James W. Cooper

C# Design Patterns

A Tutorial

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Addison-Wesley was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers discounts on this book when ordered in quantity for special sales. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact:

International Sales
(317) 581-3793
international@pearsontechgroup.com

Visit Addison-Wesley on the Web: www.awprofessional.com

Library of Congress Cataloging-in-Publication Data

Cooper, James William, 1943–
C# design patterns : a tutorial / James W. Cooper.
p. cm.
Includes bibliographical references and index.
ISBN 0-201-84453-2 (alk. paper)
1. C# (Computer program language) 2. Software patterns. I. Title.

QA76.73.C154 C664 2003
005.13'3—dc21 2002074380

Copyright © 2003 by Pearson Education, Inc.

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 the prior consent of the publisher. Printed in the United States of America. Published simultaneously in Canada.

For information on obtaining permission for use of material from this work, please submit a written request to:

Pearson Education, Inc.
Rights and Contracts Department
75 Arlington Street, Suite 300
Boston, MA 02116
Fax: (617) 848-7047

Text printed on recycled and acid-free paper.

ISBN 0201844532

7 8 9 101112 DOC 08 07 06

7th Printing April 2006

CONTENTS

Preface *xix*

Acknowledgments *xxi*

Part 1 Object-Oriented Programming in C# **1**

Chapter 1 What Are Design Patterns? **3**

Defining Design Patterns 5

The Learning Process 6

Studying Design Patterns 7

Notes on Object-Oriented Approaches 7

C# Design Patterns 8

How This Book Is Organized 8

Chapter 2 Syntax of the C# Language **11**

Data Types 12

Converting between Numbers and Strings 13

Declaring Multiple Variables 14

Numeric Constants 14

Character Constants 14

Variables 15

Declaring Variables as You Use Them *15*

Multiple Equals Signs for Initialization 16

A Simple C# Program 16

Arithmetic Operators 17

Increment and Decrement Operators 18
Combining Arithmetic and Assignment Statements 18
Making Decisions in C# 19
Comparison Operators 20
Combining Conditions 21
The Most Common Mistake 21
The Switch Statement 22
C# Comments 23
The Ornery Ternary Operator 23
Looping Statements in C# 24
The While Loop 24
The Do-While Statement 24
The For Loop 24
Declaring Variables as Needed in For Loops 25
Commas in For Loop Statements 25
How C# Differs from C 26
How C# Differs from Java 27
Summary 27

Chapter 3 Writing Windows C# Programs **29**
Objects in C# 29
Managed Languages and Garbage Collection 30
Classes and Namespaces in C# 30
Building a C# Application 31
The Simplest Window Program in C# 32
Windows Controls 35
Labels 35
TextBox 36
CheckBox 36
Buttons 37
Radio Buttons 37
ListBoxes and ComboBoxes 38
The Items Collection 38

Menus 39
ToolTips 39
The Windows Controls Program 40
Summary 41
Programs on the CD-ROM 41

Chapter 4 Using Classes and Objects in C# **43**
What Do We Use Classes For? 43
A Simple Temperature Conversion Program 43
Building a Temperature Class 45
Converting to Kelvin 47
Putting the Decisions into the Temperature Class 47
Using Classes for Format and Value Conversion 48
Handling Unreasonable Values 51
A String Tokenizer Class 51
Classes as Objects 53
Class Containment 55
Initialization 56
Classes and Properties 56
Programming Style in C# 58
Delegates 59
Indexers 61
Operator Overloading 62
Summary 63
Programs on the CD-ROM 63

Chapter 5 Inheritance **65**
Constructors 65
Drawing and Graphics in C# 66
Using Inheritance 68
Namespaces 68
Creating a Square from a Rectangle 69
Public, Private, and Protected 71

Overloading 71
Virtual and Override Keywords 72
Overriding Methods in Derived Classes 72
Replacing Methods Using New 74
Overriding Windows Controls 74
Interfaces 76
Abstract Classes 76
Comparing Interfaces and Abstract Classes 79
Summary 80
Programs on the CD-ROM 80

Chapter 6 UML Diagrams 81
Inheritance 82
Interfaces 84
Composition 84
Annotation 85
WithClass UML Diagrams 86
C# Project Files 86

Chapter 7 Arrays, Files, and Exceptions in C# 87
Arrays 87
Collection Objects 88
ArrayLists 88
Hashtables 89
SortedLists 89
Exceptions 90
Multiple Exceptions 91
Throwing Exceptions 92
File Handling 92
The File Object 92
Reading a Text File 93
Writing a Text File 93
Exceptions in File Handling 94
Testing for End of File 94

A csFile Class 95
Program on the CD-ROM 96

Part 2 Creational Patterns 97

Chapter 8 The Simple Factory Pattern **99**
How a Simple Factory Works 99
Sample Code 100
The Two Derived Classes 100
Building the Simple Factory 101
Using the Factory 102
Factory Patterns in Math Computation 103
Summary 106
Thought Questions 106
Programs on the CD-ROM 106

Chapter 9 The Factory Method **107**
The Swimmer Class 109
The Events Classes 109
StraightSeeding 110
CircleSeeding 111
Our Seeding Program 111
Other Factories 112
When to Use a Factory Method 113
Thought Question 113
Program on the CD-ROM 113

Chapter 10 The Abstract Factory Pattern **115**
A GardenMaker Factory 115
The PictureBox 118
Handling the RadioButton and Button Events 119
Adding More Classes 120
Consequences of Abstract Factory 120

Thought Question 121
Program on the CD-ROM 121

Chapter 11 The Singleton Pattern 123
Creating Singleton Using a Static Method 123
Exceptions and Instances 124
Throwing the Exception 125
Creating an Instance of the Class 125
Providing a Global Point of Access to a Singleton 126
Other Consequences of the Singleton Pattern 126
Programs on the CD-ROM 127

Chapter 12 The Builder Pattern 129
An Investment Tracker 130
The Stock Factory 132
The CheckChoice Class 133
The ListboxChoice Class 134
Using the Items Collection in the ListBox Control 135
Plotting the Data 136
The Final Choice 137
Consequences of the Builder Pattern 138
Thought Questions 138
Program on the CD-ROM 139

Chapter 13 The Prototype Pattern 141
Cloning in C# 142
Using the Prototype 142
Cloning the Class 146
Using the Prototype Pattern 147
Dissimilar Classes with the Same Interface 150
Prototype Managers 152
Consequences of the Prototype Pattern 153
Thought Question 153
Programs on the CD-ROM 154
Summary of Creational Patterns 154

Part 3 Structural Patterns **155**

Chapter 14 The Adapter Pattern **157**

Moving Data between Lists 157
Making an Adapter 159
Using the DataGrid 160
Detecting Row Selection *162*
Using a TreeView 163
The Class Adapter 164
Two-Way Adapters 166
Object versus Class Adapters in C# 166
Pluggable Adapters 166
Thought Question 167
Programs on the CD-ROM 167

Chapter 15 The Bridge Pattern **169**

The Bridger Interface 170
The VisList Classes 172
The Class Diagram 173
Extending the Bridge 173
Windows Forms as Bridges 176
Consequences of the Bridge Pattern 177
Thought Question 177
Programs on the CD-ROM 177

Chapter 16 The Composite Pattern **179**

An Implementation of a Composite 180
Computing Salaries 180
The Employee Classes 181
The Boss Class 183
Building the Employee Tree 184
Self-Promotion 186
Doubly Linked Lists 187
Consequences of the Composite Pattern 188
A Simple Composite 188

Composites in .NET 189
Other Implementation Issues 189
Thought Questions 189
Programs on the CD-ROM 190

Chapter 17 The Decorator Pattern **191**
Decorating a CoolButton 191
Handling Events in a Decorator 193
Layout Considerations *194*
Control Size and Position *194*
Multiple Decorators 195
Nonvisual Decorators 197
Decorators, Adapters, and Composites 197
Consequences of the Decorator Pattern 198
Thought Questions 198
Programs on the CD-ROM 198

Chapter 18 The Façade Pattern **199**
What Is a Database? 199
Getting Data Out of Databases 201
Kinds of Databases 202
ODBC 202
Database Structure 203
Using ADO.NET 203
Connecting to a Database *204*
Reading Data from a Database Table *204*
Executing a Query *205*
Deleting the Contents of a Table *205*
Adding Rows to Database Tables Using ADO.NET 206
Building the Façade Classes 207
Building the Price Query *207*
Making the ADO.NET Façade 209
The DBTable Class *211*
Creating Classes for Each Table 213
Building the Price Table 215

Loading the Database Tables 218
The Final Application 219
What Constitutes the Façade? 220
Consequences of the Façade 220
Thought Question 221
Program on the CD-ROM 221

Chapter 19 The Flyweight Pattern **223**
Discussion 224
Example Code 224
The Class Diagram 228
Selecting a Folder 229
Handling the Mouse and Paint Events 230
Flyweight Uses in C# 231
Sharable Objects 231
Copy-on-Write Objects 232
Thought Question 232
Program on the CD-ROM 232

Chapter 20 The Proxy Pattern **233**
Sample Code 234
Proxies in C# 236
Copy-on-Write 237
Comparison with Related Patterns 237
Thought Question 237
Program on the CD-ROM 237
Summary of Structural Patterns 237

Part 4 Behavioral Patterns 239

Chapter 21 Chain of Responsibility **241**
Applicability 242
Sample Code 243
ListBoxes 246
Programming a Help System 248
Receiving the Help Command 250

A Chain or a Tree? 251
Kinds of Requests 252
Examples in C# 252
The Chain of Responsibility 253
Thought Question 253
Programs on the CD-ROM 253

Chapter 22 The Command Pattern 255
Motivation 255
Command Objects 256
Building Command Objects 257
Consequences of the Command Pattern 259
The CommandHolder Interface 259
Providing Undo 262
Thought Questions 268
Programs on the CD-ROM 268

Chapter 23 The Interpreter Pattern 269
Motivation 269
Applicability 269
A Simple Report Example 270
Interpreting the Language 271
Objects Used in Parsing 272
Reducing the Parsed Stack 276
Implementing the Interpreter Pattern 277
The Syntax Tree 278
Consequences of the Interpreter Pattern 281
Thought Question 282
Program on the CD-ROM 282

Chapter 24 The Iterator Pattern 283
Motivation 283
Sample Iterator Code 284
Fetching an Iterator 285
Filtered Iterators 286
The Filtered Iterator 286

Keeping Track of the Clubs 288
Consequences of the Iterator Pattern 289
Programs on the CD-ROM 290

Chapter 25 The Mediator Pattern **291**
An Example System 291
Interactions between Controls 292
Sample Code 294
Initialization of the System *297*
Mediators and Command Objects 297
Consequences of the Mediator Pattern 298
Single Interface Mediators 299
Implementation Issues 299
Program on the CD-ROM 299

Chapter 26 The Memento Pattern **301**
Motivation 301
Implementation 302
Sample Code 302
A Cautionary Note *308*
Command Objects in the User Interface 309
Handling Mouse and Paint Events 310
Consequences of the Memento 311
Thought Question 312
Program on the CD-ROM 312

Chapter 27 The Observer Pattern **313**
Watching Colors Change 314
The Message to the Media 318
Consequences of the Observer Pattern 318
Program on the CD-ROM 319

Chapter 28 The State Pattern **321**
Sample Code 321
Switching between States 325
How the Mediator Interacts with the StateManager 326
The ComdToolBarButton *327*

Handling the Fill State 330
Handling the Undo List 331
The VisRectangle and VisCircle Classes 333
Mediators and the God Class 335
Consequences of the State Pattern 335
State Transitions 336
Thought Questions 336
Program on the CD-ROM 336

Chapter 29 The Strategy Pattern **337**
Motivation 337
Sample Code 338
The Context 339
The Program Commands 339
The Line and Bar Graph Strategies 340
Drawing Plots in C# 341
Making Bar Plots *341*
Making Line Plots *342*
Consequences of the Strategy Pattern 344
Program on the CD-ROM 344

Chapter 30 The Template Method Pattern **345**
Motivation 345
Kinds of Methods in a Template Class 347
Sample Code 347
Drawing a Standard Triangle *349*
Drawing an Isosceles Triangle *349*
The Triangle Drawing Program 350
Templates and Callbacks 351
Summary and Consequences 352
Programs on the CD-ROM 352

Chapter 31 The Visitor Pattern **353**
Motivation 353
When to Use the Visitor Pattern 355

Sample Code 355
Visiting the Classes 357
Visiting Several Classes 357
Bosses Are Employees, Too 359
Catch-All Operations with Visitors 360
Double Dispatching 361
Why Are We Doing This? 361
Traversing a Series of Classes 362
Consequences of the Visitor Pattern 362
Thought Question 363
Program on the CD-ROM 363

Bibliography *365*
Index *367*

This page intentionally left blank

PREFACE

This is a practical book that tells you how to write C# programs using some of the most common design patterns. It also serves as a quick introduction to programming in the new C# language. The pattern discussions are structured as a series of short chapters, each describing a design pattern and giving one or more complete working, visual example programs that use that pattern. Each chapter also includes UML diagrams illustrating how the classes interact.

