




 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... 🔍

Creating cookies without the "HttpOnly" flag is security-sensitive

Security Hotspot

Creating cookies without the "secure" flag is security-sensitive

Security Hotspot

Using hardcoded IP addresses is security-sensitive

Security Hotspot

'serialVersionUID' field should not be set to '0L' in records

Code Smell

Permitted types of a sealed class should be omitted if they are declared in the same file

Code Smell

Switch arrow labels should not use redundant keywords

Code Smell

Text blocks should not be used in complex expressions

Code Smell

Pattern Matching for "instanceof" operator should be used instead of simple "instanceof" + cast

Code Smell

Call to Mockito method "verify", "when" or "given" should be simplified

Code Smell

Character classes should be preferred over reluctant quantifiers in regular expressions

Code Smell

Consecutive AssertJ "assertThat" statements should be chained

Code Smell

Silly String operations should not be made

Analyze your code

Bug

Major

clumsy

Creating a substring from 0 to the end is silly. You'll end up with the same string you started with. Using the value of `String.length` as either the start or end of a substring has similarly predictable results.

Calling `String.contains` with the argument being identical to the `String` on which `contains` is invoked doesn't make sense.

Noncompliant Code Example

```
String speech = "Now is the time for all good people to come  
  
String s1 = speech.substring(0); // Noncompliant. Yields the  
String s2 = speech.substring(speech.length()); // Noncompliant  
String s3 = speech.substring(5, speech.length()); // Noncompliant  
  
if (speech.contains(speech)) { // Noncompliant  
    // always true  
}
```

Compliant Solution

```
String speech = "Now is the time for all good people to come  
  
String s1 = speech;  
String s2 = "";  
String s3 = speech.substring(5);
```

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-2121

1/2

| |
|---|
| <div>Chained AssertJ assertions should be simplified to the corresponding dedicated assertion</div> <div> Code Smell</div> |
| <div>Exception testing via JUnit @Test annotation should be avoided</div> <div> Code Smell</div> |
| <div>Escape sequences should not be used in text blocks</div> <div> Code Smell</div> |
| <div>Simple string literal should be used for single line strings</div> <div></div> |