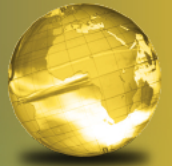


GLOBAL
EDITION



Visual C#

How to Program

SIXTH EDITION

Paul Deitel • Harvey Deitel



DIGITAL RESOURCES FOR STUDENTS

Your new textbook provides 12-month access to digital resources that may include source code, web chapters and more. Refer to the preface in the textbook for a detailed list of resources.

Follow the instructions below to register for the Companion Website for Paul Deitel and Harvey Deitel's ***Visual C# How to Program, Sixth Edition, Global Edition***.

- 1 Go to **www.pearsonglobaleditions.com**
- 2 Enter the title of your textbook or browse by author name.
- 3 Click Companion Website.
- 4 Click Register and follow the on-screen instructions to create a login name and password.

ISSDCH-NEUSS-WAXEN-PISTE-GNASH-WWRSE



Use this student access code to register for the Companion Website.

Use the login name and password you created during registration to start using the digital resources that accompany your textbook.

IMPORTANT

This access code can only be used once. This subscription is valid for 12 months upon activation and is not transferable.

For technical support go to **<https://support.pearson.com/getsupport>**

Visual C#[®]



HOW TO PROGRAM

This page intentionally left blank

Visual C#[®]



HOW TO PROGRAM

SIXTH EDITION
GLOBAL EDITION

Paul Deitel
Harvey Deitel
Deitel & Associates, Inc.



Boston Columbus Hoboken Indianapolis New York San Francisco
Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal
Toronto Delhi Mexico City São Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo

Vice President, Editorial Director: *Marcia Horton*
Acquisitions Editor: *Tracy Johnson*
Editorial Assistant: *Kristy Alaura*
Acquisitions Editor, Global Editions: *Sourabh Maheshwari*
VP of Marketing: *Christy Lesko*
Field Marketing Manager: *Demetrius Hall*
Marketing Assistant: *Jon Bryant*
Director of Product Management: *Erin Gregg*
Team Lead, Program and Project Management: *Scott Disanno*
Program Manager: *Carole Snyder*
Project Manager: *Robert Engelhardt*
Project Editor, Global Editions: *K.K. Neelakantan*
Manufacturing Buyer, Higher Ed | RR Donnelley: *Maura Zaldivar-Garcia*
Senior Manufacturing Controller, Global Editions: *Trudy Kimber*
Media Production Manager, Global Editions: *Vikram Kumar*
Cover Design: *Lumina Datamatics*
R&P Manager: *Ben Ferrini*
Inventory Manager: *Ann Lam*
Cover Photo Credit: © *Pichugin Dmitry/Shutterstock.com*

Credits and acknowledgments borrowed from other sources and reproduced, with permission, in this textbook appear on page 6.

The authors and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The authors and publisher make no warranty of any kind, expressed or implied, with regard to these programs or to the documentation contained in this book. The authors and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsonglobaleditions.com

© Pearson Education Limited 2018

The rights of Paul Deitel and Harvey Deitel to be identified as the authors of this work have been asserted by them in accordance with the Copyright, Designs and Patents Act 1988.

Authorized adaptation from the United States edition, entitled Visual C# How to Program, 6th Edition, ISBN 978-0-13-460154-0 by Paul Deitel and Harvey Deitel published by Pearson Education © 2017.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a license permitting restricted copying in the United Kingdom issued by the Copyright Licensing Agency Ltd, Saffron House, 6-10 Kirby Street, London EC1N 8TS.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

10 9 8 7 6 5 4 3 2 1

ISBN-10: 1-292-15346-6

ISBN-13: 978-1-292-15346-9

Typeset by GEX Publishing Services

Printed and bound in Malaysia

*In memory of William Siebert, Professor Emeritus of
Electrical Engineering and Computer Science at MIT:*

*Your use of visualization techniques in
your Signals and Systems lectures inspired
the way generations of engineers, computer
scientists, educators and authors present
their work.*

Harvey and Paul Deitel

Trademarks

DEITEL and the double-thumbs-up bug are registered trademarks of Deitel and Associates, Inc.

Microsoft® Windows®, and Microsoft Visual C#® are registered trademarks of the Microsoft Corporation in the U.S.A. And other countries. This book is not sponsored or endorsed by or affiliated with the Microsoft Corporation.

UNIX is a registered trademark of The Open Group.

Throughout this book, trademarks are used. Rather than put a trademark symbol in every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner, with no intention of infringement of the trademark.



Contents

Preface	23
Before You Begin	37
I Introduction to Computers, the Internet and Visual C#	41
1.1 Introduction	42
1.2 Computers and the Internet in Industry and Research	42
1.3 Hardware and Software	45
1.3.1 Moore's Law	45
1.3.2 Computer Organization	46
1.4 Data Hierarchy	47
1.5 Machine Languages, Assembly Languages and High-Level Languages	50
1.6 Object Technology	51
1.7 Internet and World Wide Web	53
1.8 C#	55
1.8.1 Object-Oriented Programming	56
1.8.2 Event-Driven Programming	56
1.8.3 Visual Programming	56
1.8.4 Generic and Functional Programming	56
1.8.5 An International Standard	57
1.8.6 C# on Non-Windows Platforms	57
1.8.7 Internet and Web Programming	57
1.8.8 Asynchronous Programming with <code>async</code> and <code>await</code>	57
1.8.9 Other Key Programming Languages	58
1.9 Microsoft's .NET	60
1.9.1 .NET Framework	60
1.9.2 Common Language Runtime	60
1.9.3 Platform Independence	61
1.9.4 Language Interoperability	61
1.10 Microsoft's Windows® Operating System	61
1.11 Visual Studio Integrated Development Environment	63
1.12 Painter Test-Drive in Visual Studio Community	63

2 Introduction to Visual Studio and Visual Programming 73

2.1	Introduction	74
2.2	Overview of the Visual Studio Community 2015 IDE	74
2.2.1	Introduction to Visual Studio Community 2015	74
2.2.2	Visual Studio Themes	75
2.2.3	Links on the Start Page	75
2.2.4	Creating a New Project	76
2.2.5	New Project Dialog and Project Templates	77
2.2.6	Forms and Controls	78
2.3	Menu Bar and Toolbar	79
2.4	Navigating the Visual Studio IDE	82
2.4.1	Solution Explorer	83
2.4.2	Toolbox	84
2.4.3	Properties Window	84
2.5	Help Menu and Context-Sensitive Help	86
2.6	Visual Programming: Creating a Simple App that Displays Text and an Image	87
2.7	Wrap-Up	96
2.8	Web Resources	97

3 Introduction to C# App Programming 105

3.1	Introduction	106
3.2	Simple App: Displaying a Line of Text	106
3.2.1	Comments	107
3.2.2	<code>using</code> Directive	108
3.2.3	Blank Lines and Whitespace	109
3.2.4	Class Declaration	109
3.2.5	Main Method	111
3.2.6	Displaying a Line of Text	111
3.2.7	Matching Left ({) and Right (}) Braces	112
3.3	Creating a Simple App in Visual Studio	112
3.3.1	Creating the Console App	112
3.3.2	Changing the Name of the App File	114
3.3.3	Writing Code and Using <i>IntelliSense</i>	114
3.3.4	Compiling and Running the App	116
3.3.5	Errors, Error Messages and the Error List Window	117
3.4	Modifying Your Simple C# App	117
3.4.1	Displaying a Single Line of Text with Multiple Statements	118
3.4.2	Displaying Multiple Lines of Text with a Single Statement	118
3.5	String Interpolation	120
3.6	Another C# App: Adding Integers	121
3.6.1	Declaring the <code>int</code> Variable <code>number1</code>	122
3.6.2	Declaring Variables <code>number2</code> and <code>sum</code>	123

3.6.3	Prompting the User for Input	123
3.6.4	Reading a Value into Variable <code>number1</code>	123
3.6.5	Prompting the User for Input and Reading a Value into <code>number2</code>	124
3.6.6	Summing <code>number1</code> and <code>number2</code>	124
3.6.7	Displaying the sum with <code>string</code> Interpolation	125
3.6.8	Performing Calculations in Output Statements	125
3.7	Memory Concepts	125
3.8	Arithmetic	126
3.8.1	Arithmetic Expressions in Straight-Line Form	127
3.8.2	Parentheses for Grouping Subexpressions	127
3.8.3	Rules of Operator Precedence	127
3.8.4	Sample Algebraic and C# Expressions	128
3.8.5	Redundant Parentheses	129
3.9	Decision Making: Equality and Relational Operators	129
3.10	Wrap-Up	134

4 Introduction to Classes, Objects, Methods and strings

146

4.1	Introduction	147
4.2	Test-Driving an Account Class	148
4.2.1	Instantiating an Object—Keyword <code>new</code>	148
4.2.2	Calling Class <code>Account</code> 's <code>GetName</code> Method	149
4.2.3	Inputting a Name from the User	149
4.2.4	Calling Class <code>Account</code> 's <code>SetName</code> Method	150
4.3	<code>Account</code> Class with an Instance Variable and <i>Set</i> and <i>Get</i> Methods	150
4.3.1	<code>Account</code> Class Declaration	150
4.3.2	Keyword <code>class</code> and the Class Body	151
4.3.3	Instance Variable name of Type <code>string</code>	151
4.3.4	<code>SetName</code> Method	152
4.3.5	<code>GetName</code> Method	154
4.3.6	Access Modifiers <code>private</code> and <code>public</code>	154
4.3.7	<code>Account</code> UML Class Diagram	155
4.4	Creating, Compiling and Running a Visual C# Project with Two Classes	156
4.5	Software Engineering with <i>Set</i> and <i>Get</i> Methods	157
4.6	<code>Account</code> Class with a Property Rather Than <i>Set</i> and <i>Get</i> Methods	158
4.6.1	Class <code>AccountTest</code> Using <code>Account</code> 's Name Property	158
4.6.2	<code>Account</code> Class with an Instance Variable and a Property	160
4.6.3	<code>Account</code> UML Class Diagram with a Property	162
4.7	Auto-Implemented Properties	162
4.8	<code>Account</code> Class: Initializing Objects with Constructors	163
4.8.1	Declaring an <code>Account</code> Constructor for Custom Object Initialization	163
4.8.2	Class <code>AccountTest</code> : Initializing <code>Account</code> Objects When They're Created	164

4.9	Account Class with a Balance; Processing Monetary Amounts	166
4.9.1	Account Class with a decimal balance Instance Variable	166
4.9.2	AccountTest Class That Uses Account Objects with Balances	169
4.10	Wrap-Up	173

5 Algorithm Development and Control Statements: Part I 182

5.1	Introduction	183
5.2	Algorithms	184
5.3	Pseudocode	184
5.4	Control Structures	185
5.4.1	Sequence Structure	185
5.4.2	Selection Statements	186
5.4.3	Iteration Statements	187
5.4.4	Summary of Control Statements	187
5.5	if Single-Selection Statement	187
5.6	if...else Double-Selection Statement	188
5.6.1	Nested if...else Statements	189
5.6.2	Dangling-else Problem	191
5.6.3	Blocks	191
5.6.4	Conditional Operator (?:)	192
5.7	Student Class: Nested if...else Statements	193
5.8	while Iteration Statement	196
5.9	Formulating Algorithms: Counter-Controlled Iteration	197
5.9.1	Pseudocode Algorithm with Counter-Controlled Iteration	197
5.9.2	Implementing Counter-Controlled Iteration	198
5.9.3	Integer Division and Truncation	200
5.10	Formulating Algorithms: Sentinel-Controlled Iteration	201
5.10.1	Top-Down, Stepwise Refinement: The Top and First Refinement	201
5.10.2	Second Refinement	202
5.10.3	Implementing Sentinel-Controlled Iteration	204
5.10.4	Program Logic for Sentinel-Controlled Iteration	205
5.10.5	Braces in a while Statement	206
5.10.6	Converting Between Simple Types Explicitly and Implicitly	206
5.10.7	Formatting Floating-Point Numbers	207
5.11	Formulating Algorithms: Nested Control Statements	208
5.11.1	Problem Statement	208
5.11.2	Top-Down, Stepwise Refinement: Pseudocode Representation of the Top	209
5.11.3	Top-Down, Stepwise Refinement: First Refinement	209
5.11.4	Top-Down, Stepwise Refinement: Second Refinement	209
5.11.5	Complete Second Refinement of the Pseudocode	210
5.11.6	App That Implements the Pseudocode Algorithm	211
5.12	Compound Assignment Operators	213

5.13	Increment and Decrement Operators	213
5.13.1	Prefix Increment vs. Postfix Increment	214
5.13.2	Simplifying Increment Statements	215
5.13.3	Operator Precedence and Associativity	216
5.14	Simple Types	216
5.15	Wrap-Up	217

6 Control Statements: Part 2 232

6.1	Introduction	233
6.2	Essentials of Counter-Controlled Iteration	233
6.3	for Iteration Statement	235
6.3.1	A Closer Look at the for Statement's Header	236
6.3.2	General Format of a for Statement	236
6.3.3	Scope of a for Statement's Control Variable	236
6.3.4	Expressions in a for Statement's Header Are Optional	237
6.3.5	Placing Arithmetic Expressions in a for Statement's Header	237
6.3.6	Using a for Statement's Control Variable in the Statement's Body	238
6.3.7	UML Activity Diagram for the for Statement	238
6.4	Examples Using the for Statement	238
6.5	App: Summing Even Integers	239
6.6	App: Compound-Interest Calculations	240
6.6.1	Performing the Interest Calculations with Math Method pow	241
6.6.2	Formatting with Field Widths and Alignment	242
6.6.3	Caution: Do Not Use float or double for Monetary Amounts	243
6.7	do...while Iteration Statement	244
6.8	switch Multiple-Selection Statement	245
6.8.1	Using a switch Statement to Count A, B, C, D and F Grades	245
6.8.2	switch Statement UML Activity Diagram	249
6.8.3	Notes on the Expression in Each case of a switch	250
6.9	Class AutoPolicy Case Study: strings in switch Statements	251
6.10	break and continue Statements	253
6.10.1	break Statement	253
6.10.2	continue Statement	254
6.11	Logical Operators	255
6.11.1	Conditional AND (&&) Operator	256
6.11.2	Conditional OR () Operator	256
6.11.3	Short-Circuit Evaluation of Complex Conditions	257
6.11.4	Boolean Logical AND (&) and Boolean Logical OR () Operators	257
6.11.5	Boolean Logical Exclusive OR (^)	258
6.11.6	Logical Negation (!) Operator	258
6.11.7	Logical Operators Example	259
6.12	Structured-Programming Summary	261
6.13	Wrap-Up	266

7	Methods: A Deeper Look	277
7.1	Introduction	278
7.2	Packaging Code in C#	279
7.2.1	Modularizing Programs	279
7.2.2	Calling Methods	280
7.3	static Methods, static Variables and Class Math	280
7.3.1	Math Class Methods	281
7.3.2	Math Class Constants PI and E	282
7.3.3	Why Is Main Declared static?	282
7.3.4	Additional Comments About Main	283
7.4	Methods with Multiple Parameters	283
7.4.1	Keyword static	285
7.4.2	Method Maximum	285
7.4.3	Assembling strings with Concatenation	285
7.4.4	Breaking Apart Large string Literals	286
7.4.5	When to Declare Variables as Fields	287
7.4.6	Implementing Method Maximum by Reusing Method Math.Max	287
7.5	Notes on Using Methods	287
7.6	Argument Promotion and Casting	288
7.6.1	Promotion Rules	289
7.6.2	Sometimes Explicit Casts Are Required	289
7.7	The .NET Framework Class Library	290
7.8	Case Study: Random-Number Generation	292
7.8.1	Creating an Object of Type Random	292
7.8.2	Generating a Random Integer	292
7.8.3	Scaling the Random-Number Range	293
7.8.4	Shifting Random-Number Range	293
7.8.5	Combining Shifting and Scaling	293
7.8.6	Rolling a Six-Sided Die	293
7.8.7	Scaling and Shifting Random Numbers	296
7.8.8	Repeatability for Testing and Debugging	297
7.9	Case Study: A Game of Chance; Introducing Enumerations	297
7.9.1	Method RollDice	300
7.9.2	Method Main's Local Variables	300
7.9.3	enum Type Status	301
7.9.4	The First Roll	301
7.9.5	enum Type DiceNames	301
7.9.6	Underlying Type of an enum	301
7.9.7	Comparing Integers and enum Constants	302
7.10	Scope of Declarations	302
7.11	Method-Call Stack and Activation Records	305
7.11.1	Method-Call Stack	305
7.11.2	Stack Frames	305
7.11.3	Local Variables and Stack Frames	306
7.11.4	Stack Overflow	306
7.11.5	Method-Call Stack in Action	306

7.12	Method Overloading	309
7.12.1	Declaring Overloaded Methods	309
7.12.2	Distinguishing Between Overloaded Methods	310
7.12.3	Return Types of Overloaded Methods	311
7.13	Optional Parameters	311
7.14	Named Parameters	312
7.15	C# 6 Expression-Bodied Methods and Properties	313
7.16	Recursion	314
7.16.1	Base Cases and Recursive Calls	314
7.16.2	Recursive Factorial Calculations	314
7.16.3	Implementing Factorial Recursively	315
7.17	Value Types vs. Reference Types	317
7.18	Passing Arguments By Value and By Reference	318
7.18.1	ref and out Parameters	319
7.18.2	Demonstrating ref, out and Value Parameters	320
7.19	Wrap-Up	322

8 Arrays; Introduction to Exception Handling 339

8.1	Introduction	340
8.2	Arrays	341
8.3	Declaring and Creating Arrays	342
8.4	Examples Using Arrays	343
8.4.1	Creating and Initializing an Array	343
8.4.2	Using an Array Initializer	344
8.4.3	Calculating a Value to Store in Each Array Element	345
8.4.4	Summing the Elements of an Array	347
8.4.5	Iterating Through Arrays with foreach	347
8.4.6	Using Bar Charts to Display Array Data Graphically; Introducing Type Inference with var	349
8.4.7	Using the Elements of an Array as Counters	352
8.5	Using Arrays to Analyze Survey Results; Intro to Exception Handling	353
8.5.1	Summarizing the Results	354
8.5.2	Exception Handling: Processing the Incorrect Response	355
8.5.3	The try Statement	355
8.5.4	Executing the catch Block	355
8.5.5	Message Property of the Exception Parameter	356
8.6	Case Study: Card Shuffling and Dealing Simulation	356
8.6.1	Class Card and Getter-Only Auto-Implemented Properties	356
8.6.2	Class DeckOfCards	357
8.6.3	Shuffling and Dealing Cards	360
8.7	Passing Arrays and Array Elements to Methods	361
8.8	Case Study: GradeBook Using an Array to Store Grades	363
8.9	Multidimensional Arrays	369
8.9.1	Rectangular Arrays	369

8.9.2	Jagged Arrays	370
8.9.3	Two-Dimensional Array Example: Displaying Element Values	371
8.10	Case Study: GradeBook Using a Rectangular Array	374
8.11	Variable-Length Argument Lists	380
8.12	Using Command-Line Arguments	382
8.13	(Optional) Passing Arrays by Value and by Reference	384
8.14	Wrap-Up	388

9 Introduction to LINQ and the List Collection 410

9.1	Introduction	411
9.2	Querying an Array of int Values Using LINQ	412
9.2.1	The from Clause	414
9.2.2	The where Clause	415
9.2.3	The select Clause	415
9.2.4	Iterating Through the Results of the LINQ Query	415
9.2.5	The orderby Clause	415
9.2.6	Interface IEnumerable<T>	416
9.3	Querying an Array of Employee Objects Using LINQ	416
9.3.1	Accessing the Properties of a LINQ Query's Range Variable	420
9.3.2	Sorting a LINQ Query's Results by Multiple Properties	420
9.3.3	Any, First and Count Extension Methods	420
9.3.4	Selecting a Property of an Object	420
9.3.5	Creating New Types in the select Clause of a LINQ Query	420
9.4	Introduction to Collections	421
9.4.1	List<T> Collection	421
9.4.2	Dynamically Resizing a List<T> Collection	422
9.5	Querying the Generic List Collection Using LINQ	426
9.5.1	The let Clause	428
9.5.2	Deferred Execution	428
9.5.3	Extension Methods ToArray and ToList	428
9.5.4	Collection Initializers	428
9.6	Wrap-Up	429
9.7	Deitel LINQ Resource Center	429

10 Classes and Objects: A Deeper Look 434

10.1	Introduction	435
10.2	Time Class Case Study; Throwing Exceptions	435
10.2.1	Time1 Class Declaration	436
10.2.2	Using Class Time1	437
10.3	Controlling Access to Members	439
10.4	Referring to the Current Object's Members with the this Reference	440
10.5	Time Class Case Study: Overloaded Constructors	442
10.5.1	Class Time2 with Overloaded Constructors	442
10.5.2	Using Class Time2's Overloaded Constructors	446

10.6	Default and Parameterless Constructors	448
10.7	Composition	449
10.7.1	Class Date	449
10.7.2	Class Employee	451
10.7.3	Class EmployeeTest	452
10.8	Garbage Collection and Destructors	453
10.9	static Class Members	453
10.10	readonly Instance Variables	457
10.11	Class View and Object Browser	458
10.11.1	Using the Class View Window	458
10.11.2	Using the Object Browser	459
10.12	Object Initializers	460
10.13	Operator Overloading; Introducing struct	460
10.13.1	Creating Value Types with struct	461
10.13.2	Value Type ComplexNumber	461
10.13.3	Class ComplexTest	463
10.14	Time Class Case Study: Extension Methods	464
10.15	Wrap-Up	467

I I Object-Oriented Programming: Inheritance 475

11.1	Introduction	476
11.2	Base Classes and Derived Classes	477
11.3	protected Members	479
11.4	Relationship between Base Classes and Derived Classes	480
11.4.1	Creating and Using a CommissionEmployee Class	481
11.4.2	Creating a BasePlusCommissionEmployee Class without Using Inheritance	485
11.4.3	Creating a CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy	490
11.4.4	CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using protected Instance Variables	493
11.4.5	CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using private Instance Variables	496
11.5	Constructors in Derived Classes	500
11.6	Software Engineering with Inheritance	500
11.7	Class object	501
11.8	Wrap-Up	502

12 OOP: Polymorphism and Interfaces 508

12.1	Introduction	509
12.2	Polymorphism Examples	511
12.3	Demonstrating Polymorphic Behavior	512
12.4	Abstract Classes and Methods	515
12.5	Case Study: Payroll System Using Polymorphism	517
12.5.1	Creating Abstract Base Class Employee	518

12.5.2	Creating Concrete Derived Class <code>SalariedEmployee</code>	520
12.5.3	Creating Concrete Derived Class <code>HourlyEmployee</code>	522
12.5.4	Creating Concrete Derived Class <code>CommissionEmployee</code>	523
12.5.5	Creating Indirect Concrete Derived Class <code>BasePlusCommissionEmployee</code>	525
12.5.6	Polymorphic Processing, Operator <code>is</code> and Downcasting	526
12.5.7	Summary of the Allowed Assignments Between Base-Class and Derived-Class Variables	531
12.6	<code>sealed</code> Methods and Classes	532
12.7	Case Study: Creating and Using Interfaces	533
12.7.1	Developing an <code>IPayable</code> Hierarchy	534
12.7.2	Declaring Interface <code>IPayable</code>	536
12.7.3	Creating Class <code>Invoice</code>	536
12.7.4	Modifying Class <code>Employee</code> to Implement Interface <code>IPayable</code>	538
12.7.5	Using Interface <code>IPayable</code> to Process Invoices and Employees Polymorphically	539
12.7.6	Common Interfaces of the .NET Framework Class Library	541
12.8	Wrap-Up	542

13 Exception Handling: A Deeper Look 547

13.1	Introduction	548
13.2	Example: Divide by Zero without Exception Handling	549
13.2.1	Dividing By Zero	550
13.2.2	Enter a Non-Numeric Denominator	551
13.2.3	Unhandled Exceptions Terminate the App	552
13.3	Example: Handling <code>DivideByZeroExceptions</code> and <code>FormatExceptions</code>	552
13.3.1	Enclosing Code in a <code>try</code> Block	554
13.3.2	Catching Exceptions	554
13.3.3	Uncaught Exceptions	555
13.3.4	Termination Model of Exception Handling	556
13.3.5	Flow of Control When Exceptions Occur	556
13.4	.NET Exception Hierarchy	557
13.4.1	Class <code>SystemException</code>	557
13.4.2	Which Exceptions Might a Method Throw?	558
13.5	<code>finally</code> Block	559
13.5.1	Moving Resource-Release Code to a <code>finally</code> Block	559
13.5.2	Demonstrating the <code>finally</code> Block	560
13.5.3	Throwing Exceptions Using the <code>throw</code> Statement	564
13.5.4	Rethrowing Exceptions	564
13.5.5	Returning After a <code>finally</code> Block	565
13.6	The <code>using</code> Statement	566
13.7	Exception Properties	567
13.7.1	Property <code>InnerException</code>	567
13.7.2	Other Exception Properties	568
13.7.3	Demonstrating Exception Properties and Stack Unwinding	568

13.7.4	Throwing an Exception with an InnerException	570
13.7.5	Displaying Information About the Exception	571
13.8	User-Defined Exception Classes	571
13.9	Checking for null References; Introducing C# 6's ?. Operator	575
13.9.1	Null-Conditional Operator (?.)	575
13.9.2	Revisiting Operators is and as	576
13.9.3	Nullable Types	576
13.9.4	Null Coalescing Operator (??)	577
13.10	Exception Filters and the C# 6 when Clause	577
13.11	Wrap-Up	578

14 Graphical User Interfaces with Windows Forms: Part 1 **584**

14.1	Introduction	585
14.2	Windows Forms	586
14.3	Event Handling	588
14.3.1	A Simple Event-Driven GUI	589
14.3.2	Auto-Generated GUI Code	591
14.3.3	Delegates and the Event-Handling Mechanism	593
14.3.4	Another Way to Create Event Handlers	594
14.3.5	Locating Event Information	595
14.4	Control Properties and Layout	596
14.4.1	Anchoring and Docking	597
14.4.2	Using Visual Studio To Edit a GUI's Layout	599
14.5	Labels, TextBoxes and Buttons	600
14.6	GroupBoxes and Panels	603
14.7	CheckBoxes and RadioButtons	606
14.7.1	CheckBoxes	606
14.7.2	Combining Font Styles with Bitwise Operators	608
14.7.3	RadioButtons	609
14.8	PictureBoxes	614
14.9	ToolTips	617
14.10	NumericUpDown Control	618
14.11	Mouse-Event Handling	621
14.12	Keyboard-Event Handling	623
14.13	Wrap-Up	627

15 Graphical User Interfaces with Windows Forms: Part 2 **637**

15.1	Introduction	638
15.2	Menus	638
15.3	MonthCalendar Control	648
15.4	DateTimePicker Control	649

15.5	LinkLabel Control	652
15.6	ListBox Control	655
15.7	CheckedListBox Control	660
15.8	ComboBox Control	663
15.9	TreeView Control	667
15.10	ListView Control	673
15.11	TabControl Control	679
15.12	Multiple Document Interface (MDI) Windows	683
15.13	Visual Inheritance	691
15.14	User-Defined Controls	696
15.15	Wrap-Up	699

16 Strings and Characters: A Deeper Look 707

16.1	Introduction	708
16.2	Fundamentals of Characters and Strings	709
16.3	string Constructors	710
16.4	string Indexer, Length Property and CopyTo Method	711
16.5	Comparing strings	712
16.6	Locating Characters and Substrings in strings	716
16.7	Extracting Substrings from strings	719
16.8	Concatenating strings	720
16.9	Miscellaneous string Methods	720
16.10	Class StringBuilder	722
16.11	Length and Capacity Properties, EnsureCapacity Method and Indexer of Class StringBuilder	723
16.12	Append and AppendFormat Methods of Class StringBuilder	725
16.13	Insert, Remove and Replace Methods of Class StringBuilder	727
16.14	Char Methods	730
16.15	Introduction to Regular Expressions (Online)	732
16.16	Wrap-Up	732

17 Files and Streams 739

17.1	Introduction	740
17.2	Files and Streams	740
17.3	Creating a Sequential-Access Text File	741
17.4	Reading Data from a Sequential-Access Text File	750
17.5	Case Study: Credit-Inquiry Program	754
17.6	Serialization	759
17.7	Creating a Sequential-Access File Using Object Serialization	760
17.8	Reading and Deserializing Data from a Binary File	764
17.9	Classes File and Directory	767
	17.9.1 Demonstrating Classes File and Directory	768
	17.9.2 Searching Directories with LINQ	771
17.10	Wrap-Up	775

