**CS-320: Project Two**

Joshua Shults

SNHU

CS-320: Software Test, Automation

Instructor Name

12/15/2024

**SUMMARY**

My method for my testing approach to each of the three developed features was to perform a structured unit testing on each requirement. I used JUNIT 5 to validate that each unit performed as required by Grand Strand Systems. My first example will come from the contact service file.

The requirement for one of the variables in the contact service file was that the contact ID had to be unique and unable to be modified. It required a unique ID, first name, last name, phone number, and address field. Variable fields could not be null or exceed the intended character limit. A specific example from the code that I had was when I was testing if a field could be null: assertThrows(IllegalArgumentException.class, () -> new Contact("12345", null, "Shults", "1234567890", "19 Wendlin Lane"));. This line of code tried to pass a null field to where the first name variable exists, and it correctly asserted that the Illegal Argument Exception was thrown for this class.

Another example comes from the Task Service file. The requirements for this file were like the example above, fields could not be null, a unique ID was required for the task, and there were character limits for the task name and task description fields. Here is the example line of code from the test:

Task task = new Task("12345", "Task Name", "Task Description");

assertEquals("Task Name", task.getName());

Here I was using the “assertEquals” command to ensure that the ability to add and update a task was performing correctly. The command will assert that the name “Task Name” uses the “getName()” command properly and that it returns the valid name.

And for a final example, in the Appointment Service file I was testing requirement constraints for an appointment ID to have a unique field, valid date and description. I created a variable for testing purposes called “futureDate” which took the system clock time and added +100000 to ensure that the date was in the future, therefore valid. In the line of code I added another variable called “pastDate” which did the same as the above but utilized a negative number from the system clock, which would throw a class exception. Here is the line of code: assertThrows(IllegalArgumentException.class, () -> new Appointment("12345", pastDate, "Test Description")); By using methods like this we can be assured that class fields will have to properly follow the constraints provided in the requirements and will correctly throw exceptions to any invalid fields.

I believe that my testing followed the requirements completely, as in I made sure to test each variable field by its defined requirements. This means that I achieved a very high test coverage for each file that I developed. High coverage ensured that each of my classes, constructors, getters and setters all followed the requirements that were intended for each variable. Because of this I believe that my tests were of high quality.

My experience in writing these tests was at first challenging, as I have never done testing further than simple “stubs” that issue print statements. While my former method of testing did work, utilizing JUNIT testing provided a better format for me to create test cases as well as do so much more efficiently. I used to have to go back and delete my stubs, or comment them out, which in the former case allowed for me to accidentally delete code that was not part of the test and in the latter case made my code much harder to read. Overall, I would say that my experience creating tests for this course was a great learning experience, and also it was simply just very fun.

**REFLECTION**

As for this project and the milestones leading up to it, I mostly employed unit testing. Unit testing focuses on each component on an individual basis. This is great, especially during early testing, to find errors early on. With unit testing a developer can be assured that the very basic functionality of their design does what it needs to do properly before further features can be added. Without unit testing a developer could add more and more to a function that already had a previously existing bug, and with the addition of new features pinpointing the problematic code is only made that much more difficult to debug. Instead, a developer should ensure that their base code is performing properly early on in the development process, this will save time in the future. In my unit testing for this project I used assertation methods to ensure that exceptions were thrown by intentionally violating variable constraints.

Later in the modules I began employing other software techniques, specifically integration testing. Integration testing focuses on verifying that interactions between classes and functions are working properly. Such as in the TaskService file where I asserted that a task was successfully created and stored in a hash map, and that the task could be retrieved by name and that it contained the correct name. System testing was not required in this project, but if these files were all put together into a runnable application I would perform system testing depending on how the application was meant to interact. A hypothetical in this case would be maybe the contact file was meant to interact with the appointment file. Say a doctor’s office that stored contacts and they were associated with an appointment. Here I would test if the two systems interacted properly, such as an appointment associates with a contact. I would test that pulling up an appointment correctly associates with the intended contact, and that all fields are formatted properly and display the correct information.

I would say that I had a cautious mindset when it came to this project. I tested each feature on its own because I wanted to be sure that everything followed its requirements properly. This might have taken more time to write, but I would much rather be sure that each line of code works properly than testing a singular line for proper exception handling and assuming that the rest of the file will behave properly as well.

For limiting bias, I intentionally tried to break my classes by inserting improperly formatted or null variables into each section. I tried to be as critical of my code as possible and took my time with each line. The applications that I developed were incredibly simple, but I did not let this influence my testing approach at all. I tested every feature in-depth without simply assuming that it would work properly.

Maintaining a disciplined commitment to quality in terms of developing applications is essential, not only for this project but also for any future projects in school or professionally. Testing is incredibly important to the development of software, and any laziness or cutting corners to time spent testing could mean compounding issues that end up plaguing development. We see this all the time in the world, whether in applications, software, video games or anything that is developed in code. A simple bug could be ignored and result in massively underperforming software. A personal example from my own experience comes with messing around with a “gravity system” in an unreal project. I incorrectly set the artificial gravity to 19.8 instead of 9.8 in the equation for how fast an object falls once thrown or otherwise dropped from a high point. If I would have initially tested it in the first case there would have only been two lines of code to examine, the formula for replicating gravity and the object class itself, like the car or human’s artificial weight value. Instead I moved on and added more and more to the mod and had to spend a lengthy time debugging to find out that I simply made a typo in my equation.

By correctly implementing the various testing strategies, as well as the great format I learned from this class, I believe that I will be much better off in the future when avoiding technical debt in any project I contribute to or work on myself. Testing is the most fun part of development for me, and this class will be invaluable to me in the future.