Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

A field should not duplicate the name of its containing class
Code Smell

JUnit4 @Ignored and JUnit5 @Disabled annotations should be used to disable tests and should provide a rationale
Code Smell

Anonymous inner classes containing only one method should become lambdas
Code Smell

"switch" statements should not have too many "case" clauses
Code Smell

"for" loop stop conditions should be invariant
Code Smell

Sections of code should not be commented out
Code Smell

Non-constructor methods should not have the same name as the enclosing class
Code Smell

Exception types should not be tested using "instanceof" in catch blocks
Code Smell

Classes from "sun.*" packages should not be used
Code Smell

Throwable and Error should not be caught
Code Smell

Unused method parameters should be removed
Code Smell

Value-based classes should not be used for locking

Analyze your code

Bug Major multi-threading java8 lock-in

According to the documentation,

> A program may produce unpredictable results if it attempts to distinguish two references to equal values of a value-based class, whether directly via reference equality or indirectly via an appeal to synchronization…

This is because value-based classes are intended to be wrappers for value types, which will be primitive-like collections of data (similar to `structs` in other languages) that will come in future versions of Java.

> Instances of a value-based class …
> - do not have accessible constructors, but are instead instantiated through factory methods which make no commitment as to the identity of returned instances;

This means that you can't be sure you're the only one trying to lock on any given instance of a value-based class, opening your code up to contention and deadlock issues.

Under Java 8 breaking this rule may not actually break your code, but there are no guarantees of the behavior beyond that.

This rule raises an issue when a known value-based class is used for synchronization. That includes all the classes in the `java.time` package except `Clock`; the date classes for alternate calendars, `HijrahDate`, `JapaneseDate`, `MinguoDate`, `ThaiBuddhistDate`; and the optional classes: `Optional`, `OptionalDouble`, `OptionalLong`, `OptionalInt`.

Note that this rule is automatically disabled when the project's `sonar.java.source` is lower than `8`.

Noncompliant Code Example

```
Optional<Foo> fOpt = doSomething();
synchronized (fOpt) {  // Noncompliant
  // ...
}
```

See

- Value-based classes

Available In:
sonarlint sonarcloud sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective

owners. All rights are expressly reserved.
Privacy Policy

Only static class initializers should be used

Code Smell

Empty arrays and collections should be returned instead of null

Code Smell

"@Override" should be used on overriding and implementing methods

Code Smell

Enumeration should not be implemented

Code Smell