Introduction to C#

CS513: System Security

Spring 2004

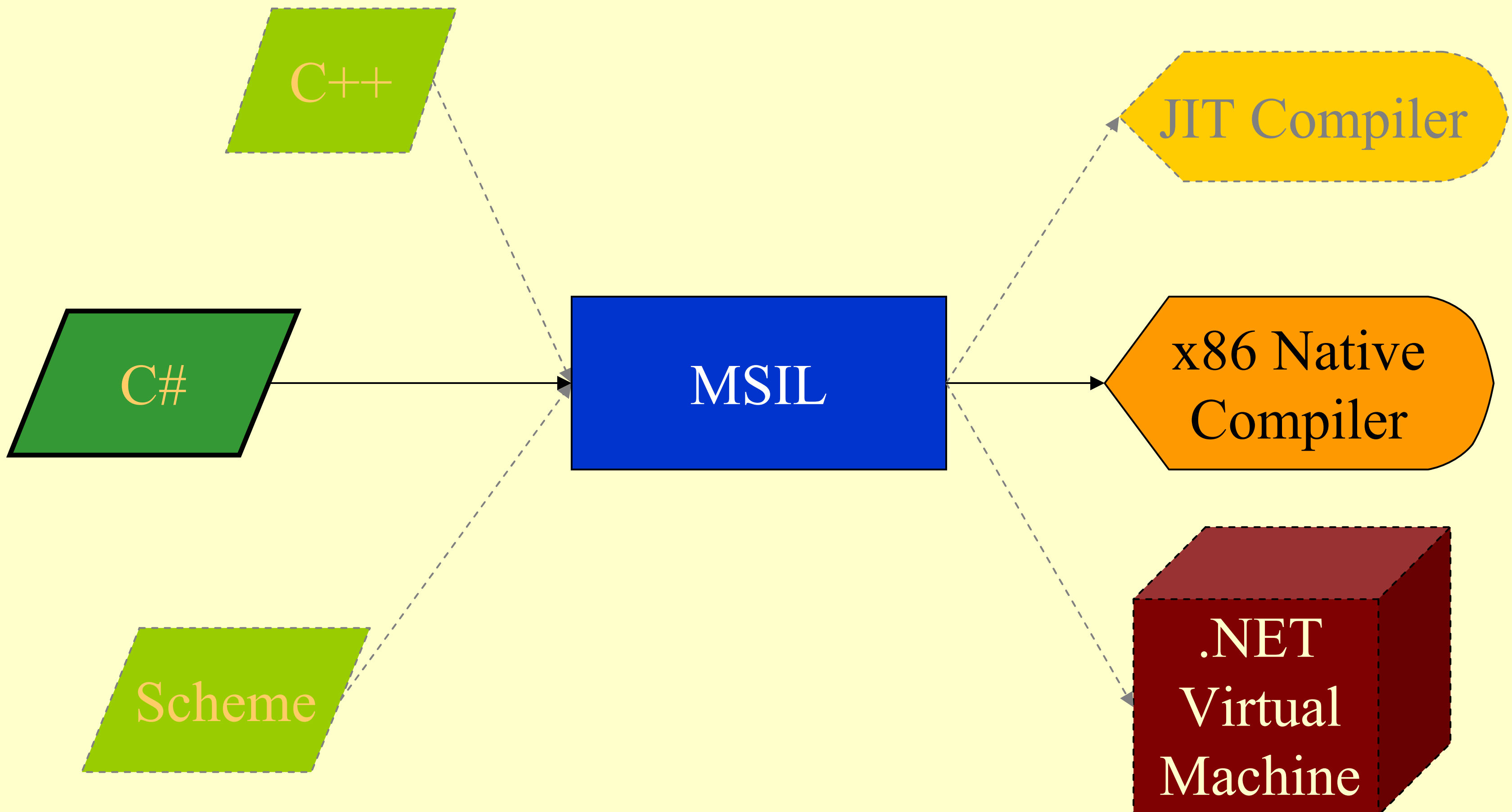
Kevin Hamlen

hamlen@cs.cornell.edu

Outline

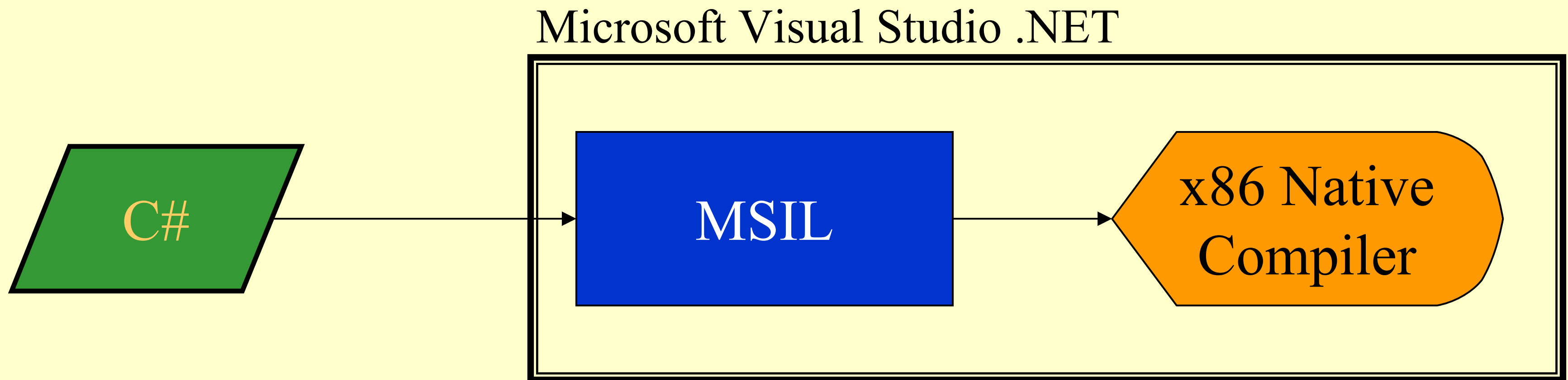
- .NET Framework Architecture
- Development Environment
- **The C# Language**