This book is not a "companion" book to the well-known *Design Patterns* text by the "Gang of Four." Instead, it is a tutorial for people who want to learn what design patterns are about and how to use them in their work. You do not have to have read *Design Patterns* to read this book, but when you are done here, you may well want to read or reread it to gain additional insights.

In this book, you will learn that design patterns are frequently used ways of organizing objects in your programs to make them easier to write and modify. You'll also see that by familiarizing yourself with them, you've gained some valuable vocabulary for discussing how your programs are constructed.

People come to appreciate design patterns in different ways—from the highly theoretical to the intensely practical—and when they finally see the great power of these patterns, an "Aha!" moment occurs. Usually this is the moment when you discover how that pattern can help you in *your* work.

In this book, we try to help you form that conceptual idea, or *gestalt,* by describing the pattern in as many ways as possible. The book is organized into six main sections: an introductory description, an introduction to C#, and descriptions of patterns that are grouped as creational, structural, and behavioral.

For each pattern, we start with a brief verbal description and then build simple example programs. Each of these examples is a visual program that you can run and examine to make the pattern as concrete a concept as possible. All of the example programs and their variations are on the companion CD-ROM, where you run them, change them, and see how the variations you create work.

Since each of the examples consists of a number of C# files for each of the classes we use in that example, we provide a C# project file for each example and place each example in a separate subdirectory to prevent any confusion. This book assumes you have and will be using a copy of Visual Studio.NET, which comes in several versions. We used the Professional Edition in developing the code samples.

If you leaf through the book, you'll see screenshots of the programs we developed to illustrate the design patterns, providing yet another way to reinforce your learning of these patterns. In addition, you'll see UML diagrams of these programs, illustrating the interactions between classes in yet another way. UML diagrams are just simple box-and-arrow illustrations of classes and their inheritance structure, where arrows point to parent classes, and dotted arrows point to interfaces. And if you're not yet familiar with UML, we provide a simple introduction in the second chapter. All of the diagrams were produced using WithClass 2000, and a demonstration version of that program is included on the CD-ROM.

When you finish this book, you'll be comfortable with the basics of design patterns and will be able to start using them in your day-to-day C# programming work.

James W. Cooper
Nantucket, MA
Wilton, CT
Kona, HI

CHAPTER 3

Writing Windows C# Programs

The C# language has its roots in C++, Visual Basic, and Java. Both C# and VB.Net use the same libraries and compile to the same underlying code. Both are managed languages with garbage collection of unused variable space, and both can be used interchangeably. Both also use classes with method names that are very similar to those in Java, so if you are familiar with Java, you will have no trouble with C#.

Objects in C#

In C#, everything is treated as an object. Objects contain data and have methods that operate on them. For example, strings are now objects. They have methods such as these.

```
Substring
ToLowerCase
ToUpperCase
IndexOf
Insert
```

Integers, float, and double variables are also objects, and they have methods.

```
string s;
float x;
x = 12.3;
s = x.ToString();
```

Note that conversion from numerical types is done using these methods rather than external functions. If you want to format a number as a particular kind of string, each numeric type has a Format method.

Managed Languages and Garbage Collection

C# and VB.Net are both *managed* languages. This has two major implications. First, both are compiled to an intermediate low-level language, and a common language runtime (CLR) is used to execute this compiled code, perhaps compiling it further first. So, not only do C# and VB.Net share the same runtime libraries, they are to a large degree two sides of the same coin and two aspects of the same language system. The differences are that VB7 is more Visual Basic–like and a bit easier for VB programmers to learn and use. C# on the other hand is more C++- and Java-like and may appeal more to programmers already experienced in those languages.

The other major implication is that managed languages are *garbage-collected*. Garbage-collected languages take care of releasing unused memory. (You never have to be concerned with this.) As soon as the garbage-collection system detects that there are no more active references to a variable, array, or object, the memory is released back to the system. Of course, it is still possible to write memory-eating code, but for the most part, you do not have to worry about memory allocation and release problems.

Classes and Namespaces in C#

All C# programs are composed entirely of classes. Visual windows forms are a type of class. You will see that all the program features we'll be writing are composed of classes. Since everything is a class, the number of names of class objects can be overwhelming. They have therefore been grouped into various functional libraries that you must specifically mention in order to use the functions in these libraries.

Under the covers these libraries are each individual DLLs. However, you need only refer to them by their base names, using the using statement, and the functions in that library are available to you.

```
using System;
using System.Drawing;
using System.Collections;
```

Logically, each of these libraries represents a different *namespace*. Each namespace is a separate group of class and method names, which the compiler will recognize after you declare that namespace. You can use namespaces that contain identically named classes or methods, but you will only be notified of a conflict if you try to use a class or method that is duplicated in more than one namespace.

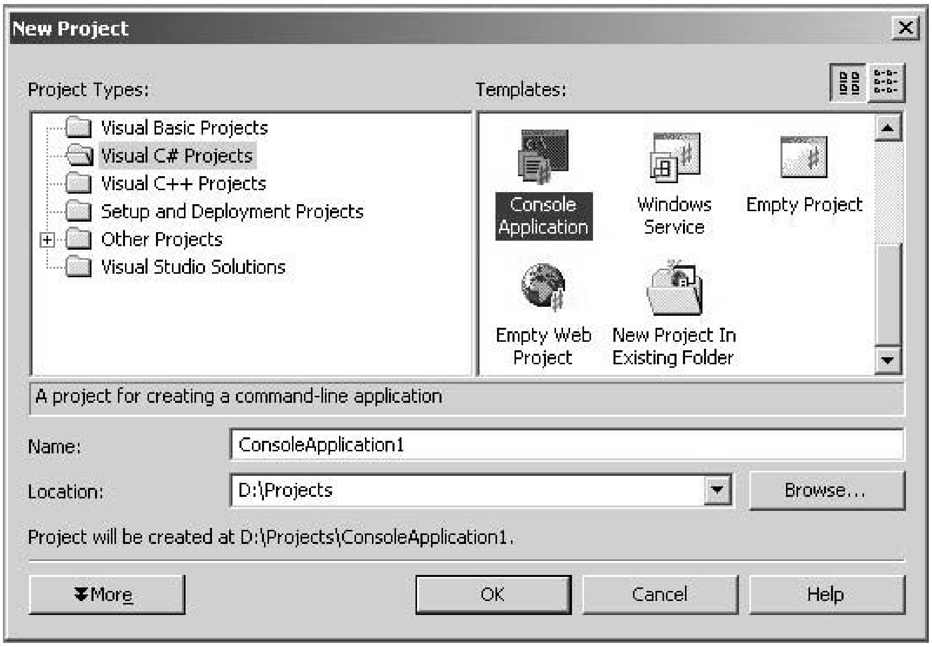
The most common namespace is the System namespace, and it is imported by default without your needing to declare it. It contains many of the most fundamental classes and methods that C# uses for access to basic classes such as Application, Array, Console, Exceptions, Objects, and standard objects such as byte, bool, and string. In the simplest C# program we can simply write out a message to the console without ever bringing up a window or form.

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

This program just writes the text "Hello C# World" to a command (DOS) window. The entry point of any program must be a Main method, and it must be declared as static.

Building a C# Application

Let's start by creating a simple console application—that is, one without any windows that just runs from the command line. Start the Visual Studio.NET program, and select File | New Project. From the selection box, choose C# Console application, as shown in Figure 3-1.



A project for creating a command-line application

Name: ConsoleApplication1

Location: D:\Projects

Project will be created at D:\Projects\ConsoleApplication1.

Figure 3-1 The New Project selection window: selecting a console application.

This will bring up a module with Main already filled in. You can type in the rest of the code as follows.

```
    Console.WriteLine ("Hello C# World");
```

You can compile this and run it by pressing F5.

When you compile and run the program by pressing F5, a DOS window will appear and print out the message "Hello C# World" and then exit.

The Simplest Window Program in C#

C# makes it very easy to create Windows GUI programs. In fact, you can create most of it using the Windows Designer. To do this, start Visual Studio.NET, select File | New Project, and select C# Windows Application. The default name (and filename) is WindowsApplication1, but you can change this before you close the New dialog box. This brings up a single form project, initially called Form1.cs. You can then use the Toolbox to insert controls, just as you can in Visual Basic.

The Windows Designer for a simple form with one text field and one button is shown in Figure 3-2.

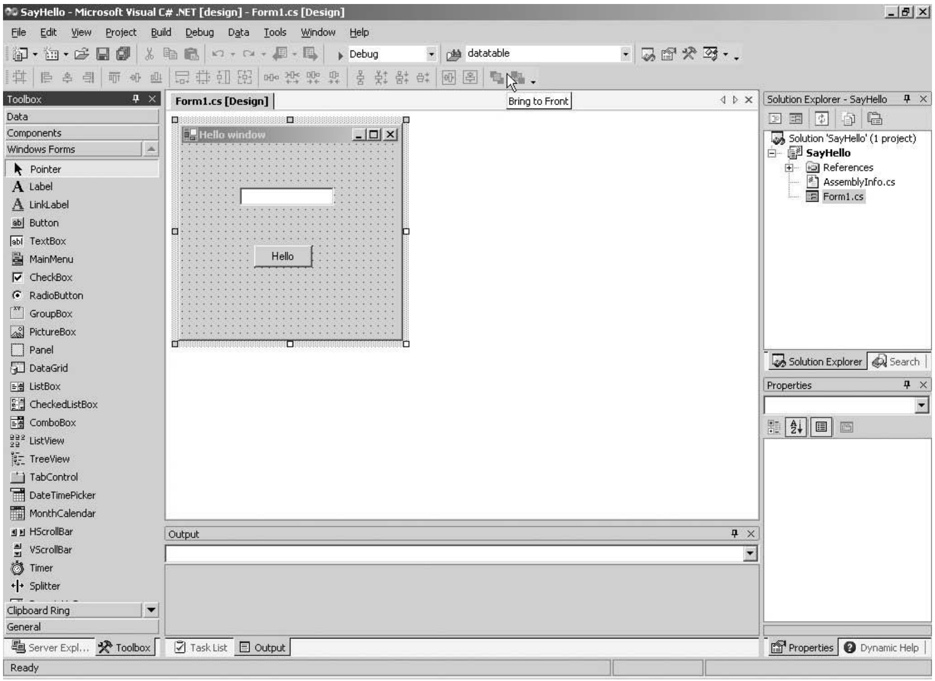


Figure 3-2 The Windows Designer in Visual Studio.NET

You can draw the controls on the form by selecting the TextBox from the Toolbox, dragging it onto the form, and then doing the same with the button. Then, to create program code, we need only double-click on the controls. In this simple form, we want to click on the “Hello” button, which copies the text from the text field to the textbox we called txHi and clears the text field. So in the designer, we double-click on that button, and this code is automatically generated.

```
private void btHello_Click(object sender, EventArgs e) {
txHi.Text ="Hello there";
}
```

Note that the Click routine passes in a sender object and an event object that you can query for further information. Under the covers, it also connects the event to this method. The running program is shown in Figure 3-3.

