sonar

RULES

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

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

Class names should comply with a naming convention

Code Smell

Code Smell

Method names should comply with a naming convention

Code Smell

Code Smell

Comma-separated labels should be used in Switch with colon case

Code Smell

Code Smell

JUnit5 test classes and methods should have default package visibility

Code Smell

Code Smell

Track uses of "TODO" tags

Code Smell

Code Smell

Deprecated code should be removed

Code Smell

Bug

Annotated Mockito objects should be initialized

Bug

Bug

Custom resources should be closed

Bug

Code Smell

Threads should not be started in constructors

Code Smell

Code Smell

"main" should not "throw" anything

Code Smell

Code Smell

Track lack of copyright and license headers

Code Smell

Code Smell

Octal values should not be used

Code Smell

Exception testing via JUnit ExpectedException rule should not be mixed with other assertions

Analyze your code

Code Smell

Major

junit tests

When testing exception via `org.junit.rules.ExpectedException` any code after the raised exception will not be executed, so adding subsequent assertions is wrong and misleading. This rule raises an issue when an assertion is done after the "expect(...)" invocation, only the code throwing the expected exception should be after "expect(...)".

You should consider using `org.junit.Assert.assertThrows` instead, it's available since JUnit 4.13 and it allows additional subsequent assertions.

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

Noncompliant Code Example

```
@Rule
public ExpectedException thrown = ExpectedException.none();

@Test
public void test() throws IndexOutOfBoundsException {
    thrown.expect(IndexOutOfBoundsException.class); // Noncompliant
    Object o = 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`:

```
Assert.assertThrows(IndexOutOfBoundsException.class, () -> g
// This test correctly fails.
Assert.assertEquals(0, 1);
```

- For JUnit < 4.13, use the [try-catch idiom](#):


```
try {
    get();
    Assert.fail("Expected an IndexOutOfBoundsException to be t
} catch (IndexOutOfBoundsException e) {}
Assert.assertEquals(0, 1); // Correctly fails.
```

See


- [JUnit exception testing documentation](#)

https://rules.sonarsource.com/java/RSPEC-5776


1/2

 Code Smell


Exit methods should not be called

 Code Smell


HTTP response headers should not be vulnerable to injection attacks

 Vulnerability




Members of Spring components should be injected

 Vulnerability

Classes should not be loaded dynamically

 Vulnerability

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](#)