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 rules 632 | Vulnerability 53 | Bug 154 | Security Hotspot 36 | Code Smell 389 | Quick Fix 42

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

Underscores should be used to make large numbers readable

Analyze your code

Code Smell | Minor | convention

Beginning with Java 7, it is possible to add underscores ('_') to numeric literals to enhance readability. The addition of underscores in this manner has no semantic meaning, but makes it easier for maintainers to understand the code.

The number of digits to the left of a decimal point needed to trigger this rule varies by base.

Base	Minimum digits
binary	9
octal	9
decimal	6
hexadecimal	9

It is only the presence of underscores, not their spacing that is scrutinized by this rule.

Note that this rule is automatically disabled when the project's `sonar.java.source` is lower than 7.

Noncompliant Code Example

```
int i = 10000000;  // Noncompliant; is this 10 million or 10
int  j = 0b01101001010011011110010101011110;  // Noncomplian
long l = 0x7fffffffffffffffL;  // Noncompliant
```

Compliant Solution

```
int i = 10_000_000;
int  j = 0b01101001_01001101_11100101_01011110;
long l = 0x7fff_ffff_ffff_ffffL;
```

Available In:

sonarlint | sonarcloud | sonarqube

Local-Variable Type Inference should be used

Code Smell

Migrate your tests from JUnit4 to the new JUnit5 annotations

Code Smell

Track uses of disallowed classes

Code Smell

Track uses of "@SuppressWarnings" annotations

Code Smell