CSCI 3901 Winter 2021

Lab 4: Exceptions and Assertions

Due Friday Feb. 5th, 08:30 AST on Brightspace

Objective

In this lab, you will practice using exceptions and assertions.

Working alone or in a group of 2, you will create two small programs that demonstrate how to use exceptions and assertions.

Preparation

- Review the discussion on exceptions from class.
- Review the recursive code for calculating the **Fibonacci** numbers from the debugging lab.
- Prepare pseudocode to implement a non-recursive **binary search** on an array of integers.

Procedure

Set-up

1. Get the recursive code for calculating the Fibonacci numbers working in a project.

Lab steps

Part 1 - Exceptions

1. Revise the Fibonacci code as follows to be able to limit the depth of the recursion:
   - Add two parameters to the method: one to say the maximum number of levels of recursion should be allowed and the second to say how deep you are now in the recursion
   - When the code calls itself recursively, increment the parameter on how deep you are into the recursion
2. Create a custom-made exception called `MaximumRecursionDepth` that extends the `RuntimeException` class. The new exception should include a `String` message and an `int` that is the depth of recursion reached. Ensure that your custom-made exception has two methods available:
   - `getMessage()` to return the string message
   - `getDepth()` to return the integer depth that was reached at the time of the exception
3. Modify your Fibonacci code to throw a `MaximumRecursionDepth` exception when the recursion depth exceeds the value of the parameter for the maximum number of levels of recursion.

Part 2 - Assertions

1. Write (non-recursive) code to perform a binary search on a sorted array. You can get help from the Internet for this part; just be sure to cite any sources that you use. This step is not the critical one of the lab.
2. Add the following assertions:
   - a loop precondition
   - a loop invariant
   - a postcondition
3. Ensure that your assertions are working.

Questions

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

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?

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

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

Reporting

1. In one file, list
   - The members of your team.
   - The answers from the Questions section of the lab.
2. Generate a PDF from the document.
3. Submit the PDF in Brightspace and all your Java files in Brightspace in the Lab/Lab 4 section.

Assessment

The assessment will be on a letter grade and will reflect how well you implemented the exceptions and assertions, how much you re-used existing code, and how well you demonstrate that you can take the work of this lab and apply it to another situation (the questions section).