message",
94                MessageBoxBut                    on.Error );
95             addressTextBox.Focus();
96
97             return;
98          }
```

If the **Success** property of

.cs

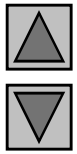
"*" quantifier signifies that the second range of characters

Last Name and **First Name** fields both accept strings of any length that begin with uppercase letter

validate First Name

Address field accepts anything in the format of: "10 Broadway" or "10 Main Street"

"\s" matches a single whitespace character



Outline

Validate.cs

```
99
100         // if city format invalid show message
101         if ( !Regex.Match( cityTextBox.Text,
102            @"^([a-zA-Z]+|[a-zA-Z]+\s[a-zA-Z]+)$" ).Success )
103         {
104            // city was incorrect
105            MessageBox.Show( "Invalid City", "Message",
106               MessageBoxButtons.OK, MessageBoxIcon.Error );
107            cityTextBox.Focus();
108
109            return;
110         }
111
112         // if state format invalid show message
113         if ( !Regex.Match( stateTextBox.Text,
114            @"^([a-zA-Z]+|[a-zA-Z]+\s[a-zA-Z]+)$" ).Success )
115         {
116            // state was incorrect
117            MessageBox.Show( "Invalid State", "Message",
118               MessageBoxButtons.OK, MessageBoxIcon.Error );
119            stateTextBox.Focus();
120
121            return;
122         }
123
124         // if zip code format invalid show message
