**Brandon Hobbs**

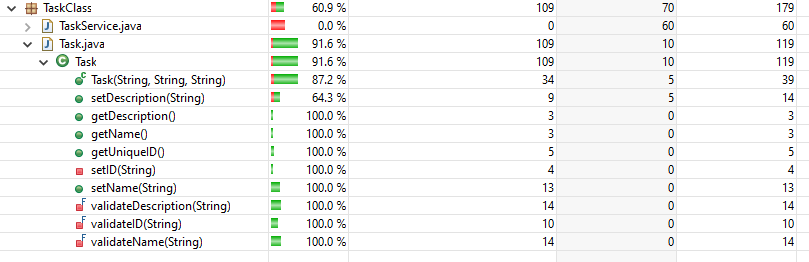
**CS-320**

**July 21, 2022**

**Module 4: Journal Entry**

**To what extent was your testing approach aligned to the software requirements? Support your claims with specific evidence.**  
  
I used the Unit Tests to try and test the boundaries of the specification. For example, the specification of the Task says that the full name cannot be null, greater than 20 characters, and is mandatory. To test this specification, no less than 4 branches are needed.  
  
*if (fullName == null || fullName.length() > 20 || fullName.equals("")) {return false; }*  
  
For example, I tested a valid version of the name, a null string, a very long string, and an empty string.  
  
*Assertions.assertThrows(IllegalArgumentException.class, () -> {tempTask.setName("");*

*Assertions.assertThrows(IllegalArgumentException.class, () -> { tempTask.setName("This name is incredibly long and why is this a name, anyway");*

**Defend the overall quality of your JUnit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**  
  
I did not try for 100% coverage of the Task and Contact classes. The constructors had 3+ parameters to set and didn’t think it was worthwhile to test good and bad Constructors for each of these parameters. Instead, I targeted 100% of the branches of the getters and setters and only a good constructor and a single invalid constructor.  
  


**How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**  
  
I tried to test “to the spirt of the code”. For example, in *ContactService* there was to be a method that deleted an ID. This could be tested in a few ways: 1) add three objects to the collection, delete one, and then confirm the collection is now size of two; 2) look through the entire collection for what you just deleted to confirm it is gone.  
  
The issue with only doing one of these is it leaves room for aberrant behavior. If only the collection size is checked then the wrong item could be deleted and the test passes. If the item is searched for as the only technique the test could pass if all the items were deleted accidently. I tested for both:

*@DisplayName("Test deleteContact")*

*@Test* ***void*** *testDeleteContact() {*

*String firstName = "Greg";*

*String lastName = "Hoffman";*

*String phoneNumber = "1234567891";*

*String address = "123 Main Street";*

***boolean*** *testBool =* ***false****;*

*ContactService test =* ***new*** *ContactService();*

*assertTrue(ContactService.contactList.isEmpty());*

*test.addContact(firstName, lastName, phoneNumber, address);//object ID 0*

*test.addContact(firstName, lastName, phoneNumber, address);//object ID 1*

*test.addContact(firstName, lastName, phoneNumber, address);//object ID 2*

*assertEquals(3,ContactService.contactList.size());*

*test.deleteContact("1");*

*assertEquals(2,ContactService.contactList.size());* ***//test one***

*//loop through to look for ID*

***//Test 2***

***for****(****int*** *i = 0; i < ContactService.contactList.size(); i++) {*

***if****(ContactService.contactList.get(i).getContactID() == 1) {*

*testBool =* ***true****;*

*}*

*}*

*assertFalse(testBool);*

*}*

**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

As mentioned above, it’s not enough to just check the size of a collection after a delete, checking that it is actually deleted is important. One step further could be to ensure the remaining objects are still intact and as expected (this was not tested for in my code).

I did, however, test to make sure objects were instantiated only when requested:

*ContactService test =* ***new*** *ContactService();*

*assertTrue(ContactService.contactList.isEmpty());*

I also checked that after a failed string update the string was still as before the attempt:

tempTask.updateTasks("1", fullName, "New description"); //bad ID

*assertNotEquals*("New description", TaskService.*tasks*.get(id).getDescription());

*assertEquals*(fullName, TaskService.*tasks*.get(id).getName()); //original string