




 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... 🔍

Custom getter method should not be used to override record's getter behavior

Code Smell

Tests should use fixed data instead of randomized data

Code Smell

Spring's ModelAndViewAssert assertions should be used instead of other assertions

Code Smell

Lambdas should not have too many lines

Code Smell

"@EnableAutoConfiguration" should be fine-tuned

Code Smell

Enum values should be compared with "=="

Code Smell

Spring components should use constructor injection

Code Smell

Regex patterns should not be created needlessly

Code Smell

Track uses of disallowed constructors

Code Smell

Java 8's "Files.exists" should not be used

Code Smell

"Optional" should not be used for parameters

Code Smell

"Arrays.stream" should be used for primitive arrays

Analyze your code

Code SmellMajor?performance

For arrays of objects, `Arrays.asList(T ... a).stream()` and `Arrays.stream(array)` are basically equivalent in terms of performance. However, for arrays of primitives, using `Arrays.asList` will force the construction of a list of boxed types, and then use *that* list as a stream. On the other hand, `Arrays.stream` uses the appropriate primitive stream type (`IntStream`, `LongStream`, `DoubleStream`) when applicable, with much better performance.

Noncompliant Code Example

```
Arrays.asList("a1", "a2", "b1", "c2", "c1").stream()
    .filter(...)
    .forEach(...);

Arrays.asList(1, 2, 3, 4).stream() // Noncompliant
    .filter(...)
    .forEach(...);
```

Compliant Solution

```
Arrays.asList("a1", "a2", "b1", "c2", "c1").stream()
    .filter(...)
    .forEach(...);

int[] intArray = new int[]{1, 2, 3, 4};
Arrays.stream(intArray)
    .filter(...)
    .forEach(...);
```





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-3631

1/2

<div>Tests should be kept in a dedicated source directory</div> <div> Code Smell</div>
<div>"this" should not be exposed from constructors</div> <div> Code Smell</div>
<div>Classes should not have too many "static" imports</div> <div> Code Smell</div>
<div>Escaped Unicode characters should not be used</div> <div> Code Smell</div>