




 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... 🔍

 Code Smell


Field names should comply with a naming convention




Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls




Case insensitive string comparisons should be made without intermediate upper or lower casing




Collection.isEmpty() should be used to test for emptiness




String.valueOf() should not be appended to a String




Interface names should comply with a naming convention




"throws" declarations should not be superfluous




Unnecessary imports should be removed



Return of boolean expressions should not be wrapped into an "if-then-else" statement





Boolean literals should not be redundant




Tests should be stable

Analyze your code

 Code Smell

 Major



tests design unpredictable

Unstable / flaky tests are tests which sometimes pass and sometimes fail, without any code change. Obviously, they slow down developments when engineers have to rerun failed tests. However the real problem is that you can't completely trust these tests, they might fail for many different reasons and you don't know if any of them will happen in production.

Some tools, such as TestNG, enable developers to automatically retry flaky tests. This might be ok as a temporary solution, but it should definitely be fixed. The more flaky tests you add, the more chances there are for a bug to arrive in production.

This rule raises an issue when the annotation `org.testng.annotations.Test` is given a `successPercentage` argument with a value lower than 100.

Noncompliant Code Example




```
import org.testng.annotations.Test;

public class PercentageTest {
    @Test(successPercentage = 80, invocationCount = 10) //
    public void flakyTest() {
    }
}
```

See

- [TestNG documentation - Annotations](#)
- [Test Flakiness - Methods for identifying and dealing with flaky tests](#)





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](#)

https://rules.sonarsource.com/java/RSPEC-5973

1/2

<div>Modifiers should be declared in the correct order</div> <div> Code Smell</div>
<div>Empty statements should be removed</div> <div> Code Smell</div>
<div>Class variable fields should not have public accessibility</div> <div> Code Smell</div>
<div>URLs should not be hardcoded</div> <div> Code Smell</div>
<div>Class names should comply with a naming convention</div>