Mark Eilers

CS 405: Secure Coding

4-2 Milestone: Unit Testing

March 30, 2024

4-2 Milestone: Unit Testing

During the static testing process, the unit test names were carefully chosen to accurately reflect the conditions being evaluated within the code. Each test was designed to address a specific aspect of the collection, ensuring that the functionality was thoroughly assessed. For instance, tests were named to check if the collection was empty upon creation, if elements were added correctly, if resizing operations behaved as expected, and if exceptions were appropriately handled. This approach ensured that the purpose and intent of each test were clearly communicated and understood.

All 13 unit tests were successfully implemented as part of the Google Test fixture, utilizing the ASSERT and EXPECT functionality to validate specific conditions. Assertions were strategically placed throughout the code to verify the expected behavior of the collection under various scenarios. For example, assertions were used to confirm the non-null status of the collection, its emptiness upon creation, the correctness of element addition, the effectiveness of resizing operations, and the handling of exceptions.

To demonstrate robustness, two of the unit tests were intentionally designed as negative tests, aiming to capture appropriate outcomes when the code encountered expected negative scenarios. These negative tests focused on situations such as attempting to reserve a negative capacity and accessing elements beyond the collection's bounds, with specific exceptions expected to be thrown. This approach ensured that the code was capable of handling error conditions gracefully and provided valuable insights into its resilience.

Throughout the development process, adherence to industry-standard best practices was maintained. The code followed clear and descriptive naming conventions for variables and functions, facilitating readability and comprehension. Additionally, consistent formatting and inline comments were utilized to enhance code clarity and maintainability. By organizing tests into a fixture class and employing helper functions for common operations, a software design pattern approach was exemplified, promoting modularity and reusability.

During debugging, various issues were identified and systematically addressed. These included incorrect test conditions, missing assertions, and logic errors within the code. By reviewing each test case meticulously and ensuring alignment between expected and actual behavior, the root causes of these issues were pinpointed and resolved. Specific types of bugs, such as incorrect comparison operators, missing null pointer checks, and flawed loop conditions, were identified and rectified through targeted adjustments and corrections. Overall, the debugging process was comprehensive and methodical, resulting in a robust and reliable codebase.

\*\*\* Please note: There are 3 screenshots below. The first screenshot is the successful use and build of the original program without the ToDo completed. The last two screenshots show the results of the testing.

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated