Microsoft® Official Course

Module 12

Creating Reusable Types and Assemblies

Module Overview

- Examining Object Metadata
- Creating and Using Custom Attributes
- Generating Managed Code
- Versioning, Signing, and Deploying Assemblies

- *lab for this module focuses on how to create custom attributes and how to consume these custom attributes by using reflection.

Lesson 1: Examining Object Metadata

- What Is Reflection?
- Loading Assemblies by Using Reflection
- Examining Types by Using Reflection
- Invoking Members by Using Reflection
- Demonstration: Inspecting Assemblies

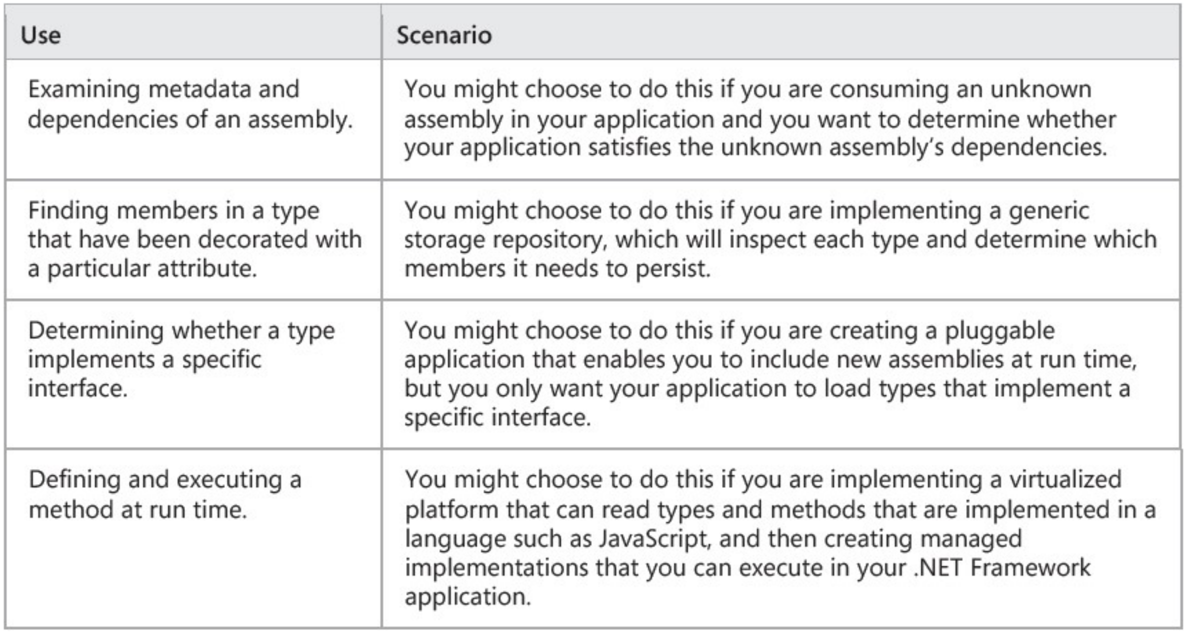
- *when developing case tools to assist in the software development process, features such as reflection enable you to implement anything from code generation platforms to testing frameworks.

What Is Reflection?

- Reflection enables you to inspect and manipulate **assemblies**, **types**, and **type members** at run time
  - E.g. **System.Runtime.Serialization** namespace uses reflection to determine which type members should be serialized when serializing types.

- for each component in an assembly, there is mapping to a class in the **System.Reflection** namespace, for example:
  - An assembly maps to the **Assembly** class.
  - A type maps to the **Type** class.
  - A constructor maps to the **ConstructorInfo** class.

- Reflection Usage Scenarios ➔



Use	Scenario
Examining metadata and dependencies of an assembly.	You might choose to do this if you are consuming an unknown assembly in your application and you want to determine whether your application satisfies the unknown assembly's dependencies.
Finding members in a type that have been decorated with a particular attribute.	You might choose to do this if you are implementing a generic storage repository, which will inspect each type and determine which members it needs to persist.
Determining whether a type implements a specific interface.	You might choose to do this if you are creating a pluggable application that enables you to include new assemblies at run time, but you only want your application to load types that implement a specific interface.
Defining and executing a method at run time.	You might choose to do this if you are implementing a virtualized platform that can read types and methods that are implemented in a language such as JavaScript, and then creating managed implementations that you can execute in your .NET Framework application.

Uses for Reflection C# - skip = extra

- The main value of Reflection is that it can be used to inspect assemblies, types, and members.
  - It's a powerful tool for determining the contents of an unknown assembly or object and can be used in a wide variety of cases

- Use **Module** to get all global and non-global methods defined in the module.
- Use **MethodInfo** to look at information such as parameters, name, return type, access modifiers and implementation details.
- Use **EventInfo** to find out the event-handler data type, the name, declaring type and custom attributes.
- Use **ConstructorInfo** to get data on the parameters, access modifiers, and implementation details of a constructor.
- Use **Assembly** to load modules listed in the assembly manifest.
- Use **PropertyInfo** to get the declaring type, reflected type, data type, name and writable status of a property or to get and set property values.
- Use **CustomAttributeData** to find out information on custom attributes or to review attributes without having to create more instances.

- Sources:
  - https://stackify.com/what-is-c-reflection/
  - https://stackoverflow.com/questions/1458256/why-is-the-use-of-reflection-in-net-recommended

Reflection in the .NET Framework

- The **System.Reflection** namespace contains classes that enable you to take advantage of reflection in your applications:
  - **Assembly:** enables you to load and inspect the metadata and types in a physical assembly
  - **TypeInfo**: enables you to inspect the characteristics of a type.
  - **ParameterInfo**: enables you to inspect the characteristics of any parameters that a member accepts.
  - **ConstructorInfo**: enables you to inspect the constructor of the type
  - **FieldInfo**: enables you to inspect the characteristics of fields that are defined within a type.
  - **MemberInfo**: enables you to inspect members that a type exposes.
  - **PropertyInfo**: enables you to inspect the characteristics of properties that are defined within a type.
  - **MethodInfo**: enables you to inspect the characteristics of the methods that are defined within a type

- The **System** namespace includes the **Type** class, which also exposes a selection of members that you will find useful when you use reflection.
  - For example, the **GetFields** instance method enables you to get a list of **FieldInfo** objects, representing the fields that are defined within a type.

- See also:
  - https://docs.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection
  - https://docs.microsoft.com/en-us/dotnet/api/system.type?redirectedfrom=MSDN&view=netframework-4.8

Loading Assemblies by Using Reflection

- Two ways to load an assembly into your application by using reflection:
  - Reflection-only context: view the metadata that is associated with the assembly and not execute code.
    - if you do try to execute it, the Common Language Runtime (CLR) will throw an **InvalidOperationException** exception
  - Execution context: you can execute the loaded assembly.

- **Assembly.LoadFrom** method - using an absolute file path to the assembly
  - in execution context

```
var assemblyPath = “C:\\FourthCoffee\\Libs\\FourthCoffee.Service.ExceptionHandling.dll";
var assembly = Assembly.LoadFrom(assemblyPath);
```

- **Assembly.ReflectionOnlyLoad** method – from a binary large object (BLOB) that represents the assembly
  - in reflection-only context

```
var assemblyPath = "C:\\FourthCoffee\\Libs\\FourthCoffee.Service.ExceptionHandling.dll";
var rawBytes = File.ReadAllBytes(assemblyPath);
var assembly = Assembly.ReflectionOnlyLoad(rawBytes);
```

- **Assembly.ReflectionOnlyLoadFrom** method - using an absolute file path to the assembly.
  - in reflection-only context

```
var assemblyPath = "C:\\FourthCoffee\\Libs\\FourthCoffee.Service.ExceptionHandling.dll";
var assembly = Assembly.ReflectionOnlyLoadFrom(assemblyPath);
```

Loading Assemblies by Using Reflection

- Some of the instance members that the **Assembly** class provides:
  - **FullName property**: the full name of the assembly, which includes the assembly version and public key token. Example of the full name of the **File** class in the **System.IO** namespace.

    mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

  - **GetReferencedAssemblies method:** a list of all of the names of any assemblies that the loaded assembly references.
  - **GlobalAssemblyCache property:** determine whether the assembly was loaded from the GAC.
  - **Location property:** get the absolute path to the assembly.
  - **ReflectionOnly property:** determine whether the assembly was loaded in a **reflection-only context** or in **an execution context**.
    - If you load an assembly in reflection-only context, you can only examine the code.
  - **GetType method:** an instance of the **Type** class that encapsulates a specific type in an assembly, based on the name of the type.
  - **GetTypes method:** all of the types in an assembly in an array of type **Type**.

- See also: https://docs.microsoft.com/en-us/dotnet/api/system.reflection.assembly?redirectedfrom=MSDN&view=netframework-4.8

Examining Types by Using Reflection

- after you create an **Assembly** object, you can iterate through the assembly and inspect the metadata of each type and each member within a type.

- Get a type by name

```
var assembly = FourthCoffeeServices.GetAssembly();
var type = assembly.GetType(“Full.Name.ClassName");
```

- Get all of the constructors

```
var constructors = type.GetConstructors();
```

- Get all of the fields

```
var fields = type.GetFields();
```

- Get all of the properties

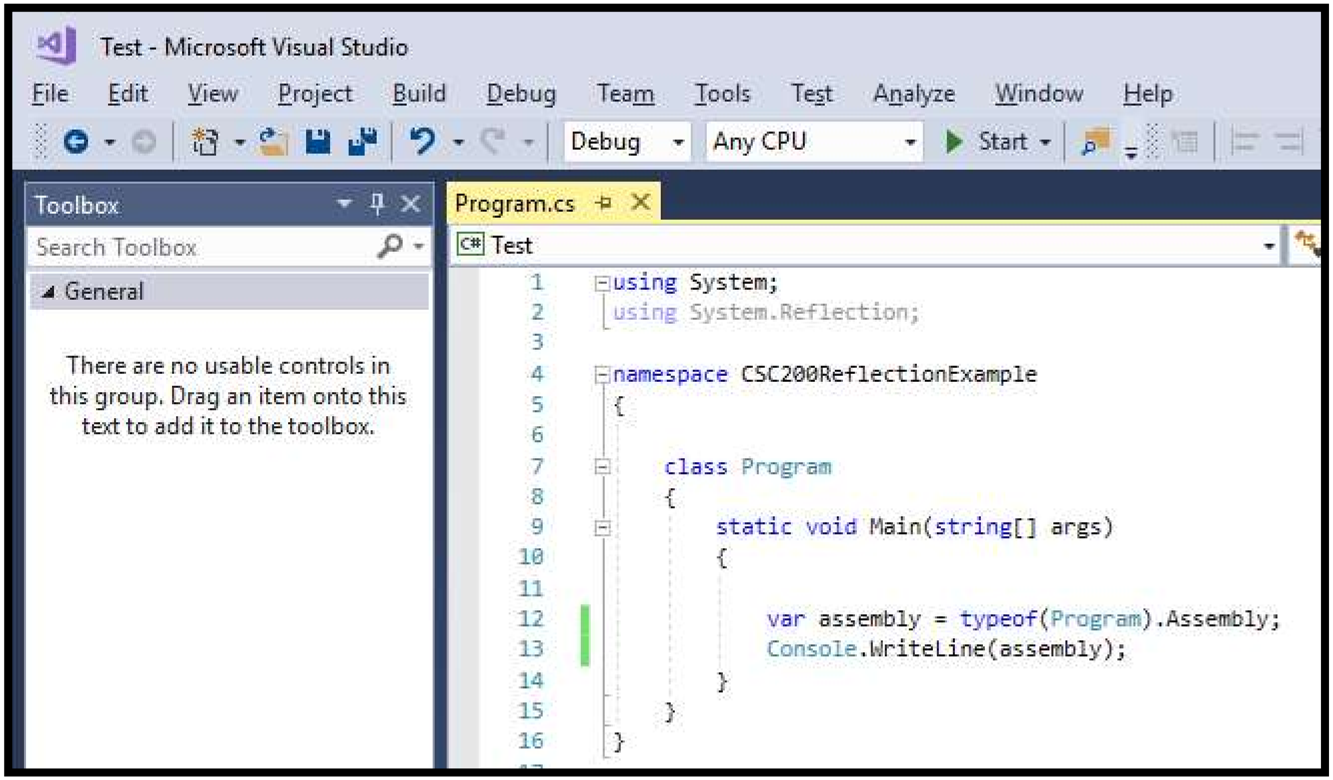
```
var properties = type.GetProperties();
```

- Get all of the methods

```
var methods = type.GetMethods();
```

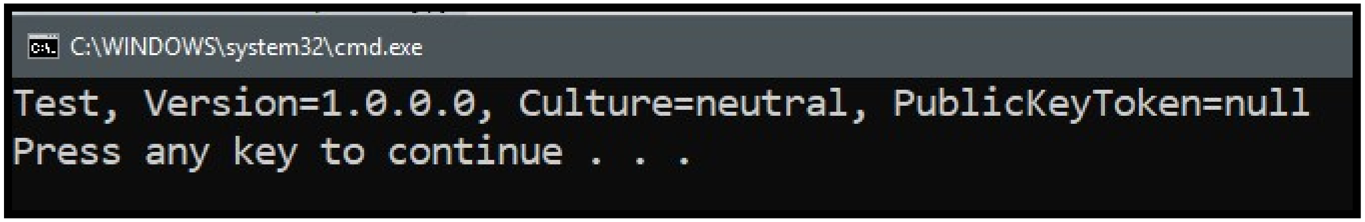
- See also: https://docs.microsoft.com/en-us/dotnet/api/system.type?redirectedfrom=MSDN&view=netframework-4.8

Current assembly



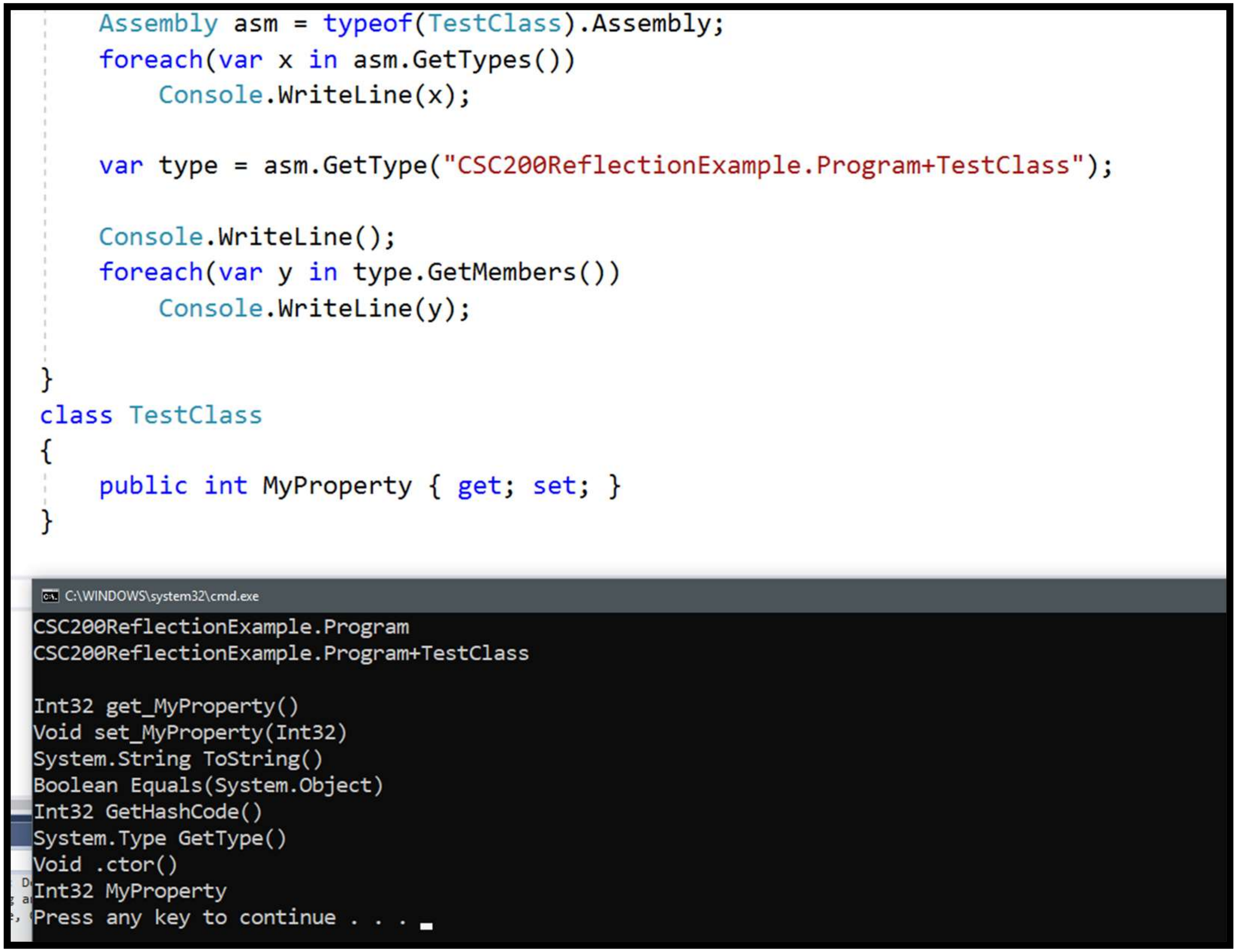
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

```
using System;
using System.Reflection;

namespace CSC200ReflectionExample
{

    class Program
    {
        static void Main(string[] args)
        {

            var assembly = typeof(Program).Assembly;
            Console.WriteLine(assembly);
        }
    }
}
```



```
Test, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
Press any key to continue . . .
```

Using Reflection to see inside the class

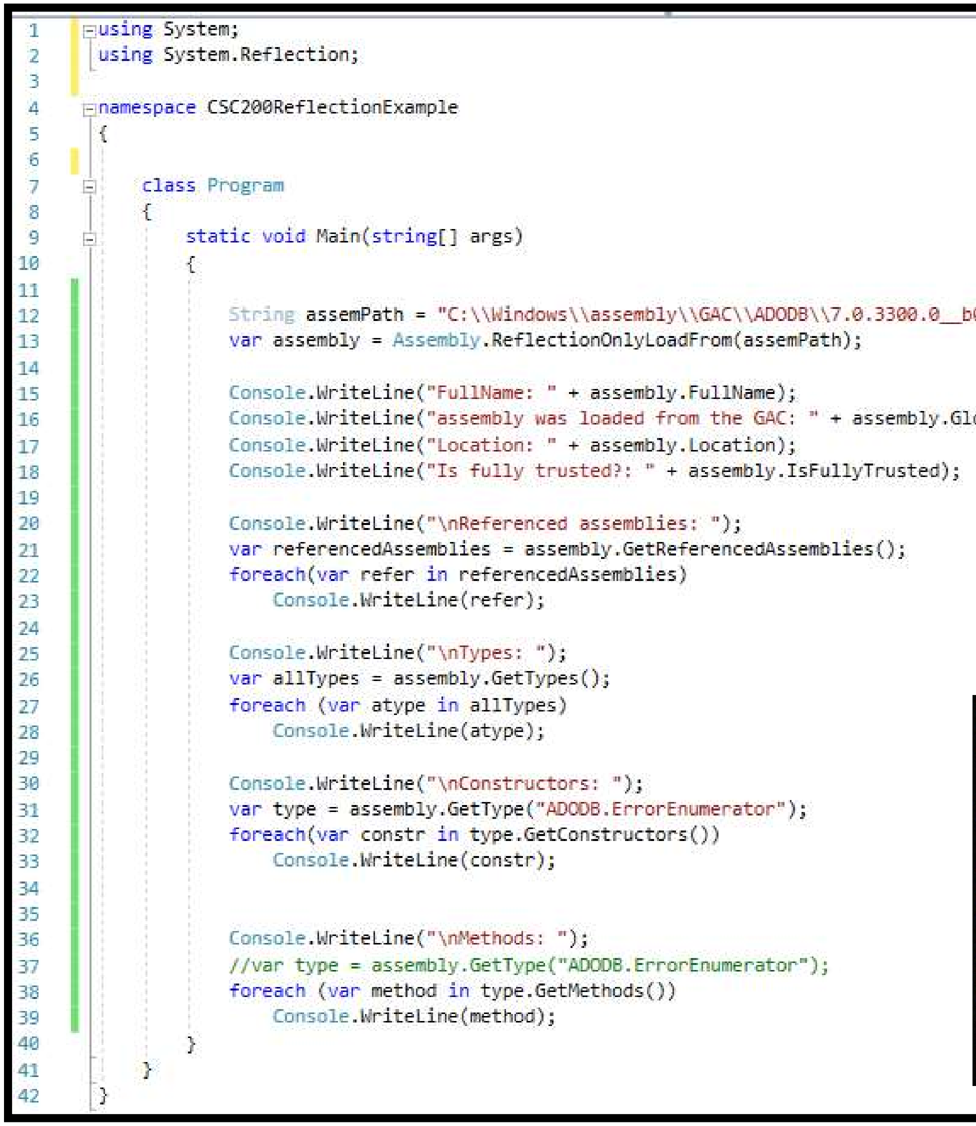


```
    Assembly asm = typeof(TestClass).Assembly;
    foreach(var x in asm.GetTypes())
        Console.WriteLine(x);

    var type = asm.GetType("CSC200ReflectionExample.Program+TestClass");

    Console.WriteLine();
    foreach(var y in type.GetMembers())
        Console.WriteLine(y);

}
class TestClass
{
    public int MyProperty { get; set; }
}
```

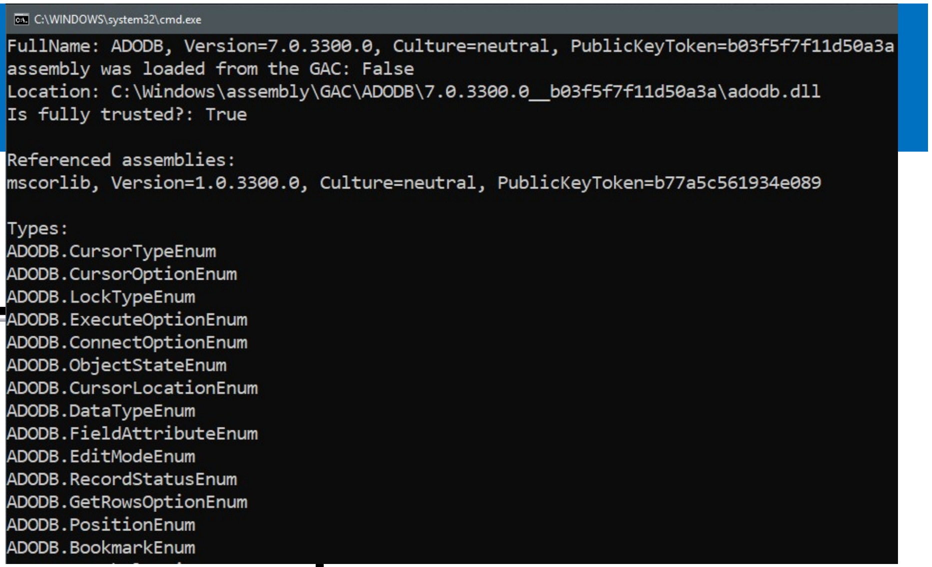
```
CSC200ReflectionExample.Program
CSC200ReflectionExample.Program+TestClass

Int32 get_MyProperty()
Void set_MyProperty(Int32)
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
System.Type GetType()
Void .ctor()
Int32 MyProperty
Press any key to continue . . .
```

Example - skip

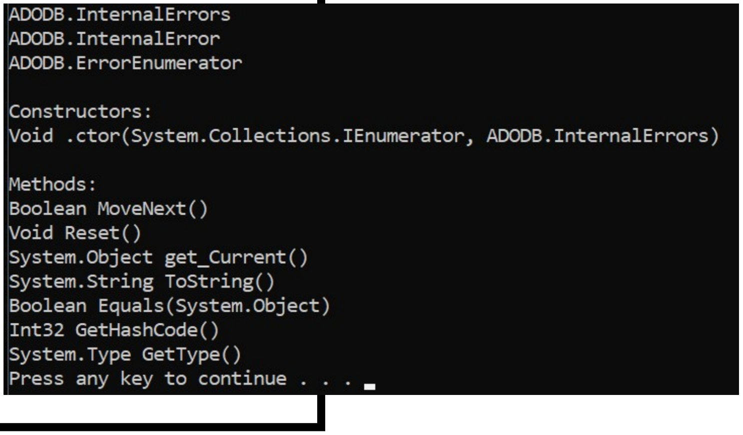
- Here we have lots of assemblies … C:\Windows\assembly
- Pick one and work with it … for example: …
- Let’s use reflection to find more about it:



```csharp
using System;
using System.Reflection;

namespace CSC200ReflectionExample
{

    class Program
    {
        static void Main(string[] args)
        {

            String assemPath = "C:\\Windows\\assembly\\GAC\\ADODB\\7.0.3300.0__b03f5f7f11d50a3a\\adodb.dll";
            var assembly = Assembly.ReflectionOnlyLoadFrom(assemPath);

            Console.WriteLine("FullName: " + assembly.FullName);
            Console.WriteLine("assembly was loaded from the GAC: " + assembly.GlobalAssemblyCache);
            Console.WriteLine("Location: " + assembly.Location);
            Console.WriteLine("Is fully trusted?: " + assembly.IsFullyTrusted);

            Console.WriteLine("\nReferenced assemblies: ");
            var referencedAssemblies = assembly.GetReferencedAssemblies();
            foreach(var refer in referencedAssemblies)
                Console.WriteLine(refer);

            Console.WriteLine("\nTypes: ");
            var allTypes = assembly.GetTypes();
            foreach (var atype in allTypes)
                Console.WriteLine(atype);

            Console.WriteLine("\nConstructors: ");
            var type = assembly.GetType("ADODB.ErrorEnumerator");
            foreach(var constr in type.GetConstructors())
                Console.WriteLine(constr);


            Console.WriteLine("\nMethods: ");
            //var type = assembly.GetType("ADODB.ErrorEnumerator");
            foreach (var method in type.GetMethods())
                Console.WriteLine(method);
        }
    }
}
```

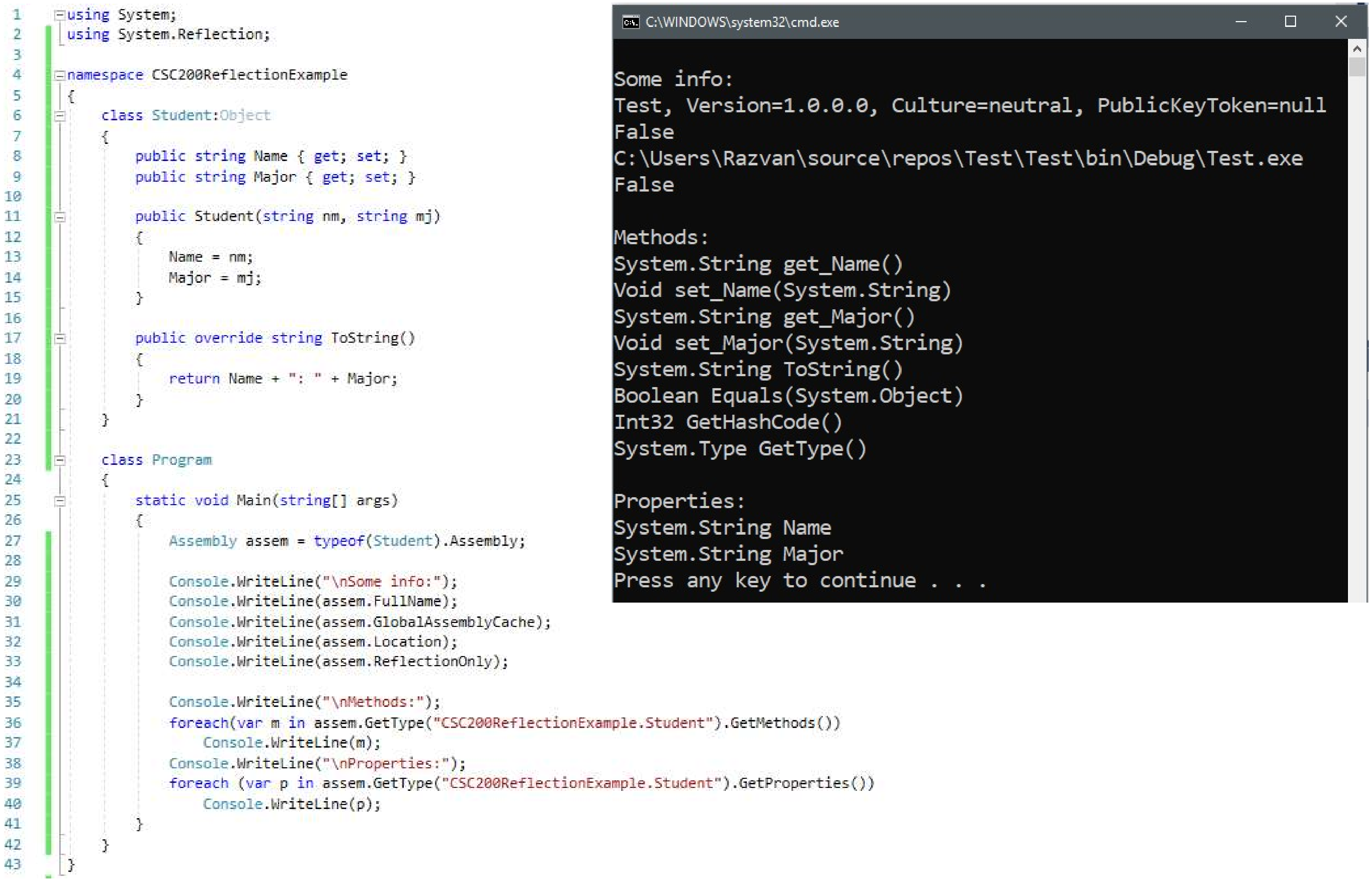


```
FullName: ADODB, Version=7.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
assembly was loaded from the GAC: False
Location: C:\Windows\assembly\GAC\ADODB\7.0.3300.0__b03f5f7f11d50a3a\adodb.dll
Is fully trusted?: True

Referenced assemblies:
mscorlib, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

Types:
ADODB.CursorTypeEnum
ADODB.CursorOptionEnum
ADODB.LockTypeEnum
ADODB.ExecuteOptionEnum
ADODB.ConnectOptionEnum
ADODB.ObjectStateEnum
ADODB.CursorLocationEnum
ADODB.DataTypeEnum
ADODB.FieldAttributeEnum
ADODB.EditModeEnum
ADODB.RecordStatusEnum
ADODB.GetRowsOptionEnum
ADODB.PositionEnum
ADODB.BookmarkEnum
```



```
ADODB.InternalErrors
ADODB.InternalError
ADODB.ErrorEnumerator

Constructors:
Void .ctor(System.Collections.IEnumerator, ADODB.InternalErrors)

Methods:
Boolean MoveNext()
Void Reset()
System.Object get_Current()
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
System.Type GetType()
Press any key to continue . . .
```

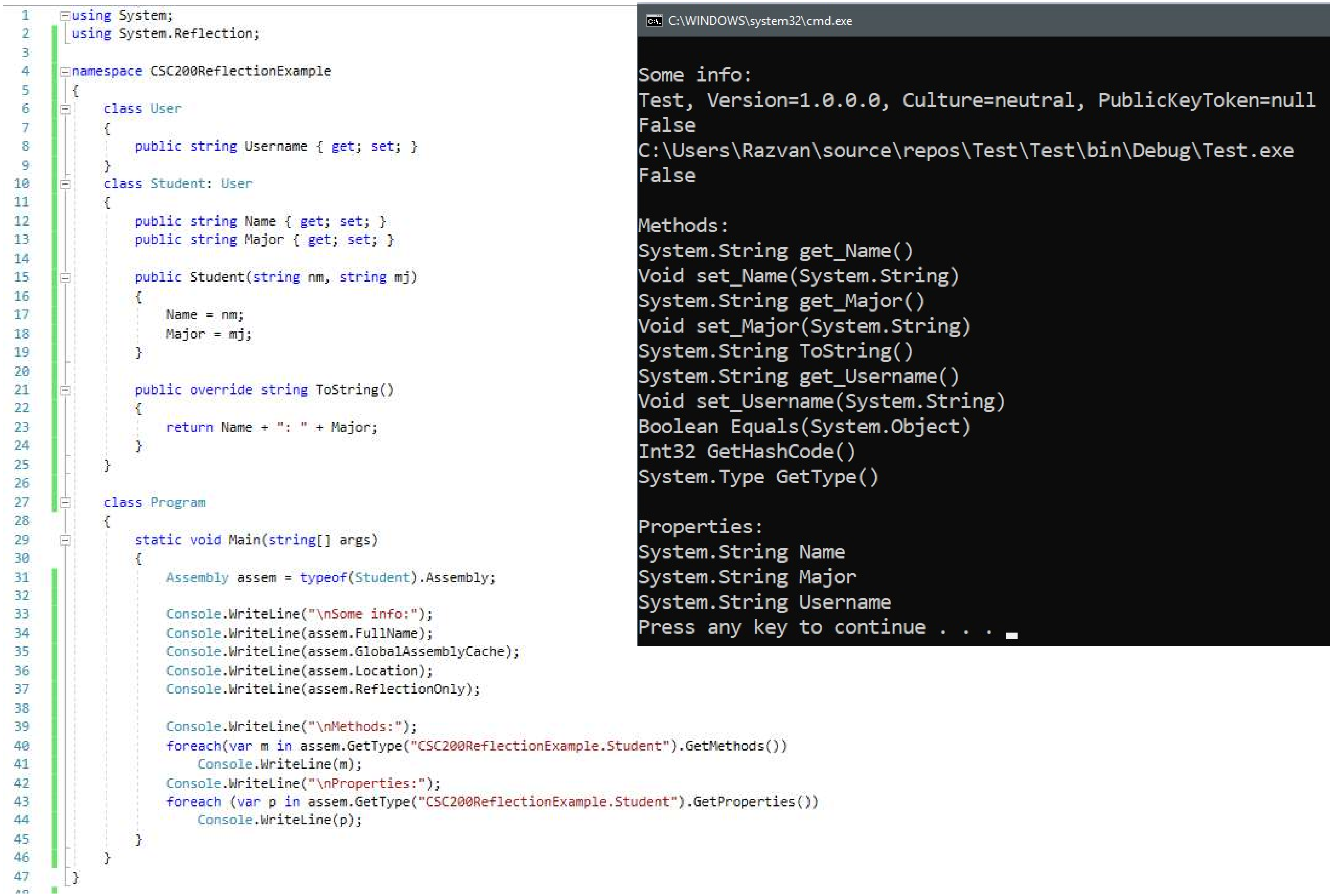
Example - skip



```csharp
using System;
using System.Reflection;

namespace CSC200ReflectionExample
{
    class Student:Object
    {
        public string Name { get; set; }
        public string Major { get; set; }

        public Student(string nm, string mj)
        {
            Name = nm;
            Major = mj;
        }

        public override string ToString()
        {
            return Name + ": " + Major;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Assembly assem = typeof(Student).Assembly;

            Console.WriteLine("\nSome info:");
            Console.WriteLine(assem.FullName);
            Console.WriteLine(assem.GlobalAssemblyCache);
            Console.WriteLine(assem.Location);
            Console.WriteLine(assem.ReflectionOnly);

            Console.WriteLine("\nMethods:");
            foreach(var m in assem.GetType("CSC200ReflectionExample.Student").GetMethods())
                Console.WriteLine(m);
            Console.WriteLine("\nProperties:");
            foreach (var p in assem.GetType("CSC200ReflectionExample.Student").GetProperties())
                Console.WriteLine(p);
        }
    }
}
```

```
Some info:
Test, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
False
C:\Users\Razvan\source\repos\Test\Test\bin\Debug\Test.exe
False

Methods:
System.String get_Name()
Void set_Name(System.String)
System.String get_Major()
Void set_Major(System.String)
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
System.Type GetType()

Properties:
System.String Name
System.String Major
Press any key to continue . . .
```

Example - skip



```csharp
using System;
using System.Reflection;

namespace CSC200ReflectionExample
{
    class User
    {
        public string Username { get; set; }
    }
    class Student: User
    {
        public string Name { get; set; }
        public string Major { get; set; }

        public Student(string nm, string mj)
        {
            Name = nm;
            Major = mj;
        }

        public override string ToString()
        {
            return Name + ": " + Major;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Assembly assem = typeof(Student).Assembly;

            Console.WriteLine("\nSome info:");
            Console.WriteLine(assem.FullName);
            Console.WriteLine(assem.GlobalAssemblyCache);
            Console.WriteLine(assem.Location);
            Console.WriteLine(assem.ReflectionOnly);

            Console.WriteLine("\nMethods:");
            foreach(var m in assem.GetType("CSC200ReflectionExample.Student").GetMethods())
                Console.WriteLine(m);
            Console.WriteLine("\nProperties:");
            foreach (var p in assem.GetType("CSC200ReflectionExample.Student").GetProperties())
                Console.WriteLine(p);
        }
    }
}
```

```
Some info:
Test, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
False
C:\Users\Razvan\source\repos\Test\Test\bin\Debug\Test.exe
False

Methods:
System.String get_Name()
Void set_Name(System.String)
System.String get_Major()
Void set_Major(System.String)
System.String ToString()
System.String get_Username()
Void set_Username(System.String)
Boolean Equals(System.Object)
Int32 GetHashCode()
System.Type GetType()

Properties:
System.String Name
System.String Major
System.String Username
Press any key to continue . . .
```

Invoking Members by Using Reflection

- to invoke an instance method, you must first initialize the type.
  When you invoke static members, there is no need to initialize the object.

- Instantiate a type

```
var type = FourthCoffeeServices.GetHandleErrorType();
...
var constructor = type.GetConstructor(new Type[0]));
...
var initializedObject = constructor.Invoke(new object[0]);
```

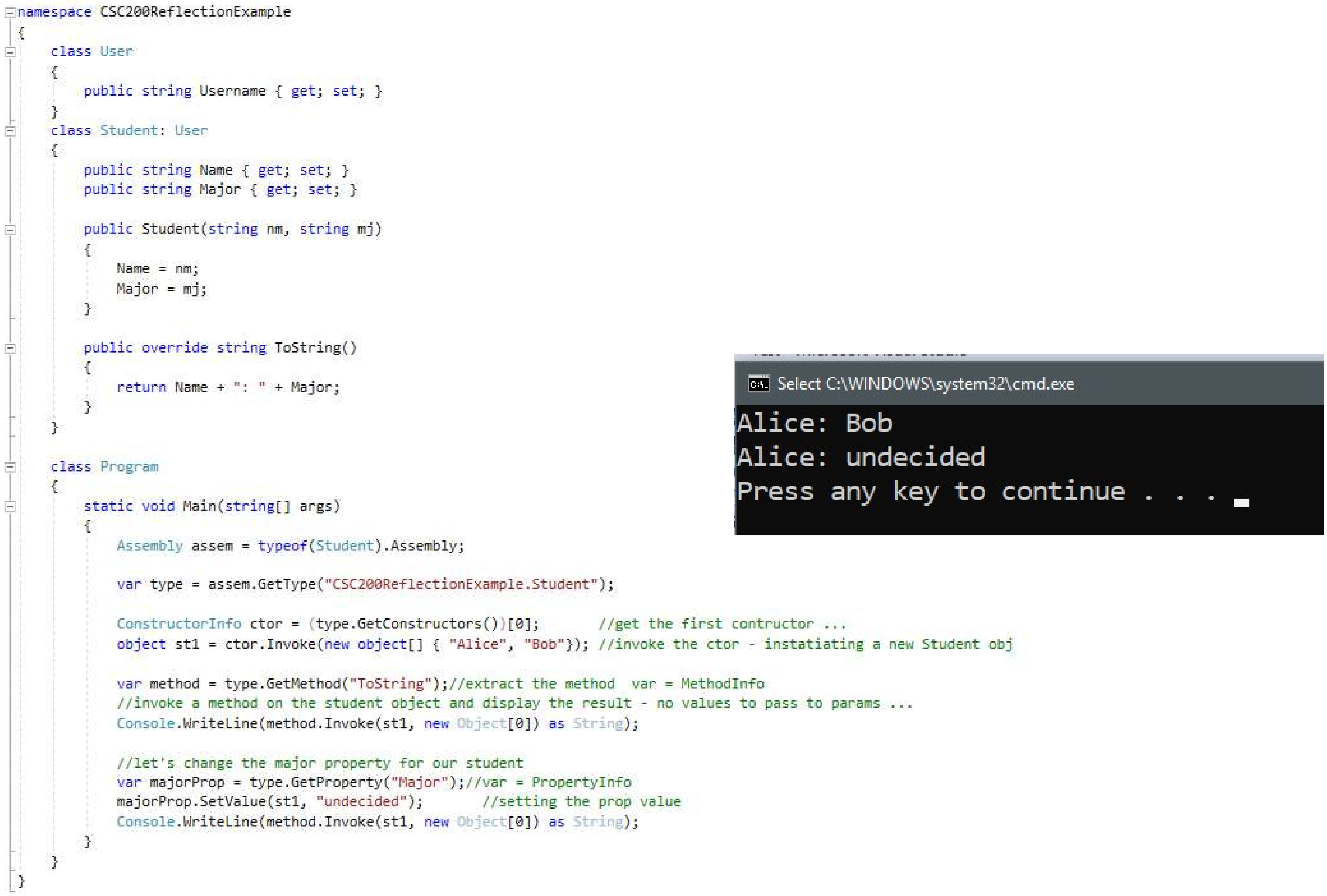
- Invoke methods on the instance

```
var methodToExecute = type.GetMethod("LogError");
var initializedObject = FourthCoffeeServices.InstantiateHandleErrorType();
...
var response = methodToExecute.Invoke(initializedObject,
   new object[] { "Error message“  }) as string;
```

- Get or set property values on the instance

```
var property = type.GetProperty(“LastErrorMessage");
var initializedObject = FourthCoffeeServices.InstantiateHandleErrorType();
...
var lastErrorMessage = property.GetValue(initializedObject) as string;
```

Invoking Members by Using Reflection- example

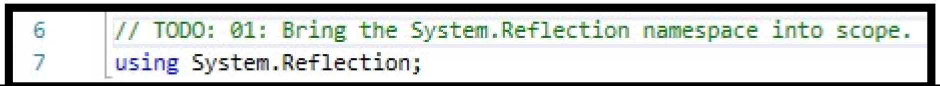


```
namespace CSC200ReflectionExample
{
    class User
    {
        public string Username { get; set; }
    }
    class Student: User
    {
        public string Name { get; set; }
        public string Major { get; set; }

        public Student(string nm, string mj)
        {
            Name = nm;
            Major = mj;
        }

        public override string ToString()
        {
            return Name + ": " + Major;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Assembly assem = typeof(Student).Assembly;

            var type = assem.GetType("CSC200ReflectionExample.Student");

            ConstructorInfo ctor = (type.GetConstructors())[0];       //get the first contructor ...
            object st1 = ctor.Invoke(new object[] { "Alice", "Bob"}); //invoke the ctor - instatiating a new Student obj

            var method = type.GetMethod("ToString");//extract the method  var = MethodInfo
            //invoke a method on the student object and display the result - no values to pass to params ...
            Console.WriteLine(method.Invoke(st1, new Object[0]) as String);

            //let's change the major property for our student
            var majorProp = type.GetProperty("Major");//var = PropertyInfo
            majorProp.SetValue(st1, "undecided");       //setting the prop value
            Console.WriteLine(method.Invoke(st1, new Object[0]) as String);
        }
    }
}
```

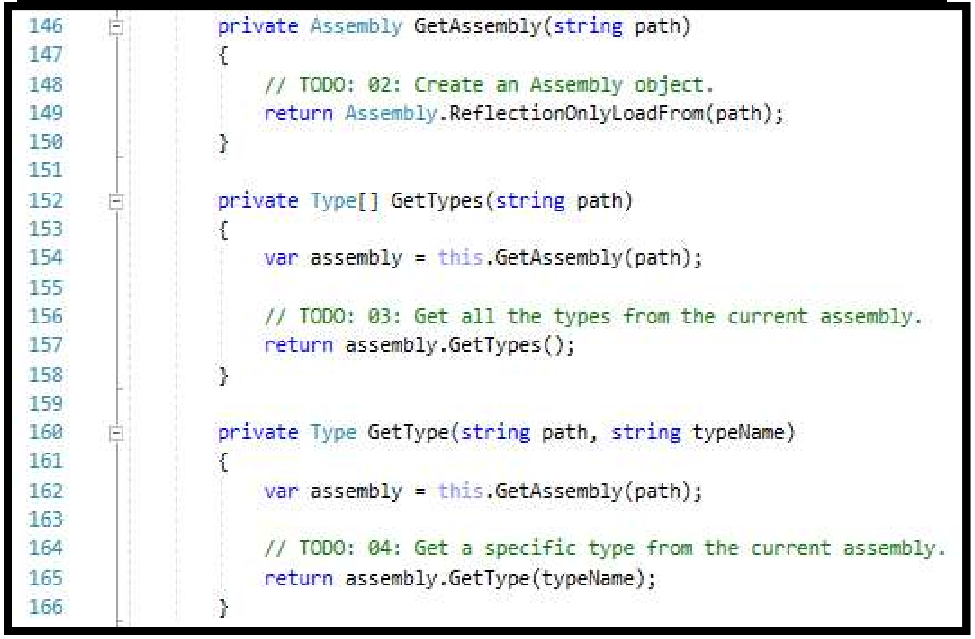
Select C:\WINDOWS\system32\cmd.exe

```
Alice: Bob
Alice: undecided
Press any key to continue . . .
```

