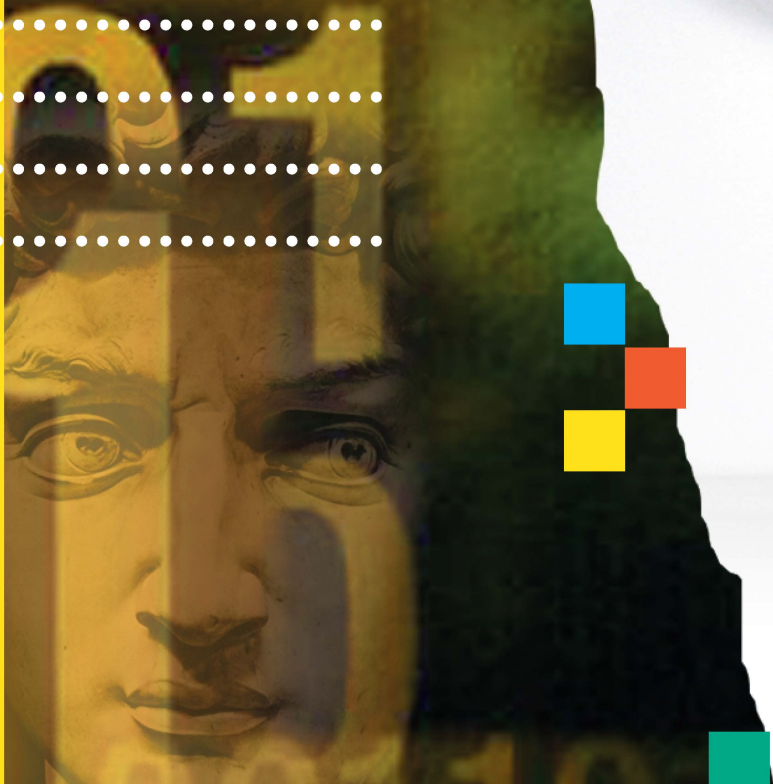


it courseware™

TRAINING MATERIALS FOR IT PROFESSIONALS

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited



Object-Oriented Programming in C#

Rev. 6.1

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.

Authors: Robert J. Oberg, Howard Lee Harkness

Special Thanks: Marianne Oberg, Leslie Koorhan, Robert Canavello, Sharman Staples, Paul Nahay

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

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

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Table of Contents (Overview)

Chapter 1	.NET: What You Need To Know
Chapter 2	First C# Programs
Chapter 3	Data Types in C#
Chapter 4	Operators and Expressions
Chapter 5	Control Structures
Chapter 6	Object-Oriented Programming
Chapter 7	Classes
Chapter 8	More about Types
Chapter 9	Methods, Properties and Operators
Chapter 10	Characters and Strings
Chapter 11	Arrays and Indexers
Chapter 12	Inheritance
Chapter 13	Virtual Methods and Polymorphism
Chapter 14	Formatting and Conversion
Chapter 15	Exceptions
Chapter 16	Interfaces
Chapter 17	.NET Interfaces and Collections
Chapter 18	Delegates and Events
Chapter 19	Introduction to Windows Forms
Chapter 20	Newer Features in C#
Appendix A	Learning Resources

Electronic File Supplements

Supplement1.pdf	Using Visual Studio 2022
Supplement2.pdf	Language Integrated Query (LINQ)
Supplement3.pdf	Unsafe Code and Pointers in C#

Directory Structure

- **The course software installs to the root directory *C:\OIC\CSharp*.**
 - 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 **CaseStudy** directory contains a case study in multiple steps.
 - The **Demos** directory is provided for performing in-class demonstrations led by the instructor.
 - Supplementary course content is provided in PDF files in the **Supplements** directory. Code examples for the supplements are in directories **Supp1**, **Supp2** and **Supp3**.

Table of Contents (Detailed)

Chapter 1 Introduction to .NET	1
What is .NET?.....	3
Libraries and Tools.....	4
Application Models.....	5
Managed Code.....	6
.NET Programming in a Nutshell	7
Visual Studio 2022.....	8
Using Visual Studio.....	9
Using Visual Studio (Cont'd).....	10
Using Visual Studio (Cont'd).....	11
Visual Studio Demo.....	12
Configure Your New Project.....	14
Editor and Solution Explorer.....	15
Project Properties.....	16
Visual Studio Intellisense.....	17
Running the Program.....	18
Build the Project.....	19
Run the Project.....	20
Visual C# and GUI Programs.....	21
.NET Documentation.....	22
Summary.....	23
Chapter 2 First C# Programs	25
Hello, World.....	27
Using the Old Console Template.....	28
Program Structure.....	30
Namespaces.....	32
Echo.....	33
Using InputWrapper.....	34
Namespaces and Using.....	35
Namespaces in Your Own Class.....	36
Variables.....	37
Expressions.....	38
Assignment.....	39
Calculations Using C#.....	40
Sample Program.....	41
More about Output in C#.....	42
Input in C#.....	43
More about Classes.....	44
InputWrapper Class.....	45
Multiple Files in Visual Studio.....	46
The .NET Class Library.....	47
Lab 2.....	49

Summary.....	50
Chapter 3 Data Types in C#.....	53
Strong Typing.....	55
Typing in C#.....	56
Typing in C++.....	57
Typing in Visual Basic 6	58
C# Types.....	59
Integer Types	60
Integer Type Range.....	61
Integer Literals.....	62
Floating-Point Types.....	63
Floating Point Literals	64
IEEE Standard for Floating Point.....	65
Decimal Type.....	66
Decimal Literals.....	67
Character Type.....	68
Character Literals.....	69
string	70
Escape Characters.....	71
Boolean Type.....	72
Implicit Conversions.....	73
Explicit Conversions.....	74
Conversions Example	75
Nullable Types.....	76
Lab 3A	77
Lab 3B.....	78
Summary.....	79
Chapter 4 Operators and Expressions.....	83
Operator Cardinality	85
Arithmetic Operators	86
Multiplication.....	87
Checking	88
Visual Studio Checked Setting	89
Division.....	90
Additive Operators.....	91
Example: A Small Calculator	92
Increment and Decrement.....	93
Relational Operators	94
Conditional Logical Operators.....	95
Short-Circuit Evaluation.....	96
Format Strings and Placeholders	98
Ternary Conditional Operator.....	99
Bitwise Operators	100
Bitwise Logical Operators	101
Bitwise Shift Operators.....	102

Assignment Operators.....	103
Expressions	104
Precedence	105
Associativity	106
Lab 4	107
Summary.....	108
Chapter 5 Control Structures.....	111
Selection Statements	113
If Test.....	114
Blocks	115
switch	116
switch in C#	117
Loops	118
while Loop.....	119
do/while Loops	120
for Loops.....	121
ForUp Example.....	122
ForDown Example.....	123
Arrays.....	124
Fibonacci Example	125
foreach Loop.....	126
break.....	127
continue.....	128
goto	129
Structured Programming.....	130
Structured Search Example.....	131
Multiple Methods.....	132
Lab 5	134
Summary.....	135
Chapter 6 Object-Oriented Programming.....	139
Objects	141
Objects in the Real World.....	142
Object Models.....	143
Reusable Software Components	144
Objects in Software.....	145
State and Behavior	146
Abstraction.....	147
Encapsulation.....	148
Classes	149
Inheritance Concept	150
Inheritance Example	151
Relationships among Classes.....	152
Polymorphism.....	153
Object Oriented Analysis and Design.....	155
Use Cases.....	156

CRC Cards and UML	157
Summary	158
Chapter 7 Classes.....	159
Classes as Structured Data.....	161
Classes and Objects	162
References.....	163
Instantiating and Using an Object.....	164
Assigning Object References.....	165
Garbage Collection	166
Sample Program.....	167
Methods	168
Method Syntax Example.....	169
Public and Private	170
Abstraction.....	172
Encapsulation.....	173
Initialization	174
Initialization with Constructors	175
Default Constructor.....	177
this.....	178
TestAccount Sample Program	179
Static Fields and Methods.....	181
Static Methods	182
Sample Program.....	183
Static Constructor	184
Constant and Readonly Fields	185
Lab 7	186
Summary.....	187
Chapter 8 More about Types.....	191
Overview of Types in C#.....	193
Structures	194
Uninitialized Variables	195
Test Program.....	196
Copying a Structure	197
Hotel.cs	198
Test Program for Hotel Copy.....	199
Results of Hotel Copy.....	200
Classes and Structs.....	201
Enumeration Types	202
Enumeration Types Examples	203
Reference Types	204
Class Types	205
object.....	206
string	207
Arrays.....	208
Default Values	209

