




 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


Redundant pairs of parentheses should be removed





 Inheritance tree of classes should not be too deep





 Nested blocks of code should not be left empty





 Methods should not have too many parameters




 Unused "private" fields should be removed



 Collapsible "if" statements should be merged



 Unused labels should be removed



 Standard outputs should not be used directly to log anything



 OS commands should not be vulnerable to argument injection attacks



 "ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization





 Logging should not be vulnerable to injection attacks



### Inappropriate regular expressions should not be used

Analyze your code

 Bug

 Major

?

Regular expressions are powerful but tricky, and even those long used to using them can make mistakes.

The following should not be used as regular expressions:




- `.` - matches any single character. Used in `replaceAll`, it matches *everything*
- `|` - normally used as an option delimiter. Used stand-alone, it matches the space between characters
- `File.separator` - matches the platform-specific file path delimiter. On Windows, this will be taken as an escape character

#### Noncompliant Code Example

```
String str = "/File|Name.txt";

String clean = str.replaceAll(".", ""); // Noncompliant; prob
String clean2 = str.replaceAll("|", "_"); // Noncompliant; yi
String clean3 = str.replaceAll(File.separator, ""); // Noncom





String clean4 = str.replaceFirst(".", ""); // Noncompliant;
String clean5 = str.replaceFirst("|", "_"); // Noncompliant;
String clean6 = str.replaceFirst(File.separator, ""); // Noncom
```

Available In:  
 |  | 

© 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-2639

1/2

 Vulnerability
Exceptions should not be thrown from servlet methods  Vulnerability
Return values should not be ignored when they contain the operation status code  Bug
Repeated patterns in regular expressions should not match the empty string  Bug
AssertJ assertions "allMatch" and "doesNotContain" should also test