




 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... 🔍

"@CheckForNull" or "@Nullable" should not be used on primitive types

Code Smell

Composed "@RequestMapping" variants should be preferred

Code Smell

"write(byte[],int,int)" should be overridden

Code Smell

Functional Interfaces should be as specialised as possible

Code Smell

Null checks should not be used with "instanceof"

Code Smell

"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

Related "if/else if" statements should not have the same condition

Analyze your code

Bug

Major

cert unused pitfall

A chain of `if/else if` statements is evaluated from top to bottom. At most, only one branch will be executed: the first one with a condition that evaluates to `true`.

Therefore, duplicating a condition automatically leads to dead code. Usually, this is due to a copy/paste error. At best, it's simply dead code and at worst, it's a bug that is likely to induce further bugs as the code is maintained, and obviously it could lead to unexpected behavior.

Noncompliant Code Example

```
if (param == 1)
    openWindow();
else if (param == 2)
    closeWindow();
else if (param == 1) // Noncompliant
    moveWindowToTheBackground();
}
```

Compliant Solution

```
if (param == 1)
    openWindow();
else if (param == 2)
    closeWindow();
else if (param == 3)
    moveWindowToTheBackground();
}
```

See

- [CERT, MSC12-C](#) - Detect and remove code that has no effect or is never executed

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](#)

https://rules.sonarsource.com/java/RSPEC-1862

1/2

 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