APPENDIX

Items Corresponding to Second Edition

Second Edition Item Number	Third Edition Item Number, Title
1	1, Consider static factory methods instead of constructors
2	2, Consider a builder when faced with many constructor parameters
3	3, Enforce the singleton property with a private constructor or an enum type
4	4, Enforce noninstantiability with a private constructor
5	6, Avoid creating unnecessary objects
6	7, Eliminate obsolete object references
7	8, Avoid finalizers and cleaners
8	10, Obey the general contract when overriding `equals`
9	11, Always override `hashCode` when you override `equals`
10	12, Always override `toString`
11	13, Override `clone` judiciously
12	14, Consider implementing `Comparable`
13	15, Minimize the accessibility of classes and members
14	16, In public classes, use accessor methods, not public fields
15	17, Minimize mutability
16	18, Favor composition over inheritance
17	19, Design and document for inheritance or else prohibit it

Second Edition Item Number	Third Edition Item Number, Title
18	20, Prefer interfaces to abstract classes
19	22, Use interfaces only to define types
20	23, Prefer class hierarchies to tagged classes
21	42, Prefer lambdas to anonymous classes
22	24, Favor static member classes over nonstatic
23	26, Don't use raw types
24	27, Eliminate unchecked warnings
25	28, Prefer lists to arrays
26	29, Favor generic types
27	30, Favor generic methods
28	31, Use bounded wildcards to increase API flexibility
29	33, Consider typesafe heterogeneous containers
30	34, Use enums instead of `int` constants
31	35, Use instance fields instead of ordinals
32	36, Use `EnumSet` instead of bit fields
33	37, Use `EnumMap` instead of ordinal indexing
34	38, Emulate extensible enums with interfaces
35	39, Prefer annotations to naming patterns
36	40, Consistently use the `Override` annotation
37	41, Use marker interfaces to define types
38	49, Check parameters for validity
39	50, Make defensive copies when needed
40	51, Design method signatures carefully
41	52, Use overloading judiciously

Second Edition Item Number	**Third Edition Item Number, Title**
42	53, Use varargs judiciously
43	54, Return empty collections or arrays, not nulls
44	56, Write doc comments for all exposed API elements
45	57, Minimize the scope of local variables
46	58, Prefer for-each loops to traditional `for` loops
47	59, Know and use the libraries
48	60, Avoid `float` and `double` if exact answers are required
49	61, Prefer primitive types to boxed primitives
50	62, Avoid strings where other types are more appropriate
51	63, Beware the performance of string concatenation
52	64, Refer to objects by their interfaces
53	65, Prefer interfaces to reflection
54	66, Use native methods judiciously
55	67, Optimize judiciously
56	68, Adhere to generally accepted naming conventions
57	69, Use exceptions only for exceptional conditions
58	70, Use checked exceptions for recoverable conditions and runtime exceptions for programming errors
59	71, Avoid unnecessary use of checked exceptions
60	72, Favor the use of standard exceptions
61	73, Throw exceptions appropriate to the abstraction
62	74, Document all exceptions thrown by each method
63	75, Include failure-capture information in detail messages
64	76, Strive for failure atomicity
65	77, Don’t ignore exceptions

Second Edition Item Number	Third Edition Item Number, Title
66	78, Synchronize access to shared mutable data
67	79, Avoid excessive synchronization
68	80, Prefer executors, tasks, and streams to threads
69	81, 81, Prefer concurrency utilities to `wait` and `notify`
70	82, Document thread safety
71	83, Use lazy initialization judiciously
72	84, Don't depend on the thread scheduler
73	(Retired)
74	85, Prefer alternatives to Java serialization 86, 86, Implement `Serializable` with great caution
75	85, Prefer alternatives to Java serialization 87, Consider using a custom serialized form
76	85, Prefer alternatives to Java serialization 88, Write `readObject` methods defensively
77	85, Prefer alternatives to Java serialization 89, For instance control, prefer enum types to `readResolve`
78	85, Prefer alternatives to Java serialization 90, Consider serialization proxies instead of serialized instances

References

[Asserts] *Programming with Assertions*. 2002. Sun Microsystems. http://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

[Beck04] Beck, Kent. 2004. *JUnit Pocket Guide*. Sebastopol, CA: O'Reilly Media, Inc. ISBN: 0596007434.

[Bloch01] Bloch, Joshua. 2001. *Effective Java Programming Language Guide.* Boston: Addison-Wesley. ISBN: 0201310058.

[Bloch05] Bloch, Joshua, and Neal Gafter. 2005. *Java Puzzlers: Traps, Pitfalls, and Corner Cases.* Boston: Addison-Wesley. ISBN: 032133678X.

[Blum14] Blum, Scott. 2014. "Faster RSA in Java with GMP." *The Square Corner* (blog). Feb. 14, 2014. https://medium.com/square-corner-blog/faster-rsa-in-java-with-gmp-8b13c51c6ec4

[Bracha04] Bracha, Gilad. 2004. "Lesson: Generics" online supplement to *The Java Tutorial: A Short Course on the Basics,* 6th ed. Upper Saddle River, NJ: Addison-Wesley, 2014. https://docs.oracle.com/javase/tutorial/extra/generics/

[Burn01] Burn, Oliver. 2001–2017. *Checkstyle.* http://checkstyle.sourceforge.net

[Coekaerts15] Coekaerts, Wouter (@WouterCoekaerts). 2015. "Billion-laughs-style DoS for Java serialization https://gist.github.com/coekie/a27cc406fc9f3dc7a70d … WONTFIX," Twitter, November 9, 2015, 9:46 a.m. https://twitter.com/woutercoekaerts/status/663774695381078016

[CompSci17] Brief of Computer Scientists as Amici Curiae for the United States Court of Appeals for the Federal Circuit, Case No. 17-1118, Oracle America, Inc. v. Google, Inc. in Support of Defendant-Appellee. (2017)

[Dagger] *Dagger*. 2013. Square, Inc. http://square.github.io/dagger/

[Gallagher16] Gallagher, Sean. 2016. "Muni system hacker hit others by scanning for year-old Java vulnerability." *Ars Technica,* November 29, 2016. https://arstechnica.com/information-technology/2016/11/san-francisco-transit-ransomware-attacker-likely-used-year-old-java-exploit/

[Gamma95] Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* Reading, MA: Addison-Wesley. ISBN: 0201633612.

[Goetz06] Goetz, Brian. 2006. *Java Concurrency in Practice*. With Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea. Boston: Addison-Wesley. ISBN: 0321349601.

