




 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... 🔍

Generic wildcard types should not be used in return types

Code Smell

"switch" statements should have "default" clauses

Code Smell

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 overriding

Code Smell

Exceptions should not be thrown in finally blocks

Code Smell

Constant names should comply with a naming convention

Code Smell

The Object.finalize() method should not be overridden

Code Smell

XML operations should not be vulnerable to injection attacks

Vulnerability

JSON operations should not be vulnerable to injection attacks

Regular expressions should be syntactically valid

Analyze your code

BugCritical?regex

Regular expressions have their own syntax that is understood by regular expression engines. Those engines will throw an exception at runtime if they are given a regular expression that does not conform to that syntax.

To avoid syntax errors, special characters should be escaped with backslashes when they are intended to be matched literally and references to capturing groups should use the correctly spelled name or number of the group.

To match a literal string instead of a regular expression, either all special characters should be escaped, the `Pattern.LITERAL` flag or methods that don't use regular expressions should be used.

Noncompliant Code Example

```
Pattern.compile("{");
str.matches("{");
str.replaceAll("{", "{}");
str.matches("\\w+-(\\d+)");
```

Compliant Solution

```
Pattern.compile("\\{\\{");
Pattern.compile("{", Pattern.LITERAL);
str.equals("{");
str.replace("{", "{}");
str.matches("\\w+-(\\d+)");
```

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-5856

1/2

 Vulnerability
XML signatures should be validated securely  Vulnerability
XML parsers should not be vulnerable to Denial of Service attacks  Vulnerability
XML parsers should not load external schemas  Vulnerability
Mobile database encryption keys should not be disclosed