CHAPTER FIVE

STRINGS, CHARACTERS, REGULAR EXPRESSIONS, STRUCTURES AND ENUMERATIONS

5.1 Introduction

This chapter introduces the .NET Framework Class Library's string- and character-processing capabilities and demonstrates how to use regular expressions to search for patterns in text. The techniques it presents can be employed in text editors, word processors, page layout software, computerized typesetting systems and other kinds of text-processing software.

We discuss regular expressions. We present classes Regex and Match from the System.Text.RegularExpressions namespace as well as the symbols that are used to form regular expressions. Finally, we introduce to structures and enumerations and using them in C# Language.

5.2 Fundamentals of Characters and Strings

Characters are the fundamental building blocks of C# source code. Every program is composed of characters that, when grouped together meaningfully, create a sequence that the compiler interprets as instructions describing how to accomplish a task. In addition to normal characters, a program also can contain character constants. A character constant is a character that's represented as an integer value, called a character code. For example, the integer value 122 corresponds to the character constant 'z'. The integer value 10 corresponds to the newline character '\n'. Character constants are established according to the Unicode character set, an international character set that contains many more symbols and letters than does the ASCII character set.

*A string is a series of characters treated as a unit. These characters can be uppercase letters, lowercase letters, digits and various special characters: +, -, *, /, $ and others. A string is an object of class string in the System namespace. We write string literals, also called string constants, as sequences of characters in double quotation marks, as follows:*

```
"John Q. Doe"
"9999 Main Street"
"Waltham, Massachusetts"
"(201) 555-1212"
```

A declaration can assign a string literal to a string reference. The declaration

```
string color = "blue";
```

initializes string reference color to refer to the string literal object "blue". On occasion, a string will contain multiple backslash characters (this often occurs in the name of a file). To avoid excessive backslash characters, it's possible to exclude escape sequences and interpret all the characters in a string literally, using the @ character. Backslashes within the double quotation marks following the @ character are not considered escape sequences, but rather regular backslash characters. Often this simplifies programming and makes the code easier to read. For example, consider the string "C:\MyFolder\MySubFolder\MyFile.txt" with the following assignment:

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

Using the verbatim string syntax, the assignment can be altered to

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

This approach also has the advantage of allowing string literals to span multiple lines by preserving all newlines, spaces and tabs.

5.3 string Constructors

Class string provides eight constructors for initializing strings in various ways. Figure 5.1 demonstrates three of the constructors.

```
1   // Fig. 5.1: StringConstructor.cs
2   // Demonstrating string class constructors.
3   using System;
4
5   class StringConstructor
6   {
7      public static void Main( string[] args )
8      {
9        // string initialization
10       char[] characterArray =
11          { 'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y' };
12       string originalString = "Welcome to C# programming!";
13       string string1 = originalString;
14       string string2 = new string( characterArray );
15       string string3 = new string( characterArray, 6, 3 );
16       string string4 = new string( 'C', 5 );
17
18         Console.WriteLine( "string1 = " + "\"" + string1 + "\"\n" +
19            "string2 = " + "\"" + string2 + "\"\n" +
20            "string3 = " + "\"" + string3 + "\"\n" +
21            "string4 = " + "\"" + string4 + "\"\n" );
22     } // end Main
23  } // end class StringConstructor
```

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

Fig. 5.1 | `string` constructors.

Lines 10–11 allocate the char array characterArray, which contains nine characters. Lines 12–16 declare the strings originalString, string1, string2, string3 and string4. Line 12 assigns string literal "Welcome to C# programming!" to string reference originalString. Line 13 sets string1 to reference the same string literal.

Line 14 assigns to string2 a new string, using the string constructor with a character array argument. The new string contains a copy of the array's characters. Line 15 assigns to string3 a new string, using the string constructor that takes a char array and two int arguments. The second argument specifies the starting index position (the offset) from which characters in the array are to be copied.

The third argument specifies the number of characters (the count) to be copied from the specified starting position in the array. The new string contains a copy of the specified characters in the array. Line 16 assigns to string4 a new string, using the string constructor that takes as arguments a character and an int specifying the number of times to repeat that character in the string.

5.4 string Indexer, Length Property and CopyTo Method

The application in Fig. 5.2 presents the string indexer, which facilitates the retrieval of any character in the string, and the string property Length, which returns the length of the string. The string method CopyTo copies a specified number of characters from a string into a char array.

```
1     // Fig. 5.2: StringMethods.cs
2     // Using the indexer, property Length and method CopyTo
3     // of class string.
4     using System;
5
6     class StringMethods
7     {
8        public static void Main( string[] args )
9        {
10          string string1 = "hello there";
11          char[] characterArray = new char[ 5 ];
12
13          // output string1
14          Console.WriteLine( "string1: \"" + string1 + "\"" );
15
16          // test Length property
17          Console.WriteLine( "Length of string1: " + string1.Length );
18
19          // loop through characters in string1 and display reversed
20          Console.Write( "The string reversed is: " );
21
22          for ( int i = string1.Length - 1; i >= 0; i-- )
23             Console.Write( string1[ i ] );
24
25          // copy characters from string1 into characterArray
26          string1.CopyTo( 0, characterArray, 0, characterArray.Length );
27          Console.Write( "\nThe character array is: " );
28
29          for ( int i = 0; i < characterArray.Length; i++ )
30             Console.Write( characterArray[ i ] );
31
32          Console.WriteLine( "\n" );
33       } // end Main
34    } // end class StringMethods
```

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

Fig. 5.2 | `string` indexer, `Length` property and `CopyTo` method.

This application determines the length of a string, displays its characters in reverse order and copies a series of characters from the string to a character array. Line 17 uses string property Length to determine the number of characters in string1. Like arrays, strings always know their own size. Lines 22–23 write the characters of string1 in reverse order using the string indexer. The string indexer treats a string as an array of chars and returns each character at a specific position in the string. The indexer receives an integer argument as the position number and returns the character at that position. As with arrays, the first element of a string is considered to be at position 0.

Line 26 uses string method CopyTo to copy the characters of string1 into a character array (characterArray). The first argument given to method CopyTo is the index from which the method begins copying characters in the string. The second argument is the character array into which the characters are copied. The third argument is the index specifying the starting location at which the method begins placing the copied characters into the character array. The last argument is the number of characters that the method will copy from the string. Lines 29–30 output the char array contents one character at a time.

5.5 Comparing strings

The next two examples demonstrate various methods for comparing strings. To understand how one string can be "greater than" or "less than" another, consider the process of alphabetizing a series of last names. The reader would, no doubt, place "Jones" before "Smith", because the first letter of "Jones" comes before the first letter of "Smith" in the alphabet. The alphabet is more than just a set of 26 letters—it's an ordered list of characters in which each letter occurs in a specific position. For example, Z is more than just a letter of the alphabet; it's specifically the twenty-sixth letter of the alphabet..

Comparing Strings with Equals, CompareTo and the Equality Operator (==)

Class string provides several ways to compare strings. The application in Fig. 5.3 demonstrates the use of method Equals, method ***CompareTo*** *and the equality operator (==). The condition in line 21 uses string method Equals to compare string1 and literal string "hello" to determine whether they're equal. Method Equals (inherited from object and overridden in string) tests any two objects for equality (i.e., checks whether the objects have identical contents). The method returns true if the objects are equal and false otherwise. In this case, the condition returns true, because string1 references string literal object "hello". Method Equals uses word sorting rules that depend on your system's currently selected culture. Comparing "hello" with "HELLO" would return false, because the lowercase letters are different from the those of corresponding uppercase letters.*

```
1  // Fig. 5.3: StringCompare.cs
2  // Comparing strings
3  using System;
4
5  class StringCompare
6  {
7     public static void Main( string[] args )
8     {
9           string string1 = "hello";
10          string string2 = "good bye";
11          string string3 = "Happy Birthday";
12          string string4 = "happy birthday";
13
14          // output values of four strings
15          Console.WriteLine( "string1 = \"" + string1 + "\"" +
16             "\nstring2 = \"" + string2 + "\"" +
17             "\nstring3 = \"" + string3 + "\"" +
18             "\nstring4 = \"" + string4 + "\"\n" );
19
20          // test for equality using Equals method
21          if ( string1.Equals( "hello" ) )
22             Console.WriteLine( "string1 equals \"hello\"" );
23          else
24             Console.WriteLine( "string1 does not equal \"hello\"" );
25
26          // test for equality with ==
27          if ( string1 == "hello" )
28             Console.WriteLine( "string1 equals \"hello\"" );
29          else
30             Console.WriteLine( "string1 does not equal \"hello\"" );
31
```

```
32          // test for equality comparing case
33          if ( string.Equals( string3, string4 ) ) // static method
34             Console.WriteLine( "string3 equals string4" );
35          else
36             Console.WriteLine( "string3 does not equal string4" );
37
38          // test CompareTo
39          Console.WriteLine( "\nstring1.CompareTo( string2 ) is " +
40             string1.CompareTo( string2 ) + "\n" +
41             "string2.CompareTo( string1 ) is " +
42             string2.CompareTo( string1 ) + "\n" +
43             "string1.CompareTo( string1 ) is " +
44             string1.CompareTo( string1 ) + "\n" +
45             "string3.CompareTo( string4 )          + "\n" +
46             string3.CompareTo( string4 ) is " +
47             "string4.CompareTo( string3 ) is " +
48             string4.CompareTo( string3 ) + "\n\n" );
49       } // end Main
50    } // end class StringCompare
```

```
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
```

Fig. 5.3 | `string` test to determine equality.

The condition in line 27 uses the overloaded equality operator (==) to compare string string1 with the literal string "hello" for equality. In C#, the equality operator also compares the contents of two strings. Thus, the condition in the if statement evaluates to true, because the values of string1 and "hello" are equal.

Line 33 tests whether string3 and string4 are equal to illustrate that comparisons are indeed case sensitive. Here, static method Equals is used to compare the values of two strings. "Happy Birthday" does not equal "happy birthday", so the condition of the if statement fails, and the message "string3 does not equal string4" is output (line 36).

Lines 40–48 use string method ***CompareTo*** *to compare strings. Method* ***CompareTo*** *returns 0 if the strings are equal, a negative value if the string that invokes* ***CompareTo*** *is less than the string that's passed as an argument and a positive value if the string that invokes* ***CompareTo*** *is greater than the string that's passed as an argument. Notice that* ***CompareTo*** *considers string3 to be greater than string4. The only difference between these two strings is that string3 contains two uppercase letters in positions where string4 contains lowercase letters.*

Determining Whether a String Begins or Ends with a Specified String

Figure 5.4 shows how to test whether a string instance begins or ends with a given string. Method ***StartsWith*** *determines whether a string instance starts with the string text passed to it as an argument. Method* ***EndsWith*** *determines whether a string instance ends with the string text passed to it as an argument. Class stringStartEnd's Main method defines an array of strings (called strings), which contains "started", "starting", "ended" and "ending". The remainder of method Main tests the elements of the array to determine whether they start or end with a particular set of characters.*

Line 13 uses method ***StartsWith****, which takes a string argument. The condition in the if statement determines whether the string at index i of the array starts with the characters "st". If so, the method returns true, and strings[i] is output along with a message.*

```
1     // Fig. 5.4: StringStartEnd.cs
2     // Demonstrating StartsWith and EndsWith methods.
3     using System;
4
5     class StringStartEnd
6     {
7        public static void Main( string[] args )
8        {
9           string[] strings = { "started", "starting", "ended", "ending" };
10
11          // test every string to see if it starts with "st"
12          for ( int i = 0; i < strings.Length; i++ )
13             if ( strings[ i ].StartsWith( "st" ) )
14                Console.WriteLine( "\"" + strings[ i ] + "\"" +
15                   " starts with \"st\"" );
16
17          Console.WriteLine();
18
```

```
19          // test every string to see if it ends with "ed"
20          for ( int i = 0; i < strings.Length; i++ )
21             if ( strings[ i ].EndsWith( "ed" ) )
22                Console.WriteLine( "\"" + strings[ i ] + "\"" +
23                   " ends with \"ed\"" );
24
25          Console.WriteLine();
26       } // end Main
27    } // end class StringStartEnd
```

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

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

Fig. 5.4 | `StartsWith` and `EndsWith` methods.