[Gosling97] Gosling, James. 1997. "The Feel of Java." *Computer* 30 no. 6 (June 1997): 53-57. http://dx.doi.org/10.1109/2.587548

[Guava] *Guava*. 2017. Google Inc. https://github.com/google/guava

[Guice] *Guice*. 2006. Google Inc. https://github.com/google/guice

[Herlihy12] Herlihy, Maurice, and Nir Shavit. 2012. *The Art of Multiprocessor Programming, Revised Reprint.* Waltham, MA: Morgan Kaufmann Publishers. ISBN: 0123973376.

[Jackson75] Jackson, M. A. 1975. *Principles of Program Design.* London: Academic Press. ISBN: 0123790506.

[Java-secure] *Secure Coding Guidelines for Java SE*. 2017. Oracle. http://www.oracle.com/technetwork/java/seccodeguide-139067.html

[Java8-feat] *What's New in JDK 8*. 2014. Oracle. http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html

[Java9-feat] *Java Platform, Standard Edition What's New in Oracle JDK 9*. 2017. Oracle. https://docs.oracle.com/javase/9/whatsnew/toc.htm

[Java9-api] *Java Platform, Standard Edition & Java Development Kit Version 9 API Specification*. 2017. Oracle. https://docs.oracle.com/javase/9/docs/api/overview-summary.html

[Javadoc-guide] *How to Write Doc Comments for the Javadoc Tool*. 2000–2004. Sun Microsystems. http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

[Javadoc-ref] *Javadoc Reference Guide.* 2014-2017. Oracle. https://docs.oracle.com/javase/9/javadoc/javadoc.htm

[JLS] Gosling, James, Bill Joy, Guy Steele, and Gilad Bracha. 2014. *The Java Language Specification, Java SE 8 Edition*. Boston: Addison-Wesley. ISBN: 013390069X.

[JMH] *Code Tools: `jmh`.* 2014. Oracle. http://openjdk.java.net/projects/code-tools/jmh/

[JSON] *Introducing JSON*. 2013. Ecma International. https://www.json.org

[Kahan91] Kahan, William, and J. W. Thomas. 1991. *Augmenting a Programming Language with Complex Arithmetic*. UCB/CSD-91-667, University of California, Berkeley.

[Knuth74] Knuth, Donald. 1974. Structured Programming with `go to` Statements. In *Computing Surveys* 6: 261–301.

[Lea14] Lea, Doug. 2014. *When to use parallel streams*. http://gee.cs.oswego.edu/dl/html/StreamParallelGuidance.html

[Lieberman86] Lieberman, Henry. 1986. Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems. In *Proceedings of the First ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 214–223, Portland, September 1986. ACM Press.

[Liskov87] Liskov, B. 1988. Data Abstraction and Hierarchy. In *Addendum to the Proceedings of OOPSLA '87* and *SIGPLAN Notices,* Vol. 23, No. 5: 17–34, May 1988.

[Naftalin07] Naftalin, Maurice, and Philip Wadler. 2007. *Java Generics and Collections*. Sebastopol, CA: O'Reilly Media, Inc. ISBN: 0596527756.

[Parnas72] Parnas, D. L. 1972. On the Criteria to Be Used in Decomposing Systems into Modules. In *Communications of the ACM* 15: 1053–1058.

[POSIX] 9945-1:1996 (ISO/IEC) [IEEE/ANSI Std. 1003.1 1995 Edition] Information Technology—Portable Operating System Interface (POSIX)—Part 1: System Application: Program Interface (API) C Language] (ANSI), IEEE Standards Press, ISBN: 1559375736.

[Protobuf] *Protocol Buffers*. 2017. Google Inc. https://developers.google.com/protocol-buffers

[Schneider16] Schneider, Christian. 2016. SWAT (Serial Whitelist Application Trainer). https://github.com/cschneider4711/SWAT/

[Seacord17] Seacord, Robert. 2017. *Combating Java Deserialization Vulnerabilities with Look-Ahead Object Input Streams (LAOIS)*. San Francisco: NCC Group Whitepaper. https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2017/june/ncc_group_combating_java_deserialization_vulnerabilities_with_look-ahead_object_input_streams1.pdf

[Serialization] *Java Object Serialization Specification*. March 2005. Sun Microsystems. http://docs.oracle.com/javase/9/docs/specs/serialization/index.html

[Sestoft16] Sestoft, Peter. 2016. *Java Precisely*, 3rd ed. Cambridge, MA: The MIT Press. ISBN: 0262529076.

[Shipilëv16] Aleksey Shipilëv. 2016. *Arrays of Wisdom of the Ancients.* https://shipilev.net/blog/2016/arrays-wisdom-ancients/

[Smith62] Smith, Robert. 1962. Algorithm 116 Complex Division. In *Communications of the ACM* 5, no. 8 (August 1962): 435.

[Snyder86] Snyder, Alan. 1986. "Encapsulation and Inheritance in Object-Oriented Programming Languages." In *Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings*, 38–45. New York, NY: ACM Press.

[Spring] *Spring Framework*. Pivotal Software, Inc. 2017. https://projects.spring.io/spring-framework/

[Stroustrup] Stroustrup, Bjarne. [ca. 2000]. "Is Java the language you would have designed if you didn't have to be compatible with C?" *Bjarne Stroustrup's FAQ*. Updated Ocober 1, 2017. http://www.stroustrup.com/bs_faq.html#Java

[Stroustrup95] Stroustrup, Bjarne. 1995. "Why C++ is not just an object-oriented programming language." In *Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications*, edited by Steven Craig Bilow and Patricia S. Bilow New York, NY: ACM. http://dx.doi.org/10.1145/260094.260207

[Svoboda16] Svoboda, David. 2016. *Exploiting Java Serialization for Fun and Profit*. Software Engineering Institute, Carnegie Mellon University. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484347

[Thomas94] Thomas, Jim, and Jerome T. Coonen. 1994. "Issues Regarding Imaginary Types for C and C++." In *The Journal of C Language Translation* 5, no. 3 (March 1994): 134–138.

[ThreadStop] *Why Are* `Thread.stop`*,* `Thread.suspend`*,* `Thread.resume` *and* `Runtime.runFinalizersOnExit` *Deprecated?* 1999. Sun Microsystems. https://docs.oracle.com/javase/8/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html

[Viega01] Viega, John, and Gary McGraw. 2001. *Building Secure Software: How to Avoid Security Problems the Right Way.* Boston: Addison-Wesley. ISBN: 020172152X.

