




 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... 🔍


 Bug


"volatile" variables should not be used with compound operators




 Bug


"getClass" should not be used for synchronization




 Bug


Min and max used in combination should not always return the same value




 Bug


Assignment of lazy-initialized members should be the last step with double-checked locking




 Bug


"String" calls should not go beyond their bounds



 Bug

Raw byte values should not be used in bitwise operations in combination with shifts



 Bug

Getters and setters should be synchronized in pairs



 Bug

Non-thread-safe fields should not be static



 Bug

"null" should not be used with "Optional"



 Bug

Unary prefix operators should not be repeated




 Bug


"+=" should not be used instead of "+="





## Methods setUp() and tearDown() should be correctly annotated starting with JUnit4

Analyze your code

 Code Smell

 Critical





The setUp() and tearDown() methods (initially introduced with JUnit3 to execute a block of code before and after each test) need to be correctly annotated with the equivalent annotation in order to preserve the same behavior when migrating from JUnit3 to JUnit4 or JUnit5.

This rule consequently raise issues on setUp() and tearDown() methods which are not annotated in test classes.

### Noncompliant Code Example

- JUnit4:

```
public void setUp() { ... } // Noncompliant; should be annot
public void tearDown() { ... } // Noncompliant; should be a
```
- JUnit5:

```
public void setUp() { ... } // Noncompliant; should be annot
public void tearDown() { ... } // Noncompliant; should be a
```

### Compliant Solution

- JUnit4:




```
@Before
public void setUp() { ... }
```

```
@After
public void tearDown() { ... }
```
- JUnit5:

```
@BeforeEach
void setUp() { ... }
```

```
@AfterEach
void tearDown() { ... }
```





Available In:

 |  | 

https://rules.sonarsource.com/java/RSPEC-5826

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of

1/2

 Bug
"read" and "readLine" return values should be used  Bug
Inappropriate regular expressions should not be used  Bug
Conditionally executed code should be reachable  Bug
"notifyAll" should be used