




 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


Types should be used in lambdas





 "java.time" classes should be used for dates and times





 The names of methods with boolean return values should start with "is" or "has"





 Files should contain only one top-level class or interface each





 Classes should not have too many fields



 The ternary operator should not be used



 Standard functional interfaces should not be redefined






 "NullPointerException" should not be caught

 "NullPointerException" should not be explicitly thrown

 Classes should not have too many methods

 Methods should not have too many lines

"Preconditions" and logging arguments should not require evaluation

 Code Smell  Major  performance

Passing message arguments that require further evaluation into a Guava `com.google.common.base.Preconditions` check can result in a performance penalty. That's because whether or not they're needed, each argument must be resolved before the method is actually called.

Similarly, passing concatenated strings into a logging method can also incur a needless performance hit because the concatenation will be performed every time the method is called, whether or not the log level is low enough to show the message.

Instead, you should structure your code to pass static or pre-computed values into `Preconditions` conditions check and logging calls.

Specifically, the built-in string formatting should be used instead of string concatenation, and if the message is the result of a method call, then `Preconditions` should be skipped altogether, and the relevant exception should be conditionally thrown instead.

Noncompliant Code Example

```
logger.log(Level.DEBUG, "Something went wrong: " + message);

logger.fine("An exception occurred with message: " + message

LOG.error("Unable to open file " + csvPath, e); // Noncompl

Preconditions.checkState(a > 0, "Arg must be positive, but g

Preconditions.checkState(condition, formatMessage()); // No

Preconditions.checkState(condition, "message: %s", formatMes
```

Compliant Solution

```
logger.log(Level.SEVERE, "Something went wrong: {0} ", messa

logger.fine("An exception occurred with message: {}", messag

logger.log(Level.SEVERE, () -> "Something went wrong: " + me

LOG.error("Unable to open file {0}", csvPath, e);






if (LOG.isDebugEnabled() {
    LOG.debug("Unable to open file " + csvPath, e); // this i
}

Preconditions.checkState(arg > 0, "Arg must be positive, but

if (!condition) {
    throw new IllegalStateException(formatMessage()); // form
```

https://rules.sonarsource.com/java/RSPEC-2629

1/2


Track uses of "NOSONAR" comments
 Code Smell
Classes and enums with private members should have a constructor
 Code Smell
Track comments matching a regular expression
 Code Smell
Statements should be on separate lines


```
}  
  
if (!condition) {  
    throw new IllegalStateException("message: " + formatMessage(  
})
```

Exceptions

catch blocks are ignored, because the performance penalty is unimportant on exceptional paths (catch block should not be a part of standard program flow). Getters are ignored as well as methods called on annotations which can be considered as getters. This rule accounts for explicit test-level testing with SLF4J methods isXXXXEnabled and ignores the bodies of such if statements.

Available In:
 |  | 