125         if ( !Regex.Match( zipTextBox.Text, @"^\d{5}$" ).Success )
126         {
127            // zip was incorrect
128            MessageBox.Show( "Invalid Zip Code", "Message",
129               MessageBoxButtons.OK, MessageBoxIcon.Error );
130            zipTextBox.Focus();
131
132            return;
133         }
```

City and **State** match any word of at least one character or any two words of at least one character if the words are separated by a single space

Must be 5 digits, exactly

Outline

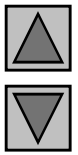
ate.cs

```
134
135       // if phone number format invalid show message
136       if ( !Regex.Match( phoneTextBox.Text,
137          @"^[1-9]\d{2}-[1-9]\d{2}-\d{4}$" ).Success )
138       {
139          //
140          Me
141
142          ph
143
144          return;
145       }
146
147       // information is valid, signal user and exit application
148       this.Hide();
149       MessageBox.Show( "Thank You!", "Information Correct",
150          MessageBoxButtons.OK, MessageBoxIcon.Information );
151
152       Application.Exit();
153
154    } // end method okButton_Click
155
156 } // end class ValidateForm
```

Phone must be the form xxx-yyy-yyy

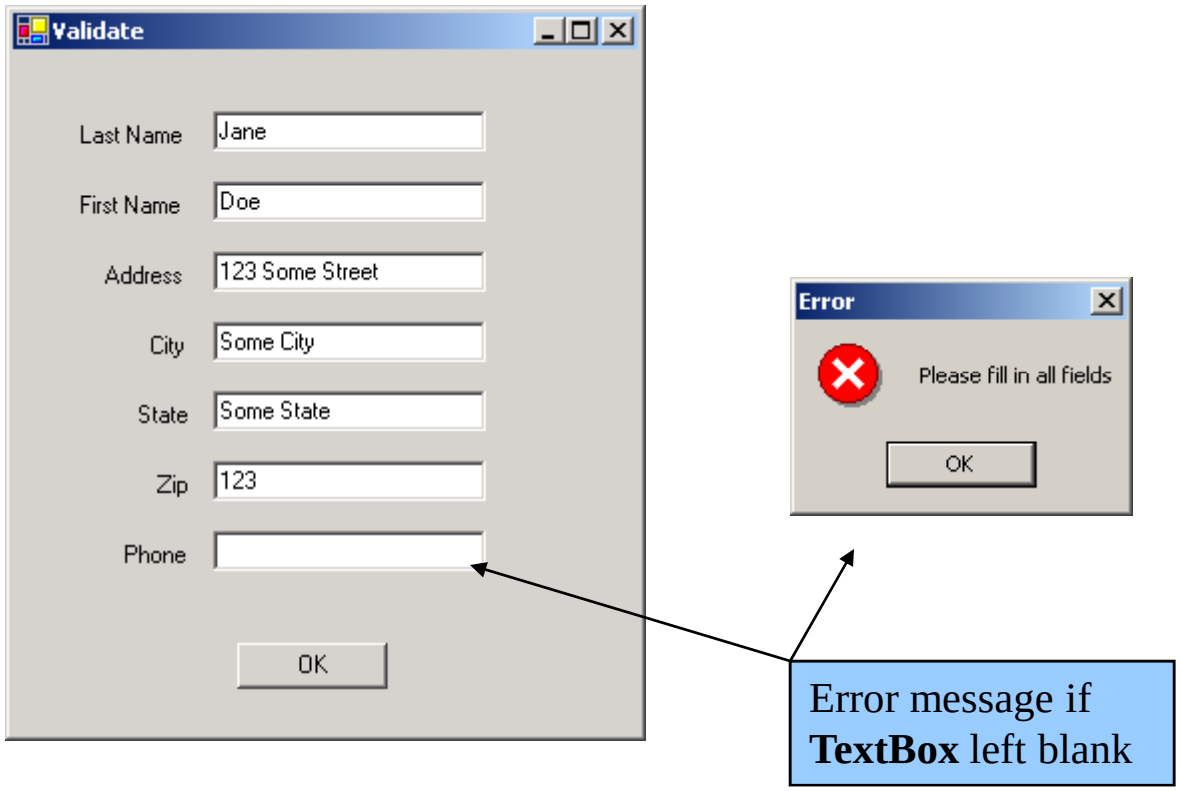
The “^” and “$” expression forces regular expression to evaluate entire string

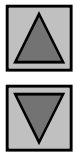
ccess is signaled
ghout the validation the
ram signals it then quits



Outline

`Validate.cs`
Program Output

