




 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... 🔍

Classes extending java.lang.Thread should override the "run" method

Bug

"Double.longBitsToDouble" should not be used for "int"

Bug

Values should not be uselessly incremented

Bug

Silly String operations should not be made

Bug

Non-serializable classes should not be written

Bug

"hashCode" and "toString" should not be called on array instances

Bug

Collections should not be passed as arguments to their own methods

Bug

"BigDecimal(double)" should not be used

Bug

Invalid "Date" values should not be used

Bug

Reflection should not be used to check non-runtime annotations

Bug

Custom serialization method signatures should meet requirements

Bug

"Externalizable" classes should have no-arguments constructors

Classes should not access their own subclasses during initialization

Analyze your code

Code SmellCritical?cert

When a parent class references a member of a subclass during its own initialization, the results might not be what you expect because the child class might not have been initialized yet. This could create what is known as an "initialisation cycle", or even a deadlock in some extreme cases.

To make things worse, these issues are very hard to diagnose so it is highly recommended you avoid creating this kind of dependencies.

Noncompliant Code Example

```
class Parent {
    static int field1 = Child.method(); // Noncompliant
    static int field2 = 42;

    public static void main(String[] args) {
        System.out.println(Parent.field1); // will display "0" i
    }
}

class Child extends Parent {
    static int method() {
        return Parent.field2;
    }
}
```

See

- CERT, DCL00-J. - Prevent class initialization cycles
- Java Language Specifications - Section 12.4: Initialization of Classes and Interfaces

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-2390

1/2

 Bug
<div>Classes should not be compared by name</div> <div> Bug</div>
<div>Related "if/else if" statements should not have the same condition</div> <div> Bug</div>
<div>Synchronization should not be done on instances of value-based classes</div> <div> Bug</div>
<div>"Iterator.hasNext()" should not call "iterator.next()"</div>