sonar RULES

Products ⌄

- 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...

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

XML parsers should not be vulnerable to Denial of Service attacks

Vulnerability

XML parsers should not load external schemas

Vulnerability

Only one method invocation is expected when testing checked exceptions

Analyze your code

Bug Critical junit tests

When verifying that code raises an exception, a good practice is to avoid having multiple method calls inside the tested code, to be explicit about what is exactly tested.

When two of the methods can raise the same **checked** exception, not respecting this good practice is a bug, since it is not possible to know what is really tested.

You should make sure that only one method can raise the expected checked exception in the tested code.

Noncompliant Code Example

```
@Test
public void testG() {
  // Do you expect g() or f() throwing the exception?
  assertThrows(IOException.class, () -> g(f(1)) ); // Noncom
}

@Test
public void testGTryCatchIdiom() {
  try { // Noncompliant
    g(f(1));
    Assert.fail("Expected an IOException to be thrown");
  } catch (IOException e) {
    // Test exception message...
  }
}

int f(int x) throws IOException {
  // ...
}

int g(int x) throws IOException {
  // ...
}
```

Compliant Solution

```
@Test
public void testG() {
  int y = f(1);
  // It is explicit that we expect an exception from g() and
  assertThrows(IOException.class, () -> g(y) );
}

@Test
public void testGTryCatchIdiom() {
  int y = f(1);
  try {
```

Mobile database encryption keys should not be disclosed

Vulnerability

Reflection should not be vulnerable to injection attacks

Vulnerability

Authorizations should be based on strong decisions

Vulnerability

OpenSAML2 should be configured to prevent authentication bypass

Vulnerability

```
    g(y);
    Assert.fail("Expected an IOException to be thrown");
  } catch (IOException e) {
    // Test exception message...
  }
}
```

Available In:

sonarlint sonarcloud sonarqube