Nathan Estrada

Professor Lopez

CS-320

6/18/2023

Grand Strand Systems Summary And Reflections Report

1. **Summary**

* Describing my unit testing approach for all three features
  + The three features required for Grand Strand Systems software were to have a service that set up appointments, a service to manage contacts, and a service to manage tasks, with each of these features having their own requirements. Throughout all of these features, I implemented logic in my code that would only accept the requested parameters for the attributes when creating an instance for each class object. For example, Grand Strand Systems requested that contact IDs are not updatable. In my contact.java file, I have no public setter methods for the contact ID, and instead use this.id when creating an instance of a contact(Line 30). Another example of meeting software requirements is shown in line 17 of the Appointment class. Part of the system requirements is that it will not be possible to schedule an appointment for a past date, so an if statement is implemented to ensure that only dates that are in the present or future are accepted when creating an appointment object. A third requirement for the software is that all Task IDs entered must be unique. In order to meet this requirement, the addTask function in line 13 of TaskService.java file is of type boolean, and is returned false if the entered task’s ID matches any other task ID within the array of tasks. If a matching ID is not found, the boolean returns true, and the task is added to the tasks array.
  + In order to create quality JUnit tests for all three features, and to ensure that all tests resulted in a high coverage percentage in whatever feature file they were testing, I thoroughly tested each file’s use cases and exceptions. For example, in order to thoroughly test the feature of adding and managing tasks, I made created two JUnit test files, one that tests the initialization of a task object (TaskTest.java) and another that tests the use of managing said tasks (TaskServiceTest.java). In the TaskTest.java file, I thoroughly tested the Task.java file, creating tests that ensure that the Task constructor, setters, and parameter assertions were all working properly. In the TaskServiceTest.java file, I achieved a high coverage percentage of the TaskService.java file by creating tests for the functionality and assertions of the functions of adding a task, deleting a task, updating a task name, and updating a task description. I applied this same method to the feature of adding and managing appointments, as well as to the feature of adding and managing contacts. Creating a test file that tests the initialization of the object itself, as well as its management service, ensures that the overall program for Grand Strand Systems will work as intended when the acceptable input is entered, and any unacceptable input will be caught instead of the program failing entirely.
* Describing my experience writing the JUnit tests
  + In order to ensure that my code was technically sound, I included parameter checks in each of the feature’s object’s constructors to ensure that only acceptable arguments can be passed for instantiation. An example of this can be found in line 23 of Contact.java, where in the class constructor, if the entered contact phone number is left null or is not exactly 10 characters in length, an illegal argument exception is thrown. However, if these 2 requirements are met, line 33 is called and the setNumber function is called. In the setNumber function, in line 55, all characters entered for the number are checked to ensure that they are numerical digits. If there are any characters that are not digits, an illegal argument exception is thrown.
  + To make sure that my code was efficient, I created high-coverage JUnit tests that took a wide range of the most likely inputs into consideration, including acceptable parameters and unacceptable parameters. For example, when testing the efficiency of AppointmentService.java, I know that it will be likely that someone will eventually mistype an appointment ID when attempting to delete an appointment. In AppointmentServiceTest.java, I tested the assertion throw for this type of invalid input in line 33, ensuring that attempting to delete an appointment that doesn’t exist will throw an assertion and display the appropriate error message to the user.

1. **Reflection**

* Testing techniques
  + The type of testing I used for this project was dynamic testing. Dynamic testing, unlike static testing, is a testing technique that is performed by running the code. Specifically, two main testing methods that I used were unit testing and regression testing. Unit testing typically uses a unit test framework to aid with testing (*Hambling, Morgan, 2019, p. 55*), and is performed on smaller segments on code called units. In my case, the unit framework I used was JUnit, and I broke up functions and categories of assertion throws as my units to be tested. I utilized regression testing by testing my code iteratively. As I would develop a function for an object class or service class, I would in tandem create a test for the function I had just created in the file’s respective test file, and adjust as I went. This method of testing helped me locate any logical errors or bugs in my code as soon as they were coded, which is much easier to debug and fix than waiting until the file being tested is already completed.
  + Some testing techniques that I did not implement in this project include state transition testing and flow charts. State transition testing is a testing technique that uses state transition diagrams to visualize how a system behaves in the aspects of static states and transitions from one state to another. (*Hambling, Morgan, 2019, p. 109*). Flow charts, similar to state transition diagrams, are a visualization of how a system works. Flow charts can contain different types of symbols, but mainly consist of how data is stored, altered, and flows throughout a system. Flow charts offer a little bit more freedom in visualizing how data is handled in a system, but they serve similar purposes in visualizing data flow. While these testing techniques can be incredibly useful for bigger and more complex projects, I found the scope of this project to deem these testing techniques unnecessary.
  + All of the testing methods I have mentioned all have their own practical uses and implications. Regression and unit testing are excellent for iteratively testing smaller segments of code to ensure that each organized segment of code is working, and to detect and correct errors and bugs as soon as they are implemented, instead of having to spend more time locating the cause of such errors after developing most of the code. Flow charts and state transition testing are great techniques to use for large-scale projects that involve hefty manipulation and transfer of data, as they help simplify and visualize the processes, giving developers a better understanding on how the system should flow.
* Mindset
  + When working on this project, I constantly kept in mind that the code I was working on was going to be suspect to all different kinds of undesired input. I heavily employed caution when writing my code and tests by implementing assertion throws for any kind of unacceptable input and writing tests for these assertion throws to ensure that they were working as intended. It’s important to understand the interrelationships of the code and functions that I am creating, as while I may think certain functions will work as intended individually, when used together, there can still be room for error. Going back to an earlier mentioned example, in class constructor for my Contact.java file, I had my code check to see if a contact’s phone number that was entered was checking to see if the the number was at least 10 characters long, and to check if the phone number entered was not null. If these requirements were met, the phone number setter would be called to assign the number to that instance of the class. While the setter worked as intended, I realized that the setter would still need to verify that all characters entered in the number were digits. This is one example of how to employ caution when testing the interrelationships of functions within code.
  + Creating code that is free of bias is incredibly important when ensuring that code that is being developed is ethical. In order for me to ensure that my code was free of bias, I implemented assertion throws and tested said assertions for possible inputs that some people may seem unnecessary to implement. For example, in line 29 of my Appointment.java file, I check to ensure that it is impossible to schedule an appointment for a date that is in the past. While attempting to schedule an appointment for a past date may not be something I could see myself doing, or some may believe that it is the user’s responsibility to be aware of what the current day is, it is important to be mindful that anyone can make honest mistakes, and that code should not be exclusive to anyone that is prone to make simple errors such as that.
  + It is crucial to stay disciplined to the commitment of developing quality code and code tests as a software engineering professional. Cutting corners when it comes to writing or testing code will lead to many undesirable traits, such as a product riddled with bugs, security faults, potential data breaches, and overall inefficiency. In order to avoid technical debt as a practitioner in the field, I plan on iteratively testing my code as I am building it, in order to detect bugs, security flaws, poor runtimes, or other negative factors as soon as they are coded. This will not only result in making it much quicker and easier to detect code defects and thus save time, but will save financial resources as well. I utilized such practices during this project by creating unit tests with each function that I would implement. For example, if I created a constructor for a class object, I would then immediately create a unit test that tested the functionality of the constructor, ensuring that acceptable parameters passed and unacceptable parameters were met with an assertion throw.

References

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing : An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning & Development Limited.