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...

Class names should comply with a naming convention
Code Smell

Method names should comply with a naming convention
Code Smell

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

JUnit5 test classes and methods should have default package visibility
Code Smell

Track uses of "TODO" tags
Code Smell

Deprecated code should be removed
Code Smell

Annotated Mockito objects should be initialized
Bug

Custom resources should be closed
Bug

Threads should not be started in constructors
Code Smell

"main" should not "throw" anything
Code Smell

Track lack of copyright and license headers
Code Smell

Octal values should not be used
Code Smell

Only one method invocation is expected when testing runtime exceptions

Analyze your code

Code Smell | Major | junit tests

When verifying that code raises a runtime exception, a good practice is to avoid having multiple method calls inside the tested code, to be explicit about which method call is expected to raise the exception.

It increases the clarity of the test, and avoid incorrect testing when another method is actually raising the exception.

Noncompliant Code Example

```
@Test
public void testToString() {
  // Do you expect get() or toString() throwing the exceptio
  org.junit.Assert.assertThrows(IndexOutOfBoundsException.cl
}

@Test
public void testToStringTryCatchIdiom() {
  try {
    // Do you expect get() or toString() throwing the except
    get().toString();
    Assert.fail("Expected an IndexOutOfBoundsException to be
  } catch (IndexOutOfBoundsException e) {
    // Test exception message...
  }
}
```

Compliant Solution

```
@Test
public void testToString() {
   Object obj = get();
   Assert.assertThrows(IndexOutOfBoundsException.class, () -
}

@Test
public void testToStringTryCatchIdiom() {
  Object obj = get();
  try {
    obj.toString();
    Assert.fail("Expected an IndexOutOfBoundsException to be
  } catch (IndexOutOfBoundsException e) {
    // Test exception message...
  }
}
```

See

- JUnit exception testing documentation

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

Equality operators should not be used

Available In:

sonarlint | sonarcloud | sonarqube