

# Contents

---

<b>Foreword</b> .....	<b>xi</b>
<b>Preface</b> .....	<b>xiii</b>
<b>Acknowledgments</b> .....	<b>xvii</b>
<b>1 Introduction</b> .....	<b>1</b>
<b>2 Creating and Destroying Objects</b> .....	<b>5</b>
Item 1: Consider static factory methods instead of constructors ...	5
Item 2: Consider a builder when faced with many constructor parameters .....	10
Item 3: Enforce the singleton property with a private constructor or an enum type .....	17
Item 4: Enforce noninstantiability with a private constructor ...	19
Item 5: Prefer dependency injection to hardwiring resources ...	20
Item 6: Avoid creating unnecessary objects .....	22
Item 7: Eliminate obsolete object references .....	26
Item 8: Avoid finalizers and cleaners .....	29
Item 9: Prefer try-with-resources to try-finally .....	34
<b>3 Methods Common to All Objects</b> .....	<b>37</b>
Item 10: Obey the general contract when overriding equals ...	37
Item 11: Always override hashCode when you override equals ..	50
Item 12: Always override toString .....	55
Item 13: Override clone judiciously .....	58
Item 14: Consider implementing Comparable .....	66
<b>4 Classes and Interfaces</b> .....	<b>73</b>
Item 15: Minimize the accessibility of classes and members ...	73
Item 16: In public classes, use accessor methods, not public fields	78
Item 17: Minimize mutability .....	80
Item 18: Favor composition over inheritance .....	87



Item 19: Design and document for inheritance or else prohibit it	93
Item 20: Prefer interfaces to abstract classes	99
Item 21: Design interfaces for posterity	104
Item 22: Use interfaces only to define types.	107
Item 23: Prefer class hierarchies to tagged classes	109
Item 24: Favor static member classes over nonstatic	112
Item 25: Limit source files to a single top-level class	115
<b>5 Generics</b>	<b>117</b>
Item 26: Don't use raw types	117
Item 27: Eliminate unchecked warnings.	123
Item 28: Prefer lists to arrays	126
Item 29: Favor generic types.	130
Item 30: Favor generic methods	135
Item 31: Use bounded wildcards to increase API flexibility	139
Item 32: Combine generics and varargs judiciously.	146
Item 33: Consider typesafe heterogeneous containers	151
<b>6 Enums and Annotations</b>	<b>157</b>
Item 34: Use enums instead of <code>int</code> constants.	157
Item 35: Use instance fields instead of ordinals	168
Item 36: Use <code>EnumSet</code> instead of bit fields.	169
Item 37: Use <code>EnumMap</code> instead of ordinal indexing.	171
Item 38: Emulate extensible enums with interfaces	176
Item 39: Prefer annotations to naming patterns	180
Item 40: Consistently use the <code>Override</code> annotation.	188
Item 41: Use marker interfaces to define types	191
<b>7 Lambdas and Streams</b>	<b>193</b>
Item 42: Prefer lambdas to anonymous classes	193
Item 43: Prefer method references to lambdas	197
Item 44: Favor the use of standard functional interfaces	199
Item 45: Use streams judiciously	203
Item 46: Prefer side-effect-free functions in streams	210
Item 47: Prefer <code>Collection</code> to <code>Stream</code> as a return type.	216
Item 48: Use caution when making streams parallel	222



<b>8</b>	<b>Methods</b> .....	<b>227</b>
	Item 49: Check parameters for validity .....	227
	Item 50: Make defensive copies when needed .....	231
	Item 51: Design method signatures carefully .....	236
	Item 52: Use overloading judiciously .....	238
	Item 53: Use varargs judiciously .....	245
	Item 54: Return empty collections or arrays, not nulls .....	247
	Item 55: Return optionals judiciously .....	249
	Item 56: Write doc comments for all exposed API elements ....	254
<b>9</b>	<b>General Programming</b> .....	<b>261</b>
	Item 57: Minimize the scope of local variables .....	261
	Item 58: Prefer for-each loops to traditional for loops .....	264
	Item 59: Know and use the libraries .....	267
	Item 60: Avoid float and double if exact answers are required .	270
	Item 61: Prefer primitive types to boxed primitives .....	273
	Item 62: Avoid strings where other types are more appropriate ..	276
	Item 63: Beware the performance of string concatenation .....	279
	Item 64: Refer to objects by their interfaces .....	280
	Item 65: Prefer interfaces to reflection .....	282
	Item 66: Use native methods judiciously .....	285
	Item 67: Optimize judiciously .....	286
	Item 68: Adhere to generally accepted naming conventions .....	289
<b>10</b>	<b>Exceptions</b> .....	<b>293</b>
	Item 69: Use exceptions only for exceptional conditions .....	293
	Item 70: Use checked exceptions for recoverable conditions and runtime exceptions for programming errors .....	296
	Item 71: Avoid unnecessary use of checked exceptions .....	298
	Item 72: Favor the use of standard exceptions .....	300
	Item 73: Throw exceptions appropriate to the abstraction .....	302
	Item 74: Document all exceptions thrown by each method .....	304
	Item 75: Include failure-capture information in detail messages ..	306
	Item 76: Strive for failure atomicity .....	308
	Item 77: Don't ignore exceptions .....	310



<b>11 Concurrency</b> .....	<b>311</b>
Item 78: Synchronize access to shared mutable data .....	311
Item 79: Avoid excessive synchronization .....	317
Item 80: Prefer executors, tasks, and streams to threads .....	323
Item 81: Prefer concurrency utilities to wait and notify .....	325
Item 82: Document thread safety .....	330
Item 83: Use lazy initialization judiciously .....	333
Item 84: Don't depend on the thread scheduler .....	336
<b>12 Serialization</b> .....	<b>339</b>
Item 85: Prefer alternatives to Java serialization .....	339
Item 86: Implement Serializable with great caution .....	343
Item 87: Consider using a custom serialized form .....	346
Item 88: Write readObject methods defensively .....	353
Item 89: For instance control, prefer enum types to readResolve .....	359
Item 90: Consider serialization proxies instead of serialized instances .....	363
<b>Items Corresponding to Second Edition</b> .....	<b>367</b>
<b>References</b> .....	<b>371</b>
<b>Index</b> .....	<b>377</b>