




 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

 Bug
Custom serialization method signatures should meet requirements
 Bug
"Externalizable" classes should have no-arguments constructors
 Bug
Classes should not be compared by name
 Bug
Related "if/else if" statements should not have the same condition
 Bug
Synchronization should not be done on instances of value-based classes
 Bug
"Iterator.hasNext()" should not call "Iterator.next()"
 Bug
Identical expressions should not be used on both sides of a binary operator
 Bug
Loops with at most one iteration should be refactored
 Bug
Variables should not be self-assigned
 Bug
"StringBuilder" and "StringBuffer" should not be instantiated with a character
 Bug
Methods should not be named "toString", "hashCode" or "equal"
 Bug

"Cloneables" should implement "clone"

Analyze your code

 Code Smell

 Critical

 convention

api-design

Simply implementing `Cloneable` without also overriding `Object.clone()` does not necessarily make the class cloneable. While the `Cloneable` interface does not include a `clone` method, it is required by convention, and ensures true cloneability. Otherwise the default JVM `clone` will be used, which copies primitive values and object references from the source to the target. I.e. without overriding `clone`, any cloned instances will potentially share members with the source instance.

Removing the `Cloneable` implementation and providing a good copy constructor is another viable (some say preferable) way of allowing a class to be copied.

Noncompliant Code Example

```
class Team implements Cloneable { // Noncompliant
    private Person coach;
    private List<Person> players;
    public void addPlayer(Person p) {...}
    public Person getCoach() {...}
}
```





Compliant Solution

```
class Team implements Cloneable {
    private Person coach;
    private List<Person> players;
    public void addPlayer(Person p) { ... }
    public Person getCoach() { ... }

    @Override
    public Object clone() {
        Team clone = (Team) super.clone();
        //...
    }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

<div><div>"Thread.run()" should not be called directly</div><div> Bug</div></div>
<div><div>"equals" method overrides should accept "Object" parameters</div><div> Bug</div></div>
<div><div>The Object.finalize() method should not be called</div><div> Bug</div></div>
<div><div>Enabling file access for WebViews is security-sensitive</div><div> Security Hotspot</div></div>