

































-  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... 🔍

Map "computeIfAbsent()" and "computeIfPresent()" should not be used to add "null" values.

Bug

Regex lookahead assertions should not be contradictory

Bug

Back references in regular expressions should only refer to capturing groups that are matched before the reference

Bug

Regex boundaries should not be used in a way that can never be matched

Bug

Regex patterns following a possessive quantifier should not always fail

Bug

Regular expressions should be syntactically valid

Bug

Assertions comparing incompatible types should not be made

Bug

JUnit5 inner test classes should be annotated with @Nested

Bug

Only one method invocation is expected when testing checked exceptions

Bug

Assertion methods should not be used within the try block of a try-catch catching an Error

Bug

Getters and setters should access the expected fields

Short-circuit logic should be used in boolean contexts

Analyze your code

Code SmellBlocker?cert

The use of non-short-circuit logic in a boolean context is likely a mistake - one that could cause serious program errors as conditions are evaluated under the wrong circumstances.

Noncompliant Code Example

```
if(getTrue() | getFalse()) { ... } // Noncompliant; both sides should be short-circuited
```

Compliant Solution

```
if(getTrue() || getFalse()) { ... } // true short-circuit logic
```

See

- [CERT, EXP46-C](#) - Do not use a bitwise operator with a Boolean-like operand





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-2178

1/2

Expected result
 Bug
Zero should not be a possible denominator
 Bug
Locks should be released
 Bug
"runFinalizersOnExit" should not be called
 Bug
"ScheduledThreadPoolExecutor" should not have 0 core threads