18 Searching and Sorting 782

18.1	Introduction	783
18.2	Searching Algorithms	784
18.2.1	Linear Search	784
18.2.2	Binary Search	788
18.3	Sorting Algorithms	792
18.3.1	Selection Sort	793
18.3.2	Insertion Sort	796
18.3.3	Merge Sort	800
18.4	Summary of the Efficiency of Searching and Sorting Algorithms	806
18.5	Wrap-Up	807

19 Custom Linked Data Structures 812

19.1	Introduction	813
19.2	Simple-Type structs, Boxing and Unboxing	813
19.3	Self-Referential Classes	814
19.4	Linked Lists	815
19.5	Stacks	828
19.6	Queues	832
19.7	Trees	835
19.7.1	Binary Search Tree of Integer Values	836
19.7.2	Binary Search Tree of IComparable Objects	843
19.8	Wrap-Up	849

20 Generics 855

20.1	Introduction	856
20.2	Motivation for Generic Methods	857
20.3	Generic-Method Implementation	859
20.4	Type Constraints	862
20.4.1	IComparable<T> Interface	862
20.4.2	Specifying Type Constraints	862
20.5	Overloading Generic Methods	865
20.6	Generic Classes	865
20.7	Wrap-Up	875

21 Generic Collections; Functional Programming with LINQ/PLINQ 881

21.1	Introduction	882
21.2	Collections Overview	884
21.3	Class Array and Enumerators	886
21.3.1	C# 6 using static Directive	888
21.3.2	Class UsingArray's static Fields	889
21.3.3	Array Method Sort	889

21.3.4	Array Method Copy	889
21.3.5	Array Method BinarySearch	889
21.3.6	Array Method GetEnumerator and Interface IEnumerator	889
21.3.7	Iterating Over a Collection with foreach	890
21.3.8	Array Methods Clear, IndexOf, LastIndexOf and Reverse	890
21.4	Dictionary Collections	890
21.4.1	Dictionary Fundamentals	891
21.4.2	Using the SortedDictionary Collection	892
21.5	Generic LinkedList Collection	896
21.6	C# 6 Null Conditional Operator ?[]	900
21.7	C# 6 Dictionary Initializers and Collection Initializers	901
21.8	Delegates	901
21.8.1	Declaring a Delegate Type	903
21.8.2	Declaring a Delegate Variable	903
21.8.3	Delegate Parameters	904
21.8.4	Passing a Method Name Directly to a Delegate Parameter	904
21.9	Lambda Expressions	904
21.9.1	Expression Lambdas	906
21.9.2	Assigning Lambdas to Delegate Variables	907
21.9.3	Explicitly Typed Lambda Parameters	907
21.9.4	Statement Lambdas	907
21.10	Introduction to Functional Programming	907
21.11	Functional Programming with LINQ Method-Call Syntax and Lambdas	909
21.11.1	LINQ Extension Methods Min, Max, Sum and Average	912
21.11.2	Aggregate Extension Method for Reduction Operations	912
21.11.3	The Where Extension Method for Filtering Operations	914
21.11.4	Select Extension Method for Mapping Operations	915
21.12	PLINQ: Improving LINQ to Objects Performance with Multicore	915
21.13	(Optional) Covariance and Contravariance for Generic Types	919
21.14	Wrap-Up	921

22 Databases and LINQ

933

22.1	Introduction	934
22.2	Relational Databases	935
22.3	A Books Database	936
22.4	LINQ to Entities and the ADO.NET Entity Framework	940
22.5	Querying a Database with LINQ	941
22.5.1	Creating the ADO.NET Entity Data Model Class Library	943
22.5.2	Creating a Windows Forms Project and Configuring It to Use the Entity Data Model	947
22.5.3	Data Bindings Between Controls and the Entity Data Model	949
22.6	Dynamically Binding Query Results	955
22.6.1	Creating the Display Query Results GUI	956
22.6.2	Coding the Display Query Results App	957
22.7	Retrieving Data from Multiple Tables with LINQ	959

22.8	Creating a Master/Detail View App	965
22.8.1	Creating the Master/Detail GUI	965
22.8.2	Coding the Master/Detail App	967
22.9	Address Book Case Study	968
22.9.1	Creating the Address Book App's GUI	970
22.9.2	Coding the Address Book App	971
22.10	Tools and Web Resources	975
22.11	Wrap-Up	975

23 Asynchronous Programming with async and await **982**

23.1	Introduction	983
23.2	Basics of async and await	985
23.2.1	async Modifier	985
23.2.2	await Expression	985
23.2.3	async, await and Threads	985
23.3	Executing an Asynchronous Task from a GUI App	986
23.3.1	Performing a Task Asynchronously	986
23.3.2	Method calculateButton_Click	988
23.3.3	Task Method Run: Executing Asynchronously in a Separate Thread	989
23.3.4	awaiting the Result	989
23.3.5	Calculating the Next Fibonacci Value Synchronously	989
23.4	Sequential Execution of Two Compute-Intensive Tasks	990
23.5	Asynchronous Execution of Two Compute-Intensive Tasks	992
23.5.1	awaiting Multiple Tasks with Task Method WhenAll	995
23.5.2	Method StartFibonacci	996
23.5.3	Modifying a GUI from a Separate Thread	996
23.5.4	awaiting One of Several Tasks with Task Method WhenAny	996
23.6	Invoking a Flickr Web Service Asynchronously with HttpClient	997
23.6.1	Using Class HttpClient to Invoke a Web Service	1001
23.6.2	Invoking the Flickr Web Service's flickr.photos.search Method	1001
23.6.3	Processing the XML Response	1002
23.6.4	Binding the Photo Titles to the ListBox	1003
23.6.5	Asynchronously Downloading an Image's Bytes	1004
23.7	Displaying an Asynchronous Task's Progress	1004
23.8	Wrap-Up	1008

Chapters on the Web **1015**

A Operator Precedence Chart **1016**

B Simple Types **1018**

C	ASCII Character Set	1020
	Appendices on the Web	1021
	Index	1023

Online Topics

PDFs presenting additional topics for advanced college courses and professionals are available through the book's Companion Website:

<http://www.pearsonglobaleditions.com/deitel>

New copies of this book come with a Companion Website access code that's located on the book's inside front cover. If the access code is already visible or there is no card, you purchased a used book or an edition that does not come with an access code.

Web App Development with ASP.NET

XML and LINQ to XML

Universal Windows Platform (UWP) GUI, Graphics, Multimedia and XAML

REST Web Services

Cloud Computing with Microsoft Azure™

Windows Presentation Foundation (WPF) GUI, Graphics, Multimedia and XAML

ATM Case Study, Part 1: Object-Oriented Design with the UML

ATM Case Study, Part 2: Implementing an Object-Oriented Design in C#

Using the Visual Studio Debugger



Preface

6

Welcome to the world of desktop, mobile and web app development with Microsoft's® Visual C#® programming language. *Visual C# How to Program, 6/e* is based on C# 6¹ and related Microsoft software technologies. You'll be using the .NET platform and the Visual Studio® Integrated Development Environment on which you'll conveniently write, test and debug your applications and run them on Windows® devices. The Windows operating system runs on desktop and notebook computers, mobile phones and tablets, game systems and a great variety of devices associated with the emerging "Internet of Things."

We believe that this book and its supplements for students and instructors will give you an informative, engaging, challenging and entertaining introduction to Visual C#. The book presents leading-edge computing technologies in a friendly manner appropriate for introductory college course sequences, based on the curriculum recommendations of two key professional organizations—the ACM and the IEEE.²

You'll study four of today's most popular programming paradigms:

- object-oriented programming,
- structured programming,
- generic programming and
- functional programming (new in this edition).

At the heart of the book is the Deitel signature *live-code approach*—rather than using code snippets, we generally present concepts in the context of complete working programs followed by sample executions. We include a broad range of example programs and exercises selected from computer science, business, education, social issues, personal utilities, sports, mathematics, puzzles, simulation, game playing, graphics, multimedia and many other areas. We also provide abundant tables, line drawings and UML diagrams for a more visual learning experience.

1. At the time of this writing, Microsoft has not yet released the official C# 6 Specification. To view an unofficial copy, visit <https://github.com/1jw1004/csharpspec/blob/gh-pages/README.md>

2. These recommendations include *Computer Science Curricula 2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, December 20, 2013, The Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM), IEEE Computer Society.

Read the Before You Begin section after this Preface for instructions on setting up your computer to run the hundreds of code examples and to enable you to develop your own C# apps. The source code for all of the book's examples is available at

<http://www.pearsonglobal editions.com/deitel>

Use the source code we provide to compile and run each program as you study it—this will help you master Visual C# and related Microsoft technologies faster and at a deeper level. Most of the book's examples work in Visual Studio on Windows 7, 8 or 10 (there is no 9). The code examples for the online presentation of the Universal Windows Platform (UWP) and XAML specifically require Windows 10.

Contacting the Authors

As you read the book, if you have a question, we're easy to reach at

deitel@deitel.com

We'll respond promptly.

Join the Deitel & Associates, Inc. Social Media Communities

Subscribe to the *Deitel*[®] *Buzz Online* newsletter

<http://www.deitel.com/newsletter/subscribe.html>

and join the Deitel social media communities on

- Facebook[®]—<http://facebook.com/DeitelFan>
- LinkedIn[®]—<http://linkedin.com/company/deitel-&-associates>
- YouTube[®]—<http://youtube.com/DeitelTV>
- Twitter[®]—<http://twitter.com/Deitel>
- Instagram[®]—<http://instagram.com/DeitelFan>
- Google+[™]—<http://google.com/+DeitelFan>

Object-Oriented Programming with an Early Objects Approach

The book introduces the basic concepts and terminology of object-oriented programming in Chapter 1. In Chapter 2, you'll *visually* manipulate objects, such as labels and images. In Chapter 3, Introduction to C# App Programming, you'll write Visual C# program code that manipulates preexisting objects. You'll develop your first customized classes and objects in Chapter 4. Presenting objects and classes early gets you “thinking about objects” immediately and mastering these concepts more thoroughly.

Our early objects presentation continues in Chapters 5–9 with a variety of straightforward case studies. In Chapters 10–12, we take a deeper look at classes and objects,

present inheritance, interfaces and polymorphism, then use those concepts throughout the remainder of the book.

New C# 6 Features

6 We introduce key new C# 6 language features throughout the book (Fig. 1)—each defining occurrence is marked with a “6” margin icon as shown next to this paragraph.

C# 6 new language feature	First introduced in
string interpolation	Section 3.5
expression bodied methods and <code>get</code> accessors	Section 7.15
auto-implemented property initializers	Section 8.6.1
getter-only auto-implemented properties	Section 8.6.1
<code>nameof</code> operator	Section 10.5.1
null-conditional operator (<code>?.</code>)	Section 13.9.1
<code>when</code> clause for exception filtering	Section 13.10
<code>using static</code> directive	Section 21.3.1
null conditional operator (<code>?[]</code>)	Section 21.6
collection initializers for any collection with an <code>Add</code> extension method	Section 21.7
index initializers	Section 21.7

Fig. 1 | C# 6 new language features.

Interesting, Entertaining and Challenging Exercises

The book contains hundreds of exercises to practice the skills you learn. Extensive self-review exercises and answers are included for self-study. Also, each chapter concludes with a substantial set of exercises, which generally include

- simple recall of important terminology and concepts,
- identifying the errors in code samples,
- writing individual program statements,
- writing methods to perform specific tasks,
- writing C# classes,
- writing complete programs and
- implementing major projects.

Figure 2 lists only a sample of the book’s hundreds of exercises, including selections from our *Making-a-Difference* exercises set, which encourage you to use computers and the Internet to research and work on significant social problems. We hope you’ll approach these exercises in the context of your own values, politics and beliefs. The solutions to most of the book’s exercises are available only to college instructors who have adopted the book for their courses. See “Instructor Supplements” later in this Preface.

A sampling of the book's exercises

Carbon Footprint Calculator	Calculating the Value of $\log(x)$	ComplexNumber Class
Body-Mass-Index Calculator	Pythagorean Triples	Vehicle Inheritance Hierarchy
Attributes of Hybrid Vehicles	Global Warming Facts Quiz	Payroll System
Gender Neutrality	Tax Plan Alternative: The “FairTax”	Accounts Payable System
Stopwatch GUI	Rounding to a Specific Decimal Place	Polymorphic Banking Program
Login Form GUI	Hypotenuse of a Right Triangle	CarbonFootprint Interface: Polymorphism
Calculator GUI	Displaying a Hollow Right-Isosceles Triangle of Any Character	Length Conversions
Alarm Clock GUI	Separating Digits	Painter
Radio GUI	Temperature Conversions	Guess the Number Game
Displaying Shapes	Amicable Numbers	Ecofont
Odd or Even?	Prime Numbers	Typing Tutor
Is a Number a Multiple of Another?	Reversing Digits	Restaurant Bill Calculator
Separating an Integer's Digits	Letter Grades to a Four-Point Scale	Story Writer
Multiplication Table of a Number	Coin Tossing	Pig Latin
Account Class	Guess-the-Number Game	Cooking with Healthier Ingredients
Student Record Class	Distance Between Two Points	Spam Elimination
Asset Class	Craps Game with Betting	SMS Language
Coaching Class	Towers of Hanoi	File of Product Details
Removing Duplicated Code	Computer-Assisted Instruction	Telephone-Number Words
Target-Heart-Rate Calculator	Wages Rate	Student Poll
Computerizing Health Records	Identifying Multiples of Eight	Phishing Scanner
Inventory Level Calculator	Dice Game of Craps	Bucket Sort
Sales-Commission Calculator	Airline Reservations System	Palindromes
Discount Calculator	Knight's Tour Chess Puzzle	Evaluating Expressions with a Stack
Find the Two Largest Numbers	Eight Queens Chess Puzzle	Building Your Own Compiler
Dangling-else Problem	Sieve of Eratosthenes	Generic Linear Search
Expanded Form	Tortoise and the Hare	SortedDictionary of Colors
Decimal Equivalent of a Binary Number	Merging Arrays	Prime Factorization
Type of Parallelogram	Building Your Own Computer (Virtual Machine)	Bucket Sort with LinkedList<int>
Permutations	Polling	Sieve of Eratosthenes with BitArray
Infinite Series: Mathematical Constant e	Querying an Array of Invoice Objects	Credit-Inquiry Program
World Population Growth	Name Connector	Rolling a Die 60,000,000 Times
Enforcing Privacy with Cryptography	Hemisphere Class	Baseball Database App
Bar Chart Display	Depreciating-Value Class	Parsing with LINQ to XML
Prime Numbers	Set of Integers	I/O-Bound vs. Compute-Bound Apps
Calculating Sales	RationalNumber Class	Recursive Fibonacci
Car-Pool Savings Calculator	HugeInteger Class	
Gas Mileage Calculator	Tic-Tac-Toe	

Fig. 2 | A sampling of the book's exercises.

A Tour of the Book

This section discusses the book's modular organization to help instructors plan their syllabi.

Introduction to Computing, Visual C# and Visual Studio 2015 Community Edition

The chapters in this module of the book

- Chapter 1, Introduction to Computers, the Internet and Visual C#
- Chapter 2, Introduction to Visual Studio and Visual Programming

introduce hardware and software fundamentals, Microsoft's .NET platform and Visual Programming. The vast majority of the book's examples will run on Windows 7, 8 and 10 using the *Visual Studio 2015 Community* edition with which we test-drive a fun **Painter** app in Section 1.12. Chapter 1's introduction to object-oriented programming defines key terminology and discusses important concepts on which the rest of the book depends.

Introduction to C# Fundamentals

The chapters in this module of the book

- Chapter 3, Introduction to C# App Programming
- Chapter 4, Introduction to Classes, Objects, Methods and Strings
- Chapter 5, Algorithm Development and Control Statements: Part 1
- Chapter 6, Control Statements: Part 2
- Chapter 7, Methods: A Deeper Look
- Chapter 8, Arrays; Introduction to Exception Handling

present rich coverage of C# programming fundamentals (data types, operators, control statements, methods and arrays) and introduce object-oriented programming through a series of case studies. These chapters should be covered in order. Chapters 5 and 6 present a friendly treatment of control statements and problem solving. Chapters 7 and 8 present rich treatments of methods and arrays, respectively. Chapter 8 briefly introduces exception handling with an example that demonstrates attempting to access an element outside an array's bounds.

Object-Oriented Programming: A Deeper Look

The chapters in this module of the book

- Chapter 9, Introduction to LINQ and the `List` Collection
- Chapter 10, Classes and Objects: A Deeper Look
- Chapter 11, Object-Oriented Programming: Inheritance
- Chapter 12, OOP: Polymorphism and Interfaces
- Chapter 13, Exception Handling: A Deeper Look

provide a deeper look at object-oriented programming, including classes, objects, inheritance, polymorphism, interfaces and exception handling. An optional online two-chapter case study on designing and implementing the object-oriented software for a simple ATM is described later in this preface.

Chapter 9 introduces Microsoft's Language Integrated Query (LINQ) technology, which provides a uniform syntax for manipulating data from various data sources, such as

arrays, collections and, as you'll see in later chapters, databases and XML. Chapter 9 is intentionally simple and brief to encourage instructors to begin covering LINQ technology early. Section 9.4 introduces the `List` collection, which we use in Chapter 12. Later in the book, we take a deeper look at LINQ, using LINQ to Entities (for querying databases) and LINQ to XML. Chapter 9's LINQ coverage can be deferred if you're in a course which either skips LINQ or defers coverage until later in the book—it's required for one example in Chapter 17 (Fig. 17.6) and many of the later chapters starting with Chapter 22, Databases and LINQ.

Windows Forms Graphical User Interfaces (GUIs)

The chapters in this module of the book

- Chapter 14, Graphical User Interfaces with Windows Forms: Part 1
- Chapter 15, Graphical User Interfaces with Windows Forms: Part 2

present a detailed introduction to building GUIs using Windows Forms—instructors teaching Visual C# still largely prefer Windows Forms for their classes. Many of the examples in GUI Chapters 14–15 can be presented after Chapter 4. We also use Windows Forms GUIs in several other print and online chapters.

There are two other GUI technologies in Windows—Windows Presentation Foundation (WPF) and Universal Windows Platform (UWP). We provide optional online treatments of both.³

Strings and Files

The chapters in this module of the book

- Chapter 16, Strings and Characters: A Deeper Look
- Chapter 17, Files and Streams

present string processing and file processing, respectively. We introduce strings beginning in Chapter 4 and use them throughout the book. Chapter 16 investigates strings in more detail. Most of Chapter 16's examples can be presented at any point after Chapter 4. Chapter 17 introduces text-file processing and object-serialization for inputting and outputting entire objects. Chapter 17 requires Windows Forms concepts presented in Chapter 14.

Searching, Sorting and Generic Data Structures

The chapters in this module of the book:

- Chapter 18, Searching and Sorting
- Chapter 19, Custom Linked Data Structures
- Chapter 20, Generics
- Chapter 21, Generic Collections; Functional Programming with LINQ/PLINQ

introduce searching, sorting and data structures. Most C# programmers should use .NET's *built-in* searching, sorting and generic collections (prepackaged data structures) capabilities, which are discussed in Chapter 21. For instructors who wish to present how to implement customized searching, sorting and data structures capabilities, we provide Chapters 18–20, which require the concepts presented in Chapters 3–8 and 10–13. Chapter 18 presents sev-

3. As of Summer 2016, Windows Forms, WPF and UWP apps all can be posted for distribution via the Windows Store. See <http://bit.ly/DesktopToUWP> for more information.

eral searching and sorting algorithms and uses Big O notation to help you compare how hard each algorithm works to do its job—the code examples use especially visual outputs to show how the algorithms work. In Chapter 19, we show how to implement your own custom data structures, including lists, stacks, queues and binary trees. The data structures in Chapter 19 store references to objects. Chapter 20 introduces C# generics and demonstrates how to create type-safe generic methods and a type-safe generic stack data structure.

Functional Programming with LINQ, PLINQ, Lambdas, Delegates and Immutability

In addition to generic collections, Chapter 21 now introduces functional programming, showing how to use it with LINQ to Objects to write code more concisely and with fewer bugs than programs written using previous techniques. In Section 21.12, with one additional method call, we'll demonstrate how PLINQ (Parallel LINQ) can improve LINQ to Objects performance substantially on multicore systems. The chapter's exercises also ask you to reimplement earlier examples using functional-programming techniques.

Database with LINQ to Entities and SQL Server

The chapter in this module

- Chapter 22, Databases and LINQ

presents a novice-friendly introduction to database programming with the ADO.NET Entity Framework, LINQ to Entities and Microsoft's free version of SQL Server that's installed with the Visual Studio 2015 Community edition. The chapter's examples require C#, object-oriented programming and Windows Forms concepts presented in Chapters 3–14. Several online chapters require the techniques presented in this chapter.

Asynchronous Programming

The chapter in this module

- Chapter 23, Asynchronous Programming with `async` and `await`

shows how to take advantage of multicore architectures by writing applications that can process tasks asynchronously, which can improve app performance and GUI responsiveness in apps with long-running or compute-intensive tasks. The `async` modifier and `await` operator greatly simplify asynchronous programming, reduce errors and enable your apps to take advantage of the processing power in today's multicore computers, smartphones and tablets. In this edition, we added a case study that uses the Task Parallel Library (TPL), `async` and `await` in a GUI app—we keep a progress bar moving along in the GUI thread in parallel with a lengthy, compute-intensive calculation in another thread.

A Tour of the Online Content

The printed book contains the core content (Chapters 1–23) for introductory and intermediate course sequences. Several optional online topics for advanced courses and professionals are available on the book's password-protected Companion Website

<http://www.pearsonglobal editions.com/deitel>

New copies of this book come with a Companion Website access code that's located on the book's inside front cover. If the access code is already visible or there isn't an access code, you purchased a used book or an edition that does not come with an access code. Figure 3 lists the online topics, and Figure 4 lists a sample of the associated exercises.

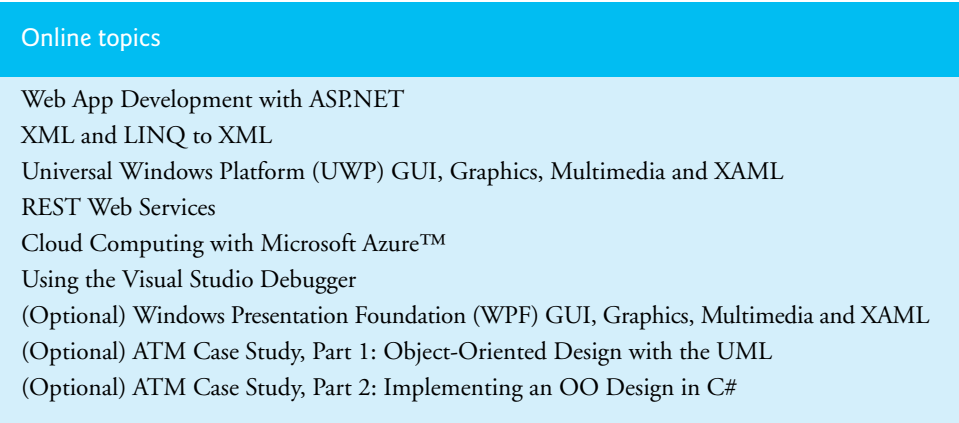


Fig. 3 | Online topics on the *Visual C# How to Program, 6/e* Companion Website.

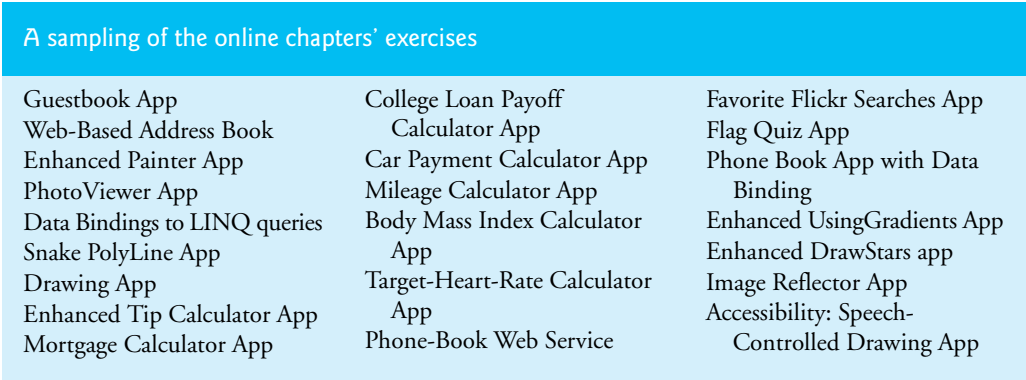


Fig. 4 | A sampling of the online chapters' exercises.

Web App Development with ASP.NET

Microsoft's .NET server-side technology, ASP.NET, enables you to create robust, scalable web-based apps. You'll build several apps, including a web-based guestbook that uses ASP.NET and the ADO .NET Entity Framework to store data in a database and display data in a web page.

Extensible Markup Language (XML)

The Extensible Markup Language (XML) is pervasive in the software-development industry, e-business and throughout the .NET platform. It's used in most of this book's online topics. XML is required to understand XAML—a Microsoft XML vocabulary that's used to describe graphical user interfaces, graphics and multimedia for Universal Windows Platform (UWP) GUI, graphics and multimedia apps, Windows 10 Mobile apps and Windows Presentation Foundation (WPF) apps. We present XML fundamentals, then discuss LINQ to XML, which allows you to query XML content using LINQ syntax.

Universal Windows Platform (UWP) for Desktop and Mobile Apps

The Universal Windows Platform (UWP) is designed to provide a common platform and user experience across all Windows devices, including personal computers, smartphones,

tablets, Xbox and even Microsoft’s new HoloLens virtual reality and augmented reality holographic headset—all using nearly identical code. We present GUI, graphics and multimedia apps, and demonstrate them on both personal computers and the smartphone emulator that comes with Visual Studio 2015 Community edition.

REST Web Services

Web services enable you to package app functionality in a manner that turns the web into a library of *reusable* services. We include a case study on building a math question generator web service that’s called by a math tutor app.

Building a Microsoft Azure™ Cloud Computing App

Microsoft Azure’s web services enable you to develop, manage and distribute your apps in “the cloud.” We’ll demonstrate how to use Azure web services to store an app’s data online.

Windows Presentation Foundation (WPF) GUI, Graphics and Multimedia

Windows Presentation Foundation (WPF)—created after Windows Forms and before UWP—is another Microsoft technology for building robust GUI, graphics and multimedia desktop apps. WPF provides you with complete control over all aspects of a GUI’s look-and-feel and includes multimedia capabilities that are not available in Windows Forms. We discuss WPF in the context of a painting app, a text editor, a color chooser, a book-cover viewer, a television video player, various animations, and speech synthesis and recognition apps.

We’re moving away from WPF in favor of UWP for creating apps that can run on desktop, mobile and other Windows devices. For this reason, the WPF introduction is provided *as is* from the previous edition—we will no longer evolve this material.

Optional Case Study: Using the UML to Develop an Object-Oriented Design and C# Implementation of an ATM (Automated Teller Machine)

The UML™ (Unified Modeling Language™) is the industry-standard graphical language for visually modeling object-oriented systems. We introduce the UML in the early chapters and provide an optional online object-oriented design case study in which we use the UML to design and implement the software for a simple ATM. We analyze a typical *requirements document* that specifies the details of the system to be built. We determine the *classes* needed to implement that system, the *attributes* the classes need to have, the *behaviors* the classes’ methods need to exhibit and we specify how the classes must *interact* with one another to meet the system requirements. From the design, we produce a complete working C# implementation. Students often report a “light bulb moment”—the case study helps them “tie it all together” and truly understand object orientation.

Teaching Approach

Visual C# How to Program, 6/e contains a rich collection of examples. We concentrate on building well-engineered software and stress program clarity.

Live-Code Approach. The book is loaded with “live-code” examples—most new concepts are presented in the context of complete working Visual C# apps, followed by one or more executions showing program inputs and outputs. In the few cases where we show a code snippet, to ensure correctness first we tested it in a complete working program then copied the code from the program and pasted it into the book.

Syntax Shading. For readability, we syntax shade the code, similar to the way Visual Studio colors the code. Our syntax-shading conventions are:

```
comments appear like this
keywords appear like this
constants and literal values appear like this
all other code appears in black
```

Code Highlighting. We emphasize key code segments by placing them in gray rectangles.

Using Fonts for Emphasis. We place the key terms and the index's page reference for each defining occurrence in colored **bold** text for easy reference. We show on-screen components in the **bold Helvetica** font (for example, the **File** menu) and Visual C# program text in the Lucida font (for example, `int count = 5;`). We use *italics* for emphasis.