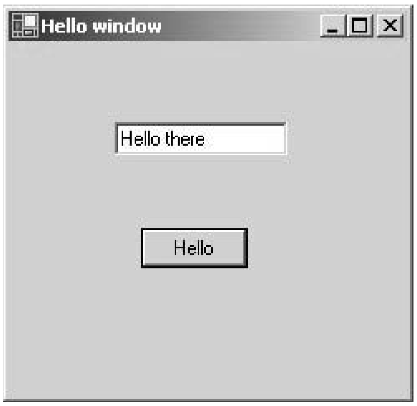


Figure 3-3 The SimpleHello form after clicking the Say Hello button

While we only had to write one line of code inside the previous subroutine, it is instructive to see how different the rest of the code is for this program. We first see that several libraries of classes are imported so the program can use them.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

Most significant is the Windows.Forms library, which is common to all the .NET languages.

The code the designer generates for the controls is illuminating—and it is right out there in the open for you to change if you want. Essentially, each control is declared as a variable and added to a container. Here are the control declarations. Note the event handler added to the btHello.Click event.

```
private System.Windows.Forms.TextBox txHi;
private System.Windows.Forms.Button btHello;

private void InitializeComponent()        {
      this.btHello = new System.Windows.Forms.Button();
      this.txHi = new System.Windows.Forms.TextBox();
      this.SuspendLayout();
      //
      // btHello
      //
      this.btHello.Location =
                  new System.Drawing.Point(80, 112);
      this.btHello.Name = "btHello";
      this.btHello.Size = new System.Drawing.Size(64, 24);
      this.btHello.TabIndex = 1;
      this.btHello.Text = "Hello";
      this.btHello.Click +=
            new EventHandler(this.btHello_Click);
      //
      // txHi
      //
      this.txHi.Location =
            new System.Drawing.Point(64, 48);
      this.txHi.Name = "txHi";
      this.txHi.Size = new System.Drawing.Size(104, 20);
      this.txHi.TabIndex = 0;
      this.txHi.Text = "";
      //
      // Form1
      //
      this.AutoScaleBaseSize =
            new System.Drawing.Size(5, 13);
      this.ClientSize = new System.Drawing.Size(240, 213);
      this.Controls.AddRange(
            new System.Windows.Forms.Control[] {
              this.btHello,
              this.txHi} );
              this.Name = "Form1";
              this.Text = "Hello window";
              this.ResumeLayout(false);

            }
```

If you change this code manually instead of using the property page, the window designer may not work anymore. We'll look more at the power of this system after we discuss objects and classes in the next chapter.

Windows Controls

All of the basic Windows controls work in much the same way as the TextBox and Button we have used so far. Many of the more common ones are shown in the Windows Controls program in Figure 3-4.

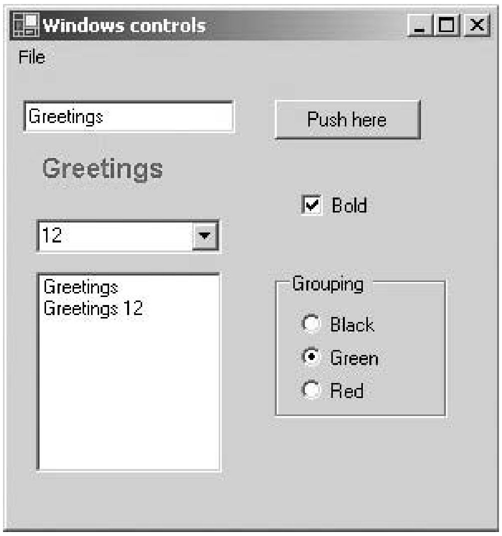


Figure 3-4 A selection of basic Windows controls

Each of these controls has properties such as Name, Text, Font, Forecolor, and Borderstyle that you can change most conveniently using the properties window shown at the right of Figure 3-2. You can also change these properties in your program code as well. The Windows Form class that the designer generates always creates a Form1 constructor that calls an InitializeComponent method like the preceding one. Once that method has been called, the rest of the controls have been created, and you can change their properties in code. Generally, we will create a private *init()* method that is called right after the InitializeComponent method, in which we add any such additional initialization code.

Labels

A label is a field on the window form that simply displays text. Usually programmers use this to label the purpose of text boxes next to them. You can't click on a label or tab to it so it obtains the focus. However, if you want, you can change the major properties in Table 3-1 either in the designer or at runtime.

Table 3-1 Properties for the Label Control

Property	Value
Name	At design time only
BackColor	A Color object
BorderStyle	None, FixedSingle, or Fixed3D
Enabled	True or false. If false, grayed out.
Font	Set to a new Font object
ForeColor	A Color object
Image	An image to be displayed within the label
ImageAlign	Where in the label to place the image
Text	Text of the label
Visible	True or false

TextBox

The TextBox is a single line or multiline editable control. You can set or get the contents of that box using its Text property.

```
TextBox tbox = new TextBox();
tbox.Text = "Hello there";
```

In addition to the properties in Table 3-1, the TextBox also supports the properties in Table 3-2.

CheckBox

A CheckBox can be either checked or not, depending on the value of the Checked property. You can set or interrogate this property in code as well as in the designer. You can create an event handler to catch the event when the box is checked or unchecked by double-clicking on the checkbox in the design mode.

CheckBoxes have an Appearance property that can be set to *Appearance.Normal* or *Appearance.Button*. When the appearance is set to the Button value,

Table 3-2 TextBox Properties

Property	Value
`Lines`	An array of strings, one per line
`Locked`	If true, you can't type into the text box
`Multiline`	True or false
`ReadOnly`	Same as locked. If true, you can still select the text and copy it, or set values from within code.
`WordWrap`	True or false

the control acts like a toggle button that stays depressed when you click on it and becomes raised when you click on it again. All the properties in Table 3-1 apply as well.

Buttons

A Button is usually used to send a command to a program. When you click on it, it causes an event that you usually catch with an event handler. Like the CheckBox, you create this event handler by double-clicking on the button in the designer. All of the properties in Table 3-1 can be used as well.

Buttons are also frequently shown with images on them. You can set the button image in the designer or at runtime. The images can be in bmp, gif, jpeg, or icon files.

Radio Buttons

Radio buttons or option buttons are round buttons that can be selected by clicking on them. Only one of a group of radio buttons can be selected at a time. If there is more than one group of radio buttons on a window form, you should put each set of buttons inside a Group box as we did in the program in Figure 3-4. As with checkboxes and buttons, you can attach events to clicking on these buttons by double-clicking on them in the designer. Radio buttons do not always have events associated with them. Instead, programmers check the Checked property of radio buttons when some other event, like an OK button click, occurs.

ListBoxes and ComboBoxes

Both ListBoxes and ComboBoxes contain an Items array of the elements in that list. A ComboBox is a single-line drop-down that programmers use to save space when selections are changed less frequently. ListBoxes allow you to set properties that allow multiple selections, but ComboBoxes do not. Some of their properties include those in Table 3-3.

Table 3-3 The ListBox and ComboBox Properties

Property	Value
Items	A collection of items in the list
MultiColumn	If true, the ColumnWidth property describes the width of each column. (Does not apply to ComboBox.)
SelectionMode	One, MultiSimple, or MultiExtended. If set to MultiSimple, you can select or deselect multiple items with a mouse click. If set to MultiExtended, you can select groups of adjacent items with a mouse. (Does not apply to ComboBox.)
SelectedIndex	Index of selected item
SelectedIndices	Returns collection of selections when ListBox selection mode is multiple.
SelectedItem	Returns the item selected

The Items Collection

You use the Items collection in the ListBox and ComboBox to add and remove elements in the displayed list. It is essentially an ArrayList, as we discuss in Chapter 7. The basic methods are shown in Table 3-4.

If you set a ListBox to a multiple selection mode, you can obtain a collection of the selected items or the selected indexes by

```
ListBox.SelectedIndexCollection it =
      new ListBox.SelectedIndexCollection (lsCommands);
ListBox.SelectedObjectCollection so =
      new ListBox.SelectedObjectCollection (lsCommands);
```

where *lsCommands* is the ListBox name.

Table 3-4 Methods for the Items Collection

Method	**Value**
Add	Add object to list
Count	Number in list
Item[i]	Element in collection
RemoveAt(i)	Remove element i

Menus

You add a menu to a window by adding a MainMenu control to the window form. Then you can select the menu control and edit its drop-down names and new main item entries, as shown in Figure 3-5.

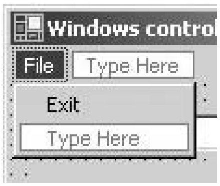


Figure 3-5 Adding a menu to a form

As with other clickable controls, double-clicking on one in the designer creates an event whose code you can fill in.

ToolTips

A ToolTip is a box that appears when your mouse pointer hovers over a control in a window. This feature is activated by adding an (invisible) ToolTip control to the form and then adding specific ToolTip control and text combinations to the control. In our example in Figure 3-4, we add ToolTips text to the button and ListBox using the *tips* control we have added to the window.

```
tips.SetToolTip (btPush, "Press to add text to list box");
tips.SetToolTip (lsCommands, "Click to copy to text box");
```

This is illustrated in Figure 3-6.

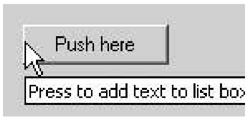


Figure 3-6 A ToolTip over a button

We discuss how to use the DataGrid and TreeList in Chapters 14 and 15, and Toolbar is discussed in Chapters 28 and 29.

The Windows Controls Program

The Windows Controls program, shown in Figure 3-4, controls changes in the text in the label.

- Font size is set from the combo box.
- Font color is set from the radio buttons.
- Boldface is set from the check box.

