Joseph Silva Jr.

12/13/2020

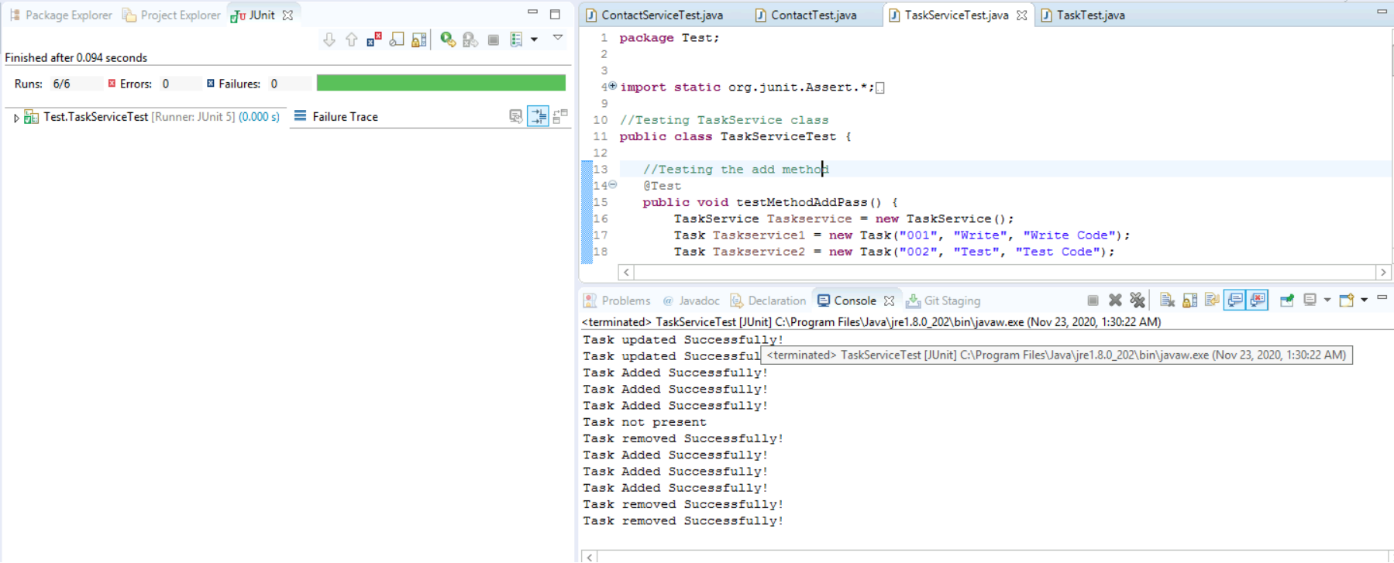
CS 320 Project

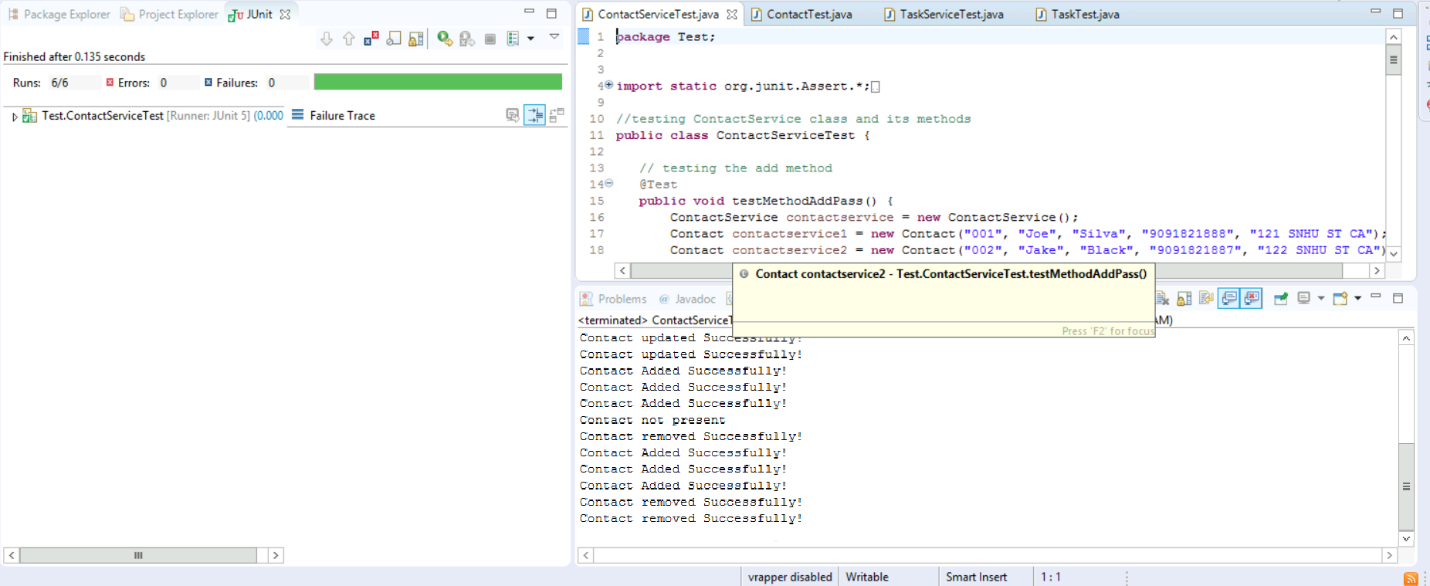
SNHU

**Summary**

* **To what extent was your testing approach aligned to the software requirements? Support your claims with specific evidence.**

