CSCI/SOFE 3060U Report – Phase 5 – Team Name Working

Ryan Morton - 100485285
Thomas Frantz - 100484424
Alvin Lee- 100484342

Our backend implementation can be broken down into 4 categories.

First, we have the reader/writer classes – ticketsRW, userAccountsRW, transactionR. These classes are responsible for reading and writing to the files. None of them have any parameters that can be played with for testing. For each of them we tested the constructors, any get functions, and we tested that the readers and writers finished their execution. We couldn't check the values using JUnit, however the files that we read and write to are available to look at, and the actions are written to the console so we can verify that way. Full statement coverage, decision coverage and loop coverage is done using the sample files that we provided.

Second, we have the ticket and user classes – ticket and user. These classes are the base object classes representing a single event and a single user. Each of these have several get functions, an update function and a constructor. To test the get functions we just compared the returned value to what we know we should get. For the updates we checked a proper update, and an improper update to ensure decision and statement coverage. There are no loops in either of these classes. For the constructors we checked a proper input, and improper inputs for every possible parameter to again ensure statement and decision coverage.

Third, we have the helper class. The helper class has all the functions that we used to manipulate the data within the users and tickets arrays according to the DailyTransaction log. This is where the bulk of the functionality is done, and where the bulk of the testing is done too. There are 7 total functions, and for each function, and for each of these we have a test covering statement, loop, and decision coverage, giving us 21 tests. For some of them they are the same.

Lastly we have the backEndMain. This class is the main class that puts everything together. We didn't test this with the JUnit tests since it's all reliant on all the other methods. We can check that it works by checking the output of the userInfo file, and by using the console output.

Results:

All of our test cases pass –

helperTest – 21/21

ticketTest – 10/10

ticketsRWTest – 3/3

userTest – 9/9

userAccountRWTest – 3/3

transactionRTest – 3/3

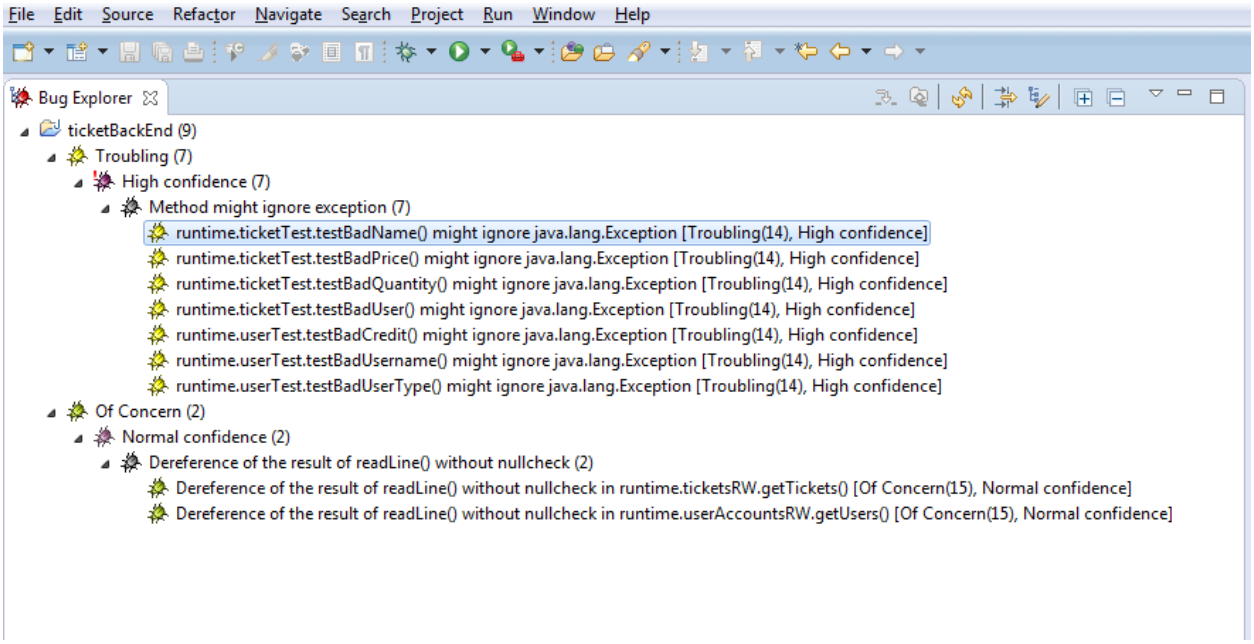
Indication of failures:

Our initial testing did have some failures because we hadn't yet implemented the parameter restrictions on user and ticket. Other than that we had no failures.

Bonus – A list of bugs identified by FindBugs:

FindBugs identified 9 bugs in our code.

7 of them were 'Troubling' , 2 were 'Of Concern'



Bonus – A list of fixes for each bug (or explanation why a bug fix is not needed)

For the 7 'Method might ignore exception' bugs a fix is not needed, since these are our test cases instead of our code, and in these cases we are looking to have the exception thrown and then do nothing with it.

For the other 2 errors there actually is a valid point there. We wrote our code for the reader assuming that the file will end with END, which would be the last line, and then we stop reading after that. In the case where there is no END though then we will get a null pointer exception. To prevent this we added a null check before the END checks. This does add another issue however, since now the user and tickets files could end without the word END and still be valid, which is not how it is stated in the requirements, but that is (probably) better than having a fatal crash.