




 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

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

Permitted types of a sealed class should be omitted if they are declared in the same file

Analyze your code

- Code Smell
- Minor
- Quick Fix
- java17

sealed classes were introduced in Java 17. This feature is very useful if there is a need to define a strict hierarchy and restrict the possibility of extending classes. In order to mention all the allowed subclasses, there is a keyword `permits`, which should be followed by subclasses' names.

This notation is quite useful if subclasses of a given sealed class can be found in different files, packages, or even modules. In case when all subclasses are declared in the same file there is no need to mention the explicitly and `permits` part of a declaration can be omitted.

This rule reports an issue if all subclasses of a sealed class are declared in the same file as their superclass.

Noncompliant Code Example

```
sealed class A permits B, C, D, E {} // Noncompliant
final class B extends A {}
final class C extends A {}
final class D extends A {}
final class E extends A {}
```

Compliant Solution

```
sealed class A {} // Compliant
final class B extends A {}
final class C extends A {}
final class D extends A {}
final class E extends A {}
```

See

- Sealed Classes specification

Available In:

sonarlint | sonarcloud | sonarqube

<div><div>Local-Variable Type Inference should be used</div><div> Code Smell</div></div>
<div><div>Migrate your tests from JUnit4 to the new JUnit5 annotations</div><div> Code Smell</div></div>
<div><div>Track uses of disallowed classes</div><div> Code Smell</div></div>
<div><div>Track uses of "@SuppressWarnings" annotations</div><div> Code Smell</div></div>