




 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

Boxing and unboxing should not be immediately reversed

Analyze your code

Bug

Minor

Quick Fix

clumsy

Boxing is the process of putting a primitive value into an analogous object, such as creating an Integer to hold an int value. Unboxing is the process of retrieving the primitive value from such an object.

Since the original value is unchanged during boxing and unboxing, there's no point in doing either when not needed. This also applies to autoboxing and auto-unboxing (when Java implicitly handles the primitive/object transition for you).

Noncompliant Code Example

```
public void examineInt(int a) {
    //...
}

public void examineInteger(Integer a) {
    // ...
}

public void func() {
    int i = 0;
    Integer iger1 = Integer.valueOf(0);
    double d = 1.0;

    int dIntValue = new Double(d).intValue(); // Noncompliant

    examineInt(new Integer(i).intValue()); // Noncompliant; ex
    examineInt(Integer.valueOf(i)); // Noncompliant; boxed in

    examineInteger(i); // Compliant; value is boxed but not th
    examineInteger(iger1.intValue()); // Noncompliant; unboxed

    Integer iger2 = new Integer(iger1); // Noncompliant; unnec
}
```

Compliant Solution

```
public void examineInt(int a) {
    //...
}

public void examineInteger(Integer a) {
    // ...
}

public void func() {
    int i = 0;
    Integer iger1 = Integer.valueOf(0);
    double d = 1.0;
```

https://rules.sonarsource.com/java/RSPEC-2153

1/2

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

```
int dIntValue = (int) d;

examineInt(i);

examineInteger(i);
examineInteger(iger1);
}
```

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](#)