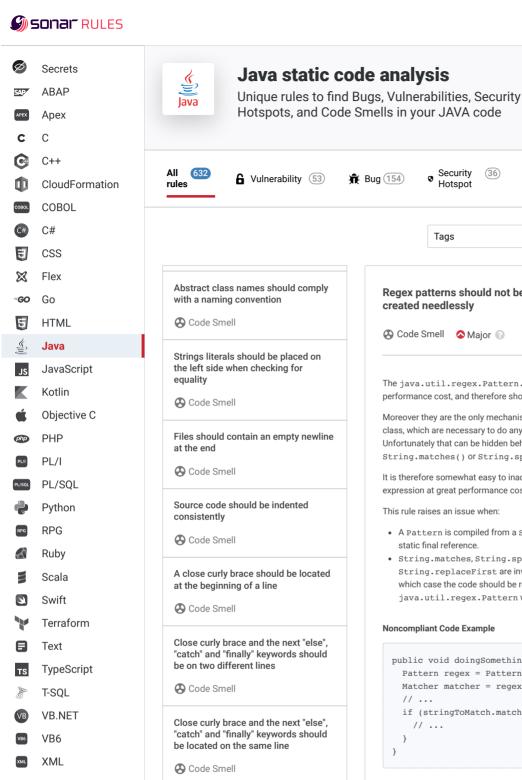
Products >



Code Smell

at the end of a line

Code Smell

Code Smell

Code Smell

Functions should not be defined with

a variable number of arguments

used

```
An open curly brace should be located
at the beginning of a line
An open curly brace should be located
Tabulation characters should not be
```

```
⊗ Code
                                                             Quick 42 Fix
                    Security
                              (36)
                                                 (389)
# Bug (154)
                   Hotspot
                                          Smell
                                                 Search by name.
                   Tags
       Regex patterns should not be
                                                       Analyze your code
       created needlessly
       regex performance
       The java.util.regex.Pattern.compile() methods have a significant
       performance cost, and therefore should be used sensibly
       Moreover they are the only mechanism available to create instances of the Pattern
       class, which are necessary to do any pattern matching using regular expressions.
       Unfortunately that can be hidden behind convenience methods like
       String.matches() or String.split().
       It is therefore somewhat easy to inadvertently repeatedly compile the same regular
       expression at great performance cost with no valid reason.
       This rule raises an issue when:
        • A Pattern is compiled from a String literal or constant and is not stored in a
           static final reference.
        • String.matches, String.split, String.replaceAll or
           String.replaceFirst are invoked with a String literal or constant. In
           which case the code should be refactored to use a
           java.util.regex.Pattern while respecting the previous rule.
       Noncompliant Code Example
         public void doingSomething(String stringToMatch) {
           Pattern regex = Pattern.compile("myRegex"); // Noncomplia
           Matcher matcher = regex.matcher("s");
           if (stringToMatch.matches("myRegex2")) { // Noncompliant
         }
       Compliant Solution
```

## Exceptions

}

// ...

 ${\tt String.split}\ doesn't\ create\ a\ regex\ when\ the\ string\ passed\ as\ argument\ meets$ either of these 2 conditions:

private static final Pattern myRegex = Pattern.compile("myRe

private static final Pattern myRegex2 = Pattern.compile("myR

public void doingSomething(String stringToMatch) {

if (myRegex2.matcher(stringToMatch).matches()) {

Matcher matcher = myRegex.matcher("s");

Local-Variable Type Inference should be used Code Smell Migrate your tests from JUnit4 to the new JUnit5 annotations Code Smell Track uses of disallowed classes Code Smell Track uses of "@SuppressWarnings" annotations Code Smell

- It is a one-char String and this character is not one of the RegEx's meta characters ".\$|()[{^?\*+\"
- It is a two-char String and the first char is the backslash and the second is not the ascii digit or ascii letter.

In which case no issue will be raised.

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