Boxing and Unboxing	211
Implicitly Typed Variables	212
Implicitly Typed Variables – Example	213
Lab 8	214
Summary	215
Chapter 9 Methods, Properties, and Operators.....	217
Static and Instance Methods	219
Method Parameters	220
No “Freestanding” Functions in C#.....	221
Classes with All Static Methods	222
Parameter Passing	223
Parameter Terminology	224
Value Parameters	225
Reference Parameters	226
Output Parameters.....	229
Structure Parameters	230
Class Parameters.....	231
Method Overloading	232
Lab 9A	234
Modifiers as Part of the Signature	235
Variable Length Parameter Lists	236
Properties	237
Properties Examples	238
Auto-Implemented Properties.....	241
Auto-Implemented Property Example.....	242
Lab 9B.....	243
Operator Overloading	244
Sample Program.....	247
Operator Overloading in the Class Library.....	248
Summary	249
Chapter 10 Characters and Strings	253
Characters	255
Sample Program.....	256
Character Codes.....	257
ASCII and Unicode.....	258
Escape Sequences	259
Strings	260
String Class	261
String Literals and Initialization	262
Concatenation	263
Index	264
Relational Operators	265
String Equality	266
String Comparisons.....	267
String Comparison	268

String Input	270
String Methods and Properties.....	271
StringBuilder Class	273
StringBuilder Equality	275
Command Line Arguments.....	276
Command Line Arguments in the IDE	277
Command Loops.....	278
Splitting a String	279
Lab 10	280
Summary.....	281
Chapter 11 Arrays and Indexers.....	285
Arrays.....	287
One Dimensional Arrays	288
System.Array	289
Sample Program.....	290
Random Number Generation	291
Next Methods.....	292
Jagged Arrays	293
Rectangular Arrays	294
Arrays as Collections	295
Bank Case Study: Step 1	297
Account Class	299
Bank Class	302
TestBank Class	305
Atm Class.....	307
Running the Case Study.....	309
Indexers.....	311
ColorIndex Example Program	313
Using the Indexer.....	314
Lab 11	315
Summary.....	316
Chapter 12 Inheritance	319
Inheritance Fundamentals	321
Inheritance in C#.....	322
Single Inheritance	323
Root Class – <i>Object</i>	324
Access Control.....	325
Public Class Accessibility.....	326
Internal Class Accessibility	327
Member Accessibility	328
Member Accessibility Qualifiers	329
Member Accessibility Example.....	330
Method Hiding.....	331
Method Hiding and Overriding.....	332
Example: Method Hiding.....	333

Initialization	334
Initialization Fundamentals.....	335
Initialization Fundamentals Example	336
Default Constructor.....	337
Overloaded Constructors	338
Example: Overloaded Constructors.....	339
Invoking Base Class Constructors	340
Base Class Initialization Example	341
Bank Case Study: Step 2.....	342
Bank Case Study Analysis.....	343
Account.....	344
CheckingAccount.....	345
SavingsAccount	347
TestAccount.....	349
Running the Case Study.....	350
Lab 12	351
Summary.....	352
Chapter 13 Virtual Methods and Polymorphism	355
Introduction to Polymorphism.....	357
Abstract and Sealed Classes.....	358
Virtual Methods and Dynamic Binding.....	359
Type Conversions in Inheritance.....	360
Converting Down the Hierarchy.....	361
Converting Up the Hierarchy.....	362
Virtual Methods	363
Virtual Method Example	364
Virtual Method Cost	365
Method Overriding	366
The Fragile Base Class Problem.....	367
<i>override</i> Keyword.....	368
Polymorphism.....	369
Polymorphism Using “Type Tags”.....	370
Polymorphism Using Virtual	371
Polymorphism Example.....	372
Abstract Classes.....	376
Keyword: <i>abstract</i>	377
Sealed Classes.....	378
Heterogeneous Collections	379
Heterogeneous Collections Example.....	380
Bank Case Study: Step 3.....	381
Case Study Classes	382
Run the Case Study.....	384
Account.....	385
CheckingAccount, SavingsAccount	386
Bank and Atm.....	387

TestBank	388
Lab 13	389
Summary	390
Chapter 14 Formatting and Conversion.....	393
Introduction to Formatting.....	395
ToString	396
ToString in Your Own Class	397
Using Placeholders	399
Format Strings.....	400
Simple Placeholders.....	401
Controlling Width.....	402
Format String.....	403
Currency.....	404
Currency Format Example.....	405
String.Format	406
PadLeft and PadRight.....	407
Bank Case Study: Step 4.....	409
Type Conversions.....	410
Conversion of Built-In Types	411
Conversion of User-Defined Types.....	412
User Defined Conversions: Example.....	414
Lab 14	416
Summary.....	417
Chapter 15 Exceptions.....	419
Introduction to Exceptions.....	421
Exception Fundamentals.....	422
.NET Exception Handling.....	423
Exception Flow of Control	424
Context and Stack Unwinding.....	425
Exception Example.....	426
System.Exception	429
User-Defined Exception Classes	430
User Exception Example	431
Structured Exception Handling.....	434
Finally Block.....	435
Bank Case Study: Step 5.....	437
Inner Exceptions	438
Checked Integer Arithmetic.....	439
Example Program	440
Lab 15	441
Summary.....	442
Chapter 16 Interfaces	445
Introduction.....	447
Interfaces in C#.....	449

Interface Inheritance	450
Programming with Interfaces.....	451
Implementing Interfaces	452
Using an Interface	454
Demo: SmallInterface.....	455
Dynamic Use of Interfaces	456
Demo: TryInterfaces	457
is Operator.....	458
as Operator.....	459
Bank Case Study: Step 6.....	460
Common Interfaces in Case Study –IAccount.....	461
Apparent Redundancy.....	462
IStatement	463
IStatement Methods	464
IChecking.....	465
ISavings	466
The Implementation.....	467
SavingsAccount	468
The Client	469
Resolving Ambiguity.....	471
Access Modifier	472
Explicit Interfaces Test Program.....	473
Summary.....	474
Chapter 17 .NET Interfaces and Collections.....	475
Overview.....	477
Collections	478
ArrayList Example.....	479
Count and Capacity.....	480
foreach Loop.....	481
Array Notation	482
Adding to the List	483
Remove Method.....	484
RemoveAt Method.....	485
Collection Interfaces.....	486
IEnumerable and IEnumerator.....	487
IEnumerable and IEnumerator Demo: <i>AccountList</i>	488
ICollection	489
IList.....	490
A Collection of User-Defined Objects.....	491
Duplicate Objects.....	492
A Correction to AccountList (Step 1).....	493
Bank Case Study: Step 7.....	494
Copy Semantics and ICloneable	495
Copy Semantics in C#.....	496
Shallow Copy and Deep Copy.....	497

Example Program	498
Reference Copy.....	499
Memberwise Clone.....	500
Using ICloneable	501
Comparing Objects	502
Sorting an Array.....	503
Anatomy of Array.Sort	504
Using the is Operator	505
The Use of Dynamic Type Checking	506
Implementing IComparable	507
Running the Program.....	508
Complete Solution	509
Lab 17A	510
Writing Generic Code.....	511
Using a Class of <i>object</i>	512
Generic Types	513
Generic Syntax in C#.....	514
Generic Example.....	515
Generic Client Code.....	516
System.Collections.Generic.....	517
Lab 17B.....	518
Object Initializers.....	519
Collection Initializers.....	520
Anonymous Types	521
Summary.....	522
Chapter 18 Delegates and Events.....	527
Overview of Delegates and Events.....	529
Callbacks and Delegates	530
Usage of Delegates	531
Declaring a Delegate.....	532
Defining a Method.....	533
Creating a Delegate Object.....	534
Calling a Delegate.....	535
Random Number Generation	536
A Random Array.....	537
Anonymous Methods.....	538
Combining Delegate Objects	539
Account.cs.....	540
DelegateAccount.cs	541
Lambda Expressions.....	542
Named Method	543
Anonymous Method	544
Lambda Expression Example	545
Events.....	546
Events in C# and .NET	547

Client Side Event Code.....	549
Chat Room Example.....	550
Lab 18	551
Summary.....	552
Chapter 19 Introduction to Windows Forms.....	555
Windows Forms Demo	557
Partial Classes.....	563
Windows Forms Event Handling.....	564
Add Events for a Control.....	565
Events Documentation.....	566
Closing a Form.....	567
ListBox Control.....	568
ListBox Example	569
Lab 19	570
Summary.....	571
Chapter 20 Newer Features in C#.....	575
<i>dynamic</i> Type.....	577
Runtime Error Example.....	578
<i>dynamic</i> versus <i>object</i>	579
Behavior of <i>object</i>	580
Behavior of <i>dynamic</i>	581
Named Arguments	582
Optional Arguments.....	583
Book Class	584
Using Optional Arguments.....	585
Variance in Generic Interfaces	586
Covariance Example.....	587
Variance with <code>IComparer<T></code>	588
Interfaces with Variance Support	589
Contravariance Example.....	590
Asynchronous Programs in C# 5	591
<code>Task</code> and <code>Task<TResult></code>	592
Aysnc Methods	593
Async Example.....	594
Synchronous Call.....	595
Async Call.....	596
Threading.....	597
New Features in C# 6.....	598
Null-Conditional Operator.....	599
Composite Format String.....	600
Interpolated Strings.....	601
New Features in C# 7.....	602
Tuples.....	603
Nullable Reference Types in C# 8.....	604
Nullable Reference Example Program	605

Record Type in C# 9.....	607
Example Program for Record Type.....	608
Top-Level Statements.....	610
Summary.....	611
Appendix A Learning Resources.....	613

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Chapter 1

Introduction to .NET

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Introduction to .NET

Objectives

After completing this unit you will be able to:

- **Give a high-level overview of .NET, with a focus on .NET 6.0, the basis of .NET implementation going forward.**
- **Outline the architectural components of .NET.**
- **Describe the essentials of creating and running a program in the .NET environment.**
- **Build and run a simple C# program.**
- **Use Visual Studio 2022 as an effective environment for creating C# programs.**
- **Use the .NET documentation.**

What is .NET?

- **.NET is Microsoft's software platform for building applications for many environments.**
 - Originally proprietary and restricted to Microsoft Windows, .NET is now open source and cross-platform.
- **.NET applications can be built using multiple languages.**
 - C# and Visual Basic are object-oriented languages.
 - F# is a functional language which also supports object-oriented and imperative programming.
- **.NET Framework is the original implementation of .NET, running on Windows.**
- **.NET Core is a package-based implementation that is cross-platform, running on Mac and Linux besides Windows.**
- **.NET 6 is the successor of .NET Core.**
 - It is the basis of .NET implementation going forward.
- **.NET Framework 4.8 is the latest version of the classical .NET Framework.**
 - It will continue to be distributed with future releases of Windows. As long as it is installed on a supported version of Windows, .NET Framework 4.8 will continue to also be supported.

Libraries and Tools

- **Microsoft and third parties provide many libraries to extend .NET.**
 - Many libraries are furnished through NuGet packages.
 - NuGet is a package manager specifically designed for .NET.
- **Visual Studio is a full-featured IDE running on Windows for building all types of .NET applications.**
 - Visual Studio 2022 is required for .NET 6.
- **Visual Studio for Mac runs on Mac computers.**
 - It can be used for building iOS and macOS.
 - It can also be used for ASP.NET Core apps and services.
- **Visual Studio Code is a lightweight code editor available for Windows, macOS and Linux.**
 - It comes with built in support for JavaScript and extensions for many languages, including C#, C++, Java, Python, PHP, and others.

Application Models

- **Web applications and services**

- Classical ASP.NET supports web applications on Windows.
- ASP.NET Core is cross-platform, targeting Windows, macOS and Linux.

- **Mobile applications**

- Cross-platform targeting iOS, Android and Windows

- **Desktop**

- Windows desktop applications using Windows Forms or Windows Presentation Foundation (WPF)
- Mac desktop applications

- **Microservices -- a design pattern in which apps are composed of small, independent modules.**

Managed Code

- .NET apps run *managed code*.
- Compilers for .NET languages generate *intermediate language (IL)*.
- An IL program does not run directly on hardware but on a virtual machine, called a *runtime*.
 - The .NET runtime is known as the Common Language Runtime, or CLR.
- The runtime provides various services such as automatic memory management, type safety, and so on.

.NET Programming in a Nutshell

Write your program in a high-level .NET language, such as C#.

Compile your program into IL.

Run your IL program, which will launch the runtime (CLR) to execute your IL, using its just-in-time compiler to translate your program to native code as it executes.

- **We will look at a simple example of a C# program and run it under .NET.**

- See **Chap01\SimpleCalc**

- Here is the code:

```
int width = 20;
int height = 5;
int area;
area = width * height;
Console.WriteLine("width = {0}", width);
Console.WriteLine("height = {0}", height);
Console.WriteLine("area = {0}", area);
```

- Here is the output when the program is run:

```
width = 20
height = 5
area = 100
```

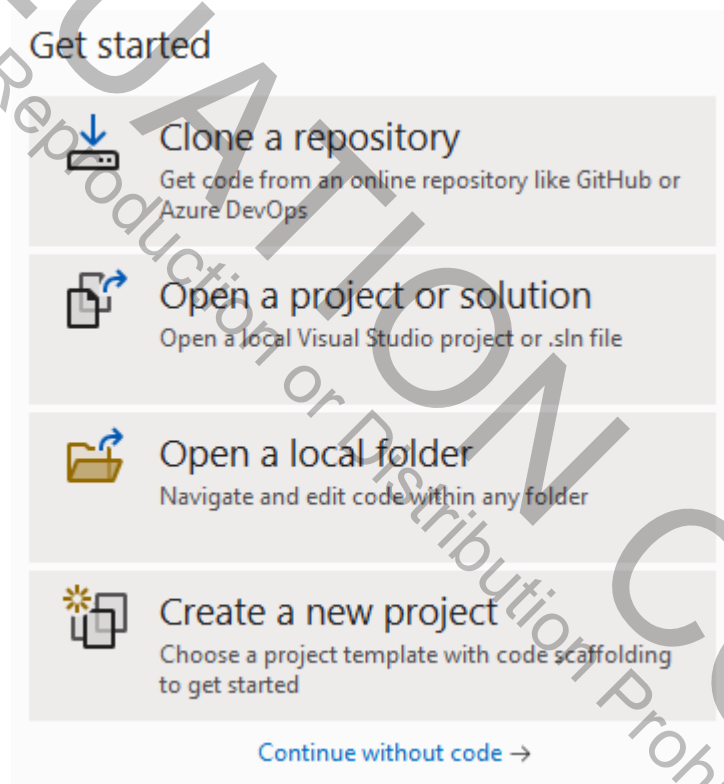
Visual Studio 2022

- **While it is possible to write C# programs using any text editor, and compile them with the command-line compiler, it is very tedious to program that way.**
- **An IDE makes the process of writing software much easier.**
 - An IDE provides convenience items, such as a syntax-highlighting editor.
 - An IDE reduces the tedium of keeping track of configurations, environment settings and file organizations.
- **You may use Visual Studio 2022 throughout this course to create and compile your C# programs.**
- **Visual Studio 2022 is discussed in more detail in Supplement 1.**
- **In this course you may use any version of VS 2022, including the free Visual Studio 2022 Community.**

Using Visual Studio

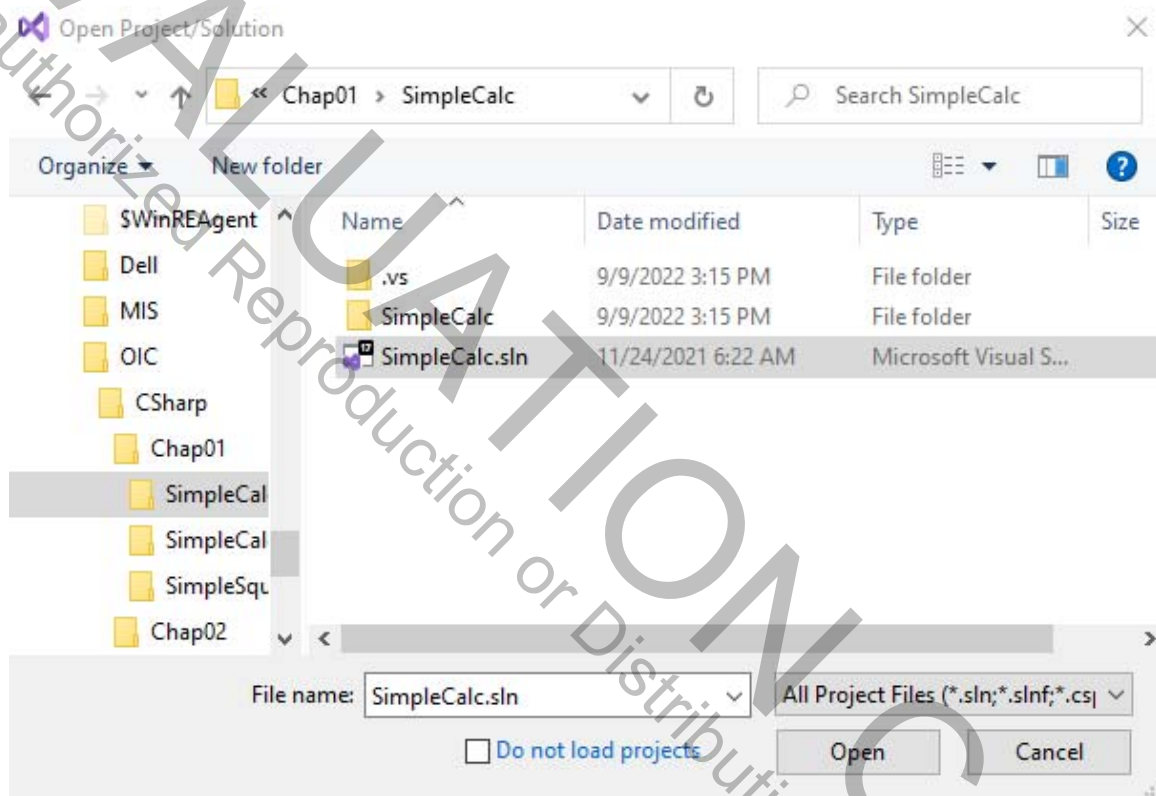
- **First, let's see how to open and run an existing program.**

1. From the Visual Studio start page click on "Open a project or solution". This will bring up the New Project dialog. (You can also open a project from the menu File | Open | Project/Solution.)



