INDEX

Symbols

! (null-forgiving; dammit) operator, 66–67
!= operator, 150, 167–168, 215–217
 arithmetic and nonarithmetic types, 208
 comparing generic variables, 162
 equality behavior in derived classes, 219
 implementing custom vs. generated behavior, 201
 method overloading, 240
 nonstandard operator behavior, 209
 optimizing equality, 266–267
 value-based equality for classes, 215–217
& (binary combination) operator, 122
&& (logical AND) operator, 216
* operator, 14–17
?? (null-coalescing) operator, 51, 151–152
| (binary combination) operator, 122
|| (logical OR) operator, 216
- operator, 17
+ (concatenation) operator, 241
+ operator, 122
+= operator, 76
< operator, 122, 206, 208, 229
== (equals-equals) operator, 122, 139–140, 150
 boxing values and identity comparison, 158–161
 class equality, 151–152, 156
 compiler-generated equality, 167–172
 custom vs. generated behavior, 201
 equality behavior in derived classes, 219–220
 floating-point values, 141–142, 145
 generic variable comparison, 162–163
 interface boxes, 161
 method overloading, 240
 nonstandard behavior, 209
 optimizing equality, 266–270
 reference equality, 146–148
 struct equality, 157–158
 value-based equality for classes, 215–217
 value semantics vs. reference semantics, 177
 whole numbers, 140
=> (expression body) syntax, 7, 24, 32, 92, 151
> operator, 208, 229

A

abstraction
 abstract types, 29, 36
 composing, 199
 encapsulation and cohesion, 192–193
 missing abstractions, 15, 28
 new expression, 50
 performance, 279
 vocabulary and, 191–192
accessors, read-only, 134–135
Action type, 93–94
actual parameters, 38. *See also* arguments
address
 of an object, 74–75
 of a variable, 128
ad hoc polymorphism (method overloading), 238–242
 generic delegates for polymorphism, 241–242

ad hoc polymorphism *(continued)*
symbolic polymorphism with overloaded operators, 240–241
anemic domain model, 184, 192–193
AngleExtensions class, 26
angle parameter, 3–5, 11
Angle type
automatic vs. nonautomatic properties, 129–133, 135
boxing, 161
composition, 199
encapsulation, 5–7, 10
floating-point fields, 155
unit conversions, 24–28
anonymous methods, 85, 242
anonymous types, 60, 91
ApproximatelyEqual method, 143, 155
Area type, implementation inheritance and, 228–230
arguments, 69–103. *See also* methods; actual parameters
by-reference parameters, 76–82, 102–103
checking for invalid, 8
copying, 118–123
custom types, 5–6
double type and, 4–5
method arguments, 70–75, 118–123
method calls, 38
named, 4–5, 7–8, 33
null-forgiving operator, 66–67
object initialization, 6
ordering, 5
ArgumentException and ArgumentOutOfRange Exception type, 8
overloading constructors, 54
parameterless constructors, 54–55
passing null as, 52, 61, 65–66
passing variables as, 37–38, 44, 47
private constructors, 57
properties as, for read-only parameters, 127–128
read-only references, 92–102
arithmetic operators, 14–15, 208–209
array elements
accessibility, 38
default initialization, 56
identifiers, 38
instance storage, 41–42
variables, 38, 42, 147–148
asynchronous methods, 86–87
AutoAppend method, by-reference parameters and, 76–77
automatic properties, 43, 58, 130–134, 213, 265–266
await keyword and statements, 86–87

B

BallisticRange method
address of a variable, 127–130
defensive copying, 132–133
benchmarking, 251–252, 271
binary combination (| and &) operators, 122
bitwise comparison, 49, 141, 256–257, 259
bne instruction, 171
boxed values, 44–45
generic code and Equals method, 163
identifying unnecessary boxing, 115–118
identity comparisons, 158–162
lock statements, 47
optimizing equality, 263–265
passing values by reference, 79
performance and, 172–173
Brush type
returning by-reference, 101
value type member layout, 40–42
value type reassignment, 113
by-reference fields, 84–87
asynchronous methods, 86–87
closures, 84–85
iterator blocks, 85–86
by-reference parameters, 76–92, 99, 102–103
defined, 70
kinds of, 70
limitations of, 82–88
output parameters, 79–82
passing arguments by reference, 70–71
passing references by reference, 77–78

passing values by reference, 78–79
reference types vs., 70–71
reference type variables vs., 76–77
side effects and direct effects, 88–92
by-reference returns, 97, 99
by-reference variables (ref locals), 84, 94, 97–102, 104, 123, 133, 270
defined, 95
keeping within scope, 97–101
performance vs. simplicity, 101–102

