CS-320-T4208 Software Test Automation & QA

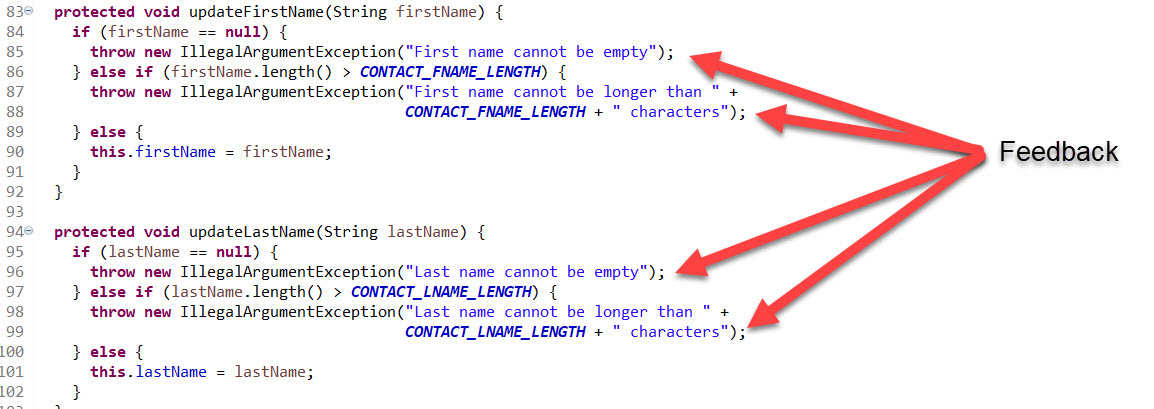
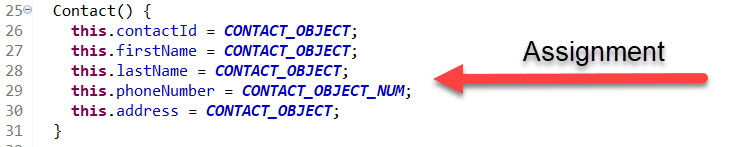
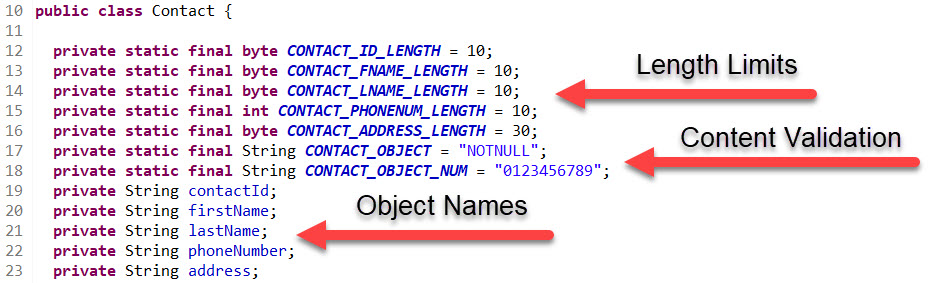
Project Two – Summary & Reflections Report

