


-  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


Labels should not be used




Utility classes should not have public constructors




Local variables should not shadow class fields




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

## "null" should not be used with "Optional"

Analyze your code

 Bug

 Major

 java8

The concept of Optional is that it will be used when null could cause errors. In a way, it replaces null, and when Optional is in use, there should never be a question of returning or receiving null from a call.

### Noncompliant Code Example

```
public void doSomething () {
    Optional<String> optional = getOptional();
    if (optional != null) { // Noncompliant
        // do something with optional...
    }
    Optional<String> text = null; // Noncompliant, a variable
    // ...
}




@Nullable // Noncompliant
public Optional<String> getOptional() {
    // ...
    return null; // Noncompliant
}
```

### Compliant Solution

```
public void doSomething () {
    Optional<String> optional = getOptional();
    optional.ifPresent(
        // do something with optional...
    );
    Optional<String> text = Optional.empty();
    // ...
}

public Optional<String> getOptional() {
    // ...
    return Optional.empty();
}
```





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-2789

1/2

<b>vulnerable to argument injection attacks</b>  Vulnerability
<b>"ActiveMQConnectionFactory" should not be vulnerable to malicious code deserialization</b>  Vulnerability
<b>Logging should not be vulnerable to injection attacks</b>  Vulnerability
<b>Exceptions should not be thrown from servlet methods</b>  Vulnerability