




 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... 🔍

inside a "while" loop

Code Smell

IllegalMonitorStateException should not be caught

Code Smell

JUnit assertions should not be used in "run" methods

Code Smell

Class names should not shadow interfaces or superclasses

Code Smell

"Cloneables" should implement "clone"

Code Smell

Try-with-resources should be used

Code Smell

"readResolve" methods should be inheritable

Code Smell

"for" loop increment clauses should modify the loops' counters

Code Smell

Fields in a "Serializable" class should either be transient or serializable

Code Smell

Package declaration should match source file directory

Code Smell

Generic wildcard types should not be used in return types

Code Smell

"switch" statements should have "default" clauses

Code Smell

Regex lookahead assertions should not be contradictory

Analyze your code

BugCritical?regex

Lookahead assertions are a regex feature that makes it possible to look ahead in the input without consuming it. It is often used at the end of regular expressions to make sure that substrings only match when they are followed by a specific pattern.

However, they can also be used in the middle (or at the beginning) of a regex. In that case there is the possibility that what comes after the lookahead does not match the pattern inside the lookahead. This makes the lookahead impossible to match and is a sign that there's a mistake in the regular expression that should be fixed.

Noncompliant Code Example

```
Pattern.compile("(?=a)b"); // Noncompliant, the same character
```

Compliant Solution

```
Pattern.compile("(?<=a)b");
Pattern.compile("a(?:=b)");
```





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-6002

1/2

Execution of the Garbage Collector should be triggered only by the JVM  Code Smell
Constants should not be defined in interfaces  Code Smell
String literals should not be duplicated  Code Smell
Methods should not be empty  Code Smell
"Object.finalize()" should remain protected (versus public) when