[W3C-validator] *W3C Markup Validation Service*. 2007. World Wide Web Consortium. http://validator.w3.org/

[Wulf72] Wulf, W. A Case Against the GOTO. 1972. In *Proceedings of the 25th ACM National Conference* 2: 791–797. New York, NY: ACM Press.

Index

Symbols

_ in numeric literals, 108
`...` variable arity argument
  See varargs
< > type parameter delimiters, 117, 123
<?> unbounded wildcard type, 120

A

abstract classes
  designing for inheritance, 97
  vs. interfaces, 99–103
  noninstantiability and, 19
  subclassing, and `equals`, 45
access control mechanisms, 74
access levels, 73–77
  module-level, 76–77
  of `readResolve`, 362
  rules of thumb for, 74–75
  of static member classes, 112
access modifiers, 74
accessor methods, 78
  defensive copies and, 80, 233
  for failure-capture data, 297, 307
  immutability and, 80
  naming conventions for, 291–292
  vs. public fields, 78–79
  for `toString` data, 57
actual type parameters, 117
Adapter pattern, 23, 101, 113
aggregate types vs. strings, 276
alien methods, 317
annotations, 157, 180–192
  defining, 180
  detecting repeated and non-repeated, 186
  documenting, 259
  functional interfaces and, 202
  vs. naming patterns, 180–187
  synthetic, 186
  See also marker annotations
anonymous classes, 112, 114, 193
  in adapters, 101
  prefer lambdas to, 193–196
  serialization and, 196
antipatterns, 2
  bit field enum pattern, 169
  bounded wildcard types as return types, 142
  breaking from naming conventions, 82
  busy waits, 336
  constant as a `hashCode`, 51
  constant interface, 107–108
  copy constructor of immutable object, 83
  difference-based comparators, 71
  empty catch blocks, 310
  `equals` with unreliable resources, 45
  exceptions for flow control, 293
  excessive string concatenation, 279
  floating point for monetary calculations, 270
  hardwiring resources, 20
  `hashCode` missing significant fields, 54
  immutable classes not effectively final, 85
  inappropriate subclassing, 92
  `int` enum pattern, 157
  naming patterns, 180
  null returns for collections, arrays, 247–248
  ordinal abuse, 168, 171
  overriding `equals` but not `hashCode`, 50
  serializable inner classes, 345
  signatures differ by parameter type order, 6
  `String` enum pattern, 158
  string overuse, 276
  tagged classes, 109–111
  value-component subclasses and `equals`, 44

API design
  access levels and, 74
  bounded wildcard types and, 139–145
  callbacks, 28
  constant interface pattern and, 107
  exceptions and, 294, 296–297
  information hiding and, 286
  inheritance and, 93–98
  interfaces as parameter types, 170
  member classes and, 114
  performance and, 286–287
  serialization and, 343–345
  singletons, 18
API elements, 4
  documenting, 254–260
API, `toString` return values as defacto, 57
arrays
  `clone` and, 65
  covariant typing, 126
  defensive copying of, 76, 234
  empty, vs. `null` as return value, 247–248
  to implement generics, 131–133
  vs. lists, 126–129
  mutability and, 234, 248
  reified, 126
  security issues, 76
assertions, 229
atomicity
  of variables, 311
  synchronization and, 312–314
autoboxing, 24, 273–275
  performance and, 275
`AutoCloseable` interface, 31–32, 35

B

backing objects, 23
base classes, 281
`BigDecimal` class
  `compareTo` inconsistent with `equals`, 68
  for monetary calculations, 270
  performance and, 271
bit fields vs. enum sets, 169–170
blocking operations, 326
bogus byte stream attacks, 354
`boolean` vs. enum types, 237
bounded type parameters, 134
  for bounded type tokens, 154
  vs. bounded wildcard types, 144
bounded type tokens, 154, 172, 178, 183
bounded wildcard types, 136, 140
  for API flexibility, 139–145
  vs. bounded type parameters, 144
  for bounded type tokens, 154
  vs. class objects, 178
  dependency injection and, 21
  PECS mnemonic for, 141
  as return types, 142
  vs. unbounded wildcard types, 121
boxed primitives
  == operator and, 274
  appropriate uses of, 275
  generics and, 134
  prefer primitive types to, 24, 273–275
Bridge pattern, 8
Builder pattern, 10–16
  adapted for method invocation, 237
busy waits, 336

C

caching
  avoiding memory leaks from, 28
  of expensive objects, 22–23
  of hash codes, 53
  immutable objects and, 82, 85
callback frameworks, wrapper classes and, 91
callbacks, avoiding memory leaks from, 28
canonical forms, 47
capabilities vs. strings, 276–277
casts
  dynamic, 153, 155
  invisible (*see* compiler-generated casts)
  unchecked, warnings of, 127, 129, 137
`char` values, and streams, 206
checked exceptions
  avoiding overuse of, 298–299
  declaring, 304
  failure atomicity and, 308
  purpose of, 296
  refactoring to unchecked, 299
  vs. unchecked, 296–297

circularities
in cleaners, 33
initialization, 333, 366
serialization attacks and, 360
`Class` class, as parameterized key, 151
class hierarchies, 110
Builder pattern and, 14
combinatorial explosions in, 100
class literals
as annotation parameter values, 183
as generics, 151
raw types in, 121
class-based frameworks, 281
classes, 73–114
access levels of, 74
anonymous (*see* anonymous classes)
base, 281
composition, 87–92
designing for inheritance, 93–98
documenting
for inheritance, 93–94
thread safety of, 330–332
generic, 117
helper, for shortening parameter lists, 237
hierarchy of (*see* class hierarchies)
immutable (*see* immutable objects)
implementation inheritance, 87–92
instances of, 3
levels of thread safety, 330
members, 3
minimizing accessibility of, 73–77
mutable, and thread-safety, 322
naming conventions for, 289–291
nested (*see* nested classes)
noninstantiable companions, 7
reusable forwarding, 89–91
singletons (*see* singletons)
summary descriptions of, 257
`SuppressWarnings` annotation and, 124
tagged, vs. class hierarchies, 109–111
unintentionally instantiable, 19
unrelated, 243
utility (*see* utility classes)
wrapper (*see* wrapper classes)
See also individual class names
classifier functions, 213
cleaners, 29–33
`clone` method, 58–65
arrays and, 65
as a constructor, 61, 96
vs. copy or conversion constructor, 65
defensive copies and, 76, 233
final fields and, 61
general contract, 58–59
immutable objects and, 83
nonfinal classes and, 233
nonfinal methods and, 64
overridable methods and, 96
thread-safety and, 64
`Cloneable` interface, 58–65
alternatives to, 65
behavior of, 58
designing for inheritance and, 64, 96
implementing, 64
collections
concurrent, 321, 325–326
empty, vs. null as return value, 247
optionals and, 252
as return types, vs. streams, 216–221
collectors, 211–215
downstream, 213
`Collectors` API, organization of, 211–215
combinatorial explosions, 100
companion classes
mutable, 84
noninstantiable, 7
`Comparable` interface, 66–72
as consumers in PECS, 143
recursive type bounds and, 137
See also `compareTo` method
comparator construction methods, 70, 194
comparators, 69
as consumers in PECS, 143
`compareTo` method, 66–68
See also `Comparable` interface
compatibility
backward, 350
binary, 107, 305
forward, 350
migration, 119
source, 305
unchecked exceptions and, 305
compiler warnings, types of, 123

