AuctionBay

Project Phase Five

CSCI3060U, Software Quality Assurance

Miljanovic, M.
Ontario Tech University, Faculty of Science

March 31, 2023

Tuesday Lab (CRN: 72674):

Rija Baig (100746674)
Preet Panchal (100707094)
Eihab Syed (100707448)
Nathaniel Tai (100662284)

AUCTIONBAY

TABLE OF CONTENTS

AUCTIONBAY

1.0. INTRODUCTION

AuctionBay is an Auction-style Sales Service. The system consists of two parts: the Front End, a point of purchase terminal for bidding and advertising items and the Back End, an overnight batch processor to maintain and update a master auctions file intended to run each day at midnight. This is Phase 5 of the project; it includes the completed Backend code for AuctionBay. It also contains a JUnit test suite with both statement coverage and path coverage testing for all the methods and operations. This document will include the testing log table which shows and explains pass and fail in different test cases.

1.1. TESTING SUITE

- **Path Coverage Testing:**
  - For Path Coverage Testing, we chose the ‘create’ operation. The create operation on the backend has one key constraint. This constraint ensures that a newly created user must have a name different from all existing users. We the following two JUnit tests: testCreateUser(), testCreateUserError() and this two tests are able to check both the pass and fail case for any new user creation.
- **Statement Coverage Testing:**
  - For Statement Coverage Testing, we test the rest of the operations. The JUnit tests for statement coverage test for each and every statement in the method for that operation.

2.0. PHASE 5 CHANGES

*All of the following JUnit test cases **PASS** in the Phase 5 Backend code.*

2.1. FINAL LIST OF JUNIT TEST SUITE

MainTest:

01. testMain():
    Testing the initial main function call.

UserTest:

02. testGetUsername():
    Testing the getter for a user's username in the User class.

03. testGetAccountType():
    Testing the getter for a user's account type in the User class.

04. testGetCreditAmount():
    Testing the getter for a user's credit amount in the User class.

05. testSetUsername():
    Testing the setter for a user's username in the User class.

06. testSetAccountType():
    Testing the setter for a user's account type in the User class.

07. testSetCreditAmount():
    Testing the setter for a user's credit amount in the User class.

08. testUpdateCreditAmount():
    Testing update credit for a user's credit amount in the User class.

09. testFormatOutput():
    Testing the output format for User class.

UserManagerTest:

10. testValidateDouble():

Testing if the validate double function is able to convert string to double.

11. testLoad():
    Testing the file reader for current_user_accounts.txt.

12. testCreateUser():
    Testing if the create user function works for unique usernames for the UserManager class.

13. testCreateUserError():
    Testing if the error handling works for the create user function for duplicate usernames for the UserManager class.

14. testDeleteUser():
    Testing if the delete user function works for UserManager class.

15. testRefund():
    Testing if the refund function works for the UserManager class.

16. testAddCredit():
    Testing if the add credit function works for the UserManager class.

17. testCloseAuction():
    Testing if the credits are updated after an item has been auctioned off for UserManager class.

18. testWrite():
    Testing if the output to the current_user_accounts.txt is correct for UserManager class.

ItemTest:

19. testGetItemName():
    Testing the getter for an item's name in the Item class.

20. testGetSellerName():
    Testing the getter for an item's seller in the Item class.

21. testGetBuyerName():
    Testing the getter for an item's buyer in the Item class.

22. testGetHighestBid():
    Testing the getter for an item's highest bid in the Item class.

23. testGetDaysRemaining():
    Testing the getter for an item's remaining days in the Item class.

24. testSetDaysRemaining():
    Testing the setter for an item's remaining days in the Item class.

25. testNewBid():
    Testing the new bid function for an item to verify the new highest bid and buyer in Item class.

26. testFormatOutput():
    Testing if the output to the available_items.txt is correct for Item class.

AuctionTest:

27. testLoad():
    Testing the file reader for available_items.txt.

28. testAdvertise():
    Testing the advertise function to post a new item in the auction for Auction class.

29. testBid():
    Testing the bid function to update an item's highest bid for Auction class.

30. testDelete():
    Testing the delete function to remove an item from the Auction class.

31. testEndyDay():
    Testing the end of day function to decrement the remaining days of an item from the Auction class.

32. testWrite():
    Testing if the output to the available_items.txt is correct for Auction class.

