Lab 3: Test Driven Development

Objectives

The purpose of this lab is to practice writing unit tests using TDD. We will practice using PHPUnit and some of its features, like assertions, stubs, fixtures, and the test runner.

Secondary objectives are to practice working with PHP, reading official documentation (from PHPUnit) and working in a collaborative environment.

Description of Task

Part 1: Setup

We will be using cyber-dojo.org as a collaborative work environment for the in-class portion of this lab. Before you begin, you may want to install 7zip if you are on Windows. cyber-dojo.org allows you to download files as a .tgz archive, which Windows does not natively support. If you wish, you can also just copy-paste code out of the editor and to a local file on your computer.

Next, download the code from Canvas. This will be our starting code. It is similar to the Lab 2 code, but with a few minor bugs fixed. Refer to Lab 2 for a description of the files.

Now, go to cyber-dojo.org and join the practice using the passcode provided to the class. It will also be posted on Canvas after it is created. If it is your first time joining, click “Join as new avatar”. If you are returning, you can “Join as existing avatar” and join as the avatar you were given. I’m not sure how long these stay online, but it is at least for a few days, so remember your avatar!

Once in the practice, you’ll want to delete the two existing files, Hiker.php and HikerTest.php. Do not delete cyber-dojo.sh! Then, add five files: Main.php, RandomBinaryOracleTest.php, CharacterModelTest.php, ChallengeModelTest.php, and AdventureControllerTest.php After you have your files, copy-paste the code from your downloaded files into the practice.

Once your files are in, click “Test”. It should show 4 tests have passed.

Part 2: Write Unit Tests Together

Together, in cyber-dojo.org, we will begin writing tests for our application. We will start with some of the easier tests, and then work our way to more harder tests. We will be working using the TDD approach described in the textbook chapter 3.7.

Part 4: Finish Unit Tests and Obtain Good Code Coverage

After we are finished writing a few tests together, you will then work on your own. Download your cyber-dojo.org practice or copy-paste the code into an editor to run locally. When running with the Docker container with the commands given in README.txt, it will generate code coverage statistics in HTML files in the coverage/html directory.

Code coverage is an important metric but is not necessarily the key to good tests. For this application, you should strive for nearly 100% code coverage and branch coverage for AdventureController.php. Do not worry about RNG class or the ad-hoc testing at the bottom. Coverage is a good way to identify missing tests, but there are times when it is impractical to get 100% coverage. If you are unsure about your coverage metrics, reach out and I can take a look at the code.

Because we are doing TDD, we have no code to test. We will write tests first, to test requirements, then add code to satisfy the tests. We will be working on an AdventureController. A controller class is a class that performs logic on data classes, also known as models. These requirements should be considered incomplete; through the design of our application via TDD, we will uncover new requirements.

Here are the requirements for AdventureController:

- As a user, the AdventureController should maintain its relevant data as class attributes
  - The controller will maintain a database of ChallengeModel classes that represent an adventure and a character that represents the character going through the challenge.
- As a user, I should be able to add ChallengeModels to the challenge database.
  - Each challenge should contain a unique id. If the ID is already present in the database, it should throw an exception.
  - When adding a challenge the the database, AdventureController should make sure that it contains 3 options for the equal_options set, and 2 options for the other option sets.
  - I should not be able to add more than 5 challenges to an adventure.
  - The ID of the ChallengeModel must not be NULL or "_end".
- As a user, I should be able to remove ChallengeModels from the challenge database, based on its ID.
  - If the ID does not exist in the database, or the database is empty, it should throw an exception.
  - If I pass an ID of NULL, it should clear the entire database.
- As a user, I should be able to validate that the ChallengeModels I added to the database constitute a valid adventure.
  - If the database does not contain a ChallengeModel with the ID of "_start", it should return false.
  - If the database does not contain a ChallengeModel with one of the next challenge ID's of "_end", it should return false.
  - If the database does not contain a path to get from the "_start" challenge model to the "_end" next ID, it should return false.
  - If the database contains a ChallengeModel with a trait that the character does not have, it should return false.
- As a user, I expect the AdventureController to correctly navigate through an adventure from start to finish, using the ID's returned by each challenge as a guide.
  - The AdventureController should correctly run challenges based on their outputted next challenge ID.
  - The AdventureController should allow award 1 brownie points on each challenge success.
  - The AdventureController, on each failure, subtract 3 brownie points (if available) and automatically pass a challenge, or, decrement the characters attribute used in the challenge.
  - After running the adventure, AdventureController should return text of what happened in the story.

