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...

"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

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

Code Smell

Abstract methods should not be redundant

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

Analyze your code

Bug Major cert

Using the same value on either side of a binary operator is almost always a mistake. In the case of logical operators, it is either a copy/paste error and therefore a bug, or it is simply wasted code, and should be simplified. In the case of bitwise operators and most binary mathematical operators, having the same value on both sides of an operator yields predictable results, and should be simplified.

Noncompliant Code Example

```
if ( a == a ) { // always true
  doZ();
}
if ( a != a ) { // always false
  doY();
}
if ( a == b && a == b ) { // if the first one is true, the s
  doX();
}
if ( a == b || a == b ) { // if the first one is true, the s
  doW();
}

int j = 5 / 5; //always 1
int k = 5 - 5; //always 0

c.equals(c); //always true
```

Exceptions

- This rule ignores `*`, `+`, and `=`.
- The specific case of testing a floating point value against itself is a valid test for `NaN` and is therefore ignored.
- Similarly, left-shifting 1 onto 1 is common in the construction of bit masks, and is ignored.

```
float f;
if(f != f) { //test for NaN value
  System.out.println("f is NaN");
}

int i = 1 << 1; // Compliant
int j = a << a; // Noncompliant
```

See

- CERT, MSC12-C. - Detect and remove code that has no effect or is never executed
- {rule:java:S1656} - Implements a check on `=`.

Code Smell

Arrays should not be copied using loops

Code Smell

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

Code Smell

JUnit rules should be used

Code Smell

Nested "enum"s should not be declared static

Code Smell

Available In:

sonarlint | sonarcloud | sonarqube