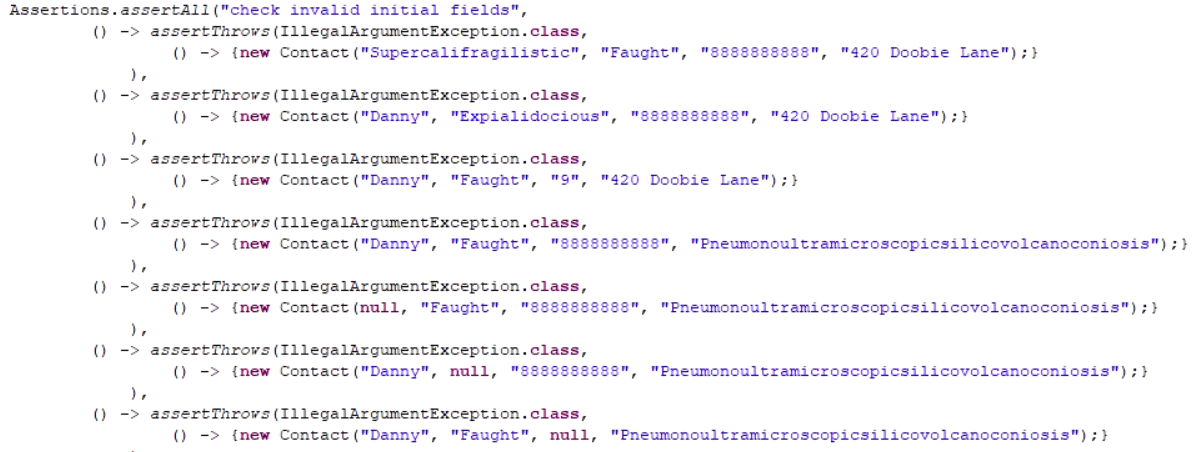
1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.
         1. The software and tests that I wrote were tied to each of the requirements listed. A specific example of this is the method I wrote to validate strings. Each part of our application required validation of some kind to ensure that fields were of a length that were not too long or too short. I created a shared method to assist with this and avoided duplication among each field.
      2. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage
         1. Using Eclipse, we can see very clearly what parts of our program our tests are covering and where they need to be fleshed out more. We can where each branch of our logic is tested due to red/yellow/green highlighting that takes place upon each method.
   2. Describe your experience writing the JUnit tests.
      1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.
         1. The magic of JUnit of course comes from its assertions. It is a very simple paradigm: put something in, list what you expect out – does it pass or fail? Using this method, I was able to successfully determine whether a function was behaving the way that I intended. As an example, here we see a snippet of where I was expecting an exception to be thrown:



* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.
       1. As mentioned in previous assignments. I made copious use of the “assertAll” method that is built into JUnit. Here I was able to test all methods at once that I felt could be grouped together. Here I am checking what happens for all invalid initial fields in one go:



1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.
         1. For this project, I wrote what are known a “unit tests” these are tests where we are directly testing the methods or functions that the developer creates. These are straightforward tests, gauging if a desired output is achieved with given input. These types of tests are typically written by the developer that wrote the methods.
      2. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
         1. There are other types of testing as well that I did not use. Regression testing is a type of testing where the practical uses of the program are tested again after a feature or enhancement is released. This ensures no breaking changes are implemented into the program.
      3. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
         1. To be clear, all types of testing have their uses. In large applications that have practical use for a large audience, hopefully all methods are used to some extent. Testing in some regard should be used for every application, but there may be a point of diminishing returns for smaller projects. As mentioned, I didn’t do regression testing on my application because it was so small. Assuming the parts were interrelated at some point if we took the project further, regression testing could be minimal.
   2. Mindset
      1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.
         1. Fortunately, this was a simple program, but when performing the tests, I quickly learned how much of a task this could be for something more complex. A test needs to be written in a way that mimics what a method will receive. This isn’t always clear, especially in languages that aren’t typed like Java. For example, if the method you are testing is expecting an integer, but your test passes it a string you may receive a false failure case for that method because the test was written incorrectly. Cases such as these need to be considered during test design.
      2. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.
         1. Bias can be a concern for developers. One may think that they are immune to bias. The tests either pass or fail. But this is not all there is to consider. A test could be written to pass when it should fail. A developer may have good intentions but is failing a fringe case and is at the same time under a deadline. A developer should test his/her own code but should not be the only person to test it.
      3. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.
         1. Being disciplined will produce quality code. This is true about most subjects. Cutting corners always leaves a greater chance for error. This will lead to defects and more development time later. In the field, it is important to be remain diligent when testing your code. If one works with testers, gather feedback from them and learn from mistakes.