Test Plan - Online Shop

Class: SalesItem

Contructor:

Contract: Creates a new sales item. Name and price of the item is passed as parameters.

Test method: testCreateTwoItems

Description: Since there's no limit to number of objects that can be created, we can simply do a positive test to check whether items can be created with no errors.

Method1: getName

Contract: returns the name of the item

Test method: testReturnName

Description: Similar to the constructor a simple test is suffice for this method. A positive test to check if the correct name is returned when invoked.

Method2: getPrice

Contract: returns the price of the item

Test method: testReturnPrice

Description: A simple positive test to check if the correct price of the item is returned.

Method3: getNumberOfComments

Contract: This method is to return the number of comments created for an item.

Test method: testNumberOfComments

Description: A boundary test can be applied for this method. Since the data structure used here is an ArrayList, it doesn't have a fixed limit. However, a test with an empty list is important. The test method tests both objects with empty list and non-empty list to check if the desired result is achieved.

Method4: addComment

Contract: Adds a comment to the comment list of a sales item. Return true if successful.

false if the comment was rejected (A comment will be rejected if it's by the same author or if the rating is invalid; a rating should be between 1-5 inclusive)

Test method: testAddComment

Description: Multiple test entries should be made for this case. Boundaries set for the Comment class include the restriction of the rating to the range 1 to 5. 1 and 5 as rating values are to be tested as they're the boundary cases. Good boundary test also includes testing values beyond the valid range of data; thus, we also test 0 and 6 as rating values and expect the method to return false.

Next, we test whether duplicate authors are correctly handled where comments by the same author is rejected. We use the test method to try adding a comment with an author name for whom a comment already exists and expect the method to return false.

Method5: removeComment

Contract: Remove the comment stored at the index given; does nothing if the wrong index is given.

Test method: testRemoveComment

Description: Since the method does nothing when an invalid index is given, a simple positive testing will be suffice. The test will invoke the method with a valid index as a parameter and it is expected to remove the comment promptly.

Method6: upvoteComment

Contract: Upvote the comment at 'index'; a method to mark some comment as more helpful.

Test method: testUpvoteComment

Description: A test to make sure the vote balance is correctly incremented every time the method is invoked for a specific item. To make it easy to test this method we also invoke getVoteBalance (getter method to be created) inside the test method.

Method7: downvoteComment

Contract: Downvote the comment at 'index'; a method to mark some comment as less helpful.

Test method: testDownvoteComment

Description: A test to make sure the vote balance is correctly decremented every time the method is invoked for a specific item. To make it easy to test this method we also invoke getVoteBalance (previously created) inside the test method.

Method8: findMostHelpfulComment

Contract: Return the comment with most upvotes.

Test method: testFindMostHelpfulComment

Description: A boundary test will be appropriate for this method. That is, when the method is invoked while the comment list is empty, we need to check whether it works as expected. Additionally, we ca also test to check what happens when two comments for an item have same number of votes.

Method9: ratingInvalid

Contract: A private method to support addComment method(public).

Test method: testRatingInvalid

Description: Since it is a private method which supports addComment method, test cases are designed to check whether comments with top and bottom boundaries are added without any issues. As mentioned earlier (in the addComment test description) it is also important to check values beyond the valid range, so we try to add comments with 0 and 6 as rating; program is expected to return false in this case.

method10: findCommentByAuthor

Contract: A private method to support addComment method(public).

Test method: testFindCommentByAuthor

Description: Since it is a private method which supports addComment method, test cases are designed to check whether

Class: Comment

Method1: upvote

Contract: Increments the vote count by 1.

Test method: testUpvote

Description: Since it's a simple method. Couple of positive tests would be suffice. We invoke the upvote method few times on the objects in the fixtures, then call the getVoteCount method to cross check the value.

Method2: downvote

Contract: Decrement the count by 1.

Test method: testDownvote

Description: Similar to the test above we invoke the downvote method few times on the objects in the fixtures, then call the getVoteCount method to cross check the value. It's important to note that invoking downvoteComment on comment with a 0-vote balance would result in a negative balance . Therefore, we also upvote the comment few times and downvote again to check whether the expected balance is returned every time.

Method3: getRating

Contract**:** returns the price of the item

Test method: testReturnRating

Description: A simple positive test to check if the correct rating of a comment is returned.