C# String Theory

Dr.Haitham A. El-Ghareeb

Information Systems Department
Faculty of Computers and Information Sciences
Mansoura University

helghareeb@gmail.com

October 7, 2012

Special Thanks GoTo

- en.csharp-online.net
- msdn.microsoft.com
- www.codeproject.com
- blogs.msdn.com

Objectives

By the end of this lecture, you shall be able to:

1. Tell the difference between string and String and String Builder
2. Capable of using String as the most famous Collection
3. Aware of the different String Methods
4. Be familiar with the importance of Pattern Matching and Regular Expression

C# Strings


كلية الحاسبات والمعلومات
CiS
Faculty of Computers and Information Sciences


جامعة المنصورة
Mansoura University

C# Strings

- In C#, a **string** is a sequential collection of Unicode characters that represents text

C# Strings

- In C#, a **string** is a sequential collection of Unicode characters that represents text
- a **String** object is a sequential collection of System.Char objects that represents a string.

C# Strings

- In C#, a **string** is a sequential collection of Unicode characters that represents text
- a **String** object is a sequential collection of System.Char objects that represents a string.
- A C# string has several properties which are critical to understanding how to use them.

string is a Reference Type

- A common misconception is that a C# string is a value type.
- In many situations it does act a bit like a value type.
- It is-in fact-a normal reference type to an object of type System.String.

string is immutable

- Once created, the data value in a string object can never be changed: it is immutable (read-only)
- Methods that appear to modify a String object-in fact-return a new String object containing the modification.
- To modify the actual contents of a string-like object, the System.Text.StringBuilder class can be used.

string may contain nulls

- The String methods will handle null characters in string values
- however, many classes-e.g. Windows Forms classes-may consider the string terminated at the first null.

string versus String

- In the .NET framework, string is simply an alias for the Common Type System (CTS) System.String class
- String class represents a sequence of characters.
- string is one of two predefined C# reference types: The other is object.
- Use them interchangeably in your code.

```
String x = string.Copy ("x");
string y = String.Copy ("y");
```

C# String Literals

- Contiguous sequence of zero or more characters enclosed in double quotation marks (")
- representing string data, rather than a reference to string data.
- A string literal is a literal of type string hard-coded in your program.
- There are two methods of representing string literals-literal (quoted) and verbatim (-quoted).
- The literal method requires certain characters such as quotation marks (") and whitespace to be escaped-that is, preceded by the backslash escape character.
- On the other hand, the verbatim method accepts any characters-except quotation marks-including whitespace without escaping.

String Class vs. String Object

- The String class represents textual data, i.e., a series of Unicode characters;
- String object is actually a sequential collection of System.Char objects.
- The value of the char is the content of the collection
- The value is immutable.

String Fields and Properties

- Empty field: String.Empty is a read-only field which represents the empty string.
- Chars property: returns the character at a specified character position in the instance.
- Length property: returns the number of characters in this instance.

String Operators

- String assignment: Strings can be assigned new values using either of these assignment operators: $=$, $+=$
- String index: The [ ] operator can be used to access individual characters in a string like this: char letter = string1[5];
- String concatenation: strings can be concatenated using the plus (+) operator, or String.Concat

String Operators

- String overloads the equality operators: Normally, the == operator calls the Equals method which compares the pointer values found in the reference variables for equality. This tests whether they point to the same object.

```
string string1 = "value";
Object string2 = "value";
if (string1 == string2)        // compares references
{
    Console.WriteLine ("==");  // comparison is true
}
```

String Manipulation

- The String class contains many public methods which can be used to manipulate strings.
- The most frequently useful of them will be discussed here.
- There are two types of string manipulation methods:
  - String instance methods are called via a string object, e.g. string1.Equals(string2).
  - String static methods-or class methods-are called via the String class, e.g. String.ReferenceEquals (string1, string2).

String instance methods

- CompareTo
- CopyTo
- EndsWith
- Equals
- GetEnumerator
- IndexOf
- IndexOfAny
- Insert
- LastIndexOf
- LastIndexOfAny
- PadLeft
- PadRight

String instance methods (cont.)