- Common Language Runtime (CLR)
- eXtensible Markup Language (XML)

.NET Framework Architecture:
MSIL (MicroSoft Intermediate Language)



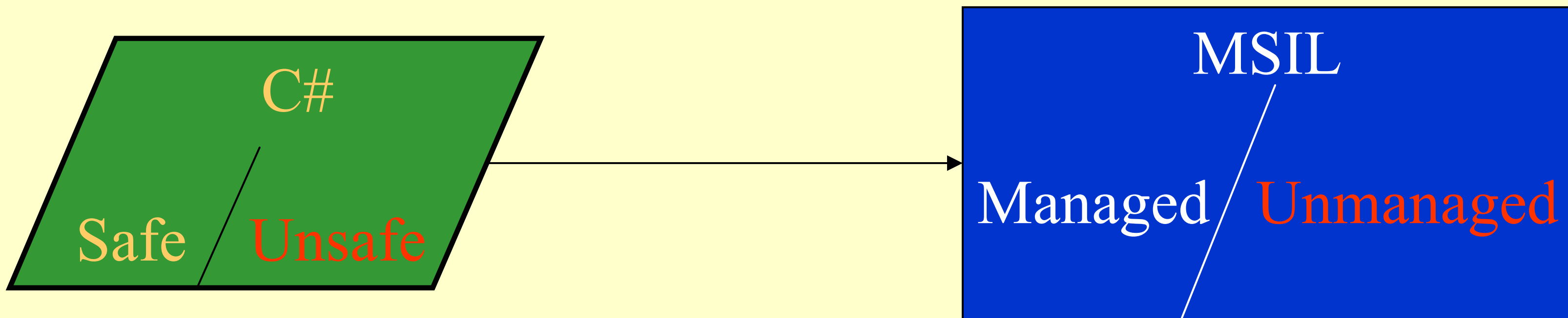
.NET Framework Architecture:

MSIL (MicroSoft Intermediate Language)



