itcourseware™

TRAINING MATERIALS FOR IT PROFESSIONALS

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

.NET Foundations
Rev. 7.0

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Student Guide

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Object Innovations.

Product and company names mentioned herein are the trademarks or registered trademarks of their respective owners.

™ is a trademark of Object Innovations.

Author: Robert J. Oberg

Copyright ©2022 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations
877-558-7246
www.objectinnovations.net

Published in the United States of America.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Table of Contents (Overview)

Chapter 1	.NET Fundamentals
Chapter 2	Class Libraries
Chapter 3	Frameworks and Packages
Chapter 4	Metadata and Reflection
Chapter 5	I/O and Serialization
Chapter 6	Delegates and Events
Chapter 7	.NET Programming Model
Chapter 8	.NET Threading
Appendix A	Learning Resources

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright ©2022 Object Innovations Enterprises, LLC
All Rights Reserved

Directory Structure

- **The course software installs to the root directory *C:\OIC\NetCs*.**
  - Example programs for each chapter are in named subdirectories of chapter directories **Chap01**, **Chap02**, and so on.
  - The **Labs** directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.
  - The **Demos** directory is provided for hand-on work during lectures.
  - The **Packages** directory is for custom packages.
  - The **Run** directory is for .NET Core executables to be launched programmatically as a process.
  - The file **Links.doc** contains link to web sites and pages mentioned in the text along with other useful links.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright ©2022 Object Innovations Enterprises, LLC
All Rights Reserved

Table of Contents (Detailed)

Chapter 1 .NET Fundamentals 1

What Is Microsoft .NET? 3
Open Standards and Interoperability 4
Windows Development Problems 5
Common Language Runtime 6
Serialization Example 7
Class Library Programming 10
Types 11
.NET Class Library 12
Everything is an Object 13
Common Type System 14
Language Interoperability 15
Managed Code 16
JIT Compilation 17
The Role of XML 18
Performance 19
.NET Core and .NET 7 20
Summary 21

Chapter 2 Class Libraries 23

Objects and Components 25
Limitation of COM Components 26
Components in .NET 27
Component Example: Customer Management System 28
Monolithic versus Component 30
Class Libraries Using Visual Studio 31
Demo: Creating a Class Library 32
Creating a Client Project 35
References in Visual Studio 37
References at Compile Time and Run Time 39
Project Dependencies 40
Lab 2 41
Summary 42

Chapter 3 Frameworks and Packages 47

.NET Implementations 49
Target Frameworks 50
.NET Standard 51
.NET and .NET Framework 52
Targeting .NET Framework 53
Targeting .NET 56
Comparing the Apps 58
What is NuGet? 59

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Packages, Metapackages and Frameworks 60
Packages 61
Key Packages for .NET 62
Metapackages 63
Frameworks 64
.NET 7 API Reference 65
.NET API Browser 66
Lab 3A 67
Example: File Output 68
Demo: A Windows Forms App 69
Dependencies 75
NuGet Package Manager 76
Installed Packages 79
Creating a Package from a Class Library 81
Example: CustomerLib Package 82
Putting a Package in a NuGet Source 85
Managing Package Sources 86
Adding a Package Source 87
Using Your Package 88
Lab 3B 89
Summary 90

Chapter 4 Metadata and Reflection 97
Metadata 99
Reflection 100
Sample Reflection Program 101
System.Reflection.Assembly 104
System.Type 105
System.Reflection.MethodInfo 107
Dynamic Invocation 108
Late Binding 109
LateBinding Example 110
Lab 4 111
Summary 112

Chapter 5 I/O and Serialization 117
Input and Output in .NET 119
Directories 120
Directory Example Program 121
Files and Streams 124
"Read" Command 126
Code for "Write" Command 127
Serialization 128
XML Serialization Demo 129
Code Walkthrough 130
XML Serialization Infrastructure 135
What Will Not Be Serialized 136

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright ©2022 Object Innovations Enterprises, LLC
All Rights Reserved

Lab 5 137
Summary 138

Chapter 6 Delegates and Events 143
Overview of Delegates and Events 145
Callbacks and Delegates 146
Usage of Delegates 147
Declaring a Delegate 148
Defining a Method 149
Creating a Delegate Object 150
Calling a Delegate 151
Random Number Generation 152
A Random Array 153
Combining Delegate Objects 154
Account.cs 155
Program.cs 156
Running the Example 157
Anonymous Methods 158
Lambda Expressions 159
Named Method Example 160
Anonymous Method Example 161
Lambda Expression Example 162
Events 163
Events in C# and .NET 164
Client Side Event Code 166
Chat Room Example 167
Lab 6 168
Summary 169

Chapter 7 .NET Programming Model 173
Garbage Collection 175
Finalize Method 176
C# Destructor Notation 177
Dispose 178
Garbage Collection Performance 179
Generations 180
Processes 181
Application Isolation 182
Process Example 183
Properties, Methods and Events 185
Exited Event 186
Lab 7 187
Command Line Arguments 188
SimpleLog Class 189
Arguments in Visual Studio 190
Starting a .NET Core Process 191
The Run Directory 193

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright ©2022 Object Innovations Enterprises, LLC
All Rights Reserved

Process Launch in .NET 7.0 194
Threads 195
Summary 196

Chapter 8 .NET Threading 199

Threads 201
.NET Threading Model 202
ThreadDemo Example 203
Sample Output 207
Race Conditions 208
Race Condition Example 209
Race Condition Output 213
Thread Synchronization 214
Monitor 215
Monitor Example 216
Monitor Example Output 217
Using C# *lock* Keyword 218
Synchronization of Collections 219
ThreadPool Class 220
ThreadPool Example 221
Starting a ThreadPool Thread 222
Foreground and Background Threads 223
Synchronizing Threads 224
Improved ThreadPool Example 225
Task Parallel Library (TPL) 227
Task Example 228
Starting Tasks 229
Waiting for Task Completion 230
Data Parallelism 231
Lab 8 232
Summary 233

Appendix A Learning Resources 241

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright ©2022 Object Innovations Enterprises, LLC
All Rights Reserved

Chapter 1

.NET Fundamentals

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

.NET Fundamentals

Objectives

After completing this unit you will be able to:

- **Understand the problems Microsoft .NET is designed to solve.**
- **Understand the basic programming model of Microsoft .NET.**
- **Understand the basic programming tools provided by Microsoft .NET.**
- **Discuss .NET Core, .NET 7 and cross-platform development.**

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

What Is Microsoft .NET?

