Using .NET and Visual Studio

CSIS 3540

Client Server Systems

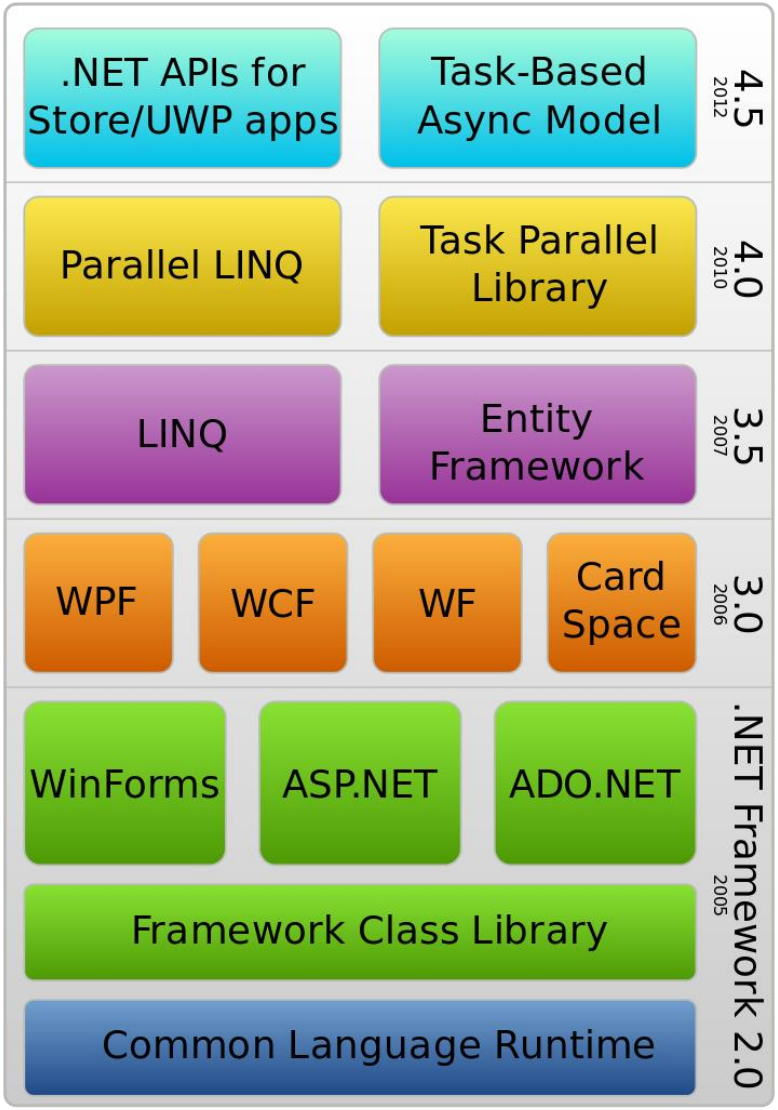
Class 01

Topics

- Overview of .NET
- Use of Visual Studio
- Creating basic applications
- Naming Conventions

.NET Framework

- .NET Framework (pronounced dot net) is a software framework developed by Microsoft that runs primarily on Microsoft Windows.
- It includes a large class library known as Base Class Library (BCL)
  - provides language interoperability (each language can use code written in other languages) across several programming languages.
- Programs in the Common Language Runtime (CLR)
  - application virtual machine that provides services such as security, memory management, and exception handling. (As such, computer code written using .NET Framework is called "managed code".)
- BCL and CLR together constitute .NET Framework.
- From Wikipedia https://en.wikipedia.org/wiki/.NET_Framework



