




 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

### Ints and longs should not be shifted by zero or more than their number of bits-1

Analyze your code

BugMinor?

Since an `int` is a 32-bit variable, shifting by more than `+/-31` is confusing at best and an error at worst. When the runtime shifts 32-bit integers, it uses the lowest 5 bits of the shift count operand. In other words, shifting an `int` by 32 is the same as shifting it by 0, and shifting it by 33 is the same as shifting it by 1.

Similarly, when shifting 64-bit integers, the runtime uses the lowest 6 bits of the shift count operand and shifting `long` by 64 is the same as shifting it by 0, and shifting it by 65 is the same as shifting it by 1.

#### Noncompliant Code Example

```
public int shift(int a) {
    int x = a >> 32; // Noncompliant
    return a << 48;  // Noncompliant
}
```

#### Compliant Solution

```
public int shift(int a) {
    int x = a >> 31;
    return a << 16;
}
```

#### Exceptions

This rule doesn't raise an issue when the shift by zero is obviously for cosmetic reasons:

- When the value shifted is a literal.
- When there is a similar shift at the same position on line before or after. E.g.:

```
bytes[loc+0] = (byte)(value >> 8);
bytes[loc+1] = (byte)(value >> 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-2183

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>