CS 278

Code Review

Name:

Review of the solution to Asgn 2 implemented by Krzysztof Zienkiewicz

Git Repo: https://github.com/krzysztofzienkiewicz/fall2013cs278

Path: Asgn2

1. Provide a 1-3 paragraph overview of the architecture of the code and the design rationale.

This project implements Dropbox that synchronize the files in a local given directory with a directory on a server. The code for this project uses reactor pattern to receive commands created from filewatcher class. The filewatcher class monitors the states of the files in the directory and creates FileEvent as commands. Then the commands are dispatched by the FileEventHandler class to specific functions.

1. Refactoring 1:

@Test

**public** **void** testAddFile() **throws** IOException {

cmd.setOpCode(OpCode.*ADD*);

protocol.addFile(invariantCornelius);

*verify*(transport).publish(*eq*(cmd));

}

@Test

**public** **void** testRemoveFile() {

cmd.setOpCode(OpCode.*REMOVE*);

cmd.setData(**null**);

protocol.removeFile(invariantCornelius);

*verify*(transport).publish(*eq*(cmd));

}

@Test

**public** **void** testUpdateFile() {

cmd.setOpCode(OpCode.*UPDATE*);

protocol.updateFile(invariantCornelius);

*verify*(transport).publish(*eq*(cmd));

}

* 1. In your three test cases, you use three verify method defined in mokito to ensure if the DropboxTransport::publish has been called(without the times of called.) When observing that these testing methods has similar testing logic as well as similar implementation, it might be better to refactor the DropboxProtocol class which combine the three method into one method to realize the three operation in order to test them together.
  2. It is good practice to analysis the logic underlying each method and group similar method and test together. This would reduce the amount of testing workload and avoid mistakes, also it will be beneficial for future unit test if the code has been improved and modified. I propose to add another methods into DropboxProtocol class that dispatches methods based on the cmd to realize combine the work of adding file, updating file and removing file.
  3. Provide a step-by-step description of the suggested refactoring.

1. *Create a method doOperations in the DropboxProctocol class that takes cmds as parameter.*
2. *Dispatch the cmds in the doOperations method.*
3. *Test them all in a single method*
4. Refactoring 2:
   1. Your existing testing code looks clean and well implemented. Besides, there are some other parts of this project should not be hard which you can try to test for this assignment. For instance, I guess you can try to test some more class including Dropbox.class and FileReactor.class and FileStates which provides some core function part for the whole projects and they are necessary to test and need more refactoring for the original code lacking of the testability. Below are two example class I would like to present.
   2. As the aspect of Refactoring the Dropbox class, creating a constructor taking its private member variable as parameters is necessary and really helpful to make the JUnit test easier. In this way, we could simply mock the parameters and passes the mock objects to Dropbox and test this class by using the verify method in mokito to check the calling as well as the times of calling within corresponding methods in these mock objects we create.
   3. As the aspect of the FileStates class, it needs to be tested for the reason that it stands for the status of a file. In order to test this class, we could simply create in all possible cases to cover as many situation as possible such as not valid path, file missing and file permission denied, which we can see if the original methods have the necessary mechanism to check and treat these extreme situation, which will reduce the potential bugs in the code.
   4. a step-by-step description of the suggested refactoring
5. Create constructor in the Dropbox class
6. Create possible cases for file in the testFileStates test.