Aspect-Oriented Programming

Overview

In this assignment, you will:

- Gain experience using Aspect-Oriented Programming
- Create an aspect for recording when a method is called.
- Create an aspect to improve the performance of a method.
- Keep track of your progress using version control.
- Use various software engineering tools to help create quality software (static and style analysis, memory leak checking, continuous integration)

Instructions

Setup

1. Fork the assignment repository so you have your own copy to work on and submit.
2. Confirm the following settings:
    a. Project visibility for your forked repository to "Private" (*Settings->General*).
        i. This should be automatic.
    b. The *Git Strategy* is set to "`git clone`". (*Settings->CI/CD->General Pipline*).
        i. You will have to change this when you create your fork.

Completing the Assignment

1. Create a local clone of your assignment repository.
    a. Run the command `git remote` and verify that there is a remote called `origin`.
        i. `origin` is the link to your repository of GitLab and is where you will be pushing your changes.
2. Build and run the project.
    a. The build will fail as the unit test will not pass.
3. Open the project in `Atom` (or whatever IDE you choose to use).

Complete the Application

4. Implement the `fibonacci` method as specified in the header file.
    a. The N-th Fibonacci number is defined as:

    `fibonacci(N) = fibonacci(N-1) + fibonacci(N-2)`

    where `fibonacci(0) = 1` and `fibonacci(1) = 1`

b. A unit test is provided to help you test your method.

c. The executable takes as an argument the number N. For example:

   i. `fibonacci 4` will output `The 4th Fibonacci number is: 3`

   ii. `fibonacci 8` will output `The 8th Fibonacci number is: 21`

Profiler Aspect

5. Write an aspect that counts the number of times the `fibonacci` method is called.

   a. Write the aspect in the `profiler.ah` file. The aspect is to:

      i. Keep track of the total number of times the method is called during one program execution.

      ii. Output the number of times the method was called.

   b. `make profiler` will create an executable with the aspect woven in.

   c. An example output is the following. The first output line is from the application itself and the second output line is from the aspect.

```
33  $ fibonacci-profile 20
34  The 20th Fibonacci number is: 6765
35  Called 21891 times.
```

Caching Aspect

As shown by the output of the previous aspect, the `fibonacci` method will get called a lot, even for small values of N (e.g. `N=8` will call `fibonacci` 67 times). By caching returned values and checking the cache before calling the `fibonacci` method, one can significantly reduce the number of method calls.

6. Write an aspect that caches the return value from calling the `fibonacci` method.

   a. Write the aspect in the `cacher.ah` file. The aspect is to:

      i. Intercept a call to `fibonacci`

      ii. Check the value of the argument passed.

      iii. If the value has already been computed, return that value.

      iv. If the value has not yet been computed, allow the execution of `fibonacci` to continue and store the returned value.

   b. `make cacher` will create an executable with both the `profiler` and `cacher` aspects woven in.

   c. An example output is the following. An example output is the following. The first output line is from the application itself and the second output line is from the aspect.

```
33  $ fibonacci-cache 20
34  The 20th Fibonacci number is: 6765
35  Called 21 times.
```

Notes

- You will likely find that the challenging part of this assignment is understanding the AOP concepts and how to apply them. The actual code is not large (i.e. total code for the two aspects should be on the order of 20-30 LOC). If you find that you are writing a lot of code for the aspects, then you are likely going down the wrong path. Meet with the instructor during office hours (or make an appointment) so you don't waste a lot of your time.
- A `Makefile` is provided which:
  - Build an executable (`make fibonacci`)
  - Builds a testing executable (`make fibonacci-test`)
  - Build an executable with the profiler aspect woven in (`make profiler`)
  - Build an executable with the profiler and cacher aspects woven in (`make cacher`)
  - Checks code coverage (`make coverage`)
  - Checks for memory leaks (`make memcheck`)
  - Runs static analysis (`make static`)
  - Runs style checking (`make style`)
- A continuous integration configuration file (`.gitlab-ci.yml`) is provided for you. It is not expected that you will need to change this file.

Grading

You will be graded based on your demonstrated understanding of:

1. Completing the application – 10%
   a. Passing the unit test
2. Aspect-oriented Programming - 60%
   a. Profiler aspect – 20%
   b. Cacher aspect – 40%
3. Version control - 10%
   a. Version control history shows an iterative progression in completing the assignment. You are expected to have **a minimum of three new commits** in your repository (i.e. one for phase of the assignment).
4. Good software engineering practices – 20%
   a. No memory leaks
   b. No static analysis issues
   c. No style analysis
   d. 100% code coverage

Submission

There is no need to submit anything, as GitLab tracks links to forks of the assignment repository.

- Ensure that **the markers (`Mark5210`) have access** to your repository. Look under `Project`

`Information -> Members` in the `Group` tab. O**therwise you will receive a 0**. There is a script that should automatically add `mark2720` as a repository member, but it is your responsibility to confirm that it is doing its job.

- Using **a repository that is not a fork of the assignment repository may result in an automatic 0 (zero)** as the marker will not be able to find your project.