




 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

Local-Variable Type Inference should be used

Analyze your code

Code Smell

Info ?

java10

In Java 10 **Local-Variable Type Inference** was introduced. It allows you to omit the expected type of a variable by declaring it with the `var` keyword.

While it is not always possible or cleaner to use this new way of declaring a variable, when the type on the left is the same as the one on the right in an assignment, using the `var` will result in a more concise code.

This rule reports an issue when the expected type of the variable is the same as the returned type of assigned expression and the type can be easily inferred by the reader, either when the type is already mentioned in the name or the initializer, or when the expression is self-explained.

Noncompliant Code Example

```
MyClass myClass = new MyClass();

int i = 10; // Type is self explained

MyClass something = MyClass.getMyClass(); // Type is already
MyClass myClass = get(); // Type is already mentionned in th
```

Compliant Solution

```
var myClass = new MyClass();

var i = 10;

var something = MyClass.getMyClass();

var myClass = get();
```

See

- [JEP 286: Local-Variable Type Inference](#)

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-6212

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>