




 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... 


Values should not be uselessly incremented




Silly String operations should not be made




Non-serializable classes should not be written




"hashCode" and "toString" should not be called on array instances




Collections should not be passed as arguments to their own methods




"BigDecimal(double)" should not be used




Invalid "Date" values should not be used




Reflection should not be used to check non-runtime annotations




Custom serialization method signatures should meet requirements



"Externalizable" classes should have no-arguments constructors




Classes should not be compared by name





Related "if/else if" statements should


### "Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop

Analyze your code

 Code Smell

 Critical



 multi-threading cert

According to the documentation of the Java Condition interface:

When waiting upon a Condition, a "spurious wakeup" is permitted to occur, in general, as a concession to the underlying platform semantics. This has little practical impact on most application programs as a Condition should always be waited upon in a loop, testing the state predicate that is being waited for. An implementation is free to remove the possibility of spurious wakeups but it is recommended that applications programmers always assume that they can occur and so always wait in a loop.

The same advice is also found for the `Object.wait(...)` method:

waits should always occur in loops, like this one:

```
synchronized (obj) {
    while (<condition does not hold>){
        obj.wait(timeout);
    }
    ... // Perform action appropriate to condition
}
```

#### Noncompliant Code Example

```
synchronized (obj) {
    if (!suitableCondition()){
        obj.wait(timeout); //the thread can wake up even if th
    }
    ... // Perform action appropriate to condition
}
```




#### Compliant Solution

```
synchronized (obj) {
    while (!suitableCondition()){
        obj.wait(timeout);
    }
    ... // Perform action appropriate to condition
}
```

See





- [CERT TH103-J](#) - Always invoke wait() and await() methods inside a loop

Available In:

 |  | 

https://rules.sonarsource.com/java/RSPEC-2274

1/2

Java static code analysis: "Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop	
related if/else if statements should not have the same condition	
 Bug	
Synchronization should not be done on instances of value-based classes	
 Bug	
"Iterator.hasNext()" should not call "Iterator.next()"	
 Bug	
Identical expressions should not be used on both sides of a binary operator	
 Bug	

© 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](#)