Lab - 5: Exceptions and Assertions

Team Members:

Jay Sanjaybhai Patel (jy451478@dal.ca)

Kenil Kevadiya (kn486501@dal.ca)

Date:

February 15, 2024

Subject:

Software Development Concepts

Professor:

Michael McAllister

1. We usually want you to re-use existing code and infrastructure whenever possible. Why might you create your own exception?

- If any of the existing exceptions do not satisfy a particular requirement or do not give valuable details, we may create our own exception.
- We have created the MaximumRecursionDepth exception as part of the Fibonacci code. This custom exception allows us to include custom variables such as message and depth, as well as custom methods such as getMessage() and getDepth(), which improves the exception's ability to convey important message.
- By creating our own exceptions, such as MaximumRecursionDepth, we make sure a clear and descriptive message is thrown when the recursion depth exceeds the specified limit. This improves code readability and maintainability by allowing programmers to immediately understand the cause of the mistake and take the appropriate action to fix it.

2. We added parameters to the Fibonacci method. However, those parameters aren't very meaningful to a general user. What would you do to the code to make it more accessible for a general user?

- To make the Fibonacci method more accessible for a general user, we remove the parameters that are used in the internal workings of the method and instead provide a simpler method that only accepts the number for which the Fibonacci number is desired.
- With this modification, users can now simply call getFibonacciNumber(n) to get the nth Fibonacci number without needing to worry about internal parameters like maxLevel or currLevel.
- The method calculateFibonacciNumber is made private to encapsulate the recursion logic, and the wrapper method getFibonacciNumber provides a clean interface for users.

3. How would you recommend for someone to develop a loop invariant?

Below are the steps to develop loop invariant:

- Determine the loop's objective and the variables that are used in the computation.
- If at all possible, convert the loop into a standard form, such as a while or for loop.
- Handle a few manual loop iterations to see how the variables change.
- Look for a relationship between the variables or a formula that holds true both before and after each iteration.
- Verify that the invariant includes the correctness of the loop and has significance.

- Validate the invariant which shows that it is true before the first iteration and remains true throughout each iteration.
- To guarantee the accuracy and dependability of loops in software applications, developers might adopt a procedure-based strategy by generating, confirming, and keeping records of loop invariants.

4. How can loop invariants help you in programming, even if you don’t include them directly as assertions in your code?

- During debugging, for or while loops often require us to check the flow in which the loop occurs, and for that, we add many print statements.
- Moreover, sometimes we also have to check on which iteration our loop breaks.
- We can do this with those print statements, but after completing debugging, we need to remove them without fail.
- Instead, we can use assert identifiers so that we don't have to remove them because by default, they are disabled.
- Loop invariants can provide insights into opportunities for optimization.
- Furthermore, loop invariants can also serve as documentation for the code, aiding in its maintenance. They act as important assumptions about the loop's behavior, making it easier for other developers.
- Loop invariants are extremely helpful in assuring accuracy, understanding code, and avoiding accidental breakage even if they aren't included directly as assertions in the code.