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...

Abstract class names should comply with a naming convention
Code Smell

Strings literals should be placed on the left side when checking for equality
Code Smell

Files should contain an empty newline at the end
Code Smell

Source code should be indented consistently
Code Smell

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

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

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

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

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

Tabulation characters should not be used
Code Smell

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

AssertJ assertions "allMatch" and "doesNotContains" should also test for emptiness

Analyze your code

Bug Minor tests assertj

AssertJ assertions `allMatch` and `doesNotContains` on an empty list always returns true whatever the content of the predicate. Despite being correct, you should make explicit if you expect an empty list or not, by adding `isEmpty()/isNotEmpty()` in addition to calling the assertion, or by testing the list's content further. It will justify the useless predicate to improve clarity or increase the reliability of the test.

This rule raises an issue when any of the methods listed are used without asserting that the list is empty or not and without testing the content.

Targetted methods:

- `allMatch`
- `allSatisfy`
- `doesNotContain`
- `doesNotContainSequence`
- `doesNotContainSubsequence`
- `doesNotContainAnyElementsOf`

Noncompliant Code Example

```
List<String> logs = getLogs();

assertThat(logs).allMatch(e -> e.contains("error")); // Nonc
assertThat(logs).doesNotContain("error"); // Noncompliant, d
```

Compliant Solution

```
List<String> logs = getLogs();

assertThat(logs).isNotEmpty().allMatch(e -> e.contains("erro
// Or
assertThat(logs).hasSize(5).allMatch(e -> e.contains("error"
// Or
assertThat(logs).isEmpty();

// Despite being redundant, this is also acceptable since it
assertThat(logs).doesNotContain("error").isEmpty();
// or test the content of the list further
assertThat(logs).contains("warning").doesNotContain("error")
```

Available In:

sonarlint sonarcloud sonarqube

Local-Variable Type Inference should be used

Code Smell

Migrate your tests from JUnit4 to the new JUnit5 annotations

Code Smell

Track uses of disallowed classes

Code Smell

Track uses of "@SuppressWarnings" annotations

Code Smell