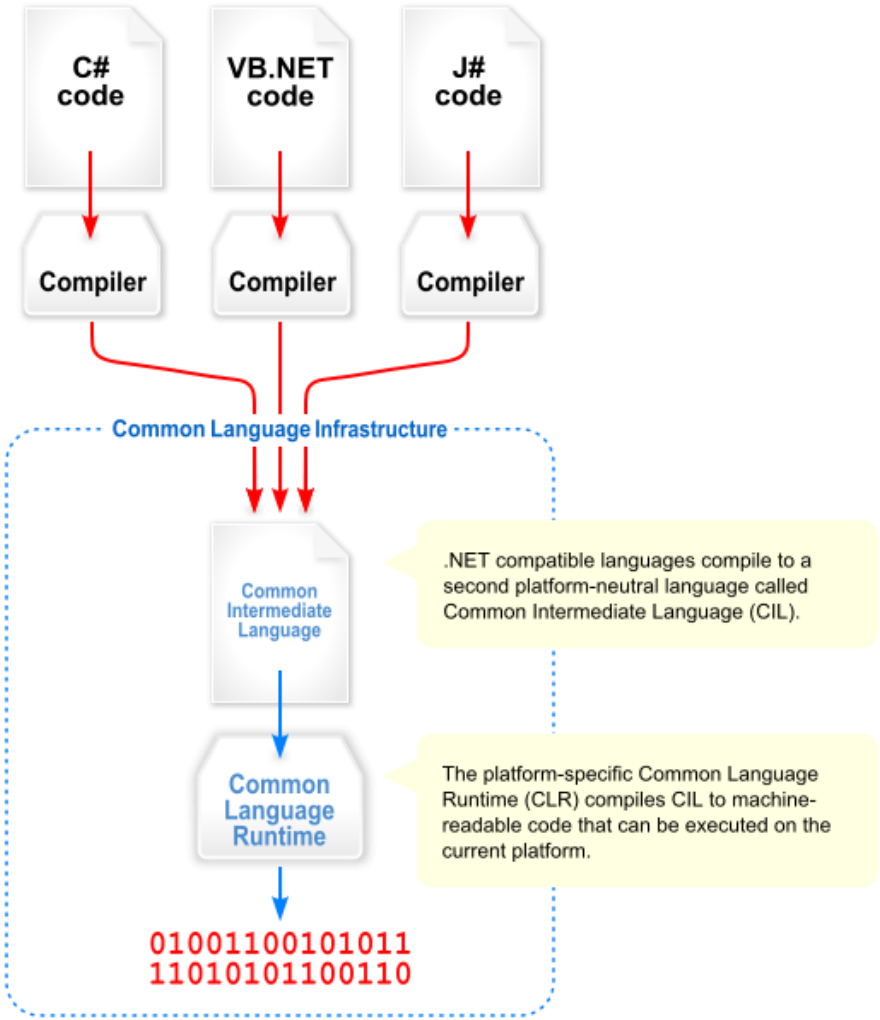
.NET Framework

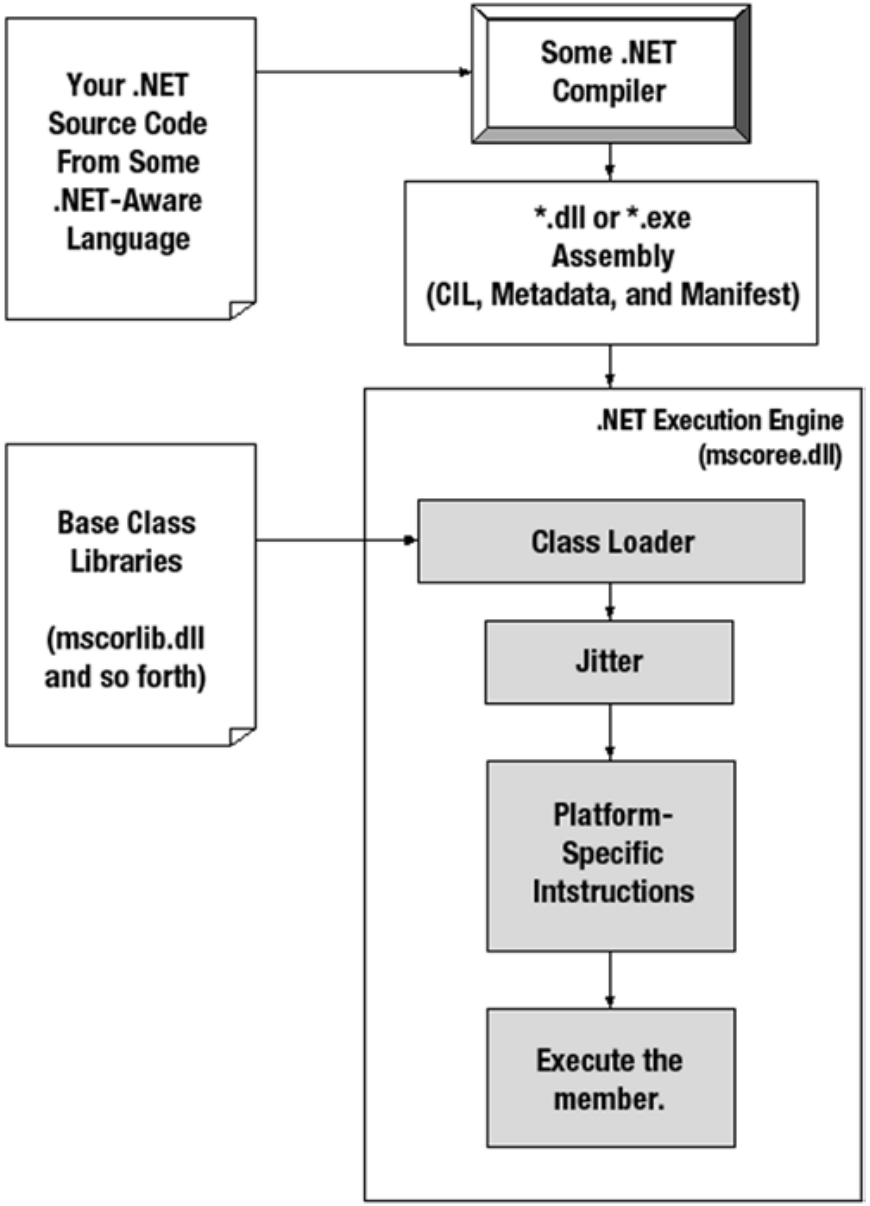
- Components and their Acronyms
  - Common Intermediate Language (CIL), and just-in-time (JIT) compilation.
  - Common Language Runtime (CLR), the Common Type System (CTS), and the Common Language Specification (CLS).
- Base Class Libraries (BCL)

.NET Benefits and Features

- Interoperability with existing code
- Support for numerous programming languages
  - .NET applications can be created using any number of programming languages (C#, Visual Basic, F#, and so on).
- A common runtime engine shared by all .NET-aware languages
  - One aspect of this engine is a well-defined set of types that each .NET-aware language understands.
- Language integration – cross language
  - inheritance
  - exception handling
  - debugging of code.
  - For example, you can define a base class in C# and extend this type in Visual Basic.
- A comprehensive base class library – predefined types to build
  - code libraries
  - simple terminal applications
  - graphical desktop application
  - enterprise-level web sites.
- A simplified deployment model
  - not registered into the system registry
  - allows multiple versions of the same *.dll to exist in harmony on a single machine.