- Remove
- Replace
- Split
- StartsWith
- Substring
- ToCharArray
- ToLower
- ToUpper
- Trim
- TrimEnd
- TrimStart

String static methods

- Compare
- CompareOrdinal
- Concat
- Copy
- Format
- Join
- ReferenceEquals

StringBuilder Class

- Represents a mutable string of characters.
- This class cannot be inherited.
- Use the StringBuilder class whenever you will be doing a great deal of string manipulation. This will be speedier and save memory in most cases.

StringBuilder Remarks

- This class represents a string-like object whose value is a mutable sequence of characters.
- The value is said to be mutable because it can be modified after creation by appending, removing, replacing, or inserting characters.
- Most of the methods that modify an instance of this class return a reference to that same instance, and you can call a method or property on the reference.
- This can be convenient if you want to write a single statement that chains successive operations.

StringBuilder Remarks (cont.)

- The capacity of a StringBuilder instance is the maximum number of characters the instance can store at any given time.
- The capacity is greater than, or equal to, the length of the string representation of the value of the instance.
- The capacity can be increased or decreased with the Capacity property or EnsureCapacity method, but it cannot be less than the value of the Length property.
- If you don't specify capacity or maximum capacity when you initialize an instance of StringBuilder, implementation-specific default values are used.

StringBuilder Functionality

- The current size of a StringBuilder object is defined by its Length property. You can access the characters in the value of a StringBuilder object by using the Chars property. Index positions start from zero.
- The StringBuilder class includes methods that can reduce the size of the current instance. The Clear method removes all characters and sets the Length property to zero. The Remove method deletes a range of characters.
- The StringBuilder class also includes methods that can expand the current instance. The Append and AppendLine methods add data to the end of the StringBuilder object, and the Insert method inserts data at a specified character position in the current StringBuilder object. The AppendFormat method uses the composite formatting feature to add formatted text to the end of a StringBuilder object.

StringBuilder Class Functionality (cont.)

- The Replace method replaces all occurrences of a character or a string in the entire StringBuilder object or in a particular character range.
- You must convert the StringBuilder object to a String object before you can pass the string represented by the StringBuilder object to a method that has a String parameter or display it in the user interface. You perform this conversion by calling the ToString method.

StringBuilder Performance Considerations

- A String object concatenation operation always creates a new object from the existing string and the new data.
- A StringBuilder object maintains a buffer to accommodate the concatenation of new data.
- New data is appended to the buffer if room is available; otherwise, a new, larger buffer is allocated, data from the original buffer is copied to the new buffer, and the new data is then appended to the new buffer.
- The performance of a concatenation operation for a String or StringBuilder object depends on the frequency of memory allocations.

StringBuilder Performance Considerations

- A String concatenation operation always allocates memory, whereas a StringBuilder concatenation operation allocates memory only if the StringBuilder object buffer is too small to accommodate the new data.
- Use the String class if you are concatenating a fixed number of String objects. In that case, the compiler may even combine individual concatenation operations into a single operation.
- Use a StringBuilder object if you are concatenating an arbitrary number of strings; for example, if you're using a loop to concatenate a random number of strings of user input.

StringBuilder Memory Allocation

- The default capacity for this implementation is 16, and the default maximum capacity is Int32.MaxValue.
- A StringBuilder object can allocate more memory to store characters when the value of an instance is enlarged, and the capacity is adjusted accordingly.
- Append, AppendFormat, EnsureCapacity, Insert, and Replace methods can enlarge the value of an instance.
- The amount of memory allocated is implementation-specific, and an exception (either ArgumentOutOfRangeException or OutOfMemoryException) is thrown if the amount of memory required is greater than the maximum capacity.

