Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

1. Components should not be vulnerable to intent redirection
   Vulnerability
2. XML parsers should not allow inclusion of arbitrary files
   Vulnerability
3. HTTP responses should not be vulnerable to session fixation
   Vulnerability
4. Extracting archives should not lead to zip slip vulnerabilities
   Vulnerability
5. Dynamic code execution should not be vulnerable to injection attacks
   Vulnerability
6. NoSQL operations should not be vulnerable to injection attacks
   Vulnerability
7. HTTP request redirections should not be open to forging attacks
   Vulnerability
8. Deserialization should not be vulnerable to injection attacks
   Vulnerability
9. Endpoints should not be vulnerable to reflected cross-site scripting (XSS) attacks
   Vulnerability
10. Database queries should not be vulnerable to injection attacks
    Vulnerability
11. XML parsers should not be vulnerable to XXE attacks
    Vulnerability
12. A secure password should be used when connecting to a database
    Vulnerability
13. XPath expressions should not be vulnerable to injection attacks
    Vulnerability
14. I/O function calls should not be vulnerable to path injection attacks
    Vulnerability
15. LDAP queries should not be vulnerable to injection attacks
    Vulnerability
16.

OS commands should not be vulnerable to command injection attacks
Vulnerability

17.

"@SpringBootApplication" and "@ComponentScan" should not be used in the default package
Bug

18.

"@Controller" classes that use "@SessionAttributes" must call "setComplete" on their "SessionStatus" objects
Bug

19.

"wait" should not be called when multiple locks are held
Bug

20.

"PreparedStatement" and "ResultSet" methods should be called with valid indices
Bug

21.

Files opened in append mode should not be used with ObjectOutputStream
Bug

22.

"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held
Bug

23.

Printf-style format strings should not lead to unexpected behavior at runtime
Bug

24.

Methods "wait(...)", "notify()" and "notifyAll()" should not be called on Thread instances
Bug

25.

Methods should not call same-class methods with incompatible "@Transactional" values
Bug

26.

Recursion should not be infinite
Bug

27.

Loops should not be infinite
Bug

28.

Double-checked locking should not be used
Bug

29.

Resources should be closed
Bug

30.

Hard-coded credentials are security-sensitive
Security Hotspot

31.

Methods returns should not be invariant
Code Smell

32.

"ThreadGroup" should not be used
Code Smell

33.

"clone" should not be overridden
Code Smell

34.

Assertions should be complete
Code Smell

35.

Tests should include assertions
Code Smell

36.

Silly bit operations should not be performed
Code Smell

37.

Child class fields should not shadow parent class fields
Code Smell

38.

JUnit test cases should call super methods
Code Smell

39.

TestCases should contain tests
Code Smell

40.

Short-circuit logic should be used in boolean contexts
Code Smell

41.

Methods and field names should not be the same or differ only by capitalization
Code Smell

42.

Switch cases should end with an unconditional "break" statement
Code Smell

43.

"switch" statements should not contain non-case labels
Code Smell

44.

Future keywords should not be used as names
Code Smell

45.

Thread suspensions should not be vulnerable to Denial of Service attacks
Vulnerability

46.

A new session should be created during user authentication
Vulnerability

47.

JWT should be signed and verified with strong cipher algorithms
Vulnerability

48.

Cipher algorithms should be robust
Vulnerability

49.

Encryption algorithms should be used with secure mode and padding scheme
Vulnerability

50.
Server hostnames should be verified during SSL/TLS connections
Vulnerability

51.
Insecure temporary file creation methods should not be used
Vulnerability

52.
Passwords should not be stored in plain-text or with a fast hashing algorithm
Vulnerability

53.
Server certificates should be verified during SSL/TLS connections
Vulnerability

54.
Persistent entities should not be used as arguments of "@RequestMapping" methods
Vulnerability

55.
"HttpSecurity" URL patterns should be correctly ordered
Vulnerability

56.
LDAP connections should be authenticated
Vulnerability

57.
Cryptographic keys should be robust
Vulnerability

58.
Weak SSL/TLS protocols should not be used
Vulnerability

59.
"SecureRandom" seeds should not be predictable
Vulnerability

60.
Cipher Block Chaining IVs should be unpredictable
Vulnerability

61.
Basic authentication should not be used
Vulnerability

62.
Regular expressions should not be vulnerable to Denial of Service attacks
Vulnerability

63.
"HttpServletRequest.getRequestedSessionId()" should not be used
Vulnerability

64.
Hashes should include an unpredictable salt
Vulnerability

65.
Calls to methods should not trigger an IllegalArgumentException
Bug

66.
Unsupported methods should not be called on some collection implementations
Bug

67.

Cast operations should not trigger a ClassCastException
Bug

68.

Members ignored during record serialization should not be used
Bug

69.

Map "computeIfAbsent()" and "computeIfPresent()" should not be used to add "null" values.
Bug

70.

Regex lookahead assertions should not be contradictory
Bug

71.

Back references in regular expressions should only refer to capturing groups that are matched before the reference
Bug

72.

Regex boundaries should not be used in a way that can never be matched
Bug

73.

Regex patterns following a possessive quantifier should not always fail
Bug

74.

Regular expressions should be syntactically valid
Bug

75.

