




 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... 🔍

"switch" statements should have at least 3 "case" clauses

Code Smell

A "while" loop should be used instead of a "for" loop

Code Smell

The default unnamed package should not be used

Code Smell

"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method

Code Smell

Package names should comply with a naming convention

Code Smell

Nested code blocks should not be used

Code Smell

Array designators "[]" should be on the type, not the variable

Code Smell

Array designators "[]" should be located after the type in method signatures

Code Smell

Type parameter names should comply with a naming convention

Code Smell

Overriding methods should do more than simply call the same method in the super class

Code Smell

Classes that override "clone" should be "Cloneable" and call "super.clone()"

Code Smell

Operator "instanceof" should be used instead of "A.class.isInstance()"

Analyze your code

Code SmellMajor?

The instanceof construction is a preferred way to check whether a variable can be cast to some type statically because a compile-time error will occur in case of incompatible types. The method `isInstance()` from `java.lang.Class` works differently and does type check at runtime only, incompatible types will therefore not be detected early in the development, potentially resulting in dead code. The `isInstance()` method should only be used in dynamic cases when the `instanceof` operator can't be used.

This rule raises an issue when `isInstance()` is used and could be replaced with an `instanceof` check.

Noncompliant Code Example

```
int f(Object o) {
    if (String.class.isInstance(o)) { // Noncompliant
        return 42;
    }
    return 0;
}

int f(Number n) {
    if (String.class.isInstance(n)) { // Noncompliant
        return 42;
    }
    return 0;
}
```

Compliant Solution

```
int f(Object o) {
    if (o instanceof String) { // Compliant
        return 42;
    }
    return 0;
}

int f(Number n) {
    if (n instanceof String) { // Compile-time error
        return 42;
    }
    return 0;
}

boolean fun(Object o, String c) throws ClassNotFoundException {
    return Class.forName(c).isInstance(o); // Compliant, can't
}
```

https://rules.sonarsource.com/java/RSPEC-6202

1/2

Public constants and fields initialized at declaration should be "static final" rather than merely "final"

 Code Smell

Local variable and method parameter names should comply with a naming convention

 Code Smell

Exception classes should be immutable

 Code Smell

Field names should comply with a

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](#)