




 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... 🔍

 Code Smell


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




Exception classes should be immutable




Field names should comply with a naming convention




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




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




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




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




Interface names should comply with a naming convention



"throws" declarations should not be superfluous




Unnecessary imports should be removed





Return of boolean expressions should

### Similar tests should be grouped in a single Parameterized test

Analyze your code

 Code Smell

 Major ?

 tests bad-practice clumsy

When multiple tests differ only by a few hardcoded values they should be refactored as a single "parameterized" test. This reduces the chances of adding a bug and makes them more readable. Parameterized tests exist in most test frameworks (JUnit, TestNG, etc...).

The right balance needs of course to be found. There is no point in factorizing test methods when the parameterized version is a lot more complex than initial tests.

This rule raises an issue when at least 3 tests could be refactored as one parameterized test with less than 4 parameters. Only test methods which have at least one duplicated statement are considered.

#### Noncompliant Code Example

with JUnit 5

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class AppTest
{
    @Test
    void test_not_null1() { // Noncompliant. The 3 following
        setupTax();
        assertNotNull(getTax(1));
    }

    @Test
    void test_not_null2() {
        setupTax();
        assertNotNull(getTax(2));
    }





    @Test
    void test_not_nul3l() {
        setupTax();
        assertNotNull(getTax(3));
    }

    @Test
    void testLevel1() { // Noncompliant. The 3 following te
        setLevel(1);
        runGame();
        assertEquals(playerHealth(), 100);
    }

    @Test
    void testLevel2() { // Similar test
        setLevel(2);
        runGame();
        assertEquals(playerHealth(), 200);
    }
}
```

https://rules.sonarsource.com/java/RSPEC-5976

1/2

not be wrapped into an "if-then-else" statement
 Code Smell
Boolean literals should not be redundant
 Code Smell
Modifiers should be declared in the correct order
 Code Smell
Empty statements should be removed
 Code Smell
Class variable fields should not have

```
    }

    @Test
    void testLevel3() { // Similar test
        setLevel(3);
        runGame();
        assertEquals(playerHealth(), 300);
    }
}
```

Compliant Solution

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

public class AppTest
{

    @ParameterizedTest
    @ValueSource(ints = {1, 2, 3})
    void test_not_null(int arg) {
        setupTax();
        assertNotNull(getTax(arg));
    }

    @ParameterizedTest
    @CsvSource({
        "1, 100",
        "2, 200",
        "3, 300",
    })
    void testLevels(int level, int health) {
        setLevel(level);
        runGame();
        assertEquals(playerHealth(), health);
    }
}
```

See

- [Modern Best Practices for Testing in Java - Philipp Hauer](#)
- [JUnit 5 documentation - Parameterized tests](#)
- [Writing Parameterized Tests With JUnit 4](#)
- [TestNG documentation - Parameters](#)

Available In:

