Liudmila Strelnikova, 08.03.2021.

Testing plans for online-shop project.

In the comment class there are two methods that need to be tested: downvote and getRating

General testing plan for the **Comment class**.

1) create five instances of the class Comment with different values for the variable "rating". Two of the instances should have the rating on the edges of the limit, and one should be within the limit range. The first comment will have 0 votes in the beginning, comment2 will have one vote, comment5 will have two votes. Here, we assume method upvote has been tested and works properly.
   comment1 = newComment(author, text, 1
   comment3 = newComment(author, text, 3)
   comment 5 = newComment(author, text, 5)
2) call the method of interest on each of the instances;
3) check if the results match the predicted values;

For method **getRating:**

1) create the 3 instances from the fixture;
2) call getRating on each of them;
3) compare with expected values:
   comment1: 1
   comment3: 3
   comment5: 5

For **method downVote**:

1) create the three instances;
2) for each instance, call downVote method;
3) call getVoteCount on each object and compare the returned values with expected:
   comment1: 0 //can't go lower than 0
   comment3: 1
   comment5: 2

In the class SalesItem, there are five public methods to be tested: getNumberOfComments, addComment, removeComment, upvoteComment, downvoteComment. Public methods getName, getPrice, showInfo and SalesItem are ignored.

General testing plan for the class SalesItem

1) create 4 instances of the class SalesItem, with different prices and different number of comments. Comments should have different number ratings and different number of votes.

//item mug will have an incorrect price set up (-1); it will have no comments
mug (“Mug”, -1);

//item time will have a price of 0, and two comments, one with rating 2, the other with rating 3. Each will have one vote for being usefull.
time (“The concept of time”, 0)
First comment: time.addComment("Bobby McGill", "Fake item but love the message", 2);
time.upvoteComment(0);
Second comment: time.addComment("NiceGuy", "I got two of these", 3);
time.upvoteComment(1);

//item dollar will have a price of 1 and have one comment with a rating of 5
dollar (“One dollar”, 1)
dollar.addComment("Lucy Lu", "That's so cheap!", 5);

//item textbook will have the biggest price of 500000, three comments: first one with two votes, second – with one
textbook (“Any university textbook you need”, 500000)
textbook.addComment("Poor Student", "I did not eat for three days but I got an A in JAVA thanks to this book", 5);
textbook.addComment("Oxford Student", "Not helpful at all, read only the intro and got bored instantly", 1);
textbook.addComment("Emma WhatSon", "Love books from this author, very good example projects but have to get a book with answers separately", 4);
textbook.upvoteComment(0);

textbook.upvoteComment(0);

textbook.upvoteComment(1);

2) call the method of interest on each instance or several times on the same instance where appropriate
3) compare the results with the expected outcomes

For method **getNumberOfComments:**

1) create all the instances in the fixture
2) call the tested method on each instance
3) compare return value with the expected values:
   mug: 0; time:2, dollar:1, textbook:3;

For method **addComment:**

1) create all the instances in the fixture
2) add a valid comment from a completely new user to any instance that already has comments (say, dollar), assert the outcome to be true;
   add a valid comment form the same user to the same instance, assert the outcome to be false;
   add a comment with rating above the limit (6 for example) to any of the instances (mug), assert the outcome to be false
   add a comment with rating below the limit (0) to any of the instances (mug), assert the outcome to be false

For method **removeComment:**

1) create all the instances in the fixture
2) use only object textbook for this testing; call removeComment(0) and then check if the first comment was removed and not some other. Do that by calling getCommentText(0) and assert the text to be equal to the text of initially second comment for the item: "Not helpful at all, read only the intro and got bored instantly" in our case.
3) call removeComment(1) now to check if the last comment was removed and not the first one. Check as before.
4) Now try to remove the comment with an index above the range(5). Nothing should happen, and upon calling of getCommentText(0), we should still get the same message as in two previous tests.
5) Lastly, try to remove the comment with an index below the range(-1). Nothing should happen, and upon calling of getCommentText(0), we should still get the same message as in two previous tests.

For method **upvoteComment:**

1) create all the instances in the fixture but use only the time one
2) call upvoteComment(0), call getVotes(0) after that and assert the number equal to 1 since it is the first vote for the comment
3) call upvoteComment(0) again and assert the number equal to two this time
4) call upvoteComment(-5) with index below the limit. Then call getVotes method on both of the comments in the instance to make sure non of them changes (should be 2 and 1) respectively
5) Do the same with upvoteComment(5) where index is above the limit

For method **downvoteComment** the plan is essentially the same as for upvoteComment except one more test is added:

1) try to downvote comment with 0 downvotes, the result should be still 0 votes and not -1