**Project Two**

Author: Jessica Ayer

CS320 Software Test Automation & QA

Instructor: Mathew McHann

22EW5: 05/19/2022

**Summary**

**Describe your unit testing approach for each of the three features:**

When writing unit tests, my approach was to focus on aligning with the requirements listed in the project’s rubric as to what the system should/should not do rather than getting caught up in what my system could do. For each of the three features, contacts, tasks, and appointments I used the requirements to determine my test cases. I tested to ensure that both valid data processed correctly and invalid data was rejected. For each feature, input parameters were defined. For example, all three required a unique ID that needed to be 10 characters or less but not null. To comply, I wrote unit tests that checked to make sure an illegal argument exception was thrown if the ID number was too long or null. The requirements also stated what data should and should not be updatable. For instance, ID numbers should not be however, the name of the contact, task, or appointment description should all be editable as well as additional details about each. For the appointment feature, an additional requirement was that appointments could not be created in the past.

All of the provided requirements for deleting and updating data were tested by inputting an ID number for the desired element to be edited. For the updates, replacement data was also provided for each method called. Tests were run to make sure the ID numbers that were used as input were now associated with the new data that was expected after the method was called. As each unit of the system was built unit tests were deployed to make sure “the code written for the unit met its’ specifications, prior to integration with other units” (Hambling et al, 2015). As I gained more experience with building JUnit tests, I gained better code coverage percentages and discovered better techniques to ensure I tested for all necessary requirements. When integrating the units of the system, I adjusted and improved the unit tests to gain 100%-line coverage on all six of the required classes.

**Describe your experience writing the JUnit tests:**

To ensure that my code was technically sound and efficient I used to built-in methods, java data structures, and OOP principles. Rather than having each contact, task, or appointment be a separate object, I chose to use an array list data structure to hold instances of each object. In each service class a new array was created and methods to add, delete, and update the instances of objects. Using an array has simple built-in methods for adding and deleting items removing the need to “reinvent the wheel”. Example lines of code from my TaskService class are taskList.add(newTask); where newTask was a new instance of the task object, and taskList.remove(i); where i represented the index point associated with an ID number to be deleted.

For all three object classes and all three service classes I was able to recycle code with simple adjustments to the variables. All of the delete and update methods had for-loops to iterate through the arrays index with nested if statements to search the associated array for matching ID numbers using the built in compareTo method. I then called set methods from the object classes to make the desired changes. Referring back to my TaskService class it looked like taskList.get(i).setTaskName(taskName); where i once again referred to the index for the matching ID number and taskName referred to the new name for the pre-existing task.

I did an iterative testing process in which not only did I use the information from the JUnit testing to make sure requirements were met but also to reduce unnecessary code. When running coverage tests, I realized I was having successful tests as far as meeting requirements was concerned by my coverage percentage was low because there were lines of code that were unused and not needed. I was able to cut out the extra lines and run regression tests making sure edits did not affect performance. For instance, I initially had set methods for the unique Id numbers that looked like public void setContactId (String contactId){ this.contactId = contactId;} however, the numbers are auto generated in the add method and not updatable. The methods were just wasted lines to comb through so they were removed. The appointment class did not require any set methods as there were no update requirements.

**Reflection**

**Testing Techniques:**

This project specifically focused on unit testing by using equivalence partitioning to “divide up all possible input test data in a set of values for which we assume to be processed in the same way” and boundary analysis to test the “upper and the lower boundary of the equivalence class”, but other testing techniques were used on a more informal level as well (Garcia, 2017). Unit testing is practical in all software development projects because it is much more efficient and cost effective to debug code on a small scale or in units rather then waiting until a project is compete before performing dynamic tests.

Other testing techniques that were used was test analysis to decide what components of the system I was testing and integration testing to “expose defects in the interfaces and the interactions between integrated components of systems (Hambling et al., 2015). Test analysis is practical in all software projects as “exhaustive testing is not feasible” in the majority of situations (Garcia, 2017). A system like the one developed for this project could receive infinite combinations of input from the user calling the various the methods in any random order. For a larger system it would take far too much time and money to even attempt exhaustive testing. For this project integration testing was a simple process as the three features even when put into one package, still operated separately and therefore did not interact. In larger project with multiple people working together this testing technique would be crucial. It would also be crucial in this system if a driver class was implemented for the user interface that called methods from all three features.

Two types of testing techniques stand out to me as ones I did not use for this project. It could be argued that I didn’t fully implement test planning as I did not produce a test plan document nor did I choose the type of testing that would be conducted because it was predetermined by the assignment guidelines, and I did not document risks (Garcia, 2017). Normally test planning would have been the first phase of testing and is an important part in any software project so that testing is designed to be in alignment with the clients’ requests and there is a set measure for when the project is done. The other type of testing that has not and will not be utilized for this project would normally be done towards the end is system testing. System testing tests the “functionality from an end-to-end perspective” to see how it will function in a live environment (Hambling et al., 2015). This would be required for any professional project that was meant to be deployed.

**Mindset:**

It’s challenging to write code and then test it objectively. As the tester I had to be cautious to make sure I wasn’t testing to prove my project worked but rather testing to see if it did not. As Hamblin et al. puts it, the creator “has a special relationship with the created object” and we must “accept that testing done by that individual cannot be assumed to be complete” (2015). While working on the ContactService class, my updatePhoneNumber method did not work and I was so caught up being determined I coded it right that I failed to find my error until I walked away from the code for a day to reset my mindset from creator to tester. “Flaws in the created object are rendered invisible to the creator” (Hambling et al., 2015). I realized that yes, I did indeed code the update method correctly however, it was interrelated with the Contact class where there was a flaw in my setPhoneNumber method. I failed to add the line this.phoneNumber = phoneNumber; That one simple line was so crucial to the success of the method but was overlooked when I was viewing the project as the creator.

Overlooking the possibility of such a simple flaw made me more aware of my bias due to my relationship with the code and going forward I tried to limit it. When building the test case for the ContactService class I struggled to separate the test cases for the update methods and get the to pass so I cut corners and combined them all into one test. When I integrated the projects and had improved at getting into the mindset as a tester, I was able to use some system.out.print statements to see where the creator side of me made mistakes. This allowed me to rebuild my test cases and separate out each method for separate testing and catch the bugs in the system.

Finding this discipline to commit to quality rather then cutting corners to prove my code “worked” was important for avoiding technical debt. As projects grow little flaws will grow into large, possible debilitating bugs in the system and maintenance would become far more challenging. Any changes to any one of the update methods would have cause all of them to fail testing when combined into one test case. It was important to pay attention to the complexity of both the code and the test cases to keep everything efficient. By finding the interrelationships amongst the objects and using patterns throughout I was able to quickly debug the code when testing found faults and fix them before the little faults caused bigger issues.

**References:**

Garcia, Boni (2017). *Mastering Software Testing with JUnit 5.*  Packt Publishing. Retrieved from https://eds.s.ebscohost.com/eds/ebookviewer/ebook/bmxlYmtfXzE2MjY5NTBfX0FO0?sid=a9a69dec-f820-4fed-9446-9c24d3f95d51@redis&vid=4&format=EB&rid=2.

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition) - 2.3.2 Integration Testing.* BCS The Chartered Institute for IT. Retrieved from  
https://app.knovel.com/hotlink/pdf/id:kt00UC2HR2/software-testing-an-istqb/integration-testing.