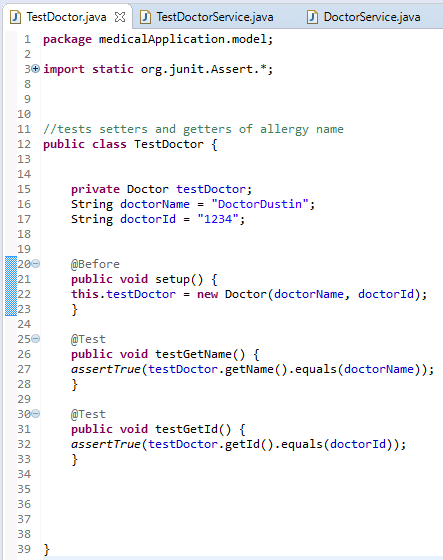
Dustin Haugh

CS 320 Milestone Two

How many JUnit tests did you create for each of the features (i.e., Doctor Information Feature, Medical Records Feature, and Allergy Find Feature) and why? In other words, what was your level of testing for each of the features? Explain your reasoning based on the best practices you have learned in the course.

**In all, I completed 13 test classes with altogether 33 test methods. Most of them were not required in the definitions and requirements list provided by the assignment, but rather a way for me to practice with simple tests while also relearning Java.**

**I started out with simple tests to instantiate a model classes. This is an example of testing a doctor class.**

****

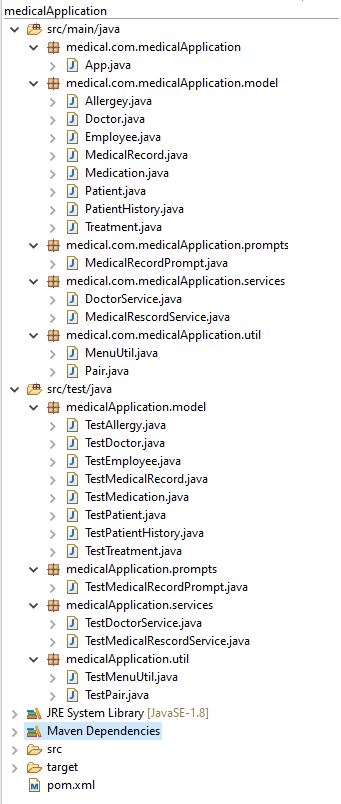
I have read that people like to have 80% of the methods tested for. In this case, I am mostly just testing setters and getters for objects used in service classes later. I believe that I was thorough in my testing, at least for my first time working with JUnit.

While the Doctor model class did not require too many tests my Medical Records Service class required six, one for each method. I have read that JUnit best practices talk about setting up a test for each method. I did not complete quite that many, but I have learned how to structure my JUnit tests so that it is not difficult to understand what I need to do.

How did you organize your tests for each of the features? For example, did you group any of your tests into test suites for any of the features? Why or why not? Explain your reasoning based on the best practices you have learned in the course.

I did not organize my tests into test suites, but I can. At this point where I am only developing JUnit tests and I may add more later, I felt it would be fine not to add test suites. If I were completed with my JUnit tests and I planned on going back into development where I would need to test everything quickly during each iteration of my project I would absolutely add test suites to make that more efficient.

That being said, I felt it was important to keep all of my test classes organized in the same manner as the source code. So, it would be organized like this:



It is best practices to keep tests within the src/test/java file, and I like to create subfolders that match the class types (model, services, etc) of the classes I am testing. This keeps everything organized and easy to troubleshoot.

Did you address all the software requirements? Describe how you met each of the requirements from the test plan (listed below).

* Doctor Information Feature Requirement: The system shall allow the user to log in and add a doctor to the list of doctors. Doctors’ names do not have to be unique, but doctors’ IDs should be unique.
* Medical Records Feature Requirement: The system shall allow the user to add a medical record to a patient.
  + Add a patient.
  + Add a medical record with treatments, medications, and allergies.
    - When you create a medical record, it is necessary to create a patient history, which will contain 1 to many treatments, 1 to many medications, and 1 to many allergies. Medications cannot be assigned to a patient history unless there has been a treatment first.
* Allergy Find Feature Requirement: The system shall allow the user to search for allergies and print all patients with allergies.

I was able to test for all of the software requirements listed above. Unfortunately, many of these did not pass my JUnit tests.

Specifically, I had four tests that did not pass, and they go as follows:

**TestMedicalRecordService.java**

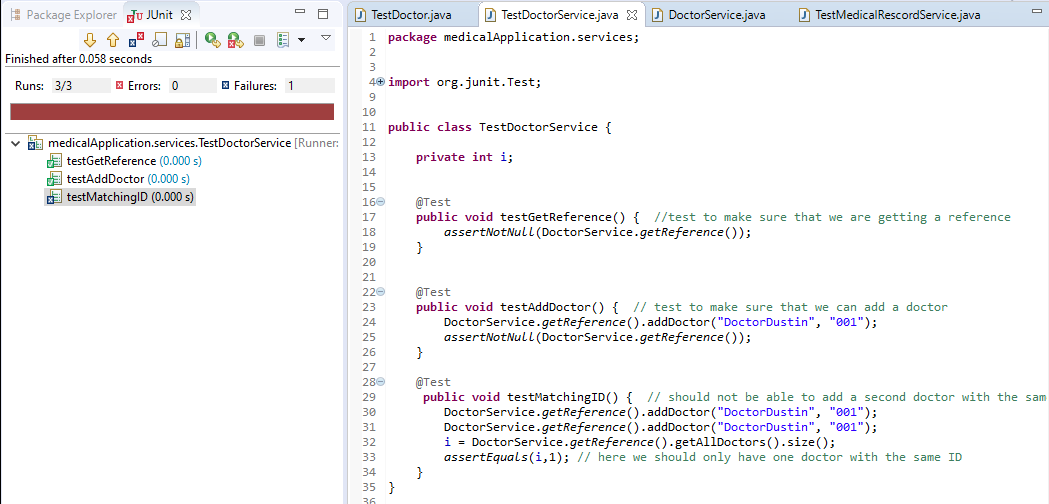
testmedicalRecordAssignedToPatient() - I cannot return a medical record for a specific patient

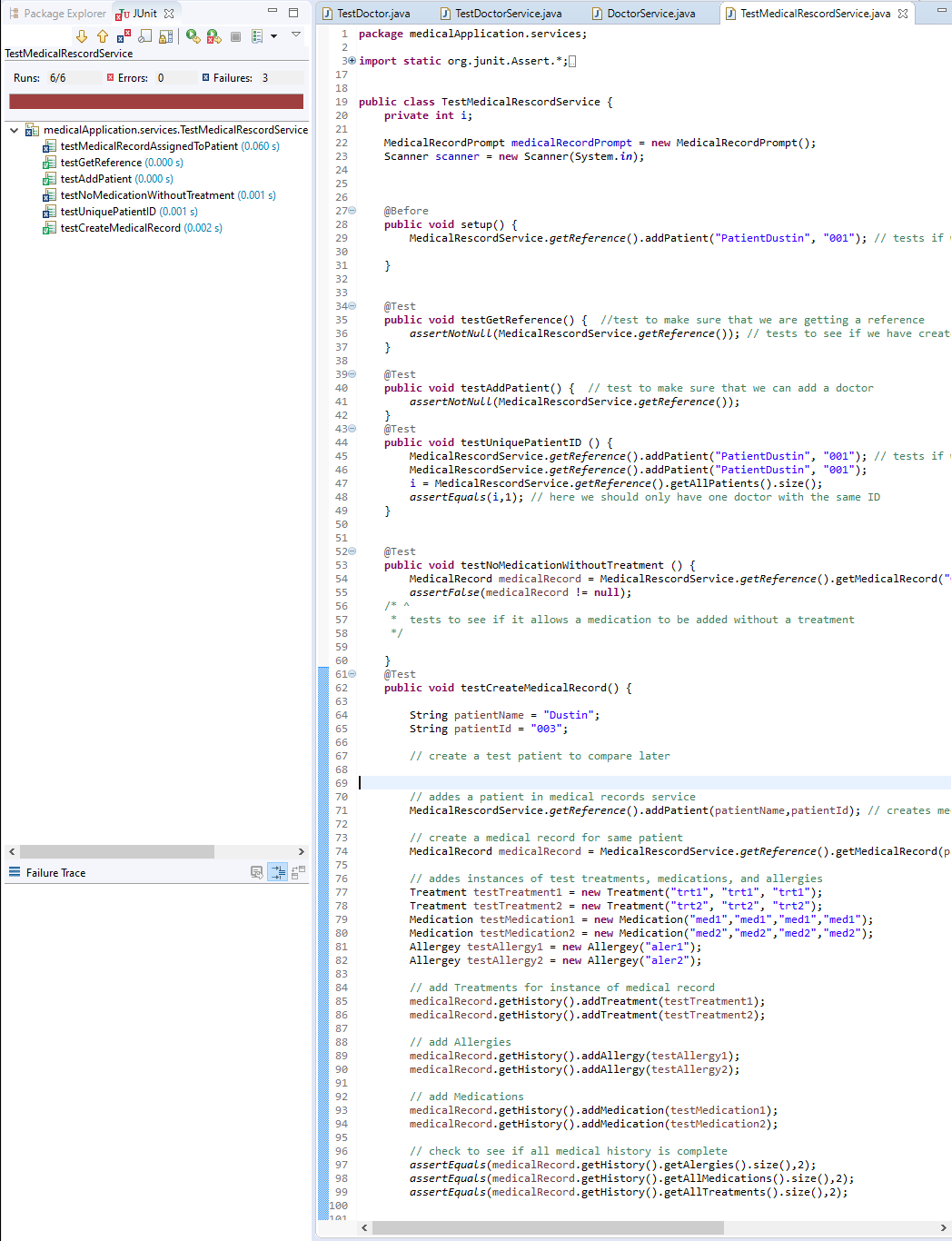
testUniquePatientID - The system allows me to add two patients with the same ID