Assertions comparing incompatible types should not be made
Bug

76.

JUnit5 inner test classes should be annotated with @Nested
Bug

77.

Only one method invocation is expected when testing checked exceptions
Bug

78.

Assertion methods should not be used within the try block of a try-catch catching an Error
Bug

79.

Getters and setters should access the expected fields
Bug

80.

Zero should not be a possible denominator
Bug

81.

Locks should be released
Bug

82.

"runFinalizersOnExit" should not be called
Bug

83.

	"ScheduledThreadPoolExecutor" should not have 0 core threads Bug
84.	
	"Random" objects should be reused Bug
85.	
	The signature of "finalize()" should match that of "Object.finalize()" Bug
86.	
	Jump statements should not occur in "finally" blocks Bug
87.	
	"super.finalize()" should be called at the end of "Object.finalize()" implementations Bug
88.	
	Using slow regular expressions is security-sensitive Security Hotspot
89.	
	Using publicly writable directories is security-sensitive Security Hotspot
90.	
	Using clear-text protocols is security-sensitive Security Hotspot
91.	
	Accessing Android external storage is security-sensitive Security Hotspot
92.	
	Receiving intents is security-sensitive Security Hotspot
93.	
	Broadcasting intents is security-sensitive Security Hotspot
94.	
	Expanding archive files without controlling resource consumption is security-sensitive Security Hotspot
95.	
	Configuring loggers is security-sensitive Security Hotspot
96.	
	Using weak hashing algorithms is security-sensitive Security Hotspot
97.	
	Using unsafe Jackson deserialization configuration is security-sensitive Security Hotspot
98.	
	Setting JavaBean properties is security-sensitive Security Hotspot
99.	
	Disabling CSRF protections is security-sensitive Security Hotspot
100.	

Using non-standard cryptographic algorithms is security-sensitive
Security Hotspot

101.

Using pseudorandom number generators (PRNGs) is security-sensitive
Security Hotspot

102.

Mocking all non-private methods of a class should be avoided
Code Smell

103.

Empty lines should not be tested with regex MULTILINE flag
Code Smell

104.

Methods setUp() and tearDown() should be correctly annotated starting with JUnit4
Code Smell

105.

Class members annotated with "@VisibleForTesting" should not be accessed from production code
Code Smell

106.

"String#replace" should be preferred to "String#replaceAll"
Code Smell

107.

Derived exceptions should not hide their parents' catch blocks
Code Smell

108.

String offset-based methods should be preferred for finding substrings from offsets
Code Smell

109.

"default" clauses should be last
Code Smell

110.

"equals" method parameters should not be marked "@Nonnull"
Code Smell

111.

A conditionally executed single line should be denoted by indentation
Code Smell

112.

Conditionals should start on new lines
Code Smell

113.

Cognitive Complexity of methods should not be too high
Code Smell

114.

Factory method injection should be used in "@Configuration" classes
Code Smell

115.

"static" base class members should not be accessed via derived types
Code Smell

116.

Instance methods should not write to "static" fields
Code Smell

117.
"indexOf" checks should not be for positive numbers
Code Smell

118.
Method overrides should not change contracts
Code Smell

119.
Whitespace and control characters in literals should be explicit
Code Smell

120.
Null should not be returned from a "Boolean" method
Code Smell

121.
Classes should not access their own subclasses during initialization
Code Smell

122.
"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop
Code Smell

123.
IllegalMonitorStateException should not be caught
Code Smell

124.
JUnit assertions should not be used in "run" methods
Code Smell

125.
Class names should not shadow interfaces or superclasses
Code Smell

126.
"Cloneables" should implement "clone"
Code Smell

127.
Try-with-resources should be used
Code Smell

128.
"readResolve" methods should be inheritable
Code Smell

129.
"for" loop increment clauses should modify the loops' counters
Code Smell

130.
Fields in a "Serializable" class should either be transient or serializable
Code Smell

131.
Package declaration should match source file directory
Code Smell

132.
Generic wildcard types should not be used in return types
Code Smell

133.
"switch" statements should have "default" clauses
Code Smell

134.
Execution of the Garbage Collector should be triggered only by the JVM
Code Smell
135.
Constants should not be defined in interfaces
Code Smell
136.
String literals should not be duplicated
Code Smell
137.
Methods should not be empty
Code Smell
138.
"Object.finalize()" should remain protected (versus public) when overriding
Code Smell
139.
Exceptions should not be thrown in finally blocks
Code Smell
140.
Constant names should comply with a naming convention
Code Smell
141.
The Object.finalize() method should not be overridden
Code Smell
142.
XML operations should not be vulnerable to injection attacks
Vulnerability
143.
JSON operations should not be vulnerable to injection attacks
Vulnerability
144.
XML signatures should be validated securely
Vulnerability
145.
XML parsers should not be vulnerable to Denial of Service attacks
Vulnerability
146.
XML parsers should not load external schemas
Vulnerability
147.
Mobile database encryption keys should not be disclosed
Vulnerability
148.
Reflection should not be vulnerable to injection attacks
Vulnerability
149.
Authorizations should be based on strong decisions
Vulnerability
150.
OpenSAML2 should be configured to prevent authentication bypass
Vulnerability