For the check box, we create a new font that is either lightface or boldface, depending on the state of the check box.

```
private void ckBold_CheckedChanged(object sender, EventArgs e) {
       if (ckBold.Checked ) {
              lbText.Font  =new Font ("Arial",
                           fontSize,FontStyle.Bold );
       }
       else {
              lbText.Font = new Font ("Arial", fontSize);
}
```

When we create the form, we add the list of font sizes to the combo box.

```
private void init() {
       fontSize = 12;
       cbFont.Items.Add ("8");
       cbFont.Items.Add ("10");
       cbFont.Items.Add ("12");
       cbFont.Items.Add ("14");
       cbFont.Items.Add ("18");
       lbText.Text ="Greetings";
       tips.SetToolTip (btPush, "Press to add text to list box");
       tips.SetToolTip (lsCommands, "Click to copy to text box");
}
```

When someone clicks on a font size in the combo box, we convert that text to a number and create a font of that size. Note that we just call the check box changing code so we don’t have to duplicate anything.

```
private void cbFont_SelectedIndexChanged(
             object sender, EventArgs e) {
       fontSize= Convert.ToInt16 (cbFont.SelectedItem );
       ckBold_CheckedChanged(null, null);
}
```

For each radio button, we click on it and insert color-changing code.

```
private void opGreen_CheckedChanged(object sender, EventArgs e) {
       lbText.ForeColor =Color.Green;
}

private void opRed_CheckedChanged(object sender, EventArgs e) {
       lbText.ForeColor =Color.Red ;
}

private void opBlack_CheckedChanged(object sender, EventArgs e) {
       lbText.ForeColor =Color.Black ;
}
```

When you click on the ListBox, it copies that text into the text box by getting the selected item as an object and converting it to a string.

```
private void lsCommands_SelectedIndexChanged(
             object sender, EventArgs e) {
       txBox.Text = lsCommands.SelectedItem.ToString () ;
}
```

Finally, when you click on the File | Exit menu item, it closes the form and, hence, the program.

```
private void menuItem2_Click(object sender, EventArgs e) {
       this.Close ();
}
```

Summary

Now that we've seen the basics of how to write programs in C#, we are ready to talk more about objects and OO programming in the chapters that follow.

Programs on the CD-ROM

Console Hello	`\IntroCSharp\Hello`
Windows Hello	`\IntroCSharp\SayHello`
Windows Controls	`\IntroCSharp\WinControls`

INDEX

Note: *Italicized* page locators refer to figures/tables.

Symbols
+, addition operator, 18
\\, backslash character, 15
\b, backspace character, 15
&, bitwise AND, 18
^, bitwise exclusive OR, 18
|, bitwise OR, 18
\r, carriage return, 15
--, decrement operator, 18, 19
/, division operator, 18
\", double quote, 15
\f, form feed, 15
>, greater than operator, 21
>=, greater than or equal to operator, 21
++, increment operator, 18, 19
==, is equal to operator, 21
!=, is not equal to operator, 21
<<*n,* left shift *n* places, 18
<, less than operator, 21
<=, less than or equal to operator, 21
&&, logical AND, 21
||, logical OR, 21
%, modulo, 18
*, multiplication operator, 18
\n, newline feed, 15
\0, null character, 15
>>*n,* right shift *n* places, 18
\', single quote, 15
-, subtraction operator, 18
\t, tab character, 15

A
Absolute zero, and Kelvin scale, 47
Abstract classes, 7, 76–80, 192
Abstract class system, and drawing rectangle and circle, *78*
AbstractEmployee class, 181
  Employee and Boss derived from, *184*
AbstractEmployee interface, 182
AbstractEmployee subclasses, 187
AbstractExpression object, in Interpreter pattern, 277
Abstract Factory pattern, 9, 97, 115–121, 154
  and Builder pattern, 138
  consequences of, 120
  and GardenMaker factory, 115–120
  more classes added to, 120
Abstraction, in Bridge pattern, 169
abstract keyword, 82

Acceptance, and learning design patterns, 6
accept method, 354, 356, 358, 361
  adding to subclasses, *355*
Accessor methods, 45
Action methods, 299
ActionVerb class, 280
ActiveX controls, 176
ActiveX Data Objects, 203
Adapter class, 318
Adapter pattern, 9, 155, 157–167, 237
  and Bridge pattern, 169
  and class adapter, 164–166
  making an adapter, 159–160
  moving data between lists, 157–158
  moving the DataGrid, 160–164
  object *versus* class adapters in C#, 166
  pluggable adapters, 166
  Proxy pattern, 237
  two-way adapters, 166
Adapters, 157
  decorators, composites and, 197–198
  making, 159–160
Addition operator (+), 18
Address class, Person and Group derived from, *129*
addToChain method, 244
Add2.cs program, 16–17
ADO DataSet manipulation, and Clone method, 142
ADO.NET, 199, 203
  connecting to database in, 204
  deleting contents of table in, 205–206
  executing query in, 205
  reading data from database table in, 204–205
  rows added to database tables with, 206–207
  using, 203–207
ADO.NET database connections
  as proxies, 236
  in Façade pattern, 9
ADO.NET Façade
  DBTable class, 211–213
  making, 209–213
AgeSwimData class, 151
  age distribution display for, *150*
Aggregation (in UML diagrams), diamonds indicating, 84
Algebraic strings, parsing, 270
Algorithms
  and Strategy pattern, 337
  Strategy pattern and dynamic selection of, 344
  and Template pattern, 345
AND operation, 17
Annotation, and UML diagrams, 85–86
AnnualGarden class, 117
ANSI standards, and SQL, 202
Appearance property, in CheckBoxes, 36
Application class, 31
Approach, 202
Arithmetic operators, 17, *18*
Arithmetic statements, combining assignment statements and, 18–19
Array class, 31
ArrayList, 38, 290, 361
  of child nodes, 180
  Kid objects in, 284
  Observer objects added to, 314
  and Prototype pattern, 147
  and Stack class, 271
  StoreFoodPrice objects in, 215
  and TriangleForm class, 350
ArrayList methods, 88, *89*
ArrayList object, 54, 88
Arrays, in C#, 87–88
Arrow-object state, 336
Assignment statements, combining arithmetic statements and, 18–19
Autohighlight feature, of C# TextBox, 75
Automobile factories, 115
AxsDatabase class, 210

B
Backslash character (\\), 15
Backspace character (\b), 15
Bar charts, 338, 345
Bar graph, and Strategy pattern, *343*
BarGraph strategy, 344
BarPlot class, 345
  LinePlot class derived from, 342
Bar plots, making, 341–342
BarPlotStrategy class, 340, 341
BarPlot window, 341
Base classes, and Template method, 345, 349, 352
Behavioral patterns, 6, 97, 239–363
  Chain of Responsibility, 241–253
  Command, 255–268
  Interpreter, 269–282
  Iterator, 283–290
  Mediator, 291–299
  Memento, 301–312
  Observer, 313–319
  State, 321–336
  Strategy, 337–344
  Template Method, 345–352
  Visitor, 353–363
Bitfields, 26
BitList class, 61
Bitmap class, 235
Bitmaps, 231
Bitwise And (&), 18
Bitwise exclusive Or (^), 18
Bitwise operators, 17
Bitwise Or (_), 18
BlueCommand class, 265
  derived from ColorCommand class, 267
BlueCommand object, and isUndo method, 265
Blue radio button, in watching colors change program, 314
bmp files, and Buttons, 37
Bonds, with check box interface, *131*
Booch, Grady, 81
Boolean operators, 17
Boolean variables, 13
Boolean variable type, spelling in C#, 27
bool object, 31
Borland Paradox, 202
Boss class, 183–184, 187, 357, 358
  derived from AbstractEmployee class, *184*
  visitor classes visiting, *359*
BossVisitor, 361
Braces ({}), 70
  classes/functions surrounded by, 17
  indentation styles for, 20
  statements in condition enclosed within, 19
break statement, 22, 27
Bridge pattern, 9, 155, 169–177, 237
  consequences of, 177
  extending the bridge, 173–176
  UML diagram for, *173*
  VisList classes, 172
  Windows forms as bridges, 176
Bridger class, 173
Bridger interface, defined, 170
btClone_Click event, 149
btConvert_click method, 44
Builder pattern, 76, 98, 107, 129–139, 154
  consequences of, 138
  inheritance relationships in, *138*
  Investment Tracker, 130–135
  and Prototype pattern, 141
  using Items collection in ListBox control, 135–137
Builder program, 176
Builder tools, 176
Butterfly class, 104
Button class, 256, 257, 260, 294, 339
Button events, handling, 119
Buttons, 37, 58, 189, 255, 256
  decorating, 191
  ToolTip over, *40*
Button value, 36

bVacationVisitor class, 358, 359
byte object, 31

C
C, 11, 313
  arrays in line with style used in, 87
  differences between C# and, 26
C#, 1, 25. *See also* Writing Windows C# programs
  application building, 31–32
  arithmetic and assignment statements combined in, 18–19
  arithmetic operators in, 17
  character constants in, 14–15
  classes used in, 43
  class module created from IDE of, 45
  cloning in, 142
  commas in for loop statements in, 25–26
  comments in, 23
  common Exception classes in, *91*
  comparison operators in, 20–21
  conditions combined in, 21
  converting between numbers and strings in, 13–14
  and database structure, 203
  decision making in, 19–20
  declaring multiple variables in, 14
  differences between C and, 26
  differences between Java and, 27
  do-while statement in, 24
  flyweight uses in, 231
  for loop in, 24–25
  IEnumerator interface in, 284
  increment and decrement operators in, 18
  logical operators in, *18*
  looping statements in, 24
  as managed language, 30
  most common mistake in, 21–22
  multiple equal signs for initialization in, 16
  numeric constants in, 14
  object *versus* class adapters in, 166
  plots drawn in, 341
  programming style in, 58
  project files, 86
  proxies in, 236
  simple program in, 16–17
  States implementing common interface in, 336
  switch statement in, 22
  syntax of, 11–27
  ternary operator, 23
  variables in, 15–16
  while loop in, 24
C++, 7, 11, 284, 313
  arrays in line with style used in, 87
  C# roots in, 29
  friend construction in, 302
Caching results, and Composite pattern, 189
Capital class, 60
Capitalization, 27
Caretaker class, 304, 307, 311
Caretaker object, and Memento pattern, 302
Carriage return (\r), 15
Cascading notifications, 318
Case sensitivity, with C#, 11
Casting, 13
Catch blocks, and multiple exceptions, 91–92
catch statement, and exceptions, 90
Celsius temperature scale, 43
  absolute zero on, 47
CEO Employee, 184
Chain class, 244, 248, 253
Chain interface, 248, 253
Chain of responsibility pattern, 8, 239, 241–253, 280
  applicability of, 242
  chain or tree?, 251–252
  class diagram for help system, *251*
  and Command pattern, 255
  examples in C#, 252–253
  Help chain, *241*
  implementation of as linear chain, *252*

implementation of as tree structure, *251*
kinds of requests with, 252
ListBoxes, 246–248
programming a Help system, 248–251
program on the CD-ROM, 253
purpose of, 253
sample code, 243–246
simple Chain of Responsibility, *242*
and syntax tree, 278
visual command interpreter program sample, *243*
Chain of Responsibility program, class structure of, *247*
Character constants, 14–15
CheckBox, 36–37
Check Box Builder, 133
Check boxes, 189, 255
Check box interface, bonds with, *131*
CheckChoice class, 133–134
Checked property, of radio buttons, 37
Child nodes, ArrayList of, 180
Children, and nodes, 179
Circle button, 321, 323
Circle class, 77
CircleSeeding, 111
instance of, returned by PrelimEvent class, 110
subclass, 108, *109*
Circle seeding
of 100 Free, *112*
of swimmers, 107
Circle state object, 324
Class adapter, 157, 164–166
list adapter approach to, *165*
object adapter *versus,* 166
Class-based Adapter, 169
Class containment, 55
Classes, 1, 17, 63
abstract, 76–78
in C#, 30–31
cloning, 145, 146–147
and design patterns, 7
fetching iterators in, 285–286
in Filtered enumeration, *288*
and flyweights, 224
for format and value conversion, 48–51
interaction between for simple drawing program, *322*
isolation of, with Abstract Factory, 120
and Mediator pattern, 291
as objects, 53–54
and properties, 56–58
for tables in Façade pattern, 213–215
tangled web of interactions between, in Mediator pattern visual interface, *293*
traversing series of, 362
uses for, 43
visiting, 357–358
Class instances, cloning, 142
Class libraries, and visitors, 355
Class methods, declaring as public, private, and protected, 71
Class modules, 45
Class patterns, object patterns *versus,* 155
Clear button, 291, 294, 297, 302, 321
behavior of, 323
Command objects for, 309
Clear method, and Mediator, 295
Click events, event handler for, 295
Client object, in Interpreter pattern, 277
cloneMe method, inheritance of by AgeSwimData class, 150
Clone method, 142, 146, 149
Cloning
in C#, 142
classes, 145, 146–147
and Prototype pattern, 141, 153
closeTriangle method, 352
CLR. *See* Common language runtime
Cocoon class, 105
Colleague, and Mediator, 293
Colleague elements, 299