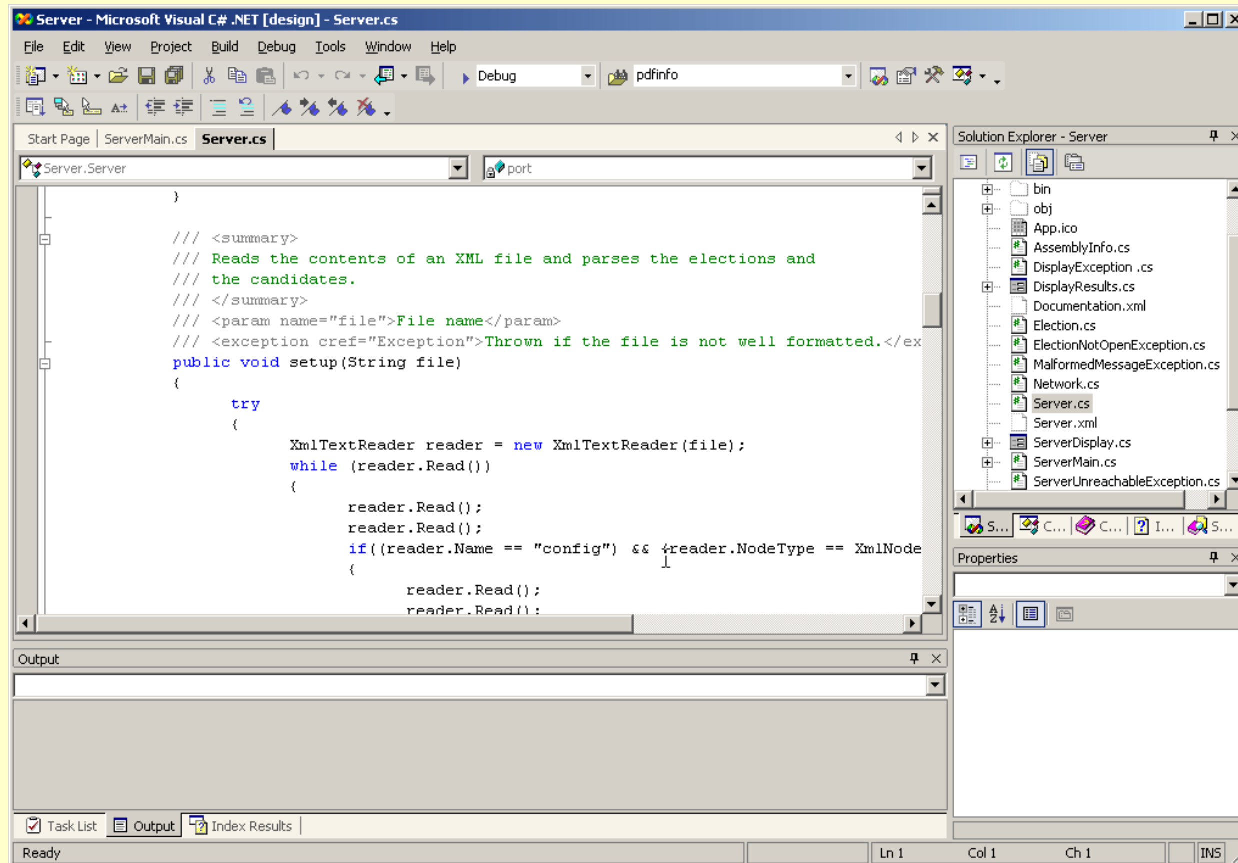
.NET Framework Architecture:

Managed vs. Unmanaged MSIL



- Only use the SAFE subset of C#:
  - Don't import your own unmanaged dll's
  - Don't use the `/unsafe` compiler option
  - Don't use the `unsafe` C# keyword

Development Environment: Microsoft Visual Studio .NET



The C# Language

- Type System
  - Value types and Reference types
  - Type Conversions
  - New Types: structs, enums, multidim. arrays, delegates
- Object System
  - Namespaces (packages)
  - Inheritance
  - Properties
  - Interfaces
    - Collection Interfaces
  - Operator Overloading & Cast Operators
- Attributes