StringBuilder Class Constructors

- StringBuilder(): Initializes a new instance of the StringBuilder class.
- StringBuilder(Int32): Initializes a new instance of the StringBuilder class using the specified capacity.
- StringBuilder(String): Initializes a new instance of the StringBuilder class using the specified string.
- StringBuilder(Int32, Int32): Initializes a new instance of the StringBuilder class that starts with a specified capacity and can grow to a specified maximum.
- StringBuilder(String, Int32): Initializes a new instance of the StringBuilder class using the specified string and capacity.
- StringBuilder(String, Int32, Int32, Int32): Initializes a new instance of the StringBuilder class from the specified substring and capacity.

StringBuilder Class Properties

- Capacity: Gets or sets the maximum number of characters that can be contained in the memory allocated by the current instance.
- Chars: Gets or sets the character at the specified character position in this instance.
- Length: Gets or sets the length of the current StringBuilder object.
- MaxCapacity: Gets the maximum capacity of this instance.

StringBuilder Methods

- Append(Boolean): Appends the string representation of a specified Boolean value to this instance.
- Append(Byte): Appends the string representation of a specified 8-bit unsigned integer to this instance.
- Append(Char): Appends the string representation of a specified Unicode character to this instance.
- Append(Char[]): Appends the string representation of the Unicode characters in a specified array to this instance.
- Append(Decimal): Appends the string representation of a specified decimal number to this instance.

StringBuilder Methods (cont.)

- Append(Double): Appends the string representation of a specified double-precision floating-point number to this instance.
- Append(Object): Appends the string representation of a specified object to this instance.
- Append(SByte): Appends the string representation of a specified 8-bit signed integer to this instance.
- Append(Single): Appends the string representation of a specified single-precision floating-point number to this instance.
- Append(String): Appends a copy of the specified string to this instance.

StringBuilder Methods (cont.)

- Append(UInt16): Appends the string representation of a specified 16-bit unsigned integer to this instance.
- Append(UInt32): Appends the string representation of a specified 32-bit unsigned integer to this instance.
- Append(UInt64): Appends the string representation of a specified 64-bit unsigned integer to this instance.
- Append(Char, Int32): Appends a specified number of copies of the string representation of a Unicode character to this instance.
- Append(Char[], Int32, Int32):Appends the string representation of a specified subarray of Unicode characters to this instance.

StringBuilder Methods (cont.)

- Append(String, Int32, Int32): Appends a copy of a specified substring to this instance.
- AppendFormat(String, Object): Appends the string returned by processing a composite format string, which contains zero or more format items, to this instance. Each format item is replaced by the string representation of a single argument.
- AppendFormat(String, Object[]): Appends the string returned by processing a composite format string, which contains zero or more format items, to this instance. Each format item is replaced by the string representation of a corresponding argument in a parameter array.
- AppendFormat(IFormatProvider, String, Object[]): Appends the string returned by processing a composite format string, which contains zero or more format items, to this instance. Each format item is replaced by the string representation of a corresponding argument in a parameter array using a specified format provider.

StringBuilder Methods (cont.)

- AppendFormat(String, Object, Object): Appends the string returned by processing a composite format string, which contains zero or more format items, to this instance. Each format item is replaced by the string representation of either of two arguments.
- AppendFormat(String, Object, Object, Object): Appends the string returned by processing a composite format string, which contains zero or more format items, to this instance. Each format item is replaced by the string representation of either of three arguments.
- AppendLine(): Appends the default line terminator to the end of the current StringBuilder object.
- AppendLine(String): Appends a copy of the specified string followed by the default line terminator to the end of the current StringBuilder object.

StringBuilder Methods (cont.)

- Clear: Removes all characters from the current StringBuilder instance.
- CopyTo: Copies the characters from a specified segment of this instance to a specified segment of a destination Char array.
- EnsureCapacity: Ensures that the capacity of this instance of StringBuilder is at least the specified value.
- Equals(Object): Determines whether the specified object is equal to the current object. (Inherited from Object.)
- Equals(StringBuilder): Returns a value indicating whether this instance is equal to a specified object.
- Finalize: Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)

StringBuilder Methods (cont.)

