




 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... 🔍

Local variable and method parameter names should comply with a naming convention

Code Smell

Exception classes should be immutable

Code Smell

Field names should comply with a naming convention

Code Smell

Primitive wrappers should not be instantiated only for "toString" or "compareTo" calls

Code Smell

Case insensitive string comparisons should be made without intermediate upper or lower casing

Code Smell

Collection.isEmpty() should be used to test for emptiness

Code Smell

String.valueOf() should not be appended to a String

Code Smell

Interface names should comply with a naming convention

Code Smell

"throws" declarations should not be superfluous

Code Smell

Unnecessary imports should be removed

Code Smell

Return of boolean expressions should not be wrapped into an "if-then-else" statement

Restricted Identifiers should not be used as Identifiers

Analyze your code

Code SmellMajor ?

Even if it is technically possible, **Restricted Identifiers** should not be used as identifiers. This is only possible for compatibility reasons, using it in Java code is confusing and should be avoided.

Note that this applies to any version of Java, including the one where these identifiers are not yet restricted, to avoid future confusion.

This rule reports an issue when restricted identifiers:

- var
- yield
- record

are used as identifiers.

**Noncompliant Code Example**

```
var var = "var"; // Noncompliant: compiles but this code is
var = "what is this?";

int yield(int i) { // Noncompliant
    return switch (i) {
        case 1: yield(0); // This is a yield from switch express
        default: yield(i-1);
    };
}

String record = "record"; // Noncompliant
```

**Compliant Solution**

```
var myVariable = "var";

int minusOne(int i) {
    return switch (i) {
        case 1: yield(0);
        default: yield(i-1);
    };
}

String myRecord = "record";
```

**See**

- [JLS16, 3.8: Identifiers](#)

Available In:






sonarlint

sonarcloud

sonarqube

https://rules.sonarsource.com/java/RSPEC-6213

1/2

 Code Smell
<b>Boolean literals should not be redundant</b>  Code Smell
<b>Modifiers should be declared in the correct order</b>  Code Smell
<b>Empty statements should be removed</b>  Code Smell
<b>Class variable fields should not have public accessibility</b>  Code Smell

© 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](#)