Core C# Programming: Types, Iteration, and Control Flow

CSIS 3540

Client Server Systems

Class 01

Topics

- Main()
  - Arguments
- Console
- System Types
- Strings
- Data Type Conversions
- Implicit Typing
- Iteration
- Control Flow

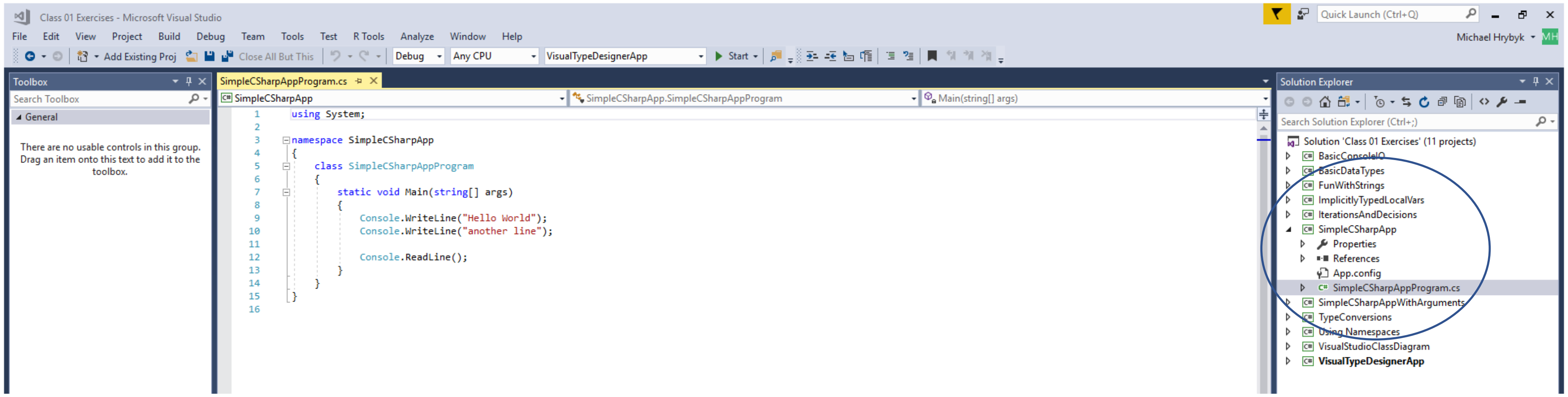
Structure of a C# program

- Create a Console project
- All C# programs REQUIRE a Main() method
    - Main() can have arguments
    - Input from the command line
- RENAME all files and objects properly
- Any added methods should be static

```
using System;

namespace SimpleCSharpApp
{
    class SimpleCSharpAppProgram
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            Console.WriteLine("another line");

            Console.ReadLine();
        }
    }
}
```



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
using System;

namespace SimpleCSharpApp
{
    class SimpleCSharpAppProgram
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            Console.WriteLine("another line");