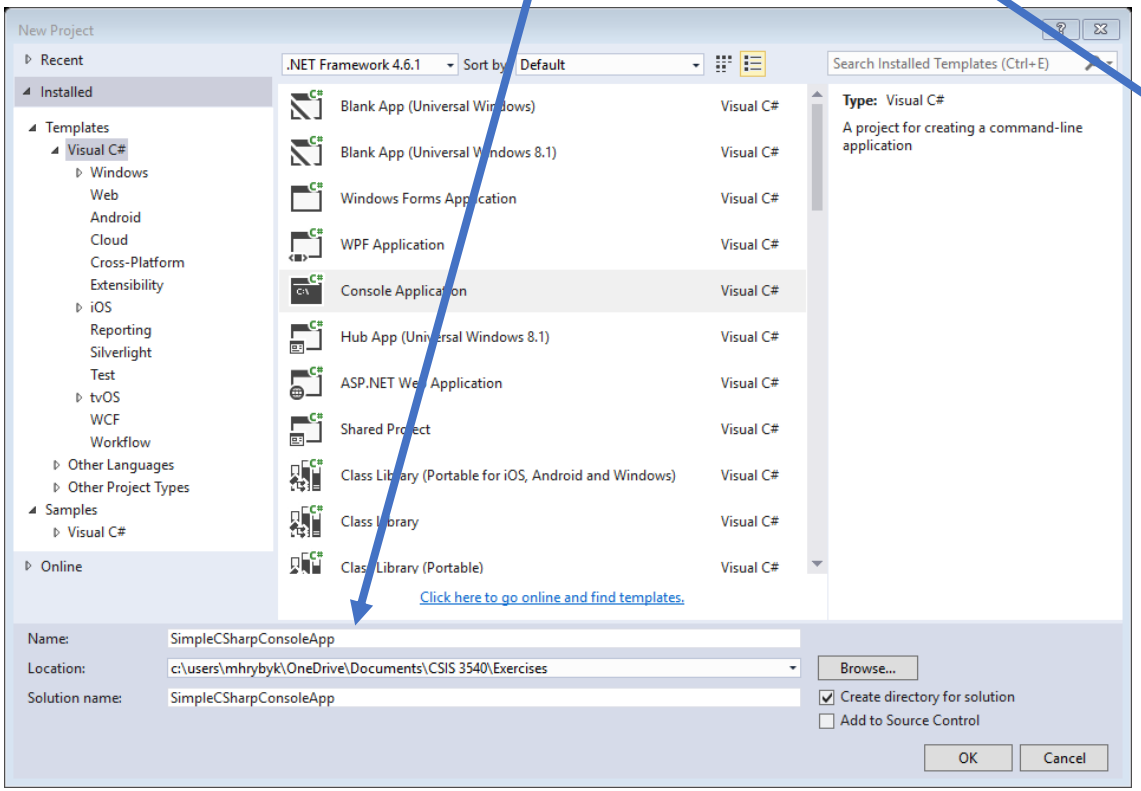
How it works …

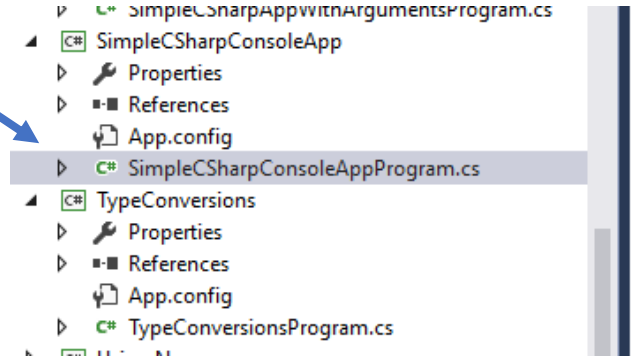




Visual Studio Project

- Create new project
- Name it SimpleCSharpConsoleApp
- Rename Program.cs to be SimpleCSharpConsoleAppProgram.cs