151.
Server-side requests should not be vulnerable to forging attacks
Vulnerability

152.
Collections should not be modified while they are iterated
Bug

153.
Equals method should be overridden in records containing array fields
Bug

154.
Reflection should not be used to increase accessibility of records' fields
Bug

155.
AssertJ assertions with "Consumer" arguments should contain assertion inside consumers
Bug

156.
The regex escape sequence \cX should only be used with characters in the @-_ range
Bug

157.
Regular expressions should not overflow the stack
Bug

158.
Tests method should not be annotated with competing annotations
Bug

159.
Assertions should not be used in production code
Bug

160.
DateTimeFormatters should not use mismatched year and week numbers
Bug

161.
Unicode Grapheme Clusters should be avoided inside regex character classes
Bug

162.
Case insensitive Unicode regular expressions should enable the “UNICODE_CASE” flag
Bug

163.
Assertions should not compare an object to itself
Bug

164.
Regex alternatives should not be redundant
Bug

165.
Alternatives in regular expressions should be grouped when used with anchors
Bug

166.
AssertJ methods setting the assertion context should come before an assertion
Bug

167.
AssertJ configuration should be applied

Bug

168.

JUnit5 test classes and methods should not be silently ignored
Bug

169.

"ThreadLocal" variables should be cleaned up when no longer used
Bug

170.

Strings and Boxed types should be compared using "equals()"
Bug

171.

InputSteam.read() implementation should not return a signed byte
Bug

172.

"compareTo" should not be overloaded
Bug

173.

"iterator" should not return "this"
Bug

174.

Map values should not be replaced unconditionally
Bug

175.

Week Year ("YYYY") should not be used for date formatting
Bug

176.

Exceptions should not be created without being thrown
Bug

177.

Collection sizes and array length comparisons should make sense
Bug

178.

Consumed Stream pipelines should not be reused
Bug

179.

Intermediate Stream methods should not be left unused
Bug

180.

All branches in a conditional structure should not have exactly the same implementation
Bug

181.

Optional value should only be accessed after calling isPresent()
Bug

182.

Overrides should match their parent class methods in synchronization
Bug

183.

Value-based classes should not be used for locking
Bug

184.

Expressions used in "assert" should not produce side effects

Bug

185.

"volatile" variables should not be used with compound operators
Bug

186.

"getClass" should not be used for synchronization
Bug

187.

Min and max used in combination should not always return the same value
Bug

188.

Assignment of lazy-initialized members should be the last step with double-checked locking
Bug

189.

"String" calls should not go beyond their bounds
Bug

190.

Raw byte values should not be used in bitwise operations in combination with shifts
Bug

191.

Getters and setters should be synchronized in pairs
Bug

192.

Non-thread-safe fields should not be static
Bug

193.

"null" should not be used with "Optional"
Bug

194.

Unary prefix operators should not be repeated
Bug

195.

"=+" should not be used instead of "+="
Bug

196.

"read" and "readLine" return values should be used
Bug

197.

Inappropriate regular expressions should not be used
Bug

198.

Conditionally executed code should be reachable
Bug

199.

"notifyAll" should be used
Bug

200.

Blocks should be synchronized on "private final" fields
Bug

201.

Non-serializable objects should not be stored in "HttpSession" objects
Bug

202.
"wait", "notify" and "notifyAll" should only be called when a lock is obviously held on an object
Bug

203.
Null pointers should not be dereferenced
Bug

204.
Loop conditions should be true at least once
Bug

205.
A "for" loop update clause should move the counter in the right direction
Bug

206.
Non-public methods should not be "@Transactional"
Bug

207.
Servlets should not have mutable instance fields
Bug

208.
"toString()" and "clone()" methods should not return null
Bug

209.
".equals()" should not be used to test the values of "Atomic" classes
Bug

210.
Return values from functions without side effects should not be ignored
Bug

211.
Child class methods named for parent class methods should be overrides
Bug

212.
Inappropriate "Collection" calls should not be made
Bug

213.
Silly equality checks should not be made
Bug

214.
Dissimilar primitive wrappers should not be used with the ternary operator without explicit casting
Bug

215.
"InterruptedException" should not be ignored
Bug

216.
Classes extending java.lang.Thread should override the "run" method
Bug

217.
"Double.longBitsToDouble" should not be used for "int"
Bug

218.

Values should not be uselessly incremented
Bug

219.

Silly String operations should not be made
Bug

220.

Non-serializable classes should not be written
Bug

221.

"hashCode" and "toString" should not be called on array instances
Bug

222.

Collections should not be passed as arguments to their own methods
Bug

223.

"BigDecimal(double)" should not be used
Bug

224.

Invalid "Date" values should not be used
Bug

225.

Reflection should not be used to check non-runtime annotations
Bug

226.

Custom serialization method signatures should meet requirements
Bug

227.

"Externalizable" classes should have no-arguments constructors
Bug

228.

Classes should not be compared by name
Bug

229.

Related "if/else if" statements should not have the same condition
Bug

230.

Synchronization should not be done on instances of value-based classes
Bug

231.

"Iterator.hasNext()" should not call "Iterator.next()"
Bug

232.

Identical expressions should not be used on both sides of a binary operator
Bug

233.

Loops with at most one iteration should be refactored
Bug

234.

Variables should not be self-assigned
Bug

235.
"StringBuilder" and "StringBuffer" should not be instantiated with a character
Bug

236.
Methods should not be named "tostring", "hashcode" or "equal"
Bug

237.
"Thread.run()" should not be called directly
Bug

238.
"equals" method overrides should accept "Object" parameters
Bug

239.
The Object.finalize() method should not be called
Bug

240.
Enabling file access for WebViews is security-sensitive
Security Hotspot

241.
Enabling JavaScript support for WebViews is security-sensitive
Security Hotspot

242.
Constructing arguments of system commands from user input is security-sensitive
Security Hotspot

243.
Using unencrypted files in mobile applications is security-sensitive
Security Hotspot

244.
Using biometric authentication without a cryptographic solution is security-sensitive
Security Hotspot

245.
Using unencrypted databases in mobile applications is security-sensitive
Security Hotspot

246.
Authorizing non-authenticated users to use keys in the Android KeyStore is security-sensitive
Security Hotspot

247.
Allowing user enumeration is security-sensitive
Security Hotspot

248.
Allowing requests with excessive content length is security-sensitive
Security Hotspot

249.
Disabling auto-escaping in template engines is security-sensitive
Security Hotspot

250.
Allowing deserialization of LDAP objects is security-sensitive
Security Hotspot

251.
Setting loose POSIX file permissions is security-sensitive

Security Hotspot

252.

Formatting SQL queries is security-sensitive
Security Hotspot

253.

Deprecated annotations should include explanations
Code Smell

254.

Restricted Identifiers should not be used as Identifiers
Code Smell

255.

Redundant constructors/methods should be avoided in records
Code Smell

256.

Records should be used instead of ordinary classes when representing immutable data structure
Code Smell

257.

"Stream.toList()" method should be used instead of "collectors" when unmodifiable list needed
Code Smell

258.

Operator "instanceof" should be used instead of "A.class.isInstance()"
Code Smell

259.

String multiline concatenation should be replaced with Text Blocks
Code Smell

260.

Single-character alternations in regular expressions should be replaced with character classes
Code Smell

261.

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string
Code Smell

262.

Constructors of an "abstract" class should not be declared "public"
Code Smell

263.

Similar tests should be grouped in a single Parameterized test
Code Smell

264.

Tests should be stable
Code Smell

265.

Test methods should not contain too many assertions
Code Smell

266.

AssertJ "assertThatThrownBy" should not be used alone
Code Smell

267.

Character classes in regular expressions should not contain the same character twice

Code Smell

268.

Names of regular expressions named groups should be used
Code Smell

269.

Regexes containing characters subject to normalization should use the CANON_EQ flag
Code Smell

270.

Regular expressions should not be too complicated
Code Smell

271.

JUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion
Code Smell

272.

Only one method invocation is expected when testing runtime exceptions
Code Smell

273.

Exception testing via JUnit ExpectedException rule should not be mixed with other assertions
Code Smell

274.

"@Deprecated" code marked for removal should never be used
Code Smell

275.

Vararg method arguments should not be confusing
Code Smell

276.

Whitespace for text block indent should be consistent
Code Smell

277.

'List.remove()' should not be used in ascending 'for' loops
Code Smell

278.

Collection constructors should not be used as java.util.function.Function
Code Smell

279.

"else" statements should be clearly matched with an "if"
Code Smell

280.

"Class.forName()" should not load JDBC 4.0+ drivers
Code Smell

281.

Java features should be preferred to Guava
Code Smell

282.

Nullness of parameters should be guaranteed
Code Smell

283.

"Integer.toHexString" should not be used to build hexadecimal strings
Code Smell

284.

Asserts should not be used to check the parameters of a public method
Code Smell

285.

Assignments should not be redundant
Code Smell

286.

Methods should not have identical implementations
Code Smell

287.

"java.nio.Files#delete" should be preferred
Code Smell

288.

Unused "private" classes should be removed
Code Smell

289.

"Stream.peek" should be used with caution
Code Smell

290.

"Map.get" and value test should be replaced with single method call
Code Smell

291.

"@RequestMapping" methods should not be "private"
Code Smell

292.

Raw types should not be used
Code Smell

293.

"Arrays.stream" should be used for primitive arrays
Code Smell

294.

Printf-style format strings should be used correctly
Code Smell

295.

Assertion arguments should be passed in the correct order
Code Smell

296.

Ternary operators should not be nested
Code Smell

297.

"writeObject" should not be the only "synchronized" code in a class
Code Smell

298.

Reflection should not be used to increase accessibility of classes, methods, or fields
Code Smell

299.

Static fields should not be updated in constructors
Code Smell

300.

"Thread.sleep" should not be used in tests
Code Smell

301.

"entrySet()" should be iterated when both the key and value are needed
Code Smell

302.

"DateUtils.truncate" from Apache Commons Lang library should not be used
Code Smell

303.

Multiline blocks should be enclosed in curly braces
Code Smell

304.

"readObject" should not be "synchronized"
Code Smell

305.

"Preconditions" and logging arguments should not require evaluation
Code Smell

306.

Boolean expressions should not be gratuitous
Code Smell

307.

"Lock" objects should not be "synchronized"
Code Smell

308.

Classes with only "static" methods should not be instantiated
Code Smell

309.

"Threads" should not be used where "Runnables" are expected
Code Smell

310.

Inner class calls to super class methods should be unambiguous
Code Smell

311.

Unused type parameters should be removed
Code Smell

312.

Parameters should be passed in the correct order
Code Smell

313.

"ResultSet.isLast()" should not be used
Code Smell

314.

"static" members should be accessed statically
Code Smell

315.

Silly math should not be performed
Code Smell

316.

Classes named like "Exception" should extend "Exception" or a subclass
Code Smell

317.

Exceptions should be either logged or rethrown but not both
Code Smell

318.

Objects should not be created only to "getClass"
Code Smell

319.

Primitives should not be boxed just for "String" conversion
Code Smell

320.

Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes
Code Smell

321.

"URL.hashCode" and "URL.equals" should be avoided
Code Smell

322.

Two branches in a conditional structure should not have exactly the same implementation
Code Smell

323.

Unused assignments should be removed
Code Smell

324.

"Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"
Code Smell

325.

A field should not duplicate the name of its containing class
Code Smell

326.

JUnit4 @Ignored and JUnit5 @Disabled annotations should be used to disable tests and should provide a rationale
Code Smell

327.

Anonymous inner classes containing only one method should become lambdas
Code Smell

328.

"switch" statements should not have too many "case" clauses
Code Smell

329.

"for" loop stop conditions should be invariant
Code Smell

330.

Sections of code should not be commented out
Code Smell

331.

Non-constructor methods should not have the same name as the enclosing class
Code Smell

332.

Exception types should not be tested using "instanceof" in catch blocks
Code Smell

333.

Classes from "sun.*" packages should not be used
Code Smell

334.
Throwable and Error should not be caught
Code Smell

335.
Unused method parameters should be removed
Code Smell

336.
Only static class initializers should be used
Code Smell

337.
Empty arrays and collections should be returned instead of null
Code Smell

338.
"@Override" should be used on overriding and implementing methods
Code Smell

339.
Enumeration should not be implemented
Code Smell

340.
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used
Code Smell

341.
Unused "private" methods should be removed
Code Smell

342.
Try-catch blocks should not be nested
Code Smell

343.
Track uses of "FIXME" tags
Code Smell

344.
Deprecated elements should have both the annotation and the Javadoc tag
Code Smell

345.
Assignments should not be made from within sub-expressions
Code Smell

346.
Generic exceptions should never be thrown
Code Smell

347.
Labels should not be used
Code Smell

348.
Utility classes should not have public constructors
Code Smell

349.
Local variables should not shadow class fields
Code Smell

350.
Redundant pairs of parentheses should be removed
Code Smell

351.
Inheritance tree of classes should not be too deep
Code Smell

352.
Nested blocks of code should not be left empty
Code Smell

353.
Methods should not have too many parameters
Code Smell

354.
Unused "private" fields should be removed
Code Smell

355.
Collapsible "if" statements should be merged
Code Smell

356.
Unused labels should be removed
Code Smell

357.
Standard outputs should not be used directly to log anything
Code Smell

358.
OS commands should not be vulnerable to argument injection attacks
Vulnerability

359.
"ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization
Vulnerability

360.
Logging should not be vulnerable to injection attacks
Vulnerability

361.
Exceptions should not be thrown from servlet methods
Vulnerability

362.
Return values should not be ignored when they contain the operation status code
Bug

363.
Repeated patterns in regular expressions should not match the empty string
Bug

364.
AssertJ assertions "allMatch" and "doesNotContains" should also test for emptiness
Bug

365.
Double Brace Initialization should not be used
Bug

366.
Non-primitive fields should not be "volatile"
Bug

367.
"toArray" should be passed an array of the proper type
Bug

368.

Neither "Math.abs" nor negation should be used on numbers that could be "MIN_VALUE"
Bug

369.

The value returned from a stream read should be checked
Bug

370.

"@NonNull" values should not be set to null
Bug

371.

"Iterator.next()" methods should throw "NoSuchElementException"
Bug

372.

"compareTo" results should not be checked for specific values
Bug

373.

Math operands should be cast before assignment
Bug

374.

Ints and longs should not be shifted by zero or more than their number of bits-1
Bug

375.

"compareTo" should not return "Integer.MIN_VALUE"
Bug

376.

Boxing and unboxing should not be immediately reversed
Bug

377.

"equals(Object obj)" should test argument type
Bug

378.

"Serializable" inner classes of non-serializable classes should be "static"
Bug

379.

The non-serializable super class of a "Serializable" class should have a no-argument constructor
Bug

380.

Method parameters, caught exceptions and foreach variables' initial values should not be ignored
Bug

381.

"equals(Object obj)" and "hashCode()" should be overridden in pairs
Bug

382.

Disclosing fingerprints from web application technologies is security-sensitive
Security Hotspot

383.

Having a permissive Cross-Origin Resource Sharing policy is security-sensitive
Security Hotspot

384.

