This document describes the tests of the system. Two functions have Loop coverage and Branch coverage, which are described below

Loop Coverage: UserFileStorage.Update

Tests:

- UpdateUser
- UpdateNonExistingUser
- UpdateFinalUser

UpdateUser runs the Update function to update the first user in the Users list, making the loop run once

UpdateNonExistingUser calls update with an empty Users list, making the loop run no times

UpdateFinalUser fills the Users list with generic users, and updates the last one on the list, ensuring the loop runs many times

Branch Coverage: CommandFactory.Create

Tests:

- ReturnAddCredit
- ReturnAdvertise
- ReturnBid
- ReturnCreate
- ReturnDelete
- ReturnRefund
- ReturnNull

Each case checks that one of the conditions are true. Since these conditions are mutually exclusive this also covers the cases that all the others are false. All the conditions are false in ReturnNull, making sure that every conditional statement is run in both a true and false condition.

Class: CommandFactory

Test	Description
ReturnAddCredit	Checks that the CommandFactory returns an AddCredit command when it should
ReturnAdvertise	Checks that the CommandFactory returns an Advertise command when it should
ReturnBid	Checks that the CommandFactory returns a Return command when it should
ReturnCreate	Checks that the CommandFactory returns a Create command when it should
ReturnDelete	Checks that the CommandFactory returns a Delete command when it should
ReturnRefund	Checks that the CommandFactory returns a Refund command when it should
ReturnNull	Checks that the CommandFactory class returns null when there is no matching command

Class: ItemFileStorage

Test	Description
CreateItem	Tests the Create function of ItemFileStorage. The ItemFileStorage is then checked to verify that the Item is actually in storage
DeleteItem	Tests the Delete function of ItemFileStorage. The ItemFileStorage is then checked to verify that the Item is no longer in storage, expecting an ItemNotFoundException
ItemFileNotFound	Verifies that a FileNotFound exception is thrown in Open() when the Items file is not found
ItemNameNotFound	Verifies that an ItemNotFound exception is thrown in Query() when there's no item with the queried name
LoadsItems	Verifies that it loads all the items from a file into storage, comparing the number of elements loaded to a specified count
Query	Tests that Items can be pulled from the ItemFileStorage with the Query() function
SellerNameNotFound	Verifies that an ItemNotFound exception is thrown in Query() when there's no item matching with the queried seller name
UpdateItem	Tests that the itemFileStorage can update existing items. The item in storage is compared with the updated values to verify that the fields have changed
UpdateNonExistingItem	Tests that an ItemNotFoundException is thrown when calling Update with an item that's not already in storage
WritesItems	Tests that the ItemFileStorage can write all items to a file. The number of items in the original file is compared to the number of items in the file which was written to. The output file is deleted at the end of the test

Class: StorageFormatter

Test	Description
PadString	Tests that the StorageFormatter can pad a string to a specified length
PadInt	Tests that the StorageFormatter can pad an integer into a string with the specified length

Class: AddCredit

Test	Description
AddCredit	Runs the AddCredit command and verifies that the users' balance has been updated
AmountBelowMinimum	Makes sure the Validate function will throw an IllegalArgumentException when the amount to add is below 0
AmountOverLimit	Makes sure the Validate function will throw an IllegalArgumentException when the amount to add is above the maximum specified in the functional requirement
UserNotFound	Makes sure a UserNotFoundException is thrown when adding credit to a user which does not exist

Class: Advertise

Test	Description
Advertise	Runs an Advertise command and verifies that the item exists in storage after
DaysOverLimit	Checks that the Validate command of Advertise throws an IllegalArgumentException when the days to auction the item is above the amount specified in the functional specification
ItemAlreadyForSale	Checks that a DuplicateItemException is thrown by the Advertise command when trying to create an item already for sale by the seller
ItemNameTooLong	Checks that an IllegalArgumentException is thrown when the item's name is longer than is specified in the functional specification
PriceAboveLimit	Checks that an IllegalArgumentException is thrown when the item's price is above the maximum amount allowed by Advertise as specified in the functional specification
SellerNotFound	Checks that a UserNotFoundException is thrown when the seller does not exist
SellerTypeNotPermitted	Checks that an IllegalArgumentException is thrown when the seller's user type does not permit it to run the Advertise command

