




 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... 🔍

Loops should not contain more than a single "break" or "continue" statement

Code Smell

Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"

Code Smell

"switch" statements should have at least 3 "case" clauses

Code Smell

A "while" loop should be used instead of a "for" loop

Code Smell

The default unnamed package should not be used

Code Smell

"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method

Code Smell

Package names should comply with a naming convention

Code Smell

Nested code blocks should not be used

Code Smell

Array designators "[]" should be on the type, not the variable

Code Smell

Array designators "[]" should be located after the type in method signatures

Code Smell

Type parameter names should comply with a naming convention

Code Smell

Records should be used instead of ordinary classes when representing immutable data structure

Analyze your code

Code SmellMajor?java16

In Java 16 `records` are finalized and can be safely used in production code. `Records` represent immutable read-only data structure and should be used instead of creating immutable classes. Immutability of records is guaranteed by the Java language itself, while implementing immutable classes on your own might lead to some bugs.

One of the important aspects of `records` is that final fields can't be overwritten using reflection.

This rule reports an issue on classes for which all these statements are true:

- all instance fields are private and final
- has only one constructor with a parameter for all fields
- has getters for all fields

Noncompliant Code Example

```
final class Person { // Noncompliant
    private final String name;
    private final int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {...}

    public int getAge() {...}

    @Override
    public boolean equals(Object o) {...}

    @Override
    public int hashCode() {...}

    @Override
    public String toString() {...}
}
```

Compliant Solution




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

See

- [Records specification](#)

https://rules.sonarsource.com/java/RSPEC-6206

1/2

Overriding methods should do more than simply call the same method in the super class  Code Smell
Classes that override "clone" should be "Cloneable" and call "super.clone()"  Code Smell
Public constants and fields initialized at declaration should be "static final" rather than merely "final"  Code Smell
Local variable and method parameter names should comply with a naming

Available In:  
 |  | 

© 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](#)