Using Visual Studio (Cont'd)

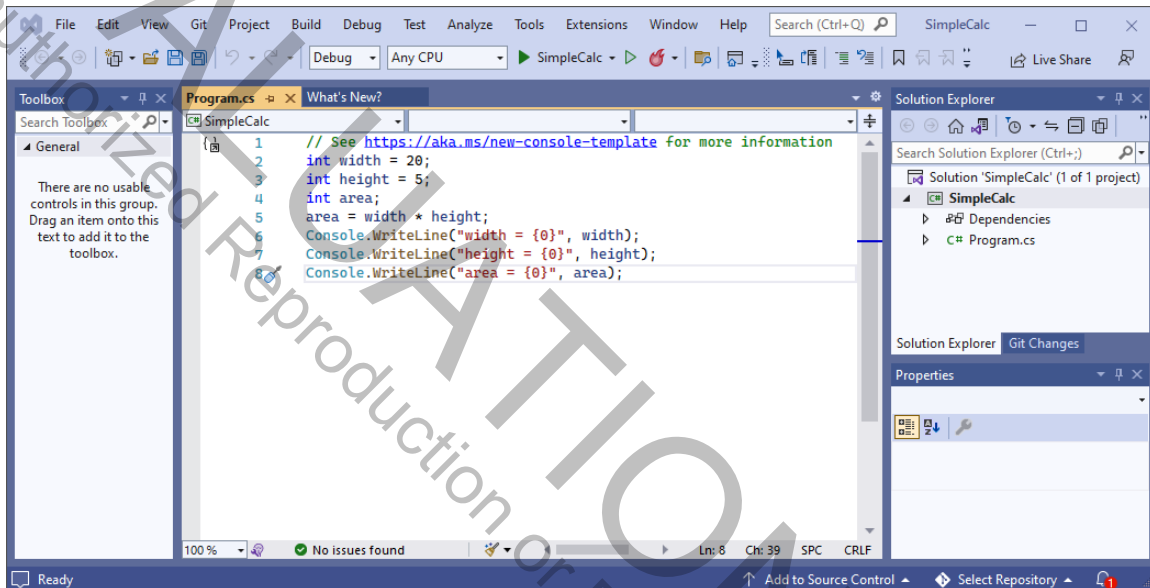
- This will bring up the Open Project/Solution dialog. Navigate to the **SimpleCalc** folder in **CSharp\Chap01**. Select the solution file **SimpleCalc.sln**.



- Click Open.

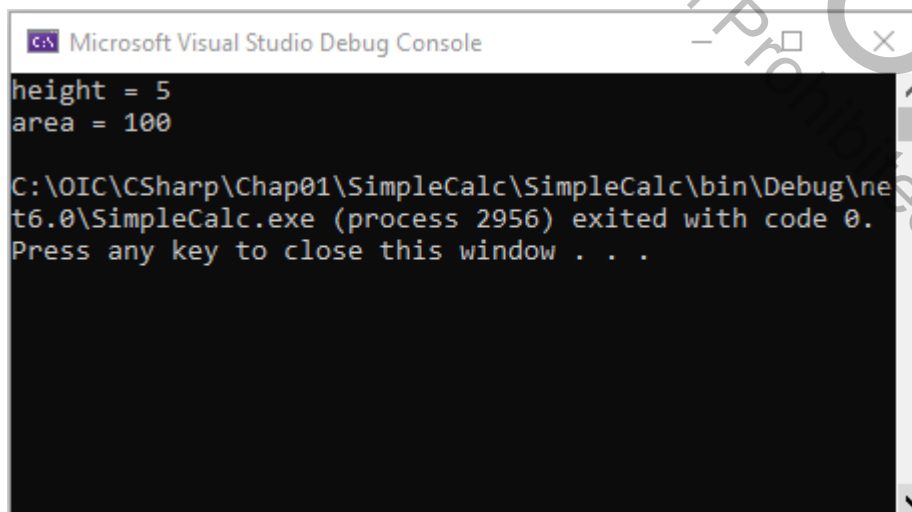
Using Visual Studio (Cont'd)

- In the middle pane you will see the program open in the Visual Studio code editor, and Solution Explorer is on the right.



- Now let's run the program, without debugging.

- The easiest way to do that is with the open green wedge on the toolbar.

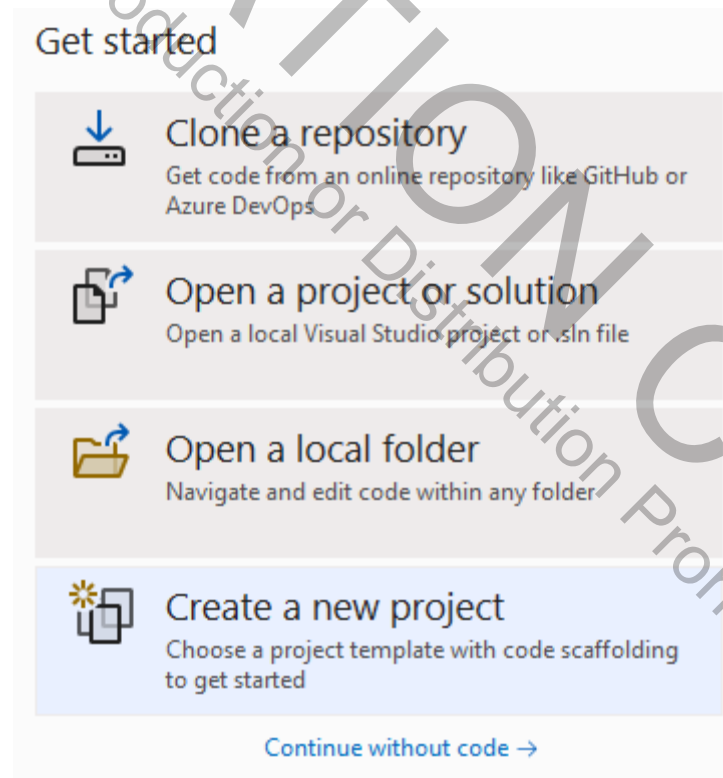


Visual Studio Demo

- **We will now create a simple console application using Visual Studio.**

– Our program will be similar to the example we looked at, only simpler and requiring less typing! It simply calculates the area of a square.

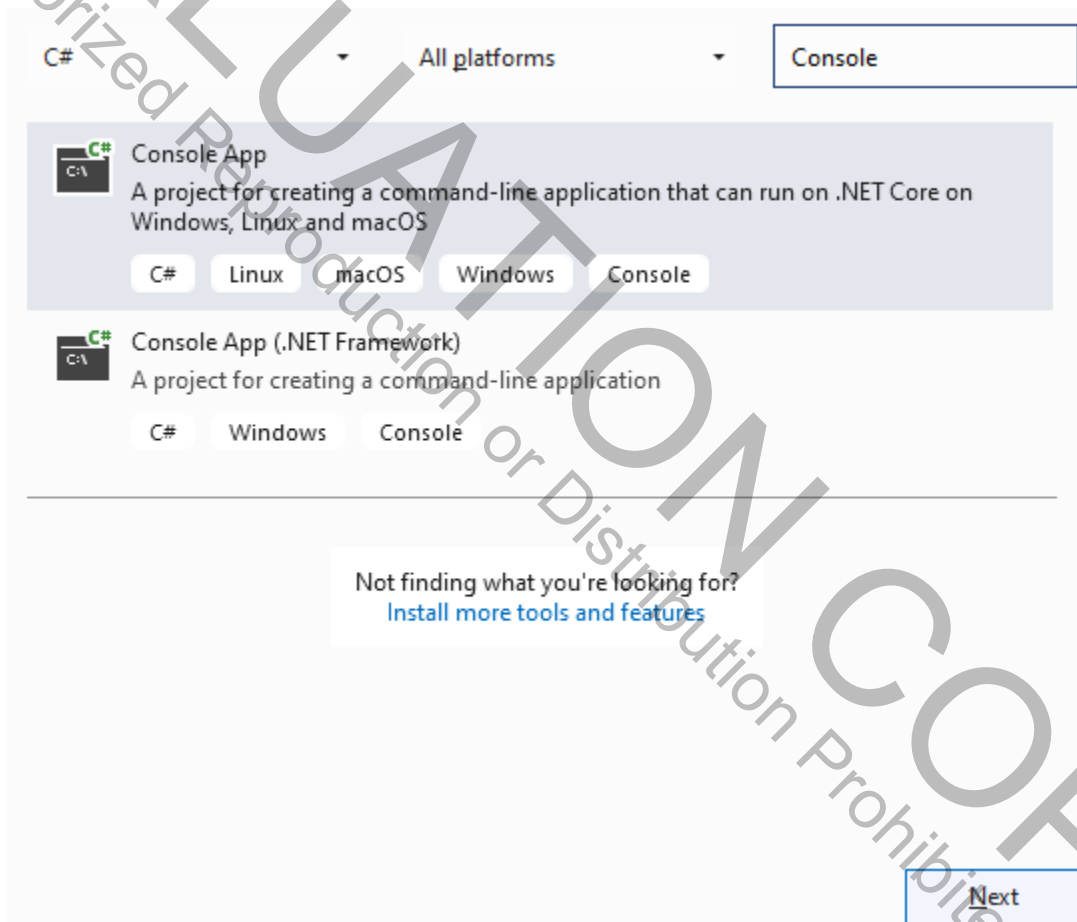
1. From the Visual Studio start page click on "Create a new project". This will bring up the New Project dialog. (You can also use the menu File | New | Project when the main Visual Studio window is open.)



