ERR03-J. Restore prior object state on method failure

Objects in general should—and security-critical objects *must*—be maintained in a consistent state even when exceptional conditions arise. Common techniques for maintaining object consistency include

- Input validation (on method arguments, for example)
- Reordering logic so that code that can result in the exceptional condition executes before the object is modified
- Using rollbacks in the event of failure
- Performing required operations on a temporary copy of the object and committing changes to the original object only after their successful completion
- Avoiding the need to modify the object at all

Noncompliant Code Example

This noncompliant code example shows a `Dimensions` class that contains three internal attributes: the `length`, `width`, and `height` of a rectangular box. The `getVolumePackage()` method is designed to return the total volume required to hold the box after accounting for packaging material, which adds 2 units to the dimensions of each side. Nonpositive values of the dimensions of the box (exclusive of packaging material) are rejected during input validation. No dimension can be larger than 10. Also, the `weight` of the object is passed in as an argument and cannot be more than 20 units.

If the `weight` is more than 20 units, it causes an `IllegalArgumentException`, which is intercepted by the custom error reporter. Although the logic restores the object's original state in the absence of this exception, the rollback code fails to execute in the event of an exception. Consequently, subsequent invocations of `getVolumePackage()` produce incorrect results.

```
class Dimensions {
  private int length;
  private int width;
  private int height;
  static public final int PADDING = 2;
  static public final int MAX_DIMENSION = 10;

  public Dimensions(int length, int width, int height) {
    this.length = length;
    this.width = width;
    this.height = height;
  }

  protected int getVolumePackage(int weight) {
    length += PADDING;
    width  += PADDING;
    height += PADDING;
    try {
      if (length <= PADDING || width <= PADDING || height <= PADDING ||
        length > MAX_DIMENSION + PADDING || width > MAX_DIMENSION + PADDING ||
        height > MAX_DIMENSION + PADDING || weight <= 0 || weight > 20) {
        throw new IllegalArgumentException();
      }

      int volume = length * width * height;
      length -= PADDING; width -= PADDING; height -= PADDING; // Revert
      return volume;
    } catch (Throwable t) {
      MyExceptionReporter mer = new MyExceptionReporter();
      mer.report(t); // Sanitize
      return -1; // Non-positive error code
    }
  }

  public static void main(String[] args) {
    Dimensions d = new Dimensions(8, 8, 8);
    System.out.println(d.getVolumePackage(21)); // Prints -1 (error)
    System.out.println(d.getVolumePackage(19));
    // Prints 1728 (12x12x12) instead of 1000 (10x10x10)
  }
}
```

The `catch` clause is permitted by exception ERR08-J-EX0 in ERR08-J. Do not catch NullPointerException or any of its ancestors because it serves as a general filter passing exceptions to the `MyExceptionReporter` class, which is dedicated to safely reporting exceptions as recommended by ERR00-J. Do not suppress or ignore checked exceptions. Although this code only throws `IllegalArgumentException`, the `catch` clause is general enough to handle any exception in case the `try` block should be modified to throw other exceptions.

Compliant Solution (Rollback)

This compliant solution replaces the `catch` block in the `getVolumePackage()` method with code that restores prior object state in the event of an exception:

```
  // ...

  } catch (Throwable t) {
    MyExceptionReporter mer = new MyExceptionReporter();
    mer.report(t); // Sanitize
    length -= PADDING; width -= PADDING; height -= PADDING; // Revert
    return -1;
  }
```

Compliant Solution (`finally` Clause)

This compliant solution uses a `finally` clause to perform rollback, guaranteeing that rollback occurs whether or not an error occurs:

```
protected int getVolumePackage(int weight) {
  length += PADDING;
  width  += PADDING;
  height += PADDING;
  try {
    if (length <= PADDING || width <= PADDING || height <= PADDING ||
      length > MAX_DIMENSION + PADDING ||
      width > MAX_DIMENSION + PADDING ||
      height > MAX_DIMENSION + PADDING ||
      weight <= 0 || weight > 20) {
      throw new IllegalArgumentException();
    }

    int volume = length * width * height;
    return volume;
  } catch (Throwable t) {
    MyExceptionReporter mer = new MyExceptionReporter();
    mer.report(t); // Sanitize
    return -1; // Non-positive error code
  } finally {
    // Revert
    length -= PADDING; width -= PADDING; height -= PADDING;
  }
}
```

Compliant Solution (Input Validation)

This compliant solution improves on the previous solution by performing input validation before modifying the state of the object. Note that the `try` block contains only those statements that could throw the exception; all others have been moved outside the `try` block.

```
protected int getVolumePackage(int weight) {
  try {
    if (length <= 0 || width <= 0 || height <= 0 ||
        length > MAX_DIMENSION || width > MAX_DIMENSION || height > MAX_DIMENSION ||
        weight <= 0 || weight > 20) {
      throw new IllegalArgumentException(); // Validate first
    }
  } catch (Throwable t) { MyExceptionReporter mer = new MyExceptionReporter();
    mer.report(t); // Sanitize
    return -1;
  }

  length += PADDING;
  width  += PADDING;
  height += PADDING;

  int volume = length * width * height;
  length -= PADDING; width -= PADDING; height -= PADDING;
  return volume;
}
```

Compliant Solution (Unmodified Object)

This compliant solution avoids the need to modify the object. The object's state cannot be made inconsistent, and rollback is consequently unnecessary. This approach is preferred to solutions that modify the object but may be infeasible for complex code.

```
protected int getVolumePackage(int weight) {
  try {
    if (length <= 0 || width <= 0 || height <= 0 ||
        length > MAX_DIMENSION || width > MAX_DIMENSION ||
        height > MAX_DIMENSION || weight <= 0 || weight > 20) {
      throw new IllegalArgumentException(); // Validate first
    }
  } catch (Throwable t) {
    MyExceptionReporter mer = new MyExceptionReporter();
    mer.report(t); // Sanitize
    return -1;
  }

  int volume = (length + PADDING) * (width + PADDING) *
               (height + PADDING);
  return volume;
}
```

Risk Assessment

Failure to restore prior object state on method failure can leave the object in an inconsistent state and can violate required state invariants.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
ERR03-J	Low	Probable	High	**P2**	**L3**

Related Vulnerabilities

CVE-2008-0002 describes a vulnerability in several versions of Apache Tomcat. If an exception occurs during parameter processing, the program can be left in the context of the wrong request, which might allow remote attackers to obtain sensitive information. An exception can be triggered by disconnecting from Tomcat during this processing.

Related Guidelines

MITRE CWE	CWE-460, Improper Cleanup on Thrown Exception

Bibliography

[Bloch 2008]	Item 64, "Strive for Failure Atomicity"