Brandon Randall

CS-320

8/13/23

Unit Testing

I used similar JUnit tests for all three features and these tests were heavily based on the software requirements. Though each function had different variables and requirements for those variables, the general tests were the same. Take the ID variable for instance. Each object class (contact, task, appointment) contained an ID that, per the software requirements, had to be at most 10 characters long and cannot be null. The constructor for these objects includes a check to verify these rules are met and then JUnit tests were designed to test that this check ran properly. The JUnit tests for ID checked that a valid ID up to 10 characters worked correctly and verified that a null or too long ID value would cause an exception. Most other tests followed similar patterns of checking the software requirements from different angles, such as with the Delete functions within the service classes. While the software requirements only stated a delete function was needed, it was necessary to test this function from multiple angles, such as testing when the object to delete is not present or testing with multiple objects present to ensure the correct object is deleted. This method of testing also ensured that all Junit test classes covered 100% of the class they were meant to test. This will also bring me into some of the issues I had with writing tests though, and how I adjusted them over time. When I initially made the delete function for the contact and task services, I ensured the JUnit test passed, but I did not test as in depth as I should have. The original test would use one object and test whether that object was deleted. This did work, but not as intended. Because I had not checked from multiple angles with multiple test cases I missed the fact that the delete function was made in a way that if the item to delete was found in the list it would only delete the first object within the list, not the correct object. I discovered this error when designing more in-depth tests for the appointment service and changed the Delete functions code to work properly, and created multiple tests to ensure that all the edge cases I could think of were covered. These tests start at line 31 in the AppointmentServiceTest file and end at line 76. Covering these extra cases ensured that the code was sound and reduced the likelihood of an error occurring. Throughout the features there are also multiple implementations that are meant to aid in efficiency of the code, such as exception handling to ensure code does not execute if an exception is caught and break points included within the Delete and Update functions to ensure the loops do not iterate longer than necessary. Specific examples of both can be seen in the delete functions for all three features. I will use the appointment service for this example.

public void deleteAppointment(String ID) {

//check if Appointments is empty

**if(Appointments.isEmpty() || !findID(ID)){**

**throw new IllegalArgumentException("Appointment does not exist");**

**} (This section of code occurs before the loop to ensure that the appointment to delete exists.)**

//iterate over Appointments

for(Appointment Appointment : Appointments) {

//check for a matching ID

**if(Appointment.getID().equals(ID)) {**

**//if found, remove this object from Appointments and break the loop.**

**Appointments.remove(Appointment);**

**break;**

**}**  **(This if statement is triggered when the correct appointment is found, and the**  **break statement ensures the loop ends before unnecessarily looping through**  **the rest of the items)**

}

}

Reflection

This project involved multiple different testing techniques. The most basic technique used was manual testing, which involves things such as proofreading and manually ensuring the code and logic are correct without running the code itself. This type of testing is necessary for any project and is the first line of defense when it comes to catching errors. One of the next major forms of testing is Unit testing, which we did through JUnit tests. This form of testing checks that individual classes and functions work as intended. This, again, is necessary for any project to ensure code functions as intended and to test edge cases. Acceptance testing goes hand in hand with unit testing for this project since many of the unit tests were made based on the software requirements. Acceptance testing ensures that these requirements are met. Regression testing was also a major part of the project since changes were implemented to fix the above issue with the Delete function. Regression testing is testing that ensures things work properly after changes are made to the code. I utilized this method of testing whenever I needed to refactor the code, such as with the changes to the delete function. This method of testing becomes more important the more integrated the system becomes because a change to a function could affect a lot more of the system.

Some testing techniques were not necessary at this stage of development. Integration testing is testing that is done to make sure integrated blocks of code work together. It could be argued that testing that a contact can be created within the the contact service could be considered integration testing, and while that might be true, it is on the most basic level of it. I would see integration testing occuring once these features are combined into one main method that utilizes them all. This type of testing is necessary any time you have units that are integrated to work together to ensure everything runs properly. System testing is the next step above integration testing and this testing ensures the entire system works as intended. This is done near the end of development when all the units are combined to make one system and a functioning product. Since these features are not part of a system yet, this stage of testing is not needed for this project. Security testing ensures that user data and other information is stored securely and that the project does not have security concerns. While this will be needed later in development, with the project holding things like contact information, it is not necessary at this time due to how early in development this project is. Lastly, I will mention non-functional testing, which looks to test nonfunctional requirements of the project, such as load times. These requirements will be more fleshed out later in development but are not present at this time.

Mindset

This was really my first time truly testing code. With a project like this where the classes are built similarly it is easy to just copy and paste and not reevaluate your work because it “worked” the first time. Implementing the feedback I got after each section was complete, I was able to look at the next section and make sure I was testing for more cases and finding flaws in my designs. I learned quickly that just copying and pasting, while it does get things done quickly, can lead to issues. Some of these issues include forgetting to change information which can cause things to work incorrectly, and potentially giving a false positive on tests. I did not have any issues with bias in my work, I know I’m still learning and expect to make mistakes. With every class I implemented more and built off the mistakes of the previous, even when I did not think I had made mistakes. One case of this happening was after submitting my initial Contact classes and while working on the Task classes I received feedback that the ID variable being unchangeable needed to be tested. I had made this variable private and the set method was also private which made it hard to test within unit tests. I knew this was the correct way to do this, but I did research to find if there was a testing method I missed and tried to implement methods to test this. After confirming with other sources, I concluded that this was all I could do to test this requirement. While I could have simply ignored the feedback because I thought I was right, I still researched it to be sure. There were many other examples of implemented feedback that aided in me creating better code as each class was completed. The issue with the Delete function would not have been found if I had not implemented more detailed tests, which was feedback given for previous files. Ultimately, ensuring code is tested properly and designed properly is necessary to produce great code. Sometimes that involves accepting feedback and admitting that we do not know everything. If this project had somehow been released without fixing this error there could have been thousands of objects created, with no efficient way to delete them without reevaluating the code, finding the error, and fixing it. This takes time, money, and resources that could set a company back.