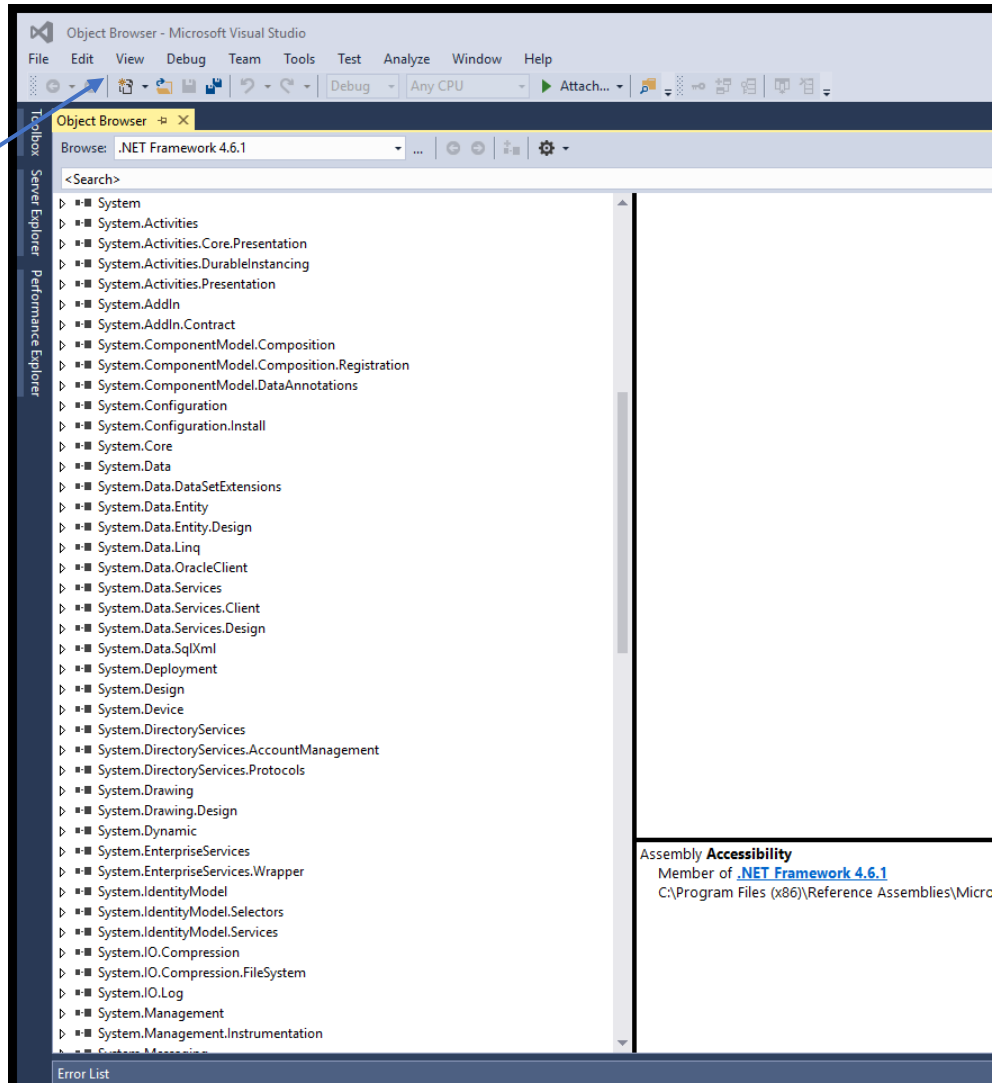


Namespaces and Types

- To keep all the types within the base class libraries well organized, the .NET platform makes extensive use of the *namespace* concept.
- A namespace is a grouping of semantically related types contained in an assembly or possibly spread across multiple related assemblies.
  - the System.IO namespace contains file I/O-related types, the System.Data namespace defines basic database types, and so on. It is important to point out that a single assembly (such as mscorlib.dll) can contain any number of namespaces, each of which can contain any number of types.
- Look at the Visual Studio Object Browser utility (which can be found under the View menu).
  - This tool allows you to examine the assemblies referenced by your current project, the namespaces within a particular assembly, the types within a given namespace, and the members of a specific type.
  - Note that the **mscorlib.dll** assembly contains many different namespaces (such as System.IO), each with its own semantically related types (e.g., BinaryReader).
- <u>A single assembly can have any number of namespaces, and namespaces can have any number of types</u>
- Namespaces and types carry across languages!

Visual Studio Object Browser

- Open Visual Studio
- Under View, click Object Browser
- Select .NET Framework and scroll to see namespaces



.NET Namespaces

- System – contains types
  - intrinsic data
  - mathematical computations
  - random number generation
  - environment variables
  - garbage collection
  - exceptions and attributes
  - Important sub namespaces of System
    - IO
    - Data
    - Windows (forms, etc)
    - Collections
    - LINQ
    - XML
    - And others …
- Microsoft
  - MS specific types