- GetHashCode: Serves as a hash function for a particular type. (Inherited from Object.)
- GetType: Gets the Type of the current instance. (Inherited from Object.)
- Insert(Int32, Boolean): Inserts the string representation of a Boolean value into this instance at the specified character position.
- Insert(Int32, Byte): Inserts the string representation of a specified 8-bit unsigned integer into this instance at the specified character position.

StringBuilder Methods (cont.)

- Insert(Int32, Char): Inserts the string representation of a specified Unicode character into this instance at the specified character position.
- Insert(Int32, Char[]): Inserts the string representation of a specified array of Unicode characters into this instance at the specified character position.
- Insert(Int32, Decimal): Inserts the string representation of a decimal number into this instance at the specified character position.
- Insert(Int32, Double): Inserts the string representation of a double-precision floating-point number into this instance at the specified character position.

StringBuilder Methods (cont.)

- Insert(Int32, Int16): Inserts the string representation of a specified 16-bit signed integer into this instance at the specified character position.
- Insert(Int32, Int32): Inserts the string representation of a specified 32-bit signed integer into this instance at the specified character position.
- Insert(Int32, Int64): Inserts the string representation of a 64-bit signed integer into this instance at the specified character position.
- Insert(Int32, Object): Inserts the string representation of an object into this instance at the specified character position.

StringBuilder Methods (cont.)

- Insert(Int32, SByte): Inserts the string representation of a specified 8-bit signed integer into this instance at the specified character position.
- Insert(Int32, Single): Inserts the string representation of a single-precision floating point number into this instance at the specified character position.
- Insert(Int32, String): Inserts a string into this instance at the specified character position.
- Insert(Int32, UInt16): Inserts the string representation of a 16-bit unsigned integer into this instance at the specified character position.

StringBuilder Methods (cont.)

- Insert(Int32, UInt32): Inserts the string representation of a 32-bit unsigned integer into this instance at the specified character position.
- Insert(Int32, UInt64): Inserts the string representation of a 64-bit unsigned integer into this instance at the specified character position.
- Insert(Int32, String, Int32): Inserts one or more copies of a specified string into this instance at the specified character position.

StringBuilder Methods (cont.)

- Insert(Int32, Char[], Int32, Int32): Inserts the string representation of a specified subarray of Unicode characters into this instance at the specified character position.
- MemberwiseClone: Creates a shallow copy of the current Object. (Inherited from Object.)
- Remove: Removes the specified range of characters from this instance.
- Replace(Char, Char): Replaces all occurrences of a specified character in this instance with another specified character.
- Replace(String, String): Replaces all occurrences of a specified string in this instance with another specified string.

StringBuilder Methods (cont.)

- Replace(Char, Char, Int32, Int32): Replaces, within a substring of this instance, all occurrences of a specified character with another specified character.
- Replace(String, String, Int32, Int32): Replaces, within a substring of this instance, all occurrences of a specified string with another specified string.
- ToString(): Converts the value of this instance to a String. (Overrides Object.ToString().)
- ToString(Int32, Int32): Converts the value of a substring of this instance to a String.

C# String Optimization

- The C# String class is designed to minimize unnecessary memory allocations.
- In the process, String operations can provide some unexpected results.
- In the first case, two string variablessiteA, siteBare declared. The first variable is assigned to a string literal ("C# Online.NET").
- The second variable is assigned to the first string variable.

Intern Pool

The intern pool is a list of strings which are currently referenced in your C# application. When a new string is created, then the intern table is checked first to see if that exact string literal already exists in the pool. If it does already exist, then both string variables will reference the same string literal at the same memory location in the intern table. Therefore, only a single copy of a unique string literal is ever created.

C# Best Practices

- Use "String" to refer specifically to the String class.
- Use "string" when referring to an object of the String class.
- Avoid using the @ symbol in order to use C# keywords as identifiers. It can obfuscate the code making it difficult to read and is just plain, poor practice.
- Use the StringBuilder class whenever you will be doing a great deal of string manipulation. This will be speedier and save memory in most cases.