Delivering code in production with debug features activated is security-sensitive
Security Hotspot

385.
Searching OS commands in PATH is security-sensitive
Security Hotspot

386.
Allowing both safe and unsafe HTTP methods is security-sensitive
Security Hotspot

387.
Creating cookies without the "HttpOnly" flag is security-sensitive
Security Hotspot

388.
Creating cookies without the "secure" flag is security-sensitive
Security Hotspot

389.
Using hardcoded IP addresses is security-sensitive
Security Hotspot

390.
'serialVersionUID' field should not be set to '0L' in records
Code Smell

391.
Permitted types of a sealed class should be omitted if they are declared in the same file
Code Smell

392.
Switch arrow labels should not use redundant keywords
Code Smell

393.
Text blocks should not be used in complex expressions
Code Smell

394.
Pattern Matching for "instanceof" operator should be used instead of simple "instanceof" + cast
Code Smell

395.
Call to Mockito method "verify", "when" or "given" should be simplified
Code Smell

396.
Character classes should be preferred over reluctant quantifiers in regular expressions
Code Smell

397.
Consecutive AssertJ "assertThat" statements should be chained
Code Smell

398.
Chained AssertJ assertions should be simplified to the corresponding dedicated assertion
Code Smell

399.
Exception testing via JUnit @Test annotation should be avoided
Code Smell

400.
Escape sequences should not be used in text blocks
Code Smell

401.

Simple string literal should be used for single line strings
Code Smell

402.

Boxed "Boolean" should be avoided in boolean expressions
Code Smell

403.

Type parameters should not shadow other type parameters
Code Smell

404.

"read(byte[],int,int)" should be overridden
Code Smell

405.

An iteration on a Collection should be performed on the type handled by the Collection
Code Smell

406.

"StandardCharsets" constants should be preferred
Code Smell

407.

"@CheckForNull" or "@Nullable" should not be used on primitive types
Code Smell

408.

Composed "@RequestMapping" variants should be preferred
Code Smell

409.

"write(byte[],int,int)" should be overridden
Code Smell

410.

Functional Interfaces should be as specialised as possible
Code Smell

411.

Null checks should not be used with "instanceof"
Code Smell

412.

"close()" calls should not be redundant
Code Smell

413.

"ThreadLocal.withInitial" should be preferred
Code Smell

414.

"Stream" call chains should be simplified when possible
Code Smell

415.

Packages containing only "package-info.java" should be removed
Code Smell

416.

Arrays should not be created for varargs parameters
Code Smell

417.

Jump statements should not be redundant
Code Smell

418.

Test classes should comply with a naming convention
Code Smell

419.

Loggers should be named for their enclosing classes
Code Smell

420.

Methods should not return constants
Code Smell

421.

"private" methods called only by inner classes should be moved to those classes
Code Smell

422.

"enum" fields should not be publicly mutable
Code Smell

423.

Abstract methods should not be redundant
Code Smell

424.

Arrays should not be copied using loops
Code Smell

425.

Static non-final field names should comply with a naming convention
Code Smell

426.

JUnit rules should be used
Code Smell

427.

Nested "enum"s should not be declared static
Code Smell

428.

"catch" clauses should do more than rethrow
Code Smell

429.

Mutable fields should not be "public static"
Code Smell

430.

The diamond operator ("<>") should be used
Code Smell

431.

"finalize" should not set fields to "null"
Code Smell

432.

Subclasses that add fields should override "equals"
Code Smell

433.

Catches should be combined
Code Smell

434.

Methods of "Random" that return floating point values should not be used in random integer generation

Code Smell

435.

Parsing should be used to convert "Strings" to primitives
Code Smell

436.

Classes should not be empty
Code Smell

437.

Fields in non-serializable classes should not be "transient"
Code Smell

438.

Boolean checks should not be inverted
Code Smell

439.

Redundant casts should not be used
Code Smell

440.

"@Deprecated" code should not be used
Code Smell

441.

"toString()" should never be called on a String object
Code Smell

442.

Annotation repetitions should not be wrapped
Code Smell

443.

Multiple variables should not be declared on the same line
Code Smell

444.

Strings should not be concatenated using '+' in a loop
Code Smell

445.

Maps with keys that are enum values should be replaced with EnumMap
Code Smell

446.

Lambdas should be replaced with method references
Code Smell

447.

Parentheses should be removed from a single lambda input parameter when its type is inferred
Code Smell

448.

Abstract classes without fields should be converted to interfaces
Code Smell

449.

Lambdas containing only one statement should not nest this statement in a block
Code Smell

450.

"Collections.EMPTY_LIST", "EMPTY_MAP", and "EMPTY_SET" should not be used
Code Smell

451.

Local variables should not be declared and then immediately returned or thrown
Code Smell

452.

Unused local variables should be removed
Code Smell

453.

Private fields only used as local variables in methods should become local variables
Code Smell

454.

"public static" fields should be constant
Code Smell

455.

Loops should not contain more than a single "break" or "continue" statement
Code Smell

456.

Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"
Code Smell

457.

"switch" statements should have at least 3 "case" clauses
Code Smell

458.

A "while" loop should be used instead of a "for" loop
Code Smell

459.

The default unnamed package should not be used
Code Smell

460.

"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method
Code Smell

461.

Package names should comply with a naming convention
Code Smell

462.

Nested code blocks should not be used
Code Smell

463.

Array designators "[]" should be on the type, not the variable
Code Smell

464.