Creating a Console App (Cont'd)

2. Choose Console App with language C#. We filtered the language to be C# and the project type to be Console.

- Note that Console App (.NET Framework) would generate an app using the classical .NET Framework that runs only on Windows.



3. Click Next.

Configure Your New Project

- In the Project name field, type **SimpleSquare** and for Location browse to **C:\OIC\CSharp\Demos**. Leave Solution name as SimpleSquare. Leave unchecked "Place solution and project in the same directory."

Configure your new project

Console App C# Linux macOS Windows Console

Project name

SimpleSquare

Location

C:\OIC\CSharp\Demos

Solution

Create new solution

Solution name ⓘ

SimpleSquare

Place solution and project in the same directory

- Click Next. Under “Additional Information” accept the suggested .NET 6.0 (Long-term support) for Framework. Leave unchecked “Do not use top-level statements”.

Additional information

Console App C# Linux macOS Windows Console

Framework ⓘ

.NET 6.0 (Long-term support)

Do not use top-level statements ⓘ

Editor and Solution Explorer

- You will see the file *Program.cs* displayed in an editor window.

– If you are familiar with .NET from earlier versions, you are in for a big surprise!

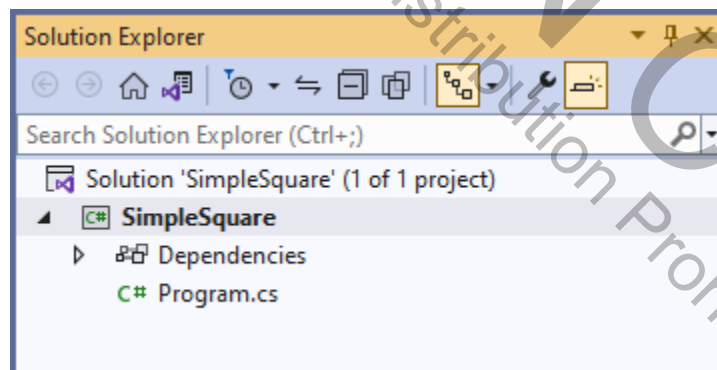


The screenshot shows a code editor window titled 'Program.cs'. The editor contains the following code:

```
1 // See https://aka.ms/new-console-template for more information
2 Console.WriteLine("Hello, World!");
3
```

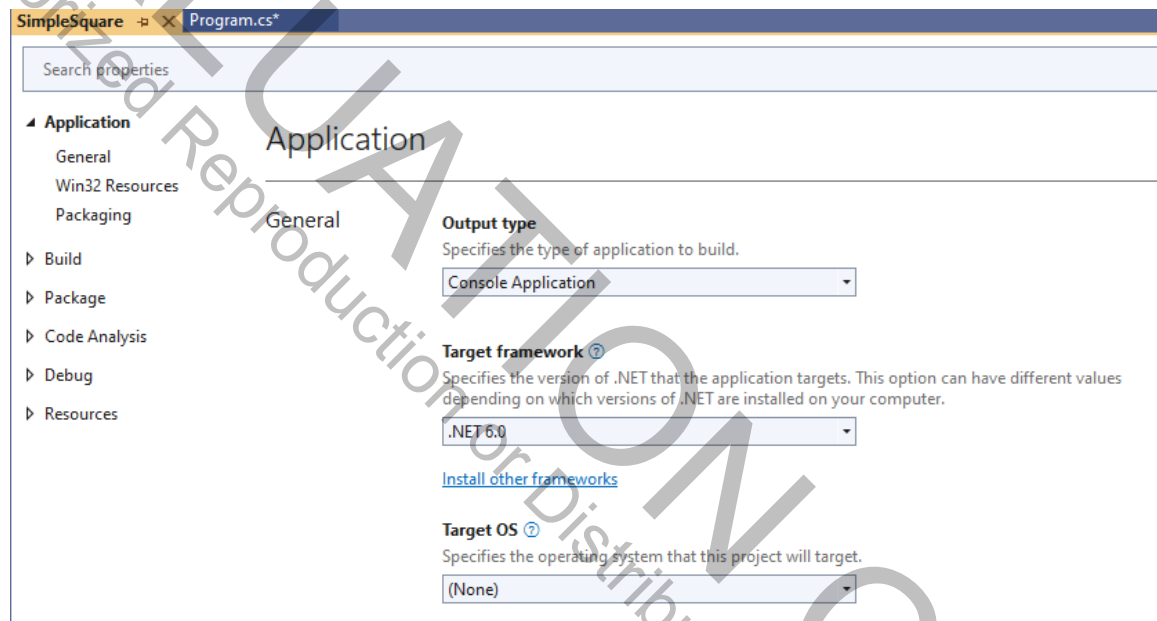
– There is a new template for console programs. You can follow the link for more information, and we will discuss it in the next chapter.

- On the right you will see Solution Explorer open.



Project Properties

- Right-click over the **SimpleSquare** project (shown highlighted in the screen capture) and select Properties at the bottom of the context menu.
- Observe that .NET 6.0 is the Target framework, under the Application | General properties.



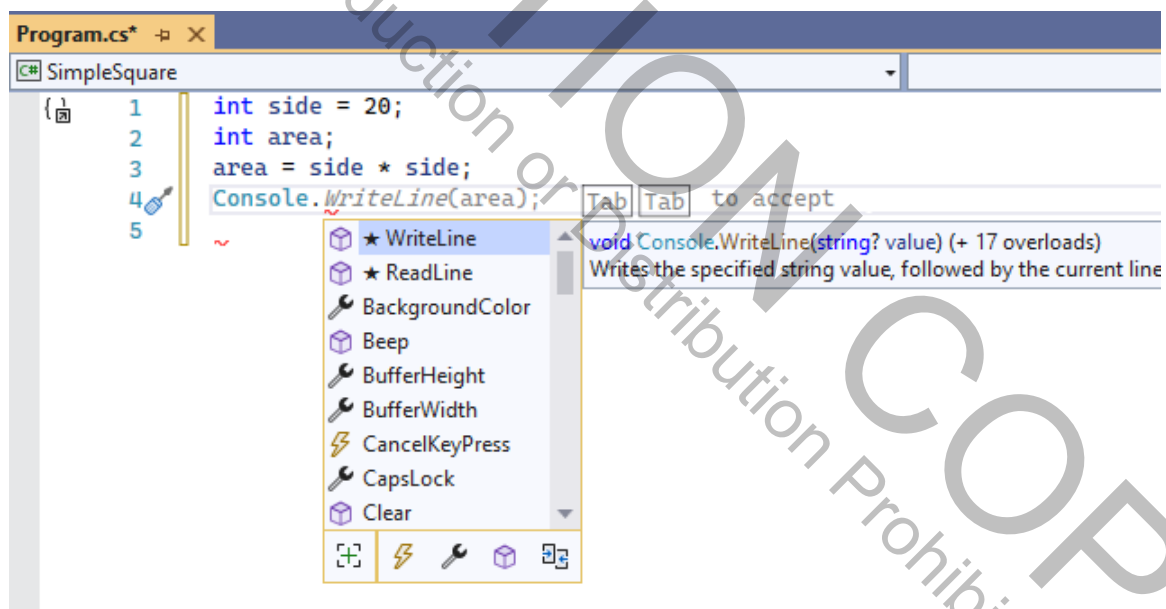
- Don't worry about the other project properties for now.
- The Properties window is set up to work with the build system and to have convenient default properties.
- Close the project file and return to **Program.cs**, in the Visual Studio code editor.

Visual Studio Intellisense

- The file *Program.cs* will be open in the Visual Studio text editor. Replace the starter code by the following, which you can type in.

```
int side = 20;
int area;
area = side * side;
Console.WriteLine("side = " + side);
Console.WriteLine("area = " + area);
```


- Notice what happens as type *Console* followed by a period.

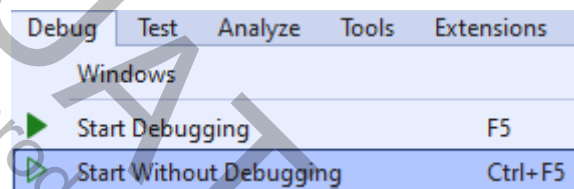


- A powerful feature of Visual Studio is *IntelliSense*.
 - IntelliSense will automatically pop up a list box allowing you to easily insert language elements directly into your code.