Using Namespaces

- a namespace is nothing more than a convenient way for us mere humans to logically understand and organize related types.
- Consider again the System namespace.
  - It looks like System.Console represents a class named Console that is contained within a namespace called System.
  - However, in the eyes of the .NET runtime, this is not so. The runtime engine sees only a single class named System.Console.
  - Think of a namespace as a tag or label that allows for shortcut naming
- In C#, the **using** keyword simplifies the process of referencing types defined in a particular namespace.

```
using System;
// using System.IO;

namespace Using_Namespaces
{
    class Program
    {
        static void Main(string[] args)
        {
            string lines =
System.IO.File.ReadAllText(@"..\..\testfile.txt");

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

```
using System;
using System.IO;

namespace Using_Namespaces
{
    class Program
    {
        static void Main(string[] args)
        {
            string lines = File.ReadAllText(@"..\..\testfile.txt");

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

Enter code and set a breakpoint

- Set a breakpoint at Console.Title line
- Compile and run
- Use
  - Debug->QuickWatch
  - to examine the value of Console.Title

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

namespace SimpleCSharpConsoleApp
{
    class SimpleCSharpConsoleAppProgram
    {
        static void Main(string[] args)
        {
            int i = 5;
            // Set up Console UI (CUI)
            i = i + 10;
            Console.Title = "My Rocking App";
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.BackgroundColor = ConsoleColor.Blue;
            Console.WriteLine("*************************************");
            Console.WriteLine("***** Welcome to My Rocking App *****");
            Console.WriteLine("*************************************");
            Console.BackgroundColor = ConsoleColor.Black;

            // Wait for Enter key to be pressed.
            Console.ReadLine();
            MessageBox.Show("All done!");
        }
    }
}
```

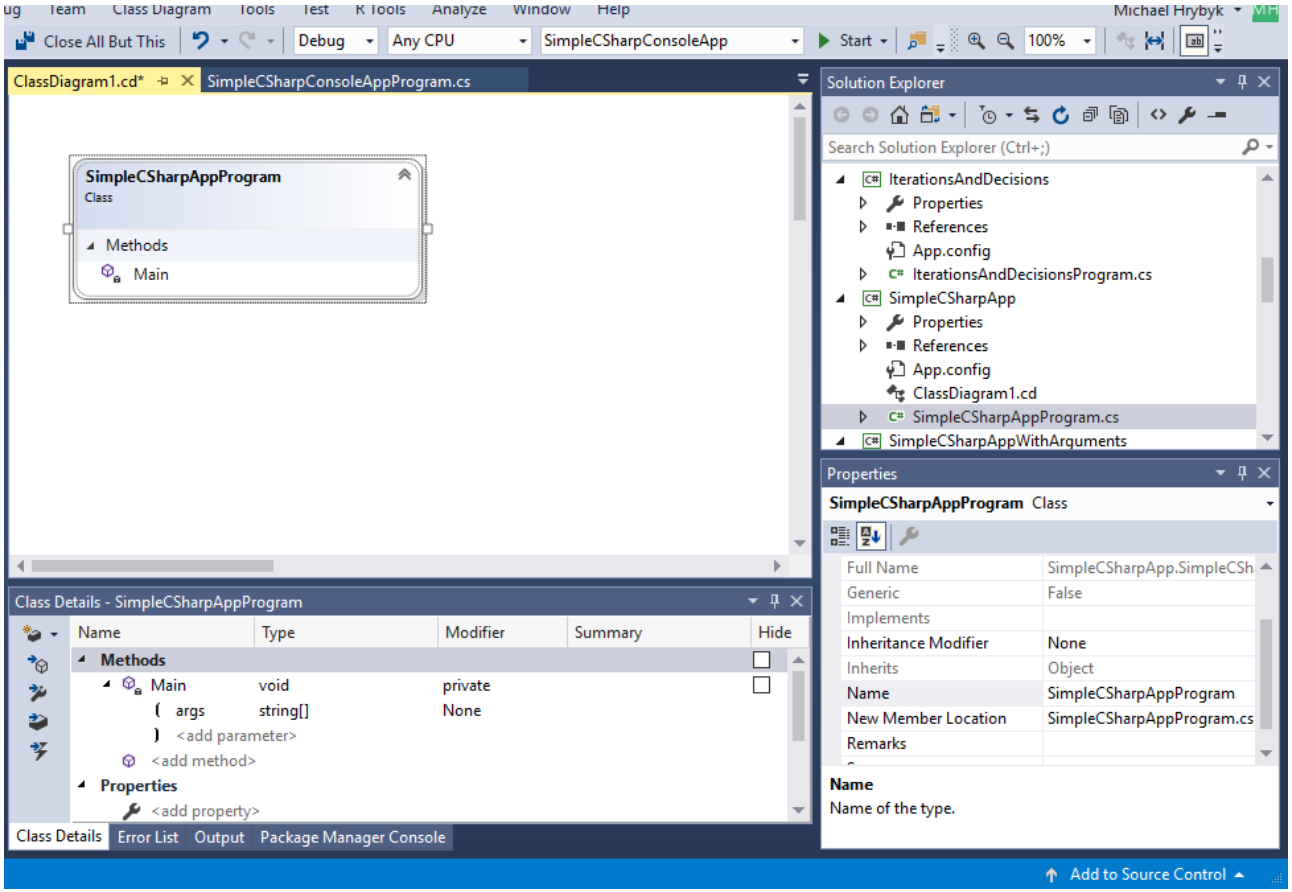
Use of Debug and System.Console

- **using Debug**
  - Can be used to output assertion statements, values, to Output window
  - Separate from actual program output
  - Will be useful when programing Windows Forms
- **using static System.Console**
  - Eliminates having to always write Console.Writeline and other Console methods.
  - Simply use Writeline
  - Using eliminates the need for a fully qualified name

