

Contents

Foreword	xi
Preface	xiii
Acknowledgments	xix
About the Authors	xxi
Chapter 1 Security	1
1. Limit the lifetime of sensitive data	2
2. Do not store unencrypted sensitive information on the client side	5
3. Provide sensitive mutable classes with unmodifiable wrappers	9
4. Ensure that security-sensitive methods are called with validated arguments	11
5. Prevent arbitrary file upload	13
6. Properly encode or escape output	16
7. Prevent code injection	20
8. Prevent XPath injection	23
9. Prevent LDAP injection	27
10. Do not use the <code>clone()</code> method to copy untrusted method parameters	31
11. Do not use <code>Object.equals()</code> to compare cryptographic keys	34

	12. Do not use insecure or weak cryptographic algorithms	36
	13. Store passwords using a hash function	37
	14. Ensure that <code>SecureRandom</code> is properly seeded	42
	15. Do not rely on methods that can be overridden by untrusted code	44
	16. Avoid granting excess privileges	50
	17. Minimize privileged code	54
	18. Do not expose methods that use reduced-security checks to untrusted code	56
	19. Define custom security permissions for fine-grained security	64
	20. Create a secure sandbox using a security manager	67
	21. Do not let untrusted code misuse privileges of callback methods	72
Chapter 2	Defensive Programming	79
	22. Minimize the scope of variables	80
	23. Minimize the scope of the <code>@SuppressWarnings</code> annotation	82
	24. Minimize the accessibility of classes and their members	84
	25. Document thread-safety and use annotations where applicable	89
	26. Always provide feedback about the resulting value of a method	96
	27. Identify files using multiple file attributes	99
	28. Do not attach significance to the ordinal associated with an enum	106
	29. Be aware of numeric promotion behavior	108
	30. Enable compile-time type checking of variable arity parameter types	112
	31. Do not apply <code>public final</code> to constants whose value might change in later releases	115
	32. Avoid cyclic dependencies between packages	118
	33. Prefer user-defined exceptions over more general exception types	121
	34. Try to gracefully recover from system errors	123
	35. Carefully design interfaces before releasing them	125
	36. Write garbage collection-friendly code	128
Chapter 3	Reliability	131
	37. Do not shadow or obscure identifiers in subscopes	132
	38. Do not declare more than one variable per declaration	134

39. Use meaningful symbolic constants to represent literal values in program logic	138
40. Properly encode relationships in constant definitions	142
41. Return an empty array or collection instead of a null value for methods that return an array or collection	143
42. Use exceptions only for exceptional conditions	146
43. Use a try-with-resources statement to safely handle closeable resources	148
44. Do not use assertions to verify the absence of runtime errors	151
45. Use the same type for the second and third operands in conditional expressions	153
46. Do not serialize direct handles to system resources	157
47. Prefer using iterators over enumerations	159
48. Do not use direct buffers for short-lived, infrequently used objects	162
49. Remove short-lived objects from long-lived container objects	163
Chapter 4 Program Understandability	167
50. Be careful using visually misleading identifiers and literals	167
51. Avoid ambiguous overloading of variable arity methods	171
52. Avoid in-band error indicators	173
53. Do not perform assignments in conditional expressions	175
54. Use braces for the body of an if, for, or while statement	178
55. Do not place a semicolon immediately following an if, for, or while condition	180
56. Finish every set of statements associated with a case label with a break statement	181
57. Avoid inadvertent wrapping of loop counters	183
58. Use parentheses for precedence of operation	186
59. Do not make assumptions about file creation	189
60. Convert integers to floating-point for floating-point operations	191
61. Ensure that the clone() method calls super.clone()	194
62. Use comments consistently and in a readable fashion	196
63. Detect and remove superfluous code and values	198
64. Strive for logical completeness	202
65. Avoid ambiguous or confusing uses of overloading	205

Chapter 5	Programmer Misconceptions	209
	66. Do not assume that declaring a reference volatile guarantees safe publication of the members of the referenced object	209
	67. Do not assume that the <code>sleep()</code> , <code>yield()</code> , or <code>getState()</code> methods provide synchronization semantics	216
	68. Do not assume that the remainder operator always returns a nonnegative result for integral operands	220
	69. Do not confuse abstract object equality with reference equality	222
	70. Understand the differences between bitwise and logical operators	225
	71. Understand how escape characters are interpreted when strings are loaded	228
	72. Do not use overloaded methods to differentiate between runtime types	231
	73. Never confuse the immutability of a reference with that of the referenced object	234
	74. Use the serialization methods <code>writeUnshared()</code> and <code>readUnshared()</code> with care	239
	75. Do not attempt to help the garbage collector by setting local reference variables to <code>null</code>	243
Appendix A	Android	245
	Glossary	249
	References	255
	Index	263