String manipulation in C#

Lecture 9

CS 638 Web Programming

Lecture overview

- String formatting
- Stringbuilder and string methods
- Regular expressions

String formatting

- `String.Format(formatstring, arguments)`
  - Also supported by `Console.WriteLine()` and others
- Format string contains groups of the form `{index[,alignment][:codes]}`
  - Index of first argument after format string is 0
  - Alignment specifies number of characters to use (padded with spaces at left if positive, at right if negative)
  - Codes are interpreted based on the value's type
  - Alignment and codes can be omitted
- Relies extensively on objects' ToString() method

Some formatting codes

- For numbers n – use comas to separate thousands,e – scientific notation, x and X – hexadecimal, 0 indicates padding with 0s, # indicates position of digits (see demo4)
- For dates and times: d and D – short/long date, t/T – short/long time, mm – a minutes, MM – month as number, MMM – month as 3 letter code, MMMM – month name
- g stands for the generic default format for all types
- For more details on formatting see http://blog.stevex.net/index.php/string-formatting-in-csharp/

String manipulation methods

- Some methods of the `String` class
  - `Trim()` removes whitespaces from both ends of string
  - `Split(char[] separators)` splits string into an array of substrings separated by the given characters
  - `SubString(int index, int length)` extracts the substring of given length from the given position
  - `IndexOf(string substring, int startIndex)` finds first occurrence of given substring after given index
  - `LastIndexOf(string substring)` finds last index where substring occurs
  - `Replace(string oldValue, string newValue)`

`StringBuilder`

- Strings are immutable objects
- Whenever a new string created, it uses a new memory location
  - This happens whenever strings are concatenated, trimmed, characters replaced, etc.
  - Inefficient if a large string built by many small changes
- The `StringBuilder` class allows more efficient in-place manipulation
  - Appending strings, replacing substrings and characters, removing substrings, etc.

Regular expressions

- A regular expression (regex) is a compact way of representing a certain type of pattern
  - For most patterns, multiple equivalent regexes exist
- Fundamental operation: regex matching – deciding if a given input string can be mapped to the pattern
- Studied by complexity theory – simple to match
- Many applications, among them
  - Used by compilers as a first step of program analysis
  - Various popular Unix commands such as `grep`
  - In web programming mostly for validating user input

Example regular expressions 1

Regex	Strings that match	Strings that don't match
a	"abc", "cba"	"xyz"
^a	"abc"	"cba", "xyz"
a$	"cba"	"abc", "xyz"
^[a-z]$	"a", "x"	"abc", "A", "^[a-z]$"
^[a-z0-9+-]$	"q", "+", "-", "5"	"15", "Q"
^[^a-zA-Z]$	"5", "+"	"p", "R", "abc", "15"
^[a-zA-Z]	"A", "q0", "abc"	"28", " f"
[^a-zA-Z]	"q0", "28", " f"	"A", "abc"
^.$	"m", "3", "%"	"13", "e4", "xyz"
.	"%", "e4", "xyz"	""

Example regular expressions 2

Regex	Strings that match	Strings that don't match
abc	"abcde", "rabco"	"a", "ABC", "bcde"
^abc	"abc", "abcde"	"rabco", "ABC", "bcde"
E[0-9]	"ME3", "E85", "EBE5"	"ACME", "E", "E 8"
E[0-9]$	"ACME3", "EBE5"	"E85", "E3EB", "E 8"
foo|bar	"foo", "bart"	"ooba"
a(b|c)d	"Tabd", "acdc"	"abcd"
^foo|bar	"fool", "Anbar"	"snafoo"
^(foo|bar)	"fool"	"Anbar", "snafoo"

Example regular expressions 3

Regex	Strings that match	Strings that don't match
^a*$	"a", "aaaa", ""	"bart"
^a+$	"a", "aaaa"	"bart", ""
^a?$	"a", ""	"aaaa", "aa"
^a{4}$	"aaaa"	"a", "", "aa"
^a{2,6}$	"aaaa", "aa"	"a", "", "aaaaaaaaaa"
(e|o){2}	"meow", "booooom"	"kenmore"
e|(o{2})	"nest", "boooom"	"port"
(a[0-9]*){2}	"aa", "ba45a"	"abacus", "a3b8a0"

Regular expressions in C#

- Implemented by the class `System.Text.RegularExpressions.Regex`
- Constructor accepts a string describing the regular expression that is "compiled" to a representation used for efficient matching
- Important methods
  - `IsMatch(string input)` checks if input string matches
  - `Replace(string input, string replacement)` replaces all matches of the regular expression in input
  - `Split(string input)` splits input interpreting each match of the regex as a separator
- See demo4 for examples on how to use regexes

C# programming

Lectures 6 - 9

CS 638 Web Programming

Differences between C# and Java

- Application structure
- Inheritance and polymorphism
- Value types and parameter passing
- Syntax changes and extensions
- For more detailed comparison that goes beyond the things we covered in class see http://www.25hoursaday.com/CsharpVsJava.html

Application structure

- Namespaces similar to Java packages, but decoupled from how source code is structured
- Multiple classes can be defined in a single file, name of classes unrelated to file name
- C# partial classes – a class defined in multiple files
- Assemblies similar to jar files
- C#'s keyword `internal` is like Java's `protected` – grants access to others from same assembly
  - In C# `protected` grants access to derived classes

Inheritance and polymorphism

- In C# you must explicitly use the virtual keyword for methods overridden in derived classes (in Java methods are virtual by default)
- By default C# methods are like final methods in Java
  - Derived class can still specify new method, but it does not lead to polymorphic behavior
- Operator overloading supported in C#, not in Java
- C# replaces the `implements` and `extends` keywords with `:`
- C# refers to the base class as `base`, not `super`

Value types

- C# has `structs` which are value types
  - `new` is optional (no memory allocation, calls constructor)
- Generics in C# can also use value types (not just classes as in Java)
- Parameter passing
  - Java passes all parameters by value
    - What does it mean to pass a reference type by value?
  - C# allows passing by reference and output parameters for both value and reference types
- Boolean variables are of type `bool`, not `boolean`

Syntax changes and extensions

- The foreach loop has different syntax
  - In C# `foreach(int i in numbers)`
  - In Java `for(int i:numbers)`
- Changes to `switch` statement,
  - Can use string literals for `case` clauses
  - Fall-through between cases is forbidden
- C# objects can have properties (use of accessors)
- C# has delegates
- C# has keywords `const` and `readonly`

Concepts

- Event-driven programming
  - Extending applications with new event handlers
- Debugging – breakpoints, stepping through program, watches, assertions
- Using language features that make it easier for the compiler to catch mistakes
  - Enums, const, readonly
- Operator overloading can help or harm
- Naming conventions (for interface names)