




 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... 🔍

"Streams" instead of "list::add"

Code Smell

Switches should be used for sequences of simple "String" tests

Code Smell

"final" classes should not have "protected" members

Code Smell

Underscores should be used to make large numbers readable

Code Smell

"Serializable" inner classes of "Serializable" classes should be static

Code Smell

Member variable visibility should be specified

Code Smell

Classes and methods that rely on the default system encoding should not be used

Code Smell

Simple class names should be used

Code Smell

Variables should not be declared before they are relevant

Code Smell

Extensions and implementations should not be redundant

Code Smell

"==" and "!=" should not be used when "equals" is overridden

Code Smell

An abstract class should have both abstract and concrete methods

Code Smell

Classes from "sun.*" packages should not be used

Analyze your code

Code Smell

Major ?

lock-in pitfall

Classes in the sun.* or com.sun.* packages are considered implementation details, and are not part of the Java API.

They can cause problems when moving to new versions of Java because there is no backwards compatibility guarantee. Similarly, they can cause problems when moving to a different Java vendor, such as OpenJDK.

Such classes are almost always wrapped by Java API classes that should be used instead.

Noncompliant Code Example

```
import com.sun.jna.Native; // Noncompliant
import sun.misc.BASE64Encoder; // Noncompliant
```

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](#)

https://rules.sonarsource.com/java/RSPEC-1191

1/2

<div><div></div></div>
<div><div>Sets with elements that are enum values should be replaced with EnumSet</div><div><div></div>Code Smell</div></div>
<div><div>String operations should not rely on the default system locale</div><div><div></div>Code Smell</div></div>
<div><div>Comments should not be located at the end of lines of code</div><div><div></div>Code Smell</div></div>
<div><div>Track uses of "CHECKSTYLE:OFF" suppression comments</div></div>