- **This course is about .NET 7, which is based on .NET Core and is cross-platform.**
- **This chapter discusses fundamental concepts of .NET in general, which includes the classical .NET Framework, which runs only on Windows.**
  - The specifics of .NET Core will be covered in the rest of the course.
  - The programming example in this chapter does use .NET Core.
- **Microsoft .NET was developed to solve three fundamental problems.**
  - First, the Microsoft Windows programming model must be unified to remove the widely varied programming models and approaches that exist among the various Microsoft development technologies.
  - Second, Microsoft based solutions must be capable of interacting with the modern world of heterogeneous computing environments.
  - Third, Microsoft needs a development paradigm that is capable of being expanded to encompass future development strategies, technologies, and customer demands.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Open Standards and Interoperability

- **The modern computing environment contains a vast variety of hardware and software systems.**
  - Computers range from mainframes and high-end servers, to workstations and PCs, and to small mobile devices such as PDAs and smartphones.
  - Operating systems include traditional mainframe systems, many flavors of Unix including Android, Linux, Apple's iOS, several versions of Windows, real-time systems and more.
  - Many different languages, databases, application development tools and middleware products are used.
- **Applications need to be able to work in this heterogeneous environment.**
  - Even shrink-wrapped applications deployed on a single PC may use the Internet for registration and updates.
- **The key to interoperability among applications is the use of *standards*, such as HTML, HTTP, XML, and TCP/IP.**

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Windows Development Problems

- **In classic Windows development design and language choice often clashed.**
  - Visual Basic vs. C++ approach
  - IDispatch, Dual, or Vtable interfaces
  - VB vs. MFC
  - ODBC or OLEDB or ADO
- **Application deployment was hard.**
  - Critical entries in Registry for COM components
  - No versioning strategy
  - DLL Hell
- **Too much time is spent in writing plumbing code that the system should provide.**
  - MTS/COM+ was a step in the right direction.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Common Language Runtime

- **The first step in solving the three fundamental problems is for Microsoft .NET to provide a set of underlying services available to all languages.**
- **The runtime environment provided by .NET that provides these services is called the *Common Language Runtime* or CLR.**
  - A runtime provides services to executing programs.
  - Traditionally there are different runtimes for different programming environments. Examples of runtimes include the standard C library, MFC, the Visual Basic 6 runtime and the Java Virtual Machine.
- **These services are available to all languages that follow the rules of the CLR.**
  - .NET languages include C#, F# and Visual Basic.
  - Object Innovations' curriculum emphasizes C#.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Serialization Example

- **Let us use serialization to illustrate how the CLR provides a set of services that unifies the Microsoft development paradigm.**
  - Every programmer has to do it.
  - It can get complicated with nested objects, complicated data structures, and a variety of data storages.
- **However, serialization can open up security vulnerabilities.**
  - In particular, Binary Serialization, part of the classical .NET Framework, is insecure and cannot be made secure. It should be avoided.
  - See this article for more information:

https://docs.microsoft.com/en-us/dotnet/standard/serialization/binaryformatter-security-guide

  - NOTE: See the file **Links.doc** in the lab distribution for links mentioned in this Student Guide.
- **The *Serialize* example in this chapter uses XmlSerialization.**

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Serialization Example (Cont'd)

- **Ignore the language details covered in a later chapter.**

```
class Customer
{
   public string name;
   public long id;
}
class Program
{
   static void Main(string[] args)
   {
      List<Customer> list = new List<Customer>();

      Customer cust = new Customer();
      cust.name = "Charles Darwin";
      cust.id = 10;
      list.Add(cust);

      cust = new Customer();
      cust.name = "Isaac Newton";
      cust.id = 20;
      list.Add(cust);

      foreach (Customer x in list)
         Console.WriteLine(x.name + ": " + x.id);

      Console.WriteLine("Saving Customer List");

      XmlSerializer ser = new
         XmlSerializer(typeof(List<Customer>));
      FileStream s = new FileStream("cust.xml",
                                    FileMode.Create);
      ser.Serialize(s, list);
      s.Close();
      ...
```

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Serialization Example (Cont'd)

```
    Console.WriteLine("Restoring to New List");

    FileStream s2 =
       new FileStream("cust.xml", FileMode.Open);
    list = (List<Customer>)ser.Deserialize(s2);

    foreach (Customer y in list2)
       Console.WriteLine(y.name + ": " + y.id);
    s2.Close();
}
```

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Class Library Programming

- **We add two *Customer* objects to the collection, and print them out. We save the collection to disk and then restore it. The identical list is printed out.**

```
Charles Darwin: 10
Isaac Newton: 20
Saving Customer List
Restoring to New List
Charles Darwin: 10
Isaac Newton: 20
Press enter to continue...
```

- **We wrote little code to save or restore the list!**
  - Nothing special was needed in the **Customer** class.
  - We used several classes in the.NET Class Library to do the heavy lifting for us in the **Main()** method.
- **Classes are used throughout .NET to provide useful functionality, such as manage collections, do I/O and work with XML.**
  - The data file created was **cust.xml**.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Types

- ***Types* are at the heart of the programming model for the CLR.**
- **A type is analogous to a class in most object-oriented programming languages, providing an abstraction of data and behavior, grouped together.**
- **A type in the CLR contains:**
  - Fields (data members)
  - Methods
  - Properties
  - Events (which are now full-fledged members of the Microsoft programming paradigm).