Objectives. The chapter objectives preview the topics covered in the chapter.

Programming Tips. We include programming tips to help you focus on important aspects of program development. These tips and practices represent the best we've gleaned from a combined nine decades of programming and teaching experience.



Good Programming Practices

The Good Programming Practices call attention to techniques that will help you produce programs that are clearer, more understandable and more maintainable.



Common Programming Errors

Pointing out these Common Programming Errors reduces the likelihood that you'll make them.



Error-Prevention Tips

These tips contain suggestions for exposing and removing bugs from your programs; many of the tips describe aspects of Visual C# that prevent bugs from getting into programs.



Performance Tips

These tips highlight opportunities for making your programs run faster or minimizing the amount of memory that they occupy.



Portability Tips

These tips help you write code that will run on a variety of platforms.



Software Engineering Observations

The Software Engineering Observations highlight architectural and design issues that affect the construction of software systems, especially large-scale systems.



Look-and-Feel Observation 3.1

These observations help you design attractive, user-friendly graphical user interfaces that conform to industry norms.

Summary Bullets. We present a detailed bullet-list summary of each chapter.

Terminology. We include an alphabetized list of the important terms defined in each chapter.

Index. We've included an extensive index for reference. Defining occurrences of key terms in the index are highlighted with a colored **bold** page number.

Obtaining the Software Used in Visual C# How to Program, 6/e

We wrote the code examples in *Visual C# How to Program, 6/e* using Microsoft's free Visual Studio 2015 Community edition. See the Before You Begin section that follows this preface for download and installation instructions.

Instructor Supplements

The following supplements are available to *qualified instructors only* through Pearson Education's online Instructor Resource Center at www.pearsonglobaleditions.com/deitel:

- *Solutions Manual* contains solutions to *most* of the end-of-chapter exercises. We've included many *Making-a-Difference* exercises, most with solutions. **Please do not write to us requesting access to the Pearson Instructor's Resource Center. Access is restricted to college instructors who have adopted the book for their courses. Instructors can obtain access through their Pearson representatives.** If you're not a registered faculty member, contact your Pearson representative or visit <http://www.pearsonglobaleditions.com/deitel>. Exercise Solutions are *not* provided for "project" exercises. Check out our Programming Projects Resource Center for lots of additional exercise and project possibilities:

<http://www.deitel.com/ProgrammingProjects>

- *Test Item File* of multiple-choice questions (approximately two per top-level book section)
- *Customizable PowerPoint® slides* containing all the code and figures in the text, plus bulleted items that summarize the key points in the text.

Microsoft DreamSpark™

Microsoft provides many of its professional developer tools to students for free via a program called DreamSpark (<http://www.dreamspark.com>). See the website for details on verifying your student status so you take advantage of this program. To compile, test, debug and run this book's examples, you need only Windows 10 and the free Visual Studio 2015 Community edition. With the exception of the online UWP examples, the book's examples also will compile and run on Windows 7 and higher.

Acknowledgments

We'd like to thank Barbara Deitel of Deitel & Associates, Inc. She painstakingly researched the new capabilities of Visual C#, Visual Studio, .NET and other key technologies. We'd also like to acknowledge Frank McCown, Ph.D., Associate Professor of Computer Science, Harding University for his suggestion to include an example that used a `ProgressBar` with `async` and `await` in Chapter 23—so we ported to C# a similar example from our textbook *Java How to Program, 10/e*.

We're fortunate to have worked with the dedicated team of publishing professionals at Pearson Higher Education. We appreciate the guidance, wisdom, energy and mentorship of Tracy Johnson, Executive Editor, Computer Science. Kristy Alaura did an extraordinary job recruiting the book's reviewers and managing the review process. Bob Engelhardt did a wonderful job bringing the book to publication.

Reviewers

The book was scrutinized by academics teaching C# courses and industry C# experts. They provided countless suggestions for improving the presentation. Any remaining flaws in the book are our own.

Sixth Edition Reviewers: Qian Chen (Department of Engineering Technology: Computer Science Technology Program, Savannah State University), Octavio Hernandez (Microsoft Certified Solutions Developer, Principal Software Engineer at Advanced Bionics), José Antonio González Seco (Parliament of Andalusia, Spain), Bradley Sward (College of Dupage) and Lucian Wischik (Microsoft Visual C# Team).

Fifth Edition Post-Publication Reviewers: To help us prepare to write 6/e, the following academics reviewed 5/e and provided many helpful suggestions: Qian Chen (Savannah State University), Hongmei Chi (Florida A&M University), Kui Du (University of Massachusetts, Boston), James Leasure (Cuyahoga Community College West), Victor Miller (Ramapo College), Gary Savard (Champlain College) and Mohammad Yusuf (New Hampshire Technical Institute).

Other recent edition reviewers: Douglas B. Bock (MCSD.NET, Southern Illinois University Edwardsville), Dan Crevier (Microsoft), Shay Friedman (Microsoft Visual C# MVP), Amit K. Ghosh (University of Texas at El Paso), Marcelo Guerra Hahn (Microsoft), Kim Hamilton (Software Design Engineer at Microsoft and co-author of *Learning UML 2.0*), Huanhui Hu (Microsoft Corporation), Stephen Hustedde (South Mountain College), James Edward Keysor (Florida Institute of Technology), Narges Kasiri (Oklahoma State University), Helena Kotas (Microsoft), Charles Liu (University of Texas at San Antonio), Chris Lovett (Software Architect at Microsoft), Bashar Lulu (INETA Country Leader, Arabian Gulf), John McIlhinney (Spatial Intelligence; Microsoft MVP Visual Developer, Visual Basic), Ged Mead (Microsoft Visual Basic MVP, DevCity.net), Anand Mukundan (Architect, Polaris Software Lab Ltd.), Dr. Hamid R. Nemati (The University of North Carolina at Greensboro), Timothy Ng (Microsoft), Akira Onishi (Microsoft), Jeffrey P. Scott (Blackhawk Technical College), Joe Stagner (Senior Program Manager, Developer Tools & Platforms, Microsoft), Erick Thompson (Microsoft), Jesús Ubaldo Quevedo-Torrero (University of Wisconsin–Parkside, Department of Computer Science), Shawn Weisfeld (Microsoft MVP and President and Founder of UserGroup.tv) and Zijiang Yang (Western Michigan University).

As you read the book, we'd sincerely appreciate your comments, criticisms, corrections and suggestions for improving the text. Please address all correspondence to:

deitel@deitel.com

We'll respond promptly. It was fun writing *Visual C# How to Program, 6/e*—we hope you enjoy reading it!

Paul Deitel
Harvey Deitel

About the Authors

Paul Deitel, CEO and Chief Technical Officer of Deitel & Associates, Inc., has over 35 years of experience in computing. He is a graduate of MIT, where he studied Information Technology. Through Deitel & Associates, Inc., he has delivered hundreds of programming courses worldwide to clients, including Cisco, IBM, Boeing, Siemens, Sun Microsystems (now Oracle), Dell, Fidelity, NASA at the Kennedy Space Center, the National Severe Storm Laboratory, NOAA (National Oceanic and Atmospheric Administration), White Sands Missile Range, Rogue Wave Software, SunGard, Nortel Networks, Puma, iRobot, Invensys and many more. He and his co-author, Dr. Harvey Deitel, are the world's best-selling programming-language textbook/professional book/video authors.

Paul was named as a Microsoft® Most Valuable Professional (MVP) for C# in 2012–2014. According to Microsoft, “the Microsoft MVP Award is an annual award that recognizes exceptional technology community leaders worldwide who actively share their high quality, real-world expertise with users and Microsoft.” He also holds the Java Certified Programmer and Java Certified Developer designations and is an Oracle Java Champion.



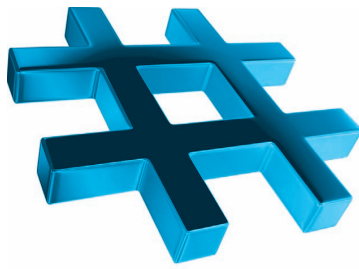
C# MVP 2012–2014

Dr. Harvey Deitel, Chairman and Chief Strategy Officer of Deitel & Associates, Inc., has over 55 years of experience in the computer field. Dr. Deitel earned B.S. and M.S. degrees in Electrical Engineering from MIT and a Ph.D. in Mathematics from Boston University—he studied computing in each of these programs before they spun off Computer Science programs. He has extensive college teaching experience, including earning tenure and serving as the Chairman of the Computer Science Department at Boston College before founding Deitel & Associates, Inc., in 1991 with his son, Paul. The Deitels' publications have earned international recognition, with translations published in Japanese, German, Russian, Spanish, French, Polish, Italian, Simplified Chinese, Traditional Chinese, Korean, Portuguese, Greek, Urdu and Turkish. Dr. Deitel has delivered hundreds of programming courses to academic, corporate, government and military clients.

Acknowledgments for the Global Edition

Pearson would like to thank and acknowledge Komal Arora for contributing to the Global Edition, and Ela Kashyap, Amity University Noida, Siddharth Nair, and Sandeep Singh, Jaypee Institute of Information Technology for reviewing the Global Edition.

This page intentionally left blank



Before You Begin

Please read this section before using the book to ensure that your computer is set up properly.

Font and Naming Conventions

We use fonts to distinguish between features, such as menu names, menu items, and other elements that appear in the program-development environment. Our convention is

- to emphasize Visual Studio features in a **sans-serif bold font** (e.g., **Properties** window) and
- to emphasize program text in a fixed-width sans-serif font (e.g., `bool x = true`).

Visual Studio 2015 Community Edition

This textbook uses Windows 10 and the free Microsoft Visual Studio 2015 Community edition—Visual Studio also can run on various older Windows versions. Ensure that your system meets Visual Studio 2015 Community edition’s minimum hardware and software requirements listed at:

<https://www.visualstudio.com/en-us/visual-studio-2015-system-requirements-vs>

Next, download the installer from

<https://www.visualstudio.com/products/visual-studio-express-vs>

then execute it and follow the on-screen instructions to install Visual Studio.

Though we developed the book’s examples on Windows 10, the examples will run on Windows 7 and higher—with the exception of those in our online Universal Windows Platform (UWP) presentation. Most examples without graphical user interfaces (GUIs) also will run on other C# and .NET implementations—see “If You’re Not Using Microsoft Visual C#...” later in this Before You Begin for more information.

Viewing File Extensions

Several screenshots in *Visual C# How to Program, 6/e* display file names with file-name extensions (e.g., .txt, .cs, .png, etc.). You may need to adjust your system’s settings to display file-name extensions. If you’re using Windows 7:

1. Open **Windows Explorer**.
2. Press the *Alt* key to display the menu bar, then select **Folder Options...** from the **Tools** menu.
3. In the dialog that appears, select the **View** tab.

4. In the **Advanced settings** pane, *uncheck* the box to the left of the text **Hide extensions for known file types**.
5. Click **OK** to apply the setting and close the dialog.

If you're using Windows 8 or higher:

1. Open **File Explorer**.
2. Click the **View** tab.
3. Ensure that the **File name extensions** checkbox is *checked*.

Obtaining the Source Code

Visual C# How to Program, 6/e's source-code examples are available for download at

<http://www.pearsonglobaleditions.com/deitel>

Click the **Examples** link to download the ZIP archive file to your computer—most browsers will save the file into your user account's **Downloads** folder. You can extract the ZIP file's contents using built-in Windows capabilities, or using a third-party archive-file tool such as WinZip (www.winzip.com) or 7-zip (www.7-zip.org).

Throughout the book, steps that require you to access our example code on your computer assume that you've extracted the examples from the ZIP file and placed them in your user account's **Documents** folder. You can extract them anywhere you like, but if you choose a different location, you'll need to update our steps accordingly. To extract the ZIP file's contents using the built-in Windows capabilities:

1. Open **Windows Explorer** (Windows 7) or **File Explorer** (Windows 8 and higher).
2. Locate the ZIP file on your system, typically in your user account's **Downloads** folder.
3. Right click the ZIP file and select **Extract All...**
4. In the dialog that appears, navigate to the folder where you'd like to extract the contents, then click the **Extract** button.

Configuring Visual Studio for Use with This Book

In this section, you'll use Visual Studio's **Options** dialog to configure several Visual Studio options. Setting these options is not required, but will make your Visual Studio match what we show in the book's Visual Studio screen captures.

Visual Studio Theme

Visual Studio has three color themes—**Blue**, **Dark** and **Light**. We used the **Blue** theme with light colored backgrounds to make the book's screen captures easier to read. To switch themes:

1. In the Visual Studio **Tools** menu, select **Options...** to display the **Options** dialog.
2. In the left column, select **Environment**.
3. Select the **Color theme** you wish to use.

Keep the **Options** dialog open for the next step.

Line Numbers

Throughout the book's discussions, we refer to code in our examples by line number. Many programmers find it helpful to display line numbers in Visual Studio as well. To do so:

1. Expand the **Text Editor** node in the **Options** dialog's left pane.
2. Select **All Languages**.
3. In the right pane, check the **Line numbers** checkbox.

Keep the **Options** dialog open for the next step.

Tab Size for Code Indents

Microsoft recommends four-space indents in source code, which is the Visual Studio default. Due to the fixed and limited width of code lines in print, we use three-space indents—this reduces the number of code lines that wrap to a new line, making the code a bit easier to read. If you wish to use three-space indents:

1. Expand the **C#** node in the **Options** dialog's left pane and select **Tabs**.
2. Ensure that **Insert spaces** is selected.
3. Enter **3** for both the **Tab size** and **Indent size** fields.
4. Click **OK** to save your settings.

If You're Not Using Microsoft Visual C#...

C# can be used on other platforms via two open-source projects managed by the .NET Foundation (<http://www.dotnetfoundation.org>)—the Mono Project and .NET Core.

Mono Project

The **Mono Project** is an open source, cross-platform C# and .NET Framework implementation that can be installed on Linux, OS X (soon to be renamed as macOS) and Windows. The code for most of the book's console (non-GUI) apps will compile and run using the Mono Project. Mono also supports Windows Forms GUI, which is used in Chapters 14–15 and several later examples. For more information and to download Mono, visit:

<http://www.mono-project.com/>

.NET Core

.NET Core is a new cross-platform .NET implementation for Windows, Linux, OS X and FreeBSD. The code for most of the book's console (non-GUI) apps will compile and run using .NET Core. At the time of this writing, a .NET Core version for Windows was available and versions were still under development for other platforms. For more information and to download .NET Core, visit:

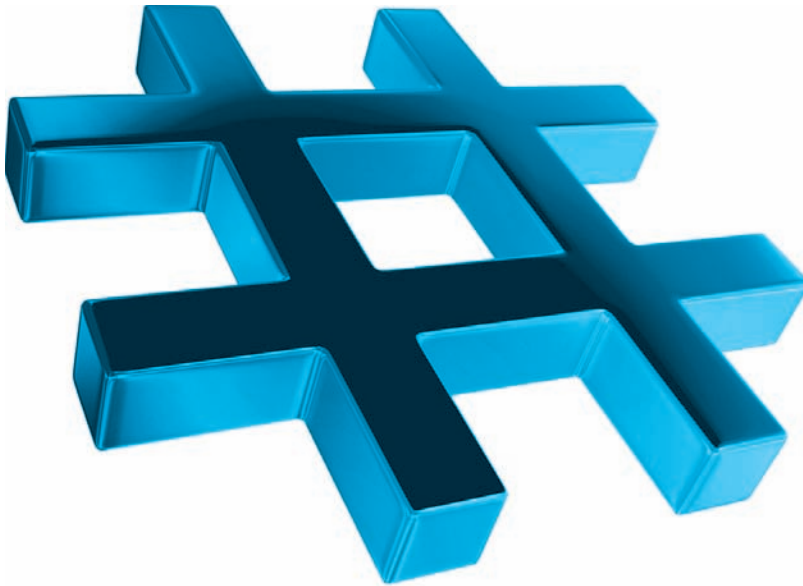
<https://dotnet.github.io/>

You're now ready to learn C# and the .NET platform with *Visual C# How to Program, 6/e*. We hope you enjoy the book!

This page intentionally left blank

Introduction to Computers, the Internet and Visual C#

1



Objectives

In this chapter you'll:

- Learn basic computer hardware, software and data concepts.
- Be introduced to the different types of computer programming languages.
- Understand the history of the Visual C# programming language and the Windows operating system.
- Learn what cloud computing with Microsoft Azure is.
- Understand the basics of object technology.
- Be introduced to the history of the Internet and the World Wide Web.
- Understand the parts that Windows, .NET, Visual Studio and C# play in the C# ecosystem.
- Test-drive a Visual C# drawing app.

- 1.1 Introduction
- 1.2 Computers and the Internet in Industry and Research
- 1.3 Hardware and Software
 - 1.3.1 Moore's Law
 - 1.3.2 Computer Organization
- 1.4 Data Hierarchy
- 1.5 Machine Languages, Assembly Languages and High-Level Languages
- 1.6 Object Technology
- 1.7 Internet and World Wide Web
- 1.8 C#
 - 1.8.1 Object-Oriented Programming
 - 1.8.2 Event-Driven Programming
 - 1.8.3 Visual Programming
 - 1.8.4 Generic and Functional Programming
 - 1.8.5 An International Standard
 - 1.8.6 C# on Non-Windows Platforms
 - 1.8.7 Internet and Web Programming
 - 1.8.8 Asynchronous Programming with `async` and `await`
 - 1.8.9 Other Key Programming Languages
- 1.9 Microsoft's .NET
 - 1.9.1 .NET Framework
 - 1.9.2 Common Language Runtime
 - 1.9.3 Platform Independence
 - 1.9.4 Language Interoperability
- 1.10 Microsoft's Windows® Operating System
- 1.11 Visual Studio Integrated Development Environment
- 1.12 Painter Test-Drive in Visual Studio Community

Self-Review Exercises | Answers to Self-Review Exercises | Exercises | Making-a-Difference Exercises | Making-a-Difference Resources

1.1 Introduction

Welcome to C#¹—a powerful computer-programming language that's easy for novices to learn and that professionals use to build substantial computer applications. Using this book, you'll write instructions commanding computers to perform powerful tasks. *Software* (i.e., the instructions you write) controls *hardware* (i.e., computers and related devices).

There are billions of personal computers in use and an even larger number of mobile devices with computers at their core. Since it was released in 2001, C# has been used primarily to build applications for personal computers and systems that support them. The explosive growth of mobile phones, tablets and other devices also is creating significant opportunities for programming mobile apps. With this new sixth edition of *Visual C# How to Program*, you'll be able to use Microsoft's new Universal Windows Platform (UWP) with Windows 10 to build C# apps for both personal computers and Windows 10 Mobile devices. With Microsoft's purchase of Xamarin, you also can develop C# mobile apps for Android devices and for iOS devices, such as iPhones and iPads.

1.2 Computers and the Internet in Industry and Research

These are exciting times in the computer field! Many of the most influential and successful businesses of the last two decades are technology companies, including Apple, IBM, Hewlett Packard, Dell, Intel, Motorola, Cisco, Microsoft, Google, Amazon, Facebook, Twitter, eBay and many more. These companies are major employers of people who study computer science, computer engineering, information systems or related disciplines. At the time of this writing, Google's parent company, Alphabet, and Apple were the two most

1. The name C#, pronounced "C-sharp," is based on the musical # notation for "sharp" notes.

valuable companies in the world. Figure 1.1 provides a few examples of the ways in which computers are improving people's lives in research, industry and society.

Name	Description
Electronic health records	These might include a patient's medical history, prescriptions, immunizations, lab results, allergies, insurance information and more. Making these available to health-care providers across a secure network improves patient care, reduces the probability of error and increases the health-care system's overall efficiency, helping control costs.
Human Genome Project	The Human Genome Project was founded to identify and analyze the 20,000+ genes in human DNA. The project used computer programs to analyze complex genetic data, determine the sequences of the billions of chemical base pairs that make up human DNA and store the information in databases, which have been made available over the Internet to researchers in many fields.
AMBER™ Alert	The AMBER (America's Missing: Broadcast Emergency Response) Alert System helps find abducted children. Law enforcement notifies TV and radio broadcasters and state transportation officials, who then broadcast alerts on TV, radio, computerized highway signs, the Internet and wireless devices. AMBER Alert partners with Facebook, whose users can "Like" AMBER Alert pages by location to receive alerts in their news feeds.
World Community Grid	People worldwide can donate their unused computer processing power by installing a free secure software program that allows the World Community Grid (http://www.worldcommunitygrid.org) to harness unused capacity. This computing power, accessed over the Internet, is used in place of expensive supercomputers to conduct scientific research projects that are making a difference—providing clean water to third-world countries, fighting cancer, growing more nutritious rice for regions fighting hunger and more.
Cloud computing	Cloud computing allows you to use software, hardware and information stored in the "cloud"—i.e., accessed on remote computers via the Internet and available on demand—popular examples are Dropbox, Google Drive and Microsoft OneDrive. You can increase or decrease resources incrementally to meet your needs at any given time, so cloud services can be more cost effective than purchasing expensive hardware to ensure that you have enough storage and processing power to meet peak-level needs. Using cloud-computing services shifts the burden of managing these applications from the business to the service provider, saving businesses time, effort and money. In an online chapter, you'll use Microsoft Azure —a cloud-computing platform that allows you to develop, manage and distribute your apps in the cloud. With Microsoft Azure, your apps can store their data in the cloud so that it's available at all times from any of your desktop computers and mobile devices. For information on Microsoft Azure's free and paid services visit https://azure.microsoft.com .
Medical imaging	X-ray computed tomography (CT) scans, also called CAT (computerized axial tomography) scans, take X-rays of the body from hundreds of different angles. Computers are used to adjust the intensity of the X-rays, optimizing the scan for each type of tissue, then to combine all of the information to create a 3D image. MRI scanners use a technique called magnetic resonance imaging to produce internal images noninvasively.

Fig. 1.1 | Improving people's lives with computers. (Part 1 of 2.)

Name	Description
GPS	Global Positioning System (GPS) devices use a network of satellites to retrieve location-based information. Multiple satellites send time-stamped signals to the GPS device, which calculates the distance to each satellite, based on the time the signal left the satellite and the time the signal arrived. This information helps determine the device's exact location. GPS devices can provide step-by-step directions and help you locate nearby businesses (restaurants, gas stations, etc.) and points of interest. GPS is used in numerous location-based Internet services such as check-in apps to help you find your friends (e.g., Foursquare and Facebook), exercise apps such as Map My Ride+, Couch to 5K and RunKeeper that track the time, distance and average speed of your outdoor ride or jog, dating apps that help you find a match nearby and apps that dynamically update changing traffic conditions.
Robots	Robots can be used for day-to-day tasks (e.g., iRobot's Roomba vacuuming robot), entertainment (e.g., robotic pets), military combat, deep sea and space exploration (e.g., NASA's Mars rover Curiosity) and more. Researchers, such as those at RoboHow (http://robohow.eu), are working to create autonomous robots that perform complex human manipulation tasks (such as cooking) and that can learn additional tasks both from the robots' own experiences and from observing humans performing other tasks.
E-mail, Instant Messaging and Video Chat	Internet-based servers support all of your online messaging. E-mail messages go through a mail server that also stores the messages. Instant Messaging (IM) and Video Chat apps, such as Facebook Messenger, WhatsApp, AIM, Skype, Yahoo! Messenger, Google Hangouts, Trillian and others, allow you to communicate with others in real time by sending your messages and live video through servers.
E-commerce	This technology has exploded with companies like Amazon, eBay, Alibaba, Walmart and many others, causing a major shift away from brick-and-mortar retailers.
Internet TV	Internet TV set-top boxes (such as Apple TV, Android TV, Roku, Chromecast and TiVo) allow you to access an enormous amount of content on demand, such as games, news, movies, television shows and more, and they help ensure that the content is streamed to your TV smoothly.
Streaming music services	Streaming music services (such as Apple Music, Pandora, Spotify and more) allow you to listen to large catalogues of music over the web, create customized "radio stations" and discover new music based on your feedback.
Self-driving cars and smart homes	These are two enormous markets. Self-driving cars are under development by many technology companies and car manufacturers—they already have an impressive safety record and soon could be widely used saving lives and reducing injuries. Smart homes use computers for security, climate control, minimizing energy costs, automated lighting systems, fire detection, window control and more.
Game programming	Global video-game revenues are expected to reach \$107 billion by 2017 (http://www.polygon.com/2015/4/22/8471789/worldwide-video-games-market-value-2015). The most sophisticated games can cost over \$100 million to develop, with the most expensive costing half a billion dollars (http://www.gamespot.com/gallery/20-of-the-most-expensive-games-ever-made/2900-104/). Bethesda's <i>Fallout 4</i> earned \$750 million in its first day of sales (http://fortune.com/2015/11/16/fallout4-is-quiet-best-seller/)!

Fig. 1.1 | Improving people's lives with computers. (Part 2 of 2.)

1.3 Hardware and Software

Computers can perform calculations and make logical decisions phenomenally faster than human beings can. Many of today's personal computers can perform billions of calculations in one second—more than a human can perform in a lifetime. *Supercomputers* are already performing *thousands of trillions (quadrillions)* of instructions per second! China's National University of Defense Technology's Tianhe-2 supercomputer can perform over 33 quadrillion calculations per second (33.86 *petaflops*)!² To put that in perspective, *the Tianhe-2 supercomputer can perform in one second about 3 million calculations for every person on the planet!* And supercomputing upper limits are growing quickly.

Computers (i.e., **hardware**) process *data* under the control of sequences of instructions called **computer programs**. These programs guide the computer through *actions* specified by people called **computer programmers**. The programs that run on a computer are referred to as **software**. In this book, you'll learn several key programming methodologies that are enhancing programmer productivity, thereby reducing software development costs—*object-oriented programming, generic programming, functional programming* and *structured programming*. You'll build C# apps (short for applications) for a variety of environments including the *desktop*, mobile devices like *smartphones* and *tablets*, and even “the *cloud*.”

Computers consist of devices referred to as hardware (e.g., the keyboard, screen, mouse, hard disks, memory, DVD drives and processing units). Computing costs are *dropping dramatically*, due to rapid developments in hardware and software technologies. Computers that filled large rooms and cost millions of dollars decades ago are now inscribed on silicon chips smaller than a fingernail, costing perhaps a few dollars each. Ironically, silicon is one of the most abundant materials on Earth—it's an ingredient in common sand. Silicon-chip technology has made computing so economical that computers have become a commodity.

1.3.1 Moore's Law

Every year, you probably expect to pay at least a little more for most products and services. The opposite has been the case in the computer and communications fields, especially with regard to the hardware supporting these technologies. For many decades, hardware costs have fallen rapidly.

Every year or two, the capacities of computers have approximately *doubled* inexpensively. This remarkable trend often is called **Moore's Law**, named for the person who identified it in the 1960s, Gordon Moore, co-founder of Intel—a leading manufacturer of the processors in today's computers and embedded systems. Moore's Law and related observations apply especially to the amount of memory that computers have for programs, the amount of secondary storage (such as disk storage) they have to hold programs and data over longer periods of time, and their processor speeds—the speeds at which they *execute* their programs (i.e., do their work). These increases make computers more capable, which puts greater demands on programming-language designers to innovate.

Similar growth has occurred in the communications field—costs have plummeted as enormous demand for communications *bandwidth* (i.e., information-carrying capacity) has attracted intense competition. We know of no other fields in which technology improves so quickly and costs fall so rapidly. Such phenomenal improvement is truly fostering the *Information Revolution*.

2. <http://www.top500.org>.

1.3.2 Computer Organization

Regardless of differences in *physical* appearance, computers can be envisioned as divided into various **logical units** or sections (Fig. 1.2).

