




 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... 🔍

Code Smell

Escaped Unicode characters should not be used

Code Smell

Code Smell

Inner classes should not have too many lines of code

Code Smell

Code Smell

Inner classes which do not reference their owning classes should be "static"

Code Smell

Code Smell

"deleteOnExit" should not be used

Code Smell

Code Smell

Public methods should not contain selector arguments

Code Smell

Code Smell

Java parser failure

Code Smell

Code Smell

Track uses of disallowed methods

Code Smell

Code Smell

Types should be used in lambdas

Code Smell

Code Smell

"java.time" classes should be used for dates and times

Code Smell

Code Smell

The names of methods with boolean return values should start with "is" or "has"

Code Smell

Code Smell

Files should contain only one top-level class or interface each

Code Smell

Code Smell

Classes should not have too many

Code Smell

### "entrySet()" should be iterated when both the key and value are needed

Analyze your code

Code Smell

Major

performance

When only the keys from a map are needed in a loop, iterating the keySet makes sense. But when both the key and the value are needed, it's more efficient to iterate the entrySet, which will give access to both the key and value, instead.

#### Noncompliant Code Example

```
public void doSomethingWithMap(Map<String,Object> map) {
    for (String key : map.keySet()) { // Noncompliant; for ea
        Object value = map.get(key);
        // ...
    }
}
```

#### Compliant Solution

```
public void doSomethingWithMap(Map<String,Object> map) {
    for (Map.Entry<String,Object> entry : map.entrySet()) {
        String key = entry.getKey();
        Object value = entry.getValue();
        // ...
    }
}
```

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-2864

1/2

<div>Classes should not have too many fields</div> <div> Code Smell</div>
<div>The ternary operator should not be used</div> <div> Code Smell</div>
<div>Standard functional interfaces should not be redefined</div> <div> Code Smell</div>
<div>"NullPointerException" should not be caught</div> <div> Code Smell</div>
<div>"NullPointerException" should not be</div>