Activity 2 - Code Review & Fixes Summary

Task 1: EqualityTest.java

Violation: Used `.equals()` to compare array contents.

CERT Rule: EXP01-J - Do not use == or equals() to compare the contents of arrays.

Fix: Replaced `.equals()` with `Arrays.equals(...)` to compare actual array values.

Summary: In this task, the original code incorrectly used the `.equals()` method to compare two arrays, which checks for reference equality instead of comparing contents. This led to incorrect output. I corrected this by using `Arrays.equals()`, which compares array elements properly, following the EXP01-J guideline to ensure accurate behavior.

Task 2: FileTest.java

Violation: Missing input validation, unsafe file handling, and no error management.

CERT Rules:

- FIO01-J: Close resources when no longer needed.
- FIO04-J: Release resources when an exception is thrown.
- ERR07-J: Do not allow exceptions to propagate unchecked.

Fix: Added args check, used try-with-resources for BufferedReader, and handled IOException.

Summary: The original file reader code lacked basic input validation and error handling. It assumed command-line arguments were present and did not properly close file resources. I applied a fix by checking for argument presence, wrapping file operations in a try-with-resources block to auto-close, and catching IOException for better reliability and user feedback. This aligns with several FIO and ERR CERT rules.

Task 3: task3_process_integers.c

Violation: Used `atoi()` for conversion, no overflow check, and improper input handling.

CERT Rules:

- INT30-C: Ensure unsigned integer operations do not wrap.
- MSC30-C: Do not use `atoi()` for numeric conversions.
- ERR33-C: Detect and handle standard library errors.

Fix: Used `strtoul()` with error checking, validated inputs, and prevented integer overflow.

Summary: The original C code used `atoi()` to convert string arguments to integers, which lacks error reporting and validation. This introduced the risk of silent overflows and undefined behavior. I fixed the code by using `strtoul()` to safely parse inputs with error checking, and added a condition to detect and avoid overflow using `UINT_MAX`. This ensures compliance with the INT30-C and MSC30-C guidelines.