*Line 21 uses method **EndsWith** to determine whether the string at index i of the array ends with the characters "ed". If so, the method returns true, and strings[i] is displayed along with a message.*

5.6 Locating Characters and Substrings in strings

*In many applications, it's necessary to search for a character or set of characters in a string. For example, a programmer creating a word processor would want to provide capabilities for searching through documents. The application in Fig. 5.5 demonstrates some of the many versions of string methods **IndexOf**, **IndexOfAny**, **LastIndexOf** and **LastIndexOfAny**, which search for a specified character or substring in a string. We perform all searches in this example on the string letters (initialized with "abcdefghi- jklmabcdefghijklm") located in method Main of class StringIndexMethods.*

*Lines 14, 16 and 18 use method **IndexOf** to locate the first occurrence of a character or substring in a string. If it finds a character, **IndexOf** returns the index of the specified character in the string; otherwise, **IndexOf** returns –1. The expression in line 16 uses a version of method **IndexOf** that takes two arguments—the character to search for and the starting index at which the search of the string should begin. The method does not examine any characters that occur prior to the starting index (in this case, 1). The expression in line 18 uses another version of method **IndexOf** that takes three arguments—the character to search for, the index at which to start searching and the number of characters to search.*

```
1   // Fig. 5.5: StringIndexMethods.cs
2   // Using string-searching methods.
3   using System;
4
5   class StringIndexMethods
6   {
7      public static void Main( string[] args )
8      {
9         string letters = "abcdefghijklmabcdefghijklm";
10        char[] searchLetters = { 'c', 'a', '$' };
11
12        // test IndexOf to locate a character in a string
13        Console.WriteLine( "First 'c' is located at index " +
14           letters.IndexOf( 'c' ) );
15        Console.WriteLine( "First 'a' starting at 1 is located at index " +
16           letters.IndexOf( 'a', 1 ) );
17        Console.WriteLine( "First '$' in the 5 positions starting at 3 " +
18           "is located at index " + letters.IndexOf( '$', 3, 5 ) );
19
20        // test LastIndexOf to find a character in a string
21        Console.WriteLine( "\nLast 'c' is located at index " +
22           letters.LastIndexOf( 'c' ) );
23        Console.WriteLine( "Last 'a' up to position 25 is located at " +
24           "index " + letters.LastIndexOf( 'a', 25 ) );
25        Console.WriteLine( "Last '$' in the 5 positions starting at 15 " +
26           "is located at index " + letters.LastIndexOf( '$', 15, 5 ) );
27
28        // test IndexOf to locate a substring in a string
29        Console.WriteLine( "\nFirst \"def\" is located at index " +
30           letters.IndexOf( "def" ) );
31        Console.WriteLine( "First \"def\" starting at 7 is located at " +
32           "index " + letters.IndexOf( "def", 7 ) );
33        Console.WriteLine( "First \"hello\" in the 15 positions " +
34           "starting at 5 is located at index " +
35           letters.IndexOf( "hello", 5, 15 ) );
36
37        // test LastIndexOf to find a substring in a string
38        Console.WriteLine( "\nLast \"def\" is located at index " +
39           letters.LastIndexOf( "def" ) );
40        Console.WriteLine( "Last \"def\" up to position 25 is located " +
41           "at index " + letters.LastIndexOf( "def", 25 ) );
42        Console.WriteLine( "Last \"hello\" in the 15 positions " +
43           "ending at 20 is located at index " +
44           letters.LastIndexOf( "hello", 20, 15 ) );
45
46        // test IndexOfAny to find first occurrence of character in array
47        Console.WriteLine( "\nFirst 'c', 'a' or '$' is " +
48           "located at index " + letters.IndexOfAny( searchLetters ) );
49        Console.WriteLine("First 'c', 'a' or '$' starting at 7 is " +
50           "located at index " + letters.IndexOfAny( searchLetters, 7 ) );
51        Console.WriteLine( "First 'c', 'a' or '$' in the 5 positions" +
52           "starting at 7 is located at index " +
53           letters.IndexOfAny( searchLetters, 7, 5 ) );
54
```

```
55          // test LastIndexOfAny to find last occurrence of character
56          // in array
57          Console.WriteLine( "\nLast 'c', 'a' or '$' is " +
58           "located at index " + letters.LastIndexOfAny( searchLetters ) );
59          Console.WriteLine( "Last 'c', 'a' or '$' up to position 1 is " +
60             "located at index " +
61             letters.LastIndexOfAny( searchLetters, 1 ) );
62          Console.WriteLine( "Last 'c', 'a' or '$' in the 5 positions " +
63             "ending at 25 is located at index " +
64             letters.LastIndexOfAny( searchLetters, 25, 5 ) );
65       } // end Main
66    } // end class StringIndexMethods
```

```
First 'c' is located at index 2
First 'a' starting at 1 is located at index 13
First '$' in the 5 positions starting at 3 is located at index -1

Last 'c' is located at index 15
Last 'a' up to position 25 is located at index 13
Last '$' in the 5 positions starting at 15 is located at index -1

First "def" is located at index 3
First "def" starting at 7 is located at index 16
First "hello" in the 15 positions starting at 5 is located at index -1

Last "def" is located at index 16
Last "def" up to position 25 is located at index 16
Last "hello" in the 15 positions ending at 20 is located at index -1

First 'c', 'a' or '$' is located at index 0
First 'c', 'a' or '$' starting at 7 is located at index 13
First 'c', 'a' or '$' in the 5 positions starting at 7 is located at index -1

Last 'c', 'a' or '$' is located at index 15
Last 'c', 'a' or '$' up to position 1 is located at index 0
Last 'c', 'a' or '$' in the 5 positions ending at 25 is located at index -1
```

