# Effective Java Optimize Practical

Monday, March 31, 2014 13:24

Item 1: Consider static factory methods instead of constructors
Item 2: Consider a builder when faced with many constructor parameters
Item 3: Enforce the singleton property with a private constructor or an enum type
Item 4: Enforce noninstantiability with a private constructor
Item 5: Avoid creating unnecessary objects
Item 6: Eliminate obsolete object references
Item 7: Avoid finalizers
Item 8: Obey the general contract when overriding equals
Item 9: Always override hashCode when you override equals
Item 10: Always override toString
Item 11: Override clone judiciously
Item 12: Consider implementing Comparable
Item 13: Minimize the accessibility of classes and members
Item 14: In public classes, use accessor methods, not public fields
Item 15: Minimize mutability
Item 16: Favor composition over inheritance
Item 17: Design and document for inheritance or else prohibit it
Item 18: Prefer interfaces to abstract classes
Item 19: Use interfaces only to define types
Item 20: Prefer class hierarchies to tagged classes
Item 21: Use function objects to represent strategies
Item 22: Favor static member classes over nonstatic
Item 23: Don’t use raw types in new code
Item 24: Eliminate unchecked warnings
Item 25: Prefer lists to arrays
Item 26: Favor generic types
Item 27: Favor generic methods
Item 28: Use bounded wildcards to increase API flexibility
Item 29: Consider typesafe heterogeneous containers
Item 30: Use enums instead of int constants
Item 31: Use instance fields instead of ordinals
Item 32: Use EnumSet instead of bit fields
Item 33: Use EnumMap instead of ordinal indexing
Item 34: Emulate extensible enums with interfaces
Item 35: Prefer annotations to naming patterns
Item 36: Consistently use the Override annotation
Item 37: Use marker interfaces to define types
Item 38: Check parameters for validity
Item 39: Make defensive copies when needed
Item 40: Design method signatures carefully
Item 41: Use overloading judiciously
Item 42: Use varargs judiciously
Item 43: Return empty arrays or collections, not nulls
Item 44: Write doc comments for all exposed API elements
Item 45: Minimize the scope of local variables
Item 46: Prefer for-each loops to traditional for loops
Item 47: Know and use the libraries
Item 48: Avoid float and double if exact answers are required
Item 49: Prefer primitive types to boxed primitives
Item 50: Avoid strings where other types are more appropriate
Item 51: Beware the performance of string concatenation
Item 52: Refer to objects by their interfaces
Item 53: Prefer interfaces to reflection

Item 54: Use native methods judiciously
Item 55: Optimize judiciously
Item 56: Adhere to generally accepted naming conventions
Item 57: Use exceptions only for exceptional conditions
Item 58: Use checked exceptions for recoverable conditions and runtime exceptions for programming errors
Item 59: Avoid unnecessary use of checked exceptions
Item 60: Favor the use of standard exceptions
Item 61: Throw exceptions appropriate to the abstraction
Item 62: Document all exceptions thrown by each method
Item 63: Include failure-capture information in detail messages
Item 64: Strive for failure atomicity
Item 65: Don’t ignore exceptions
Item 66: Synchronize access to shared mutable data
Item 67: Avoid excessive synchronization
Item 68: Prefer executors and tasks to threads
Item 69: Prefer concurrency utilities to wait and notify
Item 70: Document thread safety
Item 71: Use lazy initialization judiciously
Item 72: Don’t depend on the thread scheduler
Item 73: Avoid thread groups
Item 74: Implement Serializable judiciously
Item 75: Consider using a custom serialized form
Item 76: Write readObject methods defensively
Item 77: For instance control, prefer enum types to readResolve
Item 78: Consider serialization proxies instead of serialized instances