
































-  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 rules 632
-  Vulnerability 53
-  Bug 154
-  Security Hotspot 36
-  Code Smell 389
-  Quick Fix 42

Tags ▾

Search by name... 

 Code Smell
Mockito argument matchers should be used on all parameters
 Bug
Spring "@Controller" classes should not use "@Scope"
 Bug
Constructor injection should be used instead of field injection
 Bug
Classes that don't define "hashCode()" should not be used in hashes
 Bug
Floating point numbers should not be tested for equality
 Bug
Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression
 Code Smell
Limited dependence should be placed on operator precedence
 Code Smell
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

"Stream.peek" should be used with caution

Analyze your code

-  Code Smell
-  Major 
-  java8 pitfall

According to its JavaDocs, the intermediate Stream operation `java.util.Stream.peek()` "exists mainly to support debugging" purposes.

A key difference with other intermediate Stream operations is that the Stream implementation is free to skip calls to `peek()` for optimization purpose. This can lead to `peek()` being unexpectedly called only for some or none of the elements in the Stream.

As a consequence, relying on `peek()` without careful consideration can lead to error-prone code.

This rule raises an issue for each use of `peek()` to be sure that it is challenged and validated by the team to be meant for production debugging/logging purposes.

Noncompliant Code Example

```
Stream.of("one", "two", "three", "four")
    .filter(e -> e.length() > 3)
    .peek(e -> System.out.println("Filtered value: " +
```

Compliant Solution





```
Stream.of("one", "two", "three", "four")
    .filter(e -> e.length() > 3)
    .foreach(e -> System.out.println("Filtered value: "
```

See

- [Java 8 API Documentation](#)
- 4comprehension: [Idiomatic Peeking with Java Stream API](#)
- Data Geekery: [10 Subtle Mistakes When Using the Streams API](#)

Available In:

sonarlint  | sonarcloud  | sonarqube 

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