Logical unit	Description
Input unit	This “receiving” section obtains information (data and computer programs) from input devices and places it at the disposal of the other units for processing. Most user input is entered into computers through keyboards, touch screens and mouse devices. Other forms of input include receiving voice commands, scanning images and barcodes, reading from secondary storage devices (such as hard drives, DVD drives, Blu-ray Disc™ drives and USB flash drives—also called “thumb drives” or “memory sticks”), receiving video from a webcam and having your computer receive information from the Internet (such as when you stream videos from YouTube® or download e-books from Amazon). Newer forms of input include position data from a GPS device, motion and orientation information from an <i>accelerometer</i> (a device that responds to up/down, left/right and forward/backward acceleration) in a smartphone or game controller (such as Microsoft® Kinect® for Xbox®, Wii™ Remote and Sony® PlayStation® Move) and voice input from devices like Amazon Echo and the forthcoming Google Home.
Output unit	This “shipping” section takes information the computer has processed and places it on various output devices to make it available for use outside the computer. Most information that’s output from computers today is displayed on screens (including touch screens), printed on paper (“going green” discourages this), played as audio or video on PCs and media players (such as Apple’s iPods) and giant screens in sports stadiums, transmitted over the Internet or used to control other devices, such as robots and “intelligent” appliances. Information is also commonly output to secondary storage devices, such as solid-state drives (SSDs), hard drives, DVD drives and USB flash drives. Popular recent forms of output are smartphone and game-controller vibration, virtual reality devices like Oculus Rift and Google Cardboard and mixed reality devices like Microsoft’s HoloLens.
Memory unit	This rapid-access, relatively low-capacity “warehouse” section retains information that has been entered through the input unit, making it immediately available for processing when needed. The memory unit also retains processed information until it can be placed on output devices by the output unit. Information in the memory unit is <i>volatile</i> —it’s typically lost when the computer’s power is turned off. The memory unit is often called either memory , primary memory or RAM (Random Access Memory). Main memories on desktop and notebook computers contain as much as 128 GB of RAM, though 2 to 16 GB is most common. GB stands for gigabytes; a gigabyte is approximately one billion bytes. A byte is eight bits. A bit is either a 0 or a 1.
Arithmetic and logic unit (ALU)	This “manufacturing” section performs <i>calculations</i> , such as addition, subtraction, multiplication and division. It also contains the <i>decision</i> mechanisms that allow the computer, for example, to compare two items from the memory unit to determine whether they’re equal. In today’s systems, the ALU is implemented as part of the next logical unit, the CPU.

Fig. 1.2 | Logical units of a computer. (Part 1 of 2.)

Logical unit	Description
Central processing unit (CPU)	This “administrative” section coordinates and supervises the operation of the other sections. The CPU tells the input unit when information should be read into the memory unit, tells the ALU when information from the memory unit should be used in calculations and tells the output unit when to send information from the memory unit to certain output devices. Many of today’s computers have multiple CPUs and, hence, can perform many operations simultaneously. A multicore processor implements multiple processors on a single integrated-circuit chip—a <i>dual-core processor</i> has two CPUs, a <i>quad-core processor</i> has four and an <i>octa-core processor</i> has eight. Today’s desktop computers have processors that can execute billions of instructions per second. Chapter 23 explores how to write apps that can take full advantage of multicore architecture.
Secondary storage unit	This is the long-term, high-capacity “warehousing” section. Programs or data not actively being used by the other units normally are placed on secondary storage devices (e.g., your <i>hard drive</i>) until they’re again needed, possibly hours, days, months or even years later. Information on secondary storage devices is <i>persistent</i> —it’s preserved even when the computer’s power is turned off. Secondary storage information takes much longer to access than information in primary memory, but its cost per unit is much less. Examples of secondary storage devices include solid-state drives (SSDs), hard drives, DVD drives and USB flash drives, some of which can hold over 2 TB (TB stands for terabytes; a terabyte is approximately one trillion bytes). Typical hard drives on desktop and notebook computers hold up to 2 TB, and some desktop hard drives can hold up to 6 TB.

Fig. 1.2 | Logical units of a computer. (Part 2 of 2.)

1.4 Data Hierarchy

Data items processed by computers form a **data hierarchy** that becomes larger and more complex in structure as we progress from the simplest data items (called “bits”) to richer data items, such as characters, fields, and so on. Figure 1.3 illustrates a portion of the data hierarchy.

Bits

The smallest data item in a computer can assume the value 0 or the value 1. It’s called a **bit** (short for “binary digit”—a digit that can assume one of *two* values). Remarkably, the impressive functions performed by computers involve only the simplest manipulations of 0s and 1s—*examining a bit’s value*, *setting a bit’s value* and *reversing a bit’s value* (from 1 to 0 or from 0 to 1).

Characters

It’s tedious for people to work with data in the low-level form of bits. Instead, they prefer to work with *decimal digits* (0–9), *letters* (A–Z and a–z), and *special symbols* (e.g., \$, @, %, &, *, (,), –, +, ", :, ? and /). Digits, letters and special symbols are known as **characters**. The computer’s **character set** is the set of all the characters used to write programs and represent data items. Computers process only 1s and 0s, so a computer’s character set represents every character as a pattern of 1s and 0s. C# supports various character sets (including **Unicode**®), with

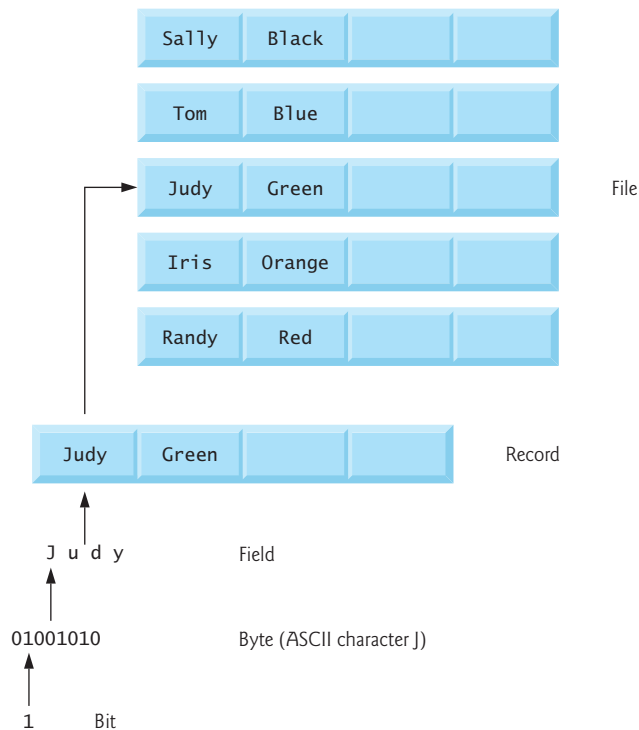


Fig. 1.3 | Data hierarchy.

some requiring more than one byte per character. Unicode supports many of the world's languages, as well as emojis. See Appendix B for more information on the [ASCII \(American Standard Code for Information Interchange\)](#) character set—the popular subset of Unicode that represents uppercase and lowercase letters of the English alphabet, digits and some common special characters. We also provide an online appendix describing Unicode.

Fields

Just as characters are composed of bits, **fields** are composed of characters or bytes. A field is a group of characters or bytes that conveys meaning. For example, a field consisting of uppercase and lowercase letters can be used to represent a person's name, and a field consisting of decimal digits could represent a person's age.

Records

Several related fields can be used to compose a **record**. In a payroll system, for example, the record for an employee might consist of the following fields (possible types for these fields are shown in parentheses):

- Employee or student identification number (a whole number).
- Name (a string of characters).
- Address (a string of characters).

- Hourly pay rate (a number with a decimal point).
- Year-to-date earnings (a number with a decimal point).
- Amount of taxes withheld (a number with a decimal point).

Thus, a record is a group of related fields. In the preceding example, all the fields belong to the *same* employee. A company might have many employees and a payroll record for each.

To facilitate the retrieval of specific records from a file, at least one field in each record is chosen as a **record key**, which identifies a record as belonging to a particular person or entity and distinguishes that record from all others. For example, in a payroll record, the employee identification number normally would be the record key.

Files

A **file** is a group of related records. More generally, a file contains arbitrary data in arbitrary formats. In some operating systems, a file is viewed simply as a *sequence of bytes*—any organization of the bytes in a file, such as organizing the data into records, is a view created by the application programmer. It's not unusual for an organization to have many files, some containing billions, or even trillions, of characters of information.

Database

A **database** is a collection of data organized for easy access and manipulation. The most popular model is the *relational database*, in which data is stored in simple *tables*. A table includes *records* and *fields*. For example, a table of students might include first name, last name, major, year, student ID number and grade-point-average fields. The data for each student is a record, and the individual pieces of information in each record are the fields. You can *search*, *sort* and otherwise manipulate the data based on its relationship to multiple tables or databases. For example, a university might use data from the student database in combination with data from databases of courses, on-campus housing, meal plans, etc.

Big Data

The amount of data being produced worldwide is enormous and growing quickly. According to IBM, approximately 2.5 quintillion bytes (2.5 *exabytes*) of data are created daily,³ and according to Salesforce.com, as of October 2015 90% of the world's data was created in just the prior 12 months!⁴ According to an IDC study, the global data supply will reach 40 *zettabytes* (equal to 40 trillion gigabytes) annually by 2020.⁵ Figure 1.4 shows some common byte measurements. **Big data** applications deal with massive amounts of data and this field is growing quickly, creating lots of opportunity for software developers. According to a study by Gartner Group, over four million IT jobs globally were expected to support big data in 2015.⁶

3. <http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html>.

4. <https://www.salesforce.com/blog/2015/10/salesforce-channel-ifttt.html>.

5. <http://recode.net/2014/01/10/stuffed-why-data-storage-is-hot-again-really/>.

6. <http://fortune.com/2013/09/04/the-big-data-employment-boom/>.

Unit	Bytes	Which is approximately
1 kilobyte (KB)	1024 bytes	10^3 (1024) bytes exactly
1 megabyte (MB)	1024 kilobytes	10^6 (1,000,000) bytes
1 gigabyte (GB)	1024 megabytes	10^9 (1,000,000,000) bytes
1 terabyte (TB)	1024 gigabytes	10^{12} (1,000,000,000,000) bytes
1 petabyte (PB)	1024 terabytes	10^{15} (1,000,000,000,000,000) bytes
1 exabyte (EB)	1024 petabytes	10^{18} (1,000,000,000,000,000,000) bytes
1 zettabyte (ZB)	1024 exabytes	10^{21} (1,000,000,000,000,000,000,000) bytes

Fig. 1.4 | Byte measurements.

1.5 Machine Languages, Assembly Languages and High-Level Languages

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate *translation* steps.

Machine Languages

Any computer can directly understand only its own **machine language** (also called *machine code*), defined by its hardware architecture. Machine languages generally consist of numbers (ultimately reduced to 1s and 0s). Such languages are cumbersome for humans.

Assembly Languages

Programming in machine language was simply too slow and tedious for most programmers. Instead, they began using English-like *abbreviations* to represent elementary operations. These abbreviations formed the basis of **assembly languages**. *Translator programs* called **assemblers** were developed to convert assembly-language programs to machine language. Although assembly-language code is clearer to humans, it's incomprehensible to computers until translated to machine language. Assembly languages are still popular today in applications where minimizing memory use and maximizing execution efficiency is crucial.

High-Level Languages

To speed up the programming process further, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks. High-level languages, such as C#, Visual Basic, C, C++, Java and Swift, allow you to write instructions that look more like everyday English and contain commonly used mathematical notations. Translator programs called **compilers** convert high-level language programs into machine language.

The process of compiling a large high-level-language program into machine language can take a considerable amount of computer time. **Interpreter** programs were developed to execute high-level language programs directly (without the need for compilation), although more slowly than compiled programs. **Scripting languages** such as the popular web languages JavaScript and PHP are processed by interpreters.



Performance Tip 1.1

Interpreters have an advantage over compilers in Internet scripting. An interpreted program can begin executing as soon as it's downloaded to the client's machine, without needing to be compiled before it can execute. On the downside, interpreted scripts generally run slower and consume more memory than compiled code. With a technique called JIT (just-in-time) compilation, interpreted languages can often run almost as fast as compiled ones.

I.6 Object Technology

C# is an object-oriented programming language. In this section we'll introduce the basics of object technology.

Building software quickly, correctly and economically remains an elusive goal at a time when demands for new and more powerful software are soaring. **Objects**, or more precisely—as we'll see in Chapter 4—the **classes** objects come from, are essentially *reusable* software components. There are date objects, time objects, audio objects, video objects, automobile objects, people objects, etc. Almost any *noun* can be reasonably represented as a software object in terms of *attributes* (e.g., name, color and size) and *behaviors* (e.g., calculating, moving and communicating). Software developers have discovered that using a modular, object-oriented design-and-implementation approach can make software-development groups much more productive than was possible with earlier techniques—object-oriented programs are often easier to understand, correct and modify.

The Automobile as an Object

Let's begin with a simple analogy. Suppose you want to *drive a car and make it go faster by pressing its accelerator pedal*. What must happen before you can do this? Well, before you can drive a car, someone has to *design* it. A car typically begins as engineering drawings, similar to the *blueprints* that describe the design of a house. These drawings include the design for an accelerator pedal. The pedal *hides* from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel *hides* the mechanisms that turn the car. This enables people with little or no knowledge of how engines, braking and steering mechanisms work to drive a car easily.

Before you can drive a car, it must be *built* from the engineering drawings that describe it. A completed car has an *actual* accelerator pedal to make the car go faster, but even that's not enough—the car won't accelerate on its own (hopefully!), so the driver must *press* the pedal to accelerate the car.

Methods and Classes

Let's use our car example to introduce some key object-oriented programming concepts. Performing a task in a program requires a **method**. The method houses the program statements that actually perform the task. It *hides* these statements from its user, just as a car's accelerator pedal hides from the driver the mechanisms of making the car go faster. In C#, we create a program unit called a class to house the set of methods that perform the class's tasks. For example, a class that represents a bank account might contain one method to *deposit* money to an account and another to *withdraw* money from an account. A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

Making Objects from Classes

Just as someone has to *build a car* from its engineering drawings before you can actually drive a car, you must *build an object* from a class before a program can perform the tasks that the class's methods define. The process of doing this is called *instantiation*. An object is then referred to as an **instance** of its class.

Reuse

Just as a car's engineering drawings can be *reused* many times to build many cars, you can *reuse* a class many times to build many objects. Reuse of existing classes when building new classes and programs saves time and effort. Reuse also helps you build more reliable and effective systems, because existing classes and components often have gone through extensive *testing* (to locate problems), *debugging* (to correct those problems) and *performance tuning*. Just as the notion of *interchangeable parts* was crucial to the Industrial Revolution, reusable classes are crucial to the software revolution that's been spurred by object technology.

Messages and Method Calls

When you drive a car, pressing its gas pedal sends a *message* to the car to perform a task—that is, to go faster. Similarly, you *send messages to an object*. Each message is implemented as a **method call** that tells a method of the object to perform its task. For example, a program might call a particular bank-account object's *deposit* method to increase the account's balance.

Attributes and Instance Variables

A car, besides having capabilities to accomplish tasks, also has *attributes*, such as its color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven (i.e., its odometer reading). Like its capabilities, the car's attributes are represented as part of its design in its engineering diagrams (which, for example, include an odometer and a fuel gauge). As you drive an actual car, these attributes are carried along with the car. Every car maintains its *own* attributes. For example, each car knows how much gas is in its own gas tank, but not how much is in the tanks of *other* cars.

An object, similarly, has attributes that it carries along as it's used in a program. These attributes are specified as part of the object's class. For example, a bank-account object has a *balance attribute* that represents the amount of money in the account. Each bank-account object knows the balance in the account it represents, but *not* the balances of the *other* accounts in the bank. Attributes are specified by the class's **instance variables**.

Properties, get Accessors and set Accessors

Attributes are not necessarily accessible directly. The car manufacturer does not want drivers to take apart the car's engine to observe the amount of gas in its tank. Instead, the driver can check the fuel gauge on the dashboard. The bank does not want its customers to walk into the vault to count the amount of money in an account. Instead, the customers talk to a bank teller or check personalized online bank accounts. Similarly, you do not need to have access to an object's instance variables in order to use them. You should use the **properties** of an object. Properties contain **get accessors** for reading the values of variables, and **set accessors** for storing values into them.

Encapsulation

Classes **encapsulate** (i.e., wrap) attributes and methods into objects created from those classes—an object’s attributes and methods are intimately related. Objects may communicate with one another, but they’re normally not allowed to know how other objects are implemented—implementation details are *hidden* within the objects themselves. This **information hiding**, as we’ll see, is crucial to good software engineering.

Inheritance

A new class of objects can be created quickly and conveniently by **inheritance**—the new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own. In our car analogy, an object of class “convertible” certainly *is an* object of the more *general* class “automobile,” but more *specifically*, the roof can be raised or lowered.

Object-Oriented Analysis and Design (OOAD)

Soon you’ll be writing programs in C#. How will you create the **code** (i.e., the program instructions) for your programs? Perhaps, like many programmers, you’ll simply turn on your computer and start typing. This approach may work for small programs (like the ones we present in the early chapters of the book), but what if you were asked to create a software system to control thousands of automated teller machines for a major bank? Or suppose you were asked to work on a team of thousands of software developers building the next generation of the U.S. air traffic control system? For projects so large and complex, you should not simply sit down and start writing programs.

To create the best solutions, you should follow a detailed **analysis** process for determining your project’s **requirements** (i.e., defining *what* the system is supposed to do) and developing a **design** that satisfies them (i.e., deciding *how* the system should do it). Ideally, you’d go through this process and carefully review the design (and have your design reviewed by other software professionals) before writing any code. If this process involves analyzing and designing your system from an object-oriented point of view, it’s called an **object-oriented analysis and design (OOAD) process**. Languages like C# are object oriented—programming in such a language, called **object-oriented programming (OOP)**, allows you to implement an object-oriented design as a working system.

The UML (Unified Modeling Language)

Although many different OOAD processes exist, a single graphical language for communicating the results of *any* OOAD process has come into wide use. This language, known as the Unified Modeling Language (UML), is now the most widely used graphical scheme for modeling object-oriented systems. We present our first UML diagrams in Chapters 4 and 5, then use them in our deeper treatment of object-oriented programming through Chapter 12. In our *optional* ATM Software Engineering Case Study in the online chapters, we present a simple subset of the UML’s features as we guide you through an object-oriented design and implementation experience.

1.7 Internet and World Wide Web

In the late 1960s, ARPA—the Advanced Research Projects Agency of the United States Department of Defense—rolled out plans for networking the main computer systems of

approximately a dozen ARPA-funded universities and research institutions. The computers were to be connected with communications lines operating at speeds on the order of 50,000 bits per second, a stunning rate at a time when most people (of the few who even had networking access) were connecting over telephone lines to computers at a rate of 110 bits per second. Academic research was about to take a giant leap forward. ARPA proceeded to implement what quickly became known as the ARPANET, the precursor to today's **Internet**. Today's fastest Internet speeds are on the order of billions of bits per second with trillion-bits-per-second speeds on the horizon!

Things worked out differently from the original plan. Although the ARPANET enabled researchers to network their computers, its main benefit proved to be the capability for quick and easy communication via what came to be known as electronic mail (e-mail). This is true even on today's Internet, with e-mail, instant messaging, file transfer and social media such as Facebook and Twitter enabling billions of people worldwide to communicate quickly and easily.

The protocol (set of rules) for communicating over the ARPANET became known as the **Transmission Control Protocol (TCP)**. TCP ensured that messages, consisting of sequentially numbered pieces called *packets*, were properly routed from sender to receiver, arrived intact and were assembled in the correct order.

The Internet: A Network of Networks

In parallel with the early evolution of the Internet, organizations worldwide were implementing their own networks for both intraorganization (that is, within an organization) and interorganization (that is, between organizations) communication. A huge variety of networking hardware and software appeared. One challenge was to enable these different networks to communicate with each other. ARPA accomplished this by developing the **Internet Protocol (IP)**, which created a true “network of networks,” the current architecture of the Internet. The combined set of protocols is now called **TCP/IP**.

Businesses rapidly realized that by using the Internet, they could improve their operations and offer new and better services to their clients. Companies started spending large amounts of money to develop and enhance their Internet presence. This generated fierce competition among communications carriers and hardware and software suppliers to meet the increased infrastructure demand. As a result, **bandwidth**—the information-carrying capacity of communications lines—on the Internet has increased tremendously, while hardware costs have plummeted.

The World Wide Web: Making the Internet User-Friendly

The **World Wide Web** (simply called “the web”) is a collection of hardware and software associated with the Internet that allows computer users to locate and view multimedia-based documents (documents with various combinations of text, graphics, animations, audios and videos) on almost any subject. In 1989, Tim Berners-Lee of CERN (the European Organization for Nuclear Research) began developing **HyperText Markup Language (HTML)**—the technology for sharing information via “hyperlinked” text documents. He also wrote communication protocols such as **HyperText Transfer Protocol (HTTP)** to form the backbone of his new hypertext information system, which he referred to as the World Wide Web.

In 1994, Berners-Lee founded the **World Wide Web Consortium (W3C)**, (<http://www.w3.org>), devoted to developing web technologies. One of the W3C's primary goals

is to make the web universally accessible to everyone regardless of disabilities, language or culture.

Web Services

Web services are software components stored on one computer that can be accessed by an app (or other software component) on another computer over the Internet. With web services, you can create *mashups*, which enable you to rapidly develop apps by combining complementary web services, often from multiple organizations, and possibly other forms of information feeds. For example, 100 Destinations (<http://www.100destinations.co.uk>) combines the photos and tweets from Twitter with the mapping capabilities of Google Maps to allow you to explore countries around the world through the photos of others.

ProgrammableWeb (<http://www.programmableweb.com/>) provides a directory of over 15,000 APIs and 6,200 mashups, plus how-to guides and sample code for creating your own mashups. According to Programmableweb, the three most widely used APIs for mashups are Google Maps, Twitter and YouTube.

Ajax

Ajax technology helps Internet-based applications perform like desktop applications—a difficult task, given that such applications suffer transmission delays as data is shuttled back and forth between your computer and server computers on the Internet. Using Ajax, applications like Google Maps have achieved excellent performance, approaching the look-and-feel of desktop applications.

The Internet of Things

The Internet is no longer just a network of computers—it's an **Internet of Things**. A *thing* is any object with an IP address—a unique identifier that helps locate that *thing* on the Internet—and the ability to send data automatically over the Internet. Such things include:

- a car with a transponder for paying tolls,
- monitors for parking-space availability in a garage,
- a heart monitor implanted in a human,
- monitors for drinkable water quality,
- a smart meter that reports energy usage,
- radiation detectors,
- item trackers in a warehouse,
- mobile apps that can track your movement and location,
- smart thermostats that adjust room temperatures based on weather forecasts and activity in the home
- and many more.

1.8 C#

In 2000, Microsoft announced the **C#** programming language. C# has roots in the C, C++ and Java programming languages. It has similar capabilities to Java and is appropriate for

the most demanding app-development tasks, especially for building today's desktop apps, large-scale enterprise apps, and web-based, mobile and cloud-based apps.

1.8.1 Object-Oriented Programming

C# is *object oriented*—we've discussed the basics of object technology and we present a rich treatment of object-oriented programming throughout the book. C# has access to the powerful **.NET Framework Class Library**—a vast collection of prebuilt classes that enable you to develop apps quickly (Fig. 1.5). We'll say more about .NET in Section 1.9.

Some key capabilities in the .NET Framework Class Library	
Database	Debugging
Building web apps	Multithreading
Graphics	File processing
Input/output	Security
Computer networking	Web communication
Permissions	Graphical user interface
Mobile	Data structures
String processing	Universal Windows Platform GUI

Fig. 1.5 | Some key capabilities in the .NET Framework Class Library.

1.8.2 Event-Driven Programming

C# graphical user interfaces (GUIs) are **event driven**. You can write programs that respond to user-initiated **events** such as mouse clicks, keystrokes, timer expirations and *touches* and *finger swipes*—gestures that are widely used on smartphones and tablets.

1.8.3 Visual Programming

Microsoft's Visual Studio enables you to use C# as a *visual programming language*—in addition to writing program statements to build portions of your apps, you'll also use Visual Studio to conveniently drag and drop predefined GUI objects like *buttons* and *text-boxes* into place on your screen, and label and resize them. Visual Studio will write much of the GUI code for you.

1.8.4 Generic and Functional Programming

Generic Programming

It's common to write a program that processes a collection of things—e.g., a collection of numbers, a collection of contacts, a collection of videos, etc. Historically, you had to program separately to handle each type of collection. With *generic programming*, you write code that handles a collection “in the general” and C# handles the specifics for each different type of collection, saving you a great deal of work. We'll study generics and generic collections in Chapters 20 and 21.

Functional Programming

With *functional programming*, you specify *what* you want to accomplish in a task, but *not how* to accomplish it. For example, with Microsoft’s LINQ—which we introduce in Chapter 9, then use in many later chapters—you can say, “Here’s a collection of numbers, give me the sum of its elements.” You do *not* need to specify the mechanics of walking through the elements and adding them into a running total one at a time—LINQ handles all that for you. Functional programming speeds application development and reduces errors. We take a deeper look at functional programming in Chapter 21.

1.8.5 An International Standard

C# has been standardized through ECMA International:

<http://www.ecma-international.org>

This enables other implementations of the language besides Microsoft’s Visual C#. At the time of this writing, the C# standard document—ECMA-334—was still being updated for C# 6. For information on ECMA-334, visit

<http://www.ecma-international.org/publications/standards/Ecma-334.htm>

Visit the Microsoft download center to find the latest version of Microsoft’s C# 6 specification, other documentation and software downloads.

1.8.6 C# on Non-Windows Platforms

Though C# was originally developed by Microsoft for the Windows platform, the language can be used on other platforms via the [Mono Project](#) and [.NET Core](#)—both are managed by the .NET Foundation

<http://www.dotnetfoundation.org/>

For more information, see the Before You Begin section after the Preface.

1.8.7 Internet and Web Programming

Today’s apps can be written with the aim of communicating among the world’s computers. As you’ll see, this is the focus of Microsoft’s .NET strategy. In online chapters, you’ll build web-based apps with C# and Microsoft’s [ASP.NET](#) technology.

1.8.8 Asynchronous Programming with `async` and `await`

In most programming today, each task in a program must finish executing before the next task can begin. This is called *synchronous programming* and is the style we use for most of this book. C# also allows *asynchronous programming* in which multiple tasks can be performed at the *same* time. Asynchronous programming can help you make your apps more responsive to user interactions, such as mouse clicks and keystrokes, among many other uses.

Asynchronous programming in early versions of Visual C# was difficult and error prone. C#’s `async` and `await` capabilities simplify asynchronous programming by enabling the compiler to hide much of the associated complexity from the developer. In Chapter 23, we provide an introduction to asynchronous programming with `async` and `await`.

1.8.9 Other Key Programming Languages

Figure 1.6 provides brief comments on several popular programming languages.

Programming language	Description
Ada	Ada, based on Pascal, was developed under the sponsorship of the U.S. Department of Defense (DOD) during the 1970s and early 1980s. The DOD wanted a single language that would fill most of its needs. The Pascal-based language was named after Lady Ada Lovelace, daughter of the poet Lord Byron. She's credited with writing the world's first computer program in the early 1800s (for the Analytical Engine mechanical computing device designed by Charles Babbage). Ada also supports object-oriented programming.
Basic	Basic was developed in the 1960s at Dartmouth College to familiarize novices with programming techniques. Many of its latest versions are object oriented.
C	C was developed in the early 1970s by Dennis Ritchie at Bell Laboratories. It initially became widely known as the UNIX operating system's development language. Today, most code for general-purpose operating systems is written in C or C++.
C++	C++, which is based on C, was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories. C++ provides several features that "spruce up" the C language, but more important, it provides capabilities for object-oriented programming.
COBOL	COBOL (COmmon Business Oriented Language) was developed in the late 1950s by computer manufacturers, the U.S. government and industrial computer users, based on a language developed by Grace Hopper, a career U.S. Navy officer and computer scientist. COBOL is still widely used for commercial applications that require precise and efficient manipulation of large amounts of data. Its latest version supports object-oriented programming.
Fortran	Fortran (FORmula TRANslator) was developed by IBM Corporation in the mid-1950s to be used for scientific and engineering applications that require complex mathematical computations. It's still widely used, and its latest versions support object-oriented programming.
Java	Sun Microsystems in 1991 funded an internal corporate research project led by James Gosling, which resulted in the C++-based object-oriented programming language called Java. A key goal of Java is to enable developers to write programs that will run on a great variety of computer systems and computer-controlled devices. This is sometimes called "write once, run anywhere." Java is used to develop large-scale enterprise applications, to enhance the functionality of web servers (the computers that provide the content we see in our web browsers), to provide applications for consumer devices (e.g., smartphones, tablets, television set-top boxes, appliances, automobiles and more) and for many other purposes. Java is also the key language for developing Android smartphone and tablet apps.
Objective-C	Objective-C is an object-oriented language based on C. It was developed in the early 1980s and later acquired by NeXT, which in turn was acquired by Apple. It became the key programming language for the OS X operating system and all iOS-powered devices (such as iPods, iPhones and iPads).

Fig. 1.6 | Some other programming languages. (Part I of 2.)

Programming language	Description
JavaScript	JavaScript is the most widely used scripting language. It's primarily used to add programmability to web pages—for example, animations and interactivity with the user. All major web browsers support it.
Pascal	Research in the 1960s resulted in <i>structured programming</i> —a disciplined approach to writing programs that are clearer, easier to test and debug and easier to modify than programs produced with previous techniques. The Pascal language, developed by Professor Niklaus Wirth in 1971, grew out of this research. It was popular for teaching structured programming for several decades.
PHP	PHP is an object-oriented, <i>open-source</i> “scripting” language supported by a community of developers and used by numerous websites. PHP is platform independent—implementations exist for all major UNIX, Linux, Mac and Windows operating systems.
Python	Python, another object-oriented scripting language, was released publicly in 1991. Developed by Guido van Rossum of the National Research Institute for Mathematics and Computer Science in Amsterdam (CWI), Python draws heavily from Modula-3—a systems programming language. Python is “extensible”—it can be extended through classes and programming interfaces.
Swift	Swift, which was introduced in 2014, is Apple's programming language of the future for developing iOS and OS X applications (apps). Swift is a contemporary language that includes popular programming-language features from languages such as Objective-C, Java, C#, Ruby, Python and others. In 2015, Apple released Swift 2 with new and updated features. According to the Tiobe Index, Swift has already become one of the most popular programming languages. Swift is now <i>open source</i> , so it can be used on non-Apple platforms as well.
Ruby on Rails	Ruby—created in the mid-1990s by Yukihiro Matsumoto—is an open-source, object-oriented programming language with a simple syntax that's similar to Python. Ruby on Rails combines the scripting language Ruby with the Rails web-application framework developed by the company 37Signals. Their book, <i>Getting Real</i> (free at http://gettingreal.37signals.com/toc.php), is a must-read for web developers. Many Ruby on Rails developers have reported productivity gains over other languages when developing database-intensive web applications.
Scala	Scala (http://www.scala-lang.org/what-is-scala.html)—short for “scalable language”—was designed by Martin Odersky, a professor at École Polytechnique Fédérale de Lausanne (EPFL) in Switzerland. Released in 2003, Scala uses both the object-oriented programming and functional programming paradigms and is designed to integrate with Java. Programming in Scala can reduce the amount of code in your applications significantly.
Visual Basic	Microsoft's Visual Basic language was introduced in the early 1990s to simplify the development of Microsoft Windows applications. Its latest versions support object-oriented programming.

Fig. 1.6 | Some other programming languages. (Part 2 of 2.)

1.9 Microsoft's .NET

In 2000, Microsoft announced its **.NET initiative** (www.microsoft.com/net), a broad vision for using the Internet and the web in the development, engineering, distribution and use of software. Rather than forcing you to use a single programming language, .NET permits you to create apps in *any* .NET-compatible language (such as C#, Visual Basic, Visual C++ and others). Part of the initiative includes Microsoft's ASP.NET technology for building web-based applications.

1.9.1 .NET Framework

The **.NET Framework Class Library** provides many capabilities that you'll use to build substantial C# apps quickly and easily. It contains *thousands* of valuable *prebuilt* classes that have been tested and tuned to maximize performance. You'll learn how to create your own classes, but you should *re-use* the .NET Framework classes whenever possible to speed up the software-development process, while enhancing the quality and performance of the software you develop.

1.9.2 Common Language Runtime

The **Common Language Runtime (CLR)**, another key part of the .NET Framework, executes .NET programs and provides functionality to make them easier to develop and debug. The CLR is a **virtual machine (VM)**—software that manages the execution of programs and hides from them the underlying operating system and hardware. The source code for programs that are executed and managed by the CLR is called *managed code*. The CLR provides various services to managed code, such as

- integrating software components written in different .NET languages,
- error handling between such components,
- enhanced security,
- automatic memory management and more.

Unmanaged-code programs do not have access to the CLR's services, which makes unmanaged code more difficult to write.⁷ Managed code is compiled into machine-specific instructions in the following steps:

1. First, the code is compiled into **Microsoft Intermediate Language (MSIL)**. Code converted into MSIL from other languages and sources can be woven together by the CLR—this allows programmers to work in their preferred .NET programming language. The MSIL for an app's components is placed into the app's *executable file*—the file that causes the computer to perform the app's tasks.
2. When the app executes, another compiler (known as the **just-in-time compiler** or **JIT compiler**) in the CLR translates the MSIL in the executable file into machine-language code (for a particular platform).
3. The machine-language code executes on that platform.

7. <http://msdn.microsoft.com/library/8bs2ecf4>.

I.9.3 Platform Independence

If the .NET Framework exists and is installed for a platform, that platform can run *any* .NET program. The ability of a program to run without modification across multiple platforms is known as **platform independence**. Code written once can be used on another type of computer without modification, saving time and money. In addition, software can target a wider audience. Previously, companies had to decide whether converting their programs to different platforms—a process called **porting**—was worth the cost. With .NET, porting programs is no longer an issue, at least once .NET itself has been made available on the platforms.

I.9.4 Language Interoperability

The .NET Framework provides a high level of **language interoperability**. Because software components written in different .NET languages (such as C# and Visual Basic) are all compiled into MSIL, the components can be combined to create a single unified program. Thus, MSIL allows the .NET Framework to be **language independent**.

The .NET Framework Class Library can be used by any .NET language. The latest release of .NET includes .NET 4.6 and .NET Core:

- .NET 4.6 introduces many improvements and new features, including ASP.NET 5 for web-based applications, improved support for today's high-resolution 4K screens and more.
- .NET Core is the cross-platform subset of .NET for Windows, Linux, OS X and FreeBSD.

I.10 Microsoft's Windows® Operating System

Microsoft's Windows is the most widely personal-computer, desktop operating system worldwide. **Operating systems** are software systems that make using computers more convenient for users, developers and system administrators. They provide *services* that allow each app to execute safely, efficiently and *concurrently* (i.e., in parallel) with other apps. Other popular desktop operating systems include Mac OS X and Linux. *Mobile operating systems* used in smartphones and tablets include Microsoft's Windows Phone, Google's Android, Apple's iOS (for iPhone, iPad and iPod Touch devices) and BlackBerry OS. Figure 1.7 presents the evolution of the Windows operating system.

Version	Description
Windows in the 1990s	In the mid-1980s, Microsoft developed the Windows operating system based on a graphical user interface with buttons, textboxes, menus and other graphical elements. The various versions released throughout the 1990s were intended for personal computing. Microsoft entered the corporate operating systems market with the 1993 release of <i>Windows NT</i> .

Fig. 1.7 | The evolution of the Windows operating system. (Part I of 2.)

Version	Description
Windows XP and Windows Vista	<i>Windows XP</i> was released in 2001 and combined Microsoft’s corporate and consumer operating-system lines. At the time of this writing, it still holds more than 10% of the operating-systems market (https://www.netmarketshare.com/operating-system-market-share.aspx). <i>Windows Vista</i> , released in 2007, offered the attractive new Aero user interface, many powerful enhancements and new apps and enhanced security. But Vista never caught on.
Windows 7	<i>Windows 7</i> is currently the world’s most widely used desktop operating system with over 47% of the operating-systems market (https://www.netmarketshare.com/operating-system-market-share.aspx). Windows added enhancements to the Aero user interface, faster startup times, further refinement of Vista’s security features, touch-screen with multitouch support, and more.
Windows 8 for Desktops and Tablets	Windows 8, released in 2012, provided a similar platform (the underlying system on which apps run) and <i>user experience</i> across a wide range of devices including personal computers, smartphones, tablets <i>and</i> the Xbox Live online game service. Its new look-and-feel featured a Start screen with <i>tiles</i> representing each app, similar to that of <i>Windows Phone</i> —Microsoft’s smartphone operating system. Windows 8 featured <i>multitouch</i> support for <i>touchpads</i> and <i>touchscreen</i> devices, enhanced security features and more.
Windows 8 UI (User Interface)	Windows 8 UI (previously called “Metro”) introduced a clean look-and-feel with minimal distractions to the user. Windows 8 apps featured a <i>chromeless window</i> with no borders, title bars and menus. These elements were <i>hidden</i> , allowing apps to fill the <i>entire</i> screen—particularly helpful on smaller screens such as tablets and smartphones. The interface elements were displayed in the <i>app bar</i> when the user <i>swiped</i> the top or bottom of the screen by holding down the mouse button, moving the mouse in the swipe direction and releasing the mouse button; or using a <i>finger swipe</i> on a touch-screen device.
Windows 10 and the Universal Windows Platform	Windows 10, released in 2015, is the current version of Windows and currently holds a 15% (and growing) share of the operating-systems market (https://www.netmarketshare.com/operating-system-market-share.aspx). In addition to many user-interface and other updates, Windows 10 introduced the Universal Windows Platform (UWP) , which is designed to provide a common platform (the underlying system on which apps run) and user experience across all Windows devices including personal computers, smartphones, tablets, Xbox and even Microsoft’s new HoloLens augmented reality holographic headset—all using nearly identical code.

Fig. 1.7 | The evolution of the Windows operating system. (Part 2 of 2.)

Windows Store

You can sell apps or offer them for free in the Windows Store. At the time of this writing, the fee to become a registered developer is \$19 for individuals and \$99 for companies. Microsoft retains 30% of the purchase price (more in some markets). See the App Developer Agreement for more information:

<https://msdn.microsoft.com/en-us/library/windows/apps/hh694058.aspx>

The Windows Store offers several business models for monetizing your app. You can charge full price for your app before download, with prices starting at \$1.49. You also can offer a time-limited trial or feature-limited trial that allows users to try the app before purchasing the full version, sell virtual goods (such as additional app features) using in-app purchases and more. To learn more about the Windows Store and monetizing your apps, visit

<https://msdn.microsoft.com/windows/uwp/monetize/index>

1.11 Visual Studio Integrated Development Environment

C# programs can be created using Microsoft's Visual Studio—a collection of software tools called an **Integrated Development Environment (IDE)**. The **Visual Studio Community** edition IDE enables you to *write, run, test* and *debug* C# programs quickly and conveniently. It also supports Microsoft's Visual Basic, Visual C++ and F# programming languages and more. Most of this book's examples were built using *Visual Studio Community*, which runs on Windows 7, 8 and 10. A few of the book's examples require Windows 10.

1.12 Painter Test-Drive in Visual Studio Community

You'll now use *Visual Studio Community* to “test-drive” an existing app that enables you to draw on the screen using the mouse. The **Painter** app allows you to choose among several brush sizes and colors. The elements and functionality you see in this app are typical of what you'll learn to program in this text. The following steps walk you through test-driving the app. For this test drive, we assume that you placed the book's examples in your user account's Documents folder in a subfolder named `examples`.

Step 1: Checking Your Setup

Confirm that you've set up your computer and the software properly by reading the book's Before You Begin section that follows the Preface.

Step 2: Locating the Painter App's Directory

Open a **File Explorer** (Windows 8 and 10) or **Windows Explorer** (Windows 7) window and navigate to

`C:\Users\yourUserName\Documents\examples\ch01`

Double click the **Painter** folder to view its contents (Fig. 1.8), then double click the `Painter.sln` file to open the app's solution in Visual Studio. An app's *solution* contains all of the app's *code files, supporting files* (such as *images, videos, data files*, etc.) and configuration information. We'll discuss the contents of a solution in more detail in the next chapter.

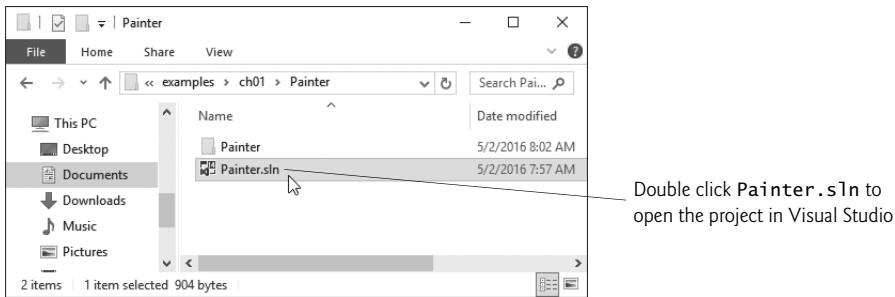


Fig. 1.8 | Contents of C:\examples\ch01\Painter.

Depending on your system configuration, **File Explorer** or **Windows Explorer** might display `Painter.sln` simply as `Painter`, without the filename extension `.sln`. To display the filename extensions in Windows 8 and higher:

1. Open **File Explorer**.
2. Click the **View** tab, then ensure that the **File name extensions** checkbox is checked.

To display them in Windows 7:

1. Open **Windows Explorer**.
2. Press *Alt* to display the menu bar, then select **Folder Options...** from **Windows Explorer**'s **Tools** menu.
3. In the dialog that appears, select the **View** tab.
4. In the **Advanced settings** pane, uncheck the box to the left of the text **Hide extensions for known file types**. [Note: If this item is already unchecked, no action needs to be taken.]
5. Click **OK** to apply the setting and close the dialog.

Step 3: Running the Painter App

To see the running **Painter** app, click the **Start** button (Fig. 1.9)



or press the *F5* key.

Figure 1.10 shows the running app and labels several of the app's graphical elements—called **controls**. These include **GroupBoxes**, **RadioButtons**, **Buttons** and a **Panel**. These controls and many others are discussed throughout the text. The app allows you to draw with a **Red**, **Blue**, **Green** or **Black** brush of **Small**, **Medium** or **Large** size. As you drag the mouse on the white **Panel**, the app draws circles of the specified color and size at the mouse pointer's current position. The slower you drag the mouse, the closer the circles will be. Thus, dragging slowly draws a continuous line (as in Fig. 1.11) and dragging quickly draws individual circles with space in between. You also can **Undo** your previous operation or **Clear** the drawing to start from scratch by pressing the **Buttons** below the **RadioButtons** in the GUI. By using existing *controls*—which are *objects*—you can create powerful apps much faster than if you had to write all the code yourself. This is a key benefit of *software reuse*.

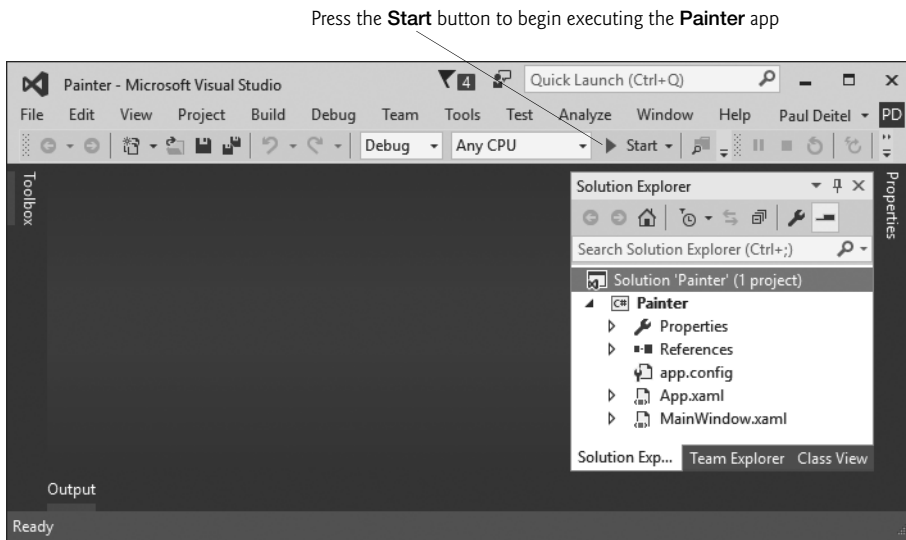


Fig. 1.9 | Running the **Painter** app.

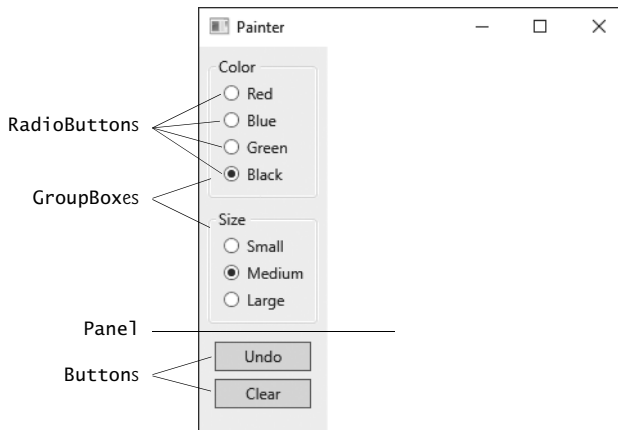


Fig. 1.10 | **Painter** app running in Windows 10.

The brush's properties, selected in the **RadioButtons** labeled **Black** and **Medium**, are *default settings*—the initial settings you see when you first run the app. Programmers include default settings to provide *reasonable* choices that the app will use if the user *does not* change the settings. Default settings also provide visual cues for users to choose their own settings. Now you'll choose your own settings as a user of this app.

Step 4: Changing the Brush Color

Click the **RadioButton** labeled **Red** to change the brush color, then click the **RadioButton** labeled **Small** to change the brush size. Position the mouse over the white **Panel**, then drag the mouse to draw with the brush. Draw flower petals, as shown in Fig. 1.11.

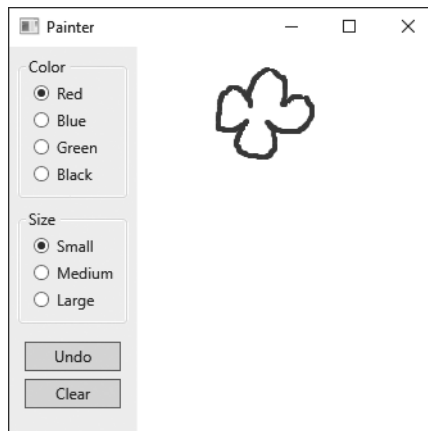


Fig. 1.11 | Drawing flower petals with a small red brush.

Step 5: Changing the Brush Color and Size

Click the **Green** RadioButton to change the brush color. Then, click the **Large** RadioButton to change the brush size. Draw grass and a flower stem, as shown in Fig. 1.12.

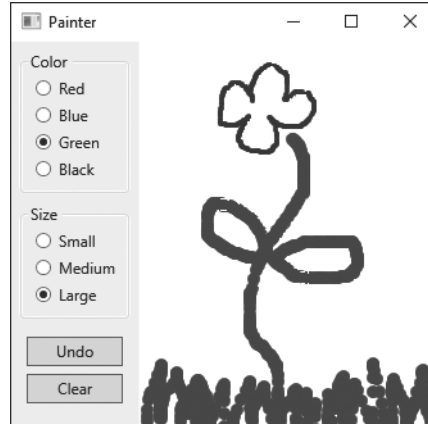


Fig. 1.12 | Drawing the flower stem and grass with a large green brush.

Step 6: Finishing the Drawing

Click the **Blue** and **Medium** RadioButton. Draw raindrops, as shown in Fig. 1.13, to complete the drawing.

Step 7: Stopping the App

When you run an app from Visual Studio, you can terminate it by clicking the stop button



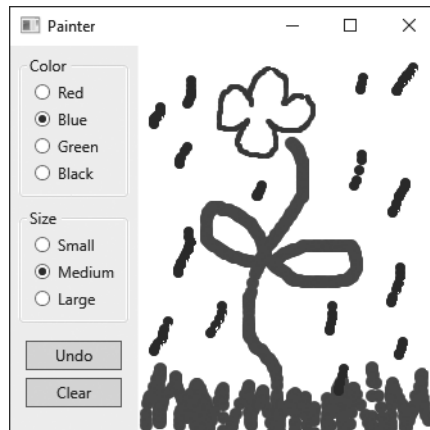


Fig. 1.13 | Drawing rain drops with a medium blue brush.

on the Visual Studio toolbar or by clicking the close box



on the running app's window.

Now that you've completed the test-drive, you're ready to begin developing C# apps. In Chapter 2, Introduction to Visual Studio and Visual Programming, you'll use Visual Studio to create your first C# program using *visual programming* techniques. As you'll see, Visual Studio will generate for you the code that builds the app's GUI. In Chapter 3, Introduction to C# App Programming, you'll begin writing C# programs containing conventional program code that you write.

Self-Review Exercises

- 1.1** Fill in the blanks in each of the following statements:
- Computers process data under the control of sequences of instructions called _____.
 - A computer consists of various devices referred to as _____, such as the keyboard, screen, mouse, hard disks, memory, DVD drives and processing units.
 - Data items processed by computers form a(n) _____ that becomes larger and more complex in structure as we progress from the simplest data items (called "bits") to richer data items, such as characters, fields, and so on.
 - Computers can directly understand only their _____ language, which is composed only of 1s and 0s.
 - The three types of computer programming languages discussed in the chapter are machine languages, _____ and _____.
 - Programs that translate high-level-language programs into machine language are called _____.
 - A(n) _____ processor implements several processors on a single "microchip"—a dual-core processor has two CPUs and a quad-core processor has four CPUs.
 - Windows 10 introduced the _____ for building Windows apps that run on desktop computers, notebook computers, tablets, phones, Xbox and even Microsoft's new HoloLens augmented reality holographic headset—all using nearly identical code.

1.2 Fill in the blanks in each of the following statements:

- Objects, or more precisely the _____ that objects come from, are essentially reusable software components.
- You send messages to an object. Each message is implemented as a method _____ that tells a method of the object to perform its task.
- A new class of objects can be created quickly and conveniently by _____; the new class absorbs the characteristics of an existing class, possibly customizing them and adding unique characteristics of its own.
- To create the best solutions, you should follow a detailed analysis process for determining your project's _____ (i.e., defining what the system is supposed to do) and developing a design that satisfies them (i.e., deciding how the system should do it).
- Visual C# is _____ driven. You'll write programs that respond to mouse clicks, key-strokes, timer expirations and touches and finger swipes.
- A key goal of Java is to be able to write programs that will run on a great variety of computer systems and computer-control devices. This is sometimes called _____.

1.3 Fill in the blanks in each of the following statements:

- The _____ executes .NET programs.
- The CLR provides various services to _____ code, such as integrating software components written in different .NET languages, error handling between such components, enhanced security and more.
- The ability of a program to run without modification across multiple platforms is known as platform _____.
- Visual Studio is a(n) _____ in which C# programs are developed.
- You can sell your own Windows Phone apps in the _____.

1.4 State whether each of the following is *true* or *false*. If *false*, explain why.

- Software objects model both abstract and real-world things.
- The most popular database model is the *relational database* in which data is stored in simple *tables*. A table includes *records* and *fields*.
- A database is a collection of data that's organized for easy access and manipulation.
- Secondary storage data takes much longer to access than data in primary memory, but the cost per unit of secondary storage is much higher than that of primary memory.
- High-level languages allow you to write instructions that look almost like everyday English and contain commonly used mathematical expressions.
- An object has attributes that it carries along as it's used in a program.
- The Transmission Control Protocol (TCP) ensures that messages, consisting of sequentially numbered pieces called bytes, were properly routed from sender to receiver, arrived intact and were assembled in the correct order.
- The information-carrying capacity of communications lines on the Internet has increased tremendously, while hardware costs have increased.
- You can build web-based apps with C# and Microsoft's ASP.NET technology.
- Java has become the key programming language for the Mac OS X desktop operating system and all iOS-based devices, such as iPods, iPhones and iPads.
- Microsoft's ASP.WEB technology is used to create web apps.
- Microsoft's Windows operating system is the most widely used desktop operating system worldwide.

1.5 Arrange these byte measurements in order from smallest to largest: terabyte, megabyte, petabyte, gigabyte and kilobyte.

1.6 Describe the two-step translation process for preparing your C# code to execute on your particular computer.

Answers to Self-Review Exercises

- 1.1** a) computer programs. b) hardware. c) data hierarchy. d) machine. e) assembly languages, high-level languages. f) compilers. g) multicore. h) Universal Windows Platform (UWP).
- 1.2** a) classes. b) call. c) inheritance. d) requirements. e) event. f) write once, run anywhere.
- 1.3** a) Common Language Runtime (CLR) of the .NET Framework. b) managed. c) independence. d) IDE. e) Windows Store.
- 1.4** a) True. b) True. c) True. d) False: The cost per unit of secondary storage is much lower than that of primary memory. e) True. f) True. g) False. The pieces are called packets, not bytes. h) False. Hardware costs have decreased. i) True. j) False. The language is Swift, not Java. k) False. It's ASP.NET technology. l) True.
- 1.5** kilobyte, megabyte, gigabyte, terabyte, petabyte.
- 1.6** C# code is first compiled into MSIL and placed in an executable file. When the app executes, another compiler called the JIT (just-in-time) compiler in the CLR translates the MSIL in the executable file into machine-language code (for a particular platform).

Exercises

- 1.7** Fill in the blanks in each of the following statements:
- Regardless of differences in physical appearance, computers can be envisioned as divided into various _____.
 - Systems such as smartphones, appliances, game controllers, cable set-top boxes and automobiles that contain small computers are called _____.
 - Just as characters are composed of bits, _____ are composed of characters or bytes.
 - Information on secondary storage devices is _____; it's preserved even when the computer's power is turned off.
 - _____ provide *services* that allow each app to execute safely, efficiently and *concurrently* with other apps.
 - In object-oriented programming languages, we create a program unit called a(n) _____ to house the set of methods that perform its tasks.
 - Use a building-block approach to creating your programs. Avoid reinventing the wheel—use existing pieces wherever possible. Such software _____ is a key benefit of object-oriented programming.
- 1.8** Fill in the blanks in each of the following statements:
- Although many different OOAD processes exist, a single graphical language for communicating the results of *any* OOAD process has come into wide use. This language, known as the _____, is now the most widely used graphical scheme for modeling object-oriented systems.
 - Tim Berners-Lee developed the _____ for sharing information via “hyperlinked” text documents on the web.
 - The CLR is a(n) _____ machine. It is software that manages the execution of programs and hides from them the underlying operating system and hardware.
 - Converting a program to run on a different platform from which it was originally intended is called _____.
 - Microsoft's Windows _____ is a cloud-computing platform that allows you to develop, manage and distribute your apps in the cloud.
 - By using existing controls—which are objects—you can create powerful apps much faster than if you had to write all the code yourself. This is a key benefit of software _____.

- 1.9** State whether each of the following is *true* or *false*. If *false*, explain why.
- The smallest data item in a computer can assume the value 1 or the value 2. Such a data item is called a **bit** (short for “binary digit”—a digit that can assume either of *two* values).
 - The Unicode character set is a popular subset of ASCII that represents uppercase and lowercase letters, digits and some common special characters.
 - The Central Processing Unit acts like an “administrative” section that coordinates and supervises the operation of the other sections like input unit, output unit, ALU etc.
 - Reuse helps you build more reliable and effective systems, because existing classes and components often have gone through extensive testing, debugging and performance tuning.
 - One of the W3C’s primary goals is to make the web universally accessible to everyone regardless of disabilities, language or culture.
 - C# is available only on Microsoft Windows.
 - C# apps can be developed only for desktop environments.
 - .NET programs can run on any platform.
 - The Universal Windows Platform (UWP) is designed to provide a common platform (the underlying system on which apps run) and user experience across all of your devices including personal computers, smartphones, tablets and Xbox Live.
- 1.10** What is a key advantage of interpreters over compilers? What is a key disadvantage?
- 1.11** What is the key advantage of using the new `async` feature in preference to using old-style multithreading?
- 1.12** How has Moore’s Law made computers more capable?
- 1.13** Why is using cloud-computing resources sometimes preferable to purchasing all the hardware you need for your own computer?
- 1.14** Which of the following falls under the class of hardware, software or a combination of both?
- Memory Unit
 - World Wide Web
 - .NET Framework class library
 - ALU
 - OS
- 1.15** .NET apps can be created using *any* .NET-compatible language such as C#, Visual Basic, Visual C++ and others. State the following statements as True or False in the context:
- The .NET Framework Class Library has many capabilities which can be used to build many C# apps quickly and easily.
 - The .NET framework class library contains very few valuable *prebuilt* classes that have been tested and tuned to maximize performance.
 - The .NET has JIT Compiler that executes .NET programs and provides functionality to make them easier to develop and debug.
 - .NET Framework classes are reused to speed up the software-development process.
- 1.16** Write the expanded forms of the following abbreviations:
- ASCII
 - JITC
 - TCP/IP
 - OOAD
 - UWP
 - GPS
- 1.17** What are the key benefits of the .NET Framework and the CLR? What are the drawbacks?