compiler-generated casts, 117, 119, 127
components, 2
composition, 8, 89
  `equals` and, 44
  vs. inheritance, 87–92
conceptual weight, 7
concurrency, 311
  documenting method behavior for, 330–332
  improving via internal synchronization, 322
concurrency utilities, 323–329
  vs. `wait` and `notify`, 325–329
concurrent collections, 321, 325–326
conditionally thread-safe classes, documenting, 331
consistency requirements
  `equals`, 38, 45
  `hashCode`, 50
consistent with `equals`, 68
  unreliable resources and, 45
constant fields, naming conventions for, 290
constant interfaces, 107
constant utility classes, 108
constants, 76
  in anonymous classes, 114
  naming conventions for, 290
constant-specific behaviors, 162–166
  lambdas for, 195
constant-specific class bodies, 162
constant-specific method implementations
  See constant-specific behaviors
constructors, 4
  calling overridable methods in, 95
  checking parameters of, 353
  `clone` as a, 61
  copy and conversion, 65
  default, 19
  defensive copying of parameters, 232
  deserialization as, 344
  establishing invariants, 82, 86
  noninstantiability and, 19
  private (*see* private constructors)
  `readObject` as a, 353
  reflection and, 282
  replacing with static factories, 5–9
  safely overloading, 240–241
  for singletons, 17–18
  summary descriptions of, 257
  `SuppressWarnings` annotation and, 124
  validity checking parameters of, 229
contention, synchronization and, 321
contracts
  `clone`, 58
  `compareTo`, 66
  documentation as, 304
  `equals`, 38–46
  `hashCode`, 50
  `toString`, 55
corrupted objects, 12, 30, 227, 309
countdown latches, 326
covariant arrays, 126
covariant return typing, 16, 60
creating objects, 5–33
cross-platform structured-data representations, 341
custom serialized forms, 346–352

D

data consistency
  maintaining in face of failure, 308–309
  synchronization for, 311–316
data corruption, 285, 312
`Date`, replacements for, 232
deadlocks
  resource ordering, 320, 351
  thread starvation, 328
Decorator pattern, 91
default access
  See package-private access level
default constructors, 19
default implementations, 104
default methods on interfaces, 99, 104–105
default serialized forms, 346–352
  disadvantages of, 348
defensive copies, 231–235
  of arrays, 234
  builders and, 14
  `clone` and, 233
  deserialization and, 353, 357
  documenting, 234
  immutable objects and, 83

of mutable internal fields, 233
of mutable parameters, 232–233
vs. object reuse, 25
performance and, 234
`readObject` and, 353, 357
transfers of control and, 235
validity checking and, 232
delegation, 91
denial-of-service attacks, 331
dependency injection, 20–21
derived fields, 47, 52
deserialization
as a constructor, 344
singletons and, 18
deserialization bombs, 340
deserialization filtering, 342
destroying objects, 26–33
detail messages, 306
diamond operator, 123
documenting
annotation types, 259
`compareTo`, 67
conditional thread safety, 331
enum types, 258, 331
exceptions, 227, 304–305
exposed API elements, 254–260
generics, 258
`hashCode`, 54
for inheritance, 93–94
methods, 254–255
multiline code examples, 255
object state after exceptions, 309
parameters, 227
return value of `toString`, 56
self-use of overridable methods, 93, 98
self-use patterns, 256
serialized fields, 347
skeletal implementations, 103
static factories, 9
`SuppressWarnings` annotation, 125
thread safety, 330–332
`writeObject` for serialization, 350
See also Javadoc
`double`
for binary floating-point arithmetic, 270
when to avoid, 270–272
double-check idiom, 334–335
downstream collectors, 213
dynamic casts, 153, 155

E

effectively immutable objects, 316
empty arrays, vs. null as return value, 247–248
encapsulation, 73, 286
broken by inheritance, 87, 94
broken by serialization, 343
of data fields, 78
enclosing instances, 112
enum types, 157–192
adding data and behaviors to, 159–161
vs. bit fields, 169–170
vs. `boolean`, 237
built-in serialization mechanism, 362
constant-specifics for, 162–166
documenting, 258, 331
extensibility and, 176–179
immutability of, 160
instance-controlled, 158
vs. `int` constants, 157–167
prefer to `readResolve`, 359–362
removing elements from, 161
singletons from, 17–18
strategy enum pattern, 166
vs. `String` constants, 158, 276
`switch` statements and, 167
as top-level or member classes, 161
`toString` and, 160
type safety from, 158
when to use, 167
enumerated types
See enum types
`EnumMap` vs. ordinals, 171–175
`EnumSet` vs. bit fields, 169–170
`equals` method, 37
accidental overloading of, 49, 188
canonical forms and, 47
`compareTo` and, 68
composition and, 44
general contract for, 38–46
`hashCode` and, 48, 50–54
how to write, 46
`Override` annotation and, 188