C

`callback` delegate, mutating arguments for read-only parameters and, 93–94
cancellation
defined, 142
mitigating limitations of, 142–144
`ceq` instruction, 141, 145–147, 151
`bne` instruction vs., 171–172
efficiency of, 216
`Character` class, default object hash codes and, 153–154
CI (continuous integration) services, 251
classes, xx, xxiii
abstract, 36
records vs., 33, 200–201
default constructors, 52
defining, 33
embedded references, 42
embedded values, 40
equality behavior in derived classes, 218–219
equality comparisons, 149–156
field assignment, 53
field initializers, 58
generics and `null`, 61, 63
immutability, 33, 35
inheritance, 34–35
iterator blocks, 85–86
memory allocation, 50
object initializers, 58–59
parameterless constructors, 54, 57, 59
protected members, 35
sealed, 35
value equality, 48, 214–218
class factory methods, 22–25, 57
custom vs. generated behavior, 201–202
returning types implied by units, 27
symmetry, 23–24, 196
class invariants
defined, 9
discarded, 13–14
establishing, 194–195
testing, 9
`class` keyword, 33, 45
`Clone` method
boxing, 115
of record types, 114, 165
clones (deep copies), 200
`Close` method, reference semantics and, 75
closures
by-reference parameters and, 84–85
defined, 84–85
looping and iteration, 280–281
code craft, 3
coercion polymorphism, 242–247
conversions for purpose, 245–247
conversions for representation, 244–245
widening vs. narrowing conversions, 244
cohesion, 192–196, 198
clarifying with symmetry, 196
eliminating duplication, 193–194
establishing class invariants, 194–195
collisions of hash table elements, 153
`ColorBuilder` type, as mutable companion type, 246–247
`ColorParser` type, generic type constraints and, 235–238
`Color` type
ad hoc polymorphism, 239–242
array elements, 41–42
boxed values, 44
classes, 33
coercion polymorphism, 243–247

Color type *(continued)*
compiler-generated equality, 165–169
constructing value types, 112–113
constructor accessibility, 57
conversion to interfaces, 116–117
copying records like value type, 114
default and generated constructors, 53
embedded values, 40–42
generic code and Equals method, 163
IEquatable interface, 164
inclusion polymorphism and subtyping, 222–229, 231–232
init-only properties, 59
instance fields of value types, 98
measuring performance with Equals, 253–259
non-destructive mutation, 60
ordering, 206
overloaded constructors, 54
overriding Equals for structs, 157–159
overriding generated methods, 203
parametric polymorphism with generics, 233, 235–238
records, 33
record structs, 34
references to references, 100
sealed value types, 213–221
structs, 32–33
uniformity and consistency, 207
value-based comparisons, 181
value type initialization, 56–57
Combine method, of HashCode, 259–261
Command type
controller object role, 190
inheritance, 34–35
reference semantics, 74–75
Common Type System, 45–46
CompareTo method
boxing, 116–117
inheriting, 228–230
sorting, 204–208
comparison operators, 122, 205, 208
composition, 199
concatenation (+) operator, 241
concrete types, 36
conjunctive pattern, 195
constant pattern, 61, 145, 151–152, 195, 216
constants
enums, 19
hidden copies, 128
replacing magic numbers with named constants, 10–11
constructors, 49–57
accessibility, 57
default and generated, 52–53
expression body syntax, 7, 32
field and property initializers, 58
new object creation, 110–111
object initializers, 58–59
overloaded, 53–54
parameterless, 54–55, 62–63
private, 57
replacing public constructors with static methods, 22–23
structs and default values, 55–56
value type construction and initialization, 56–57, 112–113
value validation, 8–9
continuous integration (CI) services, 251
contract for comparisons, 205–206
antisymmetric, 205
irreflexive, 205
safe, 206
stable, 206
transitive, 205
contract for equality, 217–218
breach of, 221
reflexive, 217
safe, 217
stable, 217
symmetric, 217
transitive, 217
upholding, 224–225
controllers
avoiding implementation inheritance, 230–231
characteristics of, 190–191
defined, 185
conversion operators, 7, 12, 122–123, 243–246

Coordinate type
  boxing, 115–116
  value type memory layout, 71–72
  value tuples, 102–103
copying, 46–48, 105–137
  copy-by-value semantics, 73–74
  defensive copies, 128–133
  identifying unnecessary boxing, 115–118
  large instances, 272
  locks and reference semantics, 46–47
  measuring cost of, 270–271
  method parameters and arguments, 118–123
  modifying return type instances, 123–128
  new object creation, 110–115
  references vs. instances, 105–106
  simple assignment, 106–110
  value equality, 47–48
  value semantics vs. reference semantics, 177–181
CPU sampling, 254, 271
CreateColor method, returning by reference and, 100
CrossGen utility, 250

D

dammit (!; null-forgiving) operator, 66–67
dangling references, 84, 99
DataAdapter type, generic type constraints and, 234–238
DataStore type, reference semantics and, 74–75
DateTime type
  boxing, 118
  IComparable interface, 206
  Try*XXX* idiom, 79–80
decimal type, 143
declaration patterns, 150, 157
declarative code
  defined, 89
  immutability, 89
  performance and, 91–92
  vs. procedural code, 89, 284
Deconstruct method, 102–103
deep copies (clones), 200
default constructor, 52–54, 58
default keyword, 62–63
default values
  avoiding pitfalls of default variables, 200–201
  default initialization, 50, 56
  generics and, 62–63
  null, 64
  object initializers, 59
  valid, 63
defensive copies, 128–133
  automatic vs. nonautomatic properties, 130–131
  avoiding, 133–136
  causes of, 133
  defending against mutation, 133–134, 136
  mutable value types and in parameters, 129–130
  read-only fields, 132–133
  read-only reference variables, 131–132
definite assignment, 39
  generics, 62
  ref and out parameters, 79
  static fields, 58
  struct variables, 56–57
deferred execution (lazy enumeration), 85
delegates
  callback delegate, 93–94
  defined, 93, 241
  generic, for polymorphism, 241–242
derived type, 34
Difference method, memory layout of method parameters and, 72
digit separators, 262
Direction property, nondestructive mutation and, 90–91, 127–128
discard pattern, 20–21, 195, 284–285
Disconnect method, reference semantics and, 178–179, 181
disjunctive pattern, 195
Displacement method, 2–29
  custom types, 5–6
  encoding units, 18–28