- 1.18** How does OOAD help in designing large and complex systems?
- 1.19** While observing a real-life car, you decide to link it with the concepts of the object oriented programming paradigm. Discuss the notion of a car using this analogy—the class it belongs to, objects of the same type, the attributes and the behaviors associated with objects.
- 1.20** What is the key accomplishment of the UML?
- 1.21** What did the chief benefit of the early Internet prove to be?
- 1.22** What is the key capability of the web?
- 1.23** What is the key vision of Microsoft's .NET initiative?
- 1.24** How does the .NET Framework Class Library facilitate the development of .NET apps?
- 1.25** Besides the obvious benefits of reuse made possible by OOP, what do many organizations report as another key benefit of OOP?

Making-a-Difference Exercises

The Making-a-Difference exercises will ask you to work on problems that really matter to individuals, communities, countries and the world.

1.26 (*Test Drive: Carbon Footprint Calculator*) Some scientists believe that carbon emissions, especially from the burning of fossil fuels, contribute significantly to global warming and that this can be combatted if individuals take steps to limit their use of carbon-based fuels. Various organizations and individuals are increasingly concerned about their “carbon footprints.” Websites such as TerraPass

<http://www.terrapass.com/carbon-footprint-calculator-2/>

and Carbon Footprint

<http://www.carbonfootprint.com/calculator.aspx>

provide carbon-footprint calculators. Test drive these calculators to determine your carbon footprint. Exercises in later chapters will ask you to program your own carbon-footprint calculator. To prepare for this, research the formulas for calculating carbon footprints.

1.27 (*Test Drive: Body-Mass-Index Calculator*) By recent estimates, two-thirds of the people in the United States are overweight and about half of those are obese. This causes significant increases in illnesses such as diabetes and heart disease. To determine whether a person is overweight or obese, you can use a measure called the body mass index (BMI). The United States Department of Health and Human Services provides a BMI calculator at <http://www.nhlbi.nih.gov/guidelines/obesity/BMI/bmicalc.htm>. Use it to calculate your own BMI. An exercise in Chapter 3 will ask you to program your own BMI calculator. To prepare for this, research the formulas for calculating BMI.

1.28 (*Attributes of Hybrid Vehicles*) In this chapter you learned the basics of classes. Now you'll begin “fleshing out” aspects of a class called “Hybrid Vehicle.” Hybrid vehicles are becoming increasingly popular, because they often get much better mileage than purely gasoline-powered vehicles. Browse the web and study the features of four or five of today's popular hybrid cars, then list as many of their hybrid-related attributes as you can. For example, common attributes include city-miles-per-gallon and highway-miles-per-gallon. Also list the attributes of the batteries (type, weight, etc.).

1.29 (*Gender Neutrality*) Some people want to eliminate sexism in all forms of communication. You've been asked to create a program that can process a paragraph of text and replace gender-specific words with gender-neutral ones. Assuming that you've been given a list of gender-specific words and their gender-neutral replacements (e.g., replace “wife” with “spouse,” “man” with “person,” “daughter” with “child” and so on), explain the procedure you'd use to read through a para-

graph of text and manually perform these replacements. How might your procedure generate a strange term like “woperchild,” which is actually listed in the Urban Dictionary (www.urbandictionary.com)? In Chapter 5, you’ll learn that a more formal term for “procedure” is “algorithm,” and that an algorithm specifies the steps to be performed and the order in which to perform them.

Making-a-Difference Resources

The *Microsoft Imagine Cup* is a global competition in which students use technology to try to solve some of the world’s most difficult problems, such as environmental sustainability, ending hunger, emergency response, literacy and more. For more information about the competition and to learn about previous winners’ projects, visit <https://www.imaginecup.com/Custom/Index/About>. You can also find several project ideas submitted by worldwide charitable organizations. For additional ideas for programming projects that can make a difference, search the web for “making a difference” and visit the following websites:

<http://www.un.org/millenniumgoals>

The United Nations Millennium Project seeks solutions to major worldwide issues such as environmental sustainability, gender equality, child and maternal health, universal education and more.

<http://www.ibm.com/smarterplanet>

The IBM® Smarter Planet website discusses how IBM is using technology to solve issues related to business, cloud computing, education, sustainability and more.

<http://www.gatesfoundation.org>

The Bill and Melinda Gates Foundation provides grants to organizations that work to alleviate hunger, poverty and disease in developing countries.

<http://nethope.org>

NetHope is a collaboration of humanitarian organizations worldwide working to solve technology problems such as connectivity, emergency response and more.

<http://www.rainforestfoundation.org>

The Rainforest Foundation works to preserve rainforests and to protect the rights of the indigenous people who call the rainforests home. The site includes a list of things you can do to help.

<http://www.undp.org>

The United Nations Development Programme (UNDP) seeks solutions to global challenges such as crisis prevention and recovery, energy and the environment, democratic governance and more.

<http://www.unido.org>

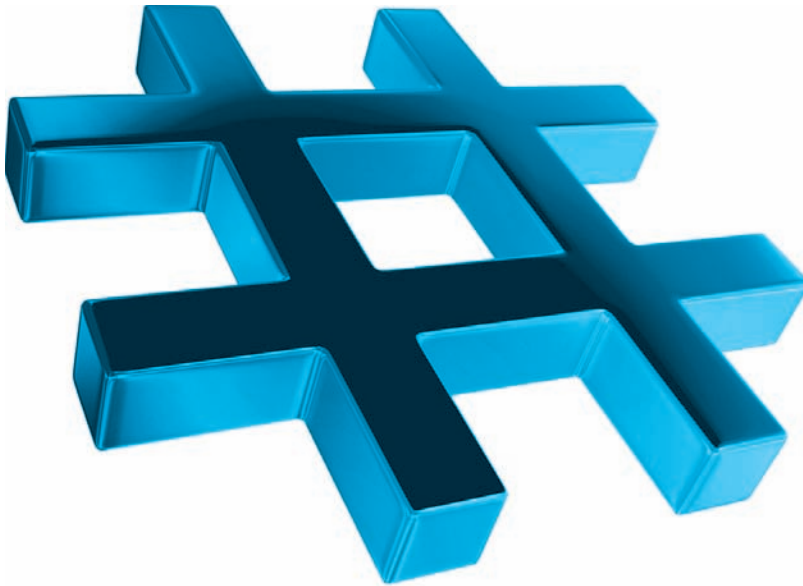
The United Nations Industrial Development Organization (UNIDO) seeks to reduce poverty, give developing countries the opportunity to participate in global trade, and promote energy efficiency and sustainability.

<http://www.usaid.gov/>

USAID promotes global democracy, health, economic growth, conflict prevention, humanitarian aid and more.

Introduction to Visual Studio and Visual Programming

2



Objectives

In this chapter you'll:

- Learn the basics of the Visual Studio Community 2015 Integrated Development Environment (IDE) for writing, running and debugging your apps.
- Create a new project using Visual Studio's **Windows Forms Application** template.
- Be introduced to Windows Forms and the controls you'll use to build graphical user interfaces.
- Use key commands contained in the IDE's menus and toolbars.
- Understand the purpose of the various windows in the Visual Studio.
- Use Visual Studio's help features.
- Use visual app development to conveniently create, compile and execute a simple Visual C# app that displays text and an image.

- 2.1 Introduction
- 2.2 Overview of the Visual Studio Community 2015 IDE
 - 2.2.1 Introduction to Visual Studio Community 2015
 - 2.2.2 Visual Studio Themes
 - 2.2.3 Links on the Start Page
 - 2.2.4 Creating a New Project
 - 2.2.5 **New Project** Dialog and Project Templates
 - 2.2.6 Forms and Controls
- 2.3 Menu Bar and Toolbar
- 2.4 Navigating the Visual Studio IDE
 - 2.4.1 **Solution Explorer**
 - 2.4.2 **Toolbox**
 - 2.4.3 **Properties** Window
- 2.5 Help Menu and Context-Sensitive Help
- 2.6 Visual Programming: Creating a Simple App that Displays Text and an Image
- 2.7 Wrap-Up
- 2.8 Web Resources

Summary | Terminology | Self-Review Exercises | Answers to Self-Review Exercises | Exercises

2.1 Introduction

Visual Studio is Microsoft's Integrated Development Environment (IDE) for creating, running and debugging apps (also called applications) written in C# and various other .NET programming languages. In this chapter, we overview the Visual Studio Community 2015 IDE, then show how to create a simple Visual C# app by dragging and dropping predefined building blocks into place—a technique known as **visual app development**.

2.2 Overview of the Visual Studio Community 2015 IDE

There are several versions of Visual Studio. This book's examples, screen captures and discussions are based on the free **Visual Studio Community 2015** running on Windows 10. See the Before You Begin section that follows the Preface for information on installing the software. With few exceptions, this book's examples can be created and run on Windows 7, 8.x or 10—we'll point out any examples that require Windows 10.

The examples will work on full versions of Visual Studio as well—though some options, menus and instructions might differ. From this point forward, we'll refer to Visual Studio Community 2015 simply as “Visual Studio” or “the IDE.” We assume that you have some familiarity with Windows.

2.2.1 Introduction to Visual Studio Community 2015

[*Note:* We use the > character to indicate when you should select a *menu item* from a *menu*. For example, notation **File > Save All** means that you should select the **Save All** menu item from the **File** menu.]

To begin, open Visual Studio. On Windows 10, click



then select **All Apps > Visual Studio 2015**. On Windows 7, click



then select **All Programs > Visual Studio 2015**. On Windows 8's **Start** screen, locate and click the Visual Studio 2015 tile, which will contain the following icon:



Initially, Visual Studio displays the **Start Page** (Fig. 2.1). Depending on your version of Visual Studio, your **Start Page** may look different. The **Start Page** contains a list of links to Visual Studio resources and web-based resources. At any time, you can return to the **Start Page** by selecting **View > Start Page**.

2.2.2 Visual Studio Themes

Visual Studio supports three themes that specify the IDE's color scheme:

- a *dark theme* (with dark window backgrounds and light text)
- a *light theme* (with light window backgrounds and dark text) and
- a *blue theme* (with light window backgrounds and dark text).

We use the blue theme throughout this book. The Before You Begin section after the Preface explains how to set this option.

2.2.3 Links on the Start Page

The **Start Page** links are organized into two columns. The left column's **Start** section contains options for building new apps or working on existing ones. The left column's **Recent** section contains links to projects you've recently created or modified.

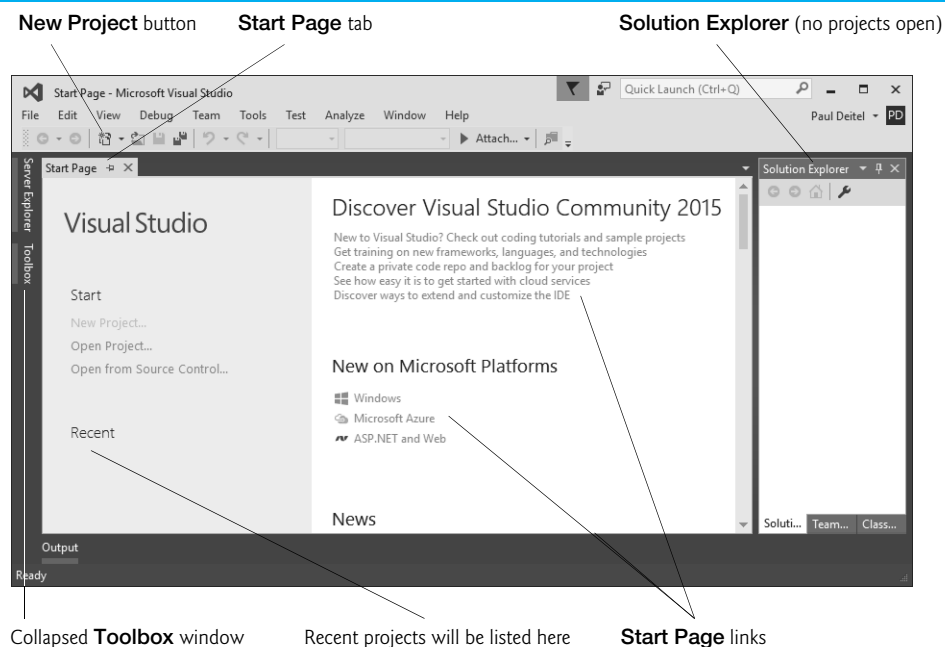


Fig. 2.1 | **Start Page** in Visual Studio Community 2015.

The **Start Page**'s right column—with **Discover Visual Studio Community 2015** at the top—contains links to various online documentation and resources to help you get started with Visual Studio and learn about Microsoft programming technologies. An Internet connection is required for the IDE to access most of this information.

To access more extensive information on Visual Studio, you can browse the **MSDN (Microsoft Developer Network)** Library at

```
https://msdn.microsoft.com/library/dd831853
```

The MSDN site contains articles, downloads and tutorials on technologies of interest to Visual Studio developers. You also can browse the web from the IDE by selecting **View > Other Windows > Web Browser**. To request a web page, type its URL into the location bar (Fig. 2.2) and press the *Enter* key—your computer, of course, must be connected to the Internet. The web page that you wish to view appears as another tab in the IDE—Figure 2.2 shows the browser tab after entering `http://msdn.microsoft.com/library`.

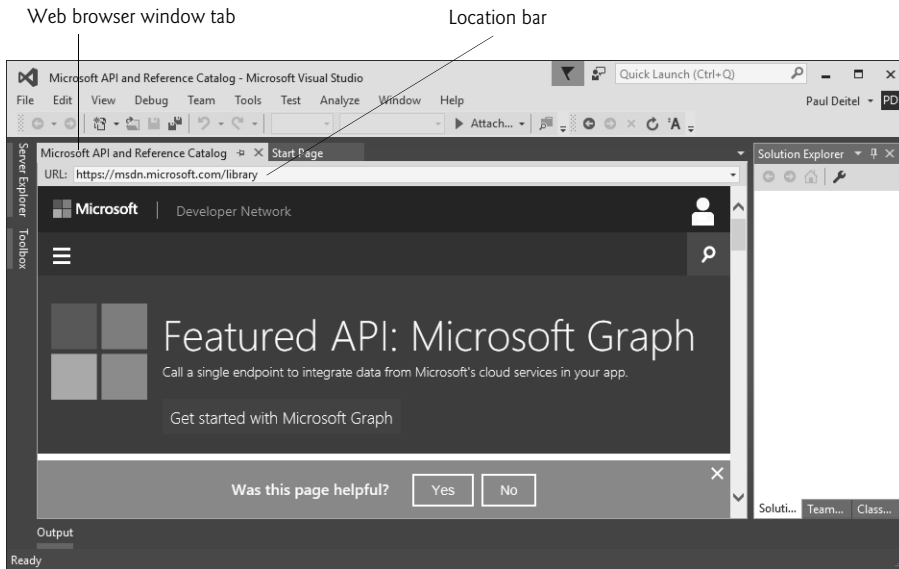


Fig. 2.2 | MSDN Library web page in Visual Studio.

2.2.4 Creating a New Project

To begin app development in Visual C#, you must create a new project or open an existing one. A **project** is a group of related files, such as the Visual C# code and any images that might make up an app. Visual Studio organizes apps into projects and **solutions**, which contain one or more projects. Multiple-project solutions are used to create large-scale apps. Most apps we create in this book consist of a solution containing a single project. You select **File > New > Project...** to create a new project or **File > Open > Project/Solution...** to

open an existing one. You also can click the corresponding links in the **Start Page**'s **Start** section.

2.2.5 New Project Dialog and Project Templates

For the discussions in the next several sections, we'll create a new project. Select **File > New > Project...** to display the **New Project dialog** (Fig. 2.3). **Dialogs** are windows that facilitate user–computer communication.

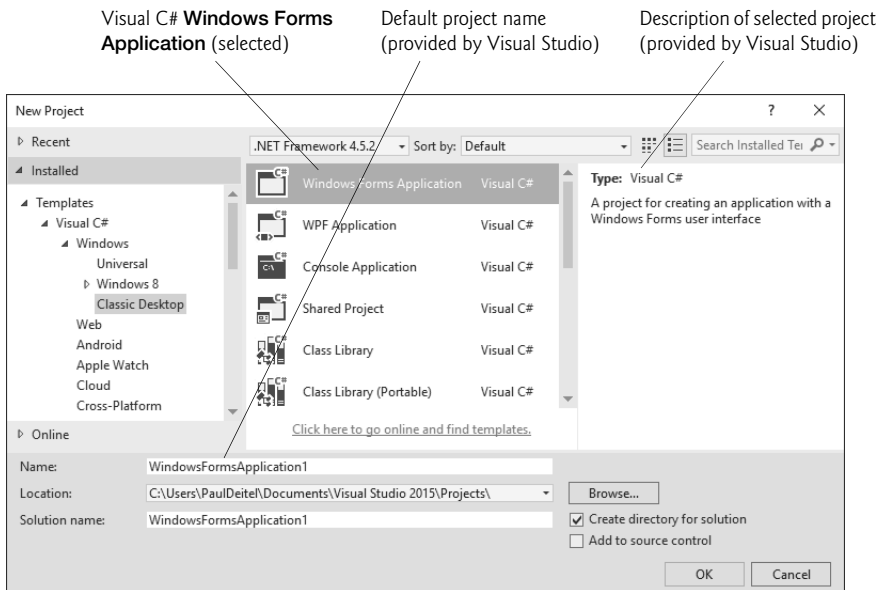


Fig. 2.3 | **New Project dialog.**

Visual Studio provides many **templates** (left column of Fig. 2.3)—the *project types* that users can create in Visual C# and other languages. The templates include Windows Forms apps, WPF apps and others—full versions of Visual Studio provide additional templates. In this chapter, you'll build a **Windows Forms Application**—an app that executes within a Windows operating system (such as Windows 7, 8 or 10) and typically has a **graphical user interface (GUI)**. Users interact with this *visual* part of the app. GUI apps include Microsoft software products like Microsoft Word, Internet Explorer and Visual Studio, software products created by other vendors, and customized software that you and other app developers create. You'll create many Windows apps in this book.

To create a **Windows Forms Application**, under **Templates** select **Visual C# > Windows > Classic Desktop**, then in the middle column select **Windows Forms Application**. By default, Visual Studio assigns the name **WindowsFormsApplication1** to a new **Windows Forms Application** project and solution (Fig. 2.3). Click **OK** to display the IDE in **Design view** (Fig. 2.4), which contains the features that enable you to create an app's GUI.

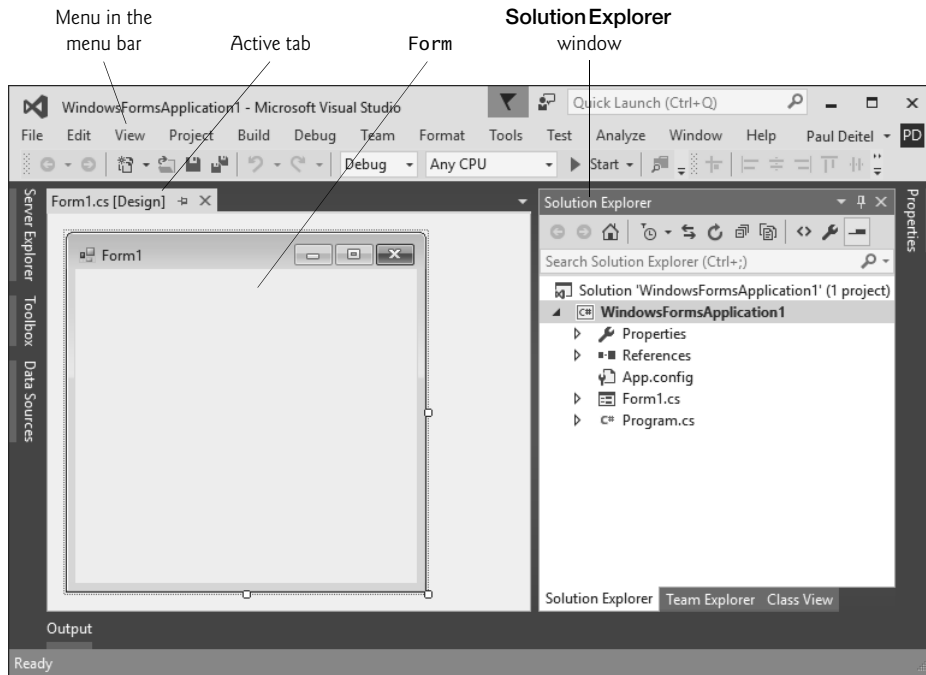


Fig. 2.4 | Design view of the IDE.

2.2.6 Forms and Controls

The rectangle in the **Design** area titled **Form1** (called a **Form**) represents the main window of the Windows Forms app that you're creating. Each Form is an object of class `Form` in the .NET Framework Class Library. Apps can have multiple Forms (windows)—however, the app you'll create in Section 2.6 and most other Windows Forms app you'll create later in this book will contain a single Form. You'll learn how to customize the Form by adding GUI **controls**—in Section 2.6, you'll add a `Label` and a `PictureBox`. A `Label` typically contains descriptive text (for example, "Welcome to Visual C#!"), and a `PictureBox` displays an image. Visual Studio has many preexisting controls and other components you can use to build and customize your apps. Many of these controls are discussed and used throughout the book. Other controls are available from third parties.

In this chapter, you'll work with preexisting controls from the .NET Framework Class Library. As you place controls on the Form, you'll be able to modify their properties (discussed in Section 2.4).

Collectively, the Form and controls make up the app's GUI. Users enter data into the app by typing at the keyboard, by clicking the mouse buttons and in a variety of other ways. Apps use the GUI to display instructions and other information for users to view. For example, the **New Project** dialog in Fig. 2.3 presents a GUI where the user clicks the mouse button to select a template type, then inputs a project name from the keyboard (the figure shows the default project name `WindowsFormsApplication1`).

Each open document's name is listed on a tab. To view a document when multiple documents are open, click its tab. The **active tab** (the tab of the currently displayed document) is highlighted (for example, **Form1.cs [Design]** in Fig. 2.4). The active tab's highlight color depends on the Visual Studio theme—the blue theme uses a yellow highlight and the light and dark themes use a blue highlight.

2.3 Menu Bar and Toolbar

Commands for managing the IDE and for developing, maintaining and executing apps are contained in menus, which are located on the **menu bar** of the IDE (Fig. 2.5). The set of menus displayed depends on what you're currently doing in the IDE.

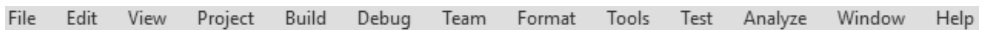


Fig. 2.5 | Visual Studio menu bar.

Menus contain groups of related commands called *menu items* that, when selected, cause the IDE to perform specific actions—for example, open a window, save a file, print a file and execute an app. For example, selecting **File > New > Project...** tells the IDE to display the **New Project** dialog. The menus depicted in Fig. 2.5 are summarized in Fig. 2.6.

Menu	Contains commands for
File	Opening, closing, adding and saving projects, as well as printing project data and exiting Visual Studio.
Edit	Editing apps, such as cut, copy, paste, undo, redo, delete, find and select.
View	Displaying IDE windows (for example, Solution Explorer , Toolbox , Properties window) and for adding toolbars to the IDE.
Project	Managing projects and their files.
Build	Turning your app into an executable program.
Debug	Compiling, debugging (that is, identifying and correcting problems in apps) and running apps.
Team	Connecting to a Team Foundation Server—used by development teams that typically have multiple people working on the same app.
Format	Arranging and modifying a Form's controls. The Format menu appears <i>only</i> when a GUI component is selected in Design view.
Tools	Accessing additional IDE tools and options for customizing the IDE.
Test	Performing various types of automated testing on your app.
Analyze	Locating and reporting violations of the .NET Framework Design Guidelines (https://msdn.microsoft.com/library/ms229042).

Fig. 2.6 | Summary of Visual Studio menus that are displayed when a Form is in **Design** view. (Part I of 2.)

Menu	Contains commands for
Window	Hiding, opening, closing and displaying IDE windows.
Help	Accessing the IDE's help features.

Fig. 2.6 | Summary of Visual Studio menus that are displayed when a Form is in Design view. (Part 2 of 2.)

You can access many common menu commands from the **toolbar** (Fig. 2.7), which contains **icons** that graphically represent commands. By default, the standard toolbar is displayed when you run Visual Studio for the first time—it contains icons for the most commonly used commands, such as opening a file, saving files and running apps (Fig. 2.7). The icons that appear on the standard toolbar may vary, depending on the version of Visual Studio you're using. Some commands are initially disabled (grayed out or unavailable to use). These commands are enabled by Visual Studio only when you can use them. For example, Visual Studio enables the command for saving a file once you begin editing a file.

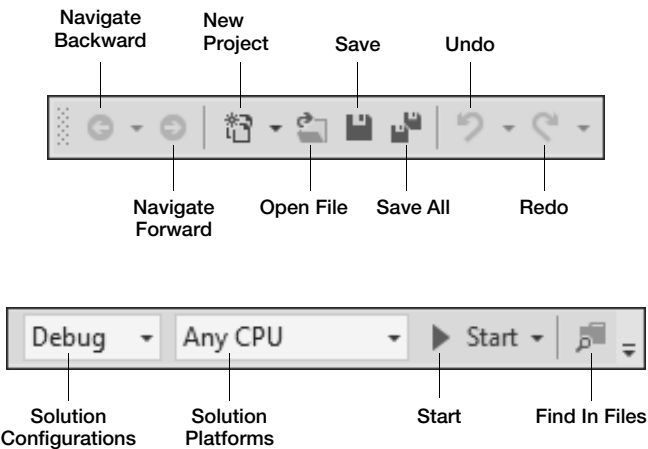


Fig. 2.7 | Standard Visual Studio toolbar.

You can customize which toolbars are displayed by selecting **View > Toolbars** then selecting a toolbar from the list in Fig. 2.8. Each toolbar you select is displayed with the other toolbars at the top of the Visual Studio window. You move a toolbar by dragging its handle



at the left side of the toolbar. To execute a command via the toolbar, click its icon.

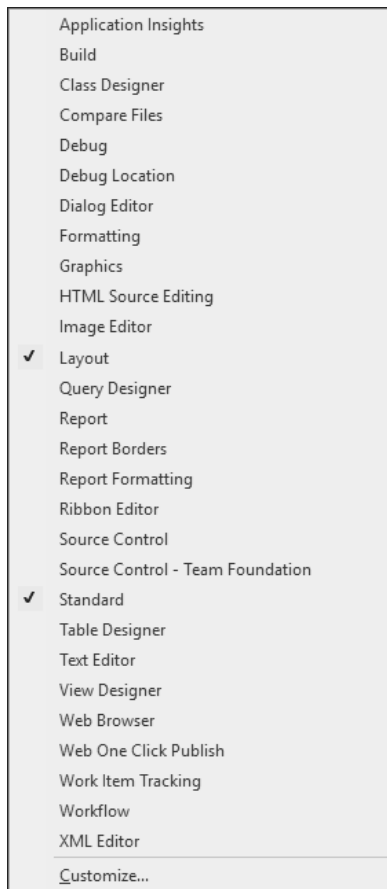


Fig. 2.8 | List of toolbars that can be added to the top of the IDE.

It can be difficult to remember what each toolbar icon represents. *Hovering* the mouse pointer over an icon highlights it and, after a brief pause, displays a description of the icon called a **tool tip** (Fig. 2.9)—these tips help you become familiar with the IDE’s features and serve as useful reminders for each toolbar icon’s functionality.

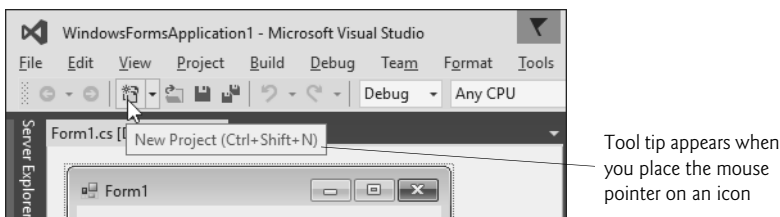


Fig. 2.9 | Tool tip for the **New Project** button.

2.4 Navigating the Visual Studio IDE

The IDE provides windows for accessing project files and customizing controls. This section introduces several windows that you'll use frequently when developing Visual C# apps. Each of the IDE's windows can be accessed by selecting its name in the **View** menu.

Auto-Hide

Visual Studio provides an **auto-hide** space-saving feature. When auto-hide is enabled for a window, a tab containing the window's name appears along the IDE window's left, right or bottom edge (Fig. 2.10). Clicking the name of an auto-hidden window displays that window (Fig. 2.11). Clicking the name again (or clicking outside) hides the window. To “pin down” a window (that is, to disable auto-hide and keep the window open), click the pin icon. When auto-hide is enabled, the pin icon is horizontal



as shown in Fig. 2.11. When a window is “pinned down,” the pin icon is vertical



as shown in Fig. 2.12.