Our First C# Program

```
class Addition {
	static void Main(string[] args) {
		decimal sum = 0;
		foreach(string s in args) {
			sum += System.Convert.ToDecimal(s);
		}
		System.Console.WriteLine("The sum is: {0}", sum);
	}
}
```

Our First C# Program

```
class Addition {
        static void Main(string[] args) {
                decimal sum = 0;
                foreach(string s in args) {
                        sum += System.Convert.ToDecimal(s);
                }
                System.Console.WriteLine("The sum is: {0}", sum);
        }
}
```

strings

arrays

numerical types

conversions

C# Value Types

- Same as Java: char, float, double, int, long, short
- bool (Java: boolean)
- sbyte (Java: byte)
- byte – unsigned byte (unlike Java!)
- New types: decimal, uint, ulong, ushort
- Structs and Enumerations… (later)

C# Reference Types

- Class types (objects)
- Strings
- Interfaces
- Arrays (like in Java, plus more)
- Delegates (method pointers; not in Java)

Value Types vs. Reference Types

```
type a, b;
a = const1;
b = a;
a = const2;
```

- Does b==a after this fragment runs?
  - No, if `type` is a Value Type
  - Yes, if `type` is a Reference Type

Conversions: Boxing and Unboxing

boxing – like allocating an instance of a one-member class and copying the value into it

unboxing – like extracting a value from a one-member class

```
int i=123, j;
object box = i; // i has been boxed
try {
  j = (int) box; // box has been unboxed
}
catch (System.InvalidCastException) {
  System.Console.WriteLine("That wasn't a boxed int!");
}
```

Conversions: Implicit vs. Explicit

- Some conversions can be done implicitly.

```
int i=123;
double j = i; // implicit conversion from int to double
```

- Others need an explicit cast.

```
int i=123;
byte j = (byte) i; // explicit cast from int to byte
```

- Yet others need require a library call.

```
string s = "123";
decimal d = System.Convert.ToDecimal(s);
           // explicit conversion from string to decimal
```

New Casting / Type-checking Operators

- The **as** keyword can be used to cast:

```
MyClass c;    // suppose MyClass extends MyParent
…
MyParent p = (c as p);
```

- The **is** keyword can be used to test (like Java's *instanceof*):

```
object o;
…
if (o is int) {  // perhaps o is a boxed int
  …
}
```

Cool New Stuff

- Structs (unboxed classes)
- Enums
- Arrays (rectangular multidimensional)
- Delegates (method pointers)

Structs

- Like classes but are value types – they get copied
- Can box or unbox to convert between structs and classes
- Structs are always children of System.Object and have no children.
- Constructor must have parameters. Default constructor always assigns 0 to all fields.
- No destructors allowed
- Instance fields must not have initializers.
- The `this` pointer has a different meaning—it can be assigned to.

Struct Example

```
public struct fraction
{
        public int numerator;
        public int denominator;
        public fraction(int n, int d)
        {
                numerator = n;
                denominator = d;
                if (denominator==0)
                  this = new fraction(0,1);
        }
}
```

Enums

```
enum TerrorAlert
{
  Low,
  Guarded,
  Elevated,
  High,
  Severe,
  Scary=High
}
```

```
enum TerrorAlert : uint
{
  Low=0,
  Guarded=1,
  Elevated=2,
  High=3,
  Severe=4,
  Scary=3
}
```

```
TerrorAlert level;

if (level >= Scary)
  Flight.Reserve(Cheney, Undisclosed_Location);
```

Arrays

- Same as Java except that C# has true multidimensional rectangular arrays:

```
int[,] Identity1 = {{1,0},{0,1}};

int[,] Identity2 = new int[2,2];
Identity2[0,0] = Identity2[1,1] = 1;
```

- C# uses arrays to implement variable-length parameter lists:

```
static long Sum(params int[] p) {
  long sum = 0;
  for (int i=0; i<p.Length; i++) sum += p[i];
  return sum;
}
long s = Sum(1,2,3,4,5);
```

