

Index

Symbols

() (parentheses)

- after method's name, 110
- Intellisense and, 110
- operator, 79–80
 - forcing precedence with, 79–80, 149, 150
 - left-associative, 80
 - property methods and, 239

{ } (curly braces), 137, 142–144

- K&R style, 143–144
- location of, 143–144
- proper usage of, 143
- statement blocks in, 137, 142–144

& (ampersand)

- && (logical AND operator), 147
- hot keys and, 43

* (asterisk)

- *= (assignment operator), 108
- multiplication operator, 63, 76
 - division operator v., 144, 145
 - precedence, 78, 79, **149**

@ (at sign), verbatim string literal character, 122–123

: (colon)

- operator, 236, 443–444. *See also* inheritance separator, 220

, (comma operator), precedence, 149

. (dot operator), 28, 29, 30, 47, 60, 110, 114, 127–128

- precedence, 149
- sliding down object chain, 127–128

“ (double quotation mark)

- escape sequence, 122
- string literals specified with, 100, 121

= (equals sign)

- != (relational operator), 136, 149
- == (relational operator), 136, 139, 149
- <= (relational operator), 136, 149
- >= (relational operator), 136, 149
- assignment operator, **27**

binary, 77

- object's state and, 30
- precedence, 78, 79, **149**
- right-associative, 79

! (exclamation point)

- != not equal (relational operator), 136, 149
- logical NOT operator, 147, 148
- unary operator, 77

/ (forward slash)

- /= (assignment operator), 108
- /* and */ , multi-line comments between, 94
- //, comments after, 93–94
- division operator, 63, 76
 - multiplication operator v., 144, 145

\ (backslash)

- escape character, 121
- escape sequences, 122
 - \\, \', \\", \0, \a, \b, \f, \n, \r, \t, \v

> (greater than), 136

- precedence, 149

>= (greater than or equal), 136

- precedence, 149

< (less than), 136

- precedence, 149

<= (less than or equal), 136

- precedence, 149

- (minus sign)

- (assignment operator), 108
- (decrement operator), precedence, 149
- Locals window, 91
- private (access specifier), 220
- subtraction operator, 63, 76
 - precedence, 78, 79, **149**

% (percent sign)

- %= (assignment operator), 108
- modulo divide operator, **63**, 76, 139
 - precedence, 78, 79, **149**

+ (plus sign)

- += (assignment operator), 108
- ++ (increment operator), 161, 182
 - precedence, 149

+ (plus sign) (continued)

+ (plus sign) (continued)

addition operator, 63, 76
precedence, 78, 79, **149**
concatenation operator, 63
Locals window, 91
`public` (access specifier), 220

; (semicolon), statement termination, **77–78, 86**

' (single quotation mark), escape sequence, **122**

_ (underscore character), **59**

|| (logical OR), **147, 148**

? : (ternary operator), **77, 141–142**

if-else statement and, 141–142
logic flow, 142
precedence, 149
syntax, 141

3D purple box, 109

A

\a (escape sequence), **122**

About box, 394

abstract classes, 459–460

abstract keyword, 460

access specifiers, 219–220

default, 220
for methods, design intent and, 222–223
`private`, 219–221, 447
protected, 447, 451
`public`, 220, 447
scope and, 220–221

aces (in card games), 247, 255, 256, 257, 270

acey-deucey, 270

action names, for API methods, 223

actions, 20. See also methods

ActiveX Data Objects, 401

Add New Item dialog box, 38

Add New Table button, 409–410

Add Reference dialog box, 37

addition operator (+), 63, 76

precedence, 78, 79, **149**

Add() method, 161, 187, 202, 250

address book program (arrays), 176–177

ADO.NET, 401–402

ADOX reference, 406

aggregates (SQL), 404–405

alarm (escape sequence), 122

algorithms, 31

for programming. See Five Program Steps

Quicksort. See Quicksort

shuffle, 255–256

sorting, 300

squaring, 162–164

alignment of fonts/columns, 160–161

Alt+D keystroke, 43

AlternatingRowsDefaultCellStyle, 431–432

Alt+X keystroke, 43

American Standard Code for Information

character set. See ASCII character set

ampersand (&)

