 sonar RULES

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... 🔍

Abstract class names should comply with a naming convention

Code Smell

Strings literals should be placed on the left side when checking for equality

Code Smell

Files should contain an empty newline at the end

Code Smell

Source code should be indented consistently

Code Smell

A close curly brace should be located at the beginning of a line

Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines

Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line

Code Smell

An open curly brace should be located at the beginning of a line

Code Smell

An open curly brace should be located at the end of a line

Code Smell

Tabulation characters should not be used

Code Smell

Functions should not be defined with a variable number of arguments

Code Smell

Value-based objects should not be serialized

Analyze your code

Code Smell

Minor

serialization java8 lock-in

According to the documentation,

A program may produce unpredictable results if it attempts to distinguish two references to equal values of a value-based class, whether directly via reference equality or indirectly via an appeal to synchronization, identity hashing, serialization...

For example (credit to Brian Goetz), imagine Foo is a value-based class:

```
Foo[] arr = new Foo[2];
arr[0] = new Foo(0);
arr[1] = new Foo(0);
```

Serialization promises that on deserialization of arr, elements 0 and 1 will not be aliased. Similarly, in:

```
Foo[] arr = new Foo[2];
arr[0] = new Foo(0);
arr[1] = arr[0];
```

Serialization promises that on deserialization of arr, elements 0 and 1 **will** be aliased.

While these promises are coincidentally fulfilled in current implementations of Java, that is not guaranteed in the future, particularly when true value types are introduced in the language.

This rule raises an issue when a `Serializable` class defines a non-transient, non-static field whose type is a known serializable value-based class. Known serializable value-based classes are: all the classes in the `java.time` package except `Clock`; the date classes for alternate calendars: `HijrahDate`, `JapaneseDate`, `MinguoDate`, `ThaiBuddhistDate`.

Noncompliant Code Example

```
class MyClass implements Serializable {
    private HijrahDate date; // Noncompliant; mark this trans
    // ...
}
```


Compliant Solution

```
class MyClass implements Serializable {
    private transient HijrahDate date;
    // ...
}
```


https://rules.sonarsource.com/java/RSPEC-3437

1/2


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

See

- [Value-based classes](#)

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](#)