




 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... 🔍

Consecutive AssertJ "assertThat" statements should be chained

Code Smell

Chained AssertJ assertions should be simplified to the corresponding dedicated assertion

Code Smell

Exception testing via JUnit @Test annotation should be avoided

Code Smell

Escape sequences should not be used in text blocks

Code Smell

Simple string literal should be used for single line strings

Code Smell

Boxed "Boolean" should be avoided in boolean expressions

Code Smell

Type parameters should not shadow other type parameters

Code Smell

"read(byte[],int,int)" should be overridden

Code Smell

An iteration on a Collection should be performed on the type handled by the Collection

Code Smell

"StandardCharsets" constants should be preferred

Code Smell

"@CheckForNull" or "@Nullable" should not be used on primitive types

Code Smell

"BigDecimal(double)" should not be used

Analyze your code

BugMajorQuick Fixcert

Because of floating point imprecision, you're unlikely to get the value you expect from the `BigDecimal(double)` constructor.

From [the JavaDocs](#):

The results of this constructor can be somewhat unpredictable. One might assume that writing `new BigDecimal(0.1)` in Java creates a `BigDecimal` which is exactly equal to 0.1 (an unscaled value of 1, with a scale of 1), but it is actually equal to 0.1000000000000000055511151231257827021181583404541015625. This is because 0.1 cannot be represented exactly as a double (or, for that matter, as a binary fraction of any finite length). Thus, the value that is being passed in to the constructor is not exactly equal to 0.1, appearances notwithstanding.

Instead, you should use `BigDecimal.valueOf`, which uses a string under the covers to eliminate floating point rounding errors, or the constructor that takes a `String` argument.

Noncompliant Code Example

```
double d = 1.1;

BigDecimal bd1 = new BigDecimal(d); // Noncompliant; see com
BigDecimal bd2 = new BigDecimal(1.1); // Noncompliant; same
```

Compliant Solution

```
double d = 1.1;

BigDecimal bd1 = BigDecimal.valueOf(d);
BigDecimal bd2 = new BigDecimal("1.1"); // using String cons
```

See

- [CERT, NUM10-J](#). - Do not construct `BigDecimal` objects from floating-point literals

Available In:

sonarlint





sonarcloud

sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.  
[Privacy Policy](#)

https://rules.sonarsource.com/java/RSPEC-2111

1/2

 Code Smell
<b>Composed "@RequestMapping" variants should be preferred</b>  Code Smell
<b>"write(byte[],int,int)" should be overridden</b>  Code Smell
<b>Functional Interfaces should be as specialised as possible</b>  Code Smell
<b>Null checks should not be used with "instanceof"</b>