


-  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

"main" should not "throw" anything

Code Smell

Track lack of copyright and license headers

Code Smell

Octal values should not be used

Code Smell

Exit methods should not be called

Code Smell

HTTP response headers should not be vulnerable to injection attacks

Vulnerability

Members of Spring components should be injected

Vulnerability

Classes should not be loaded dynamically

Vulnerability

Equality operators should not be used in "for" loop termination conditions

Code Smell

"Bean Validation" (JSR 380) should be properly configured

Code Smell

Spring beans should be considered by "@ComponentScan"

Code Smell

Number patterns should be regular

Code Smell

Lazy initialization of "static" fields should be "synchronized"

Code Smell

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

Analyze your code

Code Smell

Major ?

When `List.remove()` is called it will shrink the list. If this is done inside the ascending loop iterating through all elements it will skip the element after the removed index.

Noncompliant Code Example

```
void removeFrom(List<String> list) {
    // expected: iterate over all the elements of the list
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i).isEmpty()) {
            // actual: remaining elements are shifted, so the one
            list.remove(i); // Noncompliant
        }
    }
}
```

Compliant Solution

You can either adjust the loop index to account for the change in the size of the list

```
static void removeFrom(List<String> list) {
    // expected: iterate over all the elements of the list
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i).isEmpty()) {
            // actual: remaining elements are shifted, so the on
            list.remove(i);
            i--;
        }
    }
}
```

Or preferably it's probably better to rely on Java 8's `removeIf` method

```
static void removeFrom(List<String> list) {
    list.removeIf(String::isEmpty);
}
```

Exceptions


The descending loop doesn't have this issue, because the index will be correct when we loop in descending order

```
void removeFrom(List<String> list) {
    for (int i = list.size() - 1; i >= 0; i--) {
        if (list.get(i).isEmpty()) {
            list.remove(i);
        }
    }
}
```


https://rules.sonarsource.com/java/RSPEC-5413

1/2


Wildcard imports should not be used

 Code Smell


Modulus results should not be checked for direct equality

 Code Smell

Comparators should be "Serializable"




 Code Smell

"Serializable" classes should have a "serialVersionUID"

 Code Smell

```
    }  
    }  
}
```

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](#)