I tested each file to meet the requirement for the smaller tasks of each file given to us in the assignments. For Contact and Task files, I made sure my Junit Tests tested each string to be compliant with the smaller tasks such as each string has to be a within a certain length, and they were not allowed to be null. For ContactService.java,TaskService.java, and AppointmentService.java files, I made sure my Junit Tests tested each files ability to add, delete, and update either Contact List or Tasks List depending on the file. I also made sure my Junit test tested the AppointmentService.java file to have the ability to add and delete elements from the Appointment List.



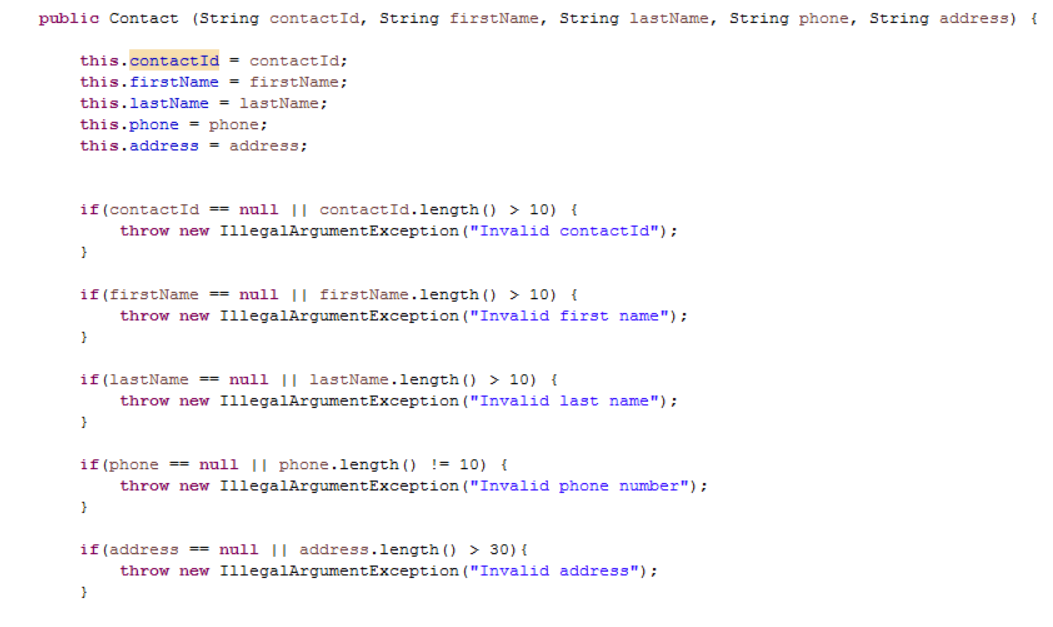


* **Defend the overall quality of your Junit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of coverage percentage?**

The overall quality of my Junit Tests for the contact service, task service, and appointment service files were great. The reason I know this is because I ran multiple tests with different sets of code and there were many times where the Junit Tests with failure traces. I also ran the three codes that I uploaded to the assignment multiple times to make sure I had a 100% run rate without errors or failures. The photos above will show on the left-hand side of the photographs about the runs, errors, and failures of the Junit Tests.

* **How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

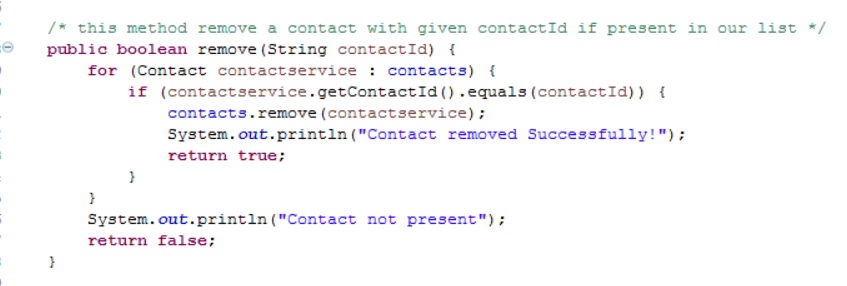
After writing a small block of code, I would save the file in order to update any errors and/or failures within the editor area of Eclipse. I also conducted multiple tests after creating small blocks of code within the test files to test newly added blocks of code. For example, I made sure my Contact.java file met all requirements necessary for each String and then I would conduct tests using the Junit Test ContactTest.java file.