            Console.ReadLine();
        }
    }
}
```

```
Solution 'Class 01 Exercises' (11 projects)
  BasicConsoleIO
  BasicDataTypes
  FunWithStrings
  ImplicitlyTypedLocalVars
  IterationsAndDecisions
  SimpleCSharpApp
    Properties
    References
    App.config
    SimpleCSharpAppProgram.cs
  SimpleCSharpAppWithArguments
  TypeConversions
  Using Namespaces
  VisualStudioClassDiagram
  VisualTypeDesignerApp
```

The args array parameter for Main()

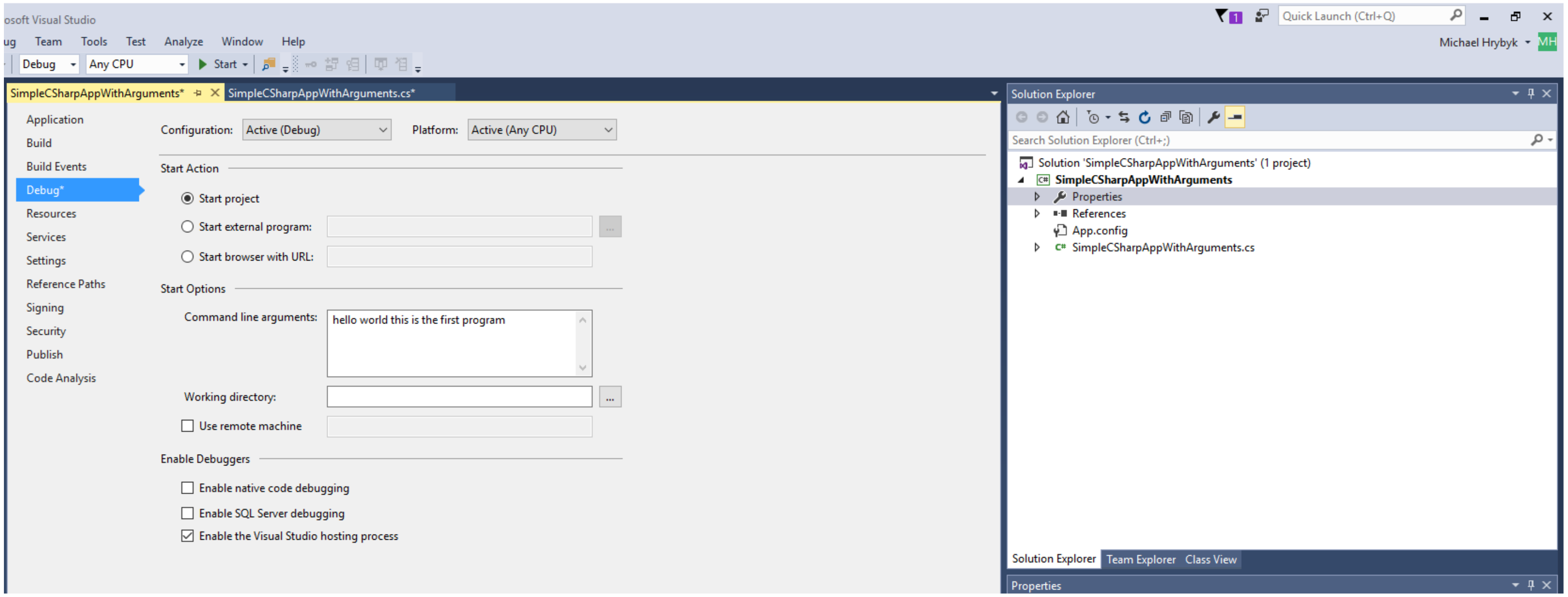
- **args** is string array
  - Iterate through this to process command line arguments
  - VERY useful for debugging programs without the need for Windows controls.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SimpleCSharpAppWithArguments
{
    class SimpleCSharpAppWithArgumentsProgram
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            for (int i = 0; i < args.Length; i++)
                Console.WriteLine(args[i]);

            Console.ReadLine();
        }
    }
}
```

Modify project properties and add arguments – then run


osoft Visual Studio
ug Team Tools Test Analyze Window Help
Debug Any CPU Start
Quick Launch (Ctrl+Q)
Michael Hrybyk MH
SimpleCSharpAppWithArguments* SimpleCSharpAppWithArguments.cs*
Application
Build
Build Events
Debug*
Resources
Services
Settings
Reference Paths
Signing
Security
Publish
Code Analysis
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SimpleCSharpAppWithArguments' (1 project)
SimpleCSharpAppWithArguments
Properties
References
App.config
SimpleCSharpAppWithArguments.cs
Solution Explorer Team Explorer Class View
Properties

Configuration: Active (Debug) Platform: Active (Any CPU)

Start Action

- [✓ Start project]
- Start external program:
- Start browser with URL:

Start Options

Command line arguments: hello world this is the first program

Working directory:

Use remote machine

Enable Debuggers

- Enable native code debugging
- Enable SQL Server debugging
- [✓ Enable the Visual Studio hosting process]

The arguments should be printed when the program is compiled and run

Console class

- In addition to Write(), WriteLine(), Read(), ReadLine()
- Methods
  - Beep() - This method forces the console to emit a beep of a specified frequency and duration.
  - Clear() This method clears the established buffer and console display area.
- Properties
  - **Title** - This property gets or sets the title of the current console.
  - Color Properties - set the background/foreground colors (ConsoleColor enumeration)
    - **BackgroundColor**
    - **ForegroundColor**
- See **BasicConsoleIO** example

Using the Console class

```
static void Main(string[] args)
{
    Console.WriteLine("***** Basic Console I/O *****");
    GetUserData();
    Console.ReadLine();
}
private static void GetUserData()
{
    // Get name and age.
    Console.Write("Please enter your name: ");
    string userName = Console.ReadLine();
    Console.Write("Please enter your age: ");
    string userAge = Console.ReadLine();

    // Change echo color, just for fun.
    ConsoleColor prevColor = Console.ForegroundColor;
    Console.ForegroundColor = ConsoleColor.Yellow;

    // Echo to the console.
    Console.WriteLine("Hello {0}! You are {1} years old.", userName, userAge);

    // Restore previous color.
    Console.ForegroundColor = prevColor;
}
```

