Contents

The syntax highlighting of the C# code in this book's easy steps is similar to the default colors in the Visual Studio Code Editor:

Default Code Color: Black   Keywords: Blue   Class Objects: Light Blue
Class Methods: Brown   String Literals: Red   Comments: Green

The source code of all examples in this book is available for free download at www.ineasysteps.com/resource-center/downloads

The screenshots in this book illustrate the actual results of executing the listed code steps. If you don't achieve the result illustrated in any example, simply compare your code to that in the original example files you have downloaded to discover where you went wrong.

1 Getting started

Welcome to the exciting world of C# programming. This chapter introduces the Visual Studio Integrated Development Environment and shows you how to create a real Windows application.

Introducing C#

C#

The introduction of the Microsoft .NET framework at the Professional Developers Conference in July 2000 also saw Microsoft introduce a new programming language called C# (pronounced "see-sharp"). The name was inspired by musical notation where a # sharp symbol indicates that a written note should be a semitone higher in pitch. This notion is similar to the naming of the C++ programming language where the ++ symbol indicates that a written value should be incremented by 1.

● C# is designed to be a simple, modern, general-purpose, object-oriented programming language, borrowing key concepts from several other languages – most notably the Java programming language. Consequently, everything in C# is a class "object" with "properties" and "methods" that can be employed by a program.

● C# is an elegant and "type-safe" programming language that enables developers to build a variety of secure and robust applications. You can use C# to create Windows client applications, XML web services, distributed components, client-server applications, database applications, and much, much more.

● C# is specifically designed to utilize the proven functionality built into the .NET framework "class libraries". Windows applications written in C# therefore require the Microsoft .NET framework to be installed on the computer running the application – typically, an integral component of the system.

Microsoft®
.NET
™

The Microsoft .NET Framework

Each version of the Microsoft .NET framework includes a unified set of class libraries and a virtual execution system called the Common Language Runtime (CLR). The CLR allows the C# language and the class libraries to work together seamlessly.

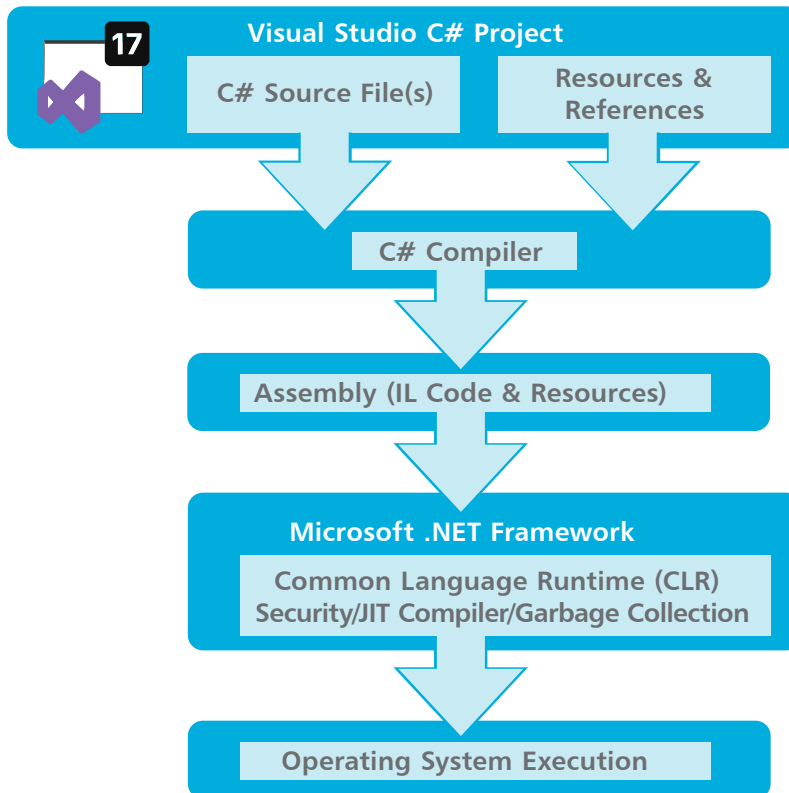To create an executable program, source code written in the C# language is compiled by the C# Compiler into Intermediate Language (IL) code. This is stored on disk, together with other program resources such as images, in an "assembly". Typically, the assembly will have a file extension of **.exe** or **.dll**. Each assembly contains a "manifest" that provides information about that program's security requirements.

