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...

Test assertions should include messages
Code Smell

Mutable members should not be stored or returned directly
Code Smell

Redundant modifiers should not be used
Code Smell

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

Files should not be empty
Code Smell

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

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

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

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

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

Underscores should be used to make large numbers readable
Code Smell

"Serializable" inner classes of

"for" loop stop conditions should be invariant

Analyze your code

Code Smell Major pitfall

A `for` loop stop condition should test the loop counter against an invariant value (i.e. one that is true at both the beginning and ending of every loop iteration). Ideally, this means that the stop condition is set to a local variable just before the loop begins.

Stop conditions that are not invariant are slightly less efficient, as well as being difficult to understand and maintain, and likely lead to the introduction of errors in the future.

This rule tracks three types of non-invariant stop conditions:

- When the loop counters are updated in the body of the `for` loop
- When the stop condition depend upon a method call
- When the stop condition depends on an object property, since such properties could change during the execution of the loop.

Noncompliant Code Example

```
for (int i = 0; i < 10; i++) {
  ...
  i = i - 1; // Noncompliant; counter updated in the body of
  ...
}
```

Compliant Solution

```
for (int i = 0; i < 10; i++) {...}
```

Available In:

sonarlint | sonarcloud | sonarqube

"Serializable" classes should be static

Code Smell

Member variable visibility should be specified

Code Smell

Classes and methods that rely on the default system encoding should not be used

Code Smell

Simple class names should be used

Code Smell