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...

Code Smell

Execution of the Garbage Collector should be triggered only by the JVM
Code Smell

Constants should not be defined in interfaces
Code Smell

String literals should not be duplicated
Code Smell

Methods should not be empty
Code Smell

"Object.finalize()" should remain protected (versus public) when overriding
Code Smell

Exceptions should not be thrown in finally blocks
Code Smell

Constant names should comply with a naming convention
Code Smell

The Object.finalize() method should not be overridden
Code Smell

XML operations should not be vulnerable to injection attacks
Vulnerability

JSON operations should not be vulnerable to injection attacks
Vulnerability

XML signatures should be validated securely
Vulnerability

JUnit5 inner test classes should be annotated with @Nested

Analyze your code

Bug | Critical | junit tests

If not annotated with `@Nested`, an inner class containing some tests will never be executed during tests execution. While you could still be able to manually run its tests in an IDE, it won't be the case during the build. By contrast, a static nested class containing some tests should not be annotated with `@Nested`, JUnit5 will not share setup and state with an instance of its enclosing class.

This rule raises an issue on inner classes and static nested classes containing JUnit5 test methods which has a wrong usage of `@Nested` annotation.

Note: This rule does not check if the context in which JUnit 5 is running (e.g. Maven Surefire Plugin) is properly configured to execute static nested classes, it could not be the case using the default configuration.

Noncompliant Code Example

```
import org.junit.jupiter.api.Test;

class MyJunit5Test {
  @Test
  void test() { /* ... */ }

  class InnerClassTest { // Noncompliant, missing @Nested an
    @Test
    void test() { /* ... */ }
  }

  @Nested
  static class StaticNestedClassTest { // Noncompliant, inva
    @Test
    void test() { /* ... */ }
  }
}
```

Compliant Solution

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Nested;

class MyJunit5Test {
  @Test
  void test() { /* ... */ }

  @Nested
  class InnerClassTest {
    @Test
    void test() { /* ... */ }
  }

  static class StaticNestedClassTest {
    @Test
```

XML parsers should not be vulnerable to Denial of Service attacks
Vulnerability

XML parsers should not load external schemas
Vulnerability

Mobile database encryption keys should not be disclosed
Vulnerability

Reflection should not be vulnerable to injection attacks
Vulnerability

```
    void test() { /* ... */ }
  }
}
```

Available In:

sonarlint | sonarcloud | sonarqube