String formatting

- Basic form
  - {N,J:Tp} where
    - N is placeholder or argument
    - J is number of characters to justify
      - -J is left
      - J is right
    - T is type
      - C or c – currency
      - D or d – decimal
      - E or e – exponential
      - F or f – fixed point
      - G or g – general (E, F, or G)
      - N or n – basic integers with commas
      - X or x – hexadecimal
      - P or p - percent
    - p is padding for EFG
      - For EFG: number of places to right of decimal point
      - For D: number of zeroes to pad in front of integer

- Example
  - WriteLine(“{0,10:c}”,”200”) would show ___$200.00 where _s are spaces

Formatting Numbers with the ToString Method

- The ToString method of an object can optionally format a number to appear in a specific way
- The following table lists the “format strings” and how they work with sample outputs

Format String	Description	Number	ToString()	Result
“N” or “n”	Number format	12.3	ToString(“n3”)	12.300
“F” or “f”	Fixed-point scientific format	123456.0	ToString("f2")	123456.00
“E” or “e”	Exponential scientific format	123456.0	ToString("e3")	1.235e+005
“C” or “c”	Currency format	-1234567.8	ToString("C")	($1,234,567.80)
“P” or “p”	Percentage format	.234	ToString("P")	23.40%

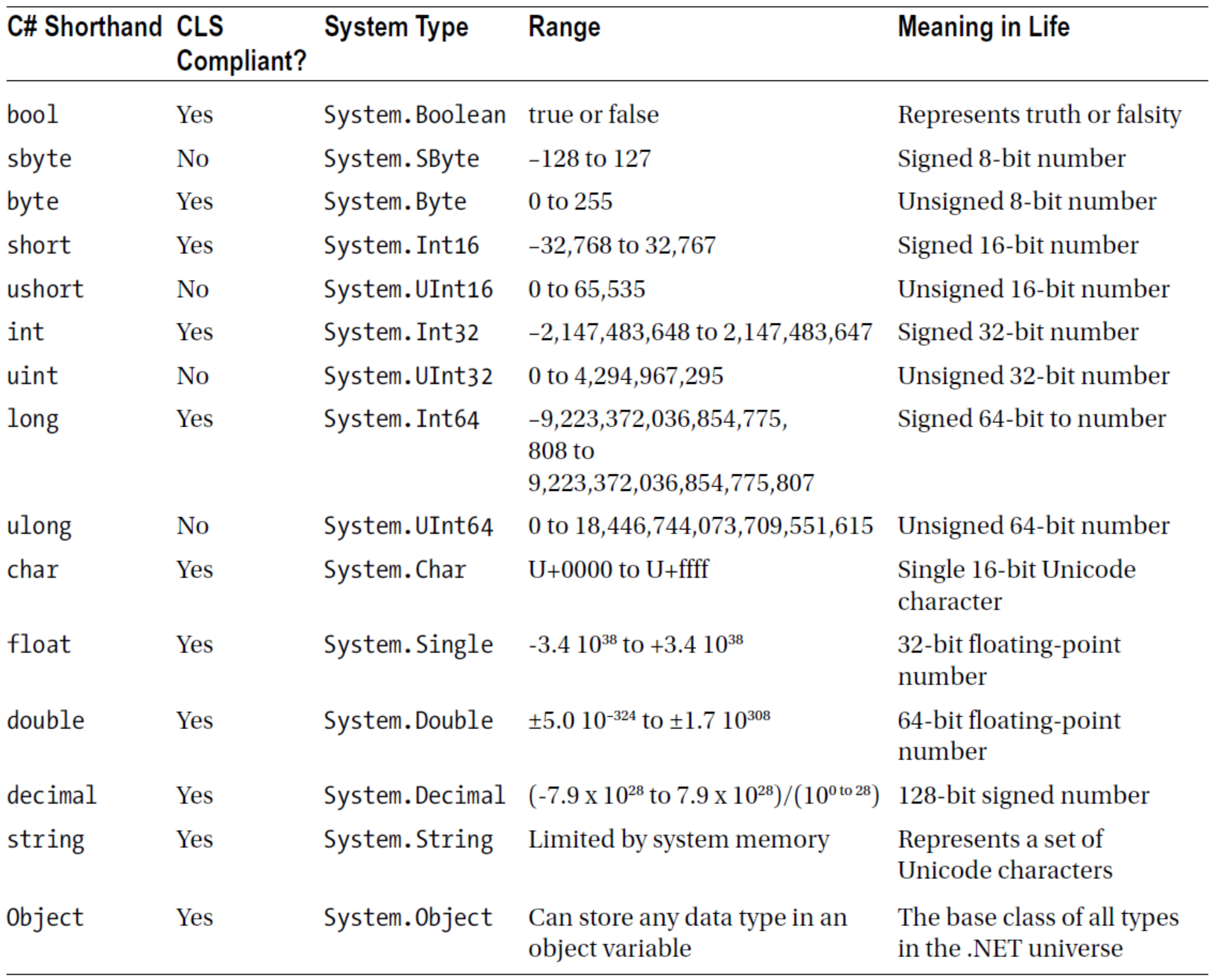
BasicConsoleIO

```
        /// <summary>
        /// Display a value using various string formats
        /// </summary>
        static void FormatNumericalData()
        {
            double percent = 0.25346;
            const int displayValue = 99999;
            WriteLine("{0} {1} {2}", "The value ", displayValue, " in various formats");
            WriteLine("10:c format: {0,10:c}", displayValue);
            WriteLine(" 20:d9 format right justified: *{0,20:d9}*", displayValue);
            WriteLine("-20:d9 format left justified: *{0,-20:d9}*", displayValue);

            WriteLine("f3 format: {0:f3}", displayValue);
            WriteLine("n4 format: {0:n4}", displayValue);
            // Notice that upper- or lowercasing for hex
            // determines if letters are upper- or lowercase.
            WriteLine("E format: {0:E}", displayValue);
            WriteLine("e format: {0:e}", displayValue);
            WriteLine("X format: {0:X}", displayValue);
            WriteLine("x format: {0:x}", displayValue);

            WriteLine($"Display {percent:P3} in various formats:");
            WriteLine("P5 percent format: {0:P5}", percent);
            WriteLine("P5 percent format tostring: " + percent.ToString("P5"));
        }
```