```
using System;
using System.Diagnostics;
using static System.Console; // notice use of this, so we don't have to write
System.Console.WriteLine over and over
using System.Windows.Forms;

namespace BasicConsoleIO
{
    class BasicConsoleIOProgram
    {
        static void Main(string[] args)
        {
            WriteLine("***** Basic Console I/O *****");
            GetUserData();
            ReadLine();
            MessageBox.Show("Waiting for input, look at output window");
            FormatNumericalData();
            ReadLine();
        }

        /// <summary>
        /// Get user's name and age, then display on console
        /// </summary>
        private static void GetUserData()
        {
            // Get name and age.
            Write("Please enter your name: ");
            string userName = ReadLine();
            Write("Please enter your age: ");
            string userAge = ReadLine();

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

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

            // Display something in the output window

            Debug.WriteLine("After output");

            // Restore previous color.
            ForegroundColor = prevColor;

        }
```

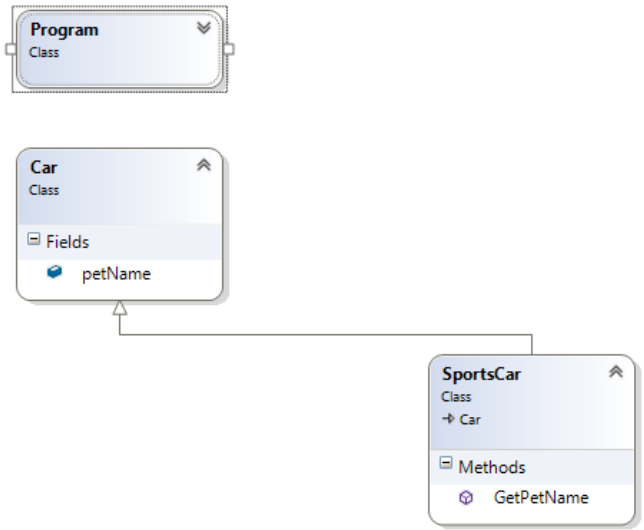
Using Class Diagram

- Can visually add methods, variables, types
- In SimpleCSharpApp, right click on SimpleCSharpAppProgram.cs
  - View Class Diagram



Also used for adding classes

