CITS5501 Software Testing and Quality Assurance
Semester 1, 2020
Week 4 workshop – Test design

Reading

It is strongly suggested you complete the recommended readings for weeks 1-3 *before* attempting this lab – in particular, chapter 7 of Barnes and Kölling, *Objects First With Java: A Practical Introduction Using BlueJ* (5th edn), ch 7 "Well-behaved objects". A PDF version is available via the Schedule page of the CITS5501 website.

Scenario

The code for this week's (adapted from *Objects First with Java – A Practical Introduction using BlueJ* by Barnes and Kolling) represents an early stage in the development of software for an online store (such as Amazon.com). The classes here represent the portion of the store that deals with customer comments for sales items.

The code contains only two classes: `SalesItem` and `Comment`. The intended functionality of the classes is as follows:

- Sales items can be created with a description and price.
- Customer comments can be added to and removed from sales items.
- Comments include a comment text, an author's name, and a rating. The rating is in the range of 1 to 5 (inclusive).
- Each person may leave only one comment. Subsequent attempts to leave a comment by the same author should be rejected.
- The user interface (not implemented in this project yet) will include a question asking, "Was this comment helpful to you?" with "Yes" and "No" buttons. A user clicking "Yes" or "No" is referred to *upvoting* or *downvoting* the comment, respectively.
  The balance of up and down votes for comments should be stored, so that the most useful comments (the ones with the highest vote balance) can be displayed at the top.

The `Comment` class represents a single comment, and the `SalesItem` class represents a single item offered for sale, including all comments left for the item.

Note that the code contains some deliberate mistakes.

Test case descriptions

Some functionality we might wish to test for these two classes is:

- Can comments be added and removed from a sales item?
- Does the `showInfo` method correctly show all the information stored about a sales item?
- Are the constraints (ratings must be 1 to 5, only one comment per person) enforced correctly?
- Can we correctly find the most helpful comment (the one with the most votes)?

Exercises:

- Identify at least three *test cases* which you could implement as tests, based on the described requirements (not on the code).

  For each of these, state what you think the preconditions and postconditions should be for that bit of functionality. (You may need to make some reasonable assumptions here – state what they are.)

- Are the test cases you have identified *unit* tests? Why or why not?
- Are there any preconditions which apply to *all* of the methods in the two classes?

Test cases:

- Each of these will be a single small bit of functionality:

  e.g. Given data for a comment, create a Comment object, add it to a SalesItem, and check it was added.

Unit tests:

- If they exercise the functionality of just one class, we can call them a unit test. If they check how two classes work together, it's an integration test.
- Ideally, unit tests should use *mocks* for classes other than the one under test – but we don't do that as yet.

Preconditions for all methods:

- Yes - in general, it's a precondition of all methods that you mustn't call methods on a null object, or a NullPointerException is thrown.

  And (if there's no documentation to the contrary) it is normally a precondition that parameters may not be null, either.

JUnit test exercises

You may find it helpful for these exercises to refer to the JUnit User Guide at https://junit.org/junit5/docs/current/user-guide/.

Remember that if using a version of BlueJ prior to 5.0, you'll need to add the JUnit 5 library classes – refer to the previous lab.

1. Can you identify two distinct test cases we might write for the `showInfo` method? What *fixtures* will be required for testing it?
2. Write tests which check that duplicate comment authors are correctly handled. What *behaviour* will your tests need to check for?
3. Write tests for the `addComment` method that ensure correct behaviour for different values of the `rating` parameter. What values would you choose? Why?
4. Write a test for the `upvoteComment` and `downvoteComment` methods. What test cases have you used, and why?
5. Write tests for the `findMostHelpfulComment` method. What test cases have you used, and why?
6. If your tests identify any bugs, fix the bugs.
   After fixing these errors, is it safe to assume that all previous tests will still work as before? If we run the tests again – what sort of testing is this called?
7. BlueJ includes the ability to *record* actions done in the IDE and use them as the basis for a test – read about this facility in the recommended reading and try it out.

1. Fixtures: need to construct a SalesItem object; may need to add comments to it.
2. If this would be a duplicate comment: `addComment` should return "false", ***and*** the comment should not be added to the list.
3. We should check *boundary* values – in general, we usually try to test on a boundary, and/or either side of it.
4. Upvote and downvote - some examples are, we want to make sure the total afterwards is correct.
5. findMostHelpful - in general, will want to check when there are 0, 1 and more comments. Will need to create and add those comments – they are part of the fixtures for the test.
6. Fixing bugs – no, we can't assume the same test results, we may have inadvertently changed the behaviour of the classes. Re-running the tests is called *regression testing*.