John Mobarry

CMSC 204

2/12/2021

**Assignment 1 Learning Document**

This assignment was initially a struggle to get to implement but after reviewing the textbook and spending time reviewing the description document/JavaDoc I gained a greater understanding of the topics we are learning in class. Prior to the implementation of this project I had a vague understanding of exceptions, but now I feel much more comfortable in writing a program that handles this topic. This project also helped me understand how to step back when confused with the specifics of an algorithm you need to create and that often pen and pencil application is better than hammering away at the keyboard. This assignment also refreshed my comfortability with JUnit testing and getting JavaFX to run correctly in the IntelliJ environment.

While working through this assignment I had to get a greater understanding of how to structure my code regarding the six exceptions that would be regularly thrown while trying to confirm that the password is correct. My first concern was not understanding the difference between throw and throws within the context of java. To address this issue I returned to the textbook and picked up the exceptions interlude and determined that throw is used to throw a specific instance of the exception within a method and throws is used to catch a specific type of the exception. I had to create all six of the exceptions InvalidSequenceException, LengthException, NoDigitException, NoUpperAlphaExcpetion, NoLowerAlphaException, NoSpecialCharacterException that extend the exception class in order to handle the potential issues with entered passwords. Each method could encounter a certain number of these exceptions while testing the passwords characteristics and the specific exceptions would be thrown within the method if certain criteria were met, which would be caught outside of the method and print the resulting message. It is important that the methods “throws” the specific exceptions and not just all exceptions as you want to have the message that is printed reflect the specific issue in order to provide feedback to the user.

I faced significant roadblocks with one specific aspect of password validity: to determine if the password had a special character within. At first I tried tokenizing the String in order to view the individual characters and confirm if they were not digit, upper case, or lower case that they must be special characters. This method worked in some aspects of the JUnit tests but it did not result in runs that would pass all forms of the tests. I tried to see if within the language there was the equivalent of Character.isDigit(String) for the special characters, and to my disappointment there was not. I reflected on what I learned in CMSC203 and realized that in one of our first lectures we discussed ASCII values and that the special characters can be represented in numerical value ranges for comparison. With this discovery I took out a notebook and pulled up the ASCII table and wrote down the numerical ranges that represent Special characters (33-47, 58-64, 91-96, and 124-126). To determine if there were special characters within the entered string I created a for loop to compare each individual character of the string against those ranges and if any of the characters were in those ranges I set a boolean value equal to true to reflect there is a special character. If the said boolean value was determined to be false after running through the length of the string then the program will throw the NoSpecialCharacterException. Sadly figuring this out took me much longer than many other aspects of the project as I spent too much time trying to fix the same issue with the same approach (what is the definition of insanity again).

This project also had me refamiliarize myself with processes I was comfortable in Eclipse with within the newly installed IDE of IntelliJ (it was more than worth the growing pains). Unlike in eclipse there is not a dedicated right click option for you to create a JUnit test file for a specific class in IntelliJ. Instead you create the JUnit file like you would any other class and then go through the process of importing the libraries that are required in order to run your JUnit test. I also had to reconfigure my understanding of JavaFX while in intelliJ as I had to import the libraries again and ensure the folder dependencies were set up. Unlike in Eclipse IntelliJ accounts for this requirement within its GUI and allows me to locate the required folders through the windows search tool instead of having to type out the path directly (which has often resulted in errors for me while working within eclipse). In general my switch to IntelliJ from Eclipse for this project has resulted in a more smooth experience and less non code related errors.

Overall I enjoyed the process of going through this assignment to get a greater understanding of the topics of Exceptions. Exceptions were a topic that greatly confused me during CMSC 203 as we never truly implemented them in a way that required us to actually understand the topic (they were usually existing in the provided code from the instructor). Now that I was tasked to go through the process of creating, extending, and implementing the exceptions required in this assignment I now have the topic committed into my brain.