Array designators "[]" should be located after the type in method signatures
Code Smell

465.

Type parameter names should comply with a naming convention
Code Smell

466.

Overriding methods should do more than simply call the same method in the super class
Code Smell

467.

Classes that override "clone" should be "Cloneable" and call "super.clone()"
Code Smell

468.

Public constants and fields initialized at declaration should be "static final" rather than merely "final"
Code Smell

469.

Local variable and method parameter names should comply with a naming convention
Code Smell

470.

Exception classes should be immutable
Code Smell

471.

Field names should comply with a naming convention
Code Smell

472.

Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls
Code Smell

473.

Case insensitive string comparisons should be made without intermediate upper or lower casing
Code Smell

474.

Collection.isEmpty() should be used to test for emptiness
Code Smell

475.

String.valueOf() should not be appended to a String
Code Smell

476.

Interface names should comply with a naming convention
Code Smell

477.

"throws" declarations should not be superfluous
Code Smell

478.

Unnecessary imports should be removed
Code Smell

479.

Return of boolean expressions should not be wrapped into an "if-then-else" statement
Code Smell

480.

Boolean literals should not be redundant
Code Smell

481.

Modifiers should be declared in the correct order
Code Smell

482.

Empty statements should be removed
Code Smell

483.

Class variable fields should not have public accessibility
Code Smell

484.

URIs should not be hardcoded
Code Smell

485.

Class names should comply with a naming convention
Code Smell

486.

Method names should comply with a naming convention
Code Smell

487.

Comma-separated labels should be used in Switch with colon case
Code Smell

488.

JUnit5 test classes and methods should have default package visibility
Code Smell

489.

Track uses of "TODO" tags
Code Smell

490.

Deprecated code should be removed
Code Smell

491.

Annotated Mockito objects should be initialized
Bug

492.

Custom resources should be closed
Bug

493.

Threads should not be started in constructors
Code Smell

494.

"main" should not "throw" anything
Code Smell

495.

Track lack of copyright and license headers
Code Smell

496.

Octal values should not be used
Code Smell

497.

Exit methods should not be called
Code Smell

498.

HTTP response headers should not be vulnerable to injection attacks
Vulnerability

499.

Members of Spring components should be injected
Vulnerability

500.

Classes should not be loaded dynamically
Vulnerability

501.

Equality operators should not be used in "for" loop termination conditions
Code Smell

502.

"Bean Validation" (JSR 380) should be properly configured
Code Smell

503.

Spring beans should be considered by "@ComponentScan"
Code Smell

504.

Number patterns should be regular
Code Smell

505.

Lazy initialization of "static" fields should be "synchronized"
Code Smell

506.

Wildcard imports should not be used
Code Smell

507.

Modulus results should not be checked for direct equality
Code Smell

508.

Comparators should be "Serializable"
Code Smell

509.

"Serializable" classes should have a "serialVersionUID"
Code Smell

510.

"switch" statements and expressions should not be nested
Code Smell

511.

Constructors should only call non-overridable methods
Code Smell

512.

Methods should not be too complex
Code Smell

513.

Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply
Code Smell

514.

"if ... else if" constructs should end with "else" clauses
Code Smell

515.

Control structures should use curly braces
Code Smell

516.

Expressions should not be too complex
Code Smell

517.

Mockito argument matchers should be used on all parameters
Bug

518.

Spring "@Controller" classes should not use "@Scope"
Bug

519.

Constructor injection should be used instead of field injection
Bug

520.

Classes that don't define "hashCode()" should not be used in hashes
Bug

521.

Floating point numbers should not be tested for equality
Bug

522.

Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression
Code Smell

523.

Limited dependence should be placed on operator precedence
Code Smell

524.

Custom getter method should not be used to override record's getter behavior
Code Smell

525.

Tests should use fixed data instead of randomized data
Code Smell

526.

Spring's ModelAndViewAssert assertions should be used instead of other assertions
Code Smell

527.

Lambdas should not have too many lines
Code Smell

528.

"@EnableAutoConfiguration" should be fine-tuned
Code Smell

529.

Enum values should be compared with "=="
Code Smell

530.

Spring components should use constructor injection
Code Smell

531.

Regex patterns should not be created needlessly
Code Smell

532.

Track uses of disallowed constructors
Code Smell

533.

Java 8's "Files.exists" should not be used
Code Smell

534.

"Optional" should not be used for parameters

Code Smell

535.

Tests should be kept in a dedicated source directory
Code Smell

536.

"this" should not be exposed from constructors
Code Smell

537.

Classes should not have too many "static" imports
Code Smell

538.

Escaped Unicode characters should not be used
Code Smell

539.

Inner classes should not have too many lines of code
Code Smell

540.

Inner classes which do not reference their owning classes should be "static"
Code Smell

541.

"deleteOnExit" should not be used
Code Smell

542.

Public methods should not contain selector arguments
Code Smell

543.

Java parser failure
Code Smell

544.

Track uses of disallowed methods
Code Smell

545.

Types should be used in lambdas
Code Smell

546.

"java.time" classes should be used for dates and times
Code Smell

547.

The names of methods with boolean return values should start with "is" or "has"
Code Smell

548.

Files should contain only one top-level class or interface each
Code Smell

549.