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

.NET Class Library

- **The *List<T>*, *XMLFormatter* and *FileStream* classes are some of the thousands of classes in the .NET Framework that provide system services.**
- **The functionality provided includes:**
  - Base Class Library (basic functionality such as strings, arrays and formatting).
  - Networking
  - Security
  - Diagnostics
  - I/O
  - Database
  - XML
  - Web programming
- **This class library is usable by all CLR compliant languages.**

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Everything is an Object

- **Every type in .NET derives from *System.Object*.**[1]
- **Every type, system or user defined, has metadata.**
  - In the sample the framework can walk through the List of Customer objects and save each one as well as the array itself.
- **All access to objects in .NET is through object references.**

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

[1] An exception is the pointer type, which is rarely used in C#.

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Common Type System

- **The Common Type System (CTS) defines the rules for the types and operations that the CLR will support.**
  - The CTS limits .NET classes to single implementation inheritance.
  - The CTS is designed for a wide range of languages, not all languages will support all features of the CTS.
- **The CTS makes it *possible* to guarantee type safety.**
  - Access to objects can be restricted to object references (no pointers), each reference refers to a defined memory. Access to that layout is only through public methods and fields.
  - By performing a local analysis of the class, you can verify to make sure that the code does not perform any inappropriate memory access. You do not have to analyze the users of the class.
- **.NET compilers emit Microsoft Intermediate Language (MSIL or IL) not native code.**
  - MSIL is platform independent.
  - Type-safe code can be restricted to a subset of verifiable MSIL expressions.
  - Once code is verified, it is verified for all platforms.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Language Interoperability

- **Having all language compilers use a common intermediate language and common base class makes it *possible* for languages to interoperate.**
  - All languages need not implement all parts of the CTS.
  - One language can have a feature that another does not.
- **The *Common Language Specification* (CLS) defines a subset of the CTS that represents the basic functionality that all .NET languages should implement if they are to interoperate with each other.**
  - For example, a class written in Visual Basic can inherit from a class written in C#.
  - Interlanguage debugging is possible.
  - CLS rule: Method calls need not support a variable number of arguments even though such a construct can be expressed in MSIL.
  - CLS prohibits the use of pointers.
  - CLS is an ECMA specification.
- **CLS compliance only applies to public features.**
  - C# code should not define public and protected class names that differ only by case sensitivity since languages such as Visual Basic are not case sensitive. Private C# fields could have such names.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Managed Code

- **In the serialization example we never freed any allocated memory.**
  - Memory that is no longer referenced can be reclaimed by the CLR's garbage collector.
  - Automatic memory management eliminates the common programming error of memory leaks.
  - Garbage collection is one of the services provided to .NET applications by the Common Language Runtime.
- **Managed code uses the services of the CLR.**
  - MSIL can express access to unmanaged data in legacy code.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

JIT Compilation

- **Before executing on the target machine, MSIL is translated by a just-in-time (JIT) compiler to native code.**
- **Some code typically will never be executed during a program run.**
  - Hence it may be more efficient to translate MSIL as needed during execution, storing the native code for reuse.
- **When a type is loaded, the loader attaches a stub to each method of the type.**
  - On the first call the stub passes control to the JIT, which translates to native code and modifies the stub to save the address of the translated native code.
  - On subsequent calls to the method transfer is then made directly to the native code.
- **As part of JIT compilation code goes through a verification process.**
  - Type safety is verified, using both the MSIL and metadata.
  - Security restrictions are checked.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

The Role of XML

- **XML is ubiquitous in .NET and is highly important in Microsoft's overall vision.**
- **Some uses of XML in .NET include:**
  - XML can be used for serialization.
  - XML is used extensively in configuration files.
  - XML documentation can be automatically generated by .NET languages.
  - .NET classes provide a very convenient API for XML programming as an alternative to DOM or SAX.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Performance

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

- **Concerns about performance of managed code are similar to the concerns assembly language programmers had with high level languages.**
- **Garbage collection usually produces faster allocation than C++ unmanaged heap allocation. Deallocation is done on a separate thread by the garbage collector.**
- **JIT compilation takes a hit the first time when verification and translation take place, but subsequent executions pay no penalty.**
- **Bottom line: for most of the code that is written, any small loss in performance is far outweighed by the gains in reliability and ease of development.**
  - High performance servers might still have to use technologies such as C++.
- **Apps targeting the Windows Universal Platform (UWP) may use .NET Native to achieve higher performance.**
  - https://learn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
  - https://learn.microsoft.com/en-us/windows/uwp/dotnet-native/

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

.NET Core and .NET 7

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

- **.NET Core is a modular version of the classical .NET Framework that is portable across multiple platforms.**
  - .NET Core represents the future of .NET.
  - Latest version of .NET Core is .NET 7.
- **Rather than one large assembly, .NET Core is released through NuGet in smaller feature-specific assembly packages.**
  - The *metapackage* convention allows a set of packages that are meaningful together to be treated as a unit.
- **.NET Core provides key functionality used in applications regardless of platform.**
  - This common functionality provides for shared code that can be used across platforms.
  - Your application then links in additional platform-specific code.
- **Microsoft platforms you can target include traditional desktop Windows and Windows phones.**
- **Other platforms include Mac and Linux.**
- **With Xamarin (now owned by Microsoft) you can target Android and iOS mobile platforms.**

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Summary

- **.NET solves problems of past Windows development.**
- **One development paradigm for all languages exists.**
- **Design and programming language no longer conflict.**
- **.NET uses managed code with services provided by the Common Language Runtime that uses the Common Type System.**
- **Plumbing code for fundamental system services is there, yet you can extend it or replace it if necessary.**
- **The .NET Class Library is a very large class library available consistently across many languages.**
- **.NET Core is a modular version of the classical .NET Framework that is portable across multiple platforms.**
  – The latest version of .NET Core is .NET 7.0.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Chapter 2

