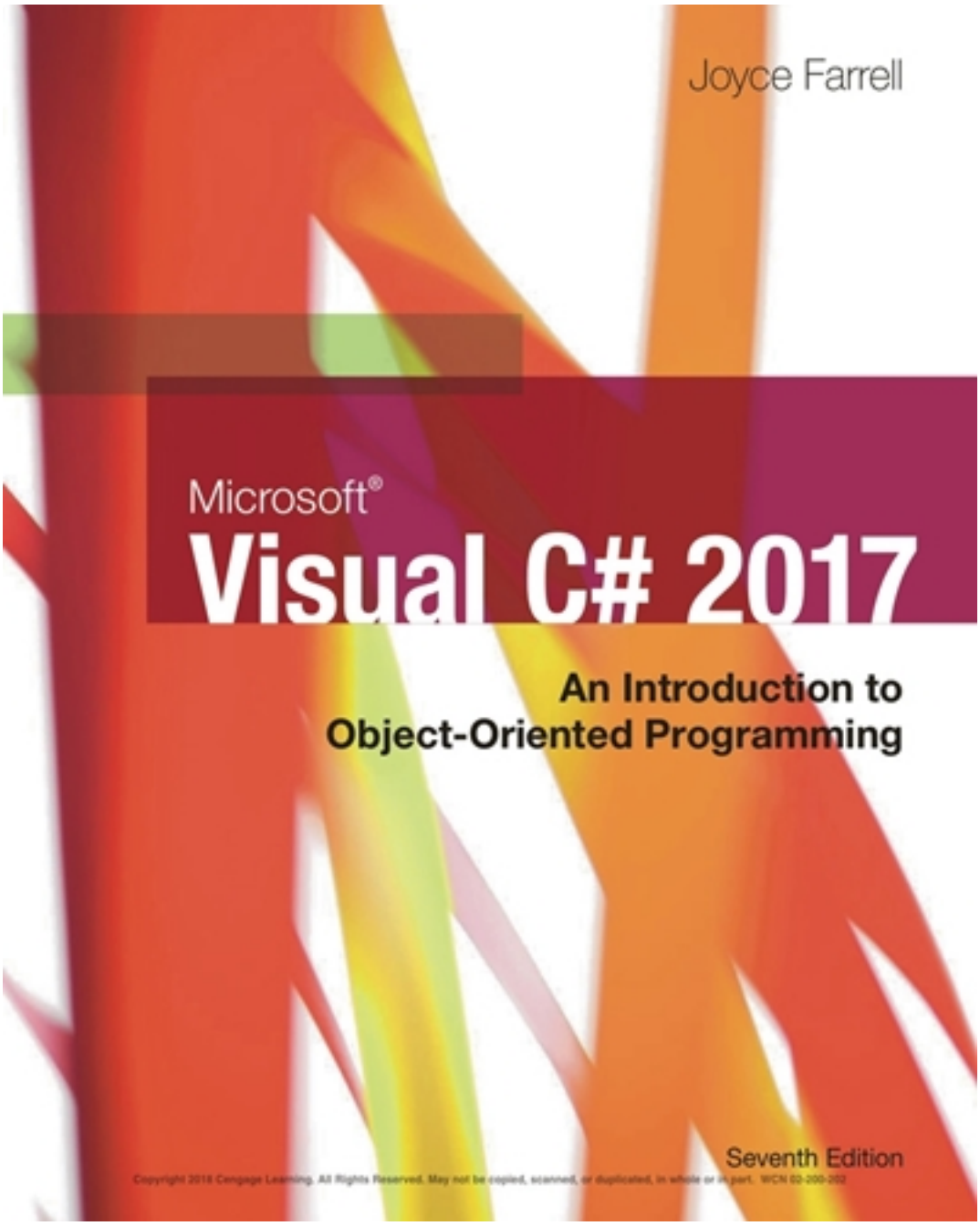
Solutions for Microsoft Visual C# 2017 An Introduction to Object-Oriented Programming 7th Edition by Farrell

CLICK HERE TO ACCESS COMPLETE Solutions



Solutions

Chapter 1

A First Program Using C#

A Guide to this Instructor's Manual:

We have designed this Instructor's Manual to supplement and enhance your teaching experience through classroom activities and a cohesive chapter summary.

This document is organized chronologically, using the same headings that you see in the textbook. Under the headings you will find: lecture notes that summarize the section, Teaching Tips, Class Discussion Topics, and Additional Projects and Resources. Pay special attention to teaching tips and activities geared towards quizzing your students and enhancing their critical thinking skills.

In addition to this Instructor's Manual, our Instructor's Resources also contain PowerPoint Presentations, Test Banks, and other supplements to aid in your teaching experience.

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Overview

Chapter 1 provides an introduction to programming using the C# programming language, as well as some basic concepts related to object-oriented programming. Your students will write a C# program that produces output, and they will learn about reserved words, identifiers, comments, and using namespaces. The final part of the chapter asks students to write a program using both the command prompt and the Visual Studio IDE.

Objectives

- Describe the programming process
- Differentiate between procedural and object-oriented programming
- Describe the features of object-oriented programming languages
- Describe the C# programming language
- Write a C# program that produces output
- Select identifiers to use within your programs
- Improve programs by adding comments and using the `System` namespace
- Compile and execute a C# program using the command prompt and using Visual Studio

Lecture Notes

The Programming Process

1. Define a computer program (also called **software**) as a set of instructions that tell a computer what to do. For emphasis you might state that computers follow instructions exactly as they are provided.
    - Distinguish between **system software** and **application software**.
    - Programmers write the instructions in terms that they can understand using **high-level programming languages** such as C# or Visual Basic. These instructions are converted, or compiled, through a series of steps into **machine language** (a series of 1s and 0s)—the language the computer understands.
2. Explain that high-level programming languages:
    - Use reasonable terms such as “read,” “write,” or “add” instead of the sequence of 1s and 0s used by the computer to perform these tasks.
    - Allow you to assign reasonable names to areas of computer memory where information can be stored (e.g., **variables**).
    - Have their own **syntax**, which are the rules of the language.

Teaching Tip	Explain that the term "syntax" refers to more than just the words that are used. It also refers to punctuation and special characters. The most prominent example in languages like C++, C#, and Java is the semicolon. Compare syntax rules to English grammar rules.

3. Discuss a language's use of **keywords**, or reserved identifiers, and how the language assigns specific meanings to these words.

4. Mention that a compiler translates high-level language statements into machine code. Explain the differences between compilers, interpreters, and assemblers.
   - **Compiler** – a computer program that translates high-level language statements into machine code.
   - Interpreter – a computer program that reads the source code created in a high-level language such as Dartmouth Basic and executes that program line by line.
   - Assembler – a computer program that translates instructions written in an assembler language into machine code.
5. Explain that programming **logic** involves executing the various statements and procedures in the correct order to produce the desired results. Different programmers can write the same program in differing ways and all get the correct result.
6. Define **debugging** as the process of removing all **syntax errors** and logical errors from a program.
7. Explain that in order for a program to perform as intended, it must be correct in both its syntax and its logic.

Teaching Tip	As they practice writing programs, have your students insert some deliberate errors into their code and look at the error messages returned when they try to compile the program. Then when they encounter these errors later on, they will recognize them and more readily know how to correct them.

Procedural and Object-Oriented Programming

1. Explain that in a **procedural program**, programmers create and name computer memory locations that can hold values (**variables**) and write a series of steps or operations to manipulate those values.
2. In a procedural program, the program and its procedures are separate from the stored data on which they operate.

3. Define an **identifier** as a one-word name with no embedded spaces used to reference a variable or a procedure. Explain that programmers in C# use **camel casing** when creating variable names and **Pascal casing** when creating method names.
4. Define procedures, or **methods**, as logical units that group individual operations used in a computer program. Procedures and methods are **called or invoked** by other procedures or methods – ultimately a `Main()` method.

Teaching Tip	Explain that one of the big advantages to using methods is that you can break large problems into smaller, less intimidating pieces. In other words you can code a method to address one part of the application, test it, and when you are satisfied with the results, move on to the next part of the application.

5. Explain that **object-oriented programming (OOP)** is an extension of procedural programming that effectively combines data and the methods that maintain it.
6. Define an object, and discuss **behaviors, states**, and **attributes of an object**. Point out that another term for behaviors is methods, while another term for attributes is properties. These terms may be used interchangeably.
7. Discuss the basic approach to OOP and the types of applications in which it is used.
8. Explain the use of **Computer Simulations**. You may want to site an example of designing a car on a computer instead of building a clay model. Other possibilities include architectural design, CGI in movies, etc.
9. List the benefits of using a **Graphical User Interface (GUI)**. Ask students if they think computers would be popular today if Windows or MAC OS didn't exist and everything had to be done via command line entries.

Teaching Tip	Although procedural and object-oriented programming techniques are somewhat similar, they raise different concerns in the design and development phases that occur before programs are written.

Features of Object-Oriented Programming Languages

1. Define and describe the following features of object-oriented programming:
   - **Classes** – an abstract representation of an object, its attributes (properties), and its behaviors (methods).
   - An **Instance of a class** is called an **Object**.
   - **Encapsulation** –a **black box** technique in which an object's attributes and methods are packaged in a cohesive unit that can be used as an undivided entity.
   - **Interfaces** – a behavior (method) that is available (exposed) to calling modules.

- **Inheritance** – a sub-class that is derived from and used to extend a super (parent) class.
- **Polymorphism** – methods that act appropriately depending on the context.

The C# Programming Language

1. Explain that the **C# programming language** was developed as an object-oriented and component-oriented language. It is part of Microsoft Visual Studio.
2. Mention that C# allows every piece of data to be treated as an object and contains a GUI interface that makes it similar to Visual Basic. C# was modeled after the C++ programming language but eliminates some of the most difficult-to-understand features in C++.
3. Point out that C# is very similar to Java because Java was also based on C++. However, every piece of data in C# is an object, unlike in Java.

Teaching Tip	Read the C# standard at: www.ecma-international.org/publications/standards/Ecma-334.htm.

Writing a C# Program that Produces Output

1. Use Figure 1-2 to describe the main parts of a C# program.
2. Define a string and explain the use of a **literal string.**
3. Explain that arguments represent information that a method needs to perform its task. Point out that arguments to methods always appear within parentheses.
4. Define a **namespace** as a scheme or mechanism that provides a way to group similar classes.
5. Describe the components of any C# method as follows:
   - The **method header**, sometimes called method signature, which includes the method name and information about what will pass into and be returned from a method.
   - The **method body**, which is contained within a pair of curly braces ({}) and includes all the instructions executed by the method.
6. Define **whitespace** as any combination of spaces, tabs, and carriage returns (blank lines) used to organize your program code and make it easier to read.
7. Explain that keywords are predefined and reserved identifiers that have special meaning to the compiler.
8. Explain that classes that contain a Main() method are **application classes** and that applications are executable or **runnable**. Note also that classes that do not contain a

Main() method are **non-application classes**, and are not runnable. Non-application classes provide support for other classes.

Teaching Tips	1. Point out how in Figure 1-2 the lines are indented. This helps visually with grouping lines of code into code blocks. It also makes how code flows in decisions and loops more understandable. 2. Emphasize that when creating code blocks, be careful to match up the braces ({}) correctly. It is far too easy to neglect a brace, add an additional brace, or insert a brace in the wrong location.

Selecting Identifiers

1. Describe the C# requirements to select an identifier, including the following:
   - Identifiers must begin with an underscore, the "at" sign (@), or a letter; letters include foreign-alphabet letters.
   - Identifiers can contain only letters or digits, not special characters such as #, $, or &.
   - Identifiers cannot be a C# reserved keyword.
   - The "at" sign (@) denotes a **verbatim identifier**.
2. Use Table 1-1 to illustrate some of the most common C# keywords.
3. Discuss some conventional, unconventional, valid, and invalid identifiers in C# using Tables 1-2 through 1-4.

Improving Programs by Adding Comments and Using the `System` Namespace

1. Emphasize that adding comments and using the `System` namespace are particularly helpful in large programs.

Teaching Tips	Explain that: • Comments can be used to describe the processing being done within a block of code. This is of particular value where the code contains complex elements. • Comments help with analysis when program changes become necessary. • Comments can be used to track code changes and those who made them.

Adding Program Comments

1. Define **program comments** as nonexecuting statements that you add to document a program. The C# compiler will ignore this line.
2. Explain the reason you might want to **comment out** a statement or statements.

3. Describe the following three types of comments in C#:
   - **Line comments** – can be on their own line or at the end of a line.
   - **Block comments** – used to enclose multi-line comments.
   - **XML documentation format comments** – create an XML document that contains the comments preceded by three forward slashes (///). One potential use of the contents of this XML file is to create help files.
4. Use Figure 1-4 to illustrate how to use comments within a C# program.

Using the `System` Namespace

1. Use Figures 1-5 through 1-7 to show how to eliminate the reference to `System` by using the `System` namespace.
2. Using Figure 1-8, explain the use of the system console.
3. Tell the students that there are many namespaces they could potentially use in their programs.

Quick Quiz 1

1. What do programmers use to translate their high-level language statements into machine code?

   a. hardware
   b. compiler
   c. object
   d. identifier

   Answer: b

2. Which of the following provides the ability to extend a class so as to create a more specific class?

   a. inheritance
   b. identifier
   c. argument
   d. namespace

   Answer: a

3. Programs that attempt to mimic real-world activities so that their processes can be improved are known as which of the following?

   a. objects
   b. methods
   c. computer simulations
   d. procedural programs

   Answer: c

4. Which of the following provides a shortcut way of referring to the classes in a namespace?

   a. using clause
   b. methods
   c. encapsulation
   d. literal string

   Answer: `a`

Compiling and Executing a C# Program

1. Describe the steps that you need to perform to view a program's output, including:

   - Compile your **source code** into **intermediate language (IL)**.
   - The C# **just in time (JIT)** compiler must translate the intermediate code into executable statements.

2. Explain the difference between the **command line** and **command prompt** as compared to an **Integrated Development Environment (IDE)**.

Teaching Tip	Explain to the students that when you compile a C# program, your source code is translated into intermediate language (IL). The JIT compiler converts IL instructions into native code at the last moment and appropriately for each different type of computer on which the code might eventually be executed. In other words, the same set of IL can be JIT compiled and executed on any supported architecture (e.g., Windows, Mac, Unix, or Linux).

Compiling Code from the Command Prompt

1. Use Figure 1-10 to explain how to compile a C# program from the **command prompt** using the `csc` command.

2. Describe some of the most common problems students might encounter when receiving an Operating System Error Message at the command prompt.

   - Misspelling the command csc.
   - Misspelling the filename.
   - Forgetting to include the extension .cs with the filename.
   - Not within the correct subdirectory or folder on your command line.
   - The C# compiler was not installed properly.

3. Describe some of the most common problems students might encounter when receiving a Programming Language Error Message at the command prompt.

   - Syntax Error.
   - Logic Error.

Compiling Code Using the Visual Studio IDE

1. Illustrate the Visual Studio IDE using Figure 1-12.
2. Explain that programming language error messages are handled in the same way in both the text editor and the IDE. Use Figure 1-13 as an example.
3. Point out that if you receive no error messages after compiling the code, the program compiled successfully and you can run the program. Explain to students that they can select Debug from the menu bar, and then select Start Without Debugging.