Delegates

```
class Repeater {
  public delegate void MyDelegate(int i);
  public static void Repeat(MyDelegate f, uint n) {
    for (int i=0; i<n; ++i) f(i);
  }
}

class CountingClass {
  public void PrintInt(int i) {
    System.Console.WriteLine("{0} ", i);
  }
  public void CountTo(uint n) {
    Repeater.MyDelegate d =
        new Repeater.MyDelegate(PrintInt);
    Repeater.Repeat(d, n);
  }
}
```

Delegates

```
class Repeater {
  public delegate void MyDelegate(int i);
  public static void Repeat(MyDelegate f, uint n) {
    for (int i=0; i<n; ++i) f(i);
  }
}

class CountingClass {
  public void PrintInt(int i) {
    System.Console.WriteLine("{0} ", i);
  }
  public void CountTo(uint n) {
    Repeater.MyDelegate d =
       new Repeater.MyDelegate(PrintInt);
    Repeater.Repeat(d, n);
  }
}
```

declares a new type named MyDelegate

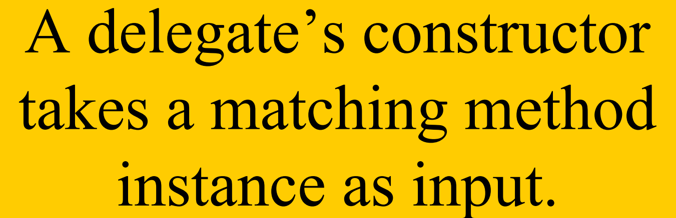
Delegates

```
class Repeater {
  public delegate void MyDelegate(int i);
  public static void Repeat(MyDelegate f, uint n) {
    for (int i=0; i<n; ++i) f(i);
  }
}

class CountingClass {
  public void PrintInt(int i) {
    System.Console.WriteLine("{0}
  }
  public void CountTo(uint n) {
    Repeater.MyDelegate d =
        new Repeater.MyDelegate(PrintInt);
    Repeater.Repeat(d, n);
  }
}
```

instances of MyDelegate are functions that take an integer and return void

Delegates

```
class Repeater {
  public delegate void MyDelegate(int i);
  public static void Repeat(MyDelegate f, uint n) {
    for (int i=0; i<n; ++i) f(i);
  }
}

class CountingClass {
  public void PrintInt(int i) {
    System.Console.WriteLine("{0} ", i);
  }
  public void CountTo(uint n) {
    Repeater.MyDelegate d =
       new Repeater.MyDelegate(PrintInt);
    Repeater.Repeat(d, n);
  }
}
```



Delegates

```
class Repeater {
  public delegate void MyDelegate(int i);
  public static void Repeat(MyDelegate f, uint n) {
    for (int i=0; i<n; ++i) f(i);
  }
}

class CountingClass {
  public void PrintInt(int i) {
    System.Console.WriteLine("{0} ", i);
  }
  public void CountTo(uint n) {
    Repeater.MyDelegate d =
       new Repeater.MyDelegate(PrintInt);
    Repeater.Repeat(d, n);
  }
}
```

The delegate now serves as a method pointer.

The C# Language

- Type System
  - Value types and Reference types
  - Type Conversions
  - New Types: structs, enums, multidim. arrays, delegates
- Object System
  - Namespaces (packages)
  - Inheritance
  - Properties
  - Interfaces
    - Collection Interfaces
  - Operator Overloading & Cast Operators
- Attributes

Namespaces

```
namespace MyStuff {
  class MyClass {
    public void Hello() {
      System.Console.WriteLine(“Hello World!");
    }
  }
}

MyStuff.MyClass.Hello();
```

A Shortcut: `using`

```
using MyStuff;

MyClass.Hello();
```

Inheritance

```
class MyClass : MyParentClass {
   …
}
```

- Single-inheritance only (like Java)
- Five possible access modifiers instead of four:
  - **public** – same as Java
  - **internal** – same-assembly access (like Java package default)
  - **protected** – same- and derived-class access only (other classes in same namespace do NOT get access)
  - **protected internal** – like Java's protected
  - **private** – same as Java

Inheritance (cont.)