Class Libraries

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Class Libraries

Objectives

After completing this unit you will be able to:

- **Describe the role of components in software development.**
- **Compare components in COM and .NET.**
- **Create class libraries using Visual Studio 2022.**
- **Use components in client programs by obtaining references to class libraries.**

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Objects and Components

- **An *object* encapsulates data and behavior, and it facilitates reuse—*at the source code level.***
  - A class library, such as Microsoft Foundation Classes (MFC) provides reusable code in the form of a hierarchy of classes.
  - But you cannot use MFC classes in a Visual Basic 6 program.
- **By contrast, a *component* is a binary piece of software that can be reused in many different programming languages.**
  - For example, you can use COM components and ActiveX controls (a particularly rich type of COM component, typically with a graphical user interface) in Visual Basic 6 programs.
  - The component could be implemented in some other language, such as C++. The Visual Basic 6 programmer does not care.
- **The basic concept of component is a black box piece of software that can be reused.**
  - By this loose definition, a DLL would be a component.
- **Usually, somewhat more is meant, such as some kind of "object orientation."**
  - Examples of such object-oriented components are COM objects, JavaBeans, and CORBA objects.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Limitation of COM Components

- **COM lacks support for implementation inheritance.**
  - You cannot start with a base component and inherit its methods. (You can achieve similar reuse by other techniques, such as containment and aggregation, but such reuse is not as easy or convenient as inheritance.)
- **Another drawback of COM components is the requirement that the component implement "plumbing" code that allows it be called in a black box fashion from another piece of software.**
  - Visual Basic 6 hid the plumbing code, but it was there, and VB6 programs could only use a subset of the capabilities of COM.
  - In C++ you could fully utilize COM, but there was a lot of work to be done in implementing the plumbing code. Specialized libraries like the Active Template Library (ATL) could do a lot of the work for you, but that required you to learn yet another piece of technology, and it applied only to C++.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Components in .NET

- **The .NET Class Library provides an exceptionally attractive environment for creating and using software components.**
- **By simply setting an appropriate compiler switch or choosing a specific project type in Visual Studio, you can build a *class library*.**
  - A class library is the .NET version of a component and is a DLL that packages the code for a set of classes.
- **There is no special plumbing code that must be provided.**
- **These class library DLLs can easily be used by other .NET programs, and you can mix .NET languages freely.**
- **Also, you can inherit from a class implemented in a class library.**
  - This inheritance mechanism extends across languages, since a class library is a binary component.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Component Example: Customer Management System

- **We will illustrate how to work with components in .NET by componentizing a monolithic application.**
- **The application manages a list of customers.**
  - The **Customers** class manages a collection of customers, as defined by the **Customer** class.

```
public class Customer
{
   public int CustomerId;
   public string FirstName;
   public string LastName;
   public string EmailAddress;
   ...
}
```

- **The *Customers* class implements the *ICustomer* interface.**

```
public interface ICustomer
{
   int RegisterCustomer(string firstName,
      string lastName, string emailAddress);
   void UnregisterCustomer(int id);
   List<Customer> GetCustomers();
   void ChangeEmailAddress(int id,
      string emailAddress);
}
```

  - The **GetCustomers()** method returns a list of customers.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Customer Management (Cont'd)

- **The folder *CustomerMonolithic* contains a Console version of a customer management system, with all code in one project.**
  - Examine the code, and then build and exercise the program.
  - The "help" command shows the available commands.
  - Here is a sample session. Note that some test data has been supplied for an initial list of customers.

```
Enter command, quit to exit
> help
The following commands are available:
        reg      -- register a customer
        unreg    -- unregister a customer
        show     -- show customers
        email    -- change email
        quit     -- exit the program
> show
1     Amy        Jones      amy@acme.com
2     Bob        Smith      smith@nc.gov
3     Carl       Grant      carl@arts.org
> unreg
CustomerId: 2
> reg
First Name: Dave
Last Name: Moore
Email Address: dave@foo.com
> show
1     Amy        Jones      amy@acme.com
3     Carl       Grant      carl@arts.org
4     Dave       Moore      dave@foo.com
>
```

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Monolithic versus Component

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

- **In this monolithic version of the program, the business logic (as specified by the *ICustomer* interface) and the user interface code are mixed together.**
- **A better structure is to isolate the business logic in a separate component.**
  - Create a DLL **Customer.dll** that contains the **Customers** class implementing the **ICustomer** interface.
- **The .NET Class Library then provides various mechanisms by which a client program can access this functionality, and there can be different kinds of clients:**
  - Console application client
  - Web client
  - Mobile app client
  - and so forth

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Class Libraries Using Visual Studio

- **Visual Studio makes it very easy to work with .NET class libraries.**
  - You can create a class library by using the Class Library project type.
  - You can use Solution Explorer to add references.
  - It is all quite painless.
- **There is one nuance involved when using Visual Studio that you need to be aware of.**
  - Visual Studio by default creates a root namespace based on the name of the project. (Note that we will usually delete unused **using** statements, which are shown in light gray.)

```
using System;
using System.Collections.Generic;

namespace CustomerLib
{
    public class Class1
    {
        public Class1()
        {
        }
    }
}
```

  - This applies to both class library and application projects.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