`&&` (logical AND operator), 147

hot keys and, 43

AND (&& logical operator), 147

ant, 96, 115

API (application programming interface), 222

class, **222**, 231–232, 269

method names, 223–224

action, 223

implementation-neutral, 224

'append to,' 63. See also concatenation operator

AppendAllText(), 337

AppendText(), 336, 338

application programming interface. See API

Application.Run(), 39, 128

architecture

client-server, 257

three-tiered, 258

two-tiered, 257

arguments, 60. See also method arguments

arithmetic operators. See math operators

array(s), 175–204

address book program, 176–177

ArrayList objects, 200–203

details about, 177–178

dimensions of, 189–190

dynamic, 200–203

elements, **177–178**

indexes, **178**

highest, 178

initializing, 193–197

jagged, 197, 203

Length property, 178, 187

loop's relationship with, 176–177

multidimensional, 189–193

N-1 Rule, 178, 192, 224, 247, 362

as objects, 187–189

one-dimensional, 189

ragged, 197

- read-only, 221
 - static, **200**
 - syntax, 175
 - System.Array class, 187–189
 - undimensioned, 203
 - weekDays, 195–197
 - zero indexed, 194
 - Array.BinarySearch (System), 188**
 - Array.Clear (System), 188**
 - Array.Copy (System), 188**
 - Array.IndexOf (System), 188**
 - Array.LastIndexOf (System), 188**
 - ArrayLists, 200–203**
 - generics v., 306–307
 - arrayName, 175**
 - Array.Reverse (System), 188**
 - Array.Sort (System), 188**
 - as operator, 325**
 - ASCII (American Standard Code for Information) character set, 102, 181**
 - Unicode v., 102–103
 - assembler program, 10, 23**
 - assembly language, 10, 82. See also lvalues; rvalues**
 - asset, present value of, 298**
 - assignment operators (shorthand), 108–109, 149. See also equals sign**
 - %=, 108, 149
 - *=, 108, 149
 - +=, 108, 149
 - =, 108, 149
 - /=, 108, 149
 - list, 108, 149
 - precedence, **149**
 - associativity rules, 79, 149–150. See also precedence**
 - asterisk (*)**
 - *= (assignment operator), 108
 - multiplication operator, 63, 76
 - division operator v., 144, 145
 - precedence, 78, 79, **149**
 - at sign (@), verbatim string literal character, 122–123**
 - Attributes property, 338**
 - AutoSize property, 13, 14, 15**
 - AutoSizeColumnMode, 431**
 - auto-sizing (of labels), 13**
 - AvailableFreeSpace, 330**
 - AVG (aggregate), 404**
 - avionics, 320, 321**
 - avoidance/reduction of errors, 93–96, 273–281**
- ## B
- \b (escape sequence), 122**
 - BackColor box, 432**
 - backslash (\)**
 - escape character, 121
 - escape sequences, 122
 - \", \', \\, \0, \a, \b, \f, \n, \r, \t, \v
 - backspace (escape sequence), 122**
 - bad loops, 154. See also well-behaved loops**
 - bar graph, 204**
 - base classes, 446**
 - derived classes v., 459
 - base keyword, 459**
 - base-ten numbering system, 54**
 - base-two numbering system, 54–55**
 - batch programs, 32. See also display step**
 - BDUF (Big Design Up Front), 254**
 - beginning execution, of program, 128–129**
 - beginning of file (BOF), 331**
 - Behavior attribute, 432**
 - Big Design Up Front (BDUF), 254**
 - big-o notation, 305**
 - binary digits, 54. See also bits**
 - binary files, 339–341**
 - length of string in, 378–379
 - binary numbers, 54–55**
 - binary operators, 76**
 - assignment operator, 77
 - math operators, 76
 - birthday cards. See Cards**
 - bits, 54**
 - on/off states, 54, 55
 - sign, 55–56
 - block scope, 210–211**
 - blood samples, 197**
 - 'blue square' icon, 154**
 - blue squiggly lines, 68, 69, 70**
 - BOF (beginning of file), 351**
 - Booch, Grady, 219**
 - bool (data type), 70–72, 136, 230, 231, 364**
 - correct usage, 72
 - true/false states, 71, 72, 136
 - bool keyword, 71**
 - Boolean data type. See bool**