- Four forms of member polymorphism:
  - **virtual** – can be overridden (default in Java but not in C# !)
  - **sealed** – cannot be overridden (*final* in Java)
  - **abstract** – must be overridden (same as Java)
  - **new** – member does NOT override matching base class member (no Java equivalent)
- Two class modifiers:
  - **abstract** – same as Java; class not instantiable
  - **sealed** – *final* in Java; class not subclassable

Properties

Properties are fields with embedded get/set accessor functions:

```
class Temperature {
  private double kelvin;
  public double Celsius {
    get { return kelvin - 273.15; }
    set { kelvin = value + 273.15; }
  }
}
…
  Temperature t1, t2;
  t1.Celsius = 42*t2.Celsius;
```

Interfaces

```
interface MyInterface { … }

class MyClass : MyParentClass, MyInterface { … }
```

- Pretty much the same as Java except for syntax (just use colon instead of *extends* or *implements*)
- A particularly important interface: **System.Collections.ICollection**
  - Any class that implements the ICollection interface can be used with the C# **foreach** statement.
  - Many classes provided by the runtime libraries implement ICollection. Example: arrays

Collections Example: Arrays

```
class Test
{
   static void Main() {
      double[,] values = { {1.2, 2.3, 3.4, 4.5},
                           {5.6, 6.7, 7.8, 8.9} };
      foreach (double n in values)
         Console.Write("{0} ", n);
      Console.WriteLine();
   }
}
```

Operator Overloading

```
class Fraction
{
  public int num, denom;
  …
  public static Fraction operator +(Fraction f1,
                                    Fraction f2) {
    return new Fraction(
      f1.num*f2.denom + f2.num*f1.denom,
      f1.denom * f2.denom
    );
  }
}
…
  Fraction f1, f2;
  …
  Fraction f3 = f1+f2;
```

The C# Language

- Type System
  - Value types and Reference types
  - Type Conversions
  - New Types: structs, enums, multidim. arrays, delegates
- Object System
  - Namespaces (packages)
  - Inheritance
  - Properties
  - Interfaces
    - Collection Interfaces
  - Operator Overloading & Cast Operators
- Attributes

Attributes

- Every class, method, field, formal parameter, etc. can be tagged with annotations called “attributes”.
- Attributes are method calls with argument lists (but these methods are not necessarily called).
- Attributes not recognized by the .NET runtime will be ignored; others cause the runtime to take specific actions.
- Even ignored attributes can be useful:
  - Using the Reflection libraries, your code can examine the attributes on accessible classes.
  - You can use annotations to tag your code for use with third-party tools like optimizers or analysis tools.

Attribute Example

```
[STAThread()]
static void Main()
{
  …
```

- tags the Main() method with an attribute
- attribute is a method call to the STAThread() member of class System.STAThreadAttribute
- This particular attribute tells the runtime that the application should be run in Single-Threaded Apartment mode.

Outline

- .NET Framework Architecture
- Development Environment
- **The C# Language**
- Common Language Runtime (CLR)
- eXtensible Markup Language (XML)

Common Language Runtime

- Similar to Java Class Library
- .NET Visual Studio has extensive online documentation
  - threads: **System.Threading**, **System.Threading.Thread**
  - streams: **System.IO.Stream**, **System.Security.Cryptography.CryptoStream**
- Tip: Hovering mouse over enigmatic bits of C# code in Visual Studio can yield useful info.

XML

- Like HTML but for representing arbitrary structured data
- Two uses in C#:
  - CLR provides libraries for doing XML I/O. (See **System.Xml** documentation.)
  - Visual Studio can generate XML documentation for your code automatically.
  - Various tools can then convert the XML into HTML or other human-readable formats.

XML for Source Documentation

- In-source XML documentation looks like:

```
/// <summary>
/// Reads the contents of an XML file and parses the
/// elections and the candidates.
/// </summary>
/// <param name="file">File name</param>
/// <exception cref="Exception">Thrown if the file is not
/// well formatted.</exception>
public void setup(String file) {
       …
```

- XML code is placed in /// comments
- Visual Studio can extract comment data to generate XML documentation data file
- See course webpage for XML language conventions