ERR54-J. Use a try-with-resources statement to safely handle closeable resources

The Java Development Kit 1.7 (JDK 1.7) introduced the try-with-resources statement (see the JLS, §14.20.3, "try-with-resources" [JLS 2013]), which simplifies correct use of resources that implement the `java.lang.AutoCloseable` interface, including those that implement the `java.io.Closeable` interface.

Using the try-with-resources statement avoids problems that can arise when closing resources with an ordinary `try-catch-finally` block, such as failing to close a resource because an exception is thrown as a result of closing another resource, or masking an important exception when a resource is closed.

Use of the try-with-resources statement is also illustrated in ERR05-J. Do not let checked exceptions escape from a finally block, FIO03-J. Remove temporary files before termination, and FIO04-J. Release resources when they are no longer needed.

Noncompliant Code Example

This noncompliant code example uses an ordinary `try-catch-finally` block in an attempt to close two resources.

```java
public void processFile(String inPath, String outPath) throws IOException{
    BufferedReader br = null;
    BufferedWriter bw = null;
    try {
        br = new BufferedReader(new FileReader(inPath));
        bw = new BufferedWriter(new FileWriter(outPath));
        // Process the input and produce the output
    } finally {
        try {
            if (br != null) {
                br.close();
            }
            if (bw != null) {
                bw.close();
            }
        } catch (IOException x) {
        // Handle error
        }
    }
}
```

However, if an exception is thrown when the `BufferedReader br` is closed, then the `BufferedWriter bw` will not be closed.

Compliant Solution (finally block)

This compliant solution uses a second `finally` block to guarantee that `bw` is properly closed even when an exception is thrown while closing `br`.
```java
public void processFile(String inPath, String outPath)
    throws IOException {
    BufferedReader br = null;
    BufferedWriter bw = null;
    try {
        br = new BufferedReader(new FileReader(inPath));
        bw = new BufferedWriter(new FileWriter(outPath));
        // Process the input and produce the output
    } finally {
        if (br != null) {
            try {
                br.close();
            } catch (IOException x) {
                // Handle error
            } finally {
                if (bw != null) {
                    try {
                        bw.close();
                    } catch (IOException x) {
                        // Handle error
                    }
                }
            }
        }
    }
}
```

**Compliant Solution (try-with-resources)**

This compliant solution uses a try-with-resources statement to manage both `br` and `bw`.

```java
public void processFile(String inPath, String outPath)
    throws IOException {
    try (BufferedReader br = new BufferedReader(new FileReader(inPath));
        BufferedWriter bw = new BufferedWriter(new FileWriter(outPath))) {
        // Process the input and produce the output
    } catch (IOException ex) {
        System.err.println("thrown exception: " + ex.toString());
        Throwable[] suppressed = ex.getSuppressed();
        for (int i = 0; i < suppressed.length; i++) {
            System.err.println("suppressed exception: " + suppressed[i].toString());
        }
    }
}
```

This solution preserves any exceptions thrown during the processing of the input while still guaranteeing that both `br` and `bw` are properly closed, regardless of what exceptions occur. Finally, this code demonstrates how to access every exception that may be produced from the try-with-resources block.

If only one exception is thrown, either during opening, processing, or closing of the files, the exception will be printed after "thrown exception:". If an exception is thrown during processing, and a second exception is thrown while trying to close either file, the first exception will be printed after "thrown exception:" and the second exception will be printed after "suppressed exception:".

**Applicability**

Failing to correctly handle all failure cases when working with closeable resources may result in some resources not being closed or in important exceptions being masked, possibly resulting in a denial of service. Note that failure to use a try-with-resources statement cannot be considered a security vulnerability in and of itself because it is possible to write a correctly structured group of nested try-catch-finally blocks guarding the resources that are in use (see [ERR05-S]. Do not let checked exceptions escape from a finally block). That said, failure to correctly handle such error cases is a common source of vulnerabilities. Use of a try-with-resources statement mitigates this issue by guaranteeing that the resources are managed correctly and that exceptions are never masked.

**Automated Detection**
### Bibliography

| [JLS 2013] | §14.20.3, "try-with-resources" |
| [Tutorials 2013] | The `try-with-resources` Statement |