Boolean variables

Boolean variables, 71

flag, 60, 71

borders. See label borders

bound controls, 423–426

boundary conditions, 289

box icon, 109

boxing, 307

Boyce, Ray, 402

braces. See curly braces

break statements, 151, 170–171, 172, 177

breaking code, 63

breakpoints, 88, 291. See also single-stepping

backing up from, 294

purpose of, 295

setting (F9 key), 88–89, 130, 140, 164, 291

variables examined with, 89–91

bricks, 352

btnCalc() click event code, 58–59

btnCalc_Click() method, 61, 68, 71, 90, 92, 137–138, 192, 217, 435

btnDisplayOutput click event method, code, 44

analysis, 46–48

critique, 48

btnRefresh_Click() method, 133

Bubble Sort, 305

Bucket Analogy, 85–86

lvalues in, 85–86, 102

reference data types in, 102, 104, 106

rvalues in, 85–86, 102

TryParse() method and, 138

value data types in, 85, 86, 106

buffer, 40, 46

bugs, 86, 272–273. See also debugging; errors; exceptions

logic errors, 87, 272–273

syntax errors, 68–70, 80, 86, 272

building blocks

of computer decision making (if statement), 139, 152

of programming languages, 76–80

byte offset, 353

bytes (data type), 54

range of values, 54, 55

signed, 56

C

C (programming language), 4

The C Programming Language (Kernighan & Ritchie), 143

C# (programming language)

benefits, 5

case-sensitive, 39, 128, 239

data types. *See* data types

design of, OOP and, 116

Express. *See* C# Express

format-insensitive, 142, 143

IDE. *See* Visual Studio IDE

Java and, 4

language rules, 69, 75–97

learning process, 3

menu objects. *See* menus

Microsoft Jet DBMS, 396

OOP development and, 4–5

program template, 38–39

copying, 40, 57

programs (common features). *See* C# programs

scope and, 214–215. *See also* scope

System.Object, 307

C++ (programming language), 4

C# Express

download, 5–6

IDE. *See* Visual Studio IDE

installation, 5–7

startup screen, 8

test program (to verify installation), 8–16

C# program template, 38–39

copying, 40, 57

***The C Programming Language* (Kernighan & Ritchie), 143**

C# programs (common features), 126–130

constructors, 128–129

execution, beginning of, 128–129

invoking the application, 129–130

#region and #endregion, 38, 126–127, 217, 296, 355

sliding down object chain (dot operator), 127–128

calendar, Gregorian, 123

caller, 61, 118

'return to the caller,' 61, 118, 129

calling a method, 61, 118

calling another form, 392–393

calling default constructors, 237

camel notation, 43, 60

card game programs

acey-deucey, 270

euchre, 270

in-between, 256–269

shuffle cards (deck-of-cards). *See* shuffle cards

- Cards (database table), 396, 397**
 - designing fields for, 397
- CardType (database table), 399–400**
- carriage return (escape sequence), 122**
- carriage return-line feed (CRLF), 341**
- Carson, Johnny, 330**
- cascading if statements, 146**
 - switch statement and, 150–152
- case statement block, 151**
- cases. See also lowercase; uppercase**
 - case-sensitive (C#), 39, 128, 239
 - of strings, 116
- cassette music tapes, 351**
- cast, 272, 307, 326, 436**
- CD copy program (example), 282–288**
 - exceptions
 - anticipating specific, 285–286
 - example, 282–283
 - fuzzy, 286–287
 - try-catch block, 284–285
- ceiling-watching, 215**
- Celsius conversion program, 73, 95**
 - with symbolic constants, 97
- central processing units. See CPUs**
- Chamberlin, Donald, 402**
- changing**
 - object's state, 30, 242
 - properties, 30
 - Text property, 14–15
 - variable's state, 84
- char (value type), 72, 182, 183, 186, 382**
- character escape sequences. See escape sequences**
- check boxes, 276–277**
- child class, 446**
- class API, 222, 231–232, 269. See also API**
- class attributes, 219. See also properties**
- class constructors. See constructors**
- class design, 207–232**
 - clsCardDeck, 244–269. *See also* card game programs
 - constructors, 233–237
 - custom, 233–270
 - design goals, 22
 - naming of properties/methods, 223–224
 - program design and, 215–232
 - property methods, 237–244
 - scope and, 209–215
 - 'think before you write,' 215, 258
 - 'think like a user,' 224
- class members, 219. See also properties**
- class methods. See methods**
- class properties. See properties**
- class scope, 212**
- Class template, 209**
- ClassDesign program. See clsDates program**
- classes, 20, 23**
 - abstract, 459–460
 - adding, to projects, 208–209
 - base, 446
 - child, 446
 - components, 20. *See also* methods; properties
 - as cookie cutters. *See* cookie analogy
 - creating/writing. *See* class design
 - custom, 233–270. *See also* class design
 - defining, 23
 - derived, 446
 - designing. *See* class design
 - diagrams. *See* UML class diagrams
 - methods. *See* methods
 - in .NET Framework, 23, 37, 56
 - object as instance of, 24
 - parent, 446
 - properties. *See* properties
 - as templates, 20, 23, 24
 - user interface for, 231–232, 269
 - writing/creating. *See* class design
- cleaning up (at end of program), 32, 35.**
 - See also* termination step
- Clear button, 58**
- clear/concise code, 298. See also code**
- Clear() method, 139–140, 255, 277**
- click event code**
 - btnCalc(), 58–59
 - btnDisplayOutput, 44, 46–47
 - Odd or Even test, 137–138
 - string tester program, 112–113
- client-server architecture, 257**
- clock, system, 123**
- Close() method, 179, 217, 349**
- CLR (Common Language Runtime), 23, 380**
- clsApartment class, 446, 455–459**
- clsBuilding class, 446, 447**
 - code, 448–451
 - analysis, 451–453
- clsCardDeck, 244–256. See also card game programs**
 - class design, 244–269
 - code, 251–253

