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

Exception testing via JUnit @Test annotation should be avoided

Analyze your code

Code Smell | Minor | junit tests

When testing exception via `@Test` annotation, having additional assertions inside that test method can be problematic because any code after the raised exception will not be executed. It will prevent you to test the state of the program after the raised exception and, at worst, make you misleadingly think that it is executed.

You should consider moving any assertions into a separate test method where possible, or using org.junit.Assert.assertThrows instead.

Alternatively, you could use try-catch idiom for JUnit version < 4.13 or if your project does not support lambdas.

Noncompliant Code Example

```
@Test(expected = IndexOutOfBoundsException.class)
public void testShouldFail() {
  get();
  // This test pass since execution will never get past this
  Assert.assertEquals(0, 1);
}

private Object get() {
  throw new IndexOutOfBoundsException();
}
```

Compliant Solution

- For JUnit >= 4.13, use org.junit.Assert.assertThrows:

```
// This test correctly fails.
@Test
public void testToString() {
    Object obj = get();
    Assert.assertThrows(IndexOutOfBoundsException.class, ()
    Assert.assertEquals(0, 1);
}
```

- For JUnit < 4.13, use the try-catch idiom:

```
@Test
public void testShouldFail() {
    Object obj = get();
    try {
        obj.toString();
        Assert.fail("Expected an IndexOutOfBoundsException t
    } catch (IndexOutOfBoundsException e) {}
    Assert.assertEquals(0, 1); // Correctly fails.
}
```

Local-Variable Type Inference should be used

Code Smell

Migrate your tests from JUnit4 to the new JUnit5 annotations

Code Smell

Track uses of disallowed classes

Code Smell

Track uses of "@SuppressWarnings" annotations

Code Smell

See

- JUnit exception testing documentation

Available In:

sonarlint | sonarcloud | sonarqube