Helpful Tips

Review Documentation Provided
I've included helpful resources on Canvas for Docker and PHP. There is also a book, "Practical PHP Testing" on Canvas. If you need references for other topics, let me know so I can help find some. If you've found helpful resources on a particular topic, let me know so I can add it to Canvas for others to use.

Use the Official PHP Docs
The official PHP docs (php.net/manual) are really good and provide plenty of examples. Try to use them instead of tutorial sites or Stack Overflow. It's good practice!

Especially Review PHPUnit Documentation
These are the sections of PHPUnit Documentation I think are most helpful:

- Test Doubles: https://phpunit.readthedocs.io/en/9.5/test-doubles.html
- Fixtures: https://phpunit.readthedocs.io/en/9.5/fixtures.html
- Assertions: https://phpunit.readthedocs.io/en/9.5/assertions.html
- General Overview: https://phpunit.readthedocs.io/en/9.5/writing-tests-for-phpunit.html

Print Statements in PHP
PHP can be a tricky language; it tries to do a lot in the background that can be opaque and frustrating. You may need to print out information to help debug issues. var_dump and print_r are PHP functions that dump variables, arrays, and classes out to the command line:

- var_dump: https://www.php.net/manual/en/function.var-dump.php
- print_r: https://www.php.net/manual/en/function.print-r.php

There May Be Bugs
Because I wrote this code without unit tests, it could be buggy! Actually, it will almost certainly be buggy. If you find a bug, feel free to fix it, and make sure to let me know!

Never Hesitate to Ask Me For Help
I'm always available to help out, either via email, office hours, or in class.

Deliverables

One of the deliverables in this assignment is a demo of your progress in cyber-dojo. This should be done by the end of class and in-person. Excused absences may arrange in advance for a demo during office hours before the next class period, but for a point penalty.

In addition to the demo, submit the following files to Canvas:

- Main.php
- RandomBinaryOracleTest.php
- CharacterModelTest.php
- ChallengeModelTest.php
- AdventureControllerTest.php
- reflection.rtf

Do not zip the files! I will use a script to collect the files and add my Dockerfile to them, so I really appreciate if you submit them separately.

Your reflection must be in rtf format. (Most editors have a save-as option that includes rtf.) Include the following in your reflection:

- What was the most valuable feature of the lab?
- How did you prepare for this lab? What changes are you considering in preparing for your next lab?
- What did you learn from this experience?
- What advice would you give someone who was preparing for this lab for the first time?
- What went well? What didn't go well?

Don't think of the reflection as a question/answer section. This should be well-written paragraph or two that discusses items like those listed above. Provide any additional feedback you want to share, I find these very useful.

Rubric: 100 Points

10 Points (Attended Class)
The student attended class and participated in the lab assignment. They worked (optionally) with a small team to solve the problem, and contributed to the solution.

20 Points (Demo)
The student performed an in-person demo of their progress by the end of the class period. They noted feedback for improvements before submission. If the task is incomplete, they have a path forward to complete the task by the due date.

10 Points (Reflection)
The reflection covers the required topics; what went well, what didn't go well, what was learned. The reflection was written in complete sentences. It features multiple sentences, is coherent, and thoughtful.

60 Points (15 Requirements, 4 points each)
Each of the requirements listed in the document should be tested in a TDD fashion. The tests should not have logic (if statements or loops). The tests should make use of stubs and fixtures where applicable. The tests should have branch coverage on the code. The resulting code must be functional and coded with good style and PHP best practices.