

 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

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

Lambdas should be replaced with method references

Analyze your code

- Code Smell
- Minor
- Quick Fix
- java8

Method/constructor references are commonly agreed to be, most of the time, more compact and readable than using lambdas, and are therefore preferred.

In some rare cases, when it is not clear from the context what kind of function is being described and reference would not increase the clarity, it might be fine to keep the lambda.

Similarly, null checks can be replaced with references to the `Objects::isNull` and `Objects::nonNull` methods, casts can be replaced with `SomeClass.class::cast` and `instanceof` can be replaced with `SomeClass.class::isInstance`.

Note that this rule is automatically disabled when the project's `sonar.java.source` is lower than 8.

Noncompliant Code Example

```
class A {
    void process(List<A> list) {
        list.stream()
            .filter(a -> a instanceof B)
            .map(a -> (B) a)
            .map(b -> b.<String>getObject())
            .forEach(b -> { System.out.println(b); });
    }
}


class B extends A {
    <T> T getObject() {
        return null;
    }
}
```

Compliant Solution


```
class A {
    void process(List<A> list) {
        list.stream()
            .filter(B.class::isInstance)
            .map(B.class::cast)
            .map(B::<String>getObject)
            .forEach(System.out::println);
    }
}

class B extends A {
    <T> T getObject() {
        return null;
    }
}
```


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

Available In:

 |  | 

© 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](#)