Intrinsic types

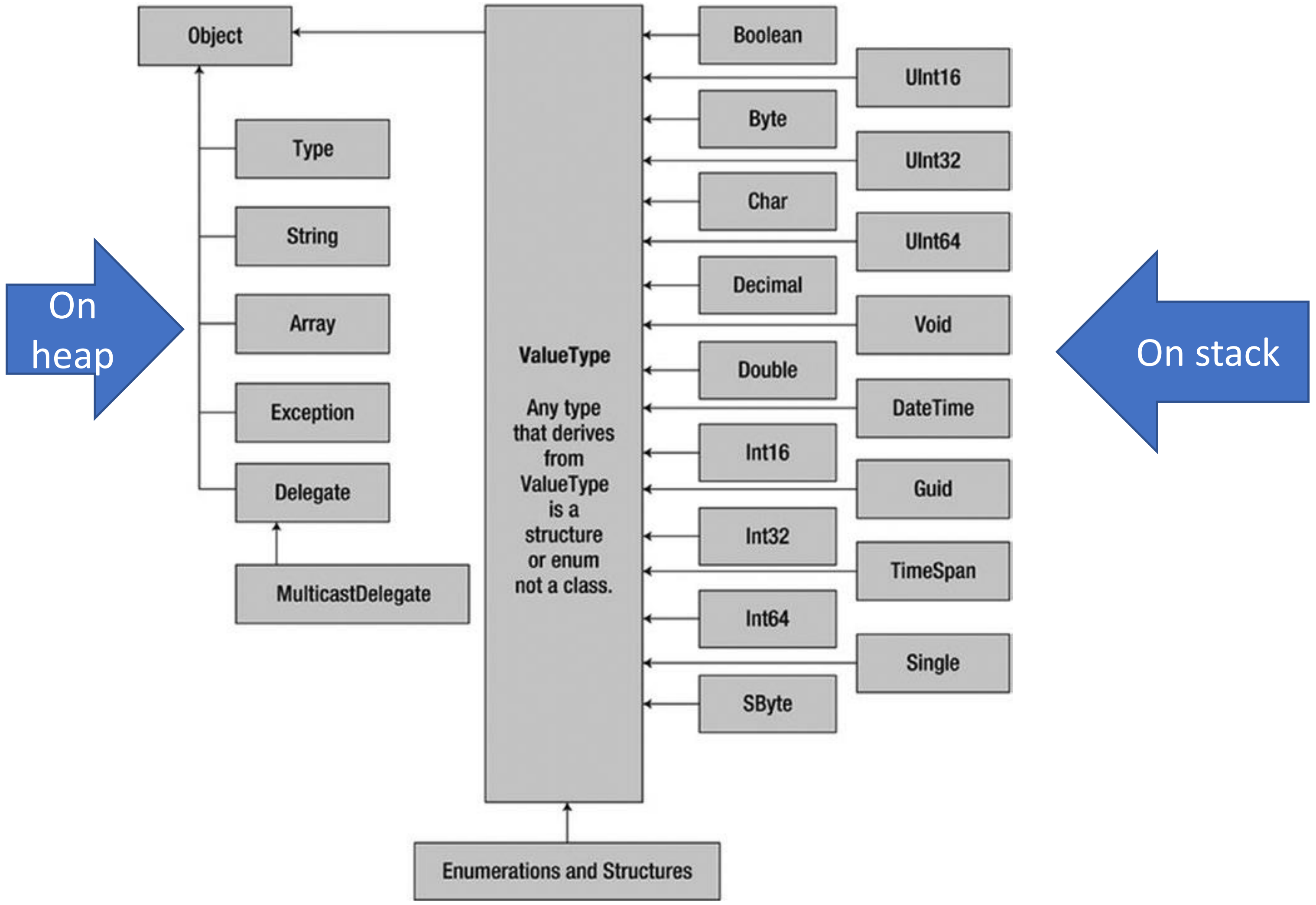


C# Shorthand	CLS Compliant?	System Type	Range	Meaning in Life
`bool`	Yes	`System.Boolean`	true or false	Represents truth or falsity
`sbyte`	No	`System.SByte`	–128 to 127	Signed 8-bit number
`byte`	Yes	`System.Byte`	0 to 255	Unsigned 8-bit number
`short`	Yes	`System.Int16`	–32,768 to 32,767	Signed 16-bit number
`ushort`	No	`System.UInt16`	0 to 65,535	Unsigned 16-bit number
`int`	Yes	`System.Int32`	–2,147,483,648 to 2,147,483,647	Signed 32-bit number
`uint`	No	`System.UInt32`	0 to 4,294,967,295	Unsigned 32-bit number
`long`	Yes	`System.Int64`	–9,223,372,036,854,775, 808 to 9,223,372,036,854,775,807	Signed 64-bit to number
`ulong`	No	`System.UInt64`	0 to 18,446,744,073,709,551,615	Unsigned 64-bit number
`char`	Yes	`System.Char`	U+0000 to U+ffff	Single 16-bit Unicode character
`float`	Yes	`System.Single`	-3.4 10^{38} to +3.4 10^{38}	32-bit floating-point number
`double`	Yes	`System.Double`	±5.0 10^{-324} to ±1.7 10^{308}	64-bit floating-point number
`decimal`	Yes	`System.Decimal`	(-7.9 x 10^{28} to 7.9 x 10^{28})/($10^{0 \text{ to } 28}$)	128-bit signed number
`string`	Yes	`System.String`	Limited by system memory	Represents a set of Unicode characters
`Object`	Yes	`System.Object`	Can store any data type in an object variable	The base class of all types in the .NET universe