`equals` method (*continued*)
return values of, and `compareTo`, 68
subclassing and, 42, 45
unreliable resources and, 45
when to override, 37–38
equivalence classes, 39
equivalence relations, 38
erasure, 119, 126
errors
generic array creation, 126–127, 133
purpose of, 297
runtime, default methods and, 105
exact results, types for obtaining, 270
exception chaining, 302–303
exception translation idiom, 230, 302
exceptions, 293–310
accessor methods for, 297, 307
checked vs. unchecked, 296–297
choosing among, 301
commonly reused, 301
control flow and, 294
detail messages for, 306–307
documenting, 227, 304–305
failure-capture data, 307
for invalid method parameters, 228
ignoring, 310
logging of, 303
multi-catch facility, 320
vs. optionals or special return values, 295
prefer standard existing, 300–301
preventing, 303
vs. state testing methods, 294
suppression of, 36
uncaught, and finalizers, 30
using appropriately, 293–295
See also individual exception names
Executor Framework, 323
executor service, 323–324
explicit type arguments, 142
export declarations, 76
exported APIs
See API design; APIs
extending classes
See inheritance; subclassing
extending interfaces, 4
extensible enums, 176–179
extralinguistic mechanisms
cloning, 58, 65
native methods, 285
reflection, 282
serialization, 344, 363
See also hidden constructors

F

Factory Method pattern, 5, 21
failure atomicity, 230, 308–309
fields
access levels of, 73–77
class invariants and, 75
constant, naming conventions for, 290
derived, 47, 52
exposing, vs. accessor methods, 78–79
final (*see* final fields)
initialization techniques for, 335
mutable, defensive copies of, 233
naming conventions for, 290, 292
public static final, for singletons, 17
reflection and, 282
summary descriptions of, 257
tags, 109
thread safety and, 75
final fields
for defining constants, 290
incompatible with cloning, 61
incompatible with serialization, 357
finalizer attacks, and prevention, 30–31
finalizers, 29–33
alternative to, 31
`float`
for binary floating-point arithmetic, 270
when to avoid, 270–272
fluent APIs, 14, 203
Flyweight pattern, 6
footprint
See space consumption
`for` loops
dual variable idiom, 263
prefer for-each loops to, 264–266
vs. `while` loops, 262
for-each loops
limitations of, 266
prefer over `for` loops, 264–266

fork-join tasks and pools, 324
formal type parameters, 117
forwarding methods, 89, 102
frameworks
  callback, 91
  class-based, 281
  executor, 323
  interface-based, 6
  nonhierarchical type, 99
  service provider, 8
function objects, 114
  vs. code blocks, 207
functional interfaces, 193
  method overloading and, 243
  organization of standard, 200–201
  using standard, 199–202
functional programming, 82

G

gadgets, 340
garbage collection, 27, 29–30, 113
general contracts
  See contracts
generic array creation errors, 126–127, 133
generic classes and interfaces, 117
generic methods, 135–138
  vs. unbounded wildcard types, 121
generic singleton factories, 18, 136
generic type parameters
  See type parameters
generic types, 14, 117, 130–134
  documenting, 258
  immutability and, 136
generic varargs parameter arrays
  heap pollution from, 147–148
  replacing with lists, 149
  unsafe as storage, 146
  unsafe to expose, 147–148
generics, 117–155
  boxed primitives and, 134
  compiler-generated casts and, 117
  erasure and, 126
  implementing atop arrays, 131–133
  incompatibility with primitive types, 134
  invariant typing, 126
  varargs and, 127, 146–150
generifying existing code, 130
Get and Put Principle, 141

H

`hashCode` method
  `equals` and, 48, 50–54
  general contract for, 50
  how to write, 51
  immutable objects and, 53
heap pollution, 133, 146–148
heap profilers, 28
helper classes, 112
  for shortening parameter lists, 237
hidden constructors, 61, 96, 339, 344, 353
  See also extralinguistic mechanisms
hierarchical builder pattern, 16

I

immutable objects
  canonical forms and, 47
  `clone` and, 59
  dependency injection and, 21
  empty arrays and, 248
  enum types and, 160
  `EnumSet`s and, 170
  failure atomicity and, 308
  functional approach and, 82
  `hashCode` and, 53
  JavaBeans and, 12
  mutable companion classes for, 84
  object reuse and, 22
  rules for, 80
  serialization and, 85, 353–358
  static factory methods and, 84
  subclassing and, 97
  thread safety and, 82
imperative programming, 82
implementation details
  documenting for inheritance, 94
  exposing, 92
implementation inheritance, 87
implementing interfaces, 4
  default methods and, 105
inconsistent with `equals`, 67–68
  unreliable resources and, 45
information hiding
  See encapsulation

inheritance, 3
  vs. composition, 87–92
  constructors and, 95
  designing for, 93–98
  documenting for, 93–94
  encapsulation and, 87
  fragility of, 89
  hooks to facilitate, 94
  implementation vs. interface, 3, 87
  of method doc comments, 259
  multiple, simulated, 102
  self-use of overridable methods and, 98
  uses of, 92
  See also subclassing
initialization
  circularities, 333, 366
  defensive copying and, 80
  of fields on deserialization, 351
  incomplete, 96
  lazy (*see* lazy initialization)
  of local variables, 261
  normal vs. lazy, 333
  at object creation, 86
inner classes, 112
  `Serializable` and, 345
  to extend skeletal implementations, 102
instance fields
  access levels of, 75
  initializing, 333
  lazy initialization of, 334
  vs. ordinals, 168
instance-controlled classes, 6, 158
  singletons, 17–18
  static factory methods and, 6
  utility classes, 19
  See also enum types
`instanceof` operator, parameter types, 121
`int` constants vs. enum types, 157–167
`int`, for monetary calculations, 270
interface-based frameworks, 6, 99–103
interface inheritance, 87
interfaces, 73–114
  vs. abstract classes, 99–103
  access levels of, 74
  accessibility of static members, 7
  default methods on, 99, 104–105
  for defining types, 107–108, 191–192
  design of, 104–106
  emulating extensible enums with, 176–179
  enabling functionality enhancements, 100
  generic, 117
  marker (*see* marker interfaces)
  mixin, 58, 99
  naming conventions for, 289–291
  for nonhierarchical type frameworks, 99
  noninstantiable companion classes and, 7
  as parameter types, 170, 237
  prefer to reflection, 282
  purpose of, 58, 107–108
  for referring to objects, 280–281
  reflective instantiation of, 283–284
  serialization and, 344
  skeletal implementations and, 100–103
  static methods and, 7
  summary descriptions of, 257
  See also individual interface names
internal field theft attacks, 360–362
invariant types, 126, 139
invariants
  `clone` and, 61
  concurrency and, 328
  constructors and, 82, 86
  corruption of, 92
  enum types and, 362
  maintaining, 229, 234, 308
  of objects and members, 75, 78