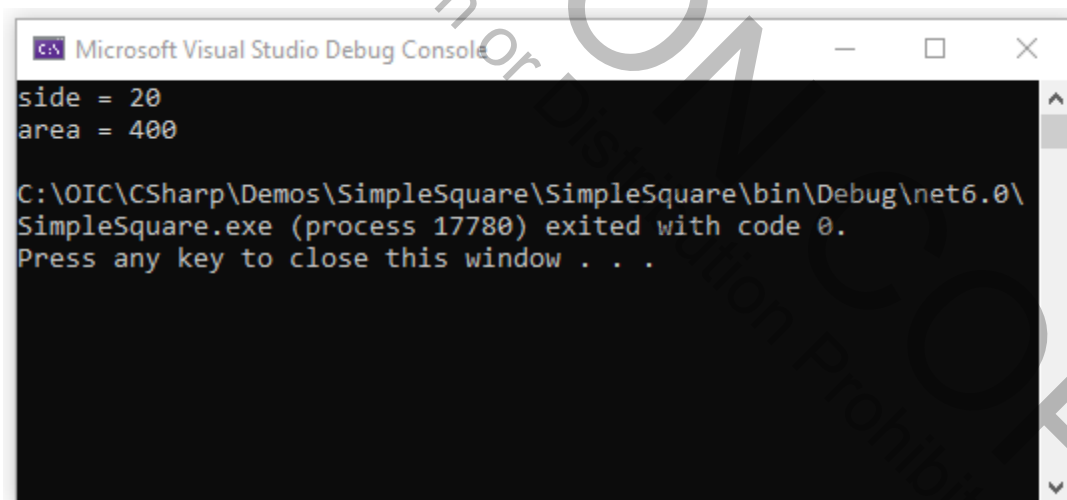
Running the Program

- **After you have typed in the complete program, you can run it without debugging in several ways:**

- Using the open green wedge on the toolbar 
- Using the keyboard shortcut Ctrl + F5
- From the menu Debug | Start Without Debugging



- **This is the result:**

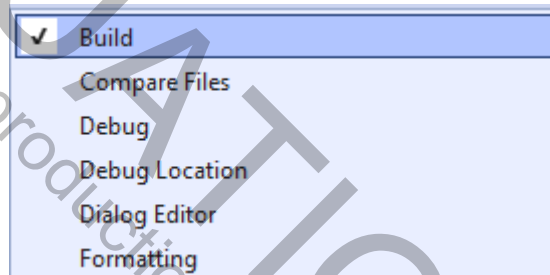




```
Microsoft Visual Studio Debug Console
side = 20
area = 400

C:\OIC\CSharp\Demos\SimpleSquare\SimpleSquare\bin\Debug\net6.0\
SimpleSquare.exe (process 17780) exited with code 0.
Press any key to close this window . . .
```

Build the Project

- **Building a project means compiling the individual source files and linking them together with any library files to create an IL executable .EXE file.**
- **To make it easier to build, add the Build toolbar (if it is not already present) by a right-click over the toolbar area. Check Build.**




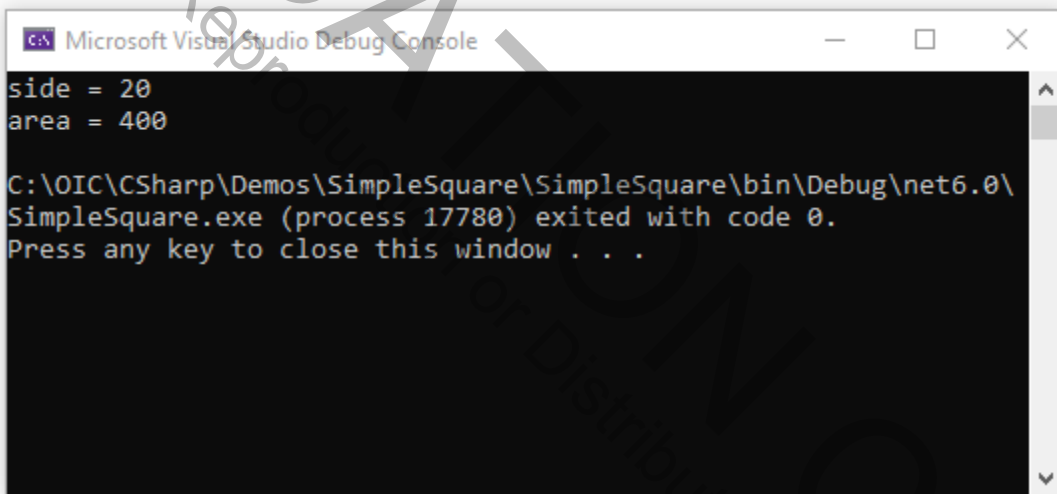
- This will add the Build toolbar to the Standard and Text Editor toolbars that are shown by default.
- **Then you can build the project by using one of the following:**
 - Menu Build | Build Solution or toolbar button  or keyboard shortcut Ctrl+Shift+B. (This builds all the projects in the solution.)
 - Menu Build | Build SimpleSquare or toolbar button . (This just builds the project SimpleSquare)¹.
- **Both forms of Start (see next page) will automatically build the project if not already built.**

¹ The two are the same in this case, because the solution has only one project, but some solutions have multiple projects, and then there is a difference.

Run the Project

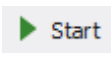
- **You can run the program without the debugger by using one of the following:**

- Menu Debug | Start Without Debugging
- Toolbar button  (This button is provided by default in Visual Studio 2022.)
- Keyboard shortcut Ctrl + F5



```
Microsoft Visual Studio Debug Console
side = 20
area = 400

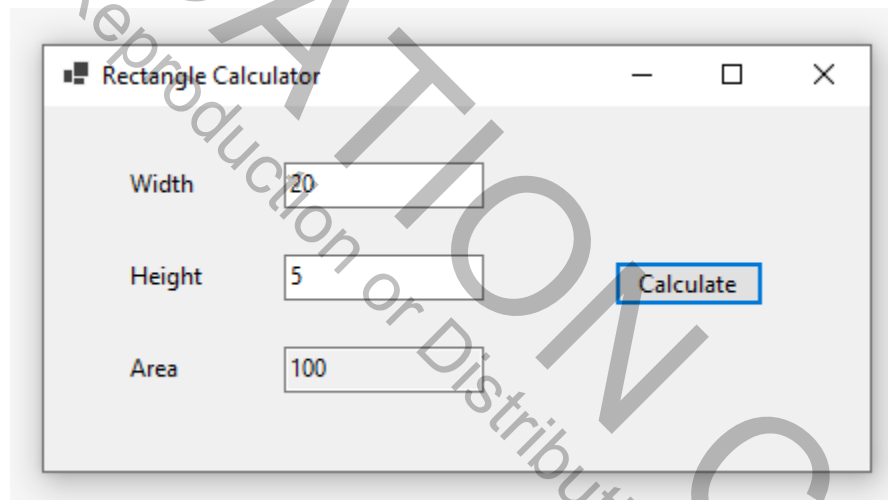
C:\OIC\CSharp\Demos\SimpleSquare\SimpleSquare\bin\Debug\net6.0\
SimpleSquare.exe (process 17780) exited with code 0.
Press any key to close this window . . .
```

- **You can run the program in the debugger by using one of the following:**
 - Menu Debug | Start Debugging
 - Toolbar button 
 - Keyboard shortcut F5.
- **We will discuss debugging later.**

Visual C# and GUI Programs

- Microsoft's implementation of the C# language, **Visual C#**, works very effectively in a **GUI environment**.
 - Using Windows Forms (available in .NET Core with .NET 5.0 and above), it is easy to create Windows GUI programs in C#.

Example: See **Chap01\SimpleCalcGui**



- We will discuss **GUI programming using C#** in **Chapter 19**.

.NET Documentation

- .NET documentation is available online.
- It is now part of comprehensive Microsoft documentation at

<https://docs.microsoft.com>

- Select .NET.

Thank you for downloading Vis X .NET documentation | Microsoft X









https://docs.microsoft.com/en-us/dotnet/

Microsoft | Docs Documentation Learn Q&A Code Samples More v Search Sign in

.NET Languages v Workloads v APIs v Resources v Download .NET

.NET documentation

Learn to use .NET to create applications on any platform using C#, F#, and Visual Basic. Browse API reference, sample code, tutorials, and more.

 DOWNLOAD Download .NET ↗	 LEARN Build .NET apps with C#
 TUTORIAL Create your first console app	 LEARN Create your first web app ↗
 LEARN Browse .NET learning paths	 GET STARTED Interactive introduction to C#
 WHAT'S NEW What's new in .NET docs	 OVERVIEW Azure for .NET developers

Summary

- **.NET is Microsoft's software platform for building applications for many environments.**
- **.NET applications can be built using multiple languages, including C#.**
 - C# is an object-oriented language designed by Microsoft for .NET.
- **The basis of .NET going forward is .NET 6.0, the most recent version of the cross-platform .NET Core.**
- **The classical .NET Framework 4.8 is being maintained.**
- **Visual Studio is a full-featured IDE running on Windows for building all types of .NET applications.**
- **With Visual Studio it is easy to create and run programs using C#.**
- **Windows Forms is supported in .NET 6.0 as an easy means of creating GUI programs running on Windows.**
- **You can access extensive .NET documentation online.**

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

Chapter 2

First C# Programs

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

First C# Programs

Objectives

After completing this unit you will be able to:

- **Write a basic “Hello, World” program in C# with and without top-level statements.**
- **Explain the difference between the old and the new console template.**
- **Describe the basic structure of C# programs.**
- **Describe how related C# classes can be grouped into namespaces and the purpose of the *using* statement.**
- **Use variables and simple expressions in C# programs.**
- **Write C# programs that can perform simple calculations.**
- **Perform simple input and output in C#.**
- **Describe objects and classes in C#.**
- **Use an input wrapper class to perform input in C#.**

Hello, World

- **Whenever learning a new programming language, a good first step is to write and run a simple program that will display a single line of text.**
 - Such a program demonstrates the basic structure of the language, including output.
 - You must learn the pragmatics of compiling and running the program.
- **When you create a new C# Console application Visual Studio will create a Hello World program for you!**
 - We have added our own comment to the starter code.

```
// See https://aka.ms/new-console-template  
// for more information  
// Hello World demo  
Console.WriteLine("Hello World!");
```

- This uses the new console template, which is the default with Visual Studio 2022. You can click on the link for more information.
- See **Hello\New** in the **Chap02** directory.