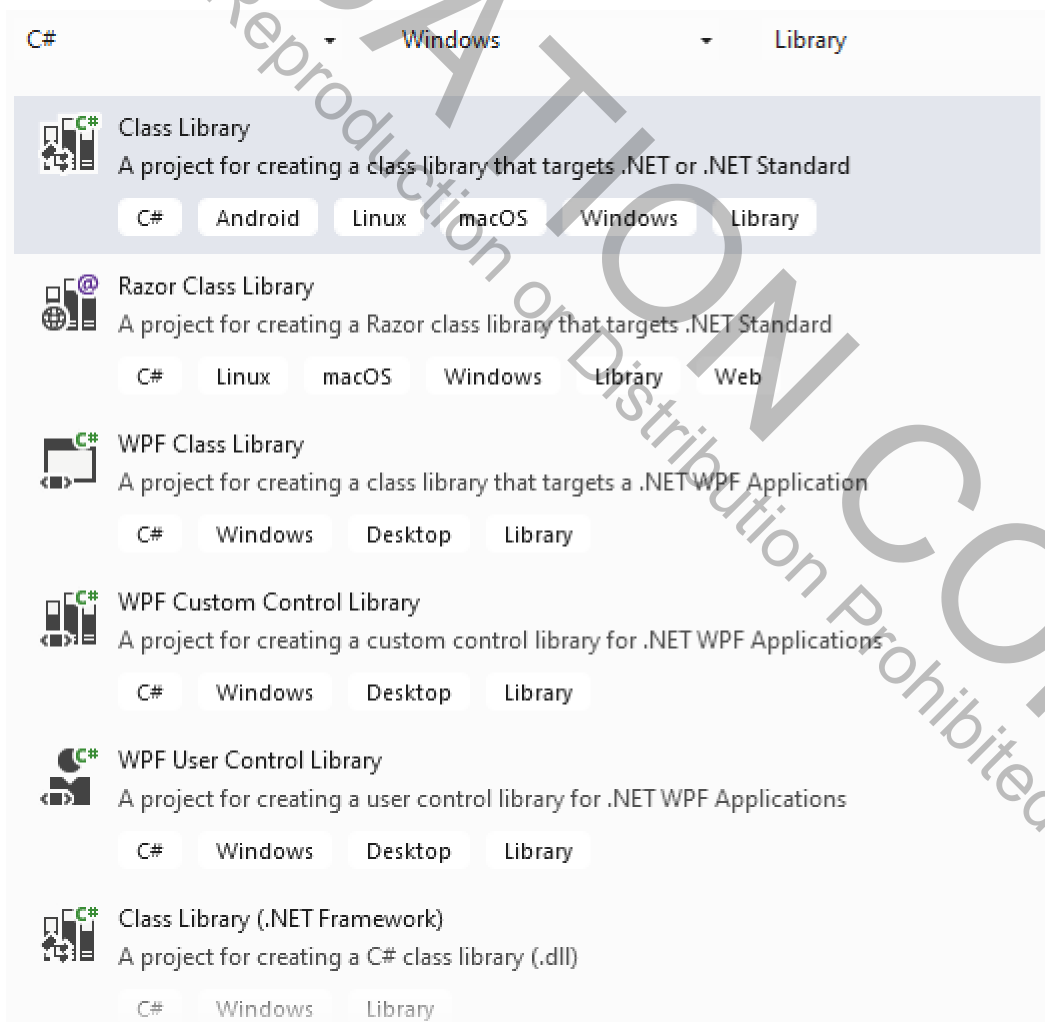
 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Demo: Creating a Class Library

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

- **We will use Visual Studio to create a class library containing the *Customers* class.**
  - Do your work in the **Demos** directory. The completed example is available in **CustomerVs** in the chapter folder.

1. Bring up the New Project dialog from the menu File | New Project. Filter by C# for language, Windows for platform, and Library for project type. Choose Class Library.



2. Click Next.

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Creating a Class Library (Cont'd)

3. Type **CustomerLib** for the Project name. Navigate to the Demos folder. For the Solution name type **CustomerVs**. Leave unchecked "Place solution and project in the same directory". That will facilitate adding a second project to the solution for a client test program. Click Next.

Configure your new project

Class Library C# Android Linux macOS Windows Library

Project name

CustomerLib

Location

C:\OIC\NetCs\Demos\

Solution

Create new solution

Solution name

CustomerVs

Place solution and project in the same directory

- **As discussed in the next chapter, class libraries in .NET 7 are made available to client projects through NuGet packages.**
  - Traditionally, .NET Standard was the preferred target for NuGet packages, but going forward .NET 6 or higher is generally recommended.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Creating a Class Library (Cont'd)

4. For framework select .NET 7.0 (Standard term support). Click Create.

Additional information

Class Library C# Android Linux macOS Windows Library

Framework

.NET 7.0 (Standard Term Support)

.NET Standard 2.0
.NET Standard 2.1
.NET 6.0 (Long Term Support)
.NET 7.0 (Standard Term Support)

5. Delete **Class1.cs** from the new project.

6. Copy the file **Customer.cs** from the **CustomerMonolithic** project to the **CustomerLib** folder for this new project. The file will automatically be added to the project.

7. Adjust the namespace.

```
namespace CustomerLib
{
    public interface ICustomer
    {
         ...
```

8. You should now be able to build the class library. The new DLL **CustomerLib.dll** will be created in **bin\Debug\net7.0**.

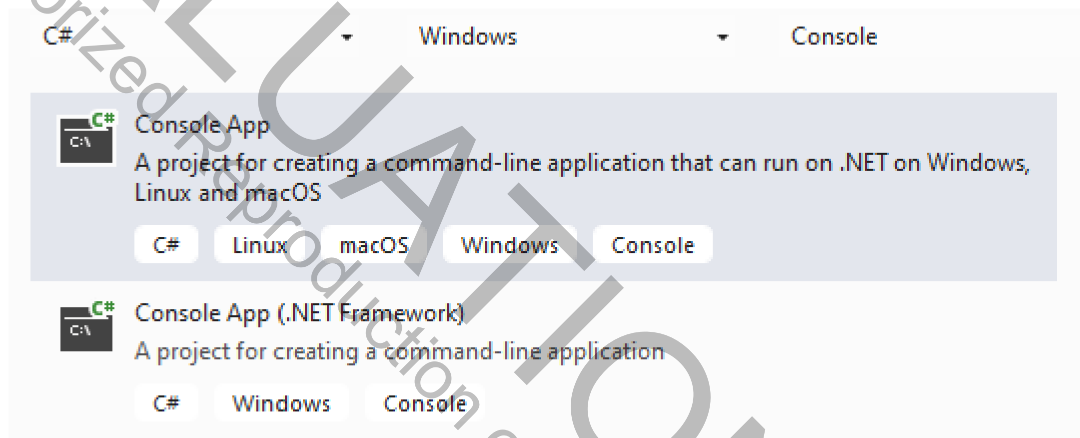
EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