Collection objects, 88–90
Collisions, avoiding, 68
ColObserver class, 315–316, 318
Colon (:), 70
  and parsing, 49
ColorChain class, 245
Color-changing code, in Windows Controls program
ColorCommand class, 265–266
  classes derived from, 267
ColorForm observer, 316
ColorImage class, 248
Color objects, 316
Color observer, *317*
Colors, watching changes in, 314–317
Columns, 199, 203
ComboBox properties, *38*
ComdToolBarButton class, 327–328
Command buttons, 297
  automatic language generation with, 281
CommandHolder approach, UML diagram of, *262*
CommandHolder interface, 259–262
Command interface, 294, 309, 339
  ExitCommand implementing, *84*
  implementing in new classes, 257
  and Undo function, 263
  and unDo method, 265
CommandMenu class, 260
Command objects, 256–257, 259, 261, 277
  building, 257–259
  interactions between Mediator object and, *198*
  Mediators and, 297
  and Undo command, 264
Command pattern, 8, 239, 255–268, 280, 311
  and building command objects, 257–259
  and CommandHolder interface, 259–262
  and command objects, 256–257
  consequences of, 259
  and Mediator pattern, 297
  menu designer interface, *258*
  menu part of using ComandHolder interface, *262*
  motivation for, 255–256
  programs on the CD-ROM, 268
  purpose of, 256
  simple program demo, *255*
  undo function provided by, 262–267
Commands, undoing, 263
Commas, in for loop statements, 25–26
Comments, 17, 23
  shown in box with turned-down corner, *86*
  in UML diagram, 85
Commercial languages, licensing fees and embedding of, 269
Common language runtime, 30
Communication patterns, 5
Company class
  with any number of instances of Employee, *85*
  instances of Person and Employee in, *84*
CompareTo method, 147, 173
Comparison operators, 20–21, *21*
Compilation errors, 22
Compiler error messages, 25
Complex class, 62
Components, ordering, 189
Composite pattern, 9, 155, 179–190, 237, 283, 355, 362
  and Boss class, 183–184
  building Employee tree, 184–186
  composites in .NET, 189
  computing salaries, 180–181
  consequences of, 188
  doubly linked lists, 187
  and Employee class, 181–183
  implementation of composite, 180
  other implementation issues with, 189
  programs on the CD-ROM, 190

and self-promotion, 186–187
and simple composite, 188–189
Composites, adapters, decorators and, 197–198
Composition, and UML diagrams, 84–85
ConcreteImplementors, 173
in Bridge pattern, 169
Concrete methods, 347
Conditions, combining, 21
Console class, 17, 31
Constants
character, 14–15
numeric, 14
const modifier, in C#, 12
Constructor method, for Rectangle class, 68
Constructors, 43, 72, 80
Containers, 189
for Command object, 259
as Decorators, 192
Containment, *5*
list of swimmers and their times using, *55*
contains method, 306
Context class, 323, 337, 339
Context interface, 344
Context object, in Interpreter pattern, 277
ControlChain class, 249–250
Controller, 4
Control names, 58
Controls
in C#, 176, 249–250
collection, 189
interactions between, 292–293
Mediator and initialization of, 297
Controls array, and Windows form containing buttons, 194
Convert methods, for converting between numbers and strings, 13
CoolBar, 192
CoolButton, *195*
decorating, 191–193
decoration of, with SlashDecorator, *196*
CoolDecorator, 196
Coplien, J. O., 141
Copy button, 291
Copy method, and Mediator, 295
Copy-on-write
objects, 232
and Proxy pattern, 237
Count property, 88, 89, 180
Coupling between objects, reducing, 253
CpyButton class, 294
CreateText method, 93
Creational patterns, 6, 97–154, 123
Abstract Factory, 115–121
Builder, 129–139
Factory Method, 107–113
Prototype, 141–154
Simple Factory, 99–106
Singleton, 123–127
summary of, 154
csFile class, 95–96, 142
csPatterns namespace, 229
Current() property, 284
currentState variable, 325
Cursor class, 235
Customer version, of ListBox, 172

D
Data
extrinsic, 223, 224, 226
getting out of databases, 201–202
intrinsic, 223, 232
moving between lists, 157–158
plotting in Builder pattern, 136
Database class, 220
Databases
connecting with ADO.Net, 204
description of, 199–201
getting data out of, 201–202
kinds of, 202
reading data from with ADO.Net, 204–205
structure of, 203

Database tables
  loading and Façade pattern, 218–219
  rows added to using ADO.Net, 206–207
DataColumns, 160
DataGrid, 9, 40
  and row selection detection, 162
  using, 160–162
DataGrid type, 166
Data manipulation, interface manipulation separated from, 45
Data model, 3
Data modification, and Iterator pattern, 289–290
DataRow, 203, 220
DataSet, 142, 203, 220
Data streams, 94
DataTable class, 160, 161, 203, 220
Data types, in C#, *12*
dBase, 202, 209, 220, 232
DBTable, 211–213, 215
  classes derived from, 207, 213
  classes wrapped in by Façade, 220
DB/2 (IBM), 2022
Decisions
  making in C#, 19–20
  in Temperature class, 47–48
Declaration
  property, 56
  multiple variable, 14
  variable, 15–16, 25
Decorator class, 191
Decorator panel, code for button added to, 194
Decorator pattern, 155, 191–198, 238
  consequences of, 198
  decorating CoolButton, 191–193
  decorators, adapters, composites and, 198
  and event handling, 193–194
  and multiple decorators, 195–196
  and nonvisual decorators, 197
  programs on the CD-ROM, 198
  and Proxy pattern, 237
Decrement operator (--), 18, 19
Deep copy, 147
Default case, for switch statements, 22
Delegates, 26, 27, *59*–61
  demo, *59*
Delete * from Table SQL statement, 205
Derived classes
  caching special cases in, 361
  overriding methods in, 72–73
  and Template method, 345, 347, 352
Design patterns, 1
  C#, 8
  defining, 5–6
  description of, 3–4
  learning, 6–7
  and object-oriented approaches, 7–8
  studying, 7
Design Patterns: Elements of Reusable Software (Gamma, et al.), 4, 6, 7, 16, 133, 179, 192, 197, 198, 223, 283, 302
  and Context class, 323
  and State pattern, 321
  Template Method discussed in, 347
  on Templates, 351
  types of iterators described by, 290
  Visitor pattern discussed in, 362
Design Patterns Smalltalk Companion, The (Alpert, Brown, and Woolf), 4, 5, 7, 231, 251, 269, 302, 307
Diamonds, in UML diagrams, 84
Director class, 133
Display classes, using Bridge pattern, 169, *170*
Division operator (/), 18
Dot (.), and Convert methods, 13
Double angle brackets, 84
Double dispatching, and Visitor pattern, 361
Double quote (\"), 15
DoubleRect class(es), 72, *73*

double type, 14
Double variable, methods for, 29
Doubly linked lists, and Composite pattern, 187
do-while loop, 24
draw code, in base Triangle class, 351
drawCommand class, draw method in, 266
DrawData objects, 265
drawHandle method, 333
Drawing, in C#, 66–67
Drawing interface
  circle and rectangle objects implementing, 332
  VisRectangle implementing, 333
Drawing program, for illustrating State pattern, 321, *322,* 323–325
drawLine method, 352
drawList collection, 265
DrawMemento class, 332
draw method, 77, 349, 350, 352
  in drawCommand class, 266
  as empty method, 79
  for Rectangle class, 68
  and Visitor class, 354
DrawObject, three subclasses of, *353*
draw routine, and Triangle class, 348
draw2ndLine method, 348, 349, 350, 352

E
Editing, and ADO style of programming, 207
Equal signs (multiple), for initialization, 16
else clause, 19, 20
e-mail address book, 129
Employee, instances of, in Company, *84*
Employee class, 181–183, 188
  derived from AbstractEmployee class, *184*
  derived from Person class, 82–83
  visitor classes visiting, *359*
  and Visitor pattern, 355–356
Employee records, in databases, 200
Employee salaries
  sorted by magnitude, *201*
  sorted by name, *201*
Employee tree, building, 184–186
EmployeeVisitor, 361
EmpNode class, 185
Empty methods, base classes with, 79
Encapsulation, 7, 8, 47, 51, 71, 337
EndChain class, 250
End of file, testing for, 94–95
End-of-file function, 94
Enumerated types, 27
Enumeration class, 287
Enumeration interface, 9
Enumerator, 182
Enumerator property, of Hashtable, 213
Equal sign (=), for representing assignment of data, 17
Equities class, 132
Error checking, and language introduction, 281
Error handling
  in C#, 90
  encapsulation of, 51
ET++, 4
Event class, 107–108
EventHandler class, 194
Event handling in Decorator, 193–194
  control size and position, 194
  layout considerations, 194
Events button, 66
Events classes, 109
Examplar pattern, 141
Examples
  FileExit class, 258
  Flyweight, 224–230
  simple report, 270–271
Exception classes, in C#, *91*
Exception errors, 13
Exception object, 94

Exceptions, *9*
 in C#, 90–91
 in file handling, 94
 and Singleton pattern, 124
Exceptions class, 31
Execute command, and Butterfly class, 104
Execute method, 259, 260, 280, 295, 328
Execute() method, 256
ExecuteNonQuery method, 205
ExitCommand, Command interface implemented by, *84*
ExitMenu class, 257
External iterators, internal iterators *versus,* 290
Extrinsic data, 223, 224, 226

F
Façade classes
 building, 207–209
 building price query, 207–208
Façade pattern, 155, 199–221, 232, 238
 ADO.NET database connections in, *9*
 and building Façade classes, 207–209
 building Price table, 215–218
 consequences of, 220
 creating classes for each table with, 213–215
 and Decorator pattern, 198
 elements in constitution of, 220
 final application with, 219
 grocery program using, *209*
 loading database tables, 218–219
 and making ADO.Net Façade, 209–213
 program on the CD-ROM, 221
Factory class, 100, 224
Factory method pattern, 97, 107–113
 Events classes, 109–110
 and other factories, 112–113
 Prototype pattern combined with, 152
 StraightSeeding, 110–111
 Swimmer class, 109
 variations on, 113
 when to use, 113
Factory pattern(s), 8, 103–105, 129, 154
Fahrenheit temperature scale, 43, 44
false reserved word, 14, 17
Fast Fourier Transform (FFT), 103
FFT "butterfly," 103
File compression, with different algorithms, 337
FileExit class, 258
File handling
 in C#, 92–94
 exceptions in, 94
 file object, 92–93
 reading to text file, 93
 writing to text file, 93–94
FileList box, 246
FileList class, 248
File object, 92–93
Files, opening for reading, 96
Fill button, 321
 behavior of, 323, 324
FillDataSet method, 204
FillState class, 325, 330–331, 336
Fill State object, 330–331
Filtered enumeration, classes used in, *288*
Filtered iterators, 286–287
FilteredIterator class, 286–287
Filtered Iterator demo, simple program-illustrated filtered
 enumeration, *288*
FinalImage class, 235
FirstFirst class, 100
First normal form (1NF), tables in, 200
fixText method, 60, 61
Floating point numbers, 13
Float variable, methods for, 29
FlyCanvas class, 228
Flyweight demo, *224*
Flyweight display, with one folder selected, *225*

