




 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

Math should not be performed on floats

Analyze your code

BugMinorcert

For small numbers, float math has enough precision to yield the expected value, but for larger numbers, it does not. BigDecimal is the best alternative, but if a primitive is required, use a double.

Noncompliant Code Example

```
float a = 16777216.0f;
float b = 1.0f;
float c = a + b; // Noncompliant; yields 1.6777216E7 not 1.6

double d = a + b; // Noncompliant; addition is still between
```

Compliant Solution

```
float a = 16777216.0f;
float b = 1.0f;
BigDecimal c = BigDecimal.valueOf(a).add(BigDecimal.valueOf(b));

double d = (double)a + (double)b;
```

Exceptions

This rule doesn't raise an issue when the mathematical expression is only used to build a string.

```
System.out.println("[+getName()+"] " +
    "\n\tMax time to retrieve connection:"+(max/1000f
```

See

- [CERT, FLP02-C](#) - Avoid using floating-point numbers when precise computation is needed

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-2164

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>