

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... 🔍

Basic authentication should not be used

Vulnerability

Regular expressions should not be vulnerable to Denial of Service attacks

Vulnerability

"HttpServletRequest.getRequestSession" should not be used

Vulnerability

Hashes should include an unpredictable salt

Vulnerability

Calls to methods should not trigger an IllegalArgumentException

Bug

Unsupported methods should not be called on some collection implementations

Bug

Cast operations should not trigger a ClassCastException

Bug

Members ignored during record serialization should not be used

Bug

Map "computeIfAbsent()" and "computeIfPresent()" should not be used to add "null" values.

Bug

Regex lookahead assertions should not be contradictory

Bug

Back references in regular expressions should only refer to capturing groups that are matched before the reference

Bug

Regex boundaries should not be used in a way that can never be matched

Bug

Tests should include assertions

Analyze your code

Code SmellBlocker ?junit tests

A test case without assertions ensures only that no exceptions are thrown. Beyond basic runnability, it ensures nothing about the behavior of the code under test.

This rule raises an exception when no assertions from any of the following known frameworks are found in a test:

- AssertJ
- Awaitility
- EasyMock
- Eclipse Vert.x
- Fest 1.x and 2.x
- Hamcrest
- JMock
- JMockit
- JUnit
- Mockito
- Rest-assured 2.x, 3.x and 4.x
- RxJava 1.x and 2.x
- Selenide
- Spring's org.springframework.test.web.servlet.ResultActions.andExpect() and org.springframework.test.web.servlet.ResultActions.andExpectAll()
- Truth Framework
- WireMock

Furthermore, as new or custom assertion frameworks may be used, the rule can be parametrized to define specific methods that will also be considered as assertions. No issue will be raised when such methods are found in test cases. The parameter value should have the following format <FullyQualifiedClassName>#<MethodName>, where MethodName can end with the wildcard character. For constructors, the pattern should be <FullyQualifiedClassName>#<init>.

Example: com.company.CompareToTester#compare*, com.company.CustomAssert#custom2<init>.

Noncompliant Code Example

```
@Test
public void testDoSomething() { // Noncompliant
    MyClass myClass = new MyClass();
    myClass.doSomething();
}
```

Compliant Solution

Example when com.company.CompareToTester#compare* is used as parameter to the rule.




```
import com.company.CompareToTester;

@Test
public void testDoSomething() {
    MyClass myClass = new MyClass();
    assertNull(myClass.doSomething()); // JUnit assertion
    assertThat(myClass.doSomething()).isNull(); // Fest assertion
}

@Test
public void testDoSomethingElse() {
```

https://rules.sonarsource.com/java/RSPEC-2699

1/2

<div>Regex patterns following a possessive quantifier should not always fail</div> <div> Bug</div>
<div>Regular expressions should be syntactically valid</div> <div> Bug</div>
<div>Assertions comparing incompatible types should not be made</div> <div> Bug</div>

```
MyClass myClass = new MyClass();
new CompareToTester().compareTo(myClass); // Compliant - custom a
CompareToTester.compareToStatic(myClass); // Compliant
}
```

Available In:

sonarlint  | **sonarcloud**  | **sonarqube** 