FlyweightFactor class, 224
Flyweight pattern, 8, 155, 223–232, 238
  copy-on write objects, 232
  discussion about flyweights, 224
  example code, 224–230
  flyweight uses in C#, 231
  handling the mouse and paint events, 230
  program on the CD-ROM, 232
  sharable objects, 231–232
Flyweights, generation of, *228*
Folder class, 226, 228
FolderFactory class, 225, 227, 228
Folders
  dynamic selection of and FolderFactory, 227
  Flyweight demo, *224*
  Flyweight example and selection of, 229–230
  selection of one in Flyweight display, *225*
Food class, derived from DBTable, 213
FoodKey, Foods table with, 207
FoodName, Foods table with, 207
Foods table, 207, 214, 218
foreach looping construct, 24, 88
For loop, 24–25
For loop statements, commas in, 25–26
Format conversion, classes used for, 48–51
Format method, 29
Form class, 43, 65, 66, 256
Form Designer, 258
Form feed (\f), 15
Form initialization method, 194, 295
Form_Load routine, 247
Form object, 259
Form1 class, simple hello window in, 65
Forms, menus added to, *39*
Form window, 68, 69
Fortran, 19
Fowler, M., 81
FoxBase, 202
Friend construction (C++), 302
ftx delegate, 60

G
Gamma, Eric, 4
"Gang of Four" (GoF), 4, 277
Garbage collection, and managed languages, 30
Garden class, 116
Gardener program
  major objects in, *117*
  user interface of, *117*
Garden interface, 116
GardenMaker factory, 115–120
  and handling RadioButton and Button events, 119–120
  PictureBox, 118–119
Garden object, methods in, 118
GardenPic class. from PictureBox, 118
Gestalt, 3
getBuilder method, 133
getChain method, 244
getChild method, 186, 188
getChild operation, 180
getClass method, 100
getConvTemp method, 47
getEnumerator method, 289
getFrname method, 102
GetHashCode property, 309
getIterator method, 285
getJob method, 82, 83
getLname method, 102
Get property, and Rectangle class, 306
getSalaries() method, 181, 184
getSeeding method, 109
getSelected method, 136
getSpooler method, 123, 124
getSubordinates method, 283
getSwimmers method, 111
getVacationDays method, 361
getVacDays function, 356

getValue method, 215
getWindow method, with Investment Tracker, 130
gif files, and Buttons, 37
Global points of access, to Singletons, 126
"God class," Mediator as, 298, 335
GoF patterns, 99
Grand, 81
Graphical editor, state of, 321
Graphics, in C#, 66–67
Graphics object, 68
Greater than operator ((>), 21
Greater than or equal to operator (>=), 21
Green radio button, in watching colors change program, 314
Grid adapter, 166
GridAdapter class, 161
Grocery Database, Price table in, *216*
Grocery pricing data, 207, *208*
Grocery program, using Façade pattern, *209*
Group class, derived from Address class, 129
GUI, 4
Guillamets, 84

H

Hashtable, 89, 290, 309, 327
  of color objects, 245
  kids clubs in, 288–289
  using, 213
  in watching colors change program, 316
Hello form, 65
Help chain, simple demonstration, *248*
Help command, receiving, 250–251
Help system, programming, 248–251
HiTextBox, instance of on Windows Designer pane of new form, *74*
HiTextBox control, 74
  new derived and regular TextBox control, *75*
HitTest object, 162
"Hollywood Principle," and Templates, 351
Hook methods, 347, 352
Hysteresis, 263

I

IBM DB/2, 202
ICloneable interface, 149
  implemented by SwimData class, 146
IComparable interface, 147, 173
Icon class, 235
Icon files, and Buttons, 37
Icons
  adding to toolbox, 257
  and Flyweight, 223
  Flyweight pattern, 224, 225
IDE. *See* Integrated development environment
IdictionaryEnumerator, 289
IEnumerator, 181
  interface, 284
if clause, 19
if-else case statements, 321
if statement, 19
If tests, 322
if-then-else, 19
Illegal filenames, exceptions for, 94
ImageChain class, 245
Image class, 235
Image control, 234
Image proxy, 234
ImageProxy class, 234
Imager class, 248
Imager interface, 234, 235
Imager object, 234
Images, on Buttons, 37
Implementing the interface, 76
Implementor
  in Bridge pattern, 169
  VisList interface defines interface of, 173
#include directive, in C and C++, 16
Increment operator (++), 18, 19
Indenting, 19, 20

Indexers, 26, 27, 61–62
  in databases, 199
  demo of bits gotten from number with, *62*
Inheritance, 1, 7, 8, 27, 65–80, 74, 80
  abstract classes, 76–78
  and adapters, 157
  constructors, 65–66
  drawing and graphics in C#, 66–67
  interfaces, 76
  interfaces and abstract classes compared, 79–80
  namespaces, 68–71
  object, *5*
  and overloading, 71–72
  overriding methods in derived classes, 72–73
  overriding Windows controls, 74–75
  relationships in Builder pattern, *138*
  replacing methods using new keyword, 74
  representation of, in UML diagram, 83
  and UML diagrams, 82–83
  using, 68
  virtual and override keywords, 72
Initialization, 56
  and constructor methods, 65
  multiple equal signs for, 16
InitializeComponent method, 35, 66, 67
init method, 66, 112
Instances, 43, 63
  of classes, 53
  and Singleton pattern, 124
Integers, methods for, 29
Integrated development environment, class module created from, 45
Interaction, between objects, 5
Interface manipulation, data manipulation separated from, 45
Interfaces, 1, 7, 76, 179
  abstract classes compared with, 79–80
  dissimilar classes with same, 150–152
  For Iterator, 283
  and Observers, 314
  and UML diagrams, 84
Interleaved Commands, 263
Internal iterators, external iterators *versus,* 290
Internalization, and learning design patterns, 6
internal keyword, 302
Internet Explorer, 191
InterpChain class, 279
Interpreter pattern, 8, 239, 269–282
  applicability of, 269–270
  and Chain of Responsibility pattern, 253
  consequences of, 281–282
  implementing, 277–280
  interpreting the language, 271–272
  motivation for, 269
  names for participating objects in, 277
  objects used in parsing, 272–276
  operation of on simple command in text field, *281*
  program on the CD-ROM, 282
  reducing the parsed stack, 276–277
  simple parsing hierarchy for, *273*
  simple report example, 270–271
  and syntax tree, 278–280
Intrinsic data, 223, 232
int type, 14
Investment Tracker, 130–135
  CheckChoice class, 133–134
  ListboxChoice class, 134–135
  StockFactory class, 132–133
is equal to operator (==), 21
isLegal, 275
is not equal to operator (!=), 21
Isosceles triangle, drawing, 349–350, *351*
isUndo method, 265
Items collection, 38, 159
  methods for, *39*
  using in ListBox control, 135–136
Iterator, fetching, 285–286

Iterator code, sample, 284–285
Iterator interface, and KidIterator class, 285
Iterator pattern, 9, 239, 283–290
  consequences of, 289–290
  and filtered iterators, 286–288
  motivation, 283–284
  programs on the CD-ROM, 290
  sample iterator code, 284–286
  and tracking clubs, 288–289

J
Jacobson, Ivar, 81
Java, 11, 302
  arrays in line with style used in, 87
  C# roots in, 29
  differences between C# and, 27
  Hashtable, 213
Java Design Patterns: A Tutorial, 4
JPEG files, and Buttons, 37

K
Kelvin temperature scale, converting to, 47
Keyboard listeners, 250
KeyDown event, 251
KeyDown event listener, 250
Keys
  and Hashtable, 213
  to Hashtable of color objects, 245
  for Price table, 215
Keywords, 43
KidClub class, 287
KidData class, 284, 287
KidIterator class, 285
KidList class, 286, 294
Kid objects, 284, 285
Kids class, 285

L
Label control, properties for, *36*
Labels, 35, 40
Languages
  applicability of, 270
  and Interpreter pattern, 269
  interpreting, 271–272
Language symbols, parsing into tokens, 271
LastFirst class, 101
Layout, in Decorator, 194
Layout managers, 27
Leaves, 179, 180, 185, 189
Left shift *n* places (<< *n*), 18
length property, for arrays, 87
Less than operator (<), 21
Less than or equal to operator (<=), 21
Licensing fees, and embedding of commercial languages, 269
LineButton command Button class, 339–340
Line graphs, 338
  plotting, 345
  and Strategy pattern, *343*
LineGraph strategy, 344
LinePlot class, 345
Line plots, making, 342
LinePlotStrategy class, 340, 341
LinePlot window, 341
List adapter, *158,* 159
  class adapter approach to, *165*
ListboxChoice class, 134–135
ListBox class, *59,* 76, *159,* 169
  KidList class derived from, 294
  MyList derived from, 164–165
  properties, *38*
ListBox control
  Items collection in, 135–136
  plotting data with, 136
ListBoxes, 38, *58*
List boxes
  automatic language generation with, 281
  food prices loaded into, 218
  kidclub, 289
Listbox pattern, 9
ListBox type, 166

ListBridge class, 171, 173
Listeners, 250
ListForm observer, 317
List interface, stocks with, *131*
List observer, *317*
ListObs window, 315
Lists, data moved between, 157–158
Logical AND operator (&&), 21
Logical Not operator (~), 21
Logical operators, 17, *18*
Logical Or operator (__), 21
Looping, through arrays, 87
Looping statements, 24
Lowercase, C# syntax in, 11
Lower class, 60

M
Macro language, 269
Macro record button, 281
main function, 17
MainMenu control, 257
  adding to window form, *39*
Managed languages, and garbage collection, 30
Manager class, 360, 361
ManagerVisitor, 361
Math computation, factory patterns in, 103–105
Mathematica, 270
Mediator
  and command buttons management, 321
  and handling of mouse and paint events, 310
  interaction between buttons and, *330*
  StateManager and, 326–328, *329*
Mediator class, 294, 296–297, 306, 326
  interactions between classes simplified with, *293*
Mediator object, interactions between Command objects and, *198*
Mediator pattern, 8, 239, 291–299, 311
  consequences of, 298–299
  example system, 291–292
  and initialization of system, 297
  interactions between controls, 292–293
  looser coupling between classes with, 291
  mediators and command objects, 297
  program on the CD-ROM, 299
  sample code, 294–297
  simple program with two lists, two buttons, and text field, *292*
  and single interface mediators, 299
Mediators, and “god class,” 298, 335
Memento class, 306, 321
Memento interface, and handling undo list, 331
Memento objects, 302, 307, 308
Memento pattern, 8, 239, 301–312, 327, 331
  cautionary note about sample code with, 308–309
  and command objects in user interface, 309–310
  consequences of, 311
  implementation, 302
  motivation, 301–302
  mouse and paint events handled with, 310
  program on the CD-ROM, 312
  purpose of, 301
  rectangles drawing pattern demo, 302, *303, 304*
  roles for objects defined by, 302
  sample code, 302–309
  UML diagram for drawing program with, *311*
MenuBar, 257
MenuItem class, 257, 260
Menu items, 255, 256
Menus adding to form, *39*
Metafile class, 235