Local variables and initialization

- Use
  - `Typename variableName`
- It is a *compiler error* to make use of a local variable before assigning an initial value.
- It is good practice to assign an initial value to your local data points at the time of declaration.

Data Class Hierarchy



Intrinsic types and properties/methods

- Object
  - ToString(), Equals(), GetHashCode()
- Numeric
  - MaxValue, MinValue, PositiveInfinity, NegativeInfinity, Epsilon
- Boolean
  - TrueString, FalseString
- Char
  - IsDigit(), IsPunctuation(), IsUpper(), IsWhiteSpace()
  - To*() transforms
- Parse Method
  - Every intrinsic value type has a parse method
  - int.Parse(“8”);

Date and Time

```
// This constructor takes (year, month, day)
DateTime dt = new DateTime(2015, 10, 17);

// What day of the month is this?
Console.WriteLine("The day of {0} is {1}", dt.Date, dt.DayOfWeek);
dt = dt.AddMonths(2);  // Month is now December.
Console.WriteLine("Daylight savings: {0}", dt.IsDaylightSavingTime());
```

```
            // This constructor takes (hours, minutes, seconds)
            TimeSpan ts = new TimeSpan(4, 30, 0);
            Console.WriteLine(ts);

            // Subtract 15 minutes from the current TimeSpan and
            // print the result.
            Console.WriteLine(ts.Subtract(new TimeSpan(0, 15, 0)));
```

