sonar RULES

Products

- 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

Unit tests should throw exceptions
Code Smell

Test methods should comply with a naming convention
Code Smell

Value-based objects should not be serialized
Code Smell

Default annotation parameter values should not be passed as arguments
Code Smell

Method parameters should be declared with base types
Code Smell

Fields should not be initialized to default values
Code Smell

Multiple loops over the same set should be combined
Code Smell

Classes without "public" constructors should be "final"
Code Smell

Unnecessary semicolons should be omitted
Code Smell

Literal boolean values and nulls should not be used in assertions
Code Smell

Test assertions should include messages
Code Smell

Mutable members should not be

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

Analyze your code

Code Smell Major suspicious

From the Java API documentation:

> `Condition` factors out the `Object` monitor methods (`wait`, `notify` and `notifyAll`) into distinct objects to give the effect of having multiple wait-sets per object, by combining them with the use of arbitrary Lock implementations. Where a `Lock` replaces the use of `synchronized` methods and statements, a `Condition` replaces the use of the `Object` monitor methods.

The purpose of implementing the `Condition` interface is to gain access to its more nuanced `await` methods. Therefore, calling the method `Object.wait(...)` on a class implementing the `Condition` interface is silly and confusing.

Noncompliant Code Example

```
final Lock lock = new ReentrantLock();
final Condition notFull  = lock.newCondition();
...
notFull.wait();
```

Compliant Solution

```
final Lock lock = new ReentrantLock();
final Condition notFull  = lock.newCondition();
...
notFull.await();
```

Available In:

sonarlint sonarcloud sonarqube

stored or returned directly

Code Smell

Redundant modifiers should not be used

Code Smell

"private" and "final" methods that don't access instance data should be "static"

Code Smell

Files should not be empty

Code Smell

Collection methods with O(n)