




 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... 🔍


constructors

 Code Smell


Local variables should not shadow class fields

 Code Smell


Redundant pairs of parentheses should be removed

 Code Smell


Inheritance tree of classes should not be too deep

 Code Smell


Nested blocks of code should not be left empty

 Code Smell


Methods should not have too many parameters

 Code Smell


Unused "private" fields should be removed

 Code Smell


Collapsible "if" statements should be merged

 Code Smell


Unused labels should be removed

 Code Smell

Standard outputs should not be used directly to log anything

 Code Smell



OS commands should not be vulnerable to argument injection attacks

 Vulnerability

"ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization

"read" and "readLine" return values should be used

Analyze your code

 Bug  Major ?

When a method is called that returns data read from some data source, that data should be stored rather than thrown away. Any other course of action is surely a bug.

This rule raises an issue when the return value of any of the following is ignored or merely null-checked: `BufferedReader.readLine()`, `Reader.read()`, and these methods in any child classes.

Noncompliant Code Example

```
public void doSomethingWithFile(String fileName) {
    BufferedReader buffReader = null;
    try {
        buffReader = new BufferedReader(new FileReader(fileName))
        while (buffReader.readLine() != null) { // Noncompliant
            // ...
        }
    } catch (IOException e) {
        // ...
    }
}
```

Compliant Solution

```
public void doSomethingWithFile(String fileName) {
    BufferedReader buffReader = null;
    try {
        buffReader = new BufferedReader(new FileReader(fileName))
        String line = null;
        while ((line = buffReader.readLine()) != null) {
            // ...
        }
    } catch (IOException e) {
        // ...
    }
}
```

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](#)

https://rules.sonarsource.com/java/RSPEC-2677

1/2

 Vulnerability
Logging should not be vulnerable to injection attacks
 Vulnerability
Exceptions should not be thrown from servlet methods
 Vulnerability
Return values should not be ignored when they contain the operation status code
 Bug
Repeated patterns in regular expressions should not match the