Project Two

CS-320

Justin Swinney

**Summary**

Given the requirements for all three features (Contact, Task, Appointment) being similar in some areas and different in some I approached each feature with a fresh mindset and ensured I understood the requirements for each feature thoroughly. I wanted to ensure that most edge cases and functionality were met through my unit testing. I like to start my approach to each feature by listing TODO: within my IDE and list each test case I need to ensure I cover based on the requirements given, from there I will complete these and try my best to think of any possible situation I could have missed. A few examples of how my approached aligned to the software requirements are as follows,

**Contact / Contact Service:**

**Requirement**

* The contact service shall be able to update contact fields per contactId. The following fields are updatable:
* firstName
* lastName
* PhoneNumber
* Address

**Verification of Requirement**

* @Test  
  void updateContactFirstName() {  
   contactService.addContact(testContact);  
   System.out.println("Starting First Name: " + testContact.getFirstName());  
   contactService.updateContactFields("555","firstname", "Timmy");  
   assertEquals("Timmy", testContact.getFirstName());  
   System.out.println("Updated First Name: " + testContact.getFirstName());  
  }

**Explanation**

* As we can see the requirements ask that the contact fields be updatable per the contactID, given this requirement we see that I find the contact ID ex. “555” and proceed to update the first name, followed by checking that the update name matches “Timmy” verifying that the unit test is aligning with the requirements although I didn’t include all requirements in this example I performed them individually in there own unit test to avoid conflicting issues.

**Task / Task Service:**

**Requirement**

* The task object shall have a required name String field that cannot be longer than 20 characters. The name field shall not be null.

**Verification of Requirement**

* @Test  
  void testNullTaskName() {  
   assertThrows(IllegalArgumentException.class, () -> new Task("123", null,"task desc."));  
  }  
  @Test  
  void testInvalidLengthTaskName() {  
   assertThrows(IllegalArgumentException.class, () -> new Task("123", "1234567891011121314151618192021","task desc."));  
  }

**Explanation**

* The name filed cannot be longer than 20 characters verified with the testInvalidLength test. An exception was thrown as the length is over the allocated amount of 20 characters. And the name shall not be null was tested by providing a null name and expecting an exception to be thrown as well. Although areas of improvement could be more edge cases regarding the length of the name.

The overall quality of my unit test meets the requirements asked and provided good code coverage at nearly 100 percent, although there are edge cases that could be added and more efficient coding methods that could be applied to refine the test cases and create a higher quality testing. Code coverage allows the user to see what methods or sections of code are touched by the unit test within a set of classes. Although 100 percent of code coverage may not necessarily mean that all aspects of possibilities are covered. I ensured that my code was technically sound by a proper naming convention for my variables and unit test as well as methods used. This allows another individual to quickly be able to identify what’s taking place within the code. I ensured that unit test only tested one thing for example a unit test to update the first name and a separate unit test to update the last name this ensures that I would have accurate results and avoid issues that could occur from trying to update multiple items at once. Implementing Junit testing such as assert throws also verifies that my code is technically sound while also testing for proper functionality. As far as efficiency goes with my knowledge of Junit testing, I felt that my code was efficient keeping my unit test relatively small allow for quick processing times and easy maintainability of the unit test. **FINISH EXPLAIN EFFICIENT HERE**

**Reflection**

TESTING TECHNIQIES HERE

I

Ii

Iii

MINDSET HERE

I

Ii

Iii