

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...

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

Subclasses that add fields should override "equals"

Analyze your code

Code Smell

Minor

suspicious

Extend a class that overrides equals and add fields without overriding equals in the subclass, and you run the risk of non-equivalent instances of your subclass being seen as equal, because only the superclass fields will be considered in the equality test.

This rule looks for classes that do all of the following:

- extend classes that override equals.
- do not themselves override equals.
- add fields.

Noncompliant Code Example

```
public class Fruit {
    private Season ripe;

    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }
        if (this.class != obj.class) {
            return false;
        }
        Fruit fobj = (Fruit) obj;
        if (ripe.equals(fobj.getRipe()) {
            return true;
        }
        return false;
    }
}

public class Raspberry extends Fruit { // Noncompliant; ins
    private Color ripeColor;
}
```





Compliant Solution

```
public class Fruit {
    private Season ripe;

    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }
        if (this.class != obj.class) {
            return false;
        }
        Fruit fobj = (Fruit) obj;
        if (ripe.equals(fobj.getRipe()) {
            return true;
```

https://rules.sonarsource.com/java/RSPEC-2160

1/2

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

```
    }
    return false;
  }
}

public class Raspberry extends Fruit {
  private Color ripeColor;

  public boolean equals(Object obj) {
    if (! super.equals(obj)) {
      return false;
    }
    Raspberry fobj = (Raspberry) obj;
    if (ripeColor.equals(fobj.getRipeColor())) { // added fi
      return true;
    }
    return false;
  }
}
```

Available In:

**sonarlint**  | **sonarcloud**  | **sonarqube** 