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 handlers should preserve the original exceptions

Analyze your code

Code Smell | Major | cwe error-handling owasp cert suspicious

When handling a caught exception, the original exception's message and stack trace should be logged or passed forward.

Noncompliant Code Example

```
try {
  /* ... */
} catch (Exception e) {   // Noncompliant - exception is los
  LOGGER.info("context");
}

try {
  /* ... */
} catch (Exception e) {  // Noncompliant - exception is lost
  LOGGER.info(e.getMessage());
}

try {
  /* ... */
} catch (Exception e) {  // Noncompliant - original exceptio
  throw new RuntimeException("context");
}
```

Compliant Solution

```
try {
  /* ... */
} catch (Exception e) {
  LOGGER.info(e);  // exception is logged
}

try {
  /* ... */
} catch (Exception e) {
  throw new RuntimeException(e);   // exception stack trace
}

try {
  /* ... */
} catch (RuntimeException e) {
  doSomething();
  throw e;  // original exception passed forward
} catch (Exception e) {
  throw new RuntimeException(e);  // Conversion into uncheck
}
```

Exceptions

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

`InterruptedException`, `NumberFormatException`, `DateTimeParseException`, `ParseException` and `MalformedURLException` exceptions are arguably used to indicate nonexceptional outcomes. Similarly, handling `NoSuchMethodException` is often required when dealing with the Java reflection API.

Because they are part of Java, developers have no choice but to deal with them. This rule does not verify that those particular exceptions are correctly handled.

```
int myInteger;
try {
  myInteger = Integer.parseInt(myString);
} catch (NumberFormatException e) {
  // It is perfectly acceptable to not handle "e" here
  myInteger = 0;
}
```

Furthermore, no issue will be raised if the exception message is logged with additional information, as it shows that the developer added some context to the error message.

```
try {
  /* ... */
} catch (Exception e) {
  String message = "Exception raised while authenticating us
  LOGGER.warn(message); // Compliant - exception message log
}
```

See

- OWASP Top 10 2021 Category A9 - Security Logging and Monitoring Failures
- OWASP Top 10 2017 Category A10 - Insufficient Logging & Monitoring
- CERT, ERR00-J. - Do not suppress or ignore checked exceptions
- MITRE, CWE-778 - Insufficient Logging

Available In:

sonarlint | sonarcloud | sonarqube