




 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... 🔍

Generic exceptions should never be thrown

Code Smell

Labels should not be used

Code Smell

Utility classes should not have public constructors

Code Smell

Local variables should not shadow class fields

Code Smell

Redundant pairs of parentheses should be removed

Code Smell

Inheritance tree of classes should not be too deep

Code Smell

Nested blocks of code should not be left empty

Code Smell

Methods should not have too many parameters

Code Smell

Unused "private" fields should be removed

Code Smell

Collapsible "if" statements should be merged

Code Smell

Unused labels should be removed

Code Smell

Standard outputs should not be used directly to log anything

Code Smell

Unary prefix operators should not be repeated

Analyze your code

Bug

Major

?

The needless repetition of an operator is usually a typo. There is no reason to write `!!!i` when `!i` will do.

On the other hand, the repetition of increment and decrement operators may have been done on purpose, but doing so obfuscates the meaning, and should be simplified.

This rule raises an issue for sequences of: `!`, `~`, `-`, and `+`.

Noncompliant Code Example

```
int i = 1;

int j = - - -i; // Noncompliant; just use -i
int k = ~~~i;   // Noncompliant; same as i
int m = + +i;   // Noncompliant; operators are useless here

boolean b = false;
boolean c = !!!b; // Noncompliant
```

Compliant Solution

```
int i = 1;

int j = -i;
int k = ~i;
int m = i;

boolean b = false;
boolean c = !b;
```

Exceptions

Overflow handling for GWT compilation using `~~` is ignored.

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-2761

1/2

<div>OS commands should not be vulnerable to argument injection attacks</div> <div> Vulnerability</div>
<div>"ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization</div> <div> Vulnerability</div>
<div>Logging should not be vulnerable to injection attacks</div> <div> Vulnerability</div>
<div>Exceptions should not be thrown from servlet methods</div>