 sonar

RULES

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

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

"write(byte[],int,int)" should be overridden

Analyze your code

Code Smell

Minor ?

performance

When directly subclassing `java.io.OutputStream` or `java.io.FilterOutputStream`, the only requirement is that you implement the method `write(int)`. However most uses for such streams don't write a single byte at a time and the default implementation for `write(byte[],int,int)` will call `write(int)` for every single byte in the array which can create a lot of overhead and is utterly inefficient. It is therefore strongly recommended that subclasses provide an efficient implementation of `write(byte[],int,int)`.

This rule raises an issue when a direct subclass of `java.io.OutputStream` or `java.io.FilterOutputStream` doesn't provide an override of `write(byte[],int,int)`.

Noncompliant Code Example

```
public class MyStream extends OutputStream { // Noncompliant
    private FileOutputStream fout;

    public MyStream(File file) throws IOException {
        fout = new FileOutputStream(file);
    }

    @Override
    public void write(int b) throws IOException {
        fout.write(b);
    }

    @Override
    public void close() throws IOException {
        fout.write("\n\n".getBytes());
        fout.close();
        super.close();
    }
}
```

Compliant Solution

```
public class MyStream extends OutputStream {
    private FileOutputStream fout;





    public MyStream(File file) throws IOException {
        fout = new FileOutputStream(file);
    }

    @Override
    public void write(int b) throws IOException {
        fout.write(b);
    }

    @Override
```

https://rules.sonarsource.com/java/RSPEC-4349

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

```
public void write(byte[] b, int off, int len) throws IOE
    fout.write(b, off, len);
}

@Override
public void close() throws IOException {
    fout.write("\n\n".getBytes());
    fout.close();
    super.close();
}
}
```

Exceptions

This rule doesn't raise an issue when the class is declared abstract.

Available In:

sonarlint  | **sonarcloud**  | **sonarqube** 