NET.C#.07 Strings and Regular Expressions

What can we use ?

There are two types of string: String and StringBuilder

1. **Building strings** — If you’re performing repeated modifications on a string befor displaying it or passing it to some method, the String class can be very inefficient. For this kind of situation, another class, System.Text.StringBuilder is more suitable, since it has been designed exactly for this situation.

2. Formatting expressions — using a couple of useful interfaces, IFormatProvider and IFormattable, and by implementing these interfaces on our own classes, we can actually define our own formatting sequences, so we can display the values of our classes in whatever way we specify.

3. Regular expressions — .NET offers some classes that deal with the situation in which you need to identify or extract substrings that satisfy certain fairly sophisticated criteri. We can use some classes from System.Text.RegularExpressions, which are designed specifically to perform this kind of processing.

System.String

System.String is a class that is specifically designed to store a string, and allow a large number of operations on the string.

Each string object is an immutable sequence of Unicode characters.
IT means that methods that appear to change the string actually return a modified copy; the original string remains intact in memory until it is garbage-collected.

```
1.   string greetingText = “Hello from all the guys at GrSU “;

2.   greetingText += “We do hope you enjoy this lesson ”;
```

1. An object of type System.String is created and initialized to hold the text. The .NET runtime allocates just enough memory to hold this text (32 chars)

2. We create a new string instance, with just enough memory allocated to store the combined text (55 chars). The old String object is now unreferensed.

Declaration of string

```
// Simple declaration
string MyString = "Hello

World";// Strings can include esca
pe characte
r

s, such as \n or \t, which //

begin with a backslash charact
er (\)strin
g
```

Manipulating Strings

Method or field	Purpose
Chars	The string indexer
Compare( )	Overloaded public static method that compares two strings
CpmpareTo()	Compares this string with another
Concat()	creates a new string from one or more strings
Copy()	Creates a new string by copying another
Join()	Concatenates string of a string array
Split()	Returns a substrings delimited by a characters in a string array
ToUpper()	Returns a copy of the string in uppercase
Length	The number of characters in the string
Substring()	Retrivies a substring

The string class provides a host of methods for comparing, searching, and manipulating strings; the most important of whic

Manipulating Strings

```
int result;//c
o
mpare two strings, case sen
sitiv
eresult = s
tring
.
C
ompare(s1, s2);// compar
e
 two strings, ignore caseres
ult =
 string.Compare(s1, s2, true);// insert the word "e
xcellent"st
ring
s10 = s
3
```

Splitting Strings

```
// create some

strings to work withstring s1 = "On
e,Two
,Three Libe
rty A
s
s
ociates, Inc.";// consta
n
ts for the space and comma character
scons
t char Space = ' ';const char Comma = ',';//
array of de
limiters to split the sentence withchar[] delimiter
s = new char[] { Space
, Com
ma };//
```

The result is array of strings

string object is an immutable

```
string returnNumber = "";
for (int i = 0; i < 1000; i++)
   retu
```

```
re
re
re
re
re
re
re
 .
ret
```

efining
nber a
op we
ld one

The problem is that the string type is not designed for this kind of operation. What you want is to create a new string by appending a formatted string each time through the loop. The class you need is StringBuilder.

Do you know when we do like that we are assigning it again and again?

It's really like assigning 999 new strings !!!

Dynamic Strings (class StringBuilder)

The System.Text.StringBuilder class is used for creating and modifying strings.

Unlike String, StringBuilder is mutable.

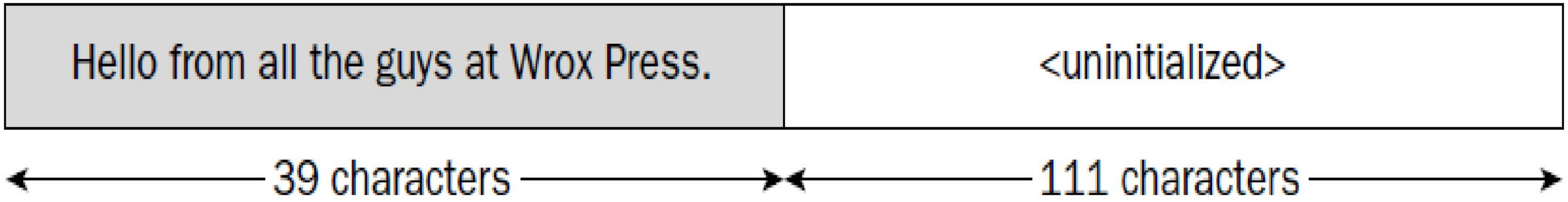
The processing you can do on a StringBuilder is limited to substitutions and appending or removing text from strings. However, it works in a much more efficient way.

When you construct a string, just enough memory gets allocated to hold the string. The StringBuilder, however, normally allocates more memory than needed

StringBuilder

```
StringBuilder greetingBuilder = new StringBuilder(“Hello from all
the guys at Wrox Press. “, 150);
```

Capacity



```
greetingBuilder.Append(“We do hope you enjoy this book as much as
we enjoyed writing it”);
```

Then, on calling the Append() method, the remaining text is placed in the empty space, without the need for more memory allocation.

Normally, we have to use StringBuilder to perform any manipulation of strings, and String to store or display the final result.

StringBuilder members

The StringBuilder class has two main properties:
Length - indicates the length of the string that it actually contains;
Capacity - indicates the maximum length of the string in the memory allocation.

```
Str
ingBuilder sb = new StringBuilder(“Hello”); //Len
gth = 5StringBuilder sb = new StringBuilder(20
); //Capacity = 20//we can set Capa
```

StringBuilder members

The following table lists the main StringBuilder methods.

Method	**Purpose**
Append()	Appends the string to the current string
AppendFormat()	Appends the string that has been worked out from a format specifier
Insert()	Inserts a substring into the current string
Remove()	Remove characters from the current string
Replace()	Replace all occurrences of a character by another character or a substring with another substring in the current string
ToString()	Returns the current string cast to System.String object (overriden from System.Object)

Regular Expressions

Regular expressions are Patterns that can be used to match strings.
Regular expressions are a powerful language for describing and manipulating text.
Regular expression is applied to a string—that is, to a set of characters. Often, that string is an text document.

With regular expressions, you can perform high-level operations on strings. For example, you can:

Identify all repeated words in a string (for example, “The computer books books” to “The computer books”)

Convert all words to title case (for example, “this is a Title” to “This Is A Title”)

Separate the various elements of a URI (for example, given http://www.wrox.com, extract the protocol, computer name, file name, and so on)

Regular Expressions

Task: write a C# console application that takes some text as an input, and determines if the text is an email address.

```
using System.Text;using
S
ystem.T
ext.RegularExpressions;string text = Consol
```

As you can see, a regular expression consists of two types of characters: literals and metacharacters.

A literal is a character you wish to match in the target string.

A metacharacter is a special symbol that acts as a command to the regular expression parser. The parser is the engine responsible for understanding the regular expression.

```
//matches any three char

where t
he first character is 'd‘ or 'a' Text:    a
bc de
f ant cow R
egex:
```

// matches the character

'a' fol
lowed by zero **or more word// c**haracters. Te
xt:
 Anna Jones
 and
a

Metacharacters and their Description

Examples

```
using System;using Sy
s
tem.Text.RegularE
xpressions;// First we
 see the input string.
string input = "/con
t
```

Thanks
for Your Attention

By