




 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

Only one method invocation is expected when testing runtime exceptions

⚙️ Code Smell

⚙️ Code Smell

Exception testing via JUnit ExpectedException rule should not be mixed with other assertions

⚙️ Code Smell

⚙️ Code Smell

"@Deprecated" code marked for removal should never be used

⚙️ Code Smell

⚙️ Code Smell

Vararg method arguments should not be confusing

⚙️ Code Smell

⚙️ Code Smell

Whitespace for text block indent should be consistent

⚙️ Code Smell

⚙️ Code Smell

'List.remove()' should not be used in ascending 'for' loops

⚙️ Code Smell

⚙️ Code Smell

Collection constructors should not be used as java.util.function.Function

⚙️ Code Smell

⚙️ Code Smell

"else" statements should be clearly matched with an "if"

⚙️ Code Smell

⚙️ Code Smell

"Class.forName()" should not load JDBC 4.0+ drivers

⚙️ Code Smell

⚙️ Code Smell

Java features should be preferred to Guava

⚙️ Code Smell

⚙️ Code Smell

Nullness of parameters should be guaranteed

Reflection should not be used to increase accessibility of records' fields

🐞 Bug

🔴 Major

🏷️ java16

Analyze your code

In general, altering or bypassing the accessibility of classes, methods, or fields violates the encapsulation principle and could lead to runtime errors. For records the case is even trickier: you cannot change the visibility of records's fields and trying to update the existing value will lead to `IllegalAccessException` at runtime.

This rule raises an issue when reflection is used to change the visibility of a record's field, and when it is used to directly update a record's field value.

Noncompliant Code Example

```
record Person(String name, int age) {}

Person person = new Person("A", 26);
Field field = Person.class.getDeclaredField("name");
field.setAccessible(true); // secondary
field.set(person, "B"); // Noncompliant
```

See

- [Records specification](#)

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](#)

https://rules.sonarsource.com/java/RSPEC-6216

1/2

 Code Smell
"Integer.toHexString" should not be used to build hexadecimal strings  Code Smell
Asserts should not be used to check the parameters of a public method  Code Smell
Assignments should not be redundant  Code Smell
Methods should not have identical implementations  Code Smell