




 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... 🔍

⚠️ Bug

Literal suffixes should be upper case

🔧 Code Smell

Unicode-aware versions of character classes should be preferred

🔧 Code Smell

Use Java 12 "switch" expression

🔧 Code Smell

"serialVersionUID" should not be declared blindly

🔧 Code Smell

"Stream.collect()" calls should not be redundant

🔧 Code Smell

Local constants should follow naming conventions for constants

🔧 Code Smell

Unit tests should throw exceptions

🔧 Code Smell

Test methods should comply with a naming convention

🔧 Code Smell

Value-based objects should not be serialized

🔧 Code Smell

Default annotation parameter values should not be passed as arguments

🔧 Code Smell

Method parameters should be declared with base types

🔧 Code Smell

Fields should not be initialized to default values

🔧 Code Smell

Two branches in a conditional structure should not have exactly the same implementation

Analyze your code

🔧 Code Smell

🔴 Major ?

🔍 design suspicious

Having two cases in a switch statement or two branches in an if chain with the same implementation is at best duplicate code, and at worst a coding error. If the same logic is truly needed for both instances, then in an if chain they should be combined, or for a switch, one should fall through to the other.

Noncompliant Code Example

```
switch (i) {
    case 1:
        doFirstThing();
        doSomething();
        break;
    case 2:
        doSomethingDifferent();
        break;
    case 3: // Noncompliant; duplicates case 1's implementation
        doFirstThing();
        doSomething();
        break;
    default:
        doTheRest();
}

if (a >= 0 && a < 10) {
    doFirstThing();
    doTheThing();
}
else if (a >= 10 && a < 20) {
    doTheOtherThing();
}
else if (a >= 20 && a < 50) {
    doFirstThing();
    doTheThing(); // Noncompliant; duplicates first condition
}
else {
    doTheRest();
}
```





Exceptions

Blocks in an if chain that contain a single line of code are ignored, as are blocks in a switch statement that contain a single line of code with or without a following break.

```
if (a == 1) {
    doSomething(); //no issue, usually this is done on purpose
} else if (a == 2) {
    doSomethingElse();
} else {
```

https://rules.sonarsource.com/java/RSPEC-1871

1/2

 Code Smell
Multiple loops over the same set should be combined
 Code Smell
Classes without "public" constructors should be "final"
 Code Smell
Unnecessary semicolons should be omitted
 Code Smell
Literal boolean values and nulls should not be used in assertions

```
doSomething();
}
```

But this exception does not apply to `if` chains without `else`-s, or to `switch`-es without default clauses when all branches have the same single line of code. In case of `if` chains with `else`-s, or of `switch`-es with default clauses, rule {rule:java:S3923} raises a bug.

```
if (a == 1) {
    doSomething(); //Noncompliant, this might have been done
} else if (a == 2) {
    doSomething();
}
```

Available In:
 |  | 