**Design / Pseudocode / Student To Do**

1. Prior to coding, what did you think about, and what did you do?
   1. In several paragraphs, capture them here
   2. That’s all I am looking for

**Test Cases**

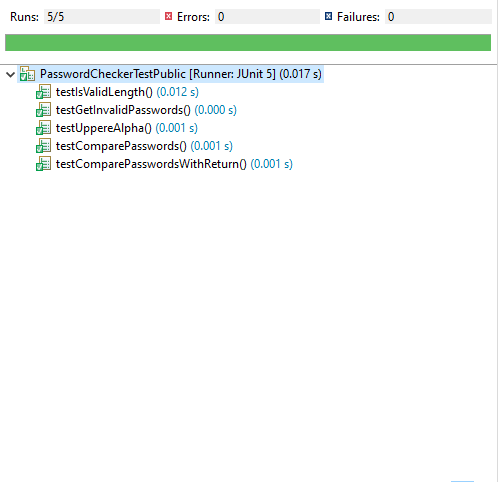
1. Test length, upper/lower alphabetic requirements, and starts with digit requirement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prompt | Input | Output Expected | Output Actual | Screenshots |
| Password? | short |  |  |  |
| Retype password | short | Length must be greater than 6 | Length must be greater than 6 |  |
| Password? | 2short |  |  |  |
| Retype password | 2short | Must contain uppercase | Must contain uppercase |  |
| Password? | 2SHORT |  |  |  |
| Retype password | 2SHORT | Must contain lowercase | Must contain lowercase |  |
| Password | 2shorT123$ |  |  |  |
| Retype password | 2shorT123$ | Invalid password cannot start with digit | Invalid password |  |

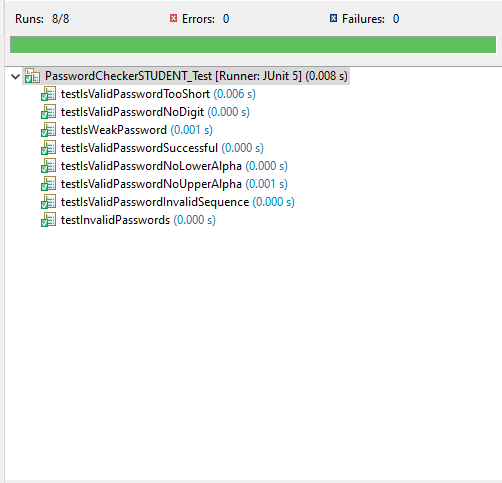
1. Test matching passwords + exception order priority
2. Test numeric character requirement + special character requirement
3. Test no more than 2 of the same character in sequence requirement + File passwords

JUnit Tests

1. Public/GFA tests



1. Student tests



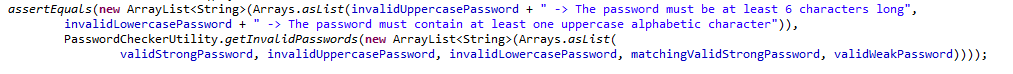
**Learning Experience**

From this project, I learned about how to use lambda expressions in assertThrows(). That’s because assertThrows() takes an executable block which is a functional interface meaning it only has one abstract method. I’ve used lambda expressions before for JavaFX programs, but this project allowed me to refresh my memory. The test allowed me to use a lambda expression to simplify my code as shown below.

Text

Description automatically generated

I also refreshed my memory on how to use regex expressions in Java because I thought it would be easier to use regex to check for special characters rather than through a complex switch() or for-loop. I also got to practice using ArrayLists and instantiated them through the Arrays.asList() built-in method so that I didn’t have to add elements in one by one as shown below.



I also learned about how to properly extend Exception classes by giving users the option to either use the no-arg constructor with a default message, or a String parameterized constructor to allow custom messages.

A challenge I faced was with regex. I discovered that “/” is treated differently by both Java and regex. Hence, if I want to see if my password has a “/” (backslash), I would have to represent it as “////” because “/” is an escape character so “//” makes it a backslash in Java. However, “//” represents an escape character in regex, so “////” represents a single backslash because you have to negate both the Java and regex escape characters.

I think I did well on this project. Everything works well and matches the Javadoc as best as possible. Both my custom JUnit tests and the GFA tests passed perfectly, and the output/feedback is as expected. Exceptions are tested in the correct order and there are no bugs that I could find. If I were to do this project again, something I would do differently is work in smaller increments throughout the week rather than cram the workload all into one day because when I did that, I had to remind myself how my progress on the project was because I hadn’t worked on it in a while.

One enhancement I would’ve made to this project, if I were the instructor, would have been to have more updated and better documented Javadocs because some of the methods in the Javadoc don’t make sense or aren’t possible.

**Additional Enhancement**

My first additional enhancement was making sure that isWeakPassword() called isValidPassword() first. This was unnecessary because the FX driver calls isValidPassword() before isWeakPassword(); however, I thought it was a good addition because if someone else wanted to use the PasswordUtilityChecker class, it would seem more intuitive that isWeakPassword() would check if a password was valid before declaring that it was weak. Hence, my additional code made the method more reliable for other driver classes.

My second enhancement was that I made the PasswordUtilityChecker constructor private. This serves two purposes. First, the constructor is private because I don’t want the user to waste memory by instantiating a PasswordUtilityChecker class; the class has all static methods so its better to call methods through the class’s name. This is a technique I saw used by Java’s Math class. The other benefit of making PasswordUtilityChecker’s constructor private is that it prevents subclasses from extending it. That’s because the subclass’s constructor wouldn’t be able to call super() as the super constructor is private. I think this is beneficial because PasswordUtilityChecker is a class built specifically for the given FX driver; it is a class that should not be extended because it was built to work only for a specific driver class (if a user wanted similar functionality, it would be better for them to create a new class that fits the needs of their driver class).

**Assumptions**

1. The user will be using JUnit 5.0 and can run JavaFX
2. The user will enter characters from the English alphabet
3. Exceptions should all extend RuntimeException because the provided LengthException class extends RuntimeException
4. hasSameCharInSequence() returns true if the password passes the requirement, even though it’s counterintuitive, since that’s what’s written in the Javadoc
5. Requirement checker methods such as hasUpperAlpha() either return true (pass) or throw an exception (fail); the Javadoc sometimes states that a method should return false when failing a password requirement but also throw an exception, which is not possible as throwing an exception and returning a value both end method execution so they’re mutually exclusive
6. isWeakPassword() either returns false if its not weak or valid, or throws a WeakPasswordException if it’s weak and valid
7. The requirement that the password doesn’t start with a digit doesn’t throw an exception as no such exception exists in the Javadoc; it simply makes isValidPassword() return false if the password starts with a digit, and it’s checked last