

































-  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 rules632

Vulnerability53

Bug154

Security Hotspot36

Code Smell389

Quick Fix42

Tags ▾

Search by name... 🔍

should not be used

Bug

Conditionally executed code should be reachable

Bug

"notifyAll" should be used

Bug

Blocks should be synchronized on "private final" fields

Bug

Non-serializable objects should not be stored in "HttpSession" objects

Bug

"wait", "notify" and "notifyAll" should only be called when a lock is obviously held on an object

Bug

Null pointers should not be dereferenced

Bug

Loop conditions should be true at least once

Bug

A "for" loop update clause should move the counter in the right direction

Bug

Non-public methods should not be "@Transactional"

Bug

Servlets should not have mutable instance fields

Bug

"toString()" and "clone()" methods should not return null

Bug

Conditionals should start on new lines

Analyze your code

Code SmellCritical?suspicious

Code is clearest when each statement has its own line. Nonetheless, it is a common pattern to combine on the same line an *if* and its resulting *then* statement. However, when an *if* is placed on the same line as the closing *}* from a preceding *then*, *else* or *else if* part, it is either an error - *else* is missing - or the invitation to a future error as maintainers fail to understand that the two statements are unconnected.

Noncompliant Code Example

```
if (condition1) {
    // ...
} if (condition2) { // Noncompliant
    //...
}
```

Compliant Solution

```
if (condition1) {
    // ...
} else if (condition2) {
    //...
}
```

Or






```
if (condition1) {
    // ...
}

if (condition2) {
    //...
}
```

Available In:

sonarlint | sonarcloud | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.
[Privacy Policy](#)

 Bug
<p>".equals()" should not be used to test the values of "Atomic" classes</p> <p> Bug</p>
<p>Return values from functions without side effects should not be ignored</p> <p> Bug</p>
<p>Child class methods named for parent class methods should be overrides</p> <p> Bug</p>
<p>Inappropriate "Collection" calls should not be made</p> <p> Bug</p>