


-  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

Classes should not access their own subclasses during initialization

Code Smell

"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop

Code Smell

IllegalMonitorStateException should not be caught

Code Smell

JUnit assertions should not be used in "run" methods

Code Smell

Class names should not shadow interfaces or superclasses

Code Smell

"Cloneables" should implement "clone"

Code Smell

Try-with-resources should be used

Code Smell

"readResolve" methods should be 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

Map "computeIfAbsent()" and "computeIfPresent()" should not be used to add "null" values.

Analyze your code

Bug

Critical

Map `computeIfAbsent` and `computeIfPresent` methods are convenient to avoid the cumbersome process to check if a key exists or not, followed by the addition of the entry. However, when the function used to compute the value returns `null`, the entry `key->null` will not be added to the Map. Furthermore, in the case of `computeIfPresent`, if the key is present the entry will be removed. These methods should therefore not be used to conditionally add an entry with a null value. The traditional way should be used instead.

This rule raises an issue when `computeIfAbsent` or `computeIfPresent` is used with a lambda always returning `null`.

Noncompliant Code Example

```
map.computeIfAbsent(key, k -> null); // Noncompliant, the map
map.computeIfPresent(key, (k, oldValue) -> null); // Noncompliant
```

Compliant Solution

```
if (!map.containsKey(key)) {
    map.put(key, null);
}
if (map.containsKey(key)) {
    map.put(key, null);
}
```

See Also

- {rule:java:S3824} - "Map.get" and value test should be replaced with a single method call

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-6104

1/2

<div>used in return types</div> <div> Code Smell</div>
<div>"switch" statements should have "default" clauses</div> <div> Code Smell</div>
<div>Execution of the Garbage Collector should be triggered only by the JVM</div> <div> Code Smell</div>
<div>Constants should not be defined in interfaces</div> <div> Code Smell</div>
<div>String literals should not be duplicated</div>