

 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 rules 632

Vulnerability 53

Bug 154

Security Hotspot 36

Code Smell 389

Quick Fix 42

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

Custom getter method should not be used to override record's getter behavior

Analyze your code

Code Smell Major ? java16

Before records appeared in Java 16, there was a common way to represent getters for private fields of a class: a method named "get" with a capitalized field name. For example, for a `String` field named "myField" the signature of the getter method will be: `public String getMyField()`

In records, getters are named differently. Getters created by default do not contain the "get" prefix. So for a record's `String` field "myField" the getter method will be: `public String myField()`

This means that if you want to override the default getter behavior it is better to use the method provided by records instead of creating a new one. Otherwise, this will bring confusion to the users of the record as two getters will be available and even leads to bugs if the behavior is different from the default one.

This rule raises an issue when a record contains a getter named "get" with a capitalized field name that is not behaving the same as the default one.

Noncompliant Code Example

```
record Person(String name, int age) {
    public String getName() { // Noncompliant
        return name.toUpperCase(Locale.ROOT);
    }
}
```

Compliant Solution

```
record Person(String name, int age) {
    @Override
    public String name() { // Compliant
        return name.toUpperCase(Locale.ROOT);
    }
}

record Person(String name, int age) {
    public String getNameUpperCase() { // Compliant
        return name.toUpperCase(Locale.ROOT);
    }
}

record Person(String name, int age) {
    public String getName() { // Compliant, is equivalent to
        return name;
    }
}

record Person(String name, int age) {
    @Override
    public String name() { // Compliant
        return name.toUpperCase(Locale.ROOT);
    }
}
```


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

```
public String getName() { // Compliant, equal to 'name()'
    return name.toUpperCase(Locale.ROOT);
}
```

Exceptions

If the implementations of `getMyField()` and `myField()` methods are equivalent, the issue should not be raised as this was probably done to support compatibility with the previous convention.

See

- [Records specification](#)

Available In:

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