Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
HTML
Java
JavaScript
Kotlin
Objective C
PHP
PL/I
PL/SQL
Python
RPG
Ruby
Scala
Swift
Terraform
Text
TypeScript
T-SQL
VB.NET
VB6
XML

Java static code analysis

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

All rules 632 | Vulnerability 53 | Bug 154 | Security Hotspot 36 | Code Smell 389 | Quick Fix 42

Tags

Search by name...

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

Composed "@RequestMapping" variants should be preferred
Code Smell

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

Functional Interfaces should be as specialised as possible
Code Smell

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

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

"ThreadLocal.withInitial" should be preferred
Code Smell

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

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

Arrays should not be created for varargs parameters
Code Smell

Jump statements should not be redundant
Code Smell

Test classes should comply with a naming convention

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

Analyze your code

Bug | Major | multi-threading cert

Objects which are pooled and potentially reused should not be used for synchronization. If they are, it can cause unrelated threads to deadlock with unhelpful stacktraces. Specifically, `String` literals, and boxed primitives such as Integers should not be used as lock objects because they are pooled and reused. The story is even worse for `Boolean` objects, because there could possibly be only two instances of `Boolean`, `Boolean.TRUE` and `Boolean.FALSE` and every class that uses a Boolean will be referring to one of the two.

Here is the list of types which shouldn't be used for synchronization:

- The primitive wrapper classes in java.lang;
- The class java.lang.Runtime.Version;
- The "optional" classes in java.util: Optional, OptionalInt, OptionalLong, and OptionalDouble;
- Many classes in the java.time API: Instant, LocalDate, LocalTime, LocalDateTime, ZonedDateTime, ZoneId, OffsetTime, OffsetDateTime, ZoneOffset, Duration, Period, Year, YearMonth, and MonthDay, and, in java.time.chrono: MinguoDate, HijrahDate, JapaneseDate, and ThaiBuddhistDate;
- The interface java.lang.ProcessHandle and its implementation classes;
- The implementation classes of the collection factories in java.util: List.of, List.copyOf, Set.of, Set.copyOf, Map.of, Map.copyOf, Map.ofEntries, and Map.entry.

Noncompliant Code Example

```
private static final Boolean bLock = Boolean.FALSE;
private static final Integer iLock = Integer.valueOf(0);
private static final String sLock = "LOCK";
private static final List<String> listLock = List.of("a", "b

public void doSomething() {

  synchronized(bLock) {  // Noncompliant
    // ...
  }
  synchronized(iLock) {  // Noncompliant
    // ...
  }
  synchronized(sLock) {  // Noncompliant
    // ...
  }
  synchronized(listLock) {  // Noncompliant
    // ...
  }
```

Compliant Solution

Code Smell

Loggers should be named for their enclosing classes

Code Smell

Methods should not return constants

Code Smell

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

Code Smell

"enum" fields should not be publicly mutable

```
private static final Object lock1 = new Object();
private static final Object lock2 = new Object();
private static final Object lock3 = new Object();
private static final Object lock4 = new Object();

public void doSomething() {

  synchronized(lock1) {
    // ...
  }
  synchronized(lock2) {
    // ...
  }
  synchronized(lock3) {
    // ...
  }
  synchronized(lock4) {
    // ...
  }
```

See

- CERT, LCK01-J. - Do not synchronize on objects that may be reused
- JEP-390. - JEP 390: Warnings for Value-Based Classes

Available In:

sonarlint | sonarcloud | sonarqube