Skylar Walker

02/19/2025

CS-320

Project 2

For this project I was tasked with creating a program that can manipulate contacts, tasks, and assignments by doing things such as creating, updating, and even deleting them. I also had to tests that the program ran correctly and as efficiently as it could by creating a series of Junit tests that covered the entire program.

When it came to the first milestone, I had to create a contact and contactservice class. This was my first experience with Junit tests. The requirements for these classes was the ability to create a contact with a unique ID that could not be updated as well as a name, address, and phone number. All of these values had to be able to be updated and be looked up based on the user ID. My testing had to ensure that every feature that has been required of the classes are being used and utilized properly. My unit tests go on the basis of testing that a class gets created or updated correctly if a correct contact is given and also that the system acts properly if the system gets an incorrect input such as a phone number that isn’t ten numbers (*Junit - Test Framework*).

Void testContactPhoneToLong() {

Assertions.assetThrows(IllegalArgumentException.class,() -> {

New Contact(“Tom”, “Williams”, “123456789012”, “Address”); )); }

This test ensures the system handles incorrect inputs properly by giving a phone number that is to long and checking the output for the exception.

Tests also let you know your coverage. Coverage allows you to see just how much of your code is actually being read when you run your code (García, 2017). By either being colored red, green, or yellow for Eclipse. While running my Junit tests I noticed that some tests had more coverage than others. To fix this I got rid of some useless code that wasn’t being run and wasn’t needed as it was redundant. I had multiple checks for the same error. If it is getting caught by the first check then the second check is not getting used and it lowers my coverage. When doing this I was able to get the coverage of my project up to 77%, which is a good margin of use. That tells me that 77% of my code is getting run every time I run the program tests.

Sound code means it both works properly as well as efficiently. I believe I do a decent job at this but I do need some more practice in some areas. When creating my classes there was a lot of reused code. I would do the same thing for different values. For example my contact firstName and appointment name look exactly alike but only differ in name. I would say my getters and setters are more efficient and sounder being that they only have the code that is needed to run.

public String getFirstName() {

return firstName; }

With such a small amount of code it leaves little room for errors as well as making it efficient.

While writing my code with testing in mind I wanted to ensure there was a good amount of checks that the tests can run. Every aspect of the program get run individually. This helps to ensure each test works properly. This also lets us know when a test fails you know exactly what aspect was being tested and where to start looking for a fix. Junit tests within Eclipse are work on a pass and fail basis. If the test passes you get a green light. If it happens to fail, you get a red light as well as a string of errors that let you know where the test failed.

Along with unit testing there is also integration testing as well as functional testing. Integration testing is the testing of a program as a whole to ensure all the parts work well when paired together (Java, 2023). Integration testing techniques such as Big Bang have uses mainly in small systems where it is easier to run all the code together for testing and doesn’t low down the process by having to look over as much code as a larger system if an error is given.

While writing the code and testes for this Program I had to very conscious that what I was writing was going to work well with my unit tests. If I had wrote some code with no form of validation or way to check for input error it can make it hard to have the system function properly and not just send an error when the incorrect input it put in. Keeping in mind that this code needs to be tested and works well with other classes helped me ensure I had the correct formatting and that I checked every input for a valid response.

When it comes to testing your own code, you can grow to have some pride in your work. This can become biased when it is time for testing. This bias can make it difficult in testing your work but being able to see past your own bias can help make you a better programmer in the future. Having the ability to test your own code and understand that you can make mistakes only helps you more in the future.

Being disciplined to me means you have an understanding with yourself that whether you want to or not, you have a responsibility and need to do it to the best of your ability. You do something not because you always want to but because it is what you are supposed to do. This means while coding I always do my best every if I feel some burnout. I might take a little break but I always try to give my best effort when it comes to writing good clean code. Any time you might cut corners when coding you risk leaving yourself open to making mistakes and being less efficient than if you took your time and did your best work *(Pullen, 2018)*. Programming is just like any other skill. If you do bad work and instill bad habits, then you wind up doing a bad job. Being able to practice, take your time, and take breaks every now and then you can see yourself improve a little each day.

Sources:

*Java integration testing explained with examples*. BairesDev. (2023, September 8). <https://www.bairesdev.com/blog/java-integration-testing/>

*Junit - Test Framework*. Tutorialspoint. (n.d.). <https://www.tutorialspoint.com/junit/junit_test_framework.htm>

Pullen, J. (2018, November 25). *No cutting corners cutting corners*. DEV Community. <https://dev.to/pulljosh/no-cutting-corners-cutting-corners-1i3c>

García, B. (2017). *Mastering software testing with junit 5: Comprehensive guide to develop high quality java applications*. Packt Publishing.