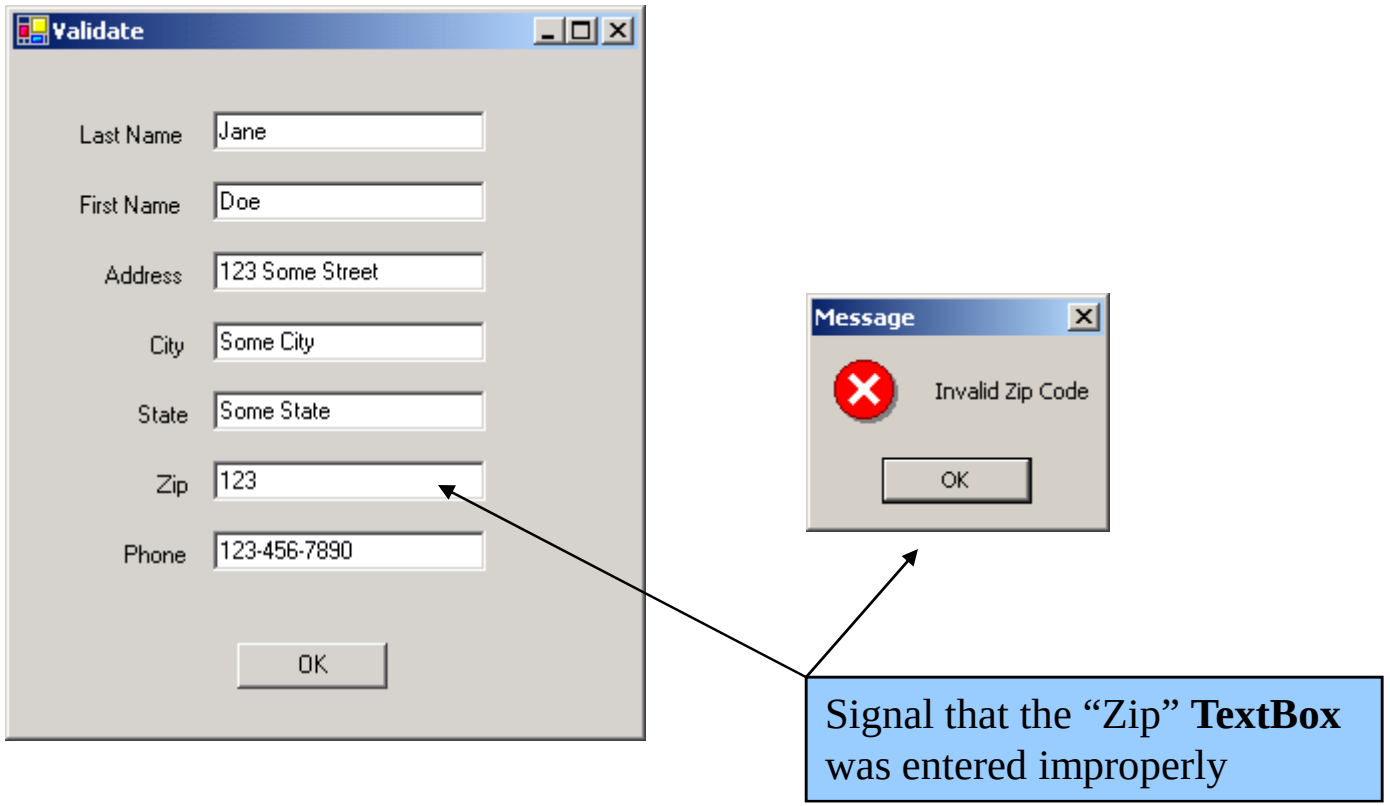


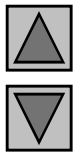


Outline

`Validate.cs`

Program Output

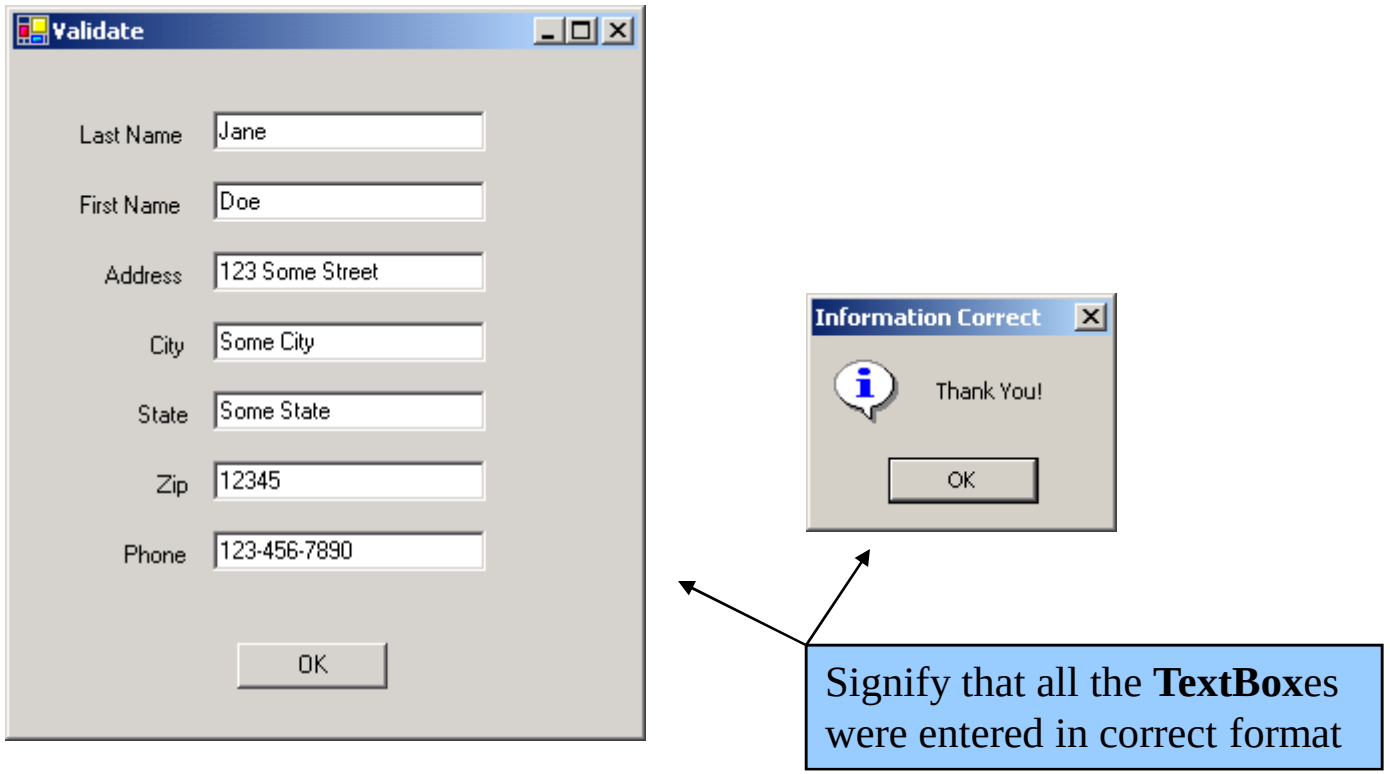


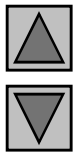


Outline

`Validate.cs`

Program Output





Outline

RegexSubstitution.cs

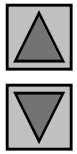
```
1   // Fig. 15.23: RegexSubstitution.cs
2   // Using Regex method Replace.
3
4   using System;
5   using System.Text.RegularExpressions;
6   using System.Windows.Forms;
7
8   // Summary description for RegexSubstitution.
9   public class RegexSubstitution1
10  {
11
12     // The main entry point for the application.
13     static void Main( string[] args )
14     {
15        string testString1 =
16           "This sentence ends in 5 stars *****";
17
18        string testString2 = "1, 2, 3, 4, 5, 6, 7, 8";
19        Regex testRegex1 = new Regex( "stars" );
20        Regex testRegex2 = new Regex( @"\d" );
21        string[] results;
22        string output = "Original String 1\t\t\t"
23
24        testString1 = Re                    );
25
26        output += "\n^ s                    1;
27                  \\  \t - tab
28        testString1 = testRegex1.Replace( testString1, "carets" );
29         // replaces every pattern described by testRegex1 ("stars")
          // that exists in testString1 with "carets"
30        output += "\n\"carets\" substituted for \"stars\"\t" +
31           testString1;
32
33        output += "\nEvery word replaced by \"word\"\t" +
