




 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 rules632

Vulnerability53

Bug154

Security Hotspot36

Code Smell389

Quick Fix42

Tags ▾

Search by name... 🔍

Code Smell

Tests should use fixed data instead of randomized data

Code Smell

Spring's ModelAndViewAssert assertions should be used instead of other assertions

Code Smell

Lambdas should not have too many lines

Code Smell

"@EnableAutoConfiguration" should be fine-tuned

Code Smell

Enum values should be compared with "=="

Code Smell

Spring components should use constructor injection

Code Smell

Regex patterns should not be created needlessly

Code Smell

Track uses of disallowed constructors

Code Smell

Java 8's "Files.exists" should not be used

Code Smell

"Optional" should not be used for parameters

Code Smell

Tests should be kept in a dedicated source directory

Code Smell

Printf-style format strings should be used correctly

Analyze your code

Code Smell

Major

cert confusing

Because printf-style format strings are interpreted at runtime, rather than validated by the compiler, they can contain errors that result in the wrong strings being created. This rule statically validates the correlation of printf-style format strings to their arguments when calling the format(...) methods of java.util.Formatter, java.lang.String, java.io.PrintStream, MessageFormat, and java.io.PrintWriter classes and the printf(...) methods of java.io.PrintStream or java.io.PrintWriter classes.

Noncompliant Code Example

```
String.format("First {0} and then {1}", "foo", "bar"); //Noncompliant
String.format("Display %3$d and then %d", 1, 2, 3); //Noncompliant
String.format("Too many arguments %d and %d", 1, 2, 3); //Noncompliant
String.format("First Line\n"); //Noncompliant; \n should be escaped
String.format("Is myObject null ? %b", myObject); //Noncompliant
String.format("value is " + value); // Noncompliant
String s = String.format("string without arguments"); // Noncompliant

MessageFormat.format("Result '{0}'.", value); // Noncompliant
MessageFormat.format("Result {0}.", value, value); // Noncompliant
MessageFormat.format("Result {0}.", myObject.toString()); // Noncompliant

java.util.Logger logger;
logger.log(java.util.logging.Level.SEVERE, "Result {0}.", myObject);
logger.log(java.util.logging.Level.SEVERE, "Result.", new Exception());
logger.log(java.util.logging.Level.SEVERE, "Result '{0}'", 1);
logger.log(java.util.logging.Level.SEVERE, "Result " + param, 1);

org.slf4j.Logger slf4jLog;
org.slf4j.Marker marker;

slf4jLog.debug(marker, "message {}");
slf4jLog.debug(marker, "message", 1); // Noncompliant - String format

org.apache.logging.log4j.Logger log4jLog;
log4jLog.debug("message", 1); // Noncompliant - String concatenation
```





Compliant Solution

```
String.format("First %s and then %s", "foo", "bar");
String.format("Display %2$d and then %d", 1, 3);
String.format("Too many arguments %d %d", 1, 2);
String.format("First Line\n");
String.format("Is myObject null ? %b", myObject == null);
String.format("value is %d", value);
String s = "string without arguments";

MessageFormat.format("Result {0}.", value);
MessageFormat.format("Result '{0}' = {0}", value);
```

https://rules.sonarsource.com/java/RSPEC-3457

1/2

"this" should not be exposed from constructors  Code Smell
Classes should not have too many "static" imports  Code Smell
Escaped Unicode characters should not be used  Code Smell
Inner classes should not have too many lines of code  Code Smell

```
MessageFormat.format("Result {0}.", myObject);

java.util.Logger logger;
logger.log(java.util.logging.Level.SEVERE, "Result {0}.", my
logger.log(java.util.logging.Level.SEVERE, "Result {0}'", 14
logger.log(java.util.logging.Level.SEVERE, exception, () ->

org.slf4j.Logger slf4jLog;
org.slf4j.Marker marker;

slf4jLog.debug(marker, "message {}");
slf4jLog.debug(marker, "message {}", 1);

org.apache.logging.log4j.Logger log4jLog;
log4jLog.debug("message {}", 1);
```

See

- [CERT, FIO47-C](#). - Use valid format strings

Available In:

sonarlint

| sonarcloud

| sonarqube