J

JavaBeans
  immutability and, 12
  method-naming conventions, 291
  pattern, 11–12
Javadoc, 254
  architecture documents and, 260
  class-level comments, 228, 331
  client-side indexes in, 258
  comment inheritance, 259
  formatting, 255–256
  module- and package-level comments, 259
  summary descriptions, 257

K

key extractor functions, 70

L

lambdas, 70, 193–225
  cleaners and, 33
  for constant-specific behaviors, 195
  prefer method references to, 197–198
  prefer to anonymous classes, 193–196
  serialization and, 196
lazy initialization, 23, 53, 85, 333–335
lazy initialization holder class idiom, 334–335
lazy initialization with a synchronized accessor idiom, 333–334
leaky abstractions, 146
libraries, 267–269
Liskov substitution principle, 43, 75
listeners, avoiding memory leaks from, 28
lists
  vs. arrays, 126–129
  for generic varargs parameter arrays, 149
  mutual comparability in, 138
liveness
  ensuring, 317–322, 328
  failures of, 224, 313
local classes, 112, 114
local variables
  minimizing scope of, 261–263
  naming conventions for, 290, 292
locality of reference, 223
locks
  fields containing, 332
  finalizers or cleaners and, 30
  private, 332
  reentrant, 320
logical equality, 37–38
`long`, for monetary calculations, 270
loops
  nested, 264–266
  See also `for` loops; for-each loops

M

maps
  member classes and, 113
  nested, 173–175
  vs. streams, behavior of, 173
marker annotations, 181
  vs. marker interfaces, 191
marker interfaces, 191–192
member classes, 112–114
  See also static member classes
members, 3
  minimizing accessibility of, 73–77
memory footprint
  See space consumption
memory leaks, 26–27
  nonstatic member classes and, 113
  self-management of memory, 28
memory model, 80
merge functions, 212
meta-annotations, 181
method chaining, 14
method overloading, 238–244
  accidental, of `equals`, 49
  effects of autoboxing and generics, 241–242
  functional interfaces and, 202, 243
  parameters and, 240
  static selection among methods, 238
method overriding, 49, 238–239
  access levels and, 75
  `clone`, 58
  dynamic selection among methods, 238
  `equals`, 37–49
  `hashCode`, 50–54
  self-use and, 98
  `toString`, 55–57
  unintentional, 190
method references, 18
  kinds of, 198
  prefer to lambdas, 197–198
methods, 3, 227–260
  access levels of, 74
  accessor (*see* accessor methods)
  alien, 317
  common to all objects, 37–72
  constant-specific, for enum-types, 162
  documenting, 254–255
    exceptions thrown by, 304–305
    overridable, 93
    summary descriptions of, 257
    thread safety of, 330–332
  failure atomicity and, 308–309
  forwarding (*see* forwarding methods)
  generic, 121, 135–138
  invocation, reflection and, 282
  legal for `SafeVarargs`, 149

methods (*continued*)
minimizing accessibility of, 73
naming conventions for, 9, 290–291
native, 31, 285
nonfinal, and `clone`, 64
overloading (*see* method overloading)
overriding (*see* method overriding)
parameter lists for, 236
private, to capture wildcard types, 145
shortening parameter lists of, 236
signatures of, 3, 236–237
size of, 263
state-testing, vs. special return value, 295
static factory (*see* static factory methods)
`SuppressWarnings` annotation and, 124
validity checking parameters, 227–230
varargs, 245–246
See also individual method names
mixin interfaces, 58, 99
mixing primitives, boxed primitives, 24, 274
modules, 76–77
monetary calculations, types for, 270–271
Monty Python reference, subtle, 247
multi-catch facility, 320
multiple inheritance, simulated, 102
mutability
JavaBeans pattern and, 11
minimizing, 80–86
mutable companion classes, 84
mutable reductions, 223
mutators, 78
mutual comparability, 138
mutual exclusion, 311

N

named optional parameters, 14
naming conventions, 236, 289–292
of generic type parameters, 131
grammatical, 291–292
of skeletal implementation classes, 101
of static factory methods, 9
streams and, 208
of type parameters, 135
naming patterns vs. annotations, 180–187
native methods, 31, 285
native peers, 31
natural ordering, 66
nested classes, 112
access levels of, 74
decreasing accessibility with, 74
in serialization proxy pattern, 363
types of, 112
nested interfaces, access levels of, 74
nested maps, 173–175
nonhierarchical type frameworks, 99
noninstantiable classes, 19
noninstantiable companion classes, 7
non-nullity of `equals`, 38, 45
non-reifiable types, 127, 131, 146
nonstatic member classes, 112–114
`notify` vs. `notifyAll`, 328–329
null checking, 228
nulling out obsolete object references, 27
`NullPointerException`, `equals` contract and, 46

O

object pools, 24
object reference fields, `equals` and, 47
objects, 3
avoiding reflective access, 282–284
base classes and, 281
creating and destroying, 5–33
creation and performance, 6, 22–23
deserialization filtering of, 342
effectively immutable, 316
eliminating obsolete references to, 26–28, 60, 308
expense of creating, 24
favor refering to by interfaces, 280–281
function, 114
immutable (*see* immutable objects)
in inconsistent states, 11–12, 96, 309 (*see also* corrupted objects)
methods common to all, 37–72
nulling out obsolete references to, 27
process, 114
reuse, 22–25
safe publication of, 316
string representations of, 55–57
when to refer to by class, 281
Observer pattern, 317
obsolete object references, 26–28, 60, 308
open calls, 321

optimizations, 286–288
  caching hash codes, 53
  lazy initialization, 333–335
  `notify` instead of `notifyAll`, 329
  object reuse, 22–25
  order of comparisons in `equals`, 47
  parallelizing streams, 224
  static initialization, 23
  `StringBuffer` and, 279
  using == in `equals`, 46
optionals, 249
  exceptions and, 295, 298
  as return values, 249–253
ordinals
  vs. enum maps, 171–175
  vs. instance fields, 168
overloading
  See method overloading
`Override` annotations, 49, 188–190
overriding
  See method overriding