Methods, 43, 45
  accessor, 45
  ArrayList, 88, *89*
  Concrete, 347
  with Garden object, 118
  Hook, 347, 352
  implementations of and interfaces, 76
  for Items Collection, *39*
  for objects, 29
  overriding in derived classes, 72–73
  polymorphic, 72
  replacing using new, 74
  Template, 347
  in Template class, 347
  virtual, 347
MicroGold, 86
Microsoft Access, 202
Microsoft Excel, 313
Microsoft Office products, VBA in, 269
Microsoft SQL Server, 202
Microsoft Works, 202
Model-View-Controller framework, 3, *4*
Modules, class, 45
Modulo operator (%), 18
Mouse, handling in C#, 230
Mouse click events, 323
MouseDown event, 162
  and State object, 323, 324
Mouse drag events, 323
MouseEventHandler, 162, 193
MouseMove, 192, 230, 252
MouseUp event, 322
move method, 333
moveNext() method, 287
MultiChoice interface, 132
  with Investment Tracker, 130
Multiple exceptions, 91–92
Multiple variables, declaring, 14
Multiplication operator (*), 18
MultVar class, 276
MultVar object, 280
MyList class, derived from ListBox class, 164–165

N
NameFactory, 102
Namer class, 100, 101
Namer Factory program, *102*
Namer program, execution of, *103*
Namespaces, 30–31, 68–70
Narrower data types, and wider data types, 13
.NET
  composites in, 189
  Visual control, 176
Netscape Navigator, 191
new keyword, methods replaced using, 74
Newline (line feed) character (\n), 15
new operator, 97
New Project selection window, *31*
nextToken method, 52
nextTop method, 271
NoCmd Chain element, 246
NoCmd class, 248
Node object class, 189
Nodes, 179, 180, 185, 189
NonTerminal Expression, 280
NonTerminalExpression object, in Interpreter pattern, 277
Nonvisual decorators, 197
Normal form, 200
Notification messages, and Observer pattern, 318
NOT operation, 17
Null character (\0), 15
Null exceptions, 94
null reserved word, 14
Numbers, converting between strings and, 13–14
Numerical types, conversion from, 29
Numeric constants, 14

O
Object adapters, 157
  class adapters *versus,* 166
Object Database Connectivity, 202–203
Object-oriented approaches, notes on, 7–8
Object-oriented languages, and signatures, 71
Object-oriented programming, 1, 8
  and design patterns, 4
  difference between procedural programming and, 53
  factory concept in, 107
  template patterns in, 352
Object patterns, class patterns *versus,* 155
Objects, 1, 43
  in C#, 29
  and Chain of Responsibility pattern, 242
  classes as, 53–54
  composition of, 8, 157
  and creational patterns, 97
  interaction between, 5
  saving state of, 302
  used in parsing, 272–276
Objects class, 31
object type, objects fetched from ArrayList and, 88
Observer interface, 315
Observer pattern, 239, 242, 299, 313–319
  candidate for, *317*
  consequences of, 318–319
  data displayed as list and in graphical mode, *313*
  and message to the media, 318
  program on the CD-ROM, 319
  simple control panel to create red, green, or blue "data," *314*
  UML diagram of observer interface and subject interface implementation of, *317*
  and watching colors change, 314–317
Observers
  notification messages sent to, 318
  and Observer pattern, 314
  update methods to, 319
  in watching colors change program, 315, 316, 317
ODBC. *See* Object Database Connectivity
OleDbAdapter, 220
OleDbCommandBuilder, 220
OleDbCommand class, 203
OleDbCommand object, 205
OleDbConnection, 220
OleDbConnection object, 203
OleDbDataAdapter, 203
One's complement (~), 18
OneSexSwimData class, displaying with one sex on the right, *148*
OpenCommand class, 261
OpenForRead statement, 95
OpenForWrite() method, 96
OpenForWrite statement, 95
OpenMenu class, 257
open method, 204
Operator overloading, 27, 62
Oracle, 202
Ordering components, and Composite pattern, 189
Organizational chart, typical, *180*
Origin, 270
Originator class, 311
Originator object, and Memento pattern, 302
OR operation, 17
Output, 270
Overloading, 71–72, 80
override keyword, 72

P
Paint Event, selecting for PictureBox window, *67*
PaintEventArgs object, 67
PaintEventHandler, 194

Paint event handler, 69, 78
Paint events, 66, 118, 193, 310
  and Decorator interface, 192
  handling in C#, 230
  in watching colors change program, 315
Paint handler, 266
Paint method, 194, 342
Paint routine, bar plot drawn in, 341
Panels, 131, 134, 136, 192
Parentheses ()
  around conditions, 19
  data type names enclosed in, 13
  and switch statement, 22
Parsed stacks, reducing, 276–277
ParseObject class, 272
ParseObjects, 281
Parser class, 274–275, 277, 280
ParseVar class, 275
ParseVar object, 273
ParseVerb class, 275
ParseVerb object, 273
Parsing, 49
  interaction of classes performing, *278*
  objects used in, 272–276
Parsing hierarchy, for Interpreter pattern, *273*
Pascal, 19
Pattern groupings, 311
Pattern Hatching (Vlissides), 361
"Pattern mining," 6
Peek method, 95
Pen object, 67, 77
Pens, and Paint method, 194
PerennialGarden class, 117
Person, instances of in Company, *84*
Person class, 81
  derived from Address class, 129
  and inheritance, 82–83
  with private, protected, and public variables, and methods, *82*
  UML diagram for with/without method types, *82*
pic_Paint event handler, 347
picPaint handler, 230
PictureBox
  control, 316
  inserting on a Form, *66*
Picturebox level, mouse motion checked at, 229
PictureBox window, Paint Event selected for, *67*
Pie charts, 338
Placeholder classes, and template classes, 352
Plant object, 116
Plot click method, 136
PlotPanel class, 347
Plots, drawing in C#, 341
PlotStrategy class, 344
  instances of, *338*
PlotWindow class, 345
Pluggable adapters, 166
Plus sign (+), to combine strings, 17
Pointers, 26, 62
Polymorphic methods, 72
pop method, 271
Positioner class, 227
Prefix characters, and numeric constants, 14
Prefixes, 58
PrelimEvent class, 108, *109*, 110, 112
Price class, 208, 213, 218
PriceKey, Price table with, 207
Price query, building, 207–208
Price table, 207, 218
  building, 215–218
  in Grocery Database, *216*
PRINT, 273
Print object, 273
PrintSpooler class, writing skeleton of, 125
Private methods, 48, 53, 63, 71
Privileged access
  and Iterator pattern, 290
  and Memento pattern, 301
Process button, 60

Product class, 170
ProductList class, 173
Product objects, 173, 174, 175
ProductTable class, 173
ProductTable version, of VisList, 172
Programming, object-oriented, 1
Programming style, in C#, 58
Programs
  on CD-ROM, 41, 63, 80, 106, 113, 121, 127, 139, 154, 167, 177, 190, 198, 221, 232, 237, 253, 268, 282, 299, 312, 319, 336, 344, 352, 363
  interpreters introduced into, 281
  Mediator and changes in behavior of, 298
  Seeding, 111–112
  simple C#, 16–17
  simplest Window, in C#, 32–34
  simple temperature conversion, 43–45
  triangle drawing, 350–351
  undoable, 264
  watching colors change, 314–317
  Windows Controls, 40–41
Properties, classes and, 56–58
Properties window, 66
  MouseMove event selected from, *231*
Property methods, 56, 58, 63
protected keyword, 71
Protected methods, 71
Protected variables, 71
Prototype, re-sort of left list box shown with, *146*
Prototype managers, 152–153
Prototype pattern, 98, 141–154, 154
  and cloning in C#, 142
  and cloning the class, 146–147
  consequences of, 153
  and prototype managers, 152–153
  using, 142–145, 147–152
Prototype program
  simple, *145*
  sorted results of, *145*
Prototype Registry, 153
Proxy image display, with image load shown, *236*
Proxy pattern, 155, 233–238
  comparison with related patterns, 237
  copy-on-write, 232, 237
  program on the CD-ROM, 237
  proxies in C#, 236
  sample code, 234–236
  use of, 233–234
Public methods, 48, 53, 63, 71
push method, 271

Q
Queries, executing in ADO.Net, 205
Queue object, 90
Queuing of commands, 257
QuickImage class, 235
Quit key, 248
Quotes ('), characters enclosed in, 14

R
Radio button events, handling, 119
Radio buttons (or option buttons), 37, 281
Reading
  opening file for, 96
  text files, 93
readLine method, 96
Recognition, and learning design patterns, 6
Rectangle
  drawing in PictureBox on form, 66
  square created from, 69–71
Rectangle button, 302, 307, 321
Rectangle class, 68, 77, 229, 306
  DoubleRect class derived from, 72, *73*
  instance of, 69
  Square class derived from, 69–70, *71*
Rectangle drawing program, *69*
Rectangle state object, 324

RectButton
  command class, 310
  Command objects for, 309
Rect button, behavior of, 323
rects property, 333
RectState object, 324
Red button, 260
RedButton class, 257
RedCommand Class, 265, 267
RedCommand object, 260, 265
Red menu item, 260
Red radio button, in watching colors change program, 314
References, to arrays, 88
Refined abstraction, in Bridge pattern, 169
Refresh method, 119, 192
registerInterest method, 314
Registry
  of Prototype classes, 153
  and Singleton instantiation, 126
Relational database, 270
remove method, 183, 188, 308
repaint method, 347
Report Generators, 270–271
Responsibilities, Decorator pattern and, 198
restore method, 308, 332
Right shift *n* places (>>*n*), 18
Rows, 199
  adding to database tables using ADO.Net, 206–207
  editing or deleting, 207
Row selection, detecting, 162
Rumbaugh, James, 81

S
Salaries computation, 180–181
SalaryType column, data in, 200
Scope, 16, 25
Scott, K., 81
Second normal form (2NF), tables in, 200
Seeding class, 108
Seeding diagram, Seeding interface and derived classes in, *109*
Select button, 321, 323
select case statements, 321
SelectedIndex property, 162
Selected instance, of Folder, 228, 229
SelectIndexChanged event, 295
Select method, and Mediator, 295
selectOne method, added to State interface, 324
Select statement, 204, 205
Select tests, 322
Self-promotion, and Composite pattern, 186–187
Semicolon (;)
  C# statements terminating with, 11
  at end of declaring draw method as abstract, 77
  statements terminating with, 17
Sender class, 248
sendNotify event, 315, 316
sendToChain method, 244, 248
Sequential operations, languages and specifying order of, 281
setBounds method, 341
setFill method, 333
setPenColor method, 344
setPlotBounds(), 341
Set property, and Rectangle class, 306
setSelected method, 333
setText method, 295
SexSwimData class, 148, 149, 151
Shallow copy, 147
Shape class, 77, 79
Shape objects, 78
Sharable objects, 231–232
Signatures, 71, 76
Simonyi, Charles, 58
SimpleComposite example, 188–189