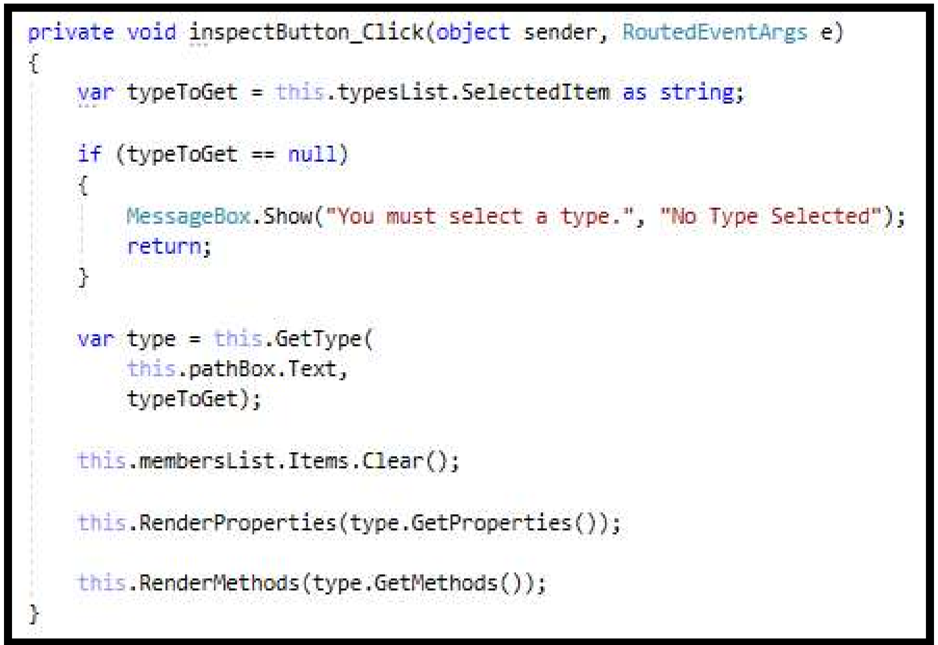
Text Continuation



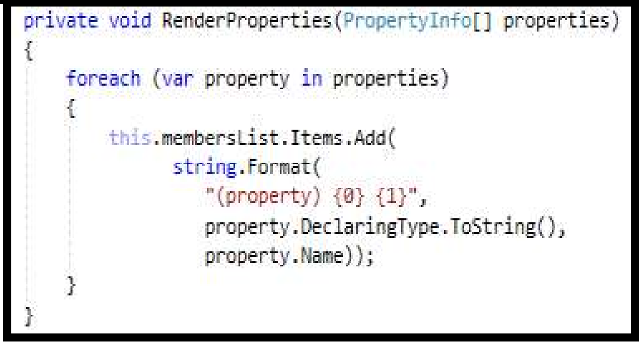
```
// TODO: 01: Bring the System.Reflection namespace into scope.
using System.Reflection;
```



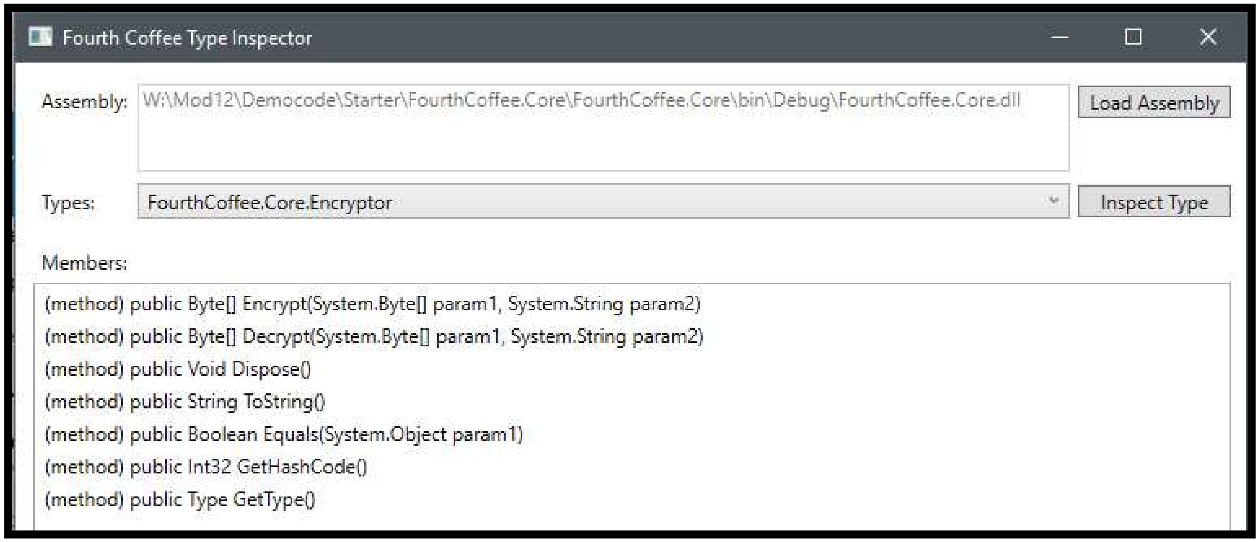
```
        private Assembly GetAssembly(string path)
        {
            // TODO: 02: Create an Assembly object.
            return Assembly.ReflectionOnlyLoadFrom(path);
        }

        private Type[] GetTypes(string path)
        {
            var assembly = this.GetAssembly(path);

            // TODO: 03: Get all the types from the current assembly.
            return assembly.GetTypes();
        }

        private Type GetType(string path, string typeName)
        {
            var assembly = this.GetAssembly(path);

            // TODO: 04: Get a specific type from the current assembly.
            return assembly.GetType(typeName);
        }
```



```
private void inspectButton_Click(object sender, RoutedEventArgs e)
{
    var typeToGet = this.typesList.SelectedItem as string;

    if (typeToGet == null)
    {
        MessageBox.Show("You must select a type.", "No Type Selected");
        return;
    }

    var type = this.GetType(
        this.pathBox.Text,
        typeToGet);

    this.membersList.Items.Clear();

    this.RenderProperties(type.GetProperties());

    this.RenderMethods(type.GetMethods());
}
```



```
private void RenderProperties(PropertyInfo[] properties)
{
    foreach (var property in properties)
    {
        this.membersList.Items.Add(
            string.Format(
                "(property) {0} {1}",
                property.DeclaringType.ToString(),
                property.Name));
    }
}
```



Assembly: W:\Mod12\Democode\Starter\FourthCoffee.Core\FourthCoffee.Core\bin\Debug\FourthCoffee.Core.dll

Types: FourthCoffee.Core.Encryptor

Members:

```
(method) public Byte[] Encrypt(System.Byte[] param1, System.String param2)
(method) public Byte[] Decrypt(System.Byte[] param1, System.String param2)
(method) public Void Dispose()
(method) public String ToString()
(method) public Boolean Equals(System.Object param1)
(method) public Int32 GetHashCode()
(method) public Type GetType()
```

Lesson 2: Creating and Using Custom Attributes

- What Are Attributes?
- Creating and Using Custom Attributes
- Processing Attributes by Using Reflection
- Demonstration: Consuming Custom Attributes by Using Reflection

What Are Attributes?

- Use attributes to provide additional metadata about an element
- Use attributes to alter run-time behavior

```
[DataContract(Name = "SalesPersonContract", IsReference=false)]
public class SalesPerson
{
   [Obsolete("This property will be removed in the next release.")]
   [DataMember]
   public string Name { get; set; }

   ...
}
```

- Some of the attributes that the .NET Framework provides:
  - **Obsolete** (**System** namespace) use to indicate that a type or a type member has been superseded and is only there to ensure backward compatibility.
  - **Serializable** (**System** namespace) can use to indicate that an **IFormatter** implementation can serialize and deserialize a type.
  - **NonSerialized** (**System** namespace) use to indicate that an **IFormatter** implementation should not serialize or deserialize a member in a type.
  - **DataContract** (**System.Runtime.Serialization** namespace) use to indicate that a **DataContractSerializer** object can serialize and deserialize a type.
  - **QueryInterceptor** (**System.Data.Services** namespace), use to control access to an entity in Window Communication Foundation (WCF) Data Services.
  - **ConfigurationProperty** (**System.Configuration** namespace), use to map a property member to a section in an application configuration file.

- All attributes in the .NET Framework derive either directly from the abstract **Attribute** base class in the **System** namespace or from another attribute.

Applying Attributes? - skip

- To use an attribute in your code, perform the following steps:
  - Bring the namespace that contains the attribute you want to use into scope.
  - Apply the attribute to the code element, satisfying any parameters that the constructor expects.
  - Optionally set any of the named parameters that the attribute exposes.

- Information provided by an attribute is also known as metadata.
  - Metadata can be examined at run time by your application to control how your program processes data, or before run time by external tools to control how your application itself is processed or maintained

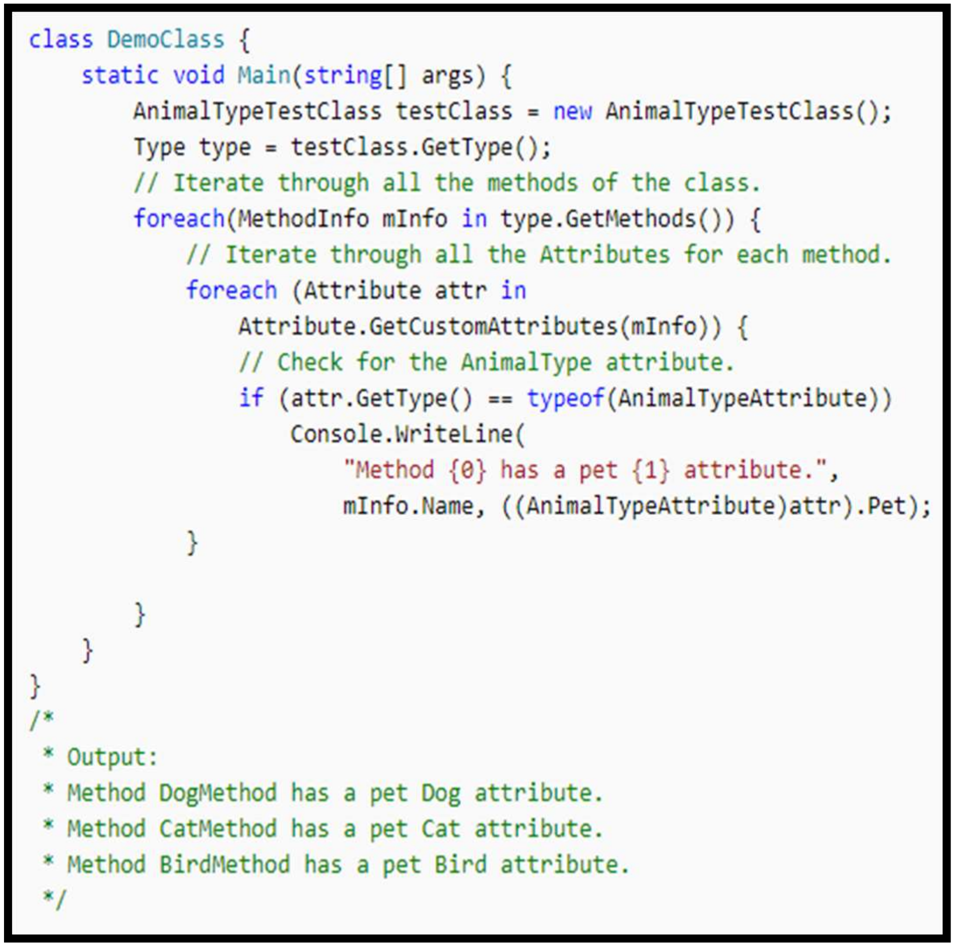
- You can **apply multiple attributes** to a single element to create a hierarchy of metadata that describes the element.

- See also: http://go.microsoft.com/fwlink/?LinkID=267867

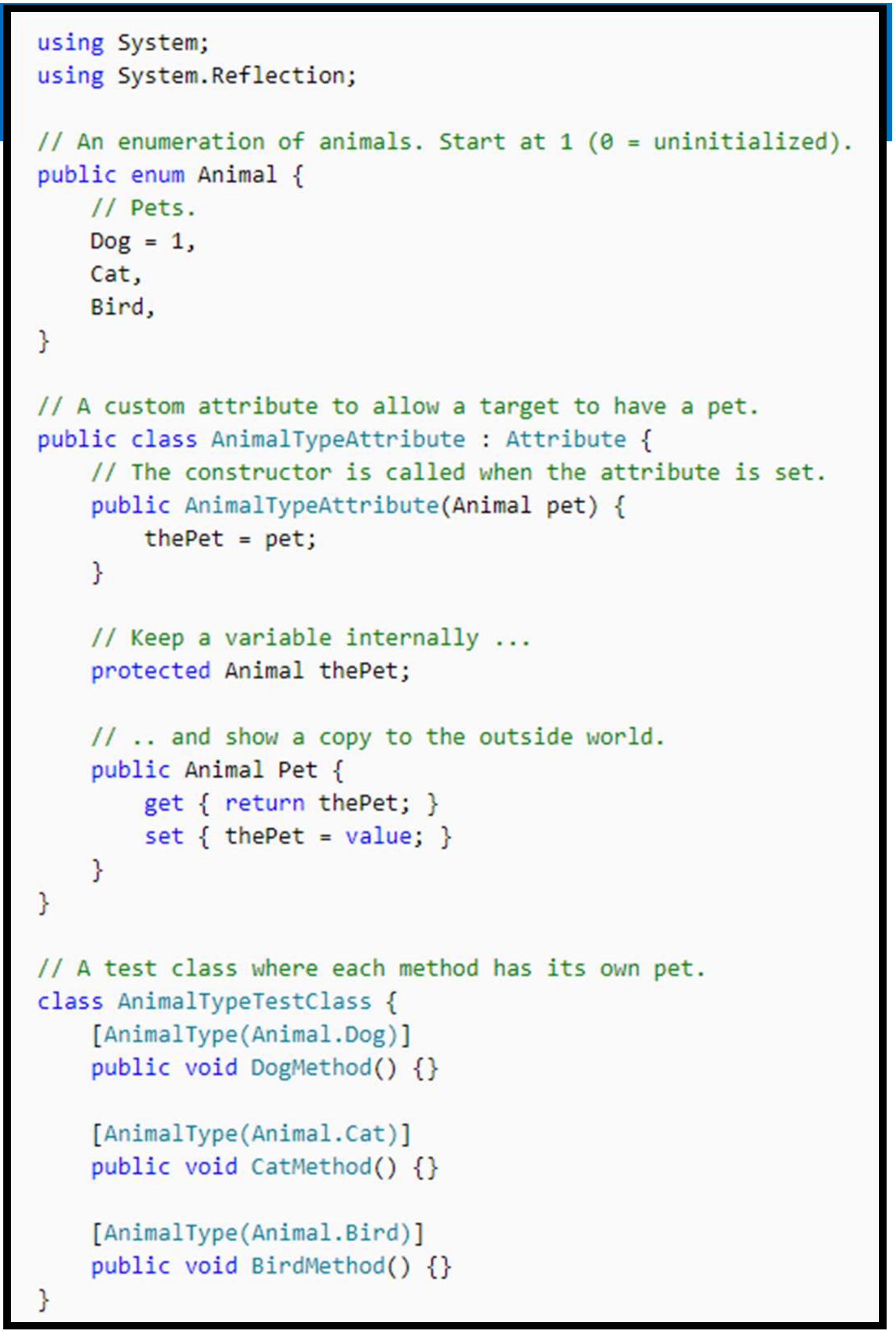
```
[DataContract(Name = "SalesPersonContract", IsReference=false)]
public class SalesPerson
{
   [Obsolete("This property will be removed in the next release.")]
   [DataMember]
   public string Name { get; set; }

   ...
}
```

Creating Attributes? (skip)

