

-  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... 🔍

Abstract class names should comply with a naming convention

 Code Smell

Strings literals should be placed on the left side when checking for equality

 Code Smell

Files should contain an empty newline at the end

 Code Smell

Source code should be indented consistently

 Code Smell

A close curly brace should be located at the beginning of a line

 Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines

 Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line

 Code Smell

An open curly brace should be located at the beginning of a line

 Code Smell

An open curly brace should be located at the end of a line

 Code Smell

Tabulation characters should not be used

 Code Smell

Functions should not be defined with a variable number of arguments

 Code Smell

Classes should not be coupled to too many other classes (Single Responsibility Principle)

Analyze your code

 Code Smell  Major  brain-overload

According to the Single Responsibility Principle, introduced by Robert C. Martin in his book "Principles of Object Oriented Design", a class should have only one responsibility:

- If a class has more than one responsibility, then the responsibilities become coupled.
- Changes to one responsibility may impair or inhibit the class' ability to meet the others.
- This kind of coupling leads to fragile designs that break in unexpected ways when changed.

Classes which rely on many other classes tend to aggregate too many responsibilities and should be split into several smaller ones.

Nested classes dependencies are not counted as dependencies of the outer class.

Noncompliant Code Example





With a threshold of 5:

```
class Foo {                                // Noncompliant - Foo dep
    T1 a1;                                  // Foo is coupled to T1
    T2 a2;                                  // Foo is coupled to T2
    T3 a3;                                  // Foo is coupled to T3

    public T4 compute(T5 a, T6 b) { // Foo is coupled to T4,
        T7 result = a.getResult(b); // Foo is coupled to T7
        return result;
    }

    public static class Bar {              // Compliant - Bar depend
        T8 a8;
        T9 a9;
    }
}
```

Available In:
 |  | 

<div>Local-Variable Type Inference should be used</div> <div> Code Smell</div>
<div>Migrate your tests from JUnit4 to the new JUnit5 annotations</div> <div> Code Smell</div>
<div>Track uses of disallowed classes</div> <div> Code Smell</div>
<div>Track uses of "@SuppressWarnings" annotations</div> <div> Code Smell</div>