Displacement method *(continued)*
  importance and value of good names, 2–3
  named arguments, 4–5
  refactoring implementation, 9–18
  value validation, 8–9
DistanceInKm method
  read-only reference parameters, 92–93
  user-defined conversions, 122–123
Distance type, refactoring and, 16–18
domain-specific types, 9
double argument, 12, 25
double values
  custom types vs., 4–5
  equality comparisons, 141–142, 145, 155, 229

E

embedded values, 40–43
  array elements, 41–42
  embedded references, 42–43
  field and property layout, 43
encapsulation, 6–7, 25–26, 28, 192–196
  abstracting types, 29
  clarifying with symmetry, 196
  cohesion and, 192
  eliminating duplication, 193–194
  establishing class invariants, 194–195
  public interface and, 196–199
  testing, 9
entities
  characteristics of, 188–189
  defined, 184–185
Enumerable class, 262, 282
enum (enumerated types)
  Common Type System, 45
  defined, 19
  itemizing units with, 19–22
EqualityComparer class, 263
  IEquatable interface and, 164–166
  performance of, 277–278
equality comparisons, 139–172
  boxing, 158–161, 163
  built-in, 140–149
  classes, 149–156
  compiler-generated, 165–172
  contract, 217
  equality behavior in derived classes, 218–219
  equivalence vs., 203–205
  floating-point numbers, 141–142, 144–145
  generics, 162–165
  records, 165–168
  reference-based, 47–48, 146–152, 177–178
  strings, 148–149
  structs, 156–162
  transitivity, 155
  type safe, 152
  type substitution, 220–221
  value-based equality for classes, 214–218
  value semantics vs. reference semantics, 177–181
EqualityContract property, 226–228, 247
equals-equals operator. *See* == operator
Equals method, 33, 44, 46, 64, 139–140, 150–151
  boxing values and identity comparison, 158–159, 161
  canonical form of, 215–217
  class equality, 151–152, 155–156
  compiler-generated equality, 165–166, 172, 277–279
  copying large instances, 272
  custom vs. generated behavior, 201
  equality behavior in derived classes, 219, 221
  floating-point values, 145
  generic variable comparison, 163–164
  GetHashCode method and, 152–156, 259–261
  IEquatable interface, 164–165
  inheriting classes, 218–221
  inheriting record types, 225–228
  input and output types of virtual methods, 223, 225
  measuring basic performance with, 253–261
  measuring cost of copying, 271
  method overloading, 238–239

object base class, 47–48, 158–161, 163–164
optimizing equality, 261–265, 268–269
overloading, 152, 164–165
overriding base class implementation, 199
overriding for structs, 156–158
records and structs, 48–49
reference equality, 146, 148, 276–277
struct equality, 156–158
value-based equality for classes, 215–217
value equality, 48
value semantics vs. reference semantics, 177
ValueType base class, 156, 256–258
EqualsOperatorComparer class, 268
EqualViaBase method, 221, 226, 228
EqualViaDerived method, 221, 226
equivalence vs. equality, 203–205
explicit conversions, 106, 243, 245
explicit interface conversion, 116–117
expressions, using with operators, 121–123
expression-bodied properties (nonautomatic properties), 130
expression body (=>) syntax, 7, 24, 32, 92, 151
extension methods
ad hoc polymorphism, 239
of built-in types, 26
by-reference parameters and, 87–88
composing abstractions, 199
extending interfaces, 197–198
iterators, 281
passing and returning by value, 119–120
returning types implied by units, 27
external interface, 198
extensionality, 204

F

fakes (test doubles), 230
field initializers, 58, 113
fields
array elements, 41–42
by-reference, 84–87
copy semantics, 46
default initialization, 50, 53
embedded fields, 40–41
embedded references, 42
embedded values, 43
field and property layout, 43
generics, 62
identity equality vs. value equality, 47–48
initializers, 58
instance, 37, 98
parameterless constructors, 54–55
properties vs., 265–266
protected, 35
read-only, 132–134
return by reference, 131–132
static, 38, 58
value semantics, 49
value type initialization, 56–57
floating-point values equality comparisons, 141–145
rounding and cancellation errors, 141–144
using as keys with hash codes, 154–156
fluent syntax form of LINQ (Language-Integrated Query), 279–280
foreach loop, 36, 66, 281–284
for loop, 283–284
formal parameters *See* parameters
FormatConnection method, output parameters of extension methods and, 87
Format method, method group overloading and, 242
FORTRAN, 182

G

garbage collection, xxiii, 37, 41, 74,
boxing, 115
managed pointers, 99
performance, 263, 275
read-only references, 131
generics, xx–xxii
arithmetic, 162

generics *(continued)*
- base-class constraint, 162–163
- deduced type parameters, 236–238
- default values, 62–63
- equality comparisons, 162–165
- generic delegates for polymorphism, 241–242
- interface constraint, 116–117
- `null` values, 61–62
- parametric polymorphism, 233–238
- partial deduction of generic parameters, 237