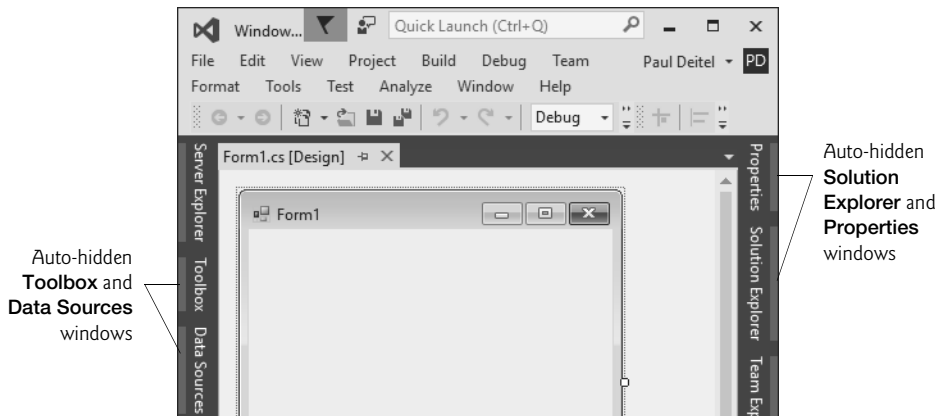


Fig. 2.10 | Auto-hide feature demonstration.

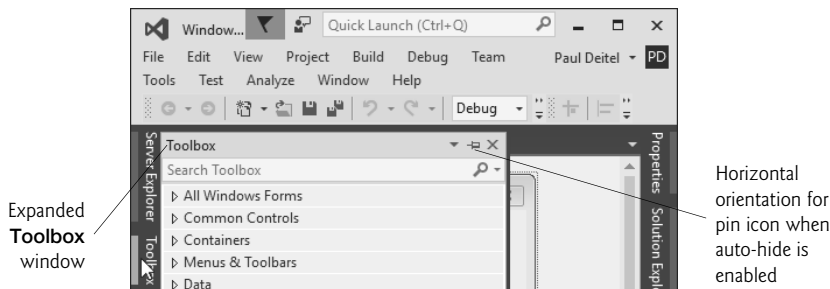


Fig. 2.11 | Displaying the hidden **Toolbox** window when auto-hide is enabled.

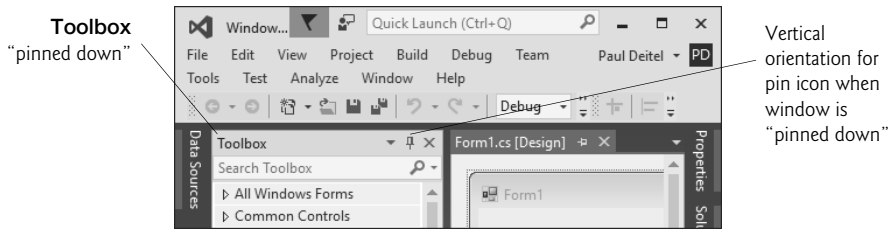


Fig. 2.12 | Disabling auto-hide—"pinning down" a window.

The next few sections present three Visual Studio's windows that you'll use frequently—the **Solution Explorer**, the **Properties** window and the **Toolbox**. These windows display project information and include tools that help you build your apps.

2.4.1 Solution Explorer

The **Solution Explorer** window (Fig. 2.13) provides access to all of a solution's files. If it's not shown in the IDE, select **View > Solution Explorer**. When you open a new or existing solution, the **Solution Explorer** displays the solution's contents.

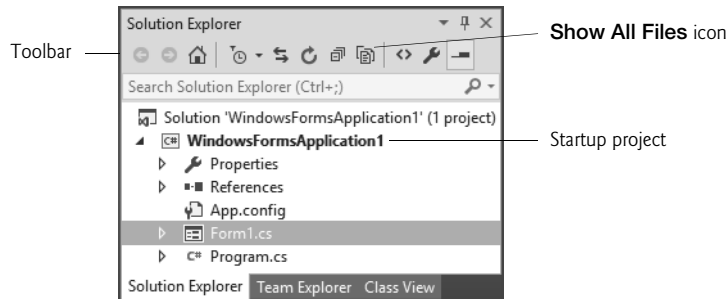


Fig. 2.13 | **Solution Explorer** window showing the **WindowsFormsApplication1** project.

The solution's **startup project** (shown in **bold** in the **Solution Explorer**) is the one that runs when you select **Debug > Start Debugging** (or press *F5*) or select **Debug > Start Without Debugging** (or press *Ctrl + F5* key). For a single-project solution like the examples in this book, the startup project is the only project (in this case, **WindowsFormsApplication1**). When you create an app for the first time, the **Solution Explorer** window appears as shown in Fig. 2.13. The Visual C# file that corresponds to the Form shown in Fig. 2.4 is named **Form1.cs** (selected in Fig. 2.13). Visual C# files use the **.cs** file-name extension, which is short for "C#."

By default, the IDE displays only files that you may need to edit—other files that the IDE generates are hidden. The **Solution Explorer** window includes a toolbar that contains several icons. Clicking the **Show All Files icon** (Fig. 2.13) displays all the solution's files, including those generated by the IDE. Clicking the arrow to the left of a node *expands* or *collapses* that node. Click the arrow to the left of **References** to display items grouped

under that heading (Fig. 2.14). Click the arrow again to collapse the tree. Other Visual Studio windows also use this convention.

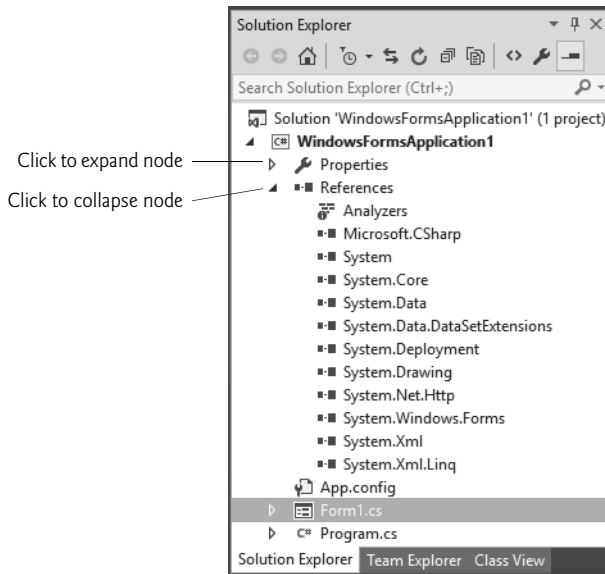


Fig. 2.14 | Solution Explorer with the **References** node expanded.

2.4.2 Toolbox

To display the **Toolbox** window, select **View > Toolbox**. The **Toolbox** contains the controls used to customize Forms (Fig. 2.15). With visual app development, you can “drag and drop” controls onto the Form and the IDE will write the code that creates the controls for you. This is faster and simpler than writing this code yourself. Just as you do not need to know how to build an engine to drive a car, you do not need to know how to build controls to use them. *Reusing* preexisting controls saves time and money when you develop apps. You’ll use the **Toolbox** when you create your first app later in the chapter.

The **Toolbox** groups the prebuilt controls into categories—**All Windows Forms**, **Common Controls**, **Containers**, **Menus & Toolbars**, **Data**, **Components**, **Printing**, **Dialogs**, **Reporting**, **WPF Interoperability** and **General** are listed in Fig. 2.15. Again, note the use of arrows for expanding or collapsing a group of controls. We discuss many of the **Toolbox**’s controls and their functionality throughout the book.

2.4.3 Properties Window

If the **Properties** window is not displayed below the **Solution Explorer**, select **View > Properties Window** to display it—if the window is in auto-hide mode, pin down the window by clicking its horizontal pin icon



The **Properties window** contains the properties for the currently selected Form, control or file in the IDE. **Properties** specify information about the Form or control, such as its size,

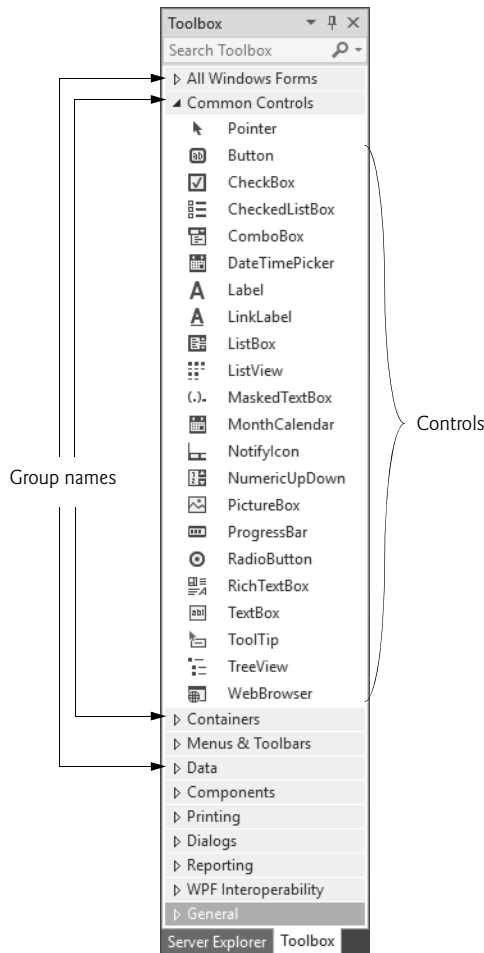


Fig. 2.15 | Toolbox window displaying controls for the **Common Controls** group.

color and position. Each Form or control has its own set of properties. When you select a property, its description is displayed at the bottom of the **Properties** window.

Figure 2.16 shows Form1's **Properties** window—you can view by clicking anywhere in the **Form1.cs [Design]** window. The left column lists the Form's properties—the right column displays the current value of each property. You can sort the properties either

- *alphabetically* (by clicking the **Alphabetical icon**) or
- *categorically* (by clicking the **Categorized icon**).

Depending on the **Properties** window's size, some properties may be hidden from your view. You can scroll through the list of properties by **dragging** the **scrollbox** up or down inside the **scrollbar**, or by clicking the arrows at the top and bottom of the scrollbar. We show how to set individual properties later in this chapter.

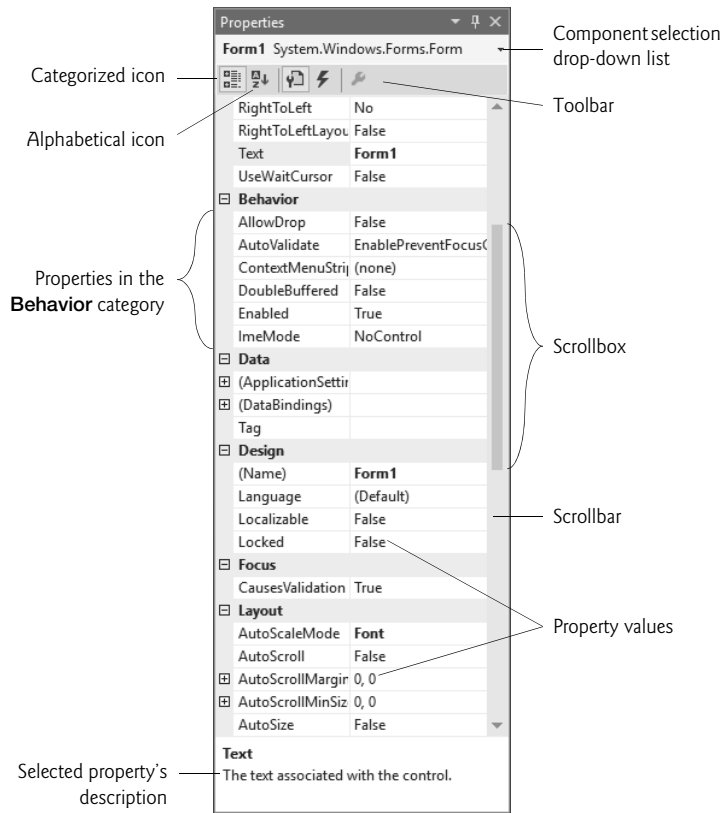


Fig. 2.16 | Properties window.

The **Properties** window is crucial to visual app development—it allows you to quickly modify a control’s properties and, rather than writing code yourself, lets the IDE write code for you “behind the scenes.” You can see which properties are available for modification and, in many cases, can learn the range of acceptable values for a given property. The **Properties** window displays a brief description of the selected property, helping you understand its purpose.

2.5 Help Menu and Context-Sensitive Help

Microsoft provides extensive help documentation via the **Help menu**, which is an excellent way to get information quickly about Visual Studio, Visual C# and more. Visual Studio provides **context-sensitive help** pertaining to the “current content” (that is, the items around the location of the mouse cursor). To use context-sensitive help, click an item, then press the *F1* key. The help documentation is displayed in a web browser window. To return to the IDE, either close the browser window or select the IDE’s icon in your Windows task bar.

2.6 Visual Programming: Creating a Simple App that Displays Text and an Image

Next, we create an app that displays the text "Welcome to C# Programming!" and an image of the Deitel & Associates bug mascot. The app consists of a Form that uses a `Label` and a `PictureBox`. Figure 2.17 shows the final app executing. The app and the bug image are available with this chapter's examples—see the Before You Begin section following the Preface for download instructions. We assume you placed the examples in your user account's Documents folder in a subfolder named `examples`.

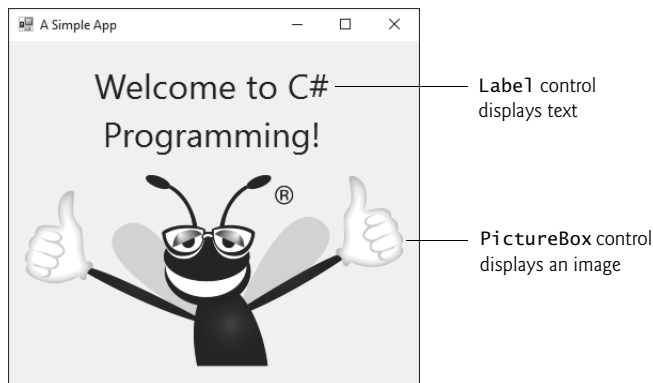


Fig. 2.17 | Simple app executing.

In this example, you won't write any C# code. Instead, you'll use visual app-development techniques. Visual Studio processes your actions (such as mouse clicking, dragging and dropping) to generate app code. Chapter 3 begins our discussion of writing app code. Throughout the book, you'll produce increasingly substantial and powerful apps that will include code written by you and code generated by Visual Studio. The generated code can be difficult for novices to understand—but you'll rarely need to look at it.

Visual app development is useful for building GUI-intensive apps that require a significant amount of user interaction. To create, save, run and terminate this first app, perform the following steps.

Step 1: Closing the Open Project

If the project you were working with earlier in this chapter is still open, close it by selecting **File > Close Solution**.

Step 2: Creating the New Project

To create a new Windows Forms app:

1. Select **File > New > Project...** to display the **New Project** dialog (Fig. 2.18).
2. Select **Windows Forms Application**. Name the project **ASimpleApp**, specify the **Location** where you want to save it and click **OK**. We stored the app in the IDE's default location—in your user account's Documents folder under the Visual Studio 2015\Projects.

As you saw earlier in this chapter, when you first create a new Windows Forms app, the IDE opens in **Design** view (that is, the app is being designed and is not executing). The text **Form1.cs [Design]** in the tab containing the Form means that we're designing the Form *visually* rather than *programmatically*. An asterisk (*) at the end of the text in a tab indicates that you've changed the file and the changes have not yet been saved.

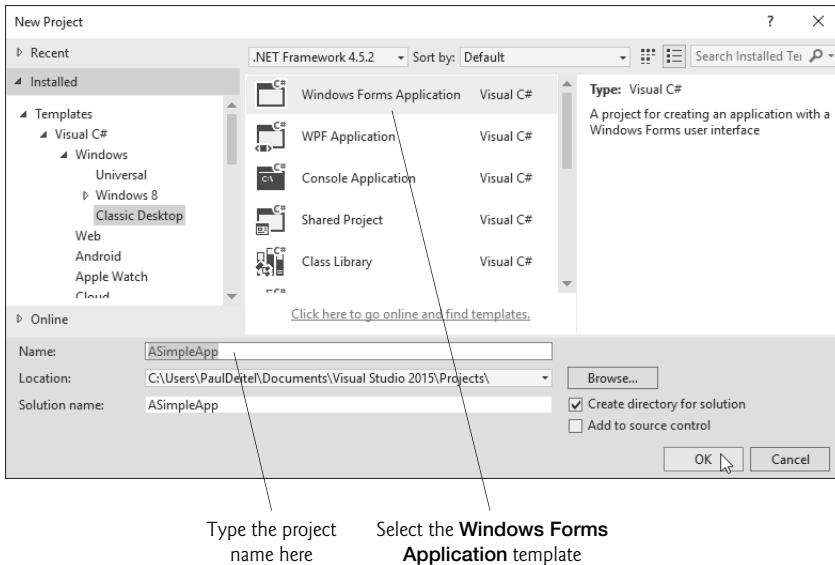


Fig. 2.18 | New Project dialog.

Step 3: Setting the Text in the Form's Title Bar

The text in the Form's title bar is determined by the Form's **Text** property (Fig. 2.19). If the **Properties** window is not open, select **View > Properties Window** and pin down the window so it doesn't auto hide. Click anywhere in the Form to display the Form's properties in the **Properties** window. In the textbox to the right of the Text property, type "A Simple App", as in Fig. 2.19. Press the **Enter** key—the Form's title bar is updated immediately (Fig. 2.20).

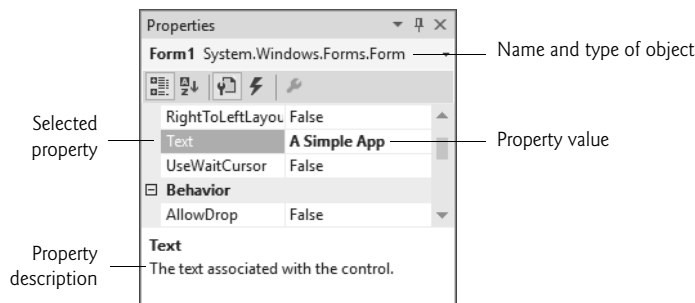


Fig. 2.19 | Setting the Form's Text property in the **Properties** window.

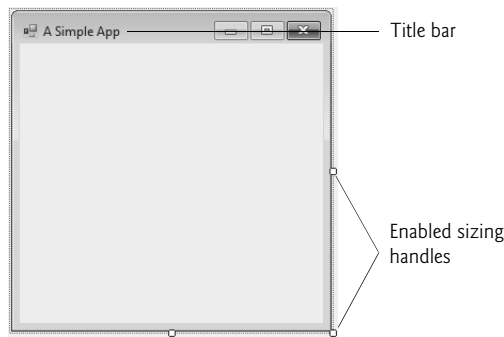


Fig. 2.20 | Form with updated title-bar text and enabled sizing handles.

Step 4: Resizing the Form

The Form's size is specified in pixels (that is, dots on the screen). By default, a Form is 300 pixels wide and 300 pixels tall. You can resize the Form by dragging one of its **sizing handles** (the small white squares that appear around the Form, as shown in Fig. 2.20). Using the mouse, select the bottom-right sizing handle and drag it down and to the right to make the Form larger. As you drag the mouse (Fig. 2.21), the IDE's status bar (at the bottom of the IDE) shows the current width and height in pixels. We set the Form to 400 pixels wide by 360 pixels tall. You also can do this via the Form's **Size** property in the **Properties** window.

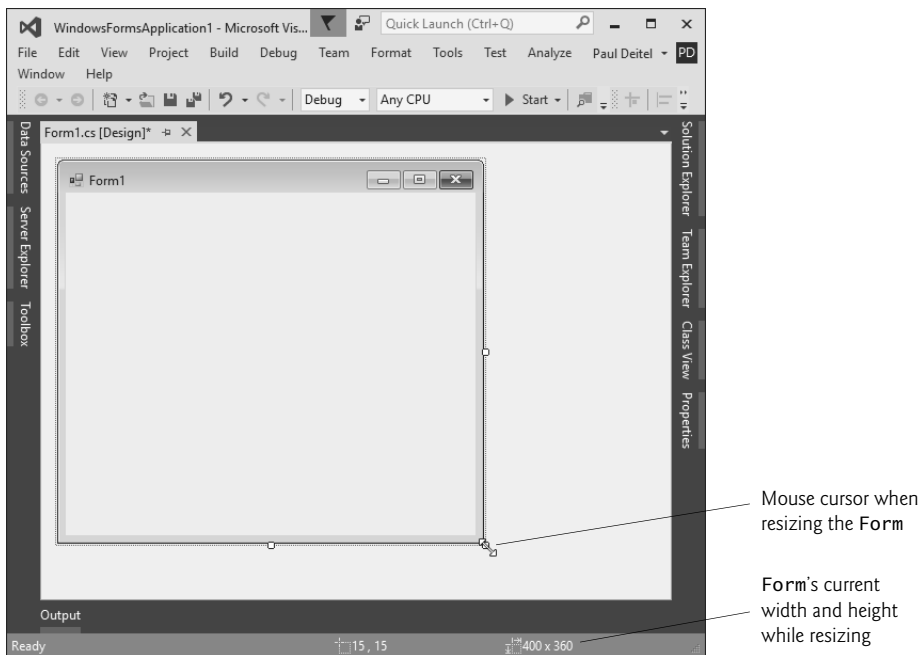


Fig. 2.21 | Resizing the Form.

Step 5: Changing the Form's Background Color

The **BackColor** property specifies a Form's or control's background color. Clicking **BackColor** in the **Properties** window causes a down-arrow button to appear next to the value of the property (Fig. 2.22). Clicking the down-arrow button displays other options, which vary depending on the property. In this case, the arrow displays tabs for **Custom**, **Web** and **System** (the default). Click the **Custom** tab to display the **palette** (a grid of colors). Select the box that represents light blue. Once you select the color, the palette closes and the Form's background color changes to light blue (Fig. 2.23).

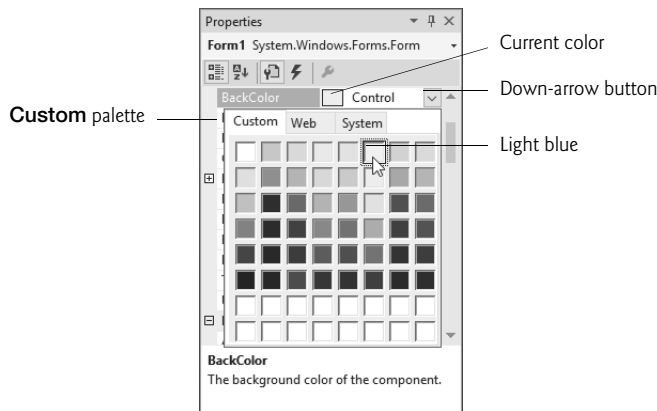


Fig. 2.22 | Changing the Form's BackColor property.

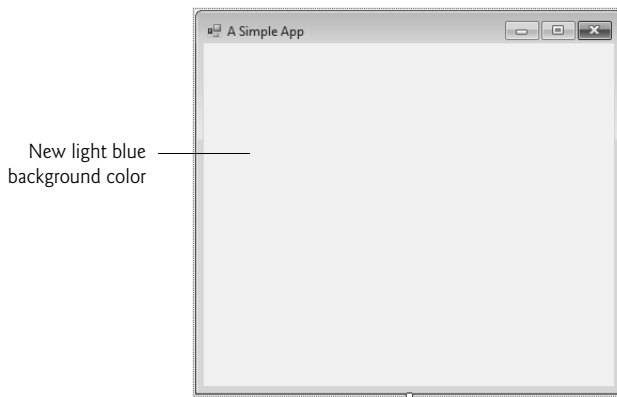


Fig. 2.23 | Form with new BackColor property applied.

Step 6: Adding a Label Control to the Form

For the app we're creating in this chapter, the typical controls we use are located in the **Toolbox's Common Controls** group, and also can be found in the **All Windows Forms** group. If the **Toolbox** is *not* already open, select **View > Toolbox** to display the set of controls you'll use for creating your apps. If either group name is collapsed, expand it by clicking the ar-

row to the left of the group name (the **All Windows Forms** and **Common Controls** groups are shown in Fig. 2.15). Next, double click the `Label` control in the **Toolbox** to add a `Label` in the Form's upper-left corner (Fig. 2.24)—each `Label` you add to the Form is an object of class `Label` from the .NET Framework Class Library. [Note: If the Form is behind the **Toolbox**, you may need to hide or pin down the **Toolbox** to see the `Label`.] Although double clicking any **Toolbox** control places the control on the Form, you also can “drag” controls from the **Toolbox** to the Form—you may prefer dragging the control because you can position it wherever you want. The `Label` displays the text `label1` by default. By default, the `Label`'s `BackColor` is the same as the Form's.

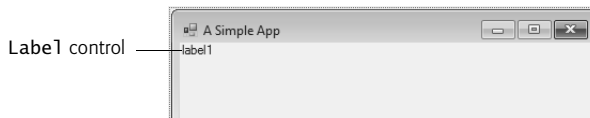


Fig. 2.24 | Adding a `Label` to the Form.

Step 7: Customizing the `Label`'s Appearance

Click the `Label`'s text in the Form to select it and display its properties in the **Properties** window. The `Label`'s `Text` property determines the text that the `Label` displays. The Form and `Label` each have their own `Text` property—Forms and controls can have the *same* property names (such as `Text`, `BackColor` etc.) without conflict. Each common properties purpose can vary by control. Perform the following steps:

1. Set the `Label`'s `Text` property to `Welcome to C# Programming!`. The `Label` resizes to fit all the typed text on one line.
2. By default, the `AutoSize` property of the `Label` is set to `True` so the `Label` can update its own size to fit all of its text. Set the `AutoSize` property to `False` so that you can change the `Label`'s size, then resize the `Label` (using the sizing handles) so that the text fits.
3. Move the `Label` to the top center of the Form by dragging it or by using the keyboard's left and right arrow keys to adjust its position (Fig. 2.25). Alternatively, when the `Label` is selected, you can center it horizontally by selecting **Format > Center In Form > Horizontally**.

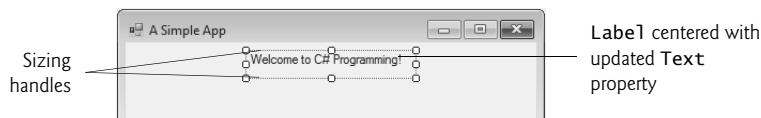


Fig. 2.25 | GUI after the Form and `Label` have been customized.

Step 8: Setting the `Label`'s Font Size

To change the font type and appearance of the `Label`'s text:

1. Select the value of the `Font` property, which causes an **ellipsis button** to appear next to the value (Fig. 2.26)—you can click this button to display a dialog of op-

tions for the property. Click the ellipsis button to display the **Font dialog** (Fig. 2.27).

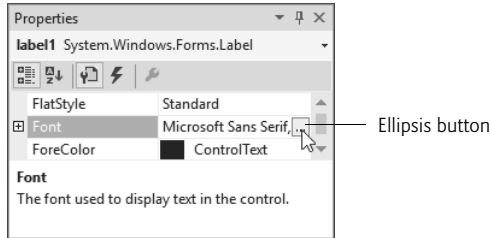


Fig. 2.26 | Properties window displaying the Label's Font property.

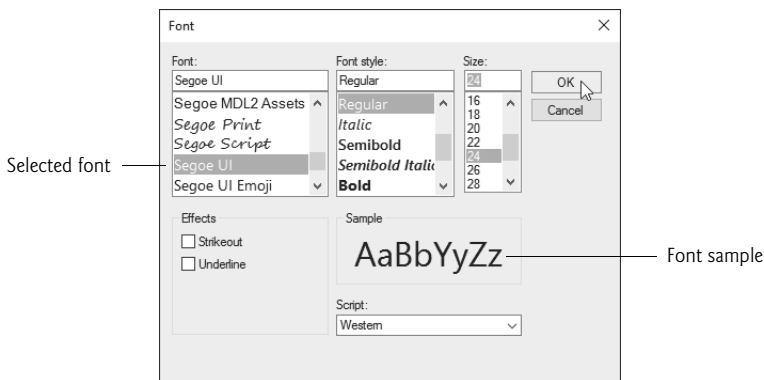


Fig. 2.27 | Font dialog for selecting fonts, styles and sizes.