Class: Bid

Test	Description
Bid	Runs the Bid command, then checks that the item's highest bidder's name and highest bid properties have been updated
BidTooLow	Makes sure an IllegalArgumentException is thrown when the bid amount isn't high enough to bid on this object
BuyerBalanceBelowBid	Makes sure an IllegalArgumentException is thrown when the balance of the buyer is below the amount they bid
BuyerInvalidPermission	Makes sure an IllegalArgumentException is thrown when the buyer's user type is not permitted to run the Bid command
BuyerNotFound	Makes sure a UserNotFoundException is thrown when the buyer does not exist

ItemNotFound	Makes sure an ItemNotFoundException is thrown when the item being bid on does not exist

Class: Create

Test	Description
CreateCommandCreditLessThanZeroTest	Makes sure that the Create command throws an exception when creating a user with less than 0 balance
CreateCommandCreditOverLimitTest	Makes sure the Create command throws an exception when the amount of credit is above the allowed amount for Create as specified in the functional specification
CreateCommandInvalidUserTypeTest	Makes sure an exception is thrown when the user's type doesn't match any UserType values
CreateCommandUsernameIsTakenTest	Makes sure an exception is thrown when creating a user with a username which is already taken

Class: DailyLogFile

Test	Description
DailyLogFileInitializeTest	Makes sure the DailyLogFile class can initialize without issue
DailyLogFileIsEmptyTestFalse	Makes sure that IsEmpty returns true when there is no more entries to read
DailyLogFileIsEmptyTestTrue	Makes sure that IsEmpty returns false when there is more entries to read
DailyLogFileNextItemTest	Makes sure that an item can be pulled from the DailyLogFile instance with NextItem. The contents of the LogEntry are verified to make sure the expected output was returned

Class: Delete

Test	Description
DeleteCommandSuccessItemTest	Makes sure that after Executing a Delete command, there is no items for sale from the deleted user
DeleteCommandSuccessUserTest	Makes sure that after Executing a Delete command, the user no longer exists
DeleteCommandUserNotFound	Makes sure that an exception is thrown in Delete.Validate when the user to delete does not exist

Class: Item, User

Test	Description
SetGetItemGetDaysRemainingTest	Tests the accessor and mutator of an item's DaysRemaining property
SetGetItemGetHighestBid	Tests the accessor and mutator of an item's HighestBid property

SetGetItemHighestBidderNameTest	Tests the accessor and mutator of an item's HighestBidderName property
SetGetItemIsOver	Tests the IsOver function of an item, that it returns true when the item's DaysRemaining is 0, false otherwise
SetGetItemNameTest	Tests the accessor and mutator of an item's Name property
SetGetItemSellerNameTest	Tests the accessor and mutator of an item's SellerName property
SetGetUserCreditTest	Tests the accessor and mutator of a user's Credit property
SetGetUserIsAdminTest	Tests the IsAdmin function of a user, that it returns true when the user's type is ADMIN, false otherwise
SetGetUserNameTest	Tests the accessor and mutator of a user's Username property
SetGetUserTypeTest	Tests the accessor and mutator of a user's Type property

Class: LogEntry

Test	Description
LogEntryArgumentsTest	Tests that a LogEntry properly returns its Arguments
LogEntryTransactionCodeTest	Tests that the LogEntry object propertly returns its TransactionCode

Class: Refund

Test	Description
RefundBuyerDoesNotExistTest	Makes sure an Exception is thrown when the buyer does not exist
RefundSellerDoesNotExistTest	Makes sure an Exception is thrown when the seller does not exist
RefundSellerDoesNotHaveCorrectCreditsTest	Makes sure an Exception is thrown when the seller does not have enough credit to refund to a user
RefundSuccessTest	Makes sure that a refund can be successfully Executed

Class: UserFileStorage

Test	Description
CreateUser	Makes sure a user can be created. It is verified that the user exists in storage after the Create call
DeleteUser	Makes sure a user can be deleted. It is verified that the user no longer exists in storage after the Delete call
LoadsUsers	Makes sure the UserFileStorage can successfully load users from a file in Open()
QueryUserByName	Makes sure that a user can be retrieved from storage by their name

UpdateNonExistingUser	Tests that a UserNotFoundException is thrown when calling Update on a user that does not already exist
UpdateUser	Makes sure a user can be updated. The user's properties in storage are compared with the expected properties after the Update call
UpdateFinalUser	Updates the last user stored in a UserFileStorage which has been populated with many users to ensure loop coverage of the loop un UserFileStorage.Update
UserNameNotFound	Makes sure that GetByName throws an exception when a user with that name does not exist
UserFileNotFound	Makes sure Open() throws a FileNotFoundException when the users file does not exist
WritesUsers	Makes sure that the UserFileStorage can write its users to a file. Amount of users in that file is compared to an expected amount to check