`get` accessor, 32–34, 128
- automatic vs. nonautomatic, 130–131
- methods for, 120–121
- read-only, 134–135
- symmetry with `set` accessor, 209

`GetAddress` method, 99
- returning by reference, 99

`get_Current` method, 282
`GetEnumerator` method, 281
`GetHashCode` method
- collision, 152–153
- defining, 154, 157
- performance, 259–261
- `ValueType` definition, 156, 259
- where used, 152

`get_Speed` method, 120–121
`GetType` method, 118, 166, 216, 227
`GetValueOrDefault` method, 169
`Gravity.Earth` constant, replacing magic numbers with, 11

H

`HashCode` class, 154, 259–261
hash codes, 152–156
- collision, 152–153
- creating suitable keys, 154
- distribution, 152–153, 259
- `Equals` and `ValueType.GetHashCode` methods, 259
- using floating-point numbers as keys, 154–15

heap, xxiii–xxiv, 37, 40, 50, 73–74, 107–108
- boxing, 158–161
- identity, 180

hidden copies, 130, 137
- boxes, 118
- parameter passing, 128
- return values, 124–126
- value type construction, 118

I

`IComparable` interface, 116–117, 204–208, 228–229, 231
`IComparer` interface, 206–207
IEEE-754, 141
identifiers for variables, 37
identity comparison, 145, 148, 276
- boxed values, 158–162
- identity equality vs. value equality, 47–48, 158–159

identity conversions, 150
`IEnumerable` interface, 85, 241, 281–282
`IEnumerator` interface, 281–282
`IEqualityComparer` interface, 165, 206–207, 267–268, 271
`IEquatable` interface
- avoiding boxing, 164
- contract for, 217
- `IComparable` interface vs., 203–205
- implementing, 164, 215–216
- performance effects, 263–265

`if...else` statements, 81, 285
`IFormattable` interface, 161
`if` statements, 284–285
immutability. *See* mutation and immutability
imperative code, 89
implementation inheritance, 213–214
- avoiding, 230–232
- containing instead of inheriting types, 231–232
- interface inheritance vs., 213
- upholding a type's contract, 224

implicit conversions, 47, 97, 117, 150
- boxing, 117–118
- by-reference variables, 95–97
- coercion polymorphism, 242–247
- defining, 12–13
- discarded invariants, 13–14
- implicit reference conversion, 150, 220–221
- primitive obsession, 5

to and from null, 147
unexpected interactions, 13
user defined, 122–123
Inch Calculator (online unit conversions), 120
inclusion polymorphism, 222–232
avoiding implementation inheritance, 230–232
inheriting record types, 225–230
input and output types of virtual methods, 223–224
upholding contract, 224–225
Incremented method
naming conventions, 89–90
passing by value, 119
Increment method
overloading by-reference parameters, 83
passing by reference, 78
indexers, 82, 106, 118, 125
defined, 121
mutating values from, 126–127
infinity, 145
inheritance, 34–36. *See also* inclusion polymorphism
init accessor, 59, 91, 112
init-only properties, 59, 112, 114, 274
In method, using for unit conversion, 21–22
in modifier and parameter
definition, 70–71
as optimization, 268–270
property values as arguments for, 127–133
using, 92–93
InMph extension method, 120
input parameters, 70, 93
InRadians property
automatic vs. nonautomatic properties, 129–130
defensive copying, 132–133
unit conversions, 24–26
instrumentation profiling (tracing), 258
intentionality, 204
interface constraints, 233–235
interface keyword, 224
interfaces
abstractness, 36
boxed values, 44, 116–117, 161–162
extension methods, 27
generic constraints and protocol, 233–236
implementing and inheritance, 35
interface inheritance, 213, 222
parametric polymorphism, 233–236
types, defined, 161
internal interface, 198–199
InternetTime class, as service object, 188
intern pools, 149
InvalidCastException error, 45, 107
IParser interface, generic type constraints and, 234–238
IsNaN static method, 145
IsReadOnlyAttribute indicator, 131–132, 134–136
iterator approach to creating sequences, 281–283
iterator blocks, 85–86
iterators, 281–282

J

JIT (just-in-time) compiler, 137, 141, 250–253, 265, 270–271, 273, 277–279, 281
Journey type indexer values, 126–127

K

keys, 152–156
creating suitable, 154
using floating-point numbers as, 154–156

L

lambdas, 66, 85, 94, 241–242, 273, 280–281
lazy enumeration (deferred execution), 85
ldnull instruction, 147
level of indirection, 70, 75, 79, 212
lexicographical ordering, 206
lifting operators, 169
local functions, 85

local variables
  associated type, 38
  constructing value types, 112
  defined, 37
  definite assignment, 39
  instance storage, 43
  keeping by-reference variables within scope, 97–100
  lifetime of, 36
  local read-only reference variables (ref readonly locals), 132–134
  passing and returning by value, 119
lock statements, 47, 180
LogEntry type, equality vs. equivalence and, 203–209
logical AND (&&) operator, 216
logical OR (||) operator, 216
Login type, reference semantics and, 178–181
loop approach to creating sequences, 283–284

