




 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

### Pattern Matching for "instanceof" operator should be used instead of simple "instanceof" + cast

Analyze your code

Code Smell

Minor

java16

In Java 16, the feature "Pattern matching for instanceof" is finalized and can be used in production. Previously developers needed to do 3 operations in order to do this: check the variable type, cast it and assign the casted value to the new variable. This approach is quite verbose and can be replaced with pattern matching for instanceof, doing these 3 actions (check, cast and assign) in one expression.

This rule raises an issue when an instanceof check followed by a cast and an assignment could be replaced by pattern matching.

#### Noncompliant Code Example

```
int f(Object o) {
    if (o instanceof String) { // Noncompliant
        String string = (String) o;
        return string.length();
    }
    return 0;
}
```

#### Compliant Solution

```
int f(Object o) {
    if (o instanceof String string) { // Compliant
        return string.length();
    }
    return 0;
}
```

See

- [JEP 394: Pattern Matching for instanceof](#)

Available In:





sonarlint

sonarcloud

sonarqube

https://rules.sonarsource.com/java/RSPEC-6201

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>