




 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... 🔍

should be consistent

Code Smell

'List.remove()' should not be used in ascending 'for' loops

Code Smell

Collection constructors should not be used as java.util.function.Function

Code Smell

"else" statements should be clearly matched with an "if"

Code Smell

"Class.forName()" should not load JDBC 4.0+ drivers

Code Smell

Java features should be preferred to Guava

Code Smell

Nullness of parameters should be guaranteed

Code Smell

"Integer.toHexString" should not be used to build hexadecimal strings

Code Smell

Asserts should not be used to check the parameters of a public method

Code Smell

Assignments should not be redundant

Code Smell

Methods should not have identical implementations

Code Smell

"java.nio.Files#delete" should be preferred

Code Smell

### The regex escape sequence \cX should only be used with characters in the @-\_ range

Analyze your code

Bug

Major

regex

In regular expressions the escape sequence \cX, where the X stands for any character that's either @, any capital ASCII letter, [, \, ], ^ or \_ , represents the control character that "corresponds" to the character following \c, meaning the control character that comes 64 bytes before the given character in the ASCII encoding.

In some other regex engines (for example in that of Perl) this escape sequence is case insensitive and \cd produces the same control character as \cD. Further using \c with a character that's neither @, any ASCII letter, [, \, ], ^ nor \_ , will produce a warning or error in those engines. Neither of these things is true in Java, where the value of the character is always XORed with 64 without checking that this operation makes sense. Since this won't lead to a sensible result for characters that are outside of the @ to \_ range, using \c with such characters is almost certainly a mistake.

#### Noncompliant Code Example

```
Pattern.compile("\\ca"); // Noncompliant, 'a' is not an upper case letter
Pattern.compile("\\c!"); // Noncompliant, '!' is outside of the @-_ range
```

#### Compliant Solution

```
Pattern.compile("\\cA"); // Compliant, this will match the "A" character
Pattern.compile("\\c^"); // Compliant, this will match the " " character
```

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-6070

1/2

<div>Unused "private" classes should be removed</div> <div> Code Smell</div>
<div>"Stream.peek" should be used with caution</div> <div> Code Smell</div>
<div>"Map.get" and value test should be replaced with single method call</div> <div> Code Smell</div>
<div>"@RequestMapping" methods should not be "private"</div> <div> Code Smell</div>