M

magic numbers, 4, 10–11
magnitudinal ordering, 206
Mail type, returning by reference and, 95–97, 99
managed pointers, 99
Math class, 10, 12–13, 17–19, 24–26, 143–144
memory profilers, 252
memory tearing, 114
method groups, 240, 242, 273, 280–281
method overloading (ad hoc polymorphism), 238–242
methods. *See also* arguments; actual parameters
  abstract, 36
  adjectives as names, 90
  anonymous, 85, 242
  asynchronous, 86–87
  class factory, 22–25, 27, 57, 196, 201–202
  encapsulation, 9
  encoding units, 22–27
  extension, 26, 197-199
    ad hoc polymorphism, 239
    of built-in types, 27
  by-reference parameters and, 87–88
  passing and returning by value, 119–120
  generics, 61–63
  identifying unnecessary boxing in method calls, 117–118
  implementing from interfaces, 35
  inheritance, 34–36
  naming identifiers, 2–5
  overloading, 16, 83–84
  overriding generated, 202–203
  protected, 35
  read-only, 134–135
  return type instance modification and mutability, 126–127
  static creation, 22–23
  value equality, 48
mock objects (test doubles), 230
ModifyByCallback method, mutating read-only reference parameter arguments and, 93–94
Monitor class, 47, 180
MoveNext method, 282
multiple inheritance, 34, 36
MusicTrack type
  customizing equality, 154, 156–157
  reference type field initialization, 50–52, 55–56
Mutable Companion pattern, 246
mutation and immutability, 7–8
  classes, 33, 35
  declarative code, 89
  defending against mutation with defensive copies, 133–134, 136
  immutable types vs. read-only properties, 109–110
  init-only properties, 59
  instance methods and mutability, 126–127
  mutable immutable properties, 101
  mutable value types and in parameters, 129–130
  mutating values from indexers, 126–127
  mutation vs. creation, 89–91
  non-destructive mutation, 60
  in parameter, 92

read-only properties vs. immutable types, 109–110
record structs, 34
return type instance modification and mutability, 126–127
structs, 32–33
value semantics vs. reference semantics, 181–182

N

`NaN` (not a number), 8, 144–145, 195, 256, 285
narrowing conversions, 244
`NegativeInfinity` method, 145
`new` keyword and expression, 14, 50–51, 56–57, 110
`newobj` instruction, 146
`NextAppointment` method, capturing by-reference parameters and, 85
nonautomatic properties (expression-bodied properties), 130
non-destructive mutation, 60
not a number (`NaN`), 8, 144–145, 195, 256, 285
not constant pattern, 216
nullable reference types, 52, 64–67, 151, 224
`Nullable` type, 64, 168–172
nullable value types, 60, 63–64, 168–170
null-coalescing (`??`) operator, 51, 151–152
null-conditional (`?`) operator, 52, 151
null-forgiving (`!`; dammit) operator, 66–67
`Nullify` method, by-reference parameters for extension methods and, 87
`NullReferenceException` error, 56
`null` references, 51–52, 56, 65–67, 146–147, 151, 206, 233–234
comparing reference types with `null`, 61
comparing value types with `null`, 61
equality comparisons with classes, 151–152
generics and, 61–62
nullable reference types, 52, 64–67, 151, 224
nullable value types, 60, 63–64, 168–170
null-forgiving operator, 66–67
parameterless constructors, 54

O

object address, 180
`object` base class
Common Type System, 33, 44–46
default equality, 48, 139, 163, 215–216, 253
generic type parameters, 233
object construction and initialization, 49–60
constructors, 51–57
copying value type instances, 110–115
default initialization, 50–51
field and property initializers, 58
measuring cost of, 273–277
memory allocation, 49–50
object initializers, 7, 58–60
`init`-only properties, 59
non-destructive mutation, 60
object deconstruction, 102–103
object identity, 180, 185–191
and boxing, 158, 161
hash codes, 153
object-oriented programming (OOP), 183–184, 211
object relationships, 183–191
characteristics of, 185–191
design refinement to model object roles, 191
kinds of objects, 184–185
object roots, 99
OOP. *See* object-oriented programming
`op_Equality` method, 167, 171–172, 268
operators
arithmetic, 14–15, 121–122, 208–209
lifting, 168–170
nonstandard behavior, 209

operators *(continued)*
  symbolic polymorphism with
    overloaded operators,
    240–241
  using expressions with, 121–123
optimization, 101, 142, 250
  boxing, 118
  mutable by-reference parameters,
    91–92
ordering, comparison for, 203–207
  contract for comparisons, 205–206
  equivalence vs. equality, 204–205
  lexicographical ordering, 206
  ordinal comparisons, 206
output parameters, 79–82, 93, 102
  deconstruction, 102–103
  defined, 70
  definite assignment, 80–81
  object deconstruction, 102
  reference parameters vs., 79
  returning by reference, 100
  selecting operations, 81–82
  `Try`*`XXX`* idiom, 79–82
overloading, 16, 152, 238–241
  by-reference parameters, 83–84
  constructors, 53–55
  operators, 14–15, 208–209
  overriding vs., 221
`override` keyword, 149

