




 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... 🔍

Track uses of "FIXME" tags

Code Smell

Deprecated elements should have both the annotation and the Javadoc tag

Code Smell

Assignments should not be made from within sub-expressions

Code Smell

Generic exceptions should never be thrown

Code Smell

Labels should not be used

Code Smell

Utility classes should not have public constructors

Code Smell

Local variables should not shadow class fields

Code Smell

Redundant pairs of parentheses should be removed

Code Smell

Inheritance tree of classes should not be too deep

Code Smell

Nested blocks of code should not be left empty

Code Smell

Methods should not have too many parameters

Code Smell

Unused "private" fields should be removed

Code Smell

### Getters and setters should be synchronized in pairs

Analyze your code

Bug

Major

multi-threading cert

When one part of a getter/setter pair is synchronized the other part should be too. Failure to synchronize both sides of a pair may result in inconsistent behavior at runtime as callers access an inconsistent method state.

This rule raises an issue when either the method or the contents of one method in a getter/setter pair are synchnoized but the other is not.

#### Noncompliant Code Example

```
public class Person {
    String name;
    int age;

    public synchronized void setName(String name) {
        this.name = name;
    }

    public String getName() { // Noncompliant
        return this.name;
    }

    public void setAge(int age) { // Noncompliant
        this.age = age;
    }

    public int getAge() {
        synchronized (this) {
            return this.age;
        }
    }
}
```

#### Compliant Solution

```
public class Person {
    String name;
    int age;






    public synchronized void setName(String name) {
        this.name = name;
    }

    public synchronized String getName() {
        return this.name;
    }

    public void setAge(int age) {
        synchronized (this) {
            this.age = age;
        }
    }
}
```

https://rules.sonarsource.com/java/RSPEC-2886

1/2


<b>Collapsible "if" statements should be merged</b>  Code Smell
<b>Unused labels should be removed</b>  Code Smell
<b>Standard outputs should not be used directly to log anything</b>  Code Smell
<b>OS commands should not be vulnerable to argument injection attacks</b>  Vulnerability

```
    }  
  }  
  
  public int getAge() {  
    synchronized (this) {  
      return this.age;  
    }  
  }  
}
```

- See
- [CERT, VNA01-J](#) - Ensure visibility of shared references to immutable objects

Available In:

 |  | 

© 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](#)