




 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... 🔍

positive numbers
Code Smell

Method overrides should not change contracts
Code Smell

Whitespace and control characters in literals should be explicit
Code Smell

Null should not be returned from a "Boolean" method
Code Smell

Classes should not access their own subclasses during initialization
Code Smell

"Object.wait(...)" and "Condition.await(...)" should be called inside a "while" loop
Code Smell

IllegalMonitorStateException should not be caught
Code Smell

JUnit assertions should not be used in "run" methods
Code Smell

Class names should not shadow interfaces or superclasses
Code Smell

"Cloneables" should implement "clone"
Code Smell

Try-with-resources should be used
Code Smell

"readResolve" methods should be inheritable
Code Smell

Unsupported methods should not be called on some collection implementations

Analyze your code

Bug Critical

The Java Collections framework defines interfaces such as `java.util.List` or `java.util.Map`. Several implementation classes are provided for each of those interfaces to fill different needs: some of the implementations guarantee a few given performance characteristics, some others ensure a given behavior, for example immutability.

Among the methods defined by the interfaces of the Collections framework, some are declared as "optional": an implementation class may choose to throw an `UnsupportedOperationException` when one of those methods is called. For example, `java.util.Collections.emptyList()` returns an implementation of `java.util.List` which is documented as "immutable": calling the `add` method on this object triggers an `UnsupportedOperationException`.

When calling one of the "optional" methods, a developer should therefore make sure that the implementation class on which the call is made indeed supports this method.

Noncompliant Code Example

```
List<String> list = Collections.emptyList();
if (someCondition) {
    list.add("hello"); // Noncompliant; throws an UnsupportedOperationException
}
return list;
```

Compliant Solution

```
List<String> list = new ArrayList<>();
if (someCondition) {
    list.add("hello");
}
return list;
```

or

```
if (someCondition) {
    return Collections.singletonList("hello");
}
return Collections.emptyList();
```

See

- [Collections Framework Overview](#) in the Java documentation

Available In:
sonarcloud

https://rules.sonarsource.com/java/RSPEC-6322

1/2

"for" loop increment clauses should modify the loops' counters

 Code Smell

Fields in a "Serializable" class should either be transient or serializable

 Code Smell

Package declaration should match source file directory

 Code Smell

Generic wildcard types should not be used in return types

 Code Smell

© 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](#)