Violation 1

Specification: Iterator_HasNext

Reason for Inspection

Besides flag, It was mentioned in the paper that iterator_hasnext was most violated as well as least effective at bug finding. So, most likely it was false alarm.

True Bug or False Alarm?
False Alarm.

If True Bug, How to Fix?
N/A, false alarm.

If False Alarm, Why Did It Occur?
The test method is in controlled environment where the the expected entries is guaranteed. So, calling getNextEntry() without hasNext() check is safe in this context. The specification flagged this usage pattern without recognizing the controlled environment.

Violation 2

Specification: InputStream_ManipulateAfterClose

Reason for Inspection
The monitor reported that ArArchiveInputStreamTest.cantReadAfterClose (line 135) manipulates an InputStream(ArArchiveInputStream) after it has been closed.

True Bug or False Alarm?
False Alarm.

If True Bug, How to Fix?
N/A, as it is intentional.

If False Alarm, Why Did It Occur?
This unit test wants to trigger reading from the stream after close() to verify that the library correctly throws IllegalStateException. So, the behavior the spec flags (“manipulate after close”) is the test’s purpose. The test is a valid check of library behavior under error conditions.

Violation 3

Specification: ByteArrayOutputStream_FlushBeforeRetrieve

Reason for Inspection
A violation was reported at TarArchiveOutputStreamTest.testWriteNonAsciiLinkPathNamePaxHeader (line 368), asserting that the code retrieves the byte contents of a ByteArrayOutputStream before explicitly calling flush().

True Bug or False Alarm?
False Alarm.

If True Bug, How to Fix?
N/A, not a real bug.

If False Alarm, Why Did It Occur?
ByteArrayOutputStream does not strictly require flush() call before toByteArray(). The entire content is already in memory. The test code calls tos.close() on the TarArchiveOutputStream, which should do all writes to the underlying ByteArrayOutputStream. Retrieved bytes (bos.toByteArray()) reflect the data so there is no actual bug. The spec's violation here is a false alarm because it is very strict about always flushing before retrieving which is not necessary.

Violation 4

Specification: OutputStream_ManipulateAfterClose

Reason for Inspection
The runtime verification tool flagged GzipCompressorOutputStream.write (line 135) in GzipCompressorOutputStream.java for manipulating an OutputStream after it has been closed. The violation suggests that a write operation is being attempted on the GzipCompressorOutputStream after the stream has already been closed.

True Bug or False Alarm?
False Alarm.

If True Bug, How to Fix?
N/A, false alarm.

If False Alarm, Why Did It Occur?
The GzipCompressorOutputStream class is designed to prevent manipulation of the underlying OutputStream after it is closed. The implementation ensures that any write operations after closure throw IOException which prevents misuse. So, the flagged write() call is safely handled by the class.

Violation 5

Specification: Closeable_MeaninglessClose

Reason for Inspection
The runtime verification tool flagged the method ByteUtilsTest.toLittleEndianToConsumerUnsignedInt32 at line 174 in ByteUtilsTest.java for invoking the close() method on a ByteArrayOutputStream. The Closeable_MeaninglessClose specification monitors instances where the close() method is called on streams that do not require closure.

True Bug or False Alarm?
False Alarm.

If True Bug, How to Fix?
N/A, false alarm.

If False Alarm, Why Did It Occur?
ByteArrayOutputStream operates in memory and does not hold external resources that need closure. So, calling close() on it has no effect. The specification likely flagged this as an issue without recognizing that ByteArrayOutputStream safely ignores redundant close() calls.

Violation 6

Specification: Closeable_MultipleClose

Reason for Inspection
The runtime verification tool flagged the method TarTestCase.testTarFileExplicitDirectoryEntry at line 300 in TarTestCase.java for invoking the close() method multiple times on a Closeable object. The violation suggests that the TarArchiveOutputStream is being closed explicitly within a try-with-resources block.

True Bug or False Alarm?
False Alarm.

If True Bug, How to Fix?
N/A, false alarm.

If False Alarm, Why Did It Occur?
ByteArrayOutputStream does not require explicit closure as it operates entirely in memory. Calling close() on it is redundant and has no effect.