When a C# program is executed, the assembly is loaded into the Common Language Runtime (CLR), and the security requirements specified in its assembly manifest are examined. When the security requirements are satisfied, the CLR performs Just-In-Time (JIT) compilation of the IL code into native machine instructions. The CLR then performs "garbage collection", exception handling, and resource management tasks before calling upon the operating system to execute the program:

Just-In-Time compilation is also known as "Dynamic Translation".

```
┌─────────────────────────────────────────────────────┐
│  17          Visual Studio C# Project                 │
│  ┌──────────────────────┐  ┌──────────────────────┐  │
│  │   C# Source File(s)   │  │  Resources &          │  │
│  │                       │  │  References           │  │
│  └──────────────────────┘  └──────────────────────┘  │
└─────────────────────────────────────────────────────┘
                │                    │
                ▼                    ▼
┌─────────────────────────────────────────────────────┐
│                   C# Compiler                         │
└─────────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────────┐
│            Assembly (IL Code & Resources)             │
└─────────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────────┐
│              Microsoft .NET Framework                 │
│         Common Language Runtime (CLR)                 │
│    Security/JIT Compiler/Garbage Collection           │
└─────────────────────────────────────────────────────┘
                        │
                        ▼
┌─────────────────────────────────────────────────────┐
│             Operating System Execution                │
└─────────────────────────────────────────────────────┘
```

Just-In-Time compilation occurs during program execution, rather than prior to its execution.

As language interoperability is a key feature of the Microsoft .NET framework, the IL code generated by the C# Compiler can interact with code generated by the .NET versions of other languages such as Visual Basic and Visual C++. The examples throughout this book demonstrate Visual C# program code.
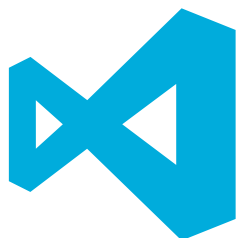
Installing Visual Studio

In order to create Windows applications with the C# programming language, you will first need to install a Visual Studio Integrated Development Environment (IDE).

Microsoft Visual Studio is the professional development tool that provides a fully Integrated Development Environment for Visual Basic, Visual C++, Visual J#, and Visual C#. Within its IDE, code can be written in Visual Basic, C++, J#, or the C# programming language to create Windows applications.

Visual Studio Community edition is a streamlined version of Visual Studio, specially created for those people learning programming. It has a simplified user interface and omits advanced features of the professional edition to avoid confusion. C# code can be written within the **Code Editor** of either version of the Visual Studio IDE to create Windows applications.

Both Visual Studio and Visual Studio Community provide an IDE for C# programming but, unlike the fully-featured Visual Studio product, the Visual Studio Community edition is completely free and can be installed on any system meeting the following minimum requirements:

Component	Requirement
Operating system	Windows 11 Windows 10 (version 1909 or higher) Windows Server 2016 or 2019 *Must be the 64-bit version of any of the above the operating systems.
CPU (processor)	1.8 GHz or faster, 64-bit processor
RAM (memory)	4 GB (16 GB recommended)
HDD (hard drive)	Up to 210 GB available space
Video Card	Minimum resolution of 1366 x 768 Optimum resolution of 1920 x 1080

The Visual Studio Community edition is used throughout this book to demonstrate programming with the C# language, but the examples can also be recreated in Visual Studio. Follow the steps opposite to install the Visual Studio Community edition.

1 Open your web browser and navigate to the Visual Studio download page – at the time of writing, this can be found at **visualstudio.microsoft.com/downloads**
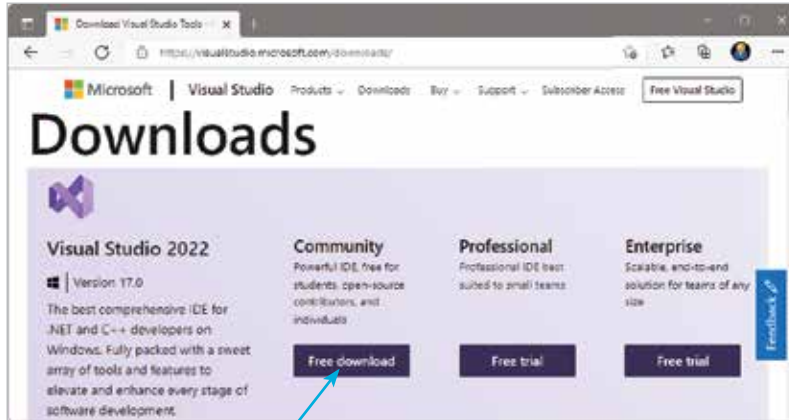
2 Click the button in the Community edition section to download a **VisualStudioSetup.exe** setup file