- Source: http://go.microsoft.com/fwlink/?LinkID=267867



```
class DemoClass {
    static void Main(string[] args) {
        AnimalTypeTestClass testClass = new AnimalTypeTestClass();
        Type type = testClass.GetType();
        // Iterate through all the methods of the class.
        foreach(MethodInfo mInfo in type.GetMethods()) {
            // Iterate through all the Attributes for each method.
            foreach (Attribute attr in
                Attribute.GetCustomAttributes(mInfo)) {
                // Check for the AnimalType attribute.
                if (attr.GetType() == typeof(AnimalTypeAttribute))
                    Console.WriteLine(
                        "Method {0} has a pet {1} attribute.",
                        mInfo.Name, ((AnimalTypeAttribute)attr).Pet);
            }

        }
    }
}
/*
 * Output:
 * Method DogMethod has a pet Dog attribute.
 * Method CatMethod has a pet Cat attribute.
 * Method BirdMethod has a pet Bird attribute.
 */
```



```
using System;
using System.Reflection;

// An enumeration of animals. Start at 1 (0 = uninitialized).
public enum Animal {
    // Pets.
    Dog = 1,
    Cat,
    Bird,
}

// A custom attribute to allow a target to have a pet.
public class AnimalTypeAttribute : Attribute {
    // The constructor is called when the attribute is set.
    public AnimalTypeAttribute(Animal pet) {
        thePet = pet;
    }

    // Keep a variable internally ...
    protected Animal thePet;

    // .. and show a copy to the outside world.
    public Animal Pet {
        get { return thePet; }
        set { thePet = value; }
    }
}

// A test class where each method has its own pet.
class AnimalTypeTestClass {
    [AnimalType(Animal.Dog)]
    public void DogMethod() {}

    [AnimalType(Animal.Cat)]
    public void CatMethod() {}

    [AnimalType(Animal.Bird)]
    public void BirdMethod() {}
}
```

Creating and Using Custom Attributes

To create a custom attribute:

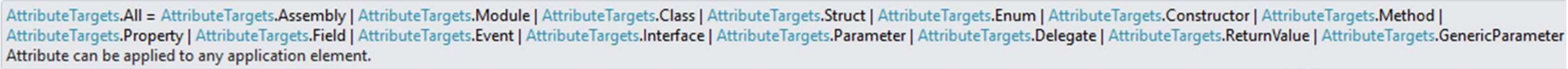
1. Create a class: must derive from the **Attribute** class or another attribute
2. Apply the **AttributeUsage** attribute to your custom attribute class to describe which elements you can apply this attribute to.
3. Define a constructor to initialize the custom attribute
4. Define any properties that you want to enable users of the attribute to optionally provide information.
   - Any properties that you define that have a **get** accessor will be exposed through the attribute as a named parameter.

```
[AttributeUsage(AttributeTargets.All)]
public class DeveloperInfo : Attribute
{
   private string _emailAddress;
   private int _revision;

   public DeveloperInfo(string emailAddress, int revision)
   {
      this._emailAddress = emailAddress;
      this._revision = revision;
   }
}
```

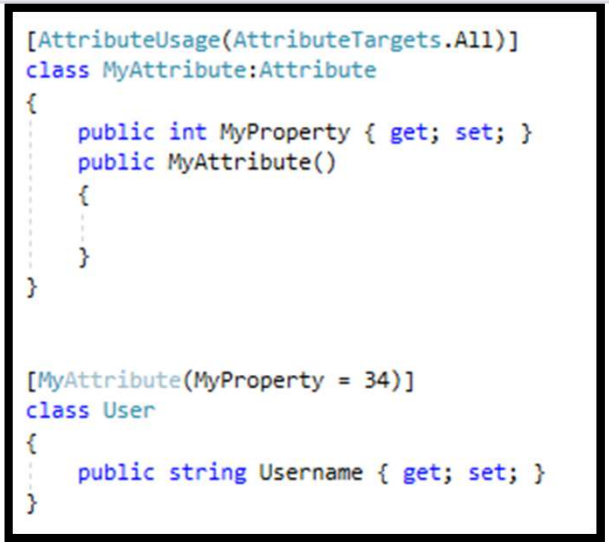
```
[DeveloperInfo("holly@fourthcoffee.com", 3)]
public class SalePerson
{
   ...
}
```

See also: http://go.microsoft.com/fwlink/?LinkID=267868

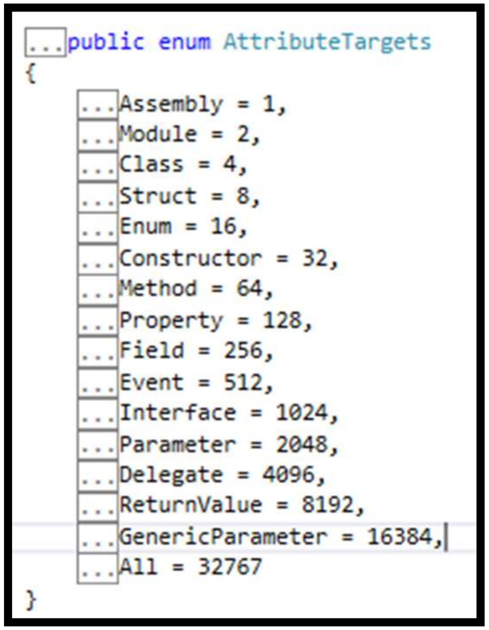
Creating and Using Custom Attributes - SKIP



AttributeTargets.All = AttributeTargets.Assembly | AttributeTargets.Module | AttributeTargets.Class | AttributeTargets.Struct | AttributeTargets.Enum | AttributeTargets.Constructor | AttributeTargets.Method | AttributeTargets.Property | AttributeTargets.Field | AttributeTargets.Event | AttributeTargets.Interface | AttributeTargets.Parameter | AttributeTargets.Delegate | AttributeTargets.ReturnValue | AttributeTargets.GenericParameter
Attribute can be applied to any application element.



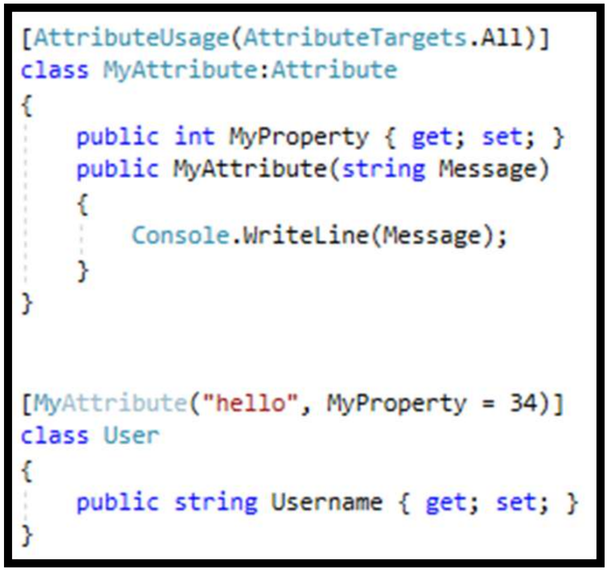
```
[AttributeUsage(AttributeTargets.All)]
class MyAttribute:Attribute
{
    public int MyProperty { get; set; }
    public MyAttribute()
    {

    }
}


[MyAttribute(MyProperty = 34)]
class User
{
    public string Username { get; set; }
}
```



```
...public enum AttributeTargets
{
    ...Assembly = 1,
    ...Module = 2,
    ...Class = 4,
    ...Struct = 8,
    ...Enum = 16,
    ...Constructor = 32,
    ...Method = 64,
    ...Property = 128,
    ...Field = 256,
    ...Event = 512,
    ...Interface = 1024,
    ...Parameter = 2048,
    ...Delegate = 4096,
    ...ReturnValue = 8192,
    ...GenericParameter = 16384,
    ...All = 32767
}
```

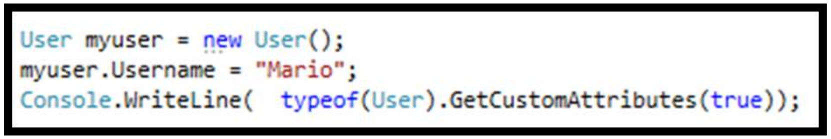
See also:
http://go.microsoft.com/fwlink/?LinkID=267868

https://stackoverflow.com/questions/1168535/when-is-a-custom-attributes-constructor-run



```
[AttributeUsage(AttributeTargets.All)]
class MyAttribute:Attribute
{
    public int MyProperty { get; set; }
    public MyAttribute(string Message)
    {
        Console.WriteLine(Message);
    }
}


[MyAttribute("hello", MyProperty = 34)]
class User
{
    public string Username { get; set; }
}
```

The attribute constructor is run when we start to examine the attribute. Note that the attribute is fetched from the type, not the instance of the type:



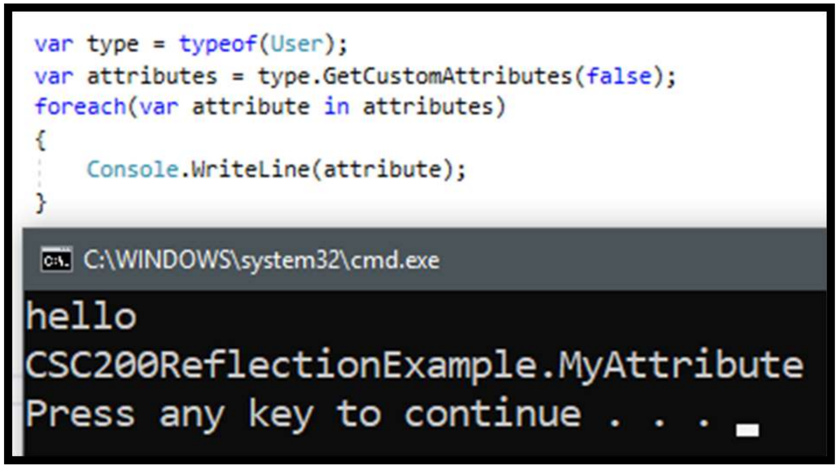
```
User myuser = new User();
myuser.Username = "Mario";
Console.WriteLine(  typeof(User).GetCustomAttributes(true));
```

Processing Attributes by Using Reflection

Use reflection to access the metadata that is encapsulated in custom attributes

- **GetCustomAttribute** method enables you to get a specific attribute that was used on an element.
- **GetCustomAttributes** method enables you to get a list of specific attributes that were used on an element

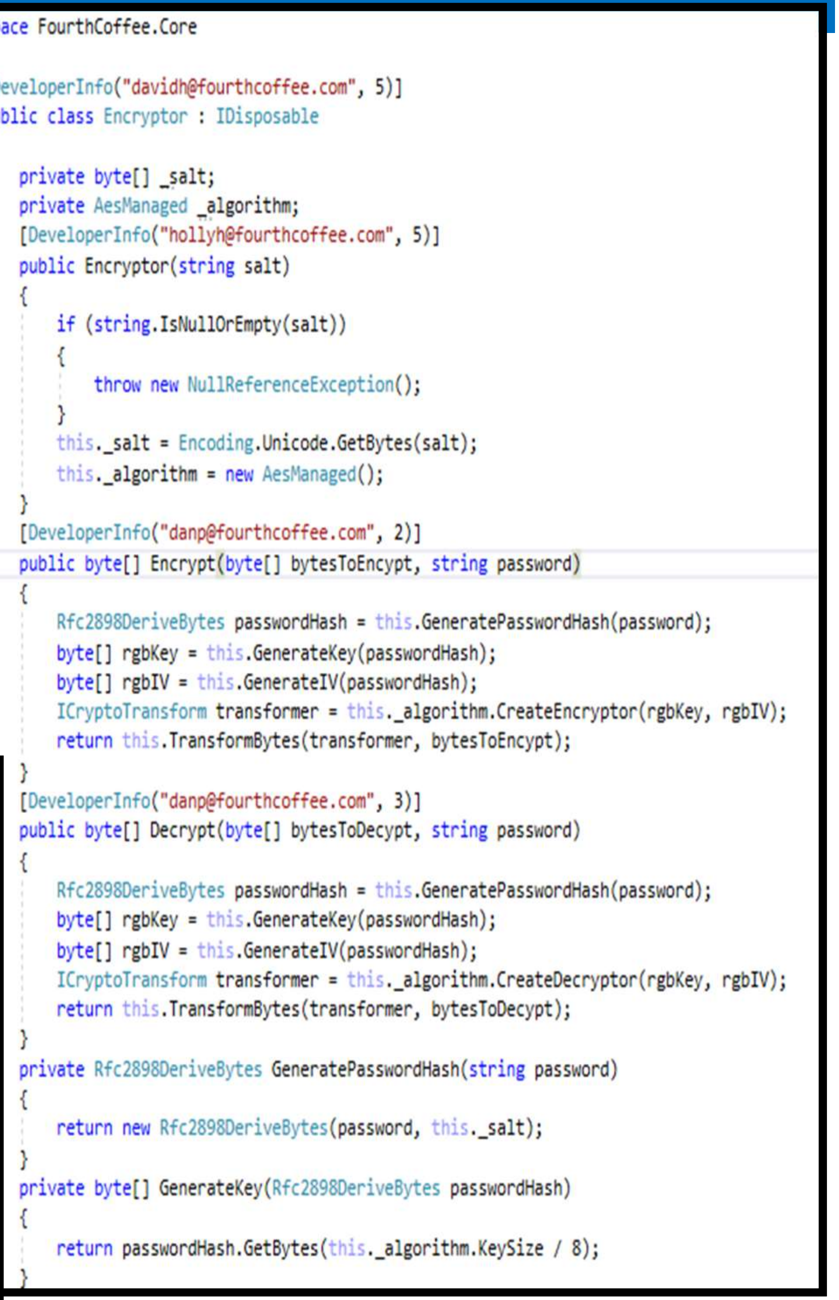
```
var type = FourthCoffee.GetSalesPersonType();

var attributes = type.GetCustomAttributes(typeof(DeveloperInfo), false);

foreach (var attribute in attributes)
{
   var developerEmailAddress = attribute.EmailAddress;
   var codeRevision = attribute.Revision;
}
```



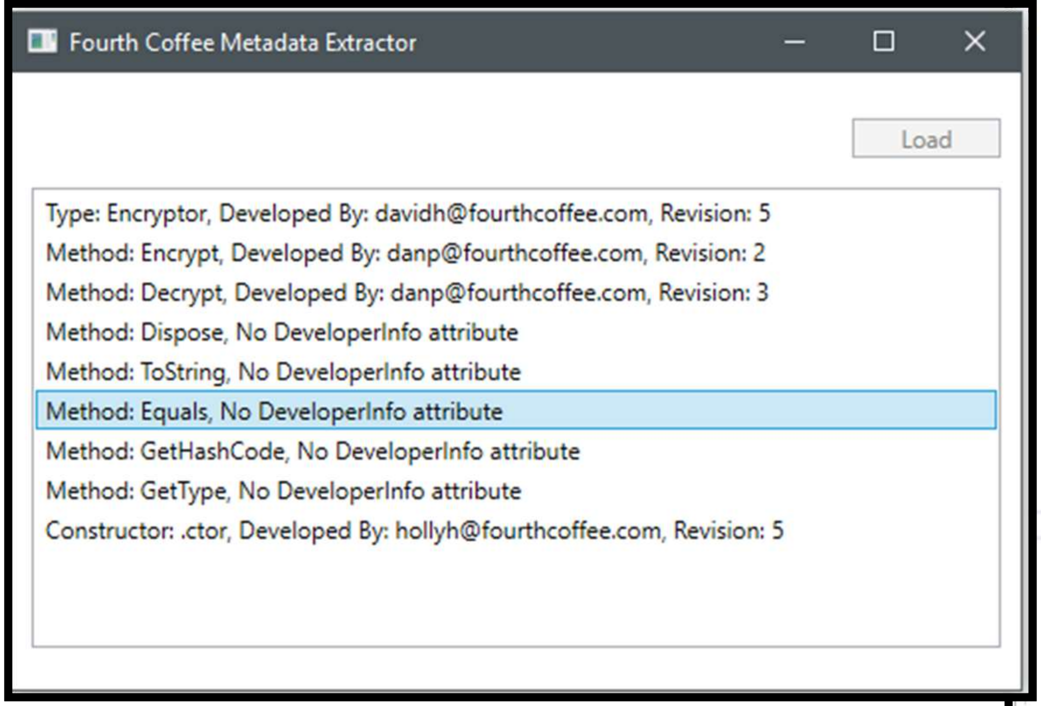
```
var type = typeof(User);
var attributes = type.GetCustomAttributes(false);
foreach(var attribute in attributes)
{
    Console.WriteLine(attribute);
}
```

```
hello
CSC200ReflectionExample.MyAttribute
Press any key to continue . . .
```