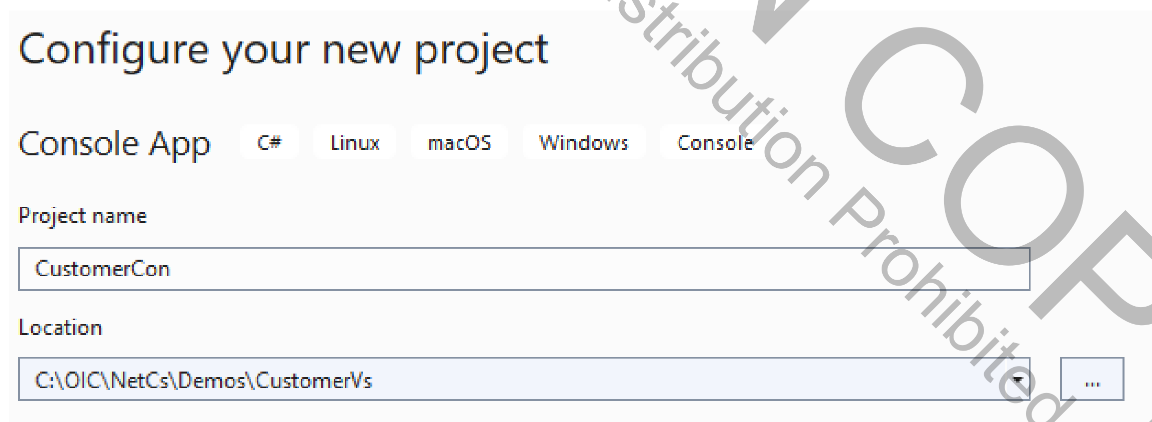
Creating a Client Project

1. Next, let's create a client project, a second project in the solution. Right-click over the solution and choose Add | New Project from the context menu.

2. Filter by Console project type and choose Console App as the template. Click Next.



3. Assign **CustomerCon** as the Project name.



4. Click Next.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Creating a Client Project (Cont'd)

5. Again, choose .NET 7.0 (Standard Term Support) as the framework.

Additional information

Console App C# Linux macOS Windows Console

Framework

.NET 7.0 (Standard Term Support)

Do not use top-level statements

6. Delete the file **Program.cs** from the new project. (Note that the generated **Program.cs** uses the new Console template and consists of a single top-level statement.)

7. Copy the files **Program.cs** and **InputWrapper.cs** from the **CustomerMonolithic** project. Change the namespace in **Program.cs** to **CustomerCon**.

8. Try building the solution. You will get a number of build errors. The biggest problem is that you are missing a reference to the class library.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

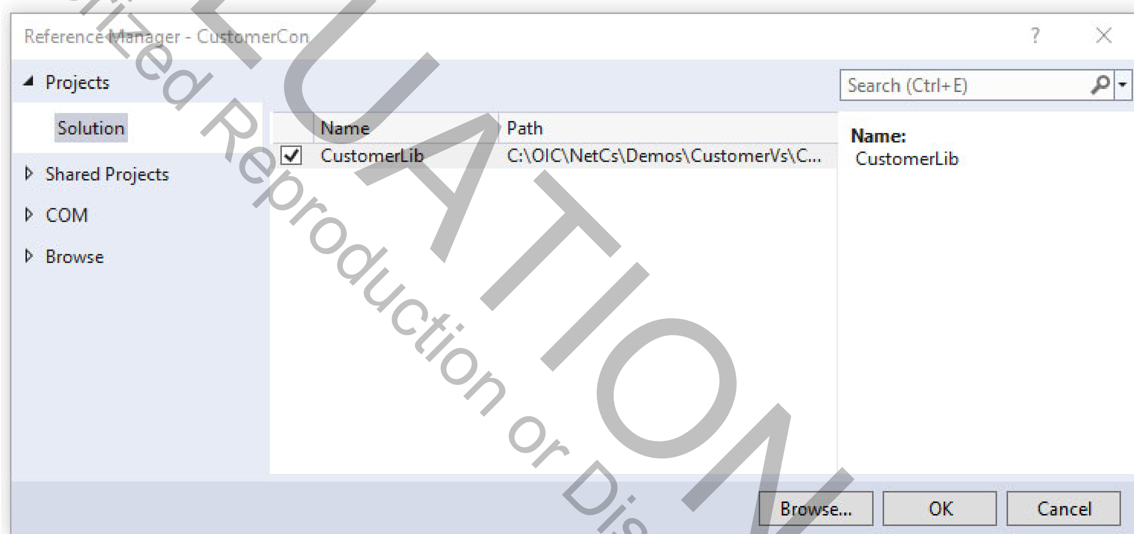
Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

References in Visual Studio

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

1. In Solution Explorer, right-click over the **CustomerCon** project and choose Add | Project Reference from the context menu, bringing up the Reference Manager dialog.

2. Click Solution, and you will see Solution highlighted. Check CustomerLib.



3. Click OK.

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

References in Visual Studio (Cont'd)

4. If you try building, you will still get errors. You need to provide a **using** statement for the class library in **Program.cs**. (An easy way to do this is to right-click over the unrecognized Customers symbol and to choose Quick Actions from the context menu.) You will also need to delete the **using CustomerMonolithic**.

```
using CustomerLib;

Customers custs = new Customers();
init(custs);
InputWrapper iw = new InputWrapper();
...
```

5. Build. It should now compile with no errors (don't worry about warnings).

- **In a multiple-project solution, you should specify a startup project.**
  - Right-click over the desired project and choose Set as Startup Project from the context menu. In our case, make **CustomerCon** the startup project.