3 Click on the [icon] setup file icon to begin setup and to run the **Visual Studio Installer**

4 Accept the suggested installation location, then click **Next**

5 Check the two C# **Installer** options shown below

6 Click the **Install** button at the bottom-right of the installer to begin the download and installation process

11

Beware

Choosing a different destination folder may require other paths to be adjusted later – it's simpler to just accept the suggested default.
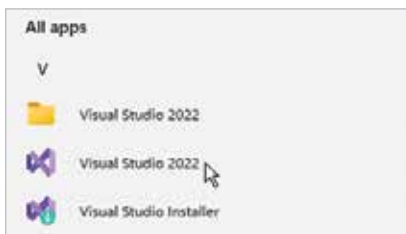
Hot tip

Both **Visual Studio** and **Visual Studio Installer** items get added to your **All Apps** menu. You can re-run the installer at a later date to add or remove features.
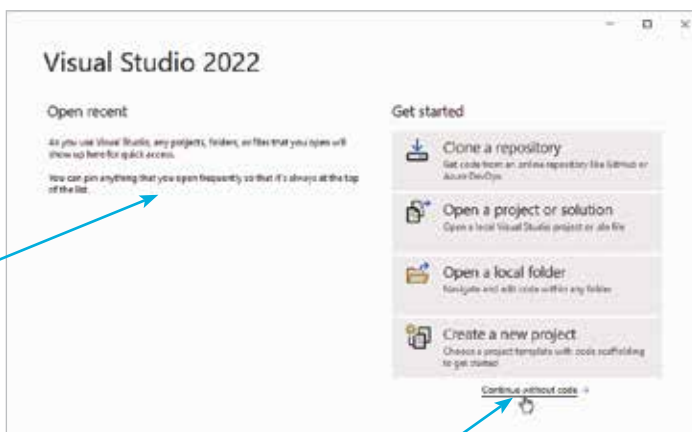
Exploring the IDE

1 Go to your **All apps** menu, then select the Visual Studio 2022 menu item added there by the installer:

2 Sign in with your Microsoft account, or register an account then sign in, to continue

3 See a default **Start Page** appear where recent projects will be listed alongside several "Get started" options

4 For now, just click the **Continue without code** link to launch the Visual Studio application

The Visual Studio Integrated Development Environment (IDE) appears, from which you have instant access to everything needed to produce complete Windows applications – from here, you can create exciting visual interfaces, enter code, compile and execute applications, debug errors, and much more.
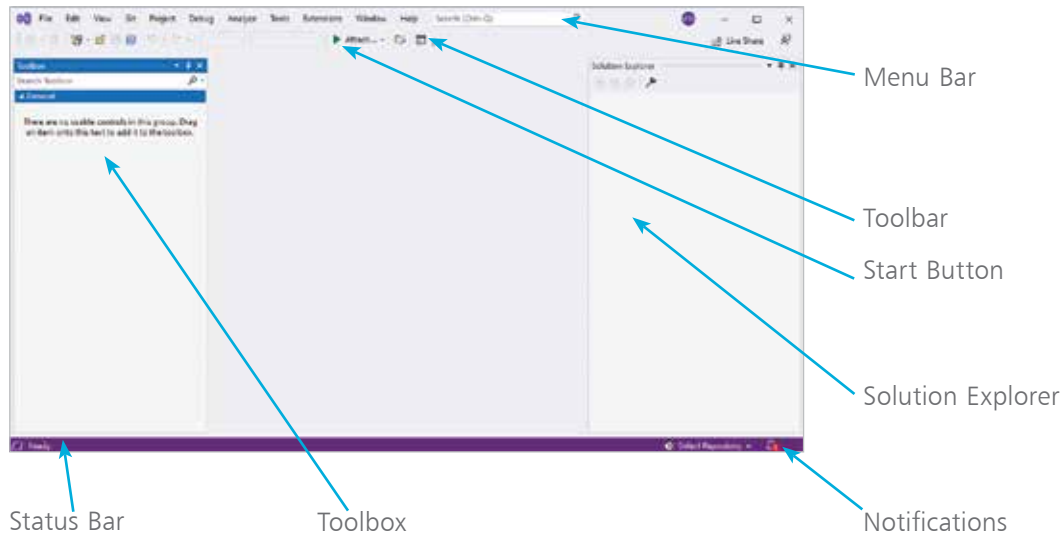
12