P

package-private access level, 4, 74, 84
packages, naming conventions for, 289–290
parallelizing streams, 222–225
parameter lists
  of builders, 14
  shortening, 236–237
  varargs and, 245–246
parameterized types, 117–122
  reifiable, 127
parameterless constructors, 19
parameters
  defensive copies of mutable, 232
  type (*see* type parameters)
  validity checking of, 227–230, 353–355
PECS mnemonic, 141
performance, 286–288
  autoboxing and, 24, 201, 275
  `BigDecimal` and, 271
  builder pattern, 16
  cleaners, 29–30
  defensive copying and, 234
  of enums, 167, 170
  of `equals`, 46–47
  of excessive synchronization, 321
  finalizers, 29–30
  for-each loops and, 264
  of `hashCode`, 50, 53
  immutable classes and, 83–85
  libraries and, 268
  measuring, 287
  memory leaks and, 27
  native methods and, 285
  object creation and, 6, 22–23
  of reflection, 282
  parallelizing streams and, 222–225
  of serialization, 348–350
  software architecture and, 286–287
  state-testing vs. special return value, 295
  static factories and, 6
  of string concatenation, 279
  `toString` and, 57
  varargs and, 246
  wrapper classes and, 91
  See also optimizations
performance model, 288
portability
  cleaners and, 29
  finalizers and, 29
  native methods and, 285
  thread priorities and, 337
  thread scheduler and, 336
predicates, 104
primitive fields
  `compareTo` and, 69
  `equals` and, 47
primitive types, 273
  incompatibility with generic types, 134
  optionals and, 253
  prefer over boxed primitives, 24, 273–275
  See also individual primitive types
private access level, 74
private constructors, 84
  for noninstantiability, 19
  for singletons, 17–18
private lock object idiom, 332
private lock objects, 332
procedural programming, 82
process objects, 114
producer-consumer queues, 326
programming principles, 2
promptness of finalization, 29

protected access level, 74–75
public access level, 74
public fields vs. accessor methods, 78–79
publicly accessible locks, 331

Q

qualified `this` construct, 112

R

racy single check idiom, 335
range checking, 229
raw types, 117–122
`readObject` method, 353–358
  defensive copies and, 80
  how to write, 358
  incompatible with instance-controlled objects, 359
  overridable methods and, 96, 358
`readResolve` method
  access levels of, 97
  choosing access levels of, 362
  prefer enum types to, 359–362
  using for instance-controlled classes, 359
recipes
  adding behaviors to individual enum constants, 162
  adding data to enum types, 160
  builder pattern, 12
  checking significant fields in `equals`, 47
  `clone`, 64
  `compareTo`, 68
  eliminating self-use, 98
  `equals`, 46
  generifying a class, 130–133
  `hashCode`, 51
  implementing generics atop arrays, 131–133
  method chaining, 14
  noninstantiable classes, 19
  `readObject`, 358
  serialization proxies, 363–364
  serialized singletons, 18
  singletons as single-element enums, 18
  singletons with private constructors, 17
  skeletal implementations, 102
  tagged classes to class hierarchies, 110–111
  See also rules
recursive type bounds, 137
recursive type parameters, 14
reduction strategy, 211
reductions, 223
reentrant locks, 320
reference types, 3, 273
reflection, 282–284
  `AccessibleObject.setAccessible` attacks, 17
  `clone` and, 58
  drawbacks of, 282
  reflective interface instantiation, 283
  uses for, 282, 284
reflexivity requirements
  `compareTo`, 68
  `equals`, 38–39
reified types, 126
resource factories, 21
resource-ordering deadlocks, 351
resources
  locked, and finalizers, 30
  releasing, 31
restricted marker interfaces, 191
return classes, varied
  serialization proxy pattern and, 365
  static factory methods and, 7–8
`return` statements, `SuppressWarnings` annotation and, 124
return types
  bounded wildcard types as, 142
  collections vs. streams, 216–221
  static factory methods and, 6
reusable forwarding classes, 89–91
rules
  accessibility, 74–75
  appropriateness of checked exceptions, 298
  choosing bounded wildcard types, 141
  choosing exception types, 296–297
  decreasing serialization dangers, 341–342
  for immutable objects, 80
  mapping domains to package names, 289
  marker interfaces vs. annotations, 192
  optimization, 286
  for performance of parallel streams, 223
  replacing type parameters with wildcards, 144

for SafeVarargs annotations, 149
static members accessibility, 112
writing doc comments, 254–260
runtime exceptions
See unchecked exceptions

S

safe array accesses, 76
safe languages, 231
safe publication, 316
safety failures, 315
parallel streams and, 224
wait and, 328
SafeVarargs annotations, 147
legal uses of, 149
scope
local variables, 261–263
SuppressWarnings annotations, 124
of variables, obsolete references and, 27
security, defensive copying for, 25, 231
security issues
accessible nonzero-length arrays, 76
AccessibleObject.setAccessible attacks, 17
denial of service attacks, 331
deserialization bombs, 339–340
ElvisStealer attacks, 362
finalizer attacks, 30–31
gadgets, 340
internal field theft attacks, 355–357
ransomware attacks, 339
reflection, 17
remote code execution, 340
rogue object reference attacks, 355–357
serialization, 339, 344, 353, 360
stealer attacks, 362
strings as keys for granting data access, 277
subclassing and, 89
time-of-check/time-of-use (TOCTOU) attacks, 233
SELF problem, 91
self-use
documenting, for inheritance, 93
eliminating, for inheritance, 98
serial version UIDs, 343, 351–352
Serializable, 343–345
serialization, 339–366
anonymous classes and, 196
costs of, 343
decreasing the dangers of, 341–342
designing for inheritance and, 96–97
documenting for, 347, 350
effect on exported APIs, 343
flexible return classes for, 365
immutability and, 85, 353
internal field theft attacks and, 360–362
lambdas and, 196
object deserialization filtering, 342
prefer alternatives to, 339–342
singletons and, 18
synchronization for, 351
transient fields for, 348
validity checking in, 357
when to use, 345
serialization proxy pattern, 363–366
serialized forms, as part of exported APIs, 343
serialized instances vs. serialization proxy pattern, 363–366
service provider frameworks, 8
short-circuiting operations, 223
signatures of methods, 3, 236–237
signum function, 67
simple implementations, 103
simulated multiple inheritance, 102
simulated self-type idiom, 14
single-check idiom, 335
singletons, 17–18
vs. dependency injection, 20
skeletal implementations, 100–101
source files, 115–116
space consumption
enum types, 175
immutable objects and, 83
memory leaks and, 27
nonstatic member classes and, 113
spliterator, 223
spurious wake-ups, 329
state-dependent modify operations, 325
state-testing methods, 294–295, 299