Example

- See BasicDataTypes project
- Notice #region directive can help outline code

Strings

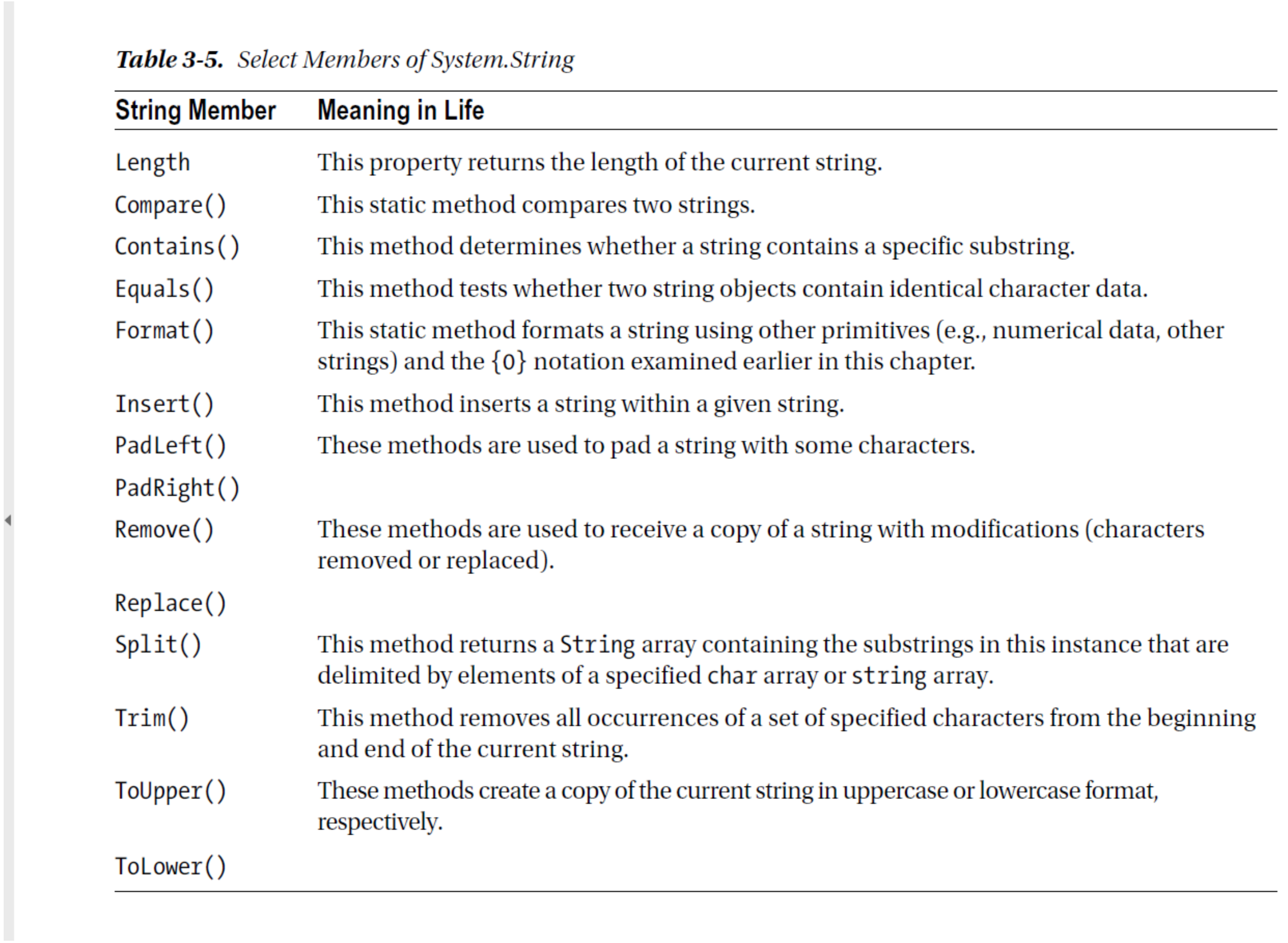


Table 3-5. Select Members of System.String

String Member	Meaning in Life
`Length`	This property returns the length of the current string.
`Compare()`	This static method compares two strings.
`Contains()`	This method determines whether a string contains a specific substring.
`Equals()`	This method tests whether two string objects contain identical character data.
`Format()`	This static method formats a string using other primitives (e.g., numerical data, other strings) and the `{0}` notation examined earlier in this chapter.
`Insert()`	This method inserts a string within a given string.
`PadLeft()`	These methods are used to pad a string with some characters.
`PadRight()`	
`Remove()`	These methods are used to receive a copy of a string with modifications (characters removed or replaced).
`Replace()`	
`Split()`	This method returns a `String` array containing the substrings in this instance that are delimited by elements of a specified `char` array or `string` array.
`Trim()`	This method removes all occurrences of a set of specified characters from the beginning and end of the current string.
`ToUpper()`	These methods create a copy of the current string in uppercase or lowercase format, respectively.
`ToLower()`	