34           Regex.Replace( testString1, @"\w+", "word" );
35
```

Instantiates **testRegex2** with argument **@"\d"**

The replacement **string**

Instance method **Replace** uses regular expression passed to constructor

Regular expression "**stars**" in **testString1** is replaced with "**carets**"



Outline

```
36          output += "\n\nOriginal String 2\t\t\t" + testString2;
37
38          output += "\nFirst 3 digits replaced by \"digit\"\t" +
39             testRegex2.Replace( [illegible] 3 );
40    // (testRegex2 includes \d)
41          output += "\nString sp[illegible]
42            //results (below) is an array of [illegible]
43          results = Regex.Split( testString2, @",\s*" );
44    // split testString2 after each occurence of a comma
45          foreach ( string result[illegible]
46          {
47             output += "\"" + res[illegible]
48          }
49
50          output = output.Substring( 0, output.Length - 2 ) + "]";
51       // Length-2 to avoid printing ", " after "8"
52          MessageBox.Show( output,
53             "Substitution using regular expressions" );
54
55       } // end method Main
56
57    } // end class RegexSubstitution
```

The regular expression

Instance method **Replace** [illegible] kes three argument

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

Replaces first three instances of a [illegible] in **testString2** with "**digit**"

[illegible] ken in any location [illegible] regular expression