Simple Factory pattern, 97, *99*
building, 101–103
sample code, 100
two derived classes, 100–101
workings of, 99–100
SimpleHello form, after clicking Say Hello button, *33*
Single interface Mediators, 299
Single quote (\'), 15
SingletonException:Exception class, 124
Singleton pattern, 8, 9, 98, 123–127, 154
consequences of, 126
and creating instance of the class, 125–126
and creating singleton using static method, 123–124
exceptions and instances, 124
global point of access provided to singletons, 126
and throwing the exception, 125
SlashDecorator, CoolButton decorated with, *196*
Smalltalk, 7, 284, 302
Model-View-Controller framework for, 3
Smalltalk Companion. See Design Patterns Smalltalk Companion
SORT, 273
SortBridge class, 173
sorted list generated using, *175*
SortedList class, 89–90
Sort object, 273
Special characters, representations of, *15*
Split method, of String class, 51
Spooler class, 123, 124
SQL. *See* Structured Query Language
SQL query, and ADO.NET Façade, 209, 211
SQL-92 standard, 202
Square, creating from rectangle, 69–71
Square class, deriving from Rectangle class, 69–70, *71*
Stack class, 271
Stack object, 90
Stack reduction, and syntax tree, 278
Stacks, 271
ParseObject pushed onto, 272
reducing during parsing, *276*
State, of graphical editor, 321
State interface, selectOne method added to, 324
StateManager, 326–328
and the Mediator, *329*
StateManager class, 323
current state tracked by, *324*
and switches between states, 325–326
State object, mouse activities handled by, 323, 324
State pattern, 8, 239, 309, 321–336
consequences of, 335–336
Fill state handled with, 330–331
and Mediator interaction with State Manager, 326–329
Mediators and "god class," 335
program on the CD-ROM, 336
sample code, 321–325
simple drawing program illustrating, *322*
and state transitions, 336
and Strategy pattern, 337
and switching between states, 325–326
undo list handled with, 331–333
using, 321
VisRectangle and VisCircle classes, 333–335
State transitions, and State pattern, 336
Static methods
in File object, 92
and global point of access, 126
singleton created with, 123–124
Static variable, 123
StockFactory class, 132–133
Stocks, with list interface, *131*
Stocks:Equities class, 132

Store class, derived from DBTable, 213
StoreFoodPrice objects, and Prices class, 215
StoreKey, Stores table with, 207
StoreName, Stores table with, 207
Stores table, 207, 214, 218
Straight seeding
  of 500 free, *112*
  of swimmers, 107
StraightSeeding class, 110, 111
StraightSeeding subclass, 108, *109*
Strategy pattern, 239, 337–344, 345
  consequences of, 344
  and Context class, 339
  and drawing plots in C#, 341–342
  line and bar graph strategies, 340
  motivation, 337–338
  program commands, 339–340
  program on the CD-ROM, 344
  sample code, 338
  simple panel to call different plots, *340*
  UML diagram of, *343*
StreamReader object, 93
StreamWriter object, 93
String class, Split method of, 51
string object, 31
Strings, 13–14, 27
StringTokenizer class, 51–52, 55, 56, 71–72, 142, 271, 274
struct types, 26
Structural patterns, 6, 97, 155–237
  Adapter, 157–167
  Bridge, 169–177
  Composite, 179–190
  Decorator, 191–198
  Façade, 199–221
  Flyweight, 223–232
  Proxy, 233–237
  summary of, 237
Structural programs, summary of, 237–238
Structured Query Language, 202, 270
Subclasses, 107, 129, 232
  and Abstract Factory pattern, 120
  of DrawObject, *353*
  of InterpChain, 280
  and interpret operation, 279
  and Prototype pattern, 148, 152, 153
  and Singleton, 126
  and Template pattern, 345
  and visitor classes, 360
Subject, and Observer pattern, 314
Subject interface, 314
Subjects, observers and abstract coupling to, 318
Subordinates, ArrayList, 188
subordinates method, 182
Subsystems, and Façade pattern, 199, 220
Subtraction operator (-), 18
Suffix characters, and numeric constants, 14
Superclasses, 107
SwimData class, 143, 146, 148
  UML diagram for, *152*
SwimInfo class, 151
Swimmer class, *53*, *54*, *55*, *56–58*, 109, 142
Swimmer object, 157, 159
Swimmers, straight seeding of, 107
SwimmerTimes display program, revised version of, *56*
switch statement, 22, 27
Sybase, 202
Symbols, in UML diagrams, 82
Syntax tree, and parsing of stack, 278–280
System.Collections namespace, 88
System.Drawing namespace, 68, 229
System namespaces, 17, 31, 68

T

Tab character (\t), 15
Tables, 199, 203, 231
  classes for in Façade pattern, 213–215
  deleting contents of, in ADO.Net, 205–206
  employee names and salary type, *200*

Temperature class
  building, 45–47
  decisions put into, 47–48
Temperature conversion program, simple, 43–45
Template class, methods in, 347
Template Method pattern, 239, 345–352
  consequences of, 352
  kinds of methods in Template class, 347
  motivation, 345–347
  programs on the CD-ROM, 352
  sample code, 347–350
  templates and callbacks and, 351–352
  Triangle drawing program, 350–351
Template methods, 347
Template pattern
  and abstract Triangle class, 348
  using, 345
Templates, 345
temp variable, 47
TerminalExpression, 280
TerminalExpression object, in Interpreter pattern, 277
Ternary operator, 23
Testing, for end of file, 94–95
Text, and changes in label, 40
TextBox class, 33, 36, 74
TextBox control, 74
  and new derived HiTextBox control, *75*
TextBoxes, 189
TextBox properties, *37*
Text comments, in UML diagrams, 85
Text data, and different line-breaking strategies, 338
Text files
  reading, 93
  writing, 93–94
Text property, 36
Third normal form (3NF), tables in, 200, 201
Third-party products, VBA embedded in, 269
Throwing exceptions, 92
throws keyword, 27
Throw statement, 92
TimedFinalEvent class, 108, *109,* 110, 112
Time entry interface, with parsing of symbols for Scratch, Disqualification, and No Time, *50*
TimerCallback class, 234
timerCall method, 234
Timer class, 234
Times class, 55
  simple parsing program using, *49*
TimeSwimData class, 151
Tokenizer class, 51–52
Tokenizer demo, *53*
Tokens
  parsing language symbols into, 271
  strings separated into, 51
ToolbarButtons, in C#, 309
Toolbars, 40, 189
Toolbox
  new control shown on, *74*
  TextBox selected from, 33
ToolTips, 39, *40*
top method, 271
ToString method, 132, 135
TPaint method, 351
Tree adapter, 166
TreeAdapter class, 163, 175
TreeDemo program, *164*
TreeList class, 40, 175
Tree list component, and Bridge, *176*
Tree list display, of composite with display of parent nodes, *188*
TreeView, 163–164, 189
TreeView control, corporate organization shown in, *186*
TreeView list, 185
TreeView pattern, 9
Triangle class, 347, 350
  draw code in, 351
  three subclasses of, *348*

Triangle drawing program, 350–351
TriangleForm class, 350
Triangles
 drawing, 349–350, *351*
 drawing isosceles, 349–350, *351*
 drawing standard, 349, *351*
TrigButterfly class, 105
true reserved word, 14, 17
try block, 125, 205
 and exceptions, 90
 file manipulation code enclosed in, 94
Two-way adapters, 166
Type checking, and registry approach, 126

U
UML. *See* Unified Modeling Language
UML diagrams, 1, 81–86
 and annotation, 85–86
 for Bridge pattern, *173*
 of Chain of Responsibility program class structure, *247*
 of CommandHolder approach, *262*
 and composition, 84–85
 for Decorators, *197*
 for drawing program using Memento, *311*
 and inheritance, 82–83
 of inheritance relationships in Builder pattern, *138*
 and interfaces, 84
 of observer interface and subject interface implementation of Observer pattern, *317*
 showing Employee derived from Person, *83*
 of Strategy pattern, *343*
 for various SwimData classes, *152*
 WithClass, 86
UndoButton, 302
 Command objects for, 309
 and rectangles drawing program with Memento, 302, 304
Undo command demo
 program drawing red and blue lines, *263*
 program drawing red and blue lines after Undo button clicked, *264*
UndoCommand object, 265
Undo commands, 301
Undo function, with Command patterns, 262–267
unDo list
 and Caretaker class, 307, 308
 handling, 331–333
unDo method, 265
Undo program, classes implementing in Command pattern implementation, *267*
Undo requests, 257
Unicode, 12
Unified Modeling Language, 81
Unreasonable values, handling, 51
Unsafe blocks, 26
Unsafe mode, 27
Unsafe sections of C# code, pointers used within, 62
Unselected instance, of Folder, 228, 229
User commands, command interpreter for parsing, 270
User interfaces, command objects in, 309–310
using statement, 16

V
Vacation data
 for bosses, 359
 for employees, 356–359
Vacations button, 359
VacationVisitor class, 356
Value conversion, classes used for, 48–51
value keyword, 56
Value method, 289

Variables, 11, 15–16
 declaring as public, private, and protected, 71
 declaring as they are used, 15–16
 delegate, 59
 protected, 71
 static, 123
VAR type, 273
VarVarParse class, 280
VBA. *See* Visual Basic for Applications
VB.NET
 and C#, 29
 as managed language, 30
VB7, 30
VeggieGarden class, 116
VERB type, 273
Video data capture, and compression schemes, 338
View, 4
virtual keyword, 72
Virtual methods, 347
VisCircle class, 333, 335
Visiting, meaning of, 354
visit method, 354, 358, 360
Visitor class, 354, 358, 360
 triangle classes visited by, *354*
Visitor object, 355
Visitor pattern, 239, 353–363
 and Boss visits, 359
 catch-all operations with visitors, 360–361
 consequences of, 362
 and double dispatching, 361
 motivation, 353–355
 program on the CD-ROM, 363
 sample code, 355–356
 times to use, 355
 traversing series of classes and, 362
 using, 353
 vacation visits demo, *360*
 visit and accept methods interactions, *354*
 visiting several classes, 357–358
 visiting the classes, 357
Visitors, catch-all operations with, 360–361
VisList interface, 173, 175, 176
VisList variable, 171
VisRectangle class, 304, 305–306, 333–335
 Drawing interface implemented by, 333
 VisCircle inheriting from, 333, 335
VisRectangle instance, 306
Visual Basic, 19
 C# roots in, 29
 major differences between C# and, 11
Visual Basic Design Patterns: VB6 and VB.NET, 4
Visual Basic for Applications, 269
Visual Basic.NET, 11
Visual command interpreter program sample, Chain of Responsibility pattern, *243*
Visual controls, interactions between, 292–293
Visual Studio IDE, and mouse move event handler, 230
Visual Studio.NET, 20, 31, 32, 86, 203
 decorating inner class in, 197
 grid table with, 160
 Integrated Development Environment, 65
 namespaces, 68
 screen from, showing properties interface, 176, *177*
 visual builder in, 44
Vlissides, J., 361

W

WealthBuilder class, 135
WealthBuilder program, with list of equities, list box, check
 boxes, and plot panel, *137*
Web sites, for learning design patterns, 7
While loop, 24
White space characters, 14
 representations of, *15*

Wider data types, and narrower data types, 13
Windows
  and data display, 313
  and ODBC, 203
Windows controls, 35–40, 80
  Buttons, 37
  CheckBox, 36–37
  Items collection, 38–39
  labels, 35
  ListBoxes and ComboBoxes, 38
  menus, 39
  overriding, 74–75
  radio buttons, 37
  selection of basic, *35*
  TextBox, 36
  ToolTips, 39–40
Windows Controls program, 40–41
Windows Designer, 74
  in Visual Studio.NET, *32*
Windows Form class, 35
Windows Form Designer, 230
Windows.Forms library, 33
WithClass UML diagrams, 86
writeLine function, 17
writeLine method, 96
Writing text files, 93–94
Writing Windows C# programs, 29–41
  building a C# application, 31–32
  classes and namespaces in C#, 30–31
  managed languages and garbage collection, 30
  objects in C#, 29
  simplest Window program in C#, 32–34
  Windows controls, 35–40
  Windows controls program, 40–41

X
XFactory class, 100
XML, and ADO.NET, 203