clsCardDeck (continued)

clsCardDeck (continued)

- analysis, 253–256
- class General methods, 254–256
- class properties, constructor, property methods, 253
- getCardsLeftInDeck(), 268–269
- getOneCard(), 268–269
- in-between card game. See in-between card game
- shuffle cards. See shuffle cards

clsCommercial class, 446, 455–459

clsDates class

- added to program, 208–209
- class organization, 227–228
- code, 225–226
 - analysis, 227–228
- customization of, 233–241. See also class design
- design of, 225–231. See also class design
- general methods, 228
- namespace modifier, 227
- property/Helper methods, 228
- static data/instance members, 228
- UML class diagram, 219

clsDates program (ClassDesign), 208–232. See also class design

- clsDates added to. See clsDates class
- code, 229–230
- Five Program Steps, 215–218
- getEaster() method, 208, 219, 222, 231, 233, 237
- getLeapYear() method, 218, 219, 222, 230, 232, 233, 237
- program design, 215–218
- user interface, 208, 229, 231–232

clsDates() method, 234

clsDB class

- code, 411–412
- GetColumnInfo() code, 415–416

clsDirectory class, code, 335–336

clsErrorLog class, 350

- code, 344–348
- using, 350

clsHome class, 446, 455–459

clsInBetweenRules class, 258, 262

- code, 263–268

clsPerson object

- instantiation of, 25–27
- Issy, 19, 20, 21, 22
 - values/methods/properties, 22
- Jack, 19, 20, 21, 22

- values/methods/properties, 22
- properties list, 20, 21

clsQuickSort class, 315–318, 319, 322, 324, 325, 326

- code, 316–318

clsRandomAccess class, 363

- code, 364–377
- record size, 377–379

clsSort class, 303–304

cocktail party, 97, 236, 305, 442

code

- breaking of, 63
- consistency in, 139, 143, 194
- decision making in, 135–152
- defensive, 93–96, 297, 298
- duplicate, methods and, 133, 134
- easy understanding of, 93–96, 135, 143, 152
- hiding/showing, 126
- less/minimal, 22, 27
 - inheritance and, 448
- reuse, **23**, 162, 207, 208, **218**, 394
- saving, 154
- saving (Ctrl+Shift+S), 154
- scaffold, 296–297, 302
- show-off, 231, 232
- simple v. complex, 23, 135, 298
- single-stepping through. See single-stepping
- style. See coding styles
- unnecessary, 145

Code File template, 38, 57

code reuse, 23, 162, 207, 208, 218, 394

coding styles, 135, 152

- consistency in, 139, 143, 194
- if/if-else statements and, 142–144

cohesion, 243–244

collapsing statements, 106–107

collection object, Items, 161

collections, 198–199

Collections program (example), 198–199

- ArrayLists, 200–203
- squares and cubes, 198–199

colon (:)

- operator, 236, 443–444. See also inheritance separator, 220

column/font alignment, 160–161

ColumnHeader Collection Editor, 183–185

columns (in listbox)

- formatting, 183–187
- listview object for, 183–187
- widths, estimation of, 186–187