6. Run the application. The program should now behave identically to the **CustomerMonolithic** application.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

References at Compile Time and Run Time

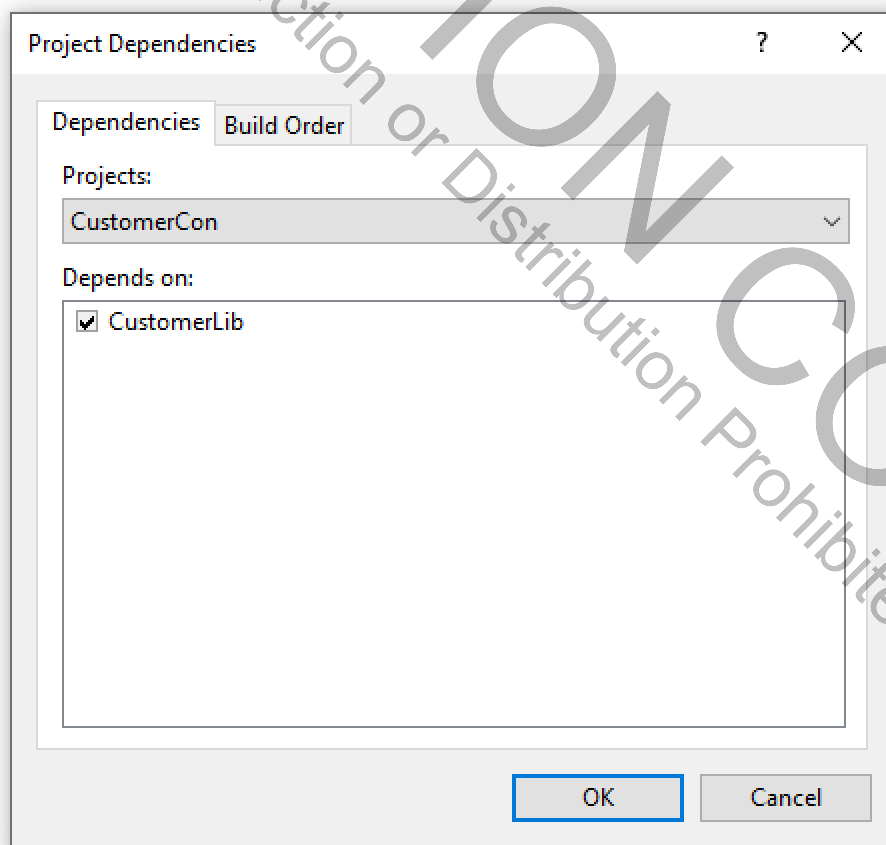
- **The assembly *CustomerLib.dll* is used both at compile time and at runtime.**
  - At compile time the metadata in the assembly is used, and at runtime the code is called.
  - When you build the solution, Visual Studio will copy the DLL into **net7.0** in **bin\Debug** or the **bin\Release**, so it will reside in the proper place at runtime.
  - Examine folder **CustomerCon\bin\Debug\net7.0**. You should now see both **CustomerCon.exe** and **CustomerLib.dll**.
- **.NET Core uses *private* deployment for DLLs in which assemblies ship with the application.**
- **With classic .NET on Windows it is also possible to have *shared* deployment, using the Global Assembly Cache.**
  - There is no GAC in .NET Core.
- **Normally in .NET Core reusable components will be in *packages*.**
  - There is a shared **packages** folder.
  - This topic will be discussed in the next chapter.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Project Dependencies

- **When you have multiple projects in a solution, it becomes important to specify project dependencies.**
  - In our example, you need to build the class library before the Console application.
- **To set project dependencies, right-click over the solution and choose Project Dependencies from the context menu.**
  - When we specified a Project Reference, the dependency was set automatically for us.



EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Lab 2

Implementing a Class Library

In this lab you will implement a class library **SimpleMath.dll** that performs elementary arithmetic operations. You will also create a Console test program to exercise your class library.

Detailed instructions are contained in the Lab 2 write-up at the end of the chapter.

Suggested time: 30 minutes

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Summary

- **A *component* is a binary piece of software that can be reused in many different programming languages.**
- **.NET makes it easy to create and use components, known in .NET as *class libraries*.**
- **You can build class libraries in Visual Studio by creating a class library project.**
- **To use a class library in a client program you must specify a *reference* to it.**
- **In the next chapter we will discuss *packages* as the deployment unit in .NET Core.**

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

Lab 2

Implementing a Class Library

Introduction

In this lab you will implement a class library **SimpleMath.dll** that performs elementary arithmetic operations. You will also create a Console client program to exercise your class library.

Suggested Time: 30 minutes

Root Directory: **OIC\NetCs**

Directories:	**Labs\Lab2**	(create your solution here)
	Chap02\SimpleMath	(answer)

Instructions

1. Use Visual Studio 2022 to create a Class Library project and solution **SimpleMath** in the folder **Labs\Lab2**. Choose .NET 7.0 as the framework.

2. Rename the file **Class1.cs** to **SimpleMath.cs**. Rename the class to **Calc**. Note that the wizard generated code places your class inside the namespace **SimpleMath**.