Noticing the Differences Between the Programs in the Text Editor and the IDE

1. Using Figure 1-14, explain that the Visual Studio IDE contains the following by default:
   - Five `using` statements at the top of the file including the `System` namespace.
   - A namespace declaration and its opening and closing curly braces. The namespace has the same name as the project.
   - A class named `Program`.
   - The words `string[] args` between the parentheses of the `Main()` method header.
     - Explain that these optional terms provide a way to pass command line parameters to the program.

Deciding Which Environment to Use

1. Describe the advantages of writing, compiling, and executing a C# program using the command line.
   - C# programs can be written with your text editor of choice and compiled as long as you have the .NET environment available.
   - This could be an acceptable solution if the application is both relatively small and straightforward and the IDE is not available.
2. Describe the advantages of writing, compiling, and executing a C# program using the Visual Studio IDE.
   - Some of the code you need is already created for you.
   - The code is displayed in color.
   - The IDE includes an auto-complete feature.
   - You can double-click an error message and the cursor will move to the line of code that contains the error.
   - Other debugging tools are available.

Quick Quiz 2

1. In order to view the program output, you must compile the C# program you wrote (called the source code) into which of the following?
   a. integrated code
   b. developer commands
   c. intermediate language (IL)
   d. debugging code

Answer: c

2. What is the name of the C# compiler that translates the intermediate code into executable code?

   Answer: just in time (JIT)
   just in time
   JIT

3. Which data type can hold a series of characters?

   a. `using`
   b. `static`
   c. `main`
   d. `string`

   Answer: d

4. You can use either the command line or which of the following to compile your C# program?

   a. debug
   b. IDE
   c. intermediate language (IL)
   d. system console

   Answer: b

Class Discussion Topics

1. You are writing a new program. What information should be contained in a comment block at the top of that program?

   (Suggestions: Program name, author, date written, purpose of the program, change history, classes used…)

2. Why is C# sometimes called a "semi-compiled" language?

Additional Projects

1. Use the Internet to determine the attributes and methods of a `string` object in C#. Report your findings.
2. Have students use Table 1-1 (the list of C# keywords) and write what each keyword is used for.
3. Write a C# program that prints the phrase "Hello, World!", but with one word per line. Your program output should look as follows:

   Hello,
   World!

Additional Resources

1. Learn Visual C#:
   https://www.microsoft.com/en-us/learning/visual-studio-training.aspx
2. XML documentation format comments:
   https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/xmldoc/xml-documentation-comments
3. C# for Dummies
   http://csharp102.info/default.htm#Beginners

Key Terms

- A computer **program** is a set of instructions that tell a computer what to do.
- **Software** is computer programs.
- **System software** describes the programs that operate the computer.
- **Application software** is the programs that allow users to complete tasks.
- **Hardware** comprises all the physical devices associated with a computer.
- **Machine language** is the most basic circuitry-level language.
- A **high-level programming language** allows you to use a vocabulary of keywords instead of the sequence of on/off switches that perform these tasks.
- **Keywords** are predefined and reserved identifiers that have special meaning to the compiler.
- **Camel casing**, also called **lower camel casing**, is a style of creating identifiers in which the first letter is not capitalized, but each new word is.
- A language's **syntax** is its set of rules.
- A **compiler** is a computer program that translates high-level language statements into machine code.
- A **syntax error** is an error that occurs when a programming language is used incorrectly.
- The **logic** behind any program involves executing the various statements and methods in the correct order to produce the desired results.
- A **bug** is an error in a computer program.
- **Debugging** a program is the process of removing all syntax and logical errors from the program.
- A **procedural program** is created by writing a series of steps or operations to manipulate values.

