
































-  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... 🔍

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) performance should be used carefully
 Code Smell
"Exception" should not be caught when not required by called methods
 Code Smell
"collect" should be used with "Streams" instead of "list::add"
 Code Smell
Switches should be used for sequences of simple "String" tests
 Code Smell
"final" classes should not have "protected" members
 Code Smell
Underscores should be used to make large numbers readable
 Code Smell
"Serializable" inner classes of "Serializable" classes should be static
 Code Smell
Member variable visibility should be specified
 Code Smell

Non-constructor methods should not have the same name as the enclosing class

Analyze your code

 Code Smell  Major  pitfall

Having a class and some of its methods sharing the same name is misleading, and leaves others to wonder whether it was done that way on purpose, or was the methods supposed to be a constructor.

Noncompliant Code Example





```
public class Foo {
    public Foo() {...}
    public void Foo(String label) {...} // Noncompliant
}
```

Compliant Solution

```
public class Foo {
    public Foo() {...}
    public void foo(String label) {...} // Compliant
}
```

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](#)

<div>Classes and methods that rely on the default system encoding should not be used</div> <div> Code Smell</div>
<div>Simple class names should be used</div> <div> Code Smell</div>
<div>Variables should not be declared before they are relevant</div> <div> Code Smell</div>
<div>Extensions and implementations should not be redundant</div> <div> Code Smell</div>
<div>"==" and "!=" should not be used when</div>