P

parameterized types, 237–238
parameterless constructors, 54–59,
    62–63
parameters, 69–103
  aliasing, 74–75
  arguments, 38, 118–120
  boxed, 160–161
  by-reference, 70–71, 76–92, 99,
    102–103
  custom types as, 6
  defining interfaces, 223–224
  formal vs. actual, 38
  generic, 61–62, 162–163, 233–238
  input, 70, 92–93, 127–128, 268–270
  kinds of, 70
  modifiers, 70
  naming, 2–5
  non-nullable, 52, 66–67
  output, 70, 79–82, 93, 102
  overloading constructors, 53
  overloading methods, 16
  passing, defined, 69–70
  read-only, 92–102, 127–128,
    268–270
  reference, 70–71, 73–74
  ref returns, 92–102
  value, 70–73
parametric polymorphism, 233–238
  generic constraints and protocol
    interfaces, 233–236
  generic method parameters and
    type deduction, 236–237
  parameterized types, 237–238
`Parse` method, `TryParse` vs., 80
passing arguments
  defined, 69–70
  by reference, 70–71, 77–79
  by value, 70–71, 119–120
pattern matching and selection
  conjunctive pattern, 195
  disjunctive pattern, 195
  `is` constant pattern, 61, 145,
    194–195
  performance of, 284-286
  relational pattern, 195
  `switch` expression, 20–21,
    284–286
performance, 249–286
  effect of common idioms and
    practices on, 279–286
  effect of types on, 270–279
  measuring and optimizing,
    250–253
  measuring with `Equals`, 253–261
  optimizing equality, 261–270
  profilers, 252–253
pessimization, 250
`Playlist` class
  field initializers, 58
  generic type parameter
    comparisons, 162
  parameterless constructors, 54–55
pointers
  managed, 99
  reference types vs., 74

polymorphism, 211–247
    ad hoc, with overloading, 238–242
    coercion, using conversions, 242–247
    inclusion and subtyping, 222–232
    inheritance vs., 211–212
    parametric, with generics, 233–238
    sealed value types, 212–221
positional records and record structs, 33, 53, 55, 131, 201–203, 273
    copying, 114
    equality, 165–168, 277–279
    inheritance, 225–227
`PositiveInfinity` method, 145, 285
precision, 142–144
Primitive Obsession code smell, 5
private constructors, 23, 57
procedural code
    declarative code vs., 89, 284
    defined, 89
`Product` type
    copying large value types, 273–279
    optimizing equality comparisons, 261–268
    read-only vs. immutable, 107–110
    value object role, 186–187
profilers, 252–253
`Projectile` type, returning by reference, 131–133
properties
    abstract, 36
    accessing, 120–121, 265–266
    as arguments for read-only parameters, 127–128
    automatic
        initializers, 58
        memory layout, 43
        nonautomatic vs., 130–134
        performance of fields vs., 265–266
    by-reference parameters and property values, 82–83
    circular dependency, 43
    expression-bodied, 130–131
    `init`-only, 59, 112, 114, 274
    mutable immutable, 101
    property forwarding, 24
    property initializers, 58
    read-only. *See* read-only properties
    returned reference type instance modification, 125–126
    simplifying, 11–14
    value of, 82
protocol interfaces, 231, 233–238
pseudorandom number generator, 254
public interface, encapsulation and, 196–199
    composing abstractions, 199
    extending interface, 197–198
    reducing internal interface, 198–199
`Purchase` type
    copying large value types, 273–280, 282–283
    optimizing equality comparisons, 261–269
    read-only vs. immutable, 107–112
    value object role, 186–187

Q

query syntax form of LINQ (Language-Integrated Query), 280–281

R

race conditions, 96
`Random` class, 254
ranges (slices), 235
reachable objects, 99
`readonly` keyword, 32–34
    ref locals, 96
    structs, 135–136
read-only properties, 7
    as arguments for read-only parameters, 130
    avoiding defensive copies, 135
    immutability vs., 109–110
    and ref returns, 104
read-only reference parameters, 88, 92–95, 129, 131–132
    mutable immutable properties, 101
    performance vs. simplicity, 101–102
    preventing modifications to data, 95–97
read-only type, 135–136
real number, 144
`record` keyword, 33, 45