- Columns dialog box, 183, 184, 185**
- Columns property, 183**
- COM (Component Object Model) tab, 406**
- combination boxes, 277–279**
- combo boxes, 277–279**
- comma operator (,), precedence, 149**
- comments, 93–94**
 - // before, 93–94
 - correct usage of, 93, 94
 - as debugging aid, 94
 - multi-line, 94
 - /* and */ , 94
 - placement of, 93–94
 - single-line, 93–94
- Common Language Runtime (CLR), 23**
- comparing strings, Equals() method for, 176**
- compiler program, 10, 23**
- complex code, simple v., 23, 135**
- complex expressions, 78**
- Component Object Model tab. See COM tab**
- computer data, 53. See also data types**
 - binary format, 54
- concatenation (string), 63, 107**
- concatenation operator (+), 63**
 - shorthand assignment operators and, 107–108
- concise/clear code, 298. See also code**
- conditional logical operators. See logical operators**
- consistency checking, 273**
- consistency, in coding, 139, 143, 194**
- const keyword, 95**
- constants, 95**
 - symbolic, 95–96
- constrained generic class, 319**
- constraints, 319. See also generics**
- constructor chaining, 236**
- constructors, 128–129, 233–237, 269. See also instantiation; specific constructors**
 - default, 234
 - calling, 237
 - fixing problem in, 236–237
 - non-default, 234–235
 - overloading, 235–237, 269
 - parameterized, 236, 237, 302, 454
 - passing arguments, 130–132
 - purpose of, 128
 - sloppiness, 235–236
- context error(s), 87**
 - symbol table, 84
- continue statement, 171–172**
- control objects, 124**
- conventions**
 - camel notation, 43, 60
 - data types (using/choosing), 56–57
 - identifiers, 59
 - context of use, 60, 95
 - prefixes (object name), 24–25, 43
 - symbolic constants in uppercase, 95
- conversion characters (DateTime), 123–124**
- conversions**
 - data to strings. See ToString() method
 - Fahrenheit to Celsius. See Celsius conversion program
 - ounces to grams, 173
 - strings to numeric. See TryParse() method
 - weight/height, 203
- cookie analogy**
 - cookie cutters (as classes), 23–24, 26, 27, 46
 - cookie dough (as memory), 24
 - cookies (as objects), 23, 24, 27, 46
- Copy(), 337**
- copy (Ctrl+C), 40**
- copying C# program template, 40, 57**
- CopyTo(), 338**
- correction step (debugging process), 88**
- cos() method, 67**
- COUNT (aggregate), 404**
- counter variable, 161, 170, 171, 177**
- counter-type variable, 410**
- count.GetUpperBound(val), 188**
- counting operations. See loops**
- count.Initialize(), 188**
- count.Rank, 188**
- count.This() method, 133**
- coupling, 243–244**
- Courier New font, 48, 160, 161**
- CPUs (central processing units), 56**
 - floating-point data types and, 67
 - integer data types and, 56
 - Intel Pentium-class, 67
 - math coprocessors, 67
 - registers, 56
- Create(), 331, 337**
- Create New DB button, 407**
- CREATE TABLE command, 408–409, 410**
- CreateDirectory(), 330**
- CreateNewDB(), 406**
- CreateSubdirectory(), 331**
- CreateText(), 337**
- creating objects. See instantiation**

creating/writing classes. See **class design**

CreationTime property, 338

CRLF (carriage return-line feed), 341

Ctrl+Alt+Delete (force quit), 154

Ctrl+C keystroke (copy), 40

Ctrl+D, E (debug exceptions), 285

Ctrl+D, I (Immediate window), 292

Ctrl+D, L (Locals window), 164, 292

Ctrl+D, W (Watch window), 293

Ctrl+F keystroke (find and replace), 65–66

Ctrl+Shift+S (save), 154

Ctrl+V keystroke (paste), 40

Ctrl+W (+S) keystroke (Solution Explorer window), 36

curly braces { }, 137, 142–144

K&R style, 143–144

location of, 143–144

proper usage of, 143

statement blocks in, 137, 142–144

current tick value, 132

custom classes (designing/writing), 233–270.