2. You can select the font name (the font options may be different, depending on your system), font style (**Regular**, **Italic**, **Bold**, etc.) and font size (16, 18, 20, etc.) in this dialog. The **Sample** text shows the selected font settings. Under **Font**, select **Segoe UI**, Microsoft's recommended font for user interfaces. Under **Size**, select 24 points and click **OK**.
3. If the Label's text does not fit on a single line, it *wraps* to the next line. Resize the Label so that the words "Welcome to" appear on the Label's first line and the words "C# Programming!" appear on the second line.
4. Re-center the Label horizontally.

Step 9: Aligning the Label's Text

Select the Label's **TextAlign** property, which determines how the text is aligned within the Label. A three-by-three grid of buttons representing alignment choices is displayed (Fig. 2.28). The position of each button corresponds to where the text appears in the Label. For this app, set the **TextAlign** property to **MiddleCenter** in the three-by-three grid—this selection centers the text horizontally and vertically within the Label. The other **TextAlign** values, such as **TopLeft**, **TopRight**, and **BottomCenter**, can be used to position

the text anywhere within a `Label`. Certain alignment values may require that you resize the `Label` to fit the text better.

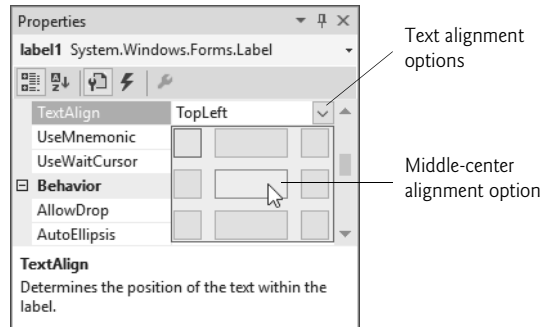


Fig. 2.28 | Centering the `Label`'s text.

Step 10: Adding a `PictureBox` to the Form

The `PictureBox` control displays images. Locate the `PictureBox` in the **Toolbox** (Fig. 2.15) and double click it to add it to the Form—each `PictureBox` you add to the Form is an object of class `PictureBox` from the .NET Framework Class Library. When the `PictureBox` appears, move it underneath the `Label`, either by dragging it or by using the arrow keys (Fig. 2.29).

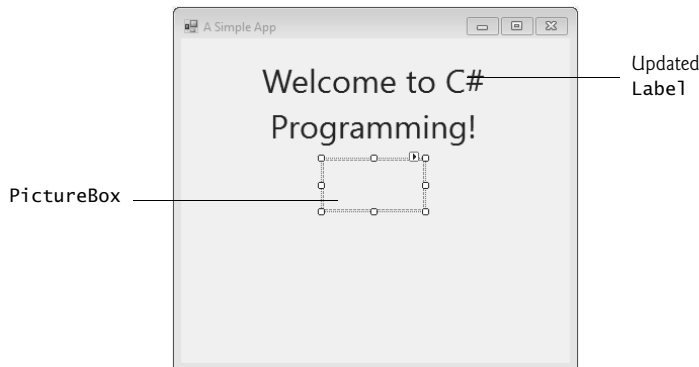


Fig. 2.29 | Inserting and aligning a `PictureBox`.

Step 11: Inserting an Image

Click the `PictureBox` to display its properties in the **Properties** window (Fig. 2.30), then:

1. Locate and select the **Image property**, which displays a preview of the selected image or (none) if no image is selected.
2. Click the ellipsis button to display the **Select Resource dialog** (Fig. 2.31), which is used to import files, such as images, for use in an app.

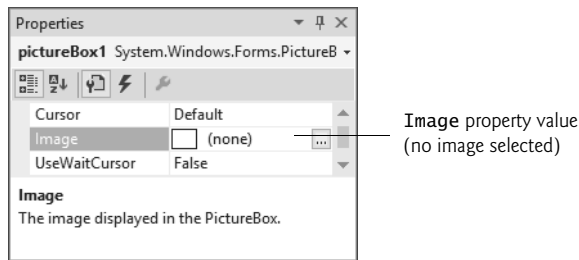


Fig. 2.30 | Image property of the PictureBox.

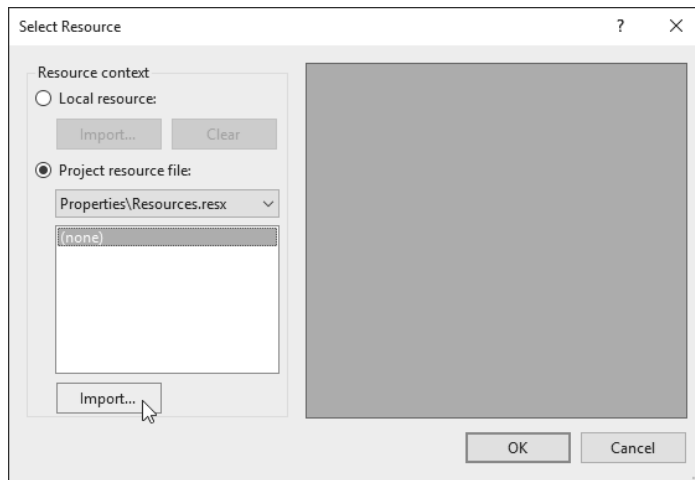


Fig. 2.31 | Select Resource dialog to select an image for the PictureBox.

3. Click the **Import...** button to browse for an image, select the image file and click **OK** to add it to your project. We used `bug.png` from this chapter's examples folder. Supported image formats include PNG (Portable Network Graphics), GIF (Graphic Interchange Format), JPEG (Joint Photographic Experts Group) and BMP (Windows bitmap). Depending on the image's size, it's possible that only a portion of the image will be previewed in the **Select Resource** dialog—you can resize the dialog to see more of the image (Fig. 2.32). Click **OK** to use the image.
4. To scale the image to fit in the PictureBox, change the **SizeMode** property to **StretchImage** (Fig. 2.33). Resize the PictureBox, making it larger (Fig. 2.34), then re-center the PictureBox horizontally.

Step 12: Saving the Project

Select **File > Save All** to save the entire solution. The solution file (which has the filename extension `.sln`) contains the name and location of its project, and the project file (which has the filename extension `.csproj`) contains the names and locations of all the files in the project. If you want to reopen your project at a later time, simply open its `.sln` file.

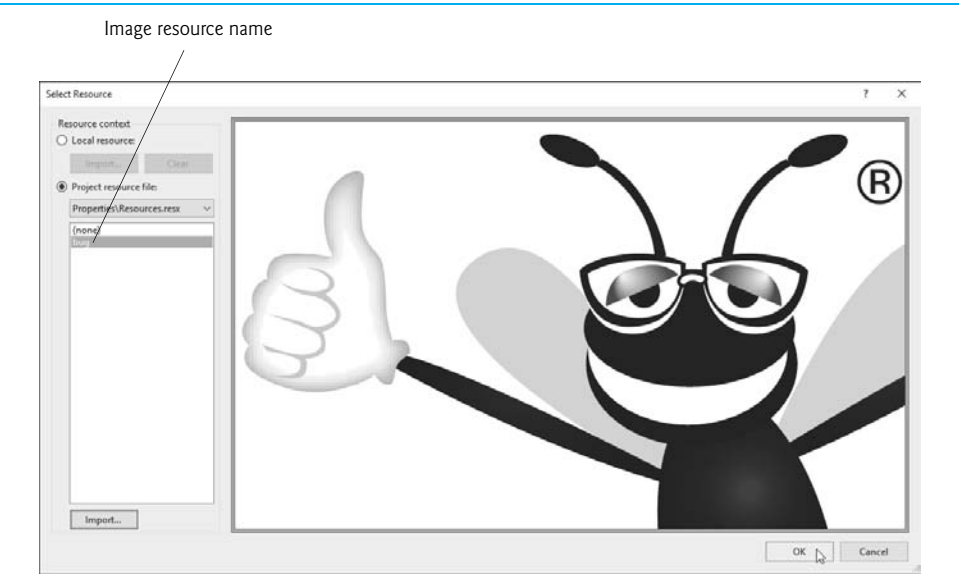


Fig. 2.32 | Select Resource dialog displaying a preview of selected image.

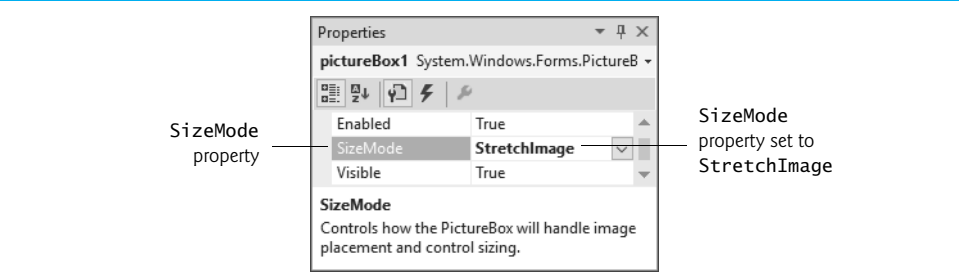


Fig. 2.33 | Scaling an image to the size of the PictureBox.



Fig. 2.34 | PictureBox displaying an image.

Step 13: Running the Project

Recall that up to this point we have been working in the IDE design mode (that is, the app being created is not executing). In **run mode**, the app is executing, and you can interact with only a few IDE features—features that are not available are disabled (grayed out). Select **Debug > Start Debugging** to execute the app (or press the *F5* key). The IDE enters run mode and displays “(Running)” next to the app’s name in the IDE’s title bar. Figure 2.35 shows the running app, which appears in its own window outside the IDE.

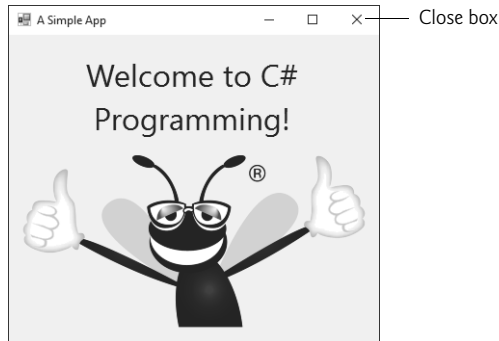


Fig. 2.35 | IDE in run mode, with the running app in the foreground.

Step 14: Terminating the App

You can terminate the app by clicking its close box



in the top-right corner of the running app’s window. This action stops the app’s execution and returns the IDE to design mode. You also can select **Debug > Stop Debugging** to terminate the app.

2.7 Wrap-Up

In this chapter, we introduced key features of the Visual Studio IDE. You visually designed a working Visual C# app without writing any code. Visual C# app development is a mixture of the two styles—visual app development allows you to develop GUIs easily and avoid tedious GUI programming and “conventional” programming (which we introduce in Chapter 3) allows you to specify the behavior of your apps.

You created a Visual C# Windows Forms app with one Form. You worked with the IDE’s **Solution Explorer**, **Toolbox** and **Properties** windows, which are essential to developing Visual C# apps. We also demonstrated context-sensitive help, which displays help topics related to selected controls or text.

You used visual app development to design an app’s GUI by adding a **Label** and a **PictureBox** control onto a Form. You used the **Properties** window to set a Form’s **Text** and **BackColor** properties. You learned that **Label** controls display text and that **PictureBoxes** display images. You displayed text in a **Label** and added an image to a **PictureBox**. You also worked with the **Label**’s **AutoSize**, **TextAlign** and **Font** properties and the **PictureBox**’s **Image** and **SizeMode** properties.

In the next chapter, we discuss “nonvisual,” or “conventional,” programming—you’ll create your first apps with C# code that you write, instead of having Visual Studio write the code. You’ll learn memory concepts and write code that displays information on the screen, receives inputs from the user at the keyboard, performs arithmetic operations and makes decisions.

2.8 Web Resources

Please take a moment to visit each of these sites.

<https://www.visualstudio.com/>

The home page for Microsoft Visual Studio. The site includes news, documentation, downloads and other resources.

<https://social.msdn.microsoft.com/Forums/vstudio/en-US/home?forum=csharpgeneral>

This site provides access to the Microsoft Visual C# forums, which you can use to get your Visual C# language and IDE questions answered.

<https://msdn.microsoft.com/magazine/default.aspx>

This is the Microsoft Developer Network Magazine site. This site provides articles and code on many Visual C# and .NET app development topics. There is also an archive of past issues.

<http://stackoverflow.com/>

In addition to the Microsoft forums, StackOverflow is an excellent site for getting your programming questions answered for most programming languages and technologies.

Summary

Section 2.1 Introduction

- Visual Studio is Microsoft’s Integrated Development Environment (IDE) for creating, running and debugging apps written in a variety of .NET programming languages.
- Creating simple apps by dragging and dropping predefined building blocks into place is called visual app development.

Section 2.2 Overview of the Visual Studio Community 2015 IDE

- The **Start Page** contains links to Visual Studio IDE resources and web-based resources.
- A project is a group of related files that compose an app.
- Visual Studio organizes apps into projects and solutions. A solution may contain one or more projects.
- Dialogs are windows that facilitate user–computer communication.
- Visual Studio provides templates for the project types you can create.
- A **Form** represents the main window of the Windows Forms app that you’re creating.
- Collectively, the **Form** and controls constitute the app’s graphical user interface (GUI), which is the visual part of the app with which the user interacts.

Section 2.3 Menu Bar and Toolbar

- Commands for managing the IDE and for developing, maintaining and executing apps are contained in the menus, which are located on the menu bar.

- Menus contain groups of commands (menu items) that, when selected, cause the IDE to perform actions (for example, open a window, save a file, print a file and execute an app).
- Tool tips help you become familiar with the IDE's features.

Section 2.4 Navigating the Visual Studio IDE

- The **Solution Explorer** window lists all the files in the solution.
- The **Toolbox** contains controls for customizing Forms.
- By using visual app development, you can place predefined controls onto the Form instead of writing the code yourself.
- Clicking an auto-hidden window's name opens that window. Clicking the name again hides it. To "pin down" a window (that is, to disable auto-hide), click its pin icon.
- The **Properties** window displays the properties for a Form, control or file (in **Design** view). Properties are information about a Form or control, such as size, color and position. The **Properties** window allows you to modify Forms and controls visually, without writing code.
- Each control has its own set of properties. The left column of the **Properties** window shows the property names and the right column displays the property values. This window's toolbar contains options for organizing properties alphabetically when the **Alphabetical** icon is selected or categorically (for example, **Appearance**, **Behavior**, **Design**) when the **Categorized** icon is selected.

Section 2.5 Help Menu and Context-Sensitive Help

- Extensive help documentation is available via the **Help** menu.
- Context-sensitive help brings up a list of relevant help articles. To use context-sensitive help, select an item and press the *FI* key.

Section 2.6 Visual Programming: Creating a Simple App that Displays Text and an Image

- Visual C# app development usually involves a combination of writing a portion of the app code and having Visual Studio generate the remaining code.
- The text that appears at the top of the Form (the title bar) is specified in the Form's Text property.
- To resize the Form, click and drag one of the Form's enabled sizing handles (the small squares around the Form). Enabled sizing handles appear as white boxes.
- The **BackColor** property specifies the background color of a Form. The Form's background color is the default background color for any controls added to the Form.
- Double clicking any **Toolbox** control icon places a control of that type on the Form. Alternatively, you can drag and drop controls from the **Toolbox** to the Form.
- The **Label**'s Text property determines the text (if any) that the **Label** displays. The Form and **Label** each have their own Text property.
- A property's ellipsis button, when clicked, displays a dialog containing additional options.
- In the **Font** dialog, you can select the font for text in the user interface.
- The **TextAlign** property determines how the text is aligned within a **Label**'s boundaries.
- The **PictureBox** control displays images. The **Image** property specifies the image to be displayed.
- An app that is in design mode is not executing.
- In run mode, the app is executing—you can interact with only a few IDE features.
- When designing an app visually, the name of the Visual C# file appears in the project tab, followed by **[Design]**.
- Terminate an app's execution by clicking its close box.

Terminology



active tab
Alphabetical icon
 auto-hide
 AutoSize property of Label
 BackColor property of Form
Categorized icon
 component selection drop-down list
 context-sensitive help
 control
Custom tab
Design view
 dialog
 dragging
 ellipsis button
Font dialog
 Font property of Label
 Form
 graphical user interface (GUI)
Help menu
 icon
 Image property of PictureBox
 Label
New Project dialog
 palette
 PictureBox
 project
Properties window
 property of a Form or control
 run mode
 scrollbar
 scrollbox
Select Resource dialog
Show All Files icon
 SizeMode property of PictureBox
 sizing handle
 solution
Solution Explorer in Visual Studio
Start Page
 startup project
 StretchImage value
 templates for projects
 Text property
 TextAlign property of Label
 tool tip
 toolbar
 visual app development
 Visual Studio
 Windows Forms app

Self-Review Exercises

- 2.1** Fill in the blanks in each of the following statements:
- The technique of _____ allows you to create GUIs without writing any code.
 - A(n) _____ is a group of one or more projects that collectively form a Visual C# app.
 - The _____ feature hides a window in the IDE.
 - A(n) _____ appears when the mouse pointer hovers over an icon.
 - The _____ window allows you to browse solution files.
 - The properties in the **Properties** window can be sorted _____ or _____.
 - A Form's _____ property specifies the text displayed in the Form's title bar.
 - The _____ contains the controls that you can add to a Form.
 - _____ displays relevant help articles, based on the current context.
 - The _____ property specifies how text is aligned within a Label's boundaries.
- 2.2** State whether each of the following is *true* or *false*. If *false*, explain why.
- ✕ toggles auto-hide for a window.
 - The toolbar icons represent various menu commands.
 - The toolbar contains icons that represent controls you can drag onto a Form.
 - Both Forms and Labels have a title bar.
 - Control properties can be modified only by writing code.
 - PictureBoxes typically display images.
 - Visual C# files use the file extension .csharp.
 - A Form's background color is set using the BackColor property.

Answers to Self-Review Exercises

2.1 a) visual app development. b) solution. c) auto-hide. d) tool tip. e) **Solution Explorer**. f) alphabetically, categorically. g) Text. h) **Toolbox**. i) context-sensitive help. j) `TextAlign`.

2.2 a) False. The pin icon () toggles auto-hide.  closes a window. b) True. c) False. The **Toolbox** contains icons that represent such controls. d) False. Forms have a title bar but Labels do not (although they do have `Label` text). e) False. Control properties can be modified using the **Properties** window. f) True. g) False. Visual C# files use the file extension `.cs`. h) True.

Exercises

- 2.3** Fill in the blanks in each of the following statements:
- When an ellipsis button is clicked, a(n) _____ is displayed.
 - Using _____ help immediately displays a relevant help article.
 - When you select a property, its description is displayed at the bottom of the _____ window.
 - The _____ property specifies which image a `PictureBox` displays.
 - The _____ menu contains commands for arranging and displaying windows.
- 2.4** State whether each of the following is *true* or *false*. If *false*, explain why.
- You can add a control to a `Form` by double clicking its control icon in the **Toolbox**.
 - The `Form`, `Label` and `PictureBox` have identical properties.
 - If your machine is connected to the Internet, you can browse websites from the Visual Studio IDE.
 - Visual C# app developers usually create complex apps without writing any code.
 - Clicking the arrow to the left of a node *expands* or *collapses* that node.
- 2.5** Some features that appear throughout Visual Studio perform similar actions in different contexts. Explain and give examples of how the ellipsis buttons, down-arrow buttons and tool tips act in this manner. Why do you think the Visual Studio IDE was designed this way?
- 2.6** Briefly describe each of the following terms:
- Project
 - IDE
 - Debug
 - Properties window
 - Dialogs
 - Visual Studio

Note Regarding Exercises 2.7–2.11

In the following exercises, you're asked to create GUIs using controls that we have not yet discussed in this book. These exercises give you practice with visual app development only—the apps do not perform any actions. You place controls from the **Toolbox** on a `Form` to familiarize yourself with what each control looks like. If you follow the step-by-step instructions, you should be able to recreate the sample GUIs we show.

- 2.7** (*Stop Watch GUI*) Create the GUI for the Stop Watch as shown in Fig. 2.36.
- Manipulating the Form's properties.* Change the `Text` property of the `Form` to `Stop_Watch_Form`.
 - Adding a `MenuStrip` control to the Form.* Add a `MenuStrip` to the `Form`. After inserting the `MenuStrip`, add items by clicking the **Type Here** section, typing a menu name (for example, **Start**, **Stop**, **Lap**, **Reset** and **Exit**) and then pressing *Enter*.

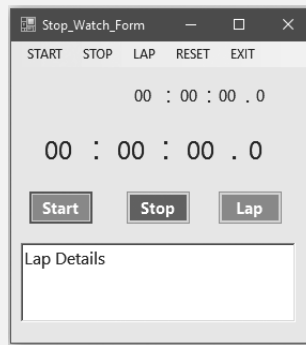


Fig. 2.36 | Notepad GUI.

- c) *Adding Labels to the Form.* Add 7 Labels each for the display of main stop watch timer and for the Lap timer. Set the Font property as **Segoe UI, 18pt, style=Regular** for main timer and **Segoe UI, 11pt, style=Regular** for lap timer. Set the Text properties accordingly.
- d) *Adding Buttons to the Form.* Add Three Buttons with Text property **Start**, **Stop** and **Lap**. Change their BackColor and ForeColor properties as shown.
- e) *Adding a RichTextBox to the Form.* Drag this control onto the Form. Use the sizing handles to resize and position the RichTextBox as shown in Fig. 2.36. Change the Text property to Lap Details.

2.8 (*Login Form GUI*) Create the GUI for the login screen as shown in Fig. 2.37.



Fig. 2.37 | Calendar and appointments GUI.

- a) *Manipulating the Form's properties.* Change the Text property of the Form to User Login Form. Set the BackColor property as shown in figure 2.37.
- b) *Adding Labels to the Form.* Add three Labels to the Form. Set the Label's Text, Font and BackColor properties to match Fig. 2.37. Use Bookman Old Style, 14.25pt font size for User ID and Password.
- c) *Adding TextBoxes to the Form.* Add two TextBox controls to the Form, for entering userID and password. Set the suitable BackColor and '*' as the PasswordChar property.

- d) *Adding Buttons to the Form.* Add Three Buttons with Text property New User, Login and Forgot password. Change their BackColor and ForeColor properties as shown.

2.9 (*Calculator GUI*) Create the GUI for the calculator as shown in Fig. 2.38.

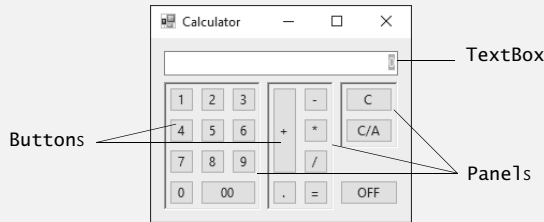


Fig. 2.38 | Calculator GUI.

- Manipulating the Form's properties.* Change the Text property of the Form to Calculator. Change the Font property to 9pt Segoe UI. Change the Size property of the Form to 258, 210.
- Adding a TextBox to the Form.* Set the TextBox's Text property in the Properties window to 0. Stretch the TextBox and position it as shown in Fig. 2.38. Set the TextAlign property to Right—this right aligns text displayed in the TextBox.
- Adding the first Panel to the Form.* Panel controls are used to group other controls. Add a Panel to the Form. Change the Panel's BorderStyle property to Fixed3D to make the inside of the Panel appear recessed. Change the Size property to 90, 120. This Panel will contain the calculator's numeric keys.
- Adding the second Panel to the Form.* Change the Panel's BorderStyle property to Fixed3D. Change the Size property to 62, 120. This Panel will contain the calculator's operator keys.
- Adding the third (and last) Panel to the Form.* Change the Panel's BorderStyle property to Fixed3D. Change the Size property to 54, 62. This Panel contains the calculator's C (clear) and C/A (clear all) keys.
- Adding Buttons to the Form.* There are 20 Buttons on the calculator. Add a Button to the Panel by dragging and dropping it on the Panel. Change the Text property of each Button to the calculator key it represents. The value you enter in the Text property will appear on the face of the Button. Finally, resize the Buttons, using their Size properties. Each Button labeled 0–9, *, /, -, = and . should have a size of 23, 23. The 00 Button has size 52, 23. The OFF Button has size 54, 23. The + Button is sized 23, 81. The C (clear) and C/A (clear all) Buttons are sized 44, 23.

2.10 (*Alarm Clock GUI*) Create the GUI for the alarm clock as shown in Fig. 2.39.

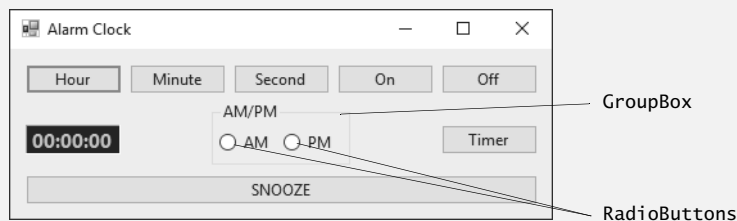


Fig. 2.39 | Alarm clock GUI.

- Manipulating the Form's properties.** Change the Text property of the Form to Alarm Clock. Change the Font property to 9pt Segoe UI. Change the Size property of the Form to 438, 170.
- Adding Buttons to the Form.** Add seven Buttons to the Form. Change the Text property of each Button to the appropriate text. Align the Buttons as shown.
- Adding a GroupBox to the Form.** GroupBoxes are like Panels, except that GroupBoxes display a title. Change the Text property to AM/PM, and set the Size property to 100, 50. Center the GroupBox horizontally on the Form.
- Adding AM/PM RadioButtons to the GroupBox.** Place two RadioButtons in the GroupBox. Change the Text property of one RadioButton to AM and the other to PM. Align the RadioButtons as shown.
- Adding the time Label to the Form.** Add a Label to the Form and change its Text property to 00:00:00. Change the BorderStyle property to Fixed3D and the BackColor to Black. Use the Font property to make the time bold and 12pt. Change the ForeColor to Silver (located in the Web tab) to make the time stand out against the black background. Position the Label as shown.

2.11 (Radio GUI) Create the GUI for the radio as shown in Fig. 2.40. [Note: The image used in this exercise is located in this chapter's examples folder.]

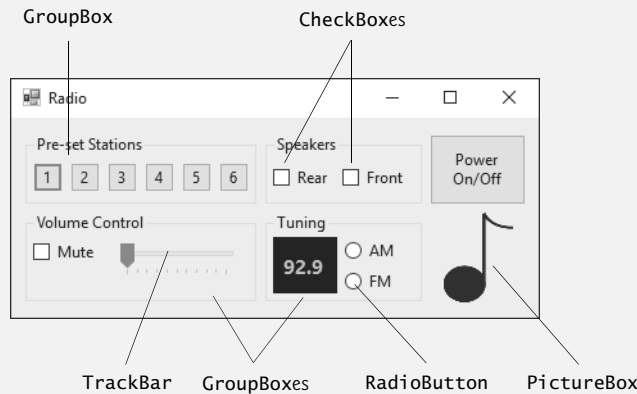


Fig. 2.40 | Radio GUI.

- Manipulating the Form's properties.** Change the Font property to 9pt Segoe UI. Change the Form's Text property to Radio and the Size to 427, 194.
- Adding the Pre-set Stations GroupBox and Buttons.** Set the GroupBox's Size to 180, 55 and its Text to Pre-set Stations. Add six Buttons to the GroupBox. Set each one's Size to 23, 23. Change the Buttons' Text properties to 1, 2, 3, 4, 5, 6, respectively.
- Adding the Speakers GroupBox and CheckBoxes.** Set the GroupBox's Size to 120, 55 and its Text to Speakers. Add two CheckBoxes to the GroupBox. Set the Text properties for the CheckBoxes to Rear and Front.
- Adding the Power On/Off Button.** Add a Button to the Form. Set its Text to Power On/Off and its Size to 75, 55.
- Adding the Volume Control GroupBox, the Mute CheckBox and the Volume TrackBar.** Add a GroupBox to the Form. Set its Text to Volume Control and its Size to 180, 70. Add a CheckBox to the GroupBox. Set its Text to Mute. Add a TrackBar to the GroupBox.

- f) *Adding the **Tuning** GroupBox, the radio station Label and the **AM/FM** RadioButtons.* Add a GroupBox to the Form. Set its Text to Tuning and its Size to 120, 70. Add a Label to the GroupBox. Set its AutoSize to False, its Size to 50, 44, its BackColor to Black, its ForeColor to Silver, its font to 12pt bold and its TextAlign to MiddleCenter. Set its Text to 92.9. Place the Label as shown in the figure. Add two RadioButtons to the GroupBox. Set the Text of one to AM and of the other to FM.
- g) *Adding the image.* Add a PictureBox to the Form. Set its SizeMode to StretchImage and its Size to 55, 70. Set the Image property to MusicNote.gif (located in this chapter's examples folder).