




 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

Neither "Math.abs" nor negation should be used on numbers that could be "MIN\_VALUE"

Analyze your code

BugMinor?

It is possible for a call to hashCode to return Integer.MIN\_VALUE. Take the absolute value of such a hashcode and you'll still have a negative number. Since your code is likely to assume that it's a positive value instead, your results will be unreliable.

Similarly, Integer.MIN\_VALUE could be returned from Random.nextInt() or any object's compareTo method, and Long.MIN\_VALUE could be returned from Random.nextLong(). Calling Math.abs on values returned from these methods is similarly ill-advised.

Noncompliant Code Example

```
public void doSomething(String str) {
    if (Math.abs(str.hashCode()) > 0) { // Noncompliant
        // ...
    }
}
```

Compliant Solution

```
public void doSomething(String str) {
    if (str.hashCode() != 0) {
        // ...
    }
}
```

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-2676

1/2

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