See also **class design**

D

“D,” “d” (conversion characters), 124

D, d (double data type suffixes), 64

Dahl, Ole-John, 4

data, 53. See also files

Boolean. See **bool**

computer, 53

binary format, 54

converted to numeric format. See **TryParse()** method

converted to strings. See **ToString()** method

hiding. See **encapsulation**

numeric, 53, 100. See also **value data types**

persisting. See **disk data files**

16-bit, 56

64-bit, 56, 67

static. See **static data**

storing of. See **data storage; databases; disk data files**

textual, 53, 70, 100, 107. See also **reference data types**

32-bit, 56, 67

Data attribute, 432

data binding (grid control/database), 423–426

without, 426–432

data declarations, 80, 83, 319

definitions v., 80, 83, 319

interfaces as, 319–325

Data Definition Language. See DDL

data definitions, 80–83

Bucket Analogy, 85–86

declarations v., 80, 83, 319

Visual Studio’s role

defining of variable, 82–83

lvalue, 83

memory address, 82, 83

memory request, 82–83

symbol table checking, 80–82

syntax checking, 80

data files. See files

data hiding. See encapsulation

data narrowing, 67

data normalization, 398

data provider element, 401–402

data sets, test, 289–290

data storage. See also disk data files

base-2 format, 54

numeric. See **value data types**

textual. See **strings**

data suffixes. See suffixes

data types

reference, 99–134. See also **reference data types**

value/numeric, 53–73. See also **value data types**

data validation, 31, 48, 273–274

consistency checking, 273

length checking, 274

range checking, 274

type checking, 273

data widening, 67

data wrappers. See wrappers

database(s), 395–442

commercially available, 400

creation of, 406–407

fields. See **fields**

queries. See **queries**

records. See **records**

relational, **395, 398**. See also **databases**

structure of, 396–397

tables. See **tables**

database (Access), 400

Visual Studio and, 400, 406, 425

database (example)

Cards table, 396, 397

designing fields for, 397

CardType table, 399–400

enhancing functionality, 399–400

- Friends table, 396, 397
 - designing fields for, 397
 - structure, 397, 400
- database management system. See DBMS**
- DataGridView object (grid control), data binding, 423–426**
 - without, 426–432
- DataSet object (ds), 430**
- datasets, 402**
- DataViewGrid properties, 431–432**
- DateTime (data type), 123**
 - conversion characters table, 124
 - formatting characters, 123–124
 - formatting date/time options, 124–134
 - properties/methods, 131–132
 - ToString() conversions and, 123–124
- DateTime tester program (example), 123–134**
 - C# template in, 126
 - control objects added to form, 124, 126
 - #region, #endregion, 126–127
- date/time user input, 279–281**
- DateTime() constructor, 130–131**
 - flavors, 132
- DateTimePicker object, 280, 281**
- DB2, 400**
- DBMS (database management system), 396**
- DBMS (example program), 405–406**
 - database MDI, 405–406
- DDL (Data Definition Language), 402**
- Debug toolbar, 294–295**
- debugger (Visual Studio), 87–93, 291–295, 297**
 - breakpoint
 - examine variables, 89–91
 - setting, 88–89, 130, 140, 164, 291
 - as learning tool, 164
 - single-stepping. *See* single-stepping
 - using, 88–93
- debugging, 87, 288–297**
 - comments as aid in, 94
 - process, 87–88, 289–291
 - correction step, 88
 - detection step, 87, 289–290
 - isolation step, 88–93, 290–291
 - stabilize step, 88
 - repetitious, 88
 - shotgun approach, 88
- decimal (data type), 67–70**
 - M, m (suffixes), 68, 69, 144
- decision making (in program code), 135–152.**
 - See also* if statement; if-else statements
- decision-making keywords, 135**
- deck-of-cards program. See shuffle cards**
- declaring variables, defining v., 80, 83, 319. See also data declarations**
- decrement operator (-), precedence, 149**
- default access specifier, 220**
- default constructors, 234**
 - calling, 237
- default keyword, 151, 152**
- default precedence, overriding, 79–80, 149, 150**
- defensive coding, 93–96, 297, 298. See also code**
- defining classes, 23**
- defining variables, declaring v., 80, 83, 319.**
 - See also* data definitions
- Delete(), 330, 331, 337, 338**
- DELETE command, 421**
- derived classes, 446**
 - base classes v., 459
- DESC, 403**
- deserialization, 382**
 - serialization-deserialization program, 382–388
- design mode, 12**
- designing**
 - classes. *See* class design
 - programs. *See* program design
 - user interface. *See* user interface
- detection step (debugging process), 87, 289–290**
- dimensions, of arrays, 189–190. See also dynamic arrays**
- dirCounter, 337**
- directories, 330–331**
- Directories program (example), 332–337**
 - clsDirectory class code, 335–336
 - user interface, 332–334
 - code analysis, 334–335
- Directory class, 330–331**
- Directory property, 339**
- DirectoryInfo class, 331**
- DirectoryName, 339**
- disk data files, 329–394**
- display step, 32, 35**
 - clsDates program, 217
- division operator (/), 63, 76. See also modulo**
- divide operator**
 - multiplication operator v., 144, 145
- dog (meowing), 87, 272**
- doodling, 215**
- DoQuery() method, 430**

