




 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... 🔍

Objects should not be created only to "getClass"

Code Smell

Primitives should not be boxed just for "String" conversion

Code Smell

Constructors should not be used to instantiate "String", "BigInteger", "BigDecimal" and primitive-wrapper classes

Code Smell

"URL.hashCode" and "URL.equals" should be avoided

Code Smell

Two branches in a conditional structure should not have exactly the same implementation

Code Smell

Unused assignments should be removed

Code Smell

"Object.wait(...)" should never be called on objects that implement "java.util.concurrent.locks.Condition"

Code Smell

A field should not duplicate the name of its containing class

Code Smell

JUnit4 @Ignored and JUnit5 @Disabled annotations should be used to disable tests and should provide a rationale

Code Smell

Anonymous inner classes containing only one method should become lambdas

Code Smell

Intermediate Stream methods should not be left unused

Analyze your code

BugMajor?java8

There are two types of stream operations: intermediate operations, which return another stream, and terminal operations, which return something other than a stream. Intermediate operations are lazy, meaning they aren't actually executed until and unless a terminal stream operation is performed on their results. Consequently if the result of an intermediate stream operation is not fed to a terminal operation, it serves no purpose, which is almost certainly an error.

Noncompliant Code Example

```
widgets.stream().filter(b -> b.getColor() == RED); // Noncompliant
```

Compliant Solution

```
int sum = widgets.stream()
    .filter(b -> b.getColor() == RED)
    .mapToInt(b -> b.getWeight())
    .sum();
Stream<Widget> pipeline = widgets.stream()
    .filter(b -> b.getColor() == RED)
    .mapToInt(b -> b.getWeight())
    .sum();
sum = pipeline.sum();
```

See

- [Stream Operations](#)

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-3958

1/2

<div><div>"switch" statements should not have too many "case" clauses</div><div> Code Smell</div></div>
<div><div>"for" loop stop conditions should be invariant</div><div> Code Smell</div></div>
<div><div>Sections of code should not be commented out</div><div> Code Smell</div></div>
<div><div>Non-constructor methods should not have the same name as the enclosing class</div><div> Code Smell</div></div>