Strings

- Are immutable
  - Stored on the heap
  - Functions return a new string
  - Garbage collected
- Concatenation
  - Use of + or += operator invokes Concat() method (overloading of operators)
- Strings are references so the equality operator (=) WILL NOT WORK AS EXPECTED
- Comparison
  - Unlike Java, the == logical operator works with strings! No need to use Compare() method.
- Use of mutable StringBuilder class to operate on objects in place
  - Use of Append() and AppendFormat()
  - Used in upcoming lab
- String Interpolation
  - Use of $ to signify that a string will include objects within {}
  - $"\tHello {name.ToUpper()} you are {age} years old."

Escape Characters and Verbatim Strings

- Need a way to include special characters like tabs and newlines.
- Prefacing a string with an @ means to use the string as is, with NO escape character processing.
  - Useful for Windows path names
  - @"C:\MyApp\bin\Debug"

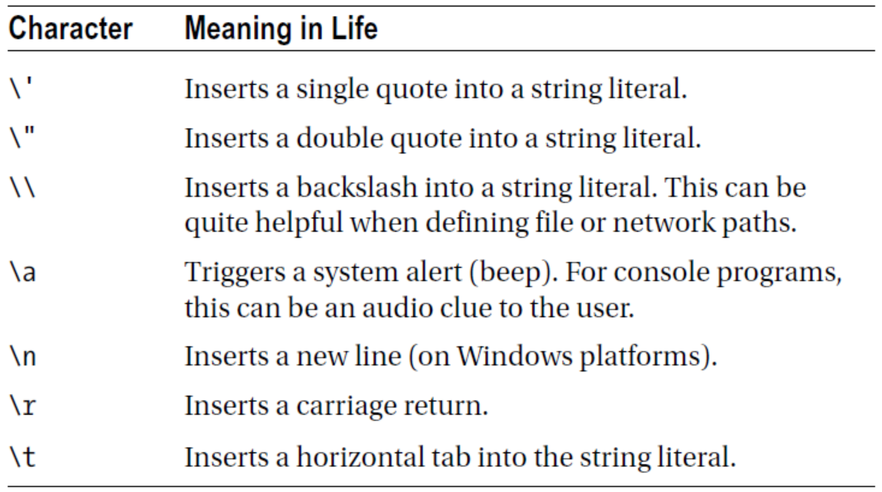


Table 3-6. *String Literal Escape Characters*

Character	Meaning in Life
`\'`	Inserts a single quote into a string literal.
`\"`	Inserts a double quote into a string literal.
`\\`	Inserts a backslash into a string literal. This can be quite helpful when defining file or network paths.
`\a`	Triggers a system alert (beep). For console programs, this can be an audio clue to the user.
`\n`	Inserts a new line (on Windows platforms).
`\r`	Inserts a carriage return.
`\t`	Inserts a horizontal tab into the string literal.

Example

- See FunWithStrings
- Pay attention to equality functions

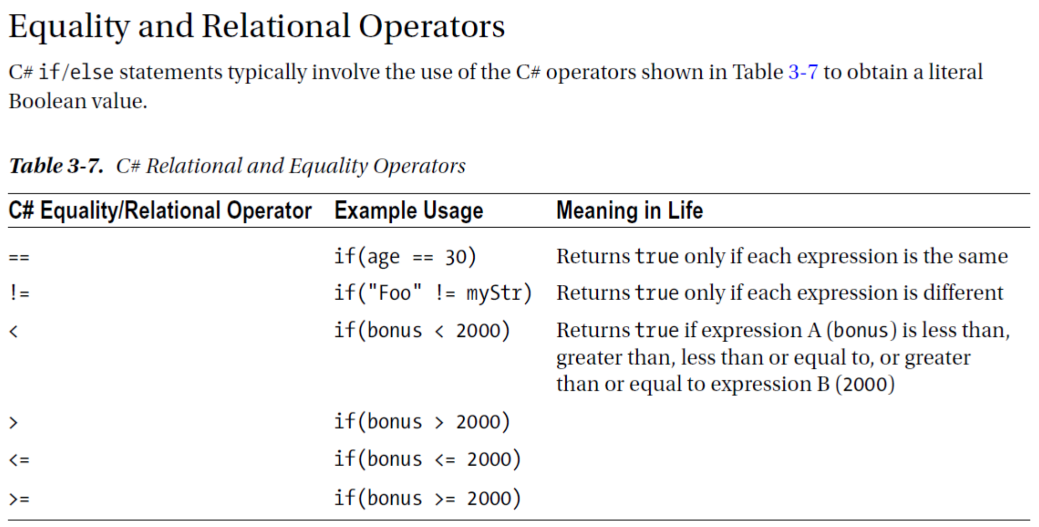
Narrowing and Widening

- Upcasting to a larger value type is fine and is done automatically
- Downcasting to a smaller one is will be rejected by the compiler unless an explicit cast is made
- See TypeConversions