records
  abstract, 36
  copying like value types, 114–115
  defining, 33–34
  equality comparisons, 165–168
  inheritance, 34
  inheriting record types, 225–230
  protected, 35
  sealed, 35
  value semantics, 48
`record struct` keywords, 34
record structs
  defining, 34
  equality comparisons, 165–168
  immutability, 34
  inheritance, 34
  value semantics, 49
refactoring, 9–10, 191
reference equality, 145–148, 178–180, 275–277
`ReferenceEquals` method, 148–149, 160–161, 216
reference parameters, 65, 70–71, 73–74, 79, 239
reference return values. *See* ref returns
reference semantics, 45–48, 176–183, 199–203
  avoiding pitfalls of default variables, 200–201
  Common Type System, 45–46
  copying and equality comparison behavior, 177–181
  copying variables, 46–48
  implementing custom vs. generated behavior, 201–202
  mechanics vs. semantics, 182–183
  mutability, 181–182
  overriding generated methods, 202–203
reference types
  array elements, 42
  by-reference parameters vs., 70–71
  classes, 33
  default initialization, 52–53
  identity comparison, 47
  inheritance, 34, 36
  instance lifetime, 37
  instance storage, 40
  locks and semantics, 46–47, 180
  nullable, 64–66
  performance, 274–275
  pointers vs., 74
  records, 33
  return type instance modification, 123, 125–126
  value of, 73–74
  value-like performance, 274–277
  value types vs., xx, 31, 70–71, 123–126, 176–181, 212–213, 273–277
reference variables
  aliasing and, 74–75, 88, 96, 107–110, 125, 178–180
  boxing and unboxing, 44
  by-reference parameters vs., 76–77
  defensive copies, 131–132
  equality comparisons, 145–148, 177
  fields of value types, 107–110
  instance storage, 42
  non-nullable reference variables, 64–65
  passing by reference, 77–78
  reference storage, 37
  scope, 37
  value of, 39
referential transparency, 180
reflection, 49, 156, 257–258
ref locals, 95–96, 132–134. *See also* by-reference variables
`ref` parameter modifier, 76–78, 80, 82–88
  defined, 70–71
  passing by reference, 77
  property value arguments for, 127–128
  returning by reference, 94–97
`ref readonly` locals (local read-only reference variables), 96, 132–134
ref returns (reference return values), 92
  keeping by-reference variables within scope, 97–101
  performance vs. simplicity, 101–102
  preventing modifications to data, 95–97
  returning values by reference, 94–95

relational operators, 195
relational pattern, 195
RelativeLuminance method, user-defined conversions and, 246
Reminder class, by-reference fields and, 84
RemoveRed method, returning by-reference parameters by reference and, 100
Reset method, variables vs. values and, 126
return type instance modification, 123–128
  instance methods and mutability, 126–127
  properties as arguments for read-only parameters, 127–128
  reference type properties, 125–126
rich domain model, 184
rounding, mitigating limitations of, 142–144
run-time type, 166, 220–221
rvalue, 124

S

sealed, 34, 212–213, 224–225, 231
seam, 230–231
Select method, 241–242, 273, 280–281, 283–284
self-documenting code, 6
semantics
  copying, 46–47, 75, 177
  equality, 166–167, 177
  reference, 46–47, 176–177
separation of concerns, 9, 117
SequenceEqual method
  comparing array elements, 148
  effect of IEquatable interface, 262–267
services
  characteristics of, 187–188
  defined, 184
set accessor, 7, 32–34
  mutable value types, 78, 89, 107–110
  object initialization, 58–59, 91, 100–112
  value type properties, 123–124
side effects and direct effects, 88–92
  declarative code and performance, 91–92
  mutation vs. creation, 89–91
  reference semantics and, 178–180
simple assignment, 106–110
  value copy behavior, 107–109
single responsibility, 28–29
slices (ranges), 235
sorting collections of values, 203–204
SpeedExtensions type, internal vs. external interface and, 197
Speed type
  anemic type, 192
  automatic vs. nonautomatic properties, 133
  encapsulation, 192–199
  passing and returning by value, 119–120
  replacing built-in types, 5–10, 12–24, 27–28
  variable vs. value, 129–130
stack, value types and, 39
static abstract interface members, 162
static creation methods, 22–23
Stopwatch class, 251–252
StringBuilder type, 125, 246
  Capacity property, mutating property values and, 125
string class
  concatenation, 209, 241
  equality, 148–149, 177
  interpolation, 52, 148
  StringBuilder vs., 246
  value of, 177
string interning, 149
structs
  cyclic dependencies, 43
  defining, 32–33
  equality comparisons, 156–162
  immutability, 32–33
  inheritance, 34
  tightly packed, 256
  value semantics, 49
stubs (test doubles), 230
Subtract method, 223–224
  derived method behavior, 223–224

subtyping, 222–232
  avoiding implementation inheritance, 230–232
  defined, 222
  inheriting record types, 225–230
  input and output types of virtual methods, 223–224
  substitutability, 222–223
  upholding contract, 224–225
`switch` expressions, 21, 194–195, 284–285
  `switch` statement vs., 285
symbolic polymorphism, 240–241
symmetry
  class factory methods, 23–24, 196
  contract for equality, 217
  encapsulation and cohesion, 196
`SynchronizationLockException` error, 47
`System.Diagnostics` namespace, 251
`System` types, 45

T

target-typed default feature, 63
target-typed new feature, 14, 55
`Temperature` type
  pattern matching, 285
  positional records, 200–202
ternary condition (`?:`) operator, 81, 151
test doubles (stubs; fakes; mock objects), 230
testing, 9, 15, 22, 218–221, 229, 245, 262
theta, 3
`this` parameter, 87–88, 120
`TimeSpan` type
  arithmetic operations, 14–18
  replacing built-in types, 6, 10, 12
  unit conversions, 24, 27
`Tolerance` constant, setting for floating-point comparisons, 143
`ToString` method
  boxing, 118
  default reference field values, 56
  implicit conversion vs., 246
`TotalSeconds` property of `TimeSpan` type
  arithmetic operations, 14–15
  `Seconds` property vs., 10
  unit conversions, 27
`ToTitleCase` method, nullable reference types and, 65–66
`TParser` parameter, generic type constraints and, 234–237
tracing (instrumentation profiling), 258
transitivity
  equality comparisons, 155, 217
  ordering comparisons, 206
`TranslucentColorParser` class, overloading method groups with, 238
`TranslucentColor` type
  composition, 231–232
  generic type constraints, 236, 238
  implementation inheritance, 213–214, 216, 218–229
  user-defined conversions, 241–244
triangular number, 258–259
truth operators, 122
`try...catch` blocks, 80
`Try`*`XXX`* idiom, 79–82
  selecting operations, 81–82
  side effects and direct effects, 88–89
tuples
  equality comparisons, 170–172
  named, 102
  tuple assignment, 32
  tuple deconstruction, 29, 102–103
two-stage initialization, 110–113
`typeof` method
  `EqualityContract` property, 226–227
  `GetType` vs., 118
types, 1–30. *See also* reference types; value types
  abstracting, 29, 36
  adding clarity through, 4–9
  benefits of understanding, xix, xxiii
  clarity, adding through, 4–9
  class vs., 222–223
  deduction, of generic type parameters, 236–237
  deduction, of `var` declarations, 106
  determining need for new types, 15–18
  effect on performance, 270–279