AUCTIONBAY

3.0. PHASE 4 FAILURE TEST LOG TABLE

Test Name (Pass/Fail)	Nature of Failure	Error in Code	Actions to Resolution
MainTest: testMain()	NA	NA	NA
UserTest: testGetUsername()	NA	NA	NA
UserTest: testGetAccountType()	NA	NA	NA
UserTest: testGetCreditAmount()	NA	NA	NA
UserTest: testSetUsername()	NA	NA	NA

UserTest: **testSetAccountType()**	NA	NA	NA
UserTest: **testSetCreditAmount()**	NA	NA	NA
UserTest: **testUpdateCreditAmount()**	NA	NA	NA
UserTest: **testFormatOutput()**	Incorrect format of user information in current_users_accounts.txt file.	Format of writing to user accounts file is wrong.	Removed extra whitespaces and newlines to fix the format and ensure it matches project requirements.
UserManagerTest: **testValidateDouble()**	NA	NA	NA
UserManagerTest: **testLoad()**	NA	NA	NA

UserManagerTest: **testCreateUser()**	Test fails as createUser() function does not create user properly.	Reading incorrect values from current_user_accounts.txt file.	Fix the indexes when splitting the string for each line of the user accounts file so that it is fetching the correct information.
UserManagerTest: **testCreateUserError()**	Test fails as there is no error handling for the project new user constraint.	There is no conditional in the code to check the constraint for new user creations.	Loop through the vector of users and check if a new user created has a username that is different from all existing users.
UserManagerTest: **testDeleteUser()**	Test was failing as it was unable to read the username to delete.	There was no instance of creating a user before user deletion.	Created an instance to create a user and then another instance to delete that user so the test can function correctly.
UserManagerTest: **testRefund()**	Test fails as refund() function does not refund to the correct user properly.	Reading wrong values for seller user and buyer user from current_user_accounts.txt file.	Fix the indexes when splitting the string for each line of the user accounts file so that it is fetching the correct information.
UserManagerTest: **testAddCredit()**	NA	NA	NA
UserManagerTest: **testCloseAuction()**	NA	NA	NA

UserManagerTest: **testWrite()**	Incorrect format of user information displayed in current_users_accounts.txt file.	Format of writing to the user accounts file is wrong.	Removed extra whitespaces and newlines to fix the format and ensure it matches project requirements.
ItemTest: **testGetItemName()**	NA	NA	NA
ItemTest: **testGetSellerName()**	NA	NA	NA
ItemTest: **testGetBuyerName()**	NA	NA	NA
ItemTest: **testGetHighestBid()**	NA	NA	NA
ItemTest: **testGetDaysRemaining()**	NA	NA	NA

ItemTest: **testSetDaysRemaining()**	NA	NA	NA
ItemTest: **testNewBid()**	NA	NA	NA
ItemTest: **testFormatOutput()**	Incorrect format of user information in available_items.txt file.	Format of writing to available items file is wrong.	Removed extra whitespaces and newlines to fix the format and ensure it matches project requirements.
AuctionTest: **testLoad()**	NA	NA	NA
AuctionTest: **testAdvertise()**	Test fails as advertise() function does not create item properly.	Reading incorrect values from available_items.txt file.	Fix the indexes when splitting the string for each line of the item file so that it is fetching the correct information.
AuctionTest: **testBid()**	Test fails as bid() function does not place a bid correctly for an auctioned item.	Updating incorrect numerical values for updated bid.	Fix the splitting string from the available_items.txt and ensure that only the bid amount is being updated.

AuctionTest: **testDelete()**	Test was failing as it was unable to read the item to delete.	There was no instance of creating an item before item deletion.	Created an instance to create an item and then another instance to delete that item so the test can function correctly.
AuctionTest: **testEndDay()**	NA	NA	NA
AuctionTest: **testWrite()**	Incorrect format of user information displayed in available_items.txt file.	Format of writing to the available items file is wrong.	Removed extra whitespaces and newlines to fix the format and ensure it matches project requirements.

4.0. INSTRUCTIONS TO RUN

Application Build: Java version 17.0.2

1) Clone repo or download ZIP folder
2) Open project via IntelliJ IDEA (https://www.jetbrains.com/idea/)

4.1. PROGRAM

1) Go to folder: `AuctionBay/PHASE-5/Backend/src/main`
2) Right-click on **Main.java** and select Run `'Main.main()'`

4.2. JUNIT TESTS

1) Go to folder: `AuctionBay/PHASE-5/Backend/test`
2) Right-click on **'test' folder** and select Run `'All Tests'`