3-2 Milestone Two Enhancement One Software Design and Engineering

Christian J. Kesler

Department of Computer Science, Southern New Hampshire University

CS 499: Computer Science Capstone

Professor Joseph M. Conlan

January 23, 2022

Software Engineering and Design Artifact

Artifact Description

The artifact I have chosen to showcase my Software Engineering and Design capabilities is the Appointment Service Java application. This project was created during CS-320 “Software Test, Automation & QA” last summer at SNHU, utilizing a series of Java classes with various attributes that were designed around the specifications of the client. The classes communicate with each other in a variety of ways, and I incorporated JUnit tests that comprehensively ensure all interactions are handled properly. The JUnit testing is thorough, offering multiple types of input and verifying expected behavior, both in nominal and exceptional cases. The JUnit tests have been verified to cover all relevant lines of code.

Inclusion in Portfolio

This artifact serves to showcase my competency with the Java programming language, the Eclipse IDE, object oriented programming, automated testing technical principles, and observation of programming best practices such as naming conventions and comment etiquette. Considering that Java, Eclipse, and JUnit are widely used in software development, I find this artifact to be extremely worthy of inclusion in my portfolio.

Enhancement Progress

I have added a variety of consistent and descriptive comments throughout the Contact Service java classes and JUnit tests. I also replaced many (if not all) of the hard coded strings and integers used in comparisons with descriptive variables that can be changed from one accessible location near the top of each file. I found repeating error statements for incorrect input both in the constructor and setters quite troublesome. While the client did not request such functionality, I chose to streamline this by creating a series of verifier methods that are passed input, check the validity of the input for a specific object attribute, and return true or false along with an error statement if applicable. This allowed me to greatly simplify the constructor and setters by using this method as the condition within an if statement, assigning the value to the object if true.

I did hope to create a loop that would run all the JUnit tests rather than listing them out individually, but that proved to be less efficient or practical than the current layout. I was successful in condensing the class functions using the input verifier methods, and in improving the readability of all files with improved commenting strategies. I am quite pleased with the progress that has been made thus far, but will likely continue to make small improvements should I have leftover time in later modules.

Enhancement Reflection

While enhancing this artifact, I realized that there was a surprising amount of variation regarding the format of the JUnit tests. The service class tests had up to ten filler entries to test the constructor and manipulate in later tests, which was more than necessary. Meanwhile, the regular class tests operated a single instance of the constructor each time, only handling a single object at a time. This shouldn’t be surprising given the nature of the service classes, which are meant to hold the appointment, task, and contact objects in lists. It does make me wonder if a redesign for the service tests might be in order, so that the general pacing of the JUnit tests are similar across all files.

I also noticed that some of the things I wanted to improve were already near their ideal state. The class constructors, for example, had a series of if else branches that checked each input value for validity before creating the object and assigning the attributes. I thought that was inefficient at a glance, hoping to streamline the input screening process. While I was able to do so from a readability perspective, the actual program performance was not nearly as suboptimal as I thought. The input must be checked for validity before being using to instantiate an object, else erroneous entries would not be handled. Realizing this, I instead focused on improving readability and code compactness while minimizing repeated statements, which proved quite rewarding.