




 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... 🔍

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

Tests should use fixed data instead of randomized data

Analyze your code

Code Smell

Major

tests design confusing

Tests should always:

- Make sure that production code behaves as expected, including edge cases.
- Be easy to debug, i.e. understandable and reproducible.

Using random values in tests will not necessarily check edge cases, and it will make test logs a lot harder to read. It is better to use easily readable hardcoded values. If this makes your code bigger you can use helper functions.

There is one valid use case for random data in tests: when testing every value would make tests impractically slow. In this case the best you can do is use random to test every value on the long run. You should however make sure that random values are logged so that you can reproduce failures. Some libraries exist to make all this easier. You can for example use property-based testing libraries such as [jqwik](#).

This rule raises an issue when new Random() or UUID.randomUUID() are called in test code.

Noncompliant Code Example

```
int userAge = new Random().nextInt(42); // Noncompliant
UUID userID = UUID.randomUUID(); // Noncompliant
```

Compliant Solution

```
int userAge = 31;
UUID userID = UUID.fromString("00000000-000-0000-0000-00000000")
```

See

- [Modern Best Practices for Testing in Java - Philipp Hauer](#)
- [Jqwik test engine](#)

Available In:

sonarlint

sonarcloud

sonarqube

© 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-5977

1/2

| |
|---|
| <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> |