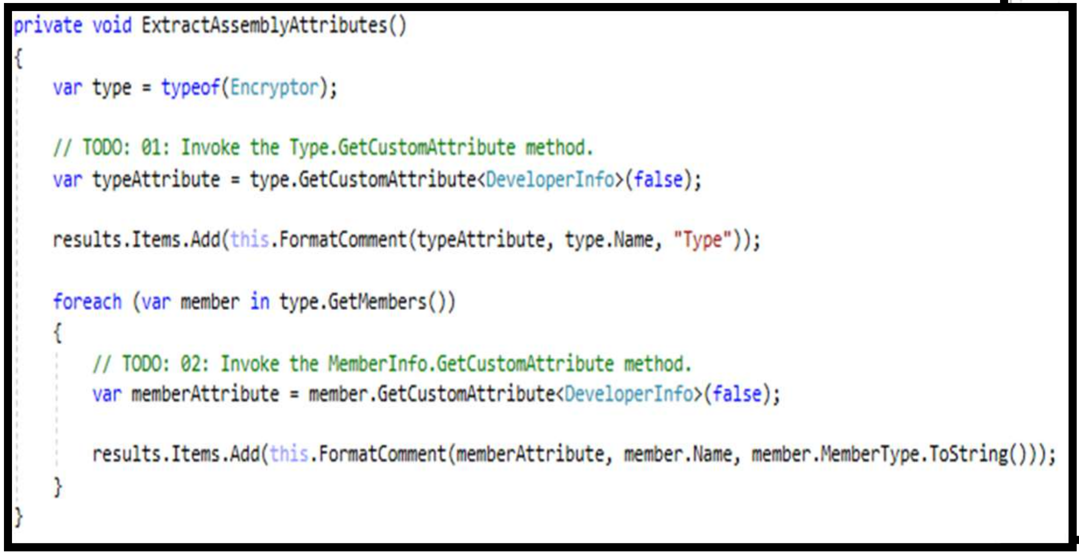
Text Continuation



```
namespace FourthCoffee.Core

    [DeveloperInfo("davidh@fourthcoffee.com", 5)]
    public class Encryptor : IDisposable
    {
        private byte[] _salt;
        private AesManaged _algorithm;
        [DeveloperInfo("hollyh@fourthcoffee.com", 5)]
        public Encryptor(string salt)
        {
            if (string.IsNullOrEmpty(salt))
            {
                throw new NullReferenceException();
            }
            this._salt = Encoding.Unicode.GetBytes(salt);
            this._algorithm = new AesManaged();
        }
        [DeveloperInfo("danp@fourthcoffee.com", 2)]
        public byte[] Encrypt(byte[] bytesToEncypt, string password)
        {
            Rfc2898DeriveBytes passwordHash = this.GeneratePasswordHash(password);
            byte[] rgbKey = this.GenerateKey(passwordHash);
            byte[] rgbIV = this.GenerateIV(passwordHash);
            ICryptoTransform transformer = this._algorithm.CreateEncryptor(rgbKey, rgbIV);
            return this.TransformBytes(transformer, bytesToEncypt);
        }
        [DeveloperInfo("danp@fourthcoffee.com", 3)]
        public byte[] Decrypt(byte[] bytesToDecypt, string password)
        {
            Rfc2898DeriveBytes passwordHash = this.GeneratePasswordHash(password);
            byte[] rgbKey = this.GenerateKey(passwordHash);
            byte[] rgbIV = this.GenerateIV(passwordHash);
            ICryptoTransform transformer = this._algorithm.CreateDecryptor(rgbKey, rgbIV);
            return this.TransformBytes(transformer, bytesToDecypt);
        }
        private Rfc2898DeriveBytes GeneratePasswordHash(string password)
        {
            return new Rfc2898DeriveBytes(password, this._salt);
        }
        private byte[] GenerateKey(Rfc2898DeriveBytes passwordHash)
        {
            return passwordHash.GetBytes(this._algorithm.KeySize / 8);
        }
```



```
Type: Encryptor, Developed By: davidh@fourthcoffee.com, Revision: 5
Method: Encrypt, Developed By: danp@fourthcoffee.com, Revision: 2
Method: Decrypt, Developed By: danp@fourthcoffee.com, Revision: 3
Method: Dispose, No DeveloperInfo attribute
Method: ToString, No DeveloperInfo attribute
Method: Equals, No DeveloperInfo attribute
Method: GetHashCode, No DeveloperInfo attribute
Method: GetType, No DeveloperInfo attribute
Constructor: .ctor, Developed By: hollyh@fourthcoffee.com, Revision: 5
```



```
private void ExtractAssemblyAttributes()
{
    var type = typeof(Encryptor);

    // TODO: 01: Invoke the Type.GetCustomAttribute method.
    var typeAttribute = type.GetCustomAttribute<DeveloperInfo>(false);

    results.Items.Add(this.FormatComment(typeAttribute, type.Name, "Type"));

    foreach (var member in type.GetMembers())
    {
        // TODO: 02: Invoke the MemberInfo.GetCustomAttribute method.
        var memberAttribute = member.GetCustomAttribute<DeveloperInfo>(false);

        results.Items.Add(this.FormatComment(memberAttribute, member.Name, member.MemberType.ToString()));
    }
}
```

Lesson 3: Generating Managed Code

- What Is CodeDOM?
- Defining a Type and Type Members
- Compiling a CodeDOM Model
- Compiling Source Code into an Assembly

What Is CodeDOM? - SKIP

- **CodeDOM** (or Code Document Object Model) is a mechanism provided by the .NET Framework which lets us generate source code in multiple languages using a single model.
  - We create code graphs and use the methods provided for CodeDOM to generate code in a language of our choice. Then we can use dynamic code compilation classes (also provided by CodeDOM) to generate assemblies which can then be loaded and used dynamically.
  - The .NET Framework includes code generators & code compilers for C#, JScript, and Visual Basic.

- Source: https://www.codeproject.com/Articles/18676/Dynamic-Code-Generation-using-CodeDOM

What Is CodeDOM?

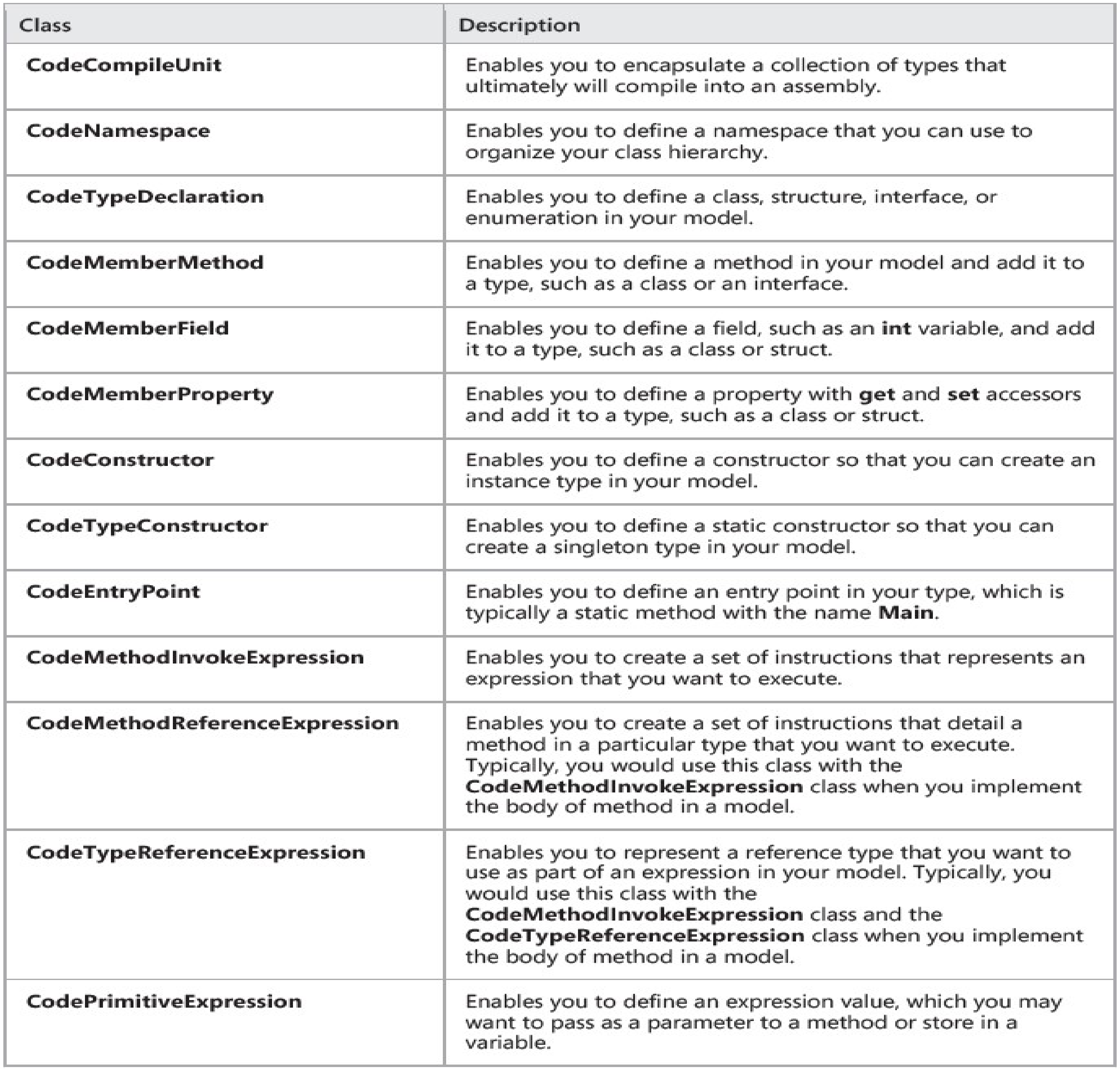
- **CodeDOM** provides the infrastructure for you to model and compile Visual C#, Microsoft JScript, and Microsoft Visual Basic code at run time.
  Some possible uses for CodeDOM:
  - Template generator for source files.
  - Proxy generator for a web service or a database model.

- Define a model that represents your code by using:
  - The **CodeCompileUnit** class
  - The **CodeNamespace** class
  - The **CodeTypeDeclaration** class
  - The **CodeMemberMethod** class

- Generate source code from the model:
  - Visual C# by using the **CSharpCodeProvider** class
  - JScript by using the **JScriptCodeProvider** class
  - Visual Basic by using the **VBCodeProvider** class

- Generate a .dll or a .exe that contains your code

CodeDOM Classes



Class	Description
CodeCompileUnit	Enables you to encapsulate a collection of types that ultimately will compile into an assembly.
CodeNamespace	Enables you to define a namespace that you can use to organize your class hierarchy.
CodeTypeDeclaration	Enables you to define a class, structure, interface, or enumeration in your model.
CodeMemberMethod	Enables you to define a method in your model and add it to a type, such as a class or an interface.
CodeMemberField	Enables you to define a field, such as an **int** variable, and add it to a type, such as a class or struct.
CodeMemberProperty	Enables you to define a property with **get** and **set** accessors and add it to a type, such as a class or struct.
CodeConstructor	Enables you to define a constructor so that you can create an instance type in your model.
CodeTypeConstructor	Enables you to define a static constructor so that you can create a singleton type in your model.
CodeEntryPoint	Enables you to define an entry point in your type, which is typically a static method with the name **Main**.
CodeMethodInvokeExpression	Enables you to create a set of instructions that represents an expression that you want to execute.
CodeMethodReferenceExpression	Enables you to create a set of instructions that detail a method in a particular type that you want to execute. Typically, you would use this class with the **CodeMethodInvokeExpression** class when you implement the body of method in a model.
CodeTypeReferenceExpression	Enables you to represent a reference type that you want to use as part of an expression in your model. Typically, you would use this class with the **CodeMethodInvokeExpression** class and the **CodeTypeReferenceExpression** class when you implement the body of method in a model.
CodePrimitiveExpression	Enables you to define an expression value, which you may want to pass as a parameter to a method or store in a variable.

Defining a Type and Type Members

- Defining a type by using CodeDOM follows the same pattern as defining a type in native Visual C#.
  - The only difference is that when using CodeDOM, you write a set of instructions that a code generator provider will interpret to generate the source code that represents your model.

- Defining a type with a **Main** method

```
var unit = new CodeCompileUnit();                              //object to represent the assembly containing the code

var dynamicNamespace = new CodeNamespace("FourthCoffee.Dynamic");  //define a namespace …
unit.Namespaces.Add(dynamicNamespace);

dynamicNamespace.Imports.Add(new CodeNamespaceImport("System"));   //import namespace System

var programType = new CodeTypeDeclaration("Program");              //create a type named Program
dynamicNamespace.Types.Add(programType);

var mainMethod = new CodeEntryPointMethod();           //represent the static main method in the Program type
programType.Members.Add(mainMethod);

var expression = new CodeMethodInvokeExpression(                   //the body of the Main method
   new CodeTypeReferenceExpression("Console"), "WriteLine",
   new CodePrimitiveExpression("Hello Development Team..!!"));

mainMethod.Statements.Add(expression);                  //adding it to Main()
```

- After you have defined your model, you can then use a code generator provider to compile and generate your code.

