Sean Flynn

12/16/23

CS-320 Project 2

For the milestone stone projects I needed to write a task service, a contact service, and an appointment service. All of which were very similar to each other. They all needed to first setup the service then add objects containing simple data, all of which were strings limited to a certain size and usually excluded being null. Writing the code to set these up was fairly straight forward. I then used JUnit tests to confirm that each service was able to perform the required functions as well as keep it's attributes within the requests parameters.

During the course of my testing I was able to reach coverage on 97% which is a number I am very happy with. I actually enjoyed doing the testing process and writing the tests which I was a little surprised by. I have been a programming as a hobby for many years, mostly simple games using Python. I would look up things on occasion, but I really enjoyed solving problems myself rather than looking up the correct or most efficient solution. Printing to console has always been my best and hilariously only method of testing when I had an issue I was trying to solve. Learning real methods and the tools to enact them was a fun experience. The feedback from the IDE is great and I found it very easy to parse. I could easily pinpoint what was and wasn't being checked and adjust the tests to increase coverage.

One test that I ran that I felt was important was to test the function that searched for contacts/task. Because this function is used to check for duplicates as well as for updating, several other functions relied on it. It was important to make sure that not only would the search function correctly find what it was searching for, but also to ensure it would correctly return null if it's search term was not found. Using a search function reduced the need to retype code as several other functions required it's use.

*@Test*

void testFindContact() {

ContactService contactService = new ContactService();

Contact testContact = new Contact("A1B2C3D4E5", "Sean", "Flynn", "1112223333", "Monkey Island");

contactService.createContact(testContact);

*assertEquals*(testContact, contactService.findContact("A1B2C3D4E5"));

*assertNull*(contactService.findContact("Waffle House"));

}

There area many different types of testing as software is very complicated today and has many moving parts that are interacting with other software. One method of testing that was not needed here was integration testing. Integration testing is used to test that different components of a larger project work together as expected. Each milestone required only two things: the object, and the service to handle that object. Down the line, should the services be combined into a larger application, then it would be appropriate to test that they fit together as they should. Another type of testing that was no used due to the simple nature of the problem was performance testing. This would include things like load testing and stress testing. Software needs to be able to meet the demands expected of it in the same way buildings do. An application designed for multiple users, for example, would need to be tested to ensure it does not slow down, crash, or experience other problems when put under that load. In the gaming industry this is often done as part open and closed betas, in which the game is opened up to a flood of players to see how it responds under stress.

In this case I used JUint testing in order to test the services made. Unit testing works well in this case as the project is not large, and does not need to communicate with another structure such as a database. In addition the code is easy to divide into units, and straight forward to test. Each unit is isolated and tested individually to make sure it is working as intended. This is a common first step practice, before code is integrated into a larger project.

During the course of testing my code I found that I did have a change of mindset. After getting some hands on with the testing tools, I started thinking about how I wrote my code. The coding and testing should go hand in hand so programmers should follow best practices, and tried methods so that testing is not unnecessarily difficult. Code should be clear and use good flow control so that during testing there aren't 'dead ends' or other odd bits that prove difficult to test. Learning about how the testing works and what methods it uses to check for coverage made me think about how I setup my own code structure. Although I could see bias being a problem on a larger scale or while working in a larger group of people I can't say it is something I thought about while working on my projects. I can say that sometimes it helps to have someone else check for mistakes as we sometimes become blind to our own. For example when writing a letter you may have missed a word but you can't notice. In you head when you read it over you already know what it's supposed to say and just skip over the omission. Coding is much the same way. With the rise of AI tools it will be interesting to see what effect they have on development, especially when used by individuals and small teams who are short on available man hours.

Lastly, I believe that discipline is very important in software development. Many security breaches could be prevented it code was better tested and better made. As I pointed in my discussion, there is a game that could have lost a lot money because they did not correctly program their online store. It was a mistake, and an obvious one, that was picked up on very quickly by an outsider. Someone was either too lazy or didn't think ahead enough to check for an overflow error, which is a very common method of trying to break software. It should have been tested so that any input, and therefore every edge case was accounted for. Especially since it was a simple as managing a single integer.