Using the Old Console Template

- **Now let's create the same Hello World program without using top-level statements.**
 - We'll use Visual Studio 2022 just like in the previous chapter, but with one difference along the way.
 1. From the Visual Studio start page click on "Create a new project". If Visual Studio is already running, use the menu File | New | Project ...
 2. Choose Console App.
 3. Browse to **Demos** for the location and type **Hello** as the project name. Click Next.
 4. In the next window check "Do not use top-level statements".

Additional information

Console App C# Linux macOS Windows Console

Framework ⓘ

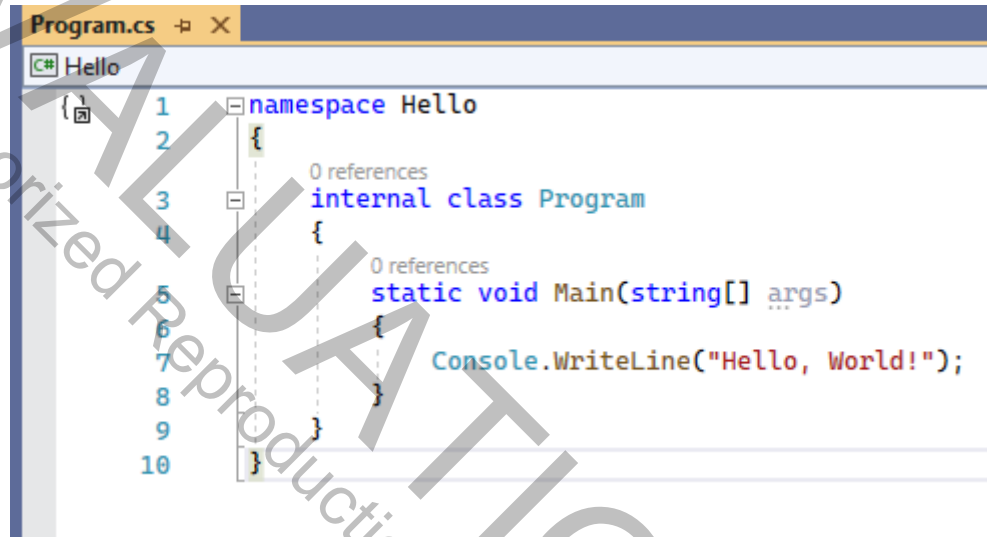
.NET 6.0 (Long-term support)

Do not use top-level statements ⓘ

5. Click Create.

Old Console Template (Cont'd)

6. Examine **Program.cs** in the project that Visual Studio created for you



```
1 namespace Hello
2 {
3     internal class Program
4     {
5         static void Main(string[] args)
6         {
7             Console.WriteLine("Hello, World!");
8         }
9     }
10 }
```

- This program has several additional program elements, which we will examine in this course. For now, we will focus on the key elements of program structure.
- Again, we will provide a comment.

```
// Hello World demo
```

- This version of the Hello World program is saved in **Chap02\Hello\Old**.

Program Structure

```
// Hello World demo
...
internal class Program
{
    ...
}
...
```

- **Every C# program has at least one *class*¹.**
 - A class is the foundation of C#'s support of object-oriented programming.
 - A class encapsulates data (represented by **variables**) and behavior (represented by **methods**).
 - All the code defining the class (its variables and methods) will be contained between the curly braces.
 - We will discuss classes in detail later, including use of the modifier **internal**.
- **Note the *comment* at the beginning of the program.**
 - A line beginning with a double slash is present only for documentation purposes and is ignored by the compiler.
- **C# files have the extension *.cs*.**

¹ With the New Console Template in Visual Studio 2022 the compiler will generate a class,

Program Structure (Cont'd)

```
...
internal class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
...
```

- **Every method in C# has one or more *statements*.**
- **A statement is terminated by a semicolon.**
 - A statement may be spread out over several lines.
- **The *Console* class provides support for standard output and standard input.**
 - The method **WriteLine()** displays a string, followed by a new line.

Namespaces

- **Much standard functionality in C# is provided through many classes in the .NET class library.**
- **Related classes are grouped into *namespaces*.**
- **The old console template explicitly shows the namespace in code.**

```
namespace Hello
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

- **Note: For brevity we will sometimes omit the enclosing namespace in code listings. Thus**

```
internal class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

Echo

- **The Hello program illustrated console output.**
- **The Echo program illustrates both console input and output.**
 - The program prompts the user for a name, and then prints out a personalized greeting.
 - See **Chap02\Echo**.
- **This project has two files.**
 - **InputWrapper.cs** defines a wrapper class that simplifies input.
 - **Program.cs** provides the main code, using the new console template.

```
// Echo
//
// Prompts user to enter name and then
// prints out greeting using name
```

```
using OI;
```

```
InputWrapper iw = new InputWrapper();
string name = iw.getString("Enter your name: ");
Console.WriteLine("Hello, " + name);
```

Using InputWrapper

- **The bolded statements illustrate how to use the *InputWrapper* class.**
 - Instantiate an **InputWrapper** object **iw** by using **new**.
 - Prompt to obtain input data by calling the appropriate **getXXX()** method.
- **You may examine the code in the file *InputWrapper.cs* if you wish.**
 - But you don't have to understand the implementation of the **InputWrapper** class to use it!
 - That is the case with the .NET class library—the beauty of encapsulation.

Namespaces and Using

- **The fully qualified name of a class is specified by the namespace, followed by a dot, followed by class name.**

```
System.Console
```

- **A *using* statement allows a class to be referred to by its class name alone.**
 - Examine **Chap02\Hello\Vs2019**, a project created originally using Visual Studio 2019 and .NET 5.0.

```
using System;
```

```
namespace Hello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

- If you comment out the **using** statement, you will get a compiler error. You could correct it by using the fully qualified name of the **Console** class.

```
System.Console.WriteLine("Hello World!")
```

- **.NET 6.0, used in Visual Studio 2022, has a new feature of *implicit using* statements.**
 - Common namespaces are provided automatically.

Namespaces in Your Own Class

- **When you define a class, it is usually a good idea to place it within a namespace.**

- We did this for the **InputWrapper** class, putting in the namespace **OI**.

```
namespace OI
{
    public class InputWrapper
    {
        public int getInt(string prompt)
        {
            Console.Write(prompt);
            string buf = Console.ReadLine();
            return Convert.ToInt32(buf);
        }
        ...
    }
}
```

- **Then in a code file where you make use of the class, provide a *using* statement.**

- See **Program.cs** in **Chap02\Echo**.

```
using OI;
```

```
InputWrapper iw = new InputWrapper();
string name = iw.getString("Enter your name: ");
Console.WriteLine("Hello, " + name);
```

Variables

- In C#, you can define *variables* to hold data.
- Variables represent storage locations in memory.
- In C#, variables are of a specific data *type*.
 - Some common types are **int** for integers and **double** for floating point numbers.
 - You must declare variables before you can use them.
- A variable declaration reserves memory space for the variable and may optionally specify an initial value.

```
int kilo = 1024; // reserves space and assigns
                // an initial value
int mega;      // reserves space but does
                // not initialize
```

- If a variable is not initialized in its declaration, it should be assigned prior to being used.

```
int kilo;
kilo = 1024;
// Now you may use kilo
```

Expressions

- You can combine variables and constants (or “literals”) via *operators* to form *expressions*.
- Examples of operators include the standard arithmetic operators:

+	addition
-	subtraction
*	multiplication
/	division

- Here are some examples of expressions:

```
kilo * 1024  
(fahrenheit - 32) * 5 / 9  
3.1416 * radius * radius
```

Assignment

- **You can assign a value to a variable by using the = symbol.**
 - On the left-hand side is a variable.
 - On the right-hand side is an expression.
 - The expression is evaluated, and its value is assigned to the variable on the left.
 - Assignment is a statement and must be terminated by a semicolon.

```
mega = kilo * 1024;  
celsius = (fahrenheit - 32) * 5 / 9;  
area = 3.1416 * radius * radius;
```

- **Note that the same variable can be used on both sides of an assignment statement.**

```
int item = 5;  
int total = 30;  
total = total + item;
```

- The expression **total + item** evaluates to 35, using the old value of **total**, and this value is assigned to **total**, creating a new value.

Calculations Using C#

- **You can use C# to perform calculations.**
 - Declare whatever variables you need.
 - Create expressions and assign values to your variables.
 - Print out the answer using `Console.WriteLine()`.
- **You can easily perform labeled output, relying on two features of C#:**
 - The operator `+` performs concatenation for **string** data.
 - There is an automatic, implicit conversion available that converts numeric data to string data when required.
 - Hence this code ...