encoding units, 18–28
generic constraint, 117, 162–163, 233–234
implied by units, returning, 27–28
inference, 106
instances and storage, 39–45
naming, 2–4
new features in C3, xxiii
`null` values and default values, 60–67
object construction and initialization, 49–60
purpose and focus of book, xx–xxi
refactoring implementation, 9–18
semantics, 45–49
user defined, 32–37
type substitution, 220–221
breach of contract, 221
effects of, 220–221

U

ubiquitous language, 191–192, 288
unboxing, 44–45, 116, 163, 263
unconstrained generic type, 61–62, 162–163
uninitialized variables, 39
units, 18–28
choosing most natural usage, 25–26
itemizing with enumeration types, 19–22
making explicit, 24–25
returning types implied by, 27–28
static creation methods, 22–23
symmetry in design, 23–24
`Units` enumeration type, 19–21
unit testing, 251. *See also* testing
user-defined types. *See* classes; records; record structs; structs

V

validation, 8–9, 184, 193–194
value-based comparisons, 48, 177, 180–181, 215–216
records, 166–168
reference equality vs., 156, 158, 162–163
strings, 148–149, 177
value-like behavior and characteristics
classes and records, 35
records, 48
value object role, 184, 186–187
value parameters, 71–73
by-reference parameters vs., 76
defined, 70, 118
passing and returning by value, 119–120
passing arguments, 70
replacing with `in` parameters, 269
values, 175–210
abstraction and vocabulary, 191–192
boxed, 44–45, 47, 79, 115–118, 158–163
by-reference vs., 79
optimizing equality, 263–265
characteristics of, 186–187
comparison for ordering, 203–207
default, 50, 56, 59, 62–64, 200–201
defined, 184
definite assignment, 39
embedded, 40–43
encapsulation and cohesion, 192–196
encapsulation and public interface, 196–199
equality, 47–48
extensionality vs. intentionality, 204
identity comparison, 47–48
object relationships, 183–191
passing and returning by, 119–120
perils of uniformity and consistency, 207–209
simplifying, 11–14
unit conversions and value comparisons, 20–21
validation of, 8–9
variables vs., 38–39
value semantics
benefits of understanding, xxiv
reference semantics vs., 176–183, 199–203
value tuples
equality comparisons, 170–172
tuple deconstruction, 102–103

ValueTuple type, 170, 172
ValueType class
  Common Type System, 45–46, 211
  copying and identity, 48–49, 177–8
  default equality, 156–159, 163, 165, 253–260
value types
  advantages of, xxiii–xxiv
  arithmetic, 14–15
  avoiding defensive copies, 135–136
  construction, 52–53, 112–113
  copying, 110–115
  defensive copies, 129–130
  embedded fields, 43
  identity comparison, 47
  inheritance, 36
  initialization, 56–57
  instance fields of, 98
  instance lifetime, 36–37
  instance storage, 39
  nullable, 63–64
  parameters and, 71–73
  passing variables by reference, 78–79
  polymorphism, 211–247
  record structs, 34
  reference types vs., xx, 31, 70–71, 123–126, 176–181, 212–213, 273–277
  return type instance modification, 123–124
  sealed, 35, 212–221
  semantics, 45–48
  size of instances, 91–92, 136, 270–274
  structs, 33
variables, 37–39. *See also* array variables; by-reference variables; local variables; parameters; reference variables
  associated types, 38
  avoiding pitfalls of default variables, 200–201
  capturing, 84–85
  copy-by-value semantics, 73–74
  copying, 46–48
  defined, 37
  definite assignment, 39
  embedded, 40–43
  identifiers, 37
  kinds of, 37–38
  lifetime of, 36–37, 131–132
  read-only, 134–136
  values vs., 38–39, 82, 123–127
Velocity type
  abstraction, 17, 28–29, 199
  by-reference parameter limitations, 82
  non-destructive mutation, 90–91
  perils of mutable value types, 126–127
  property methods, 120–123
virtual dispatch, 212, 228
virtual methods, 34, 36, 118
vocabulary, 191–192, 199
Volume type, implementation inheritance and, 228–230

W
Where method, 85, 281
whole numbers, 140–141
widening conversions, 244
with keyword, 60, 90–91, 114–115
WithPercentAdded method, 197–198
WriteLine method, 118

Y
yield statement, 85–86

Z
ZeroKelvin constant, using for validation, 285