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 static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

All rules 632 | Vulnerability 53 | Bug 154 | Security Hotspot 36 | Code Smell 389 | Quick Fix 42

Tags | Search by name...

to test for emptiness
Code Smell

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

Interface names should comply with a naming convention
Code Smell

"throws" declarations should not be superfluous
Code Smell

Unnecessary imports should be removed
Code Smell

Return of boolean expressions should not be wrapped into an "if-then-else" statement
Code Smell

Boolean literals should not be redundant
Code Smell

Modifiers should be declared in the correct order
Code Smell

Empty statements should be removed
Code Smell

Class variable fields should not have public accessibility
Code Smell

URIs should not be hardcoded
Code Smell

Class names should comply with a naming convention
Code Smell

Character classes in regular expressions should not contain the same character twice

Analyze your code

Code Smell   Major   regex

Character classes in regular expressions are a convenient way to match one of several possible characters by listing the allowed characters or ranges of characters. If the same character is listed twice in the same character class or if the character class contains overlapping ranges, this has no effect.

Thus duplicate characters in a character class are either a simple oversight or a sign that a range in the character class matches more than is intended or that the author misunderstood how character classes work and wanted to match more than one character. A common example of the latter mistake is trying to use a range like `[0-99]` to match numbers of up to two digits, when in fact it is equivalent to `[0-9]`. Another common cause is forgetting to escape the `-` character, creating an unintended range that overlaps with other characters in the character class.

Noncompliant Code Example

```
str.matches("[0-99]") // Noncompliant, this won't actually m
str.matches("[0-9.-_]") // Noncompliant, .-_ is a range that
```

Compliant Solution

```
str.matches("[0-9]{1,2}")
str.matches("[0-9.\\-_]")
```

Available In:
sonarlint | sonarcloud | sonarqube

Method names should comply with a naming convention

Code Smell

Comma-separated labels should be used in Switch with colon case

Code Smell

JUnit5 test classes and methods should have default package visibility

Code Smell

Track uses of "TODO" tags

Code Smell