Computer Science Department

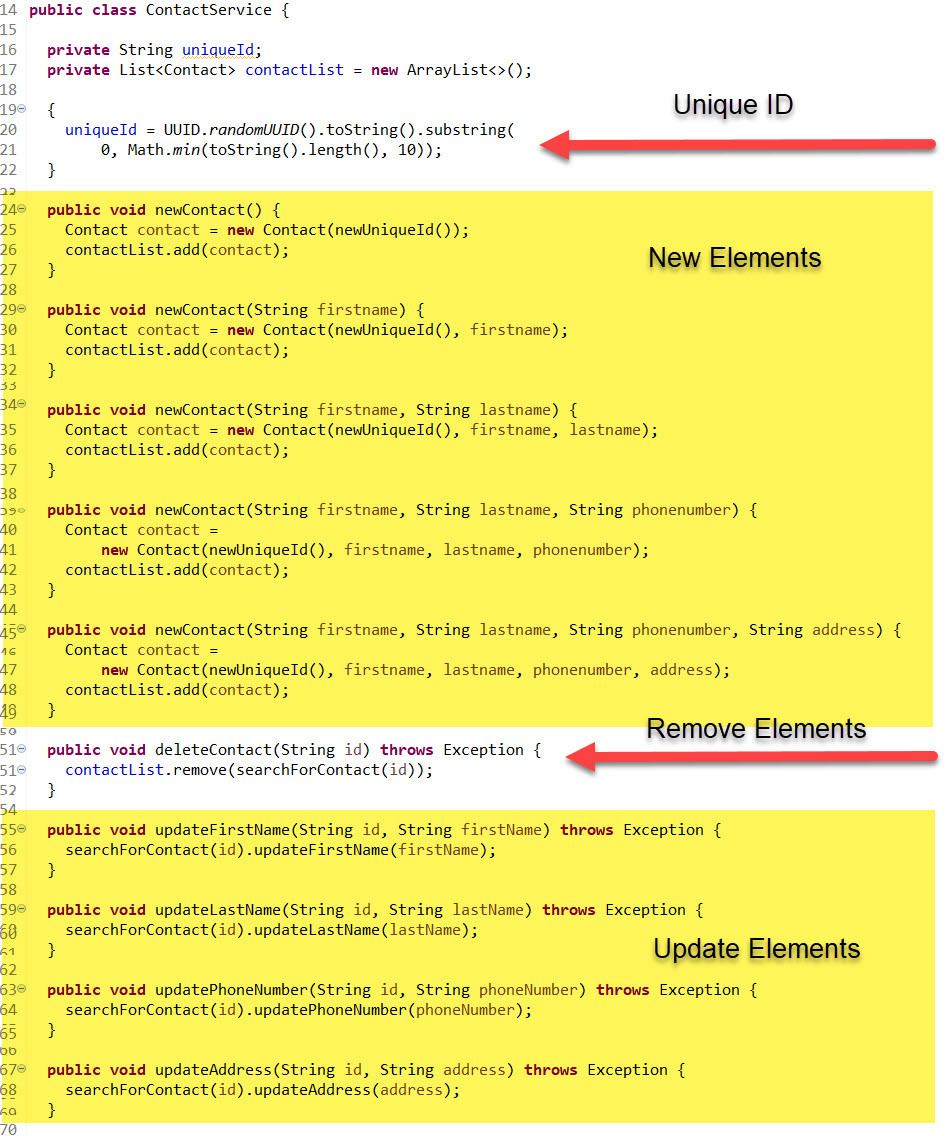
Southern New Hampshire University

Eric Farkas

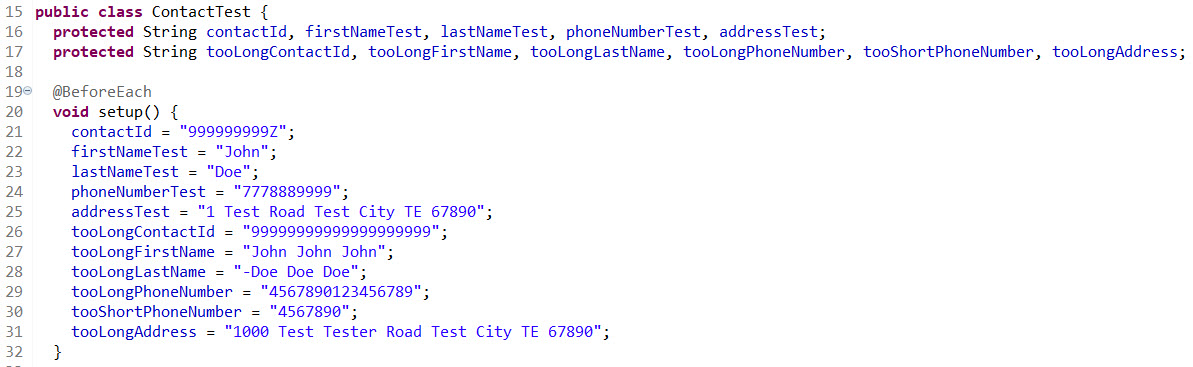
4/16/2023

This summary addresses elements of the approach used, compliance with the specification, testing quality, coverage and the efficiency of the code produced to support three elements detailed in the customer requirements. Compliance with the specification was easy as there was little ambiguity regarding functional requirements. Each element included class names for its user and service modules. Each class included object names for the key deliverables and guidance for the data type, character limits, nullity and uniqueness requirements. A specific example of this from the contact class requirements is: The contact object shall have a required firstName String field that cannot be longer than 10 characters. The firstName field shall not be null. This unambiguous statement made it easy to code the class with proper object names, control length, validate input and decline empty inputs. The code required several iterations to achieve something simple and efficient but ultimately, I defined the characteristics and checked them with an if loop that validates the input or throws exceptions when invalid. This loop provides guidance to the user should invalid data be entered.

The code examples provided are for the Contact element but the same methodology was used for the Appointment and Task elements. Using the same approach makes the code easily readable and updateable. Each element is paired with a service class in an effort to maintain records. The service classes respect the same rules as the initiating class but add specifications regarding how a record can be manipulated. All service classes include adding and deleting records but contact and task require updating the record and a unique ID. This set of requirements is very clear and easy to code. Each service class checks each object for changes during maintenance to modify a record properly. They also have a method to delete records and generate a unique ID where required. The ContactService class is included below as an example.



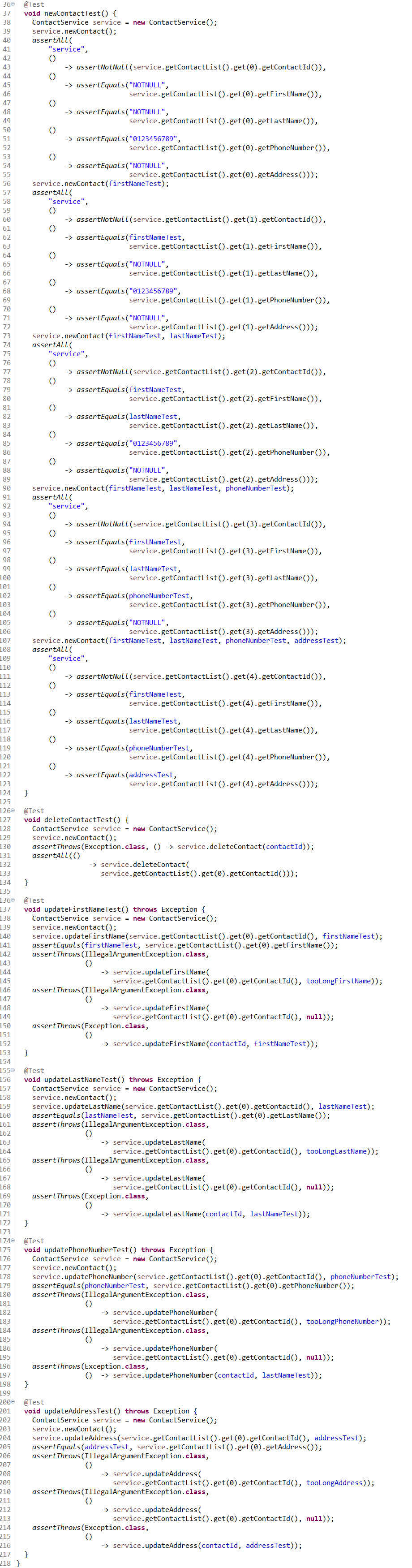
With the base code optimized and performing input validation, the JUnit testing code was greatly simplified. The strategy was to run through each element with properly formatted and sized input then switch to input strings that are too large and then too small. Testing alternate data formats was unnecessary as the code was manually reviewed and the validation works as expected. It was required to test each element and combination of elements to accomplish a value of 100% coverage. This made the test code slightly less efficient but we should not be concerned with the efficiency of the test code as it will not be part of the released code to the customer. The test parameters were established and illustrated in the code below.



The same approach was used with the Appointment and Task test code as the requirements are similar. To ensure the coverage was comprehensive, I mapped all possible scenarios where elements would change and ran the tests individually and in combination. The final result for ContactTest is illustrated below.



The testing of the Service classes needed to ensure that add, modify and remove were properly executed. The methodology used is the same as the base classes and the need for redundancy for comprehensive coverage is the same when dealing with add and modify operations. In a similar fashion, I crafted acceptable size, oversized and undersized testing data and built a testing structure based on the operations.



The same methodology was used on the other two elements. I am confident that the JUnit testing was adequate as no assumptions were made. The testing pattern was developed based on all possible scenarios and the coverage resulted in 100% for each base class and service class. I originally added some mutation testing for peace of mind but removed it once I obtained verification as it was out of the scope of the requirements. Although the test code is somewhat inefficient, the base code was made efficient by removing redundancy and testing loop conditions to ensure no possibility of infinite loops. The code review was most beneficial, but the JUnit testing provides a real metric with the coverage value. My concern is that I added extra lines of code to achieve 100% and I’m uncertain if those extra lines add value beyond obtaining a high coverage score. I believe the approach was reasonable for this project, but I must carefully consider this strategy for other projects.

The type of testing used is generally classified as white box testing. This is because we were focused on the internal functionality of the code and adherence to rigid specifications. Additionally, the development and testing were performed by the same person. The approach is loosely defined as incremental integration testing even though the approach does not test integration. This approach seems very efficient for small developments carried out by a single developer or a close team but I can see how this approach would become problematic in a larger development team. The testing for this project focused heavily on function. Many non-functional elements such as performance, security, scalability and interoperability were out of scope. No black box testing was performed during these milestones as the code and testing were performed by a single person. This forced additional caution in reviewing the test methodology as a second set of eyes would not be involved in the testing so no additional opportunities to catch bugs would be realized. This also meant that bias had to be eliminated from the testing. I used a randomization method to create input values to ensure the input validation and JUnit testing would address each random value. This took away my subject matter expertise and created various unexpected values.

It is reasonable that the white box testing is performed by the developer in this case. It raises the product quality as the developer has to plan more thoroughly to handle improper data. This type of testing could be cumbersome and time-consuming with larger developments and difficult to normalize in large teams. The fundamentals are still valuable and should be used as best practice when the scope is manageable. The input validation tests are always necessary. Too many documented vulnerabilities and attack methodologies exist to allow unrestricted data fields on public-facing systems. The strategy used should significantly reduce quality and user testing cycles and allow the product to release to production faster. Regardless of where these techniques fit best, testing and releasing a quality product is important. There are many contributing factors to answer the question of how much testing is enough but it has become apparent that anything available to the public or utilizing public infrastructure justifies extra testing and in many cases the use of a third party with specialized skills regarding cybersecurity.

Discipline is extremely important when testing your own code. It is difficult to think differently when developing and testing. I found taking a break and “switching gears " useful when transitioning between the two. It gave me an opportunity to flush the development code and refresh my mind about the requirements and possible inputs. This strategy worked well for small systems like this mobile application but I fear it would be more difficult or even impossible on more complex systems. I believe this is why complicated systems have dedicated testers. This may not be the primary reason, but it would greatly improve focus and maintain testing discipline. There are many methods to avoid technical debt. The methods and strategies in this report cover testing and optimized code. Some additional methods I’d like to gain experience with are agile development practices and issue tracking. I realize these methods are typically reserved for larger developments but I found some of the concepts helpful even though I deployed them using paper and performed multiple roles myself. I like working towards goals and measuring with key performance indicators. There are many opportunities to introduce measurement into coding and testing resulting in a better-quantified result.