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...

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

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

Regex patterns following a possessive quantifier should not always fail

Analyze your code

Bug Critical regex

Possessive quantifiers in Regex patterns like below improve performance by eliminating needless backtracking:

```
?+ , *+ , ++ , {n}+ , {n,}+ , {n,m}+
```

But because possessive quantifiers do not keep backtracking positions and never give back, the following sub-patterns should not match only similar characters. Otherwise, possessive quantifiers consume all characters that could have matched the following sub-patterns and nothing remains for the following sub-patterns.

Noncompliant Code Example

```
Pattern pattern1 = Pattern.compile("a++abc");       // Nonco
Pattern pattern2 = Pattern.compile("\\d*+[02468]"); // Nonco
```

Compliant Solution

```
Pattern pattern1 = Pattern.compile("aa++bc");           //
Pattern pattern2 = Pattern.compile("\\d*+(?<=[02468])"); //
```

Available In:

sonarlint sonarcloud sonarqube

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

Vulnerability