Menu Bar

Toolbar

Start Button

Solution Explorer

Status Bar   Toolbox   Notifications

Visual Studio IDE components
The Visual Studio IDE initially provides these standard features:

- **Menu Bar** – Where you can select actions to perform on all your project files and to access Help. When a project is open, extra menus of Project and Build are shown in addition to the default menu selection of File, Edit, View, Git, Project, Debug, Analyze, Tools, Extensions, Window, and Help.

- **Toolbar** – Where you can perform the most popular menu actions with a single click on their associated shortcut icons.

- **Toolbox** – Where you can select visual elements to add to a project. Click View, Toolbox or a side bar button to see its contents. When a project is open, "controls" such as Button, Label, CheckBox, RadioButton, and TextBox are shown here.

- **Solution Explorer** – Where you can see at a glance all the files and resource components contained within an open project.

- **Status Bar** – Where you can read the state of the current activity being undertaken. When building an application, a "Build started" message is displayed here, changing to a "Build succeeded" or "Build failed" message upon completion.
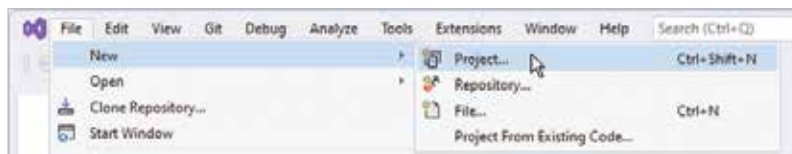

Hot tip

The IDE may have a **Light** color theme by default. To change the color theme, choose the **Tools**, **Options** menu then select **Environment**, **General**, **Color Theme** and select **Blue** or **Dark** theme, or select the **Use system setting** option.
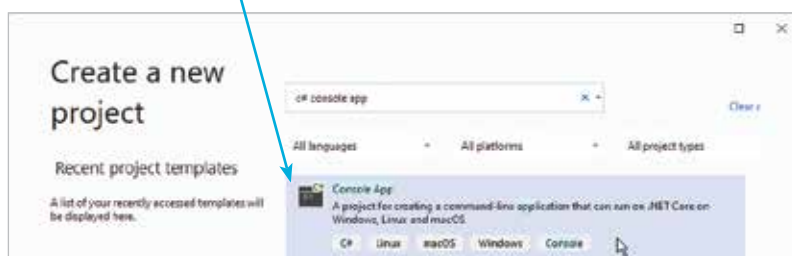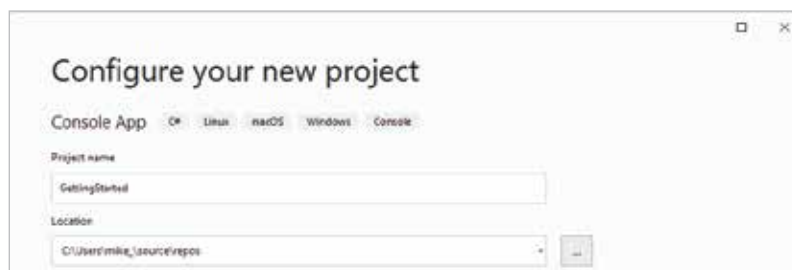
13
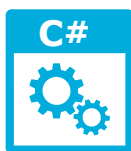
Starting a Console project

1 On the Menu Bar, click **File**, **New**, **Project...** or press **Ctrl** + **Shift** + **N**, to open the "Create a new project" dialog

| File | Edit | View | Git | Debug | Analyze | Tools | Extensions | Window | Help | Search (Ctrl+Q) |

- New ▸ ▯ Project...   Ctrl+Shift+N
- Open ▸ Repository...
- Clone Repository...   File...   Ctrl+N
- Start Window   Project From Existing Code...

2 In the "Create a new project" dialog box, select the **C# Console App** item (for .NET Core on Windows, Linux and macOS), then click **Next**

Create a new project

Recent project templates
A list of your recently accessed templates will be displayed here.

