sonar RULES

Products

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...

Sections of code should not be commented out
Code Smell

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

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

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

Throwable and Error should not be caught
Code Smell

Unused method parameters should be removed
Code Smell

Only static class initializers should be used
Code Smell

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

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

Enumeration should not be implemented
Code Smell

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

"volatile" variables should not be used with compound operators

Analyze your code

Bug Major multi-threading cert

Using compound operators as well as increments and decrements (and toggling, in the case of `booleans`) on primitive fields are not atomic operations. That is, they don't happen in a single step. For instance, when a `volatile` primitive field is incremented or decremented you run the risk of data loss if threads interleave in the steps of the update. Instead, use a guaranteed-atomic class such as `AtomicInteger`, or synchronize the access.

Noncompliant Code Example

```
private volatile int count = 0;
private volatile boolean boo = false;

public void incrementCount() {
  count++;  // Noncompliant
}

public void toggleBoo(){
  boo = !boo;  // Noncompliant
}
```

Compliant Solution

```
private AtomicInteger count = 0;
private boolean boo = false;

public void incrementCount() {
  count.incrementAndGet();
}

public synchronized void toggleBoo() {
  boo = !boo;
}
```

See

- CERT, VNA02-J. - Ensure that compound operations on shared variables are atomic

Available In:

sonarlint sonarcloud sonarqube

Unused "private" methods should be removed

Code Smell

Try-catch blocks should not be nested

Code Smell

Track uses of "FIXME" tags

Code Smell

Deprecated elements should have both the annotation and the Javadoc tag

Code Smell

Assignments should not be made