Classes should not have too many fields
Code Smell

550.

The ternary operator should not be used
Code Smell

551.

Standard functional interfaces should not be redefined

Code Smell

552.

"NullPointerException" should not be caught
Code Smell

553.

"NullPointerException" should not be explicitly thrown
Code Smell

554.

Classes should not have too many methods
Code Smell

555.

Methods should not have too many lines
Code Smell

556.

Track uses of "NOSONAR" comments
Code Smell

557.

Classes and enums with private members should have a constructor
Code Smell

558.

Track comments matching a regular expression
Code Smell

559.

Statements should be on separate lines
Code Smell

560.

Classes should not be coupled to too many other classes (Single Responsibility Principle)
Code Smell

561.

"java.lang.Error" should not be extended
Code Smell

562.

Anonymous classes should not have too many lines
Code Smell

563.

Public types, methods and fields (API) should be documented with Javadoc
Code Smell

564.

Exception handlers should preserve the original exceptions
Code Smell

565.

Checked exceptions should not be thrown
Code Smell

566.

Public methods should throw at most one checked exception
Code Smell

567.

"switch case" clauses should not have too many lines of code
Code Smell

568.

Methods should not have too many return statements

Code Smell

569.

Magic numbers should not be used
Code Smell

570.

Files should not have too many lines of code
Code Smell

571.

Lines should not be too long
Code Smell

572.

JEE applications should not "getClassLoader"
Bug

573.

Math should not be performed on floats
Bug

574.

"equals" methods should be symmetric and work for subclasses
Bug

575.

Literal suffixes should be upper case
Code Smell

576.

Unicode-aware versions of character classes should be preferred
Code Smell

577.

Use Java 12 "switch" expression
Code Smell

578.

"serialVersionUID" should not be declared blindly
Code Smell

579.

"Stream.collect()" calls should not be redundant
Code Smell

580.

Local constants should follow naming conventions for constants
Code Smell

581.

Unit tests should throw exceptions
Code Smell

582.

Test methods should comply with a naming convention
Code Smell

583.

Value-based objects should not be serialized
Code Smell

584.

Default annotation parameter values should not be passed as arguments
Code Smell

585.

Method parameters should be declared with base types

Code Smell

586.
Fields should not be initialized to default values
Code Smell

587.
Multiple loops over the same set should be combined
Code Smell

588.
Classes without "public" constructors should be "final"
Code Smell

589.
Unnecessary semicolons should be omitted
Code Smell

590.
Literal boolean values and nulls should not be used in assertions
Code Smell

591.
Test assertions should include messages
Code Smell

592.
Mutable members should not be stored or returned directly
Code Smell

593.
Redundant modifiers should not be used
Code Smell

594.
"private" and "final" methods that don't access instance data should be "static"
Code Smell

595.
Files should not be empty
Code Smell

596.
Collection methods with O(n) performance should be used carefully
Code Smell

597.
"Exception" should not be caught when not required by called methods
Code Smell

598.
"collect" should be used with "Streams" instead of "list::add"
Code Smell

599.
Switches should be used for sequences of simple "String" tests
Code Smell

600.
"final" classes should not have "protected" members
Code Smell

601.
Underscores should be used to make large numbers readable
Code Smell

602.
"Serializable" inner classes of "Serializable" classes should be static

Code Smell

603.
Member variable visibility should be specified
Code Smell

604.
Classes and methods that rely on the default system encoding should not be used
Code Smell

605.
Simple class names should be used
Code Smell

606.
Variables should not be declared before they are relevant
Code Smell

607.
Extensions and implementations should not be redundant
Code Smell

608.
"==" and "!=" should not be used when "equals" is overridden
Code Smell

609.
An abstract class should have both abstract and concrete methods
Code Smell

610.
Sets with elements that are enum values should be replaced with EnumSet
Code Smell

611.
String operations should not rely on the default system locale
Code Smell

612.
Comments should not be located at the end of lines of code
Code Smell

613.
Track uses of "CHECKSTYLE:OFF" suppression comments
Code Smell

614.
Loggers should be "private static final" and should share a naming convention
Code Smell

615.
Track uses of "NOPMD" suppression comments
Code Smell

616.
Packages should have a javadoc file 'package-info.java'
Code Smell

617.
The members of an interface or class declaration should appear in a pre-defined order
Code Smell

618.
Abstract class names should comply with a naming convention
Code Smell

619.
Strings literals should be placed on the left side when checking for equality

Code Smell

620.
Files should contain an empty newline at the end
Code Smell

621.
Source code should be indented consistently
Code Smell

622.
A close curly brace should be located at the beginning of a line
Code Smell

623.
Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines
Code Smell

624.
Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line
Code Smell

625.
An open curly brace should be located at the beginning of a line
Code Smell

626.
An open curly brace should be located at the end of a line
Code Smell

627.
Tabulation characters should not be used
Code Smell

628.
Functions should not be defined with a variable number of arguments
Code Smell

629.
Local-Variable Type Inference should be used
Code Smell

630.
Migrate your tests from JUnit4 to the new JUnit5 annotations
Code Smell

631.
Track uses of disallowed classes
Code Smell

632.
Track uses of "@SuppressWarnings" annotations
Code Smell