

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... 🔍

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

"private" and "final" methods that don't access instance data should be "static"

Analyze your code

Code Smell

Minor

Quick Fix

pitfall

Non-overridable methods (private or final) that don't access instance data can be static to prevent any misunderstanding about the contract of the method.

Noncompliant Code Example

```
class Utilities {
    private static String magicWord = "magic";

    private String getMagicWord() { // Noncompliant
        return magicWord;
    }

    private void setMagicWord(String value) { // Noncompliant
        magicWord = value;
    }
}
```

Compliant Solution

```
class Utilities {
    private static String magicWord = "magic";

    private static String getMagicWord() {
        return magicWord;
    }

    private static void setMagicWord(String value) {
        magicWord = value;
    }
}
```

Exceptions

When java.io.Serializable is implemented the following three methods are excluded by the rule:

- private void writeObject(java.io.ObjectOutputStream out) throws IOException;
- private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException;
- private void readObjectNoData() throws ObjectStreamException;

Available In:
sonarlint | sonarcloud | sonarqube

https://rules.sonarsource.com/java/RSPEC-2325

1/2

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

© 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](#)