Console App
A project for creating a command-line application that can run on .NET Core on Windows, Linux and macOS
C#   Linux   macOS   Windows   Console

3 In the next dialog, enter a project name plus location and click the **Create** button, then select the .NET 6.0 (Long-term support) framework and click **Create** again

Configure your new project

Console App   C#   Linux   macOS   Windows   Console

Project name
GettingStarted

Location
C:\Users\mike\source\repos

Visual Studio now creates your new project and loads it into the IDE. A **Code Editor** window appears, containing default skeleton project code generated by Visual Studio.

4 Drag the **Code Editor** window tab to undock the **Code Editor** window from the Visual Studio IDE frame

...cont'd

The undocked window title displays the project name, and the tab displays the file name of the code as "Program.cs".

The top-left drop-down box indicates the name of the project to which this file belongs – in this case, it's "GettingStarted".
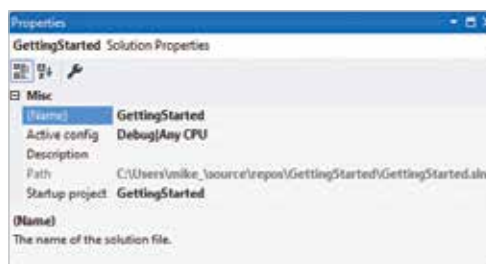
5 Select the **View, Solution Explorer** menu to open a **Solution Explorer** window, to discover all the items in your project – click the arrow buttons to expand or collapse categories
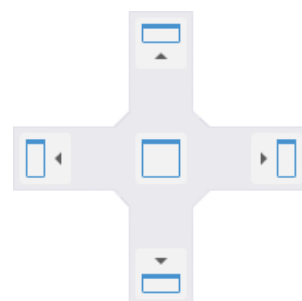

6 Select the **View, Properties Window** menu to open a **Properties** window, then select any item in the **Solution Explorer** window to see its properties then appear in the **Properties** window

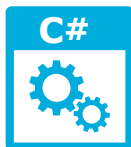The **Code Editor** window is where you write C# code to create an application. The Visual Studio IDE has now gathered all the resources needed to build a default Console application.

Hot tip

You can drag the title bar of any window to undock that window from the Visual Studio IDE frame. When dragging, you can drop a window on the "guide diamond" (shown below) to dock the window in your preferred position.

15

Running a Console project

In order to run a program, Visual Studio will first build the app then execute the app. This can be done in one of two modes:

● **Debug Mode** – The program is compiled with symbolic debugging information included in the program files. This allows Visual Studio's built-in debugger to find bugs, but has optimization of Intermediate Language (IL) code disabled.

● **Release Mode** – The program is compiled without debugging information included in the program files, but has optimization of Intermediate Language (IL) code enabled.

During program development it is generally preferable to run your programs in Debug mode. There is an option to run a program in Debug mode without debugging, but it's seldom desirable as this executes the program without the possibility of stepping through the code to breakpoints.

Hot tip

You will discover how to use breakpoints in Chapter 8 (Solving problems).

1 On the toolbar, set the solution configuration to **Debug**

Hot tip

Alternatively, select **Debug**, **Start Debugging** to build and run a program with debugging enabled.

2 Now, click the green arrow "**Start**" button to build and run the program with debugging enabled
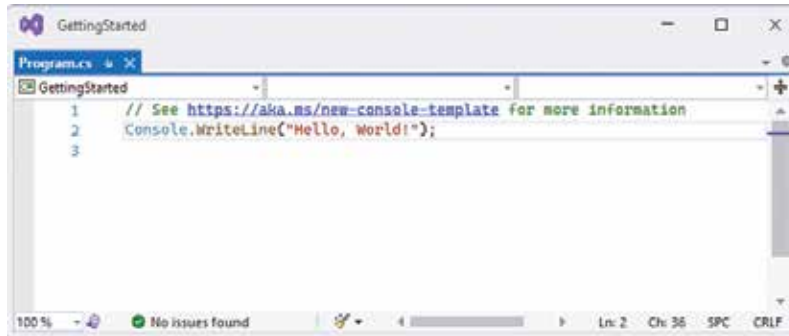
3 See the app display a traditional greeting in the Console

4 Press any keyboard key to close the Console window, and return to the Visual Studio **Code Editor**

Code analysis

Examination of the code helps to understand what is happening:

- The first line is a <u>comment</u>. Anything on a line after **//** is ignored by the compiler. By default, comments are colored green in the Visual Studio **Code Editor**. In this case, the comment includes a hyperlink that will open a page in your web browser. The page explains how the .NET 6 framework uses new templates to simplify the creation of Console apps.