```
int total = 35;  
System.Console.WriteLine("The total is " + total);
```

- ... will produce this output:

```
The total is 35
```

Sample Program

- **This program will convert temperature from Fahrenheit to Celsius.**

– See **ConvertTemp\Step1²**.

```
// ConvertTemp - Step 1
//
// Program converts a hardcoded temperature in
// Fahrenheit to Celsius

int fahr = 86;
int celsius = (fahr - 32) * 5 / 9;
Console.WriteLine("fahrenheit = " + fahr);
Console.WriteLine("celsius = " + celsius);
```

² We will usually omit the chapter number in identifying a sample program when it is in the current chapter. This example program is **Chap02\ConvertTemp\Step1**.

More about Output in C#

- The *Console* class in the *System* namespace supports two simple methods for performing output:

- **WriteLine()** writes out a string followed by a new line.
- **Write()** writes out just the string without the new line.

```
int x = 24;
int y = 5;
int z = x * y;
Console.Write("Product of " + x + " and " + y);
Console.WriteLine(" is " + z);
```

- The output is all on one line:

```
Product of 24 and 5 is 120
```

- A more convenient way to build up an output string is to use *placeholders* {0}, {1}, etc.

- An equivalent way to do the output shown above is:

```
Console.WriteLine("Product of {0} and {1} is {2}",
    x, y, z);
```

- The program **OutputDemo** illustrates the output operations just discussed.
- Later in the course we will see how to control formatting of output, and occasionally in examples we will throw in some simple use of formatting.

Input in C#

- **Our first *ConvertTemp* program is not too useful, because the Fahrenheit temperature is hard-coded.**
 - To convert a different temperature, you would have to edit the source file and recompile.
- **What we really want to do is allow the user of the program to enter a value at runtime for the Fahrenheit temperature.**
- **Although simple console input in C# is fairly easy, we can make it even easier using object-oriented programming.**
 - We can encapsulate or “wrap” the details of input in a class.
 - It will be easy to use the wrapper class.
- **We already have seen an example of an *InputWrapper* class in our *Echo* sample program**

More about Classes

- **Although we will discuss classes in more detail later, there is a little more you need to know now.**
- **A class can be thought of as a template for creating objects.**
 - An **object** is an instance of a **class**.
- **A class specifies data and behavior.**
 - The data is different for each object instance.
- **In C#, you instantiate a class by using the *new* keyword.**

```
InputWrapper iw = new InputWrapper();
```

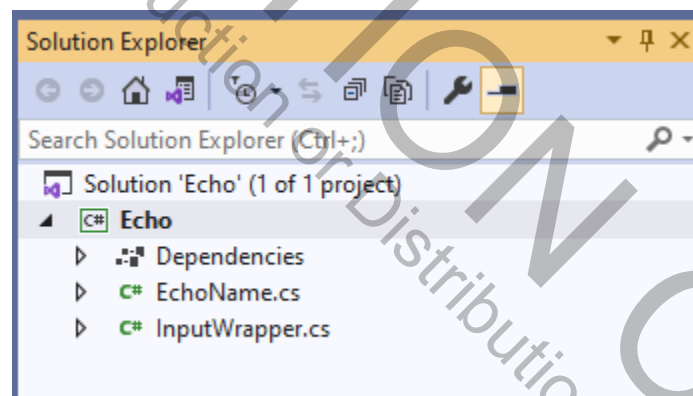
- This code creates the object instance **iw** of the **InputWrapper** class.



InputWrapper Class

- **The *InputWrapper* class “wraps” interactive input for several basic data types.**
 - The supported data types are **int**, **double**, **decimal** and **string**.
 - Methods **getInt()**, **getDouble()**, **getDecimal()** and **getString()** are provided.
 - A prompt string is passed as an input parameter.
 - See files **InputWrapper.cs** in directory **TestInputWrapper**, which implements the class, and **TestInputWrapper.cs**, which tests the class.
- **Although the code is quite short, it is a little complex, involving a number of different methods from different .NET classes in the .NET Class Library.**
- **However, you do not need to be familiar with the implementation of *InputWrapper* in order to use it.**
 - That is the beauty of “encapsulation”—complex functionality can be hidden by an easy-to-use interface.

Multiple Files in Visual Studio

- It is very easy to work with Visual Studio projects that have multiple files.
- As an example, open the Visual Studio *solution* that is specified by the file *Chap02\Echo\Echo.sln*.
- A solution can contain one or more *projects*.
 - In this case there is a single project file **Echo.csproj**.
- You can see all the files in a solution through the *Solution Explorer*.



- When you build the solution via the toolbar button , you will build all the projects in the solution.
 - You can also build just the current project via the toolbar button .

The .NET Class Library³

- **.NET has a very large class library (several thousand classes).**
- **To make all of this functionality more manageable, the classes are partitioned into *namespaces*.**
- **The fundamental namespace is *System*, which contains many useful classes, among them:**
 - **Console** provides access to standard input, output, and error streams for I/O.
 - **Convert** provides conversions among base data types.
 - **Math** provides mathematical constants and functions.

³ Traditionally the term “.NET Framework” was used to refer to a collection of class libraries running on Windows. Then Microsoft introduced a new cross-platform implementation built on .NET Core. To avoid confusion, we will use “.NET Framework” to refer to the classical framework running on Windows. We will use the term “.NET Class Library” to refer to Microsoft’s class libraries running on .NET Core. This topic is discussed in greater detail in Object Innovations course “.NET Frameworks”.

The .NET Class Library (Cont'd)

- **Underneath *System*, there are other namespaces, among them:**
 - **System.Data** contains classes constituting the ADO.NET architecture for accessing databases.
 - **System.Xml** provides standards-based support for processing XML.
 - **System.Drawing** contains classes providing GDI+ graphics functionality.
 - **System.Windows.Forms** provides support for creating applications with rich Windows-based interfaces.
 - **System.Web** provides support for browser/server communication.
 - **System.IO** provides support for reading and writing with streams and files. Both synchronous and asynchronous I/O are supported.
 - **System.Net** provides support for several standard network protocols.

Lab 2

C# Programs for Calculation

In this lab you modify or implement several C# programs to perform calculations. You need to perform input (through a wrapper class), perform a calculation, and output the result. Do as many of these exercises as time permits. If you have extra time, do some of the optional experiments suggested in some of the exercises, or make up some experiments on your own.

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

Suggested time: 30 minutes

Summary

- Every C# application has a class with a method *Main*, which is the entry point into the application.
- The *Console* class includes methods for performing output, such as *WriteLine()*.
- Expressions in C# are formed from literals, variables, and operators.
- With the assignment statement, you can assign a value computed by an expression to a variable.
- Input in C# is a little more complicated than output, but you can use a wrapper class that encapsulates the required C# classes and presents a simple programming interface.
- .NET has a large class library that is partitioned into namespaces.

Lab 2

C# Programs for Calculation

Introduction

In this lab, you modify or implement several C# programs to perform calculations. You need to perform input (through a wrapper class), perform a calculation, and output the result. Do as many of these exercises as time permits. If you have extra time, do some of the optional experiments suggested in some of the exercises, or make up some experiments on your own.

Suggested Time: 30 minutes

Root Directory: OIC\CSharp

Directories:	Labs\Lab2\ConvertTemp	(Exercise 1 work)
	Chap02\ConvertTemp\Step1	(Backup of Exercise 1 starter files)
	Chap02\ConvertTemp\Step2	(Answer to Exercise 1)
	Chap02\TestInputWrapper	(InputWrapper class)
	Labs\Lab2	(Exercise 2 work)
	Chap02\Circle	(Exercise 2 answer)

Exercise 1. Fahrenheit to Celsius Conversion

Examine the code of the starter program. Build and run. Notice that the Fahrenheit temperature to be converted is hard-coded. Modify the program to prompt the user for a Fahrenheit temperature, read in the value entered by the user, and print out the result. Make use of the wrapper class **InputWrapper** that was discussed in this chapter.

The starter program uses **int** as the data type for temperatures. An optional⁴ experiment is to use **double** as the data type. Could you input the Fahrenheit temperature as an **int** and calculate the Celsius temperature as a **double**?

Exercise 2. Calculate the Area of a Circle

1. Use Visual Studio to create a C# Console App project. Click Next.
2. Assign project name **Circle** and for Location navigate to the **Lab2** folder. Click Next.
3. For Framework choose .NET 6.0 and click Create. This will use the new Console template.
4. Copy the file **InputWrapper.cs** from Exercise 1 into the new project.

⁴ In optional exercises we sometimes preview things that are discussed later!

5. Delete the starter WriteLine() of “Hello, World!”.
6. Provide C# code to prompt the user for the radius, read in the value entered by the user, calculate the area of the circle, and print out the result. For pi, use the approximation 3.1416. What is an appropriate data type to use for radius and area? (Remember your high school geometry! $\text{Area} = \text{Pi} * \text{radius squared.}$)
7. As an optional experiment, use the class **Math** (in the namespace **System**) for a more accurate value of pi.

EVALUATION COPY
Unauthorized Reproduction or Distribution Prohibited

EVALUATION COPY
Unauthorized Reproduction Prohibited



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