

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

632

All rules

53

Vulnerability

154

Bug

36

Security Hotspot

389

Code Smell

42

Quick Fix

Java

Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

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

"Bean Validation" (JSR 380) should be properly configured

Analyze your code

Code Smell

Critical

Bean Validation as per defined by JSR 380 can be triggered programmatically or also executed by the Bean Validation providers. However something should tell the Bean Validation provider that a variable must be validated otherwise no validation will happen. This can be achieved by annotating a variable with javax.validation.Valid and unfortunately it's easy to forget to add this annotation on complex Beans.

Not annotating a variable with @Valid means Bean Validation will not be triggered for this variable, but readers may overlook this omission and assume the variable will be validated.

This rule will run by default on all Class'es and therefore can generate a lot of noise. This rule should be restricted to run only on certain layers. For this reason, the "Restrict Scope of Coding Rules" feature should be used to check for missing @Valid annotations only on some packages of the application.

Noncompliant Code Example

```
import javax.validation.Valid;
import javax.validation.constraints.NotNull;

public class User {
    @NotNull
    private String name;
}

public class Group {
    @NotNull
    private List<User> users; // Noncompliant; User instances
}

public class MyService {
    public void login(User user) { // Noncompliant; parameter
    }
}
```

Compliant Solution





```
import javax.validation.Valid;
import javax.validation.constraints.NotNull;

public class User {
    @NotNull
    private String name;
}

public class Group {
    @Valid
    @NotNull
    private List<User> users; // Compliant; User instances are
```

https://rules.sonarsource.com/java/RSPEC-5128

1/2

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

```
@NotNull
// preferred style as of Bean Validation 2.0
private List<@Valid User> users2; // Compliant; User insta
}

public class MyService {
    public void login(@Valid User user) { // Compliant
    }
}
```

See

- [Bean Validation 2.0 \(JSR 380\)](#)

Available In:

