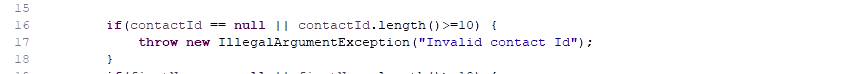
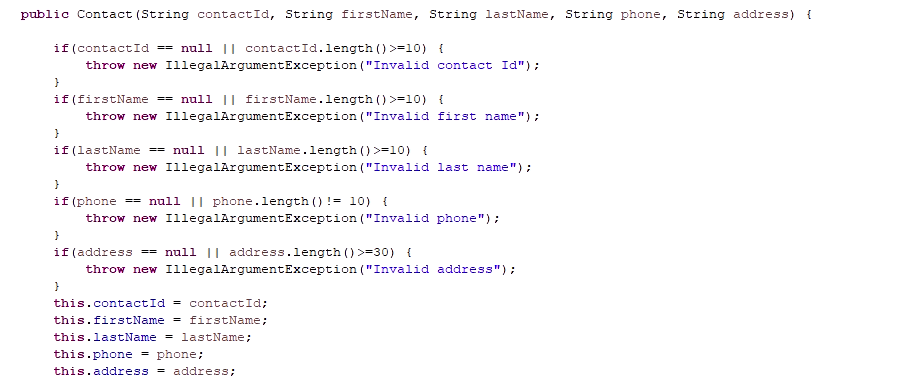
Project Two CS – 320

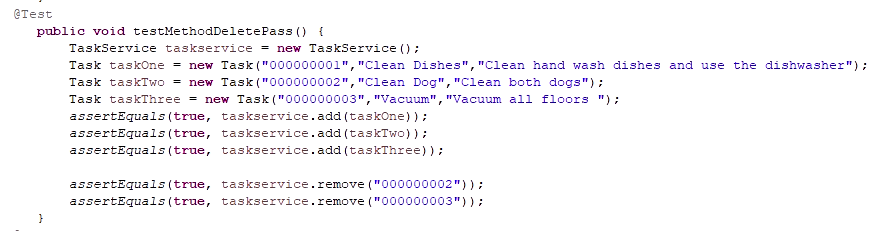
When building this project, it was vital to understand each requirement asked by the customer. I approached this in two different ways, and found one method to be much easier. I first tried to build a class around what was being asked. Since the Contact class was first one to build out, I began with the contact object having a Contact ID. It needed to be no longer than 10 characters, it could not be null, and it is not updatable. The code is below to see how the Contact ID would meet these requirements.



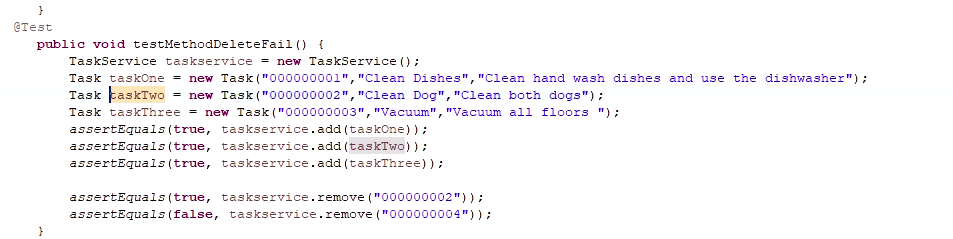
I read all other requirements for the Contact class and wrote each code in the class file. Below is this example.



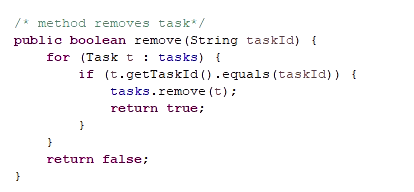
For the service class, I utilized a different method, one that I ended up using more throughout the project. On the Service class, I built the Junit test first. This way I knew exactly what I was trying to answer. This method made building my Contact Service class much easier. It helped me keep a clear mind and also test a lot more often than when I built the class. I knew that this method was better because I didn’t have to go back and fix my class like I did in the Contact class. This method was also better as evidenced by the coverage percentage. This shows whether my JUnit tests all my code and whether I built the code in such a way that it would pass the requirements. I know that I created effective code because, in the end, my percentage covered by JUnit test was 100%.

Because my JUnit tests were efficient, I could also see how it tested my code technically. In the Task Service class, I needed to test if the code made an array of objects and then would remove these objects. The code shown below passed because the test code created an array of objects that were tasks and then removed two of them. 

For the next code I wanted to check, the code would return false if the test built an array and then tried to remove something not in the array.



This all stems from the code below that lets us see that the code is created technically sound.



The above example also lets me know that my code is efficient. This means that I was able to reuse this code in the other services when removing an object. It helped me build the code in a way that allowed me to use the same code over and over. This was not the only part of my code that was efficient. I reused a lot of my code, helping me save more time in completing the project.

I learned a lot from building the test before building the classes. This has helped me understand what is truly needed for the code, and has helped me to determine if the process I am considering will work. I always thought building the class first was best, but I was wrong. Diligently studying the requirements to build the test should have always been my step one. The second step was then to build the test in a way that I could copy the base of the test and use it repeatedly, with the tests being changed every time so that it fits the requirement being asked. In the previous examples, I use testMethodDeletePass() and testMethodDeleteFail(). Both look the same, and I only changed one thing to help me test the class method. Breaking each test down like this really helped me understand the requirements and then be able to build a better class.

I did not use any experience based techniques as this class was my first introduction to Junit testing. There are acceptable cases where my knowledge of another project might be used to test code out. An example of this could be a past class or even in my job where I know and use VBA on a regular basis. Experience, in my opinion, can trump a lot of different types of testing. This doesn’t mean we shouldn’t test. We should always put code through testing before delivering to the customer.

Both of the above testing techniques can and should be used when they are able to be. Sometimes, I don’t need to build a test case out because we have built similar code out so many times. Testing this small part of the code can be built in a way that a large portion of code can be tested and the smaller portion is tested within it. This means that testing can be simplified due to past experiences. Building the test out first based on the requirement helped me see the helpfulness of doing it this way. It helped me stream line my thoughts so I can keep my code clean when building the classes.

Building this project as a tester, I had to build around my test cases and leverage them to help me understand what was wrong with my code. I am familiar with having developer bias of just creating code and making a working program with no regards to identifying bugs. Due to multiple classes and how service classes used the main classes, I needed to shift my mindset to a more granular approach to ensure each portion of the program performed perfectly. By the third set of tests, I found creating the test was fun and easy, so I am glad that I did not cut corners. I also changed my ways to be more disciplined in building the code correctly by using each test built from the requirements. This experience has taught me a great deal that I have already started implementing in my current career.