3. Implement static methods to add and subtract two integers.

```
namespace SimpleMath
{
    public class Calc
    {
        public static int Add(int x, int y)
        {
            return x + y;
        }
        public static int Multiply(int x, int y)
        {
            return x * y;
        }
    }
}
```

4. Build the solution. Find the DLL in your project folders.

5. Use Visual Studio 2022 to add a Console App project **TestMath** to your solution. Set framework to .NET 7.0.

6. Edit the file **Program.cs** to create a simple test program for the **Calc** class. Note that Visual Studio 2022 uses the new console template, so your program will have top-

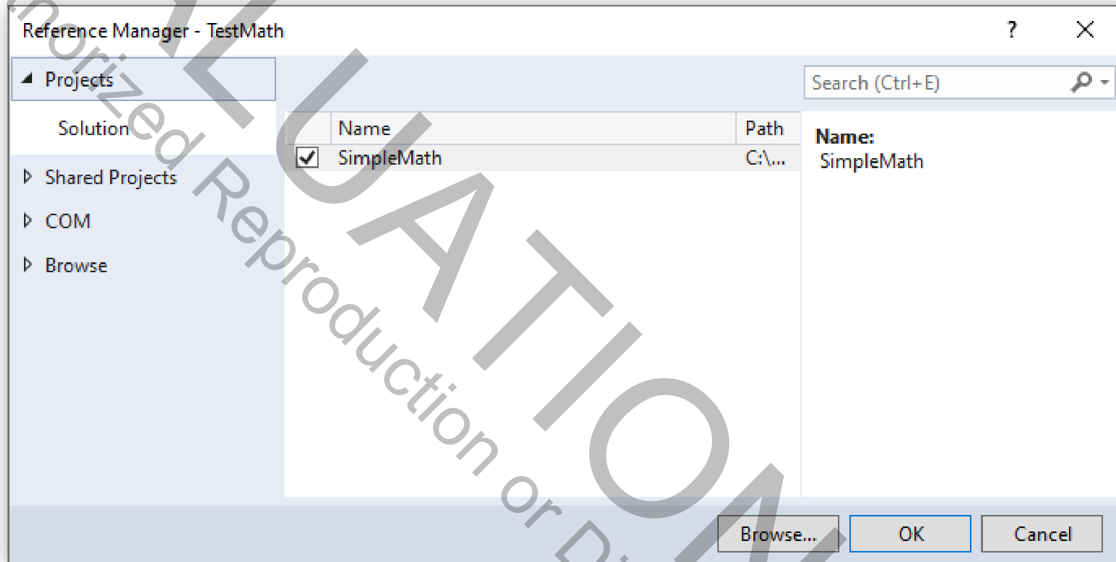
EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

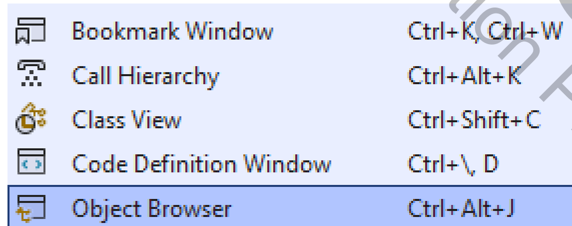
level statements. With this template you get automatic using statements, so you do not need a **using System;** statement.

```
int x = 5;
int y = 7;
int sum = Calc.Add(x, y);
int prod = Calc.Multiply(x, y);
Console.WriteLine("{0} + {1} = {2}", x, y, sum);
Console.WriteLine("{0} + {1} * {2}", x, y, prod);
```

7. Add a reference to the **SimpleMath** project.



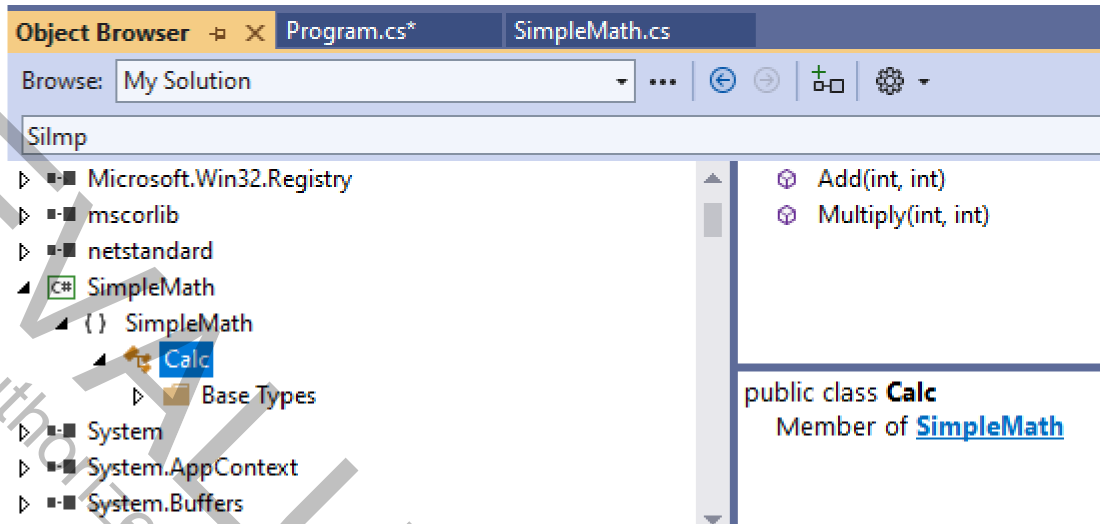
8. Examine the **SimpleMath** DLL in the Object Browser, which you can bring up from the View menu.



...

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved



9. Notice that the class **Calc** is in the namespace **SimpleMath**. Provide an appropriate **using** statement in the file **Program.cs**.

```
using SimpleMath;

int x = 5;
int y = 7;
int sum = Calc.Add(x, y);
int prod = Calc.Multiply(x, y);
Console.WriteLine("{0} + {1} = {2}", x, y, sum);
Console.WriteLine("{0} + {1} * {2}", x, y, prod);
```

10. Make **TestMath** the startup project of the solution.

11. Build and run.

```
5 + 7 = 12
5 + 7 * 35
```

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

 Copyright © 2022 Object Innovations Enterprises, LLC
All Rights Reserved

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

7400 E. Orchard Road, Suite 1450 N
Greenwood Village, Colorado 80111
Ph: 303-302-5280
www.ITCourseware.com

9-08-00440-000-12-20-22