```
using System;

namespace VisualTypeDesignerApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(new SportsCar().GetPetName());
            Console.ReadLine();
        }
    }

    public class Car
    {
        /// <summary>
        /// The name of the car
        /// </summary>
        public string petName;
    }

    public class SportsCar : Car
    {
        /// <summary>
        /// gets the name of the car
        /// </summary>
        public string GetPetName()
        {
            petName = "Old Wreck";
            return petName;
        }
    }
}
```



Rules for Naming Solutions, Projects, Files and Forms

- Solutions/Projects must be named using PascalCase
  - Example: MyFirstProject
  - NOT my FIRST project OR myfirstproject
  - VS will automatically name files using this string
- For a C# or WindowsForms project, VS will create a Program.cs file
  - RENAME THIS to ProjectNameProgram.cs
  - Example: MyFirstProjectProgram.cs
  - VS will ask you if you want to rename identifiers.
    - Answer YES
  - Related to Namespace naming rules (see Microsoft Guidelines).
- For a Windows Forms project there will also be a Form1.cs created by VS
  - RENAME this to MyFirstProjectForm.cs and rename the Form similarly.
  - If you have additional forms, simply use PascalCase for these with the word Form attached.
  - **NOTE: This is contra to control naming convention, and only applies to Forms.**
- Summary:
  - Solutions/Projects: Use PascalCase
  - Program/Form: Use ProjectNameProgram.cs or ProjectNameForm.cs

Naming Conventions

- Objects, Methods, Interfaces, Properties
  - PascalCase –
    - Concatentated descriptive words
    - Each word capitalized
  - Example:
    - MyClass
    - myClass.MyMethod()
- Variables, Instances, Controls
  - camelCase –
    - Concatenated descriptive words
    - first word lower case, all other words capitalized
  - Variables
    - i, n, nDays, maxHours
  - Instances
    - MyClass myHours;
  - Controls
    - textBox1 should be textBox<description
    - textBoxInputHours
- **Do not use 'Hungarian'**

```
class MyClass {
   private int maxHours;
   int GetMaxHours() {
      return(maxHours);
   }
}
…
Class MyClass myHours;
…
employeeMaxHours = myHours.GetMaxHours();
```

Rules for Naming Controls

- A control's name identifies the controls
- The naming conventions are:
  - Must be camelCase
  - The name cannot contain spaces
- VS creates a control with the control name followed by a numeral.
  - For example textBox1, textBox2, etc.
- Rule: backspace over the numeral, and replace it with descriptive text
  - Use camelCase
  - textBox1 becomes textBoxCity or textBoxProvince or whatever you need.
- Examples of good names are:
  - buttonShowDay
  - labelDisplayTotal
  - labelScore

Microsoft Naming Guidelines

- Word Choice
  - ✓ DO choose easily readable identifier names.
    - For example, a property named HorizontalAlignment is more English-readable than AlignmentHorizontal.
  - ✓ DO favor readability over brevity.
    - The property name CanScrollHorizontally is better than ScrollableX (an obscure reference to the X-axis).
  - DO NOT use underscores, hyphens, or any other nonalphanumeric characters.
  - **DO NOT use Hungarian notation.**
  - AVOID using identifiers that conflict with keywords of widely used programming languages.
    - According to Rule 4 of the Common Language Specification (CLS), all compliant languages must provide a mechanism that allows access to named items that use a keyword of that language as an identifier. C#, for example, uses the @ sign as an escape mechanism in this case. However, it is still a good idea to avoid common keywords because it is much more difficult to use a method with the escape sequence than one without it.
- Using Abbreviations and Acronyms
  - DO NOT use abbreviations or contractions as part of identifier names.
    - For example, use GetWindow rather than GetWin.
  - DO NOT use any acronyms that are not widely accepted, and even if they are, only when necessary.

Microsoft Naming Conventions

- We will follow these in this class!
- Guidelines
  - https://msdn.microsoft.com/en-us/library/ms229002(v=vs.110).aspx
- General Naming Conventions
  - https://msdn.microsoft.com/en-us/library/ms229045(v=vs.110).aspx
- Capitalization
  - https://msdn.microsoft.com/en-us/library/ms229043(v=vs.110).aspx

Summary

- .NET is powerful, comprehensive, and cross-platform
- Namespaces are useful for keeping types separate and eliminating name clashes
- Visual Studio is very powerful
  - Managing code and projects
  - Class diagrams
- Naming Conventions
  - MUST BE FOLLOWED