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...

classes should be moved to those classes
Code Smell

"enum" fields should not be publicly mutable
Code Smell

Abstract methods should not be redundant
Code Smell

Arrays should not be copied using loops
Code Smell

Static non-final field names should comply with a naming convention
Code Smell

JUnit rules should be used
Code Smell

Nested "enum"s should not be declared static
Code Smell

"catch" clauses should do more than rethrow
Code Smell

Mutable fields should not be "public static"
Code Smell

The diamond operator ("<>") should be used
Code Smell

"finalize" should not set fields to "null"
Code Smell

Subclasses that add fields should override "equals"
Code Smell

"equals" method overrides should accept "Object" parameters

Analyze your code

Bug | Major | suspicious

"equals" as a method name should be used exclusively to override `Object.equals(Object)` to prevent any confusion.

It is tempting to overload the method to take a specific class instead of `Object` as parameter, to save the class comparison check. However, this will not work as expected when that is the only override.

Noncompliant Code Example

```
class MyClass {
  private int foo = 1;

  public boolean equals(MyClass o) {  // Noncompliant; does
    return o != null && o.foo == this.foo;
  }

  public static void main(String[] args) {
    MyClass o1 = new MyClass();
    Object o2 = new MyClass();
    System.out.println(o1.equals(o2));  // Prints "false" be
  }
}

class MyClass2 {
  public boolean equals(MyClass2 o) {  // Ignored; `boolean
    //..
  }

  public boolean equals(Object o) {
    //...
  }
}
```

Compliant Solution

```
class MyClass {
  private int foo = 1;

  @Override
  public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
      return false;
    }

    MyClass other = (MyClass)o;
    return this.foo == other.foo;
  }

  /* ... */
}

class MyClass2 {
  public boolean equals(MyClass2 o) {
    //..
  }

  public boolean equals(Object o) {
    //...
  }
}
```

Available In:

sonarlint sonarcloud sonarqube

Catches should be combined

Code Smell

Methods of "Random" that return floating point values should not be used in random integer generation

Code Smell

Parsing should be used to convert "Strings" to primitives

Code Smell

Classes should not be empty

Code Smell