NUM07-J. Do not attempt comparisons with NaN

According to *The Java Language Specification (JLS)*, §4.2.3, "Floating-Point Types, Formats, and Values" [JLS 2015]:

NaN (not-a-number) is unordered, so the numerical comparison operators <, <=, >, and >= return false if either or both operands are NaN. The equality operator == returns false if either operand is NaN, and the inequality operator != returns true if either operand is NaN.

Because this unordered property is often unexpected, direct comparisons with NaN must not be performed. Problems can arise when programmers write code that compares floating-point values without considering the semantics of NaN. For example, input validation checks that fail to consider the possibility of a NaN value as input can produce unexpected results (see NUM08-J. Check floating-point inputs for exceptional values for additional information).

Noncompliant Code Example

This noncompliant code example attempts a direct comparison with NaN. In accordance with the semantics of NaN, all comparisons with NaN yield false (with the exception of the != operator, which returns true). Consequently, this comparison always return false, and the "result is NaN" message is never printed.

```java
public class NaNComparison {
    public static void main(String[] args) {
        double x = 0.0;
        double result = Math.cos(1/x); // Returns NaN if input is infinity
        if (result == Double.NaN) { // Comparison is always false!
            System.out.println("result is NaN");
        }
    }
}
```

Compliant Solution

This compliant solution uses the method `Double.isNaN()` to check whether the expression corresponds to a NaN value:

```java
public class NaNComparison {
    public static void main(String[] args) {
        double x = 0.0;
        double result = Math.cos(1/x); // Returns NaN when input is infinity
        if (Double.isNaN(result)) {
            System.out.println("result is NaN");
        }
    }
}
```

Risk Assessment

Comparisons with NaN values can lead to unexpected results.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM07-J</td>
<td>Low</td>
<td>Probable</td>
<td>Medium</td>
<td>P4</td>
<td>L3</td>
</tr>
</tbody>
</table>

Automated Detection

Automated detection of comparison with NaN is straightforward. Sound determination of whether the possibility of an unordered result has been correctly handled is not feasible in the general case. Heuristic checks could be useful.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeSonar</td>
<td></td>
<td>PMD.Design.BadComparison</td>
<td>Bad comparison</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FB.CORRECTNESS.FE_TEST_IF_EQUAL_TO_NOT_A_NUMBER</td>
<td>Doomed test for equality to NaN</td>
</tr>
<tr>
<td>Parasoft Jtest</td>
<td>10.3</td>
<td>PB.NUM.NAN</td>
<td>Implemented</td>
</tr>
<tr>
<td>Reference</td>
<td>Note</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FindBugs 2008</td>
<td>FE: Doomed test for equality to NaN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JLS 2015</td>
<td>§4.2.3, &quot;Floating-Point Types, Formats, and Values&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seacord 2015</td>
<td>NUM07-J. Do not attempt comparisons with NaN LiveLesson</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>