- **Console.WriteLine("Hello, World!") ;** This is a <u>statement</u> that calls upon the **WriteLine( )** method of the **Console** class to output the text string enclosed in quote marks within its parentheses. Notice that the statement is terminated by a **;** semi-colon character. By default, class names are colored light blue, method names are colored brown, strings are colored red, and other code here is colored black.

5. Add another line to the code, as a statement to output a second text string containing your own name such as...
   Console.WriteLine("Good afternoon, Mike!") ;

6. Run the modified code in Debug mode to see the result

17

Following the rules

As with all programming languages, C# has a number of syntax rules that must be precisely followed to ensure the code is correctly formatted for the C# compiler to clearly understand:

- **Case-sensitivity** – C# is a case-sensitive language, which means that uppercase "A" and lowercase "a" are regarded as totally different items.

- **Termination** – All statements in C# language must be terminated by a **;** semicolon character, just as all sentences in English language must be terminated by a . period character – for example: **Console.WriteLine("Hello World!") ;**

- **Single-line comments** – Brief comments on a single line must begin with **//** two forward slash characters – for example: **// Output the traditional greeting.**

- **Block comments** – Extended comments on multiple lines must begin with **/*** forward slash and asterisk characters, and must end with the reverse ***/** asterisk and forward slash – for example:
/*
   C# Programming in easy steps.
   Getting started with the traditional greeting.
*/

- **White space** – Spaces, tabs, newline characters, and comments are ignored by the C# compiler, so can be used extensively to organize code without performance penalty.

- **Escape sequences** – The C# compiler recognizes **\n** as a newline character and **\t** as a tab character, so these can be used to format output –
for example: **Console.WriteLine("Line One \n Line Two") ;**

- **Naming conventions** – A programmer-defined identifier name in C# code may begin with an _ underscore character or a letter in uppercase or lowercase. The name may also contain underscore, letter, and number characters – for example: **class MyNo1_Class**

- **Keywords** – The C# language has a number of keywords (listed opposite) that have special syntactic meaning and may not be used to name programmer-defined items in code.

It is recommended that you comment your code to make it readily understood by others or when revisiting your own code later.

The **WriteLine( )** method automatically adds a newline after its output.

C# Reserved Keywords

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	override	params	private
protected	public	readonly	ref
return	sbyte	sealed	short
sizeof	stackalloc	static	string
struct	switch	this	throw
true	try	typeof	uint
ulong	unchecked	unsafe	ushort
using	virtual	void	volatile
while			

Hot tip

If you absolutely must use a keyword to name a programmer-defined element, it may be prefixed by an @ character to distinguish it from the keyword – but this is best avoided.

19

C# Contextual Keywords

add	and	alias	ascending	async
await	by	descending	dynamic	equals
from	get	global	group	init
into	join	let	managed	nameof
nint	not	notnull	nuint	on
or	orderby	partial	record	remove
select	set	unmanaged	value	var
when	where	with	yield	

Don't forget

Contextual keywords have special significance in certain code – for example, **get** and **set** in method declarations.

Summary

- **C#** is an object-oriented programming language that utilizes the proven functionality of the Microsoft **.NET** class libraries.

- The C# compiler generates **Intermediate Language (IL)** code that is stored on disk alongside resources in an assembly.

- The **Common Language Runtime (CLR)** examines an assembly's security requirements before JIT compilation.

- **Just-In-Time** compilation translates IL code into native machine code for execution by the operating system.

- Microsoft Visual Studio provides a fully **Integrated Development Environment (IDE)** for C# programming.

- A new Visual C# **Console** application generates default skeleton project code in the Visual Studio **Code Editor**.

- The Visual Studio **Solution Explorer** shows all files in a project, and the **Properties** window shows their properties.

- The default C# skeleton code provided in the **Code Editor** can create a C# Console program that outputs a text string.

- The **Console** class has a **WriteLine( )** method that can be used to output a quoted text string specified within its parentheses.

- A C# program can be run in the Visual Studio IDE by selecting the **Debug**, **Start Debugging** menu, or by setting a **Debug** solution configuration and clicking the **Start** button.

- C# is a case-sensitive programming language in which all statements must be terminated by a ; semicolon character.

- Single-line **//** comments and **/* */** block comments can be incorporated to explain C# program code.

- C# has keywords that have special syntactic meaning, so cannot be used to name programmer-defined code items.