Java static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVA code

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

Lazy initialization of "static" fields should be "synchronized"

Analyze your code

Code Smell | Critical | multi-threading

In a multi-threaded situation, un-`synchronized` lazy initialization of static fields could mean that a second thread has access to a half-initialized object while the first thread is still creating it. Allowing such access could cause serious bugs. Instead. the initialization block should be `synchronized`.

Similarly, updates of such fields should also be `synchronized`.

This rule raises an issue whenever a lazy static initialization is done on a class with at least one `synchronized` method, indicating intended usage in multi-threaded applications.

Noncompliant Code Example

```
private static Properties fPreferences = null;

private static Properties getPreferences() {
        if (fPreferences == null) {
            fPreferences = new Properties(); // Noncompliant
            fPreferences.put("loading", "true");
            fPreferences.put("filterstack", "true");
            readPreferences();
        }
        return fPreferences;
    }
}
```

Compliant Solution

```
private static Properties fPreferences = null;

private static synchronized Properties getPreferences() {
        if (fPreferences == null) {
            fPreferences = new Properties();
            fPreferences.put("loading", "true");
            fPreferences.put("filterstack", "true");
            readPreferences();
        }
        return fPreferences;
    }
}
```

Available In:

sonarlint | sonarcloud | sonarqube

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