static factory methods, 5
  advantages of, 5–8
  anonymous classes within, 114
  in API documentation, 8
  vs. cloning, 65
  copy and conversion factories, 65
  flexibility in returned classes, 7–8
  for generic singletons, 18, 136
  immutable objects and, 22, 82, 84
  instance-controlled classes and, 6
  limitations of, 8–9
  naming conventions for, 9, 292
  replacing constructors with, 5–9, 22, 240
  return types of, 6–8
  for service provider frameworks, 8
  for singletons, 17
  subclassing and, 8
static fields
  for defining constants, 290
  lazy initialization of, 334
  synchronization of mutable, 322
static import facility, 108
static imports, 70
static member classes, 112
  cleaners and, 33
  common uses of, 112–113
  for enum types, 161
  vs. nonstatic, 112, 114
  for representing aggregates, 276
  for shortening parameter lists, 237
static members, accessibility in interfaces, 7
storage pools, 28
strategy enum pattern, 166
Strategy pattern, 193
stream pipelines, 203
  side-effect free, 210–215
stream unique identifiers
  See serial version UIDs
streams, 193, 203–225
  `char` values and, 206
  collectors for, 211–215
  for functional programming, 210–215
  vs. maps, behavior of, 173
  parallelizing, 222–225
  preserving order from parallel, 224
  as return types, vs. collections, 216–221
  specifying collectors for, 173, 214
  strengths of, 207
  vs. threads, 323–324
  using, 203–209
  See also collectors
`String` constants vs. enum types, 158
string representations, 55–57, 306
strings
  concatenating, 279
  as substitutes for other types, 276–278
subclassing, 3, 87
  abstract classes, and `equals`, 45
  access levels and, 75
  appropriateness of, 92
  `Cloneable` and, 96
  `compareTo` and, 68
  `equals` and, 40, 42
  finalizer attacks and, 31
  fragility, 89
  invariant corruption and, 92
  method access levels and, 75
  prohibiting, 8, 18–19, 85, 97
  serialization and, 344
  skeletal implementations, 102
  static factory methods and, 8
  as test of design for inheritance, 95
  See also inheritance
subtype relations, 134, 140
summary descriptions in Javadoc, 257
supertype relations, 141
`SuppressWarnings` annotation, 123–125
`switch` statements, and enum types, 164, 167
symmetry requirements
  `compareTo`, 68
  `equals`, 38–39
synchronization
  of atomic data, 312–314
  excessive, 317–322
    and performance, 321
    ramifications of, 317
  internal, 322
  for mutual exclusion, 311
  serialization and, 351
  for shared mutable data, 311–316
  techniques for, 314–316
  for thread communication, 312–314

synchronized regions
  alien methods and, 317–321
  minimizing work in, 321
synchronizers, 326
synthetic annotations, 186

T

tag fields, 109
tardy finalization, 29
tasks, 324
  vs. threads, 323–324
telescoping constructor pattern, 10–11
Template Method pattern, 101, 199
this, in doc comments, 256
`this`, in lambdas vs. anonymous classes, 196
thread pools, 323
  sizing of, 336
thread priorities, 337
thread safety
  documenting, 322, 330–332
  immutable objects and, 82
  levels of, 330–331
  mutability and, 75, 322
thread schedulers, 336–337
thread starvation deadlocks, 328
`Thread.yield` method, avoiding, 337
threads, busy-waiting, 336
throwables, types of, 296
time-of-check/time-of-use attacks, 233
TOCTOU attacks, 233
`toString` method, 55–57
  enum types and, 160
  general contract for, 55
  when to override, 57
transient fields, 348–351
  with `readResolve`, 360
  when to use, 351
transitivity requirements
  `compareTo`, 68
  `equals`, 38, 40–45
`try-finally`
  prefer `try`-with-resources to, 34
`try`-with-resources
  prefer to `try-finally`, 34–36
type bounds, recursive, 137
type inference, 70, 123, 142, 194
type parameter lists, 135
type parameters, 117, 135
  bounded, 134, 154
  naming conventions for, 135, 290
  recursively bound, 14, 137
  vs. wildcards, 144
type safety
  dynamic casts and, 154
  from enum types, 158
  heap pollution and, 146
  parameterized types and, 119
  raw types and, 119
type tokens, 151
types
  conversion of, 65, 291
  generic, 117, 130–134
  interfaces for defining, 107–108, 191–192
  non-reifiable, 127, 131
  parameterized, 117–122
  primitive, 273
  raw, 117
  reference, 273
  See also bounded wildcard types; unbounded wildcard types
typesafe heterogeneous container pattern, 151–155
  incompatibility with nonreifiable types, 154

U

unbounded type parameters
  vs. bounded wildcard types, 144
unbounded wildcard types, 120
  vs. bounded wildcard types, 121
  nested, 152
  vs. raw types, 121
  reifiable, 127
  vs. unbounded type parameters, 144
unchecked exceptions
  vs. checked exceptions, 296–297
  compatibility and, 305
  excluding from method declarations, 304
  purpose of, 296
unchecked warnings, 123–125
  of casts, 127, 129, 137
underscores, in numeric literals, 108
unintentional object retentions
  See memory leaks

unintentionally instantiable classes, 19
users of APIs, 4
utility classes, 19
  vs. constant interfaces, 108
  vs. dependency injection, 20

V

validity checking
  builders and, 14
  constructor parameters, 229
  defensive copying and, 232
  of deserialized objects, 355
  implicit, 230
  parameters, 227–230
    failure atomicity and, 308
  `readObject` parameters, 353–355
value classes, 38
  `toString` and, 56
value types vs. strings, 276
varargs, 245–246
  builders and, 16
  with generics, 146–150
  generics, and compiler warnings, 127
  performance, 246
variable arity methods, 245
variable return classes
  serialization proxy pattern and, 365
  static factory methods and, 7–8
variables
  atomic operations on, 311
  local (*see* local variables)
  naming conventions for, 290
  scope of, and obsolete references, 27
  to avoid subclassing, 44, 68
  to maintain invariants, 234
  naming conventions for, 291
  object reuse and, 23
`volatile` modifier, 314–315

W

wait loop idiom, 328–329
warnings, unchecked
  See unchecked warnings
weak references, 28
`while` loops vs. `for` loops, 262
wildcard types
  capturing, 145
  vs. type parameters, 144
  See also bounded wildcard types; unbounded wildcard types
window of vulnerability, 233
work queues, 326
wrapper class idiom, 100
wrapper classes, 89–91
  defensive copying and, 235
  incompatible with callback frameworks, 91
  vs. subclassing, 97
`writeReplace` method, access levels of, 97

Z

zero-length arrays, immutability of, 248