Compiling a CodeDOM Model

- compiling and generating an assembly contains the following parts:
  - 1. Compiling the model and generating source code files for each type.
  - 2. Generating an assembly that contains the necessary references and the types that are defined in the source code files.
- Note: you do not have to generate files that contain the source code before you can generate the assembly. You can do it all in memory

Compiling a CodeDOM Model into Source Code

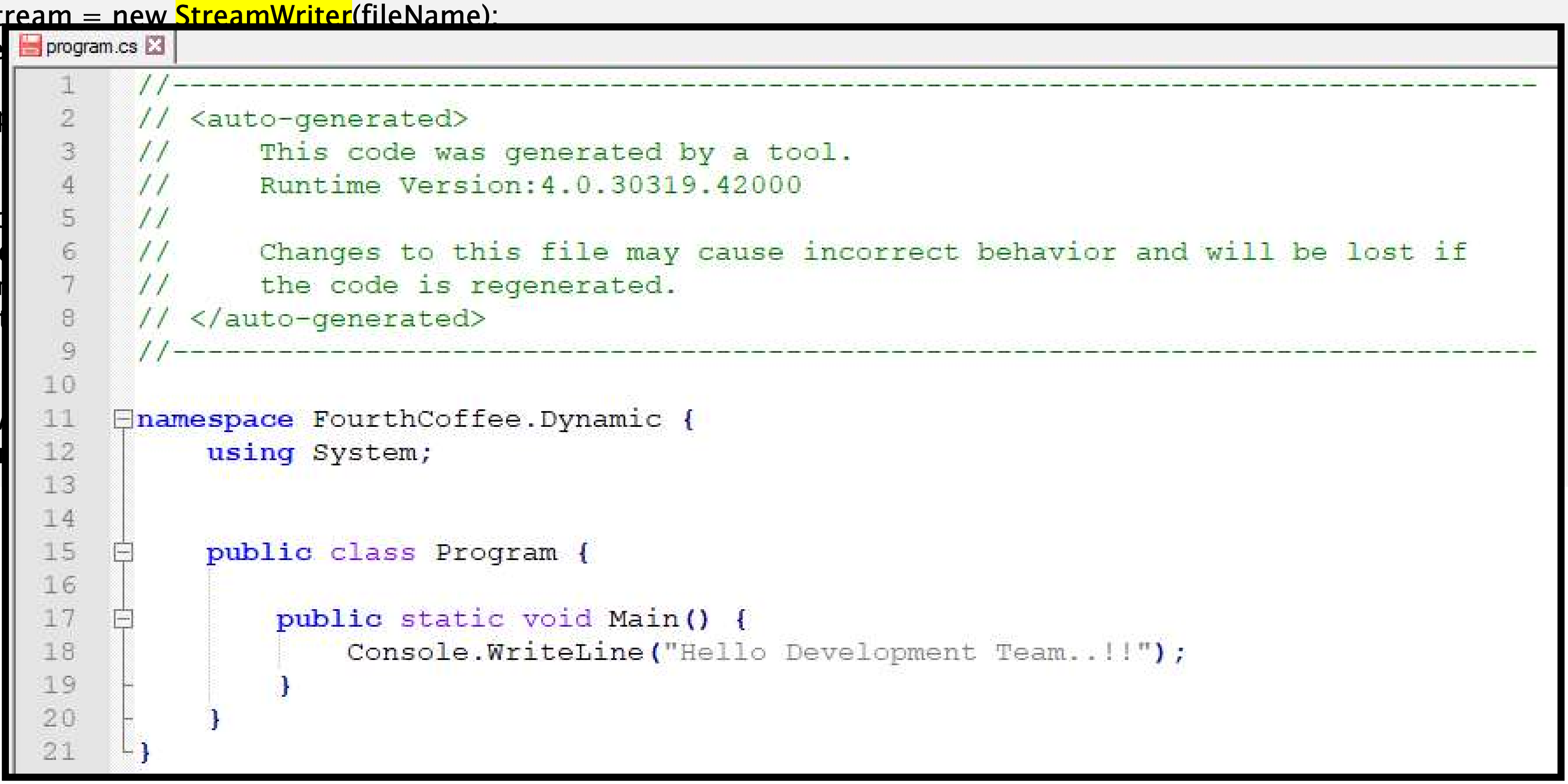
- Compiling a Model into a Source Code File

```
var provider = new CSharpCodeProvider();   //1. Create an instance of the code generator provider you want to use

var fileName = "program.cs";                  //2. Create a StreamWriter object use to write compiled code to a file
var stream = new StreamWriter(fileName);
var textWriter = new IndentedTextWriter(stream); //3. will write the indented source code to a file.

var options = new CodeGeneratorOptions(); //4. object that encapsulates your code generation settings.
options.BlankLinesBetweenMembers = true;

var compileUnit = FourthCoffee.GetModel(); //5. generate the source code
provider.GenerateCodeFromCompileUnit(
   compileUnit,                                  // use the unit generated on the previous slide ...
   textWriter,
   options);

textWriter.Close();                              //6. Close the IndentedTextWriter and StreamWriter objects
stream.Close();
```

Compiling a CodeDOM Model into Source Code

- Compiling a Model into a Source Code File

```
var provider = new CSharpCodeProvider();   //1. Create an instance of the code generator provider you want to use

var fileName = "program.cs";                //2. Create a StreamWriter object use to write compiled code to a file
var stream = new StreamWriter(fileName);
var te

var op
optio

var co
provi
   con
   text
   opt

textW
strea
```



```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace FourthCoffee.Dynamic {
    using System;


    public class Program {

        public static void Main() {
            Console.WriteLine("Hello Development Team..!!");
        }
    }
}
```

Compiling Source Code into an Assembly

- Generate an assembly from your source code files

```
var provider = new CSharpCodeProvider();          //Create an instance of the code generator provider you want to use

var compilerSettings = new CompilerParameters(); //object that you will use to define the settings for the compiler
compilerSettings.ReferencedAssemblies.Add("System.dll");
compilerSettings.GenerateExecutable = true;                         //whether to generate a .dll or an .exe file
compilerSettings.OutputAssembly = "FourthCoffee.exe";

var sourceCodeFileName = "program.cs";
var compilationResults = provider.CompileAssemblyFromFile(          //generate the assembly.
   compilerSettings,
   sourceCodeFileName);

var buildFailed = false;
foreach (var error in compilationResults.Errors)
{
   var errorMessage = error.ToString();
   buildFailed = true;
}
```

- Note: **CompileAssemblyFromFile** method also accepts an array of source file names, so you can compile several source code files into a single assembly.
- See also: http://go.microsoft.com/fwlink/?LinkID=267872

Lesson 4: Versioning, Signing, and Deploying Assemblies

- What Is an Assembly?
- What Is the GAC?
- Signing Assemblies
- Versioning Assemblies
- Installing an Assembly into the GAC
- Demonstration: Signing and Installing an Assembly into the GAC
- Demonstration: Specifying the Data to Include in the Grades Report Lab

*tasks that you should perform after the code development of your application is complete. Typically, these tasks should form part of the build process for your applications.

What Is an Assembly?

- An assembly is a collection of types & resources that form a unit of functionality
  - An assembly might consist of a single portable executable (PE) file, such as an executable (.exe) program or dynamic link library (.dll) file, or
    it might consist of multiple PE files and external resource files, such as bitmaps or data files.

- An assembly is the building block of a .NET Framework application because an application consists of one or more assemblies.

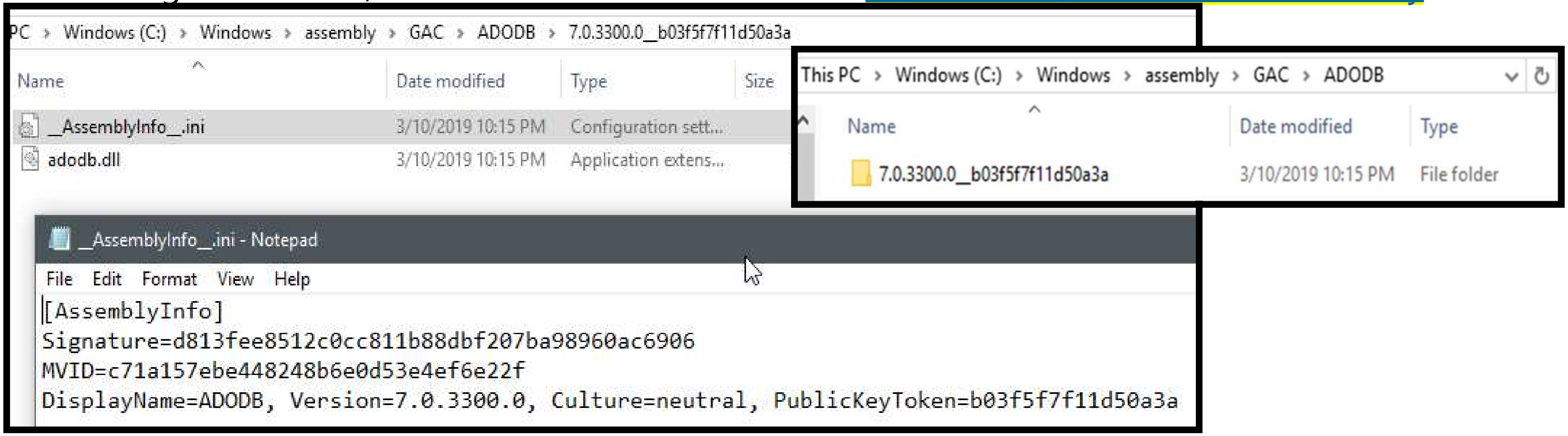
- An assembly can contain:
  - Intermediate Language (IL) code: set of instructions that the just-in-time (JIT) compiler then translates to CPU-specific code before the application runs.
  - Resources: include images & assembly metadata (in the form of assembly manifest).
  - Type metadata: information about available classes, interfaces, methods, and properties
  - Assembly Manifest: provides information about the assembly such as the title, the description, and version information.
    - also contains information about links to the other files in the assembly.

- See also: http://go.microsoft.com/fwlink/?LinkID=267873

What Is the GAC (global assembly cache)?

- When you create an assembly, by default you create a private assembly that a single application can use.
  - If you need to create an assembly that multiple applications can share, you should give the assembly a strong name and install the assembly into the GAC.
  - A strong name is a unique name for an assembly that consists of the assembly's name, version number, culture information (if applicable), & a digital signature that contains a public & private key.

- The GAC provide a robust solution to share assemblies between multiple application on the same machine
  - Find the contents of the GAC at C:\Windows\assembly
  - Starting with .NET 4, the default location for GAC is: C:\Windows\Microsoft.net\assembly



PC › Windows (C:) › Windows › assembly › GAC › ADODB › 7.0.3300.0__b03f5f7f11d50a3a

Name	Date modified	Type	Size
__AssemblyInfo__.ini	3/10/2019 10:15 PM	Configuration sett...	
adodb.dll	3/10/2019 10:15 PM	Application extens...	

This PC › Windows (C:) › Windows › assembly › GAC › ADODB

Name	Date modified	Type
7.0.3300.0__b03f5f7f11d50a3a	3/10/2019 10:15 PM	File folder

__AssemblyInfo__.ini - Notepad

```
[AssemblyInfo]
Signature=d813fee8512c0cc811b88dbf207ba98960ac6906
MVID=c71a157ebe448248b6e0d53e4ef6e22f
DisplayName=ADODB, Version=7.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
```

What Is the GAC (global assembly cache)?

Benefits of using the GAC:

- Side-by-side deployment:
  - different versions of an assembly in the GAC do not affect each other
- Improved loading time:
  - When you install an assembly in the GAC, it undergoes strong-name validation, which ensures that the digital signature is valid. The process occurs at installation time, so assemblies in the GAC load faster at run time than assemblies that are not installed in the GAC.
- Reduced memory consumption:
  - If multiple applications reference an assembly, the operating system loads only one instance of the assembly,
- Improved search time:
  - the runtime checks the GAC for a referenced assembly before it checks other locations.
- Improved maintainability:
  - a single file that multiple applications share

Signing Assemblies

- When you sign an assembly, you give the assembly a strong name
  - A strong name provides an assembly with a globally unique name
  - A strong name requires two cryptographic keys, a public key and a private key, known as a key pair. The compiler uses the key pair at build time to create the strong name.
  - The strong name consists of the simple text name of the assembly, the version number, optional culture information, the public key, and a digital signature.

- Sign an assembly:
  - Create a key file

```
sn –k FourthCoffeeKeyFile.snk
```

    - In the Visual Studio Command Prompt window, use the Strong Name (Sn.exe) tool
  - Associate the key file with an assembly [signing your assembly]
    - use the **Signing** tab in the project properties pane
    - then Visual Studio adds the **AssemblyKeyFileAttribute** attribute to the **AssemblyInfo** class.

```
[assembly: AssemblyKeyFileAttribute("FourthCoffeeKeyFile.snk")]
```

Signing Assemblies

- When you sign an assembly, you might not have access to a private key.
    - For example, for security reasons, some organizations restrict access to their private key to just a few individuals.
    - The public key will generally be available because it is publicly accessible.
    - In this situation, you can use delayed signing at build time.
    - You provide the public key and reserve space in the PE file for the strong name signature.
    - However, you defer the addition of the private key until a later stage, typically just before the assembly ships.

- Delay the signing of an assembly:
    1. Open the **properties** for the project
    2. Click the **Signing** tab
    3. Select the **Sign** the assembly check box
    4. Specify a **key** file
    5. Select the **Delay sign only** check box
        1. Later … re-sign using the –R option …

```
sn –R FourthCoffee.Core.dll sgKey.snk
```

- Note: You cannot run or debug a delay-signed project.
    - You can, however, use the Sn.exe tool with the –Vr option to skip verification

Versioning Assemblies

- it is important to version assemblies so you can keep track of which version of your application users are using.
  - Without a version number, debugging and reproducing production issues are difficult.
  - All assemblies are given a version number by Visual Studio, which is typically 1.0.0.0.
  - It is the responsibility of the developer to increment the assembly’s version number.
  - By default, applications only run with the version of an assembly with which they were built

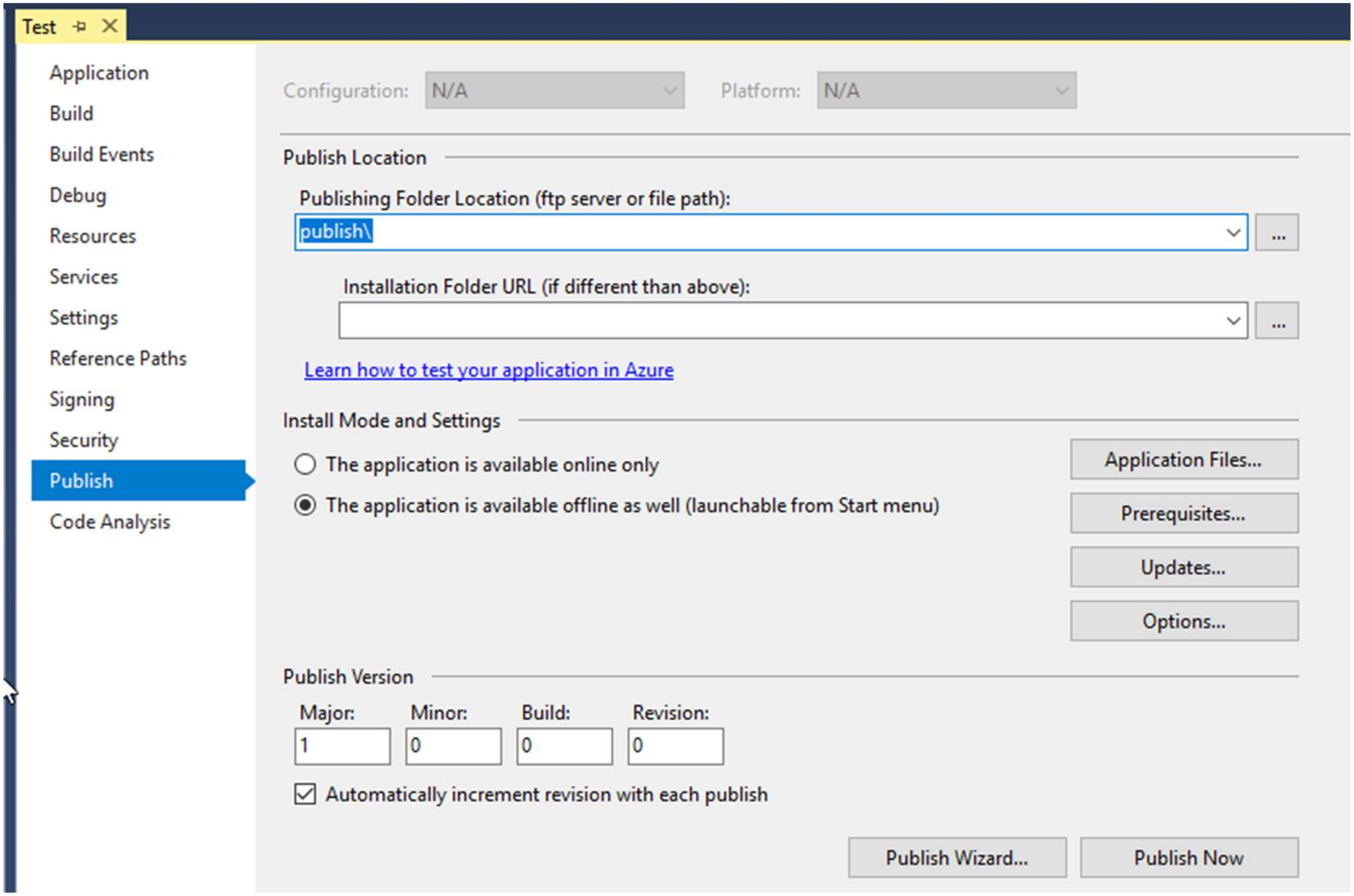
- A version number of an assembly is a four-part string:

<major version>.<minor version>.<build number>.<revision>

- Applications reference particular versions of assemblies
  - specifies that the runtime should use version 2.0.0.0 instead of the assembly version 1.0.0.0

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="...">
     <dependentAssembly>
       <assemblyIdentity name="FourthCoffee.Core"
           publicKeyToken="32ab4ba45e0a69a1" culture="en-us" />
       <bindingRedirect oldVersion="1.0.0.0" newVersion="2.0.0.0"/>
     </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Versioning Assemblies



Configuration: N/A

Platform: N/A

Publish Location

Publishing Folder Location (ftp server or file path): publish\

Installation Folder URL (if different than above):

Learn how to test your application in Azure

Install Mode and Settings

○ The application is available online only

● The application is available offline as well (launchable from Start menu)

Publish Version

Major:	Minor:	Build:	Revision:
1	0	0	0

☑ Automatically increment revision with each publish

Installing an Assembly into the GAC

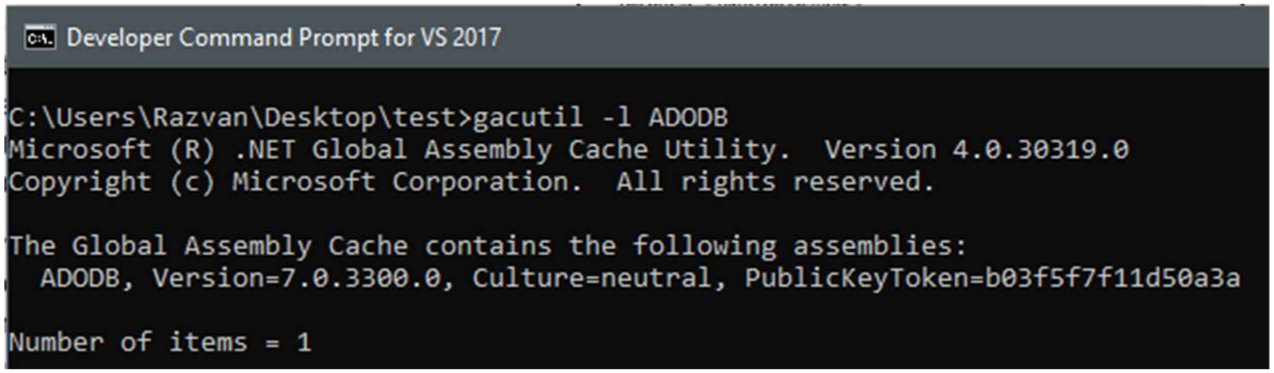
You can Install an assembly in the GAC by using:

- Global Assembly Cache tool (Gautil.exe) OR
  - It's only for development purposes … you should not use it for production assemblies
  - To install: use Visual Studio Command Prompt and the following command:

```
gacutil –i "<pathToAssembly>"
```

  - To view an assembly installed into the GAC, in VS Command Prompt use the command:

```
gacutil –l "<assemblyName>"
```

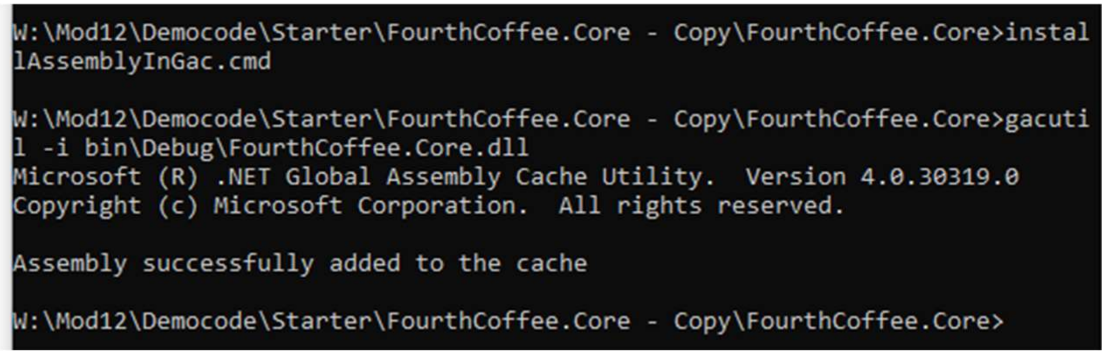


```
C:\Users\Razvan\Desktop\test>gacutil -l ADODB
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.0
Copyright (c) Microsoft Corporation.  All rights reserved.

The Global Assembly Cache contains the following assemblies:
  ADODB, Version=7.0.3300.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a

Number of items = 1
```

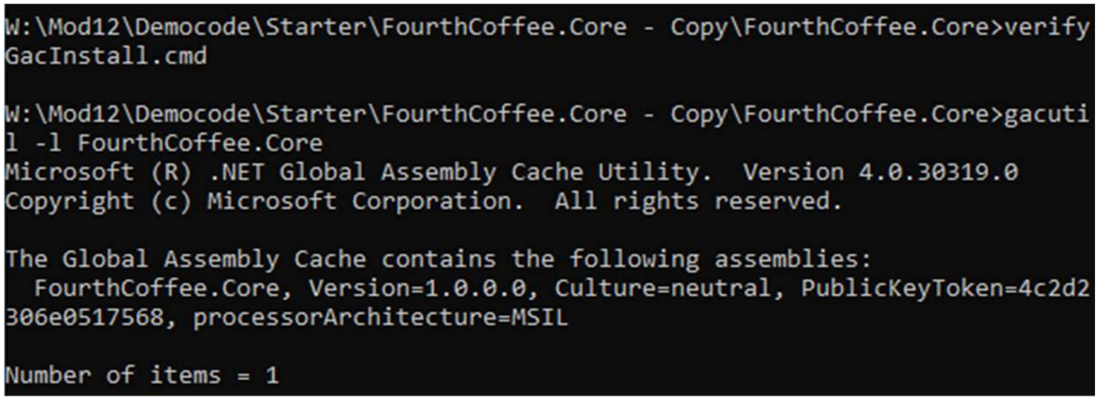
- Microsoft Windows Installer
  - This is the recommended and most common way to add assemblies to the GAC

Text Continuation

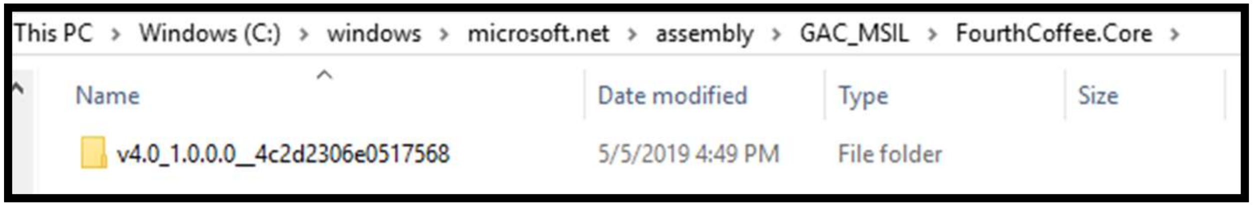
- Note: you may want to read this: [Gacutil.exe successfully adds assembly, but assembly not viewable in explorer. Why?]()
  - Instead of using C:\WINDOWS\assembly,
  - the .NET 4.0 version of gacutil.exe stores the assembly in a different GAC: c:\windows\microsoft.net\assembly



```
W:\Mod12\Democode\Starter\FourthCoffee.Core - Copy\FourthCoffee.Core>instal
lAssemblyInGac.cmd

W:\Mod12\Democode\Starter\FourthCoffee.Core - Copy\FourthCoffee.Core>gacuti
l -i bin\Debug\FourthCoffee.Core.dll
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.0
Copyright (c) Microsoft Corporation.  All rights reserved.

Assembly successfully added to the cache

W:\Mod12\Democode\Starter\FourthCoffee.Core - Copy\FourthCoffee.Core>
```

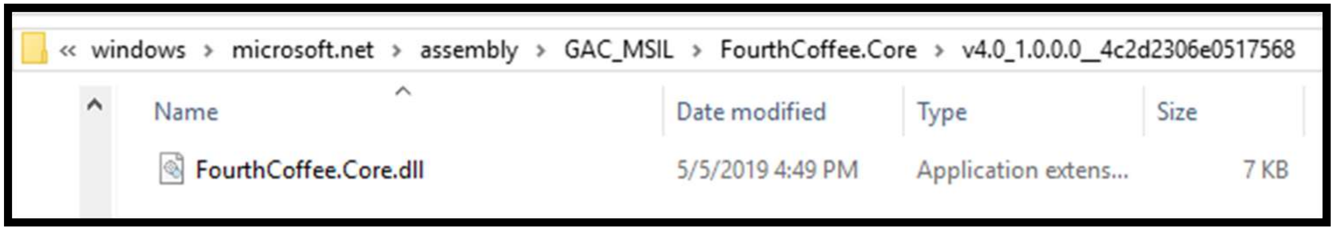


```
W:\Mod12\Democode\Starter\FourthCoffee.Core - Copy\FourthCoffee.Core>verify
GacInstall.cmd

W:\Mod12\Democode\Starter\FourthCoffee.Core - Copy\FourthCoffee.Core>gacuti
l -l FourthCoffee.Core
Microsoft (R) .NET Global Assembly Cache Utility.  Version 4.0.30319.0
Copyright (c) Microsoft Corporation.  All rights reserved.

The Global Assembly Cache contains the following assemblies:
  FourthCoffee.Core, Version=1.0.0.0, Culture=neutral, PublicKeyToken=4c2d2
306e0517568, processorArchitecture=MSIL

Number of items = 1
```



This PC › Windows (C:) › windows › microsoft.net › assembly › GAC_MSIL › FourthCoffee.Core ›

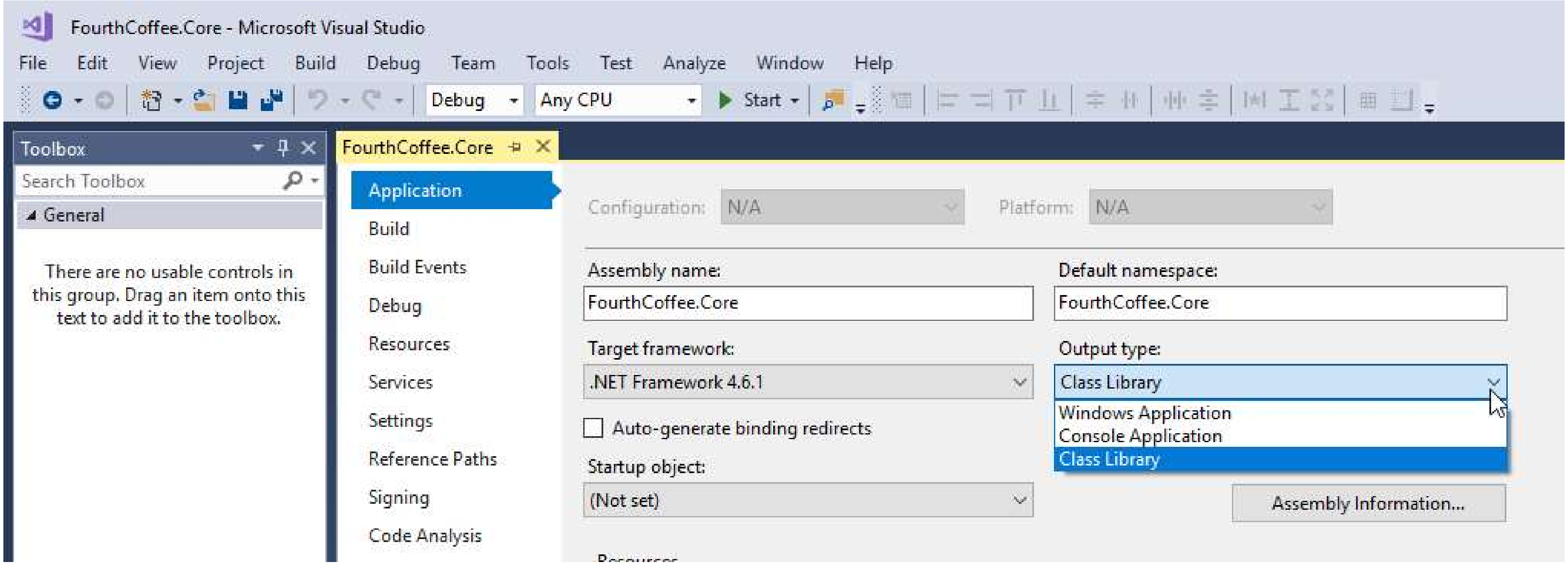
Name	Date modified	Type	Size
v4.0_1.0.0.0__4c2d2306e0517568	5/5/2019 4:49 PM	File folder	



« windows › microsoft.net › assembly › GAC_MSIL › FourthCoffee.Core › v4.0_1.0.0.0__4c2d2306e0517568

Name	Date modified	Type	Size
FourthCoffee.Core.dll	5/5/2019 4:49 PM	Application extens...	7 KB

Creating .dll (class library) vs executable file



FourthCoffee.Core - Microsoft Visual Studio

Configuration: N/A

Platform: N/A

Assembly name: FourthCoffee.Core

Default namespace: FourthCoffee.Core

Target framework: .NET Framework 4.6.1

Output type: Class Library

- Windows Application
- Console Application
- Class Library

□ Auto-generate binding redirects

Startup object: (Not set)

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

Module Review and Takeaways

- **Question:** You are developing an application that enables users to browse the object model of a compiled type. At no point will the application attempt to execute any code; it will merely serve as a viewer. You notice the code that loads the assembly uses the **Assembly.LoadFrom** static method. This is the most suitable method taking into account the requirements of the application.
  - ( )False
  - ( )True
- **Question:** You are developing a custom attribute. You want to derive your custom attribute class from the abstract base class that underpins all attributes. Which class should you use?
  - ( )Option 1: Attribute
  - ( )Option 2: ContextAttribute
  - ( )Option 3: ExtensionAttribute
  - ( )Option 4: DataAttribute
  - ( )Option 5: AddInAttribute
- **Question:** You are reviewing some code that uses CodeDOM to generate managed Visual C# at run time. What does the following line of code do?

  ```
  var method = new CodeEntryPointMethod();
  ```

  - ( )Option 1: Defines an instance method with a random name.
  - ( )Option 2: Defines an instance method named EntryPoint.
  - ( )Option 3: Defines a static method named EntryPoint.
  - ( )Option 4: Defines an instance method named Main.
  - ( )Option 5: Defines a static method named Main.
- **Question:** The **FourthCoffee.Core.dll** assembly has 2.1.0.24 as its version number. The number 24 in the version number refers to the build number.
  - ( )False
  - ( )True