Implicit types using var

- Variables can be implicitly typed using var
  - var myInt = 0;
  - var myBool = true;
  - var myString = "Time, marches on...";
  - myString = 2; // is an error!! C# is strongly typed
- Cannot be used as a return value
- Must be initialized
  - But not to null
- See ImplicitlyTypedLocalVars Example
  - Pay attention to LINQ example
  - Implicit types very useful in LINQ

Equality and Relational Operators

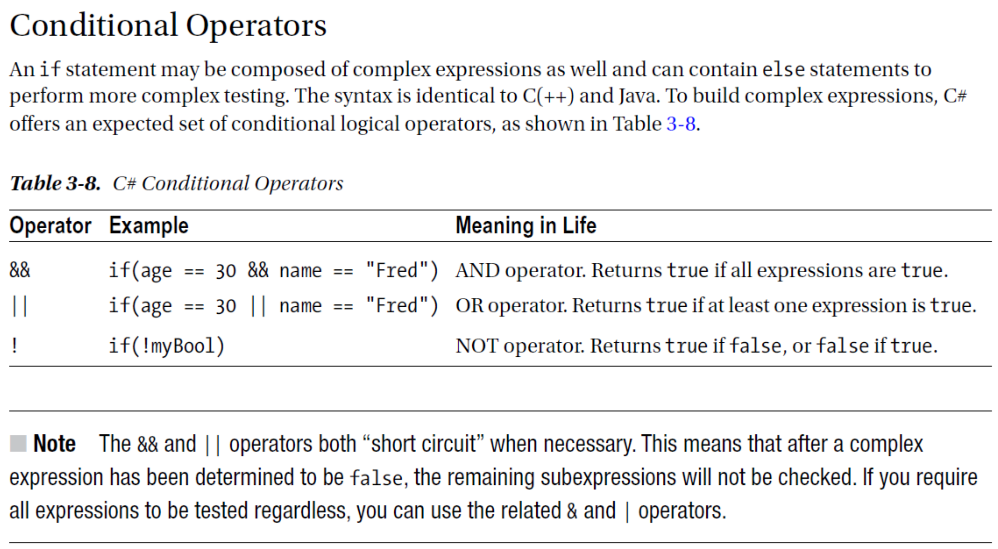


Equality and Relational Operators

C# `if/else` statements typically involve the use of the C# operators shown in Table 3-7 to obtain a literal Boolean value.

Table 3-7. C# Relational and Equality Operators

C# Equality/Relational Operator	Example Usage	Meaning in Life
`==`	`if(age == 30)`	Returns `true` only if each expression is the same
`!=`	`if("Foo" != myStr)`	Returns `true` only if each expression is different
`<`	`if(bonus < 2000)`	Returns `true` if expression A (`bonus`) is less than, greater than, less than or equal to, or greater than or equal to expression B (`2000`)
`>`	`if(bonus > 2000)`	
`<=`	`if(bonus <= 2000)`	
`>=`	`if(bonus >= 2000)`	



Conditional Operators

An `if` statement may be composed of complex expressions as well and can contain `else` statements to perform more complex testing. The syntax is identical to C(++) and Java. To build complex expressions, C# offers an expected set of conditional logical operators, as shown in Table 3-8.

Table 3-8. C# Conditional Operators

Operator	Example	Meaning in Life
`&&`	`if(age == 30 && name == "Fred")`	AND operator. Returns `true` if all expressions are `true`.
`||`	`if(age == 30 || name == "Fred")`	OR operator. Returns `true` if at least one expression is `true`.
`!`	`if(!myBool)`	NOT operator. Returns `true` if `false`, or `false` if `true`.

Note The `&&` and `||` operators both "short circuit" when necessary. This means that after a complex expression has been determined to be `false`, the remaining subexpressions will not be checked. If you require all expressions to be tested regardless, you can use the related `&` and `|` operators.

Iterations and Decisions

- for loop
- foreach/in loop
- while loop
- do/while loop
- if-then-else
- switch
- Example
    - See IterationsAndDecisions

Summary

- This has been a whirlwind tour of C# basics
  - From Types to Strings to Iteration and Control Flow
- For review, also see the Gaddis book from CSIS1175