Fig. 5.5 | Searching for characters and substrings in `strings`.

Lines 22, 24 and 26 use method ***LastIndexOf*** *to locate the last occurrence of a character in a string. Method* ***LastIndexOf*** *performs the search from the end of the string to the beginning of the string. If it finds the character,* ***LastIndexOf*** *returns the index of the specified character in the string; otherwise,* ***LastIndexOf*** *returns –1. There are three versions of method* ***LastIndexOf****. The expression in line 22 uses the version that takes as an argument the character for which to search. The expression in line 24 uses the version that takes two arguments—the character for which to search and the highest index from which to begin searching backward for the character. The expression in line 26 uses a third version of method* ***LastIndexOf*** *that takes three arguments—the character for which to search, the starting index from which to start searching backward and the number of*

characters (the portion of the string) to search.

Lines 29–44 use versions of ***IndexOf*** *and* ***LastIndexOf*** *that take a string instead of a character as the first argument. These versions of the methods perform identically to those described above except that they search for sequences of characters (or substrings) that are specified by their string arguments. Lines 47–64 use methods* ***IndexOfAny*** *and* ***LastIndexOfAny****, which take an array of characters as the first argument.*

5.7 Extracting Substrings from strings

Class string provides two Substring methods, which create a new string by copying part of an existing string. Each method returns a new string. The application in Fig. 5.6 demonstrates the use of both methods.

```
1     // Fig. 5.6: SubString.cs
2     // Demonstrating the string Substring method.
3     using System;
4
5     class SubString
6     {
7        public static void Main( string[] args )
8        {
9           string letters = "abcdefghijklmabcdefghijklm";
10
11          // invoke Substring method and pass it one parameter
12          Console.WriteLine( "Substring from index 20 to end is \"" +
13             letters.Substring( 20 ) + "\"" );
14
15          // invoke Substring method and pass it two parameters
16          Console.WriteLine( "Substring from index 0 of length 6 is \"" +
17             letters.Substring( 0, 6 ) + "\"" );
18       } // end method Main
19    } // end class SubString
```

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

Fig. 5.6 | Substrings generated from `strings`.

The statement in line 13 uses the Substring method that takes one int argument. The argument specifies the starting index from which the method copies characters in the original string. The substring returned contains a copy of the characters from the starting index to the end of the string. The second version of method Substring (line 17) takes two int arguments.

