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 rules 632 | Vulnerability 53 | Bug 154 | Security Hotspot 36 | Code Smell 389 | Quick Fix 42

Tags

Search by name...

Abstract class names should comply with a naming convention
Code Smell

Strings literals should be placed on the left side when checking for equality
Code Smell

Files should contain an empty newline at the end
Code Smell

Source code should be indented consistently
Code Smell

A close curly brace should be located at the beginning of a line
Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be on two different lines
Code Smell

Close curly brace and the next "else", "catch" and "finally" keywords should be located on the same line
Code Smell

An open curly brace should be located at the beginning of a line
Code Smell

An open curly brace should be located at the end of a line
Code Smell

Tabulation characters should not be used
Code Smell

Functions should not be defined with a variable number of arguments
Code Smell

Unused "private" fields should be removed

Analyze your code

Code Smell | Major | Quick Fix | unused

If a `private` field is declared but not used in the program, it can be considered dead code and should therefore be removed. This will improve maintainability because developers will not wonder what the variable is used for.

Note that this rule does not take reflection into account, which means that issues will be raised on `private` fields that are only accessed using the reflection API.

Noncompliant Code Example

```
public class MyClass {
  private int foo = 42;

  public int compute(int a) {
    return a * 42;
  }

}
```

Compliant Solution

```
public class MyClass {
  public int compute(int a) {
    return a * 42;
  }
}
```

Exceptions

The rule admits 3 exceptions:

- Serialization id fields
- Annotated fields
- Fields from classes with native methods

Serialization id fields

The Java serialization runtime associates with each serializable class a version number, called `serialVersionUID`, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.

A serializable class can declare its own `serialVersionUID` explicitly by declaring a field named `serialVersionUID` that must be static, final, and of type long. By definition those `serialVersionUID` fields should not be reported by this rule:

```
public class MyClass implements java.io.Serializable {
  private static final long serialVersionUID = 42L;
}
```

Local-Variable Type Inference should be used

Code Smell

Migrate your tests from JUnit4 to the new JUnit5 annotations

Code Smell

Track uses of disallowed classes

Code Smell

Track uses of "@SuppressWarnings" annotations

Code Smell

Annotated fields

The unused field in this class will not be reported by the rule as it is annotated.

```
public class MyClass {
  @SomeAnnotation
  private int unused;
}
```

Fields from classes with native methods

The unused field in this class will not be reported by the rule as it might be used by native code.

```
public class MyClass {
  private int unused = 42;
  private native static void doSomethingNative();
}
```

Available In:

sonarlint | sonarcloud | sonarqube