* **How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

I tested my code multiple times to make sure the runs had no errors or failures. I then checked the output console to make sure each Junit Tests gave me the output I expected. Specifically, for ContactServiceTest.java, TaskServiceTest.java and AppointmentService.java, I checked for the files ability to add, remove, and update information depending on files as I previously stated above. In photos show below, I wrote small blocks of code to test ContactService.java file ability to delete contact information using a contactId. I then wrote small blocks of code to test my code in ContactService.java file using ContactServiceTest.java Junit Test to make sure the I obtained the expected output without any errors or failures.





**Reflection**

* **What were the software testing techniques that you employed for each of the milestones? Describe their characteristics using specific details.**

The following is a list of software testing techniques that I have employed during each of the milestones: static, dynamic, automated, functionality, white-box testing, and unit. While using Eclipse, I was able to use static and dynamic testing techniques. In the edit area of eclipse, I was notified if I had any errors or warning withing each class of code. This would be static testing because I know about the errors/failures in my code structure without having to execute a testing program. The Junit Tests allowed me to use dynamic testing technique when executing the code to see if there were any failures/errors after the test classes tested my class files.

Next, the automated testing was when I used Junit Tests for the Contact files, Task files, and Appointment files. I also used the functionality test while using Junit Tests because they tested each file without implementing these files into a full program. The two-last type of testing technique was unit testing and white-box testing. I used these two types of testing techniques during each milestone. Each class file is being tested separately as a unit to make sure they individually work without any errors or failures. Unit testing is a form of white-box testing because my code is going through a workflow to make sure everything in the individual class does not conflict with each other.

* **What are the other software testing techniques that you did not use for the milestones? Describe their characteristics using specific details.**

The following is a list of software testing techniques that I have not used in the milestones: manual, integration, system, and acceptance. I did not perform any manual tests regardless of me writing the test class files myself because I was able to configure a test program while using the Junit Testing technique. The next two, integration and system, were not used because I was testing each class as an individual file and not the compatibility with each other. I have not tested a milestone’s files with another milestone’s files to make sure files such as Contact.java and Task.java will work together. Since I have not used these programs together, I did not build a main structure to test these files at an entire program. The last type of testing I did not use was acceptance because I need a full program to make sure it meets a user requirement, but since there is no main class file, I am unable to conduct this type of test.

* **For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

All the techniques are very important when development a software project. Techniques like static and dynamic are used throughout the entire project especially when you are writing code on a program like Eclipse because these techniques help prevent any errors and/or failures while writing the code. Automated and Manual testing can be implemented when it comes to the amount of input you want to test to make sure it’s the expected output. If we are working with a few possible inputs, manual testing can be done using a small amount of time, but if we are trying to test thousands of inputs automated tests will save a tester time by having the program conduct the tests for you. The rest of tests such as unit, integration, and structure tests are implemented to make sure there is no error throughout the entire program. I compare these types of tests to real-world examples such as a workforce. We start with unit testing to make sure each class file works properly on its own. This type of test can be compared to an individual law enforcement officer. We run tests on the individual law enforcement officer to make sure they are physically, mentally, emotionally, and intellectually fit for the job. The next is integration tests make sure each file works together and notifies the testers of any conflicts between these files. I compare this to a law enforcement work shift because we have to ask does every member of this shift work well with each other as a team. The structure tests are the testing of the entire program has a whole to make sure there are no errors/failures as an entity. I compare this type of testing as a law enforcement station or department.

**Mindset**

* **Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.**

I adopted a cautious mindset while working on this project. The reason I adopted this mindset was to prevent myself from going overboard in trying to make new features. Since I was working with multiple files, I did not want to get jeopardize not being able to meet the requirements of the customer by trying to make my files unique. I think it is important to appreciate the complexity of the code being tested because any small mistake can mess with the entire project. When I was writing the code for items such as customerID. I had to make sure the proper tests were conducted to make sure it was not null, and it did not go over the max limit of characters.

* **Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.**

The way I limited my bias in my code review is by letting the Junit Tests do their job and relied on their outcome. I used the Junit Tests as guides in order to fix any error issues using dynamic testing. I can imagine the bias about testing my own code because a lot of people do not want to admit their work is wrong. Peer reviews and Junit Tests are very important because it allows an individual to take themselves out of equation and let their work be judged for its ability to work. I know when I was writing files such as ContactService.java, I saw code that I wrote with no errors / failures notified to me by Eclipse. However, I ran Junit Tests, and the outcome of the code did not meet the requirements. I rewrote the code and I was able to meet requirements, but I was only aware of my error due to the Junit Test.

* **Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.**

Quality is very important in the software engineering professional because your reputation is on the line to put out good work. I think once you have a bad reputation for cutting corners or not producing quality work, no one will want you on their team or assistance in a project. I believe in order to make sure I avoid technical debt; I will test my work constantly. I will also continue to grow by gaining knowledge and experience in writing code. I will prevent myself from putting half done work because not only does it hurt the project, but it hurts myself as well.