

sonar


RULES


Products


▼


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

## 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...

Overriding methods should do more than simply call the same method in the super class

Code Smell

Classes that override "clone" should be "Cloneable" and call "super.clone()"

Code Smell

Public constants and fields initialized at declaration should be "static final" rather than merely "final"

Code Smell

Local variable and method parameter names should comply with a naming convention

Code Smell

Exception classes should be immutable

Code Smell

Field names should comply with a naming convention

Code Smell

Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls

Code Smell

Case insensitive string comparisons should be made without intermediate upper or lower casing

Code Smell

Collection.isEmpty() should be used to test for emptiness

Code Smell

String.valueOf() should not be appended to a String

Code Smell

Interface names should comply with a

### Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string

Analyze your code

Code Smell

Major

regex

When a reluctant quantifier (such as \*? or +?) is followed by a pattern that can match the empty string or directly by the end of the regex, it will always match the empty string when used with methods that find partial matches (such as find, replaceAll, split etc.).

Similarly, when used with methods that find full matches, a reluctant quantifier that's followed directly by the end of the regex (or a pattern that always matches the empty string, such as ()) behaves indistinguishably from a greedy quantifier while being less efficient.

This is likely a sign that the regex does not work as intended.

Noncompliant Code Example

```
"start123endstart456".replaceAll("start\\w*(end)?", "x");  
str.matches("\\d*"); // Noncompliant. Matches the same as "
```

Compliant Solution

```
"start123endstart456".replaceAll("start\\w*(end|$)", "x");  
str.matches("\\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-6019

1/2

<div><div>naming convention</div><div> Code Smell</div></div>
<div><div>"throws" declarations should not be superfluous</div><div> Code Smell</div></div>
<div><div>Unnecessary imports should be removed</div><div> Code Smell</div></div>
<div><div>Return of boolean expressions should not be wrapped into an "if-then-else" statement</div><div> Code Smell</div></div>