The first argument specifies the starting index from which the method copies characters from the original string. The second argument specifies the length of the substring to copy. The substring returned contains a copy of the specified characters from the original string.

5.8 Concatenating strings

The + operator is not the only way to perform string concatenation. The static method Concat of class string (Fig. 5.7) concatenates two strings and returns a new string containing the combined characters from both original strings. Line 16 appends the characters from string2 to the end of a copy of string1, using method Concat. The statement in line 16 does not modify the original strings.

```
1  // Fig. 5.7: SubConcatenation.cs
2  // Demonstrating string class Concat method.
3  using System;
4
5  class StringConcatenation
6  {
7     public static void Main( string[] args )
8     {
9        string string1 = "Happy ";
10          string string2 = "Birthday";
11
12          Console.WriteLine( "string1 = \"" + string1 + "\"\n" +
13             "string2 = \"" + string2 + "\"" );
14          Console.WriteLine(
15             "\nResult of string.Concat( string1, string2 ) = " +
16             string.Concat( string1, string2 ) );
17          Console.WriteLine( "string1 after concatenation = " + string1 );
18       } // end Main
19    } // end class StringConcatenation
```

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

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

Fig. 5.7 | `Concat static` method.

5.9 Miscellaneous string Methods

Class string provides several methods that return modified copies of strings. The application in Fig. 5.8 demonstrates the use of these methods, which include string methods Replace, ToLower, ToUpper and Trim.

```
1  // Fig. 5.8: StringMethods2.cs
2  // Demonstrating string methods Replace, ToLower, ToUpper, Trim,
3  // and ToString.
4  using System;
5
6  class StringMethods2
7  {
8     public static void Main( string[] args )
9     {
10          string string1 = "cheers!";
11          string string2 = "GOOD BYE ";
12          string string3 = "   spaces   ";
13
14          Console.WriteLine( "string1 = \"" + string1 + "\"\n" +
15             "string2 = \"" + string2 + "\"\n" +
16             "string3 = \"" + string3 + "\"" );
17
18          // call method Replace
19          Console.WriteLine(
20             "\nReplacing \"e\" with \"E\" in string1: \"" +
21             string1.Replace( 'e', 'E' ) + "\"" );
22
23          // call ToLower and ToUpper
24          Console.WriteLine( "\nstring1.ToUpper() = \"" +
25             string1.ToUpper() + "\"\nstring2.ToLower() = \"" +
26             string2.ToLower() + "\"" );
27
28          // call Trim method
29          Console.WriteLine( "\nstring3 after trim = \"" +
30             string3.Trim() + "\"" );
31
32          Console.WriteLine( "\nstring1 = \"" + string1 + "\"" );
33       } // end Main
34    } // end class StringMethods2
```

```
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!"
```

Fig. 5.8 | `string` methods `Replace`, `ToLower`, `ToUpper` and `Trim`.

Line 21 uses string method Replace to return a new string, replacing every occurrence in string1 of character 'e' with 'E'. Method Replace takes two arguments—a char for which to search and another char with which to replace all matching occurrences of the first argument. The original string remains unchanged. If there are no occurrences of the first argument in the string, the method returns the original string. An over- loaded version of this method allows you to provide two strings as arguments.

The string method ToUpper generates a new string (line 25) that replaces any lowercase letters in string1 with their uppercase equivalents. The method returns a new string containing the converted string; the original string remains unchanged. If there are no characters to convert, the original string is returned. Line 26 uses string method ToLower to return a new string in which any uppercase letters in string2 are replaced by their lowercase equivalents. The original string is unchanged. As with ToUpper, if there are no characters to convert to lowercase, method ToLower returns the original string.

Line 30 uses string method Trim to remove all whitespace characters that appear at the beginning and end of a string. Without otherwise altering the original string, the method returns a new string that contains the string, but omits leading and trailing whitespace characters. This method is particularly useful for retrieving user input (i.e., via a TextBox). Another version of method Trim takes a character array and returns a copy of the string that does not begin or end with any of the characters in the array argument.