Project Checklist

SE 333
Spring 2021

Overview

The class project is an opportunity for you to incorporate what you have learned, into a single simple but sort-of-realistic project.

All of the production code is provided for you. There may be a few isolated instances where you need to alter the production code but for the most part, your task is to:

1. add tests and other testing artifacts to this project
2. document observations about the project

This project definitely contains defects. Your responsibility is to find them, not fix them. When I grade this work I expect certain tests to fail, therefore you will not get full credit if you do not provide the expected failing test.

Below is a checklist of the tasks to be completed for this project. It is not expected that you can do all these tasks at the moment. As the quarter unfolds, you will learn what you need to know to complete each task.

The application you are testing is a web site that provides a very minimalistic online shopping experience.

Checklist

General items

1. Provide test code that meets the class coding standard
2. Provide properly functioning gradle script to facilitate the items below
3. Provide changes to gradle script to ensure results are included results in the project's output zip file
4. Provide analysis documents where indicated

Specific items to provide in your project

1. Provide **normal Boundary tests** for the the total cost calculation function
    a. provide a spreadsheet that contains:
        i. variable definitions needed for the tests
        ii. boundary values selected for the relevant variables in the test
        iii. test case definitions including expected results (use the week 1 spreadsheet as a guide for this section

iv. In defining boundary values for money, the level of precision should be to the penny.

b. provide junit tests that are implementations of the test cases defined above. These tests can use any test technique you want.

2. Provide **Equivalence partitioning** tests of total cost calculation function
   a. include **Strong / Normal** tests
   b. include **Weak / Robust** tests
   c. extend spreadsheet from 1.a above to include
      i. partition definitions needed for these tests
   d. implement the tests mentioned in 2.a and 2.b as junit parameterized tests
      i. If you used external data files, update the packageDistribution section of build.gradle so that your data files are included in the zip when you execute `gradle clean build`

3. Provide unit tests using **mocks**, of the database interface
   a. evaluate the correctness of the averagePurchase() function, without actually interacting with the database
   b. evaluate whether PurchaseAgent makes the correct call to PurchaseDBO in the save() function. Again do not interact with the actual database.

4. Provide one of the four test types described below:
   a. Functional test using **Cucumber**
      i. Include 3 or more scenarios in 1 feature file
      ii. Provide at least 2 scenarios that use the same backing code

   b. Performance test using **JMeter**
      i. Provide a performance test of at least 1 URL
      ii. Provide summary document of performance results
         1. number of test runs
         2. number of simulated users
         3. min, max, mean and standard deviation of results
         4. Analysis of your findings
            a. describe any failures found while testing
            b. give overall appraisal of performance
      iii. Save your JMeter test (*.JMX) and include it when you upload your project. Suggestion: add it to src/test/resources

   c. UI tests using **Selenium + JUnit**
      i. Make sure each control on the page is manipulated
      ii. evaluate the results of each button click

   d. Mutation tests using **PITest**
      i. add appropriate configuration to your project to produce PITest output

ii. analyze the results and provide 1 to 2 pages of your findings
    1. What do the results say about the quality of your tests?
    2. Make corrections to your tests if needed
    3. Make sure the PITest reports are added to the project output zip file

5. Add **Code Coverage Analysis** to the project
    a. The details for this part of the project are defined in assignment 7. The gist of the task is to add Jacoco coverage analysis to the project and interpret the results. 1 to 2 pages

6. Add **Static Analysis** to the project
    a. The details for this part of the project are defined in assignment 8. The gist of the task is to add PMD static analysis to the project and interpret the results. 1 to 2 pages.

7. Provide a **Readiness appraisal document** for the project quality as a whole.

    The essence of this document is to give a readiness judgement on the product. Please ignore the fact that the application, even if it were perfect in what it is trying to do, it isn't trying to do much. That aside, is the application ready for release? If not, what needs to be done to make it ready?

    The template for this task is the same as used for manual testing in week 1.

Tips

1. The boundary tests and the equivalence tests may be provided in a single test suite if you clearly delineate which tests belong with which checklist item

2. The database used by the application is stored in src/main/resources/purchases.mv.db. If you want to reset the database for any reason you can do so by:
    a. delete the file mentioned above
    b. restart the web application
    c. refresh the page view in the browser