Joshua Ladue

April 20, 2025

CS 320

Professor Haney

Project 2

I would say that my testing approach lined up pretty nicely to the project requirements. My biggest experience using JUnit is this current course and both of the assignments have relatively similar requirements. As someone who thinks it is a good idea to code in parts and test as you go, the test cases were pretty specific in what they were looking to test so it was pretty easy to test the specific cases as you are comfortable testing them out.

Going off of the module 4 assignment, I would say that my JUnit tests were mostly effective because my coverage was mostly in the 90’s for the tests. It was not 100 percent coverage for everything, but I do believe that I was able to reach a pretty high number for coverage.

To make sure that the code was technically sound, I used some techniques. For the test cases, I made sure that there was cleanup made using “@AfterEach”, “void tearDown() throws exception”, TaskService.tasks.clear();. This makes sure that after each test case, there is a degree of a restart to ensure that the tests do not interfere in any way and generate the best results possible. Encapsulating code using functions for the getters and setters for the variables was also used, such as “public String getName” and “public void setName”

I did some things to ensure that my code was relatively efficient. First I tried to name most of the variables and functions with best practices in mind like camel-case, and make them have a similar naming scheme to increase the readability of the code, such as “taskID”, “taskName”, “taskDescription”. I also made sure to have displays for my tests using “@DisplayName” so that the tests were easy to recognize when I run the JUnit tests to see what was working or not.

The main technique that we have been using so far in the work for this class is unit testing using JUnit. For the tasks, appointments, and contacts, we have had to both write the code for the actual classes and services, but also write unit tests for all of them as well. These unit tests test the different parts of the code separately to make sure that they work individually to each other, for both the functions of the code, and making sure that the validation of the variables is going to be correct when the code runs.

A technique that we have not used yet in this course is integration testing. This makes sense however given that integration testing is based on how different parts of the code work with each other versus how they work alone for a unit test. This will surely come later in the course as all of the different parts of the code are made so we can test them all together to make sure that everything runs as well as it should without any bugs or errors. Another technique that has not been used is regression testing. This also makes sense given that regression testing tests how code works after any updates or changes to make sure nothing is broken in the updates. We have not updated any parts of the code yet so far after initially writing them, so there is no need yet to do a regression test.

Unit testing is very useful while focusing on a small or medium amount of code to test as you develop things. This is great to test how components work within themselves, but will not help much later as you are putting all of the pieces together for a program. Integration testing will help here as it will test how things integrate into each other. Lastly regression testing will help as you are updating things to test if anything has broken as you change any code.

Throughout the coding and testing process, I would say that I did not really employ too much caution, or at least not more than I would usually do. It was very important to realize the complexities of the code and interconnectedness of it. Each part of the project had two classes in two packages so it was important to make sure that the relevant classes are able to access each other. Specifically it was important to import the classes so that the “class” class and the “test” class for each different part could access each other so that the different functions are able to work.

I do believe that bias is an important part of code testing, and having a bias towards your code is particularly dangerous to how well it is going to perform as you can grow too comfortable in your own style and abilities and as a result not put in as much effort into quality or testing. I would say that the way that I assessed bias in my code and testing is that I mostly tested using JUnit, which is a pretty sure way to make sure that your testing is done exactly without any bias because the test cases will tell you exactly if they failed, passed, or had an error running it.

Being disciplined is an essential part of software engineering. Keeping standards high is an important part of that. There are some ways that I intend to reduce technical debt in the future. I can make sure to write high quality and efficient code that can reduce the amount of code that I have to write or rewrite, implementing reusable code is a good way to improve efficiency. Another way to reduce technical debt is doing what we have been mostly doing throughout this course, testing. Having regular and effective testing can greatly improve your time coding as you have a good chance of reducing the time that you will have to take later to fix anything wrong with your code as you can catch the errors earlier. An error domino effect is also a possibility if you do not catch any imperfections early on as issues can easily compound onto each other.