- **Variables** are named computer memory locations that hold values that might vary during program execution.
- An **identifier** is the name of a program component such as a variable, class, or method.
- **Methods** are compartmentalized, named program units containing instructions that accomplish tasks.
- **Pascal casing**, also called **upper camel casing**, is a style of creating identifiers in which the first letter of all new words in a name, even the first one, is capitalized.
- A program **calls** or **invokes** methods.
- **Object-oriented programming (OOP)** is a programming technique that features objects, classes, encapsulation, interfaces, polymorphism, and inheritance.
- An **object** is a concrete entity that has attributes and behaviors; an object is an instance of a class.
- The **attributes of an object** represent its characteristics.
- The **state of an object** is the collective value of all its attributes at any point in time.
- The **behaviors of an object** are its methods.
- Taking an **object-oriented approach** to a problem means defining the objects needed to accomplish a task and developing classes that describe the objects so each maintains its own data and carries out tasks when another object requests them.
- **Computer simulations** are programs that attempt to mimic real-world activities to foster a better understanding of them.
- **Graphical user interfaces** Graphical User Interfaces, or GUIs (pronounced *gooeys*), are program elements that allow users to interact with a program in a graphical environment.
- A **class** is a category of objects or a type of object.
- An **instance of a class** is an object.
- The **properties** of an object are its values.
- **Encapsulation** is the technique of packaging an object's attributes and methods into a cohesive unit that can be used as an undivided entity.
- A **black box** is a device you use without regard for the internal mechanisms.
- An **interface** is the interaction between a method and an object.
- **Inheritance** is the ability to extend a class so as to create a more specific class that contains all the attributes and methods of a more general class; the extended class usually contains new attributes or methods as well.
- **Polymorphism** is the ability to create methods that act appropriately depending on the context.

- The **C# programming language** was developed as an object-oriented and component-oriented language. It exists as part of Visual Studio, a package used for developing applications for the Windows family of operating systems.
- A **literal string** of characters is a series of characters enclosed in double quotes that is used exactly as entered.
- An **argument** to a method represents information that a method needs to perform its task; it is used between parentheses when you call a method.
- A **namespace** is a construct that acts like a container to provide a way to group similar classes.
- A **method header** includes the method name and information about what will pass into and be returned from a method.
- The **method body** of every method is contained within a pair of curly braces ({}) and includes all the instructions executed by the method.
- **Whitespace** is any combination of spaces, tabs, and carriage returns (blank lines). You use whitespace to organize your program code and make it easier to read.
- The keyword **`static`**, when used in a method header, indicates that a method will be executed through a class and not by an object.
- The keyword **`void`**, when used in a method header, indicates that the method does not return any value.
- **Application classes** contain a Main() method and are executable programs.
- **Runnable** describes files that are executable.
- **Non-application classes** do not contain a Main() method; they provide support for other classes.
- A **verbatim identifier** is an identifier with an @ prefix.
- **Program comments** are nonexecuting statements that you add to document a program.
- To **comment out** a statement is to make a statement nonexecuting.
- **Line comments** start with two forward slashes (//) and continue to the end of the current line. Line comments can appear on a line by themselves or at the end of a line following executable code.
- **Block comments** start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/). Block comments can appear on a line by themselves, on a line before executable code, or after executable code. They also can extend across as many lines as needed.
- **XML-documentation format comments** use a special set of tags within angle brackets to create documentation within a program.
- A **`using` clause** or **`using` directive** declares a namespace.
- **Source code** is the statements you write when you create a program.

- **Intermediate language (IL)** is the language into which source code statements are compiled.
- The C# **just in time (JIT)** compiler translates intermediate code into executable code.
- The **command line** is the line on which you type a command in a system that uses a text interface.
- The **command prompt** is a request for input that appears at the beginning of the command line.
- The **Developer Command Prompt** is a special version of the command prompt for which specific settings have been enabled so that you can easily compile and run C# programs.
- An **Integrated Development Environment (IDE)** is a program development environment that allows you to select options from menus or by clicking buttons. An IDE provides such helpful features as color coding and automatic statement completion.
- The **menu bar** in the IDE lies horizontally across the top of the window, and includes submenus that list additional options.