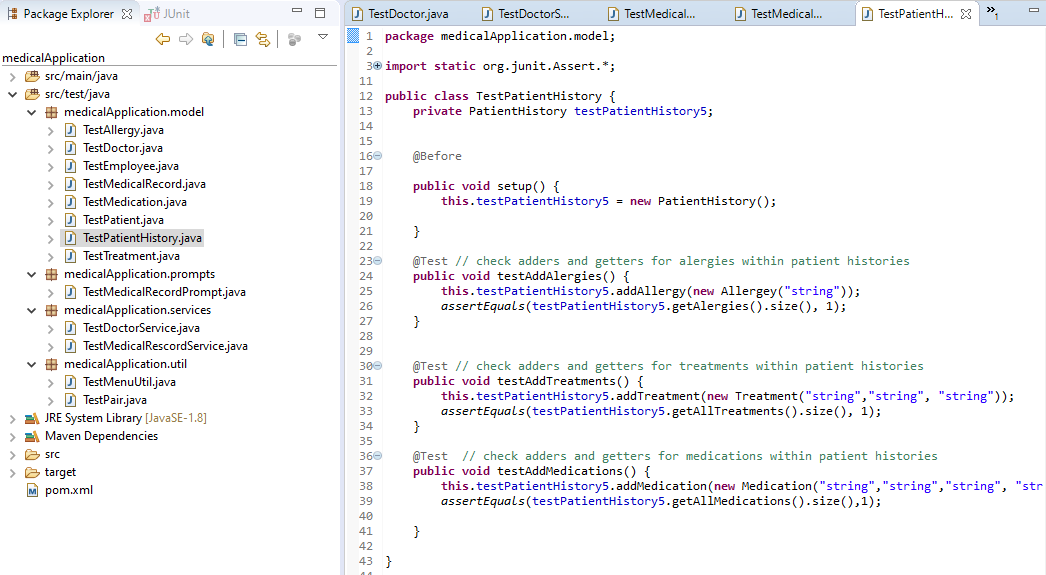
testNoMedicationWithoutTreatment() – A medication can be added despite not having a treatment for

**TestDoctorService.java**

testMatchingID - I can have two doctors with the same ID





How did you ensure that your code was technically sound and efficient? Include screenshots of your code demonstrating that it is the following:

* Syntactically accurate (style and syntax)
* Logical (how well your program satisfies the requirements of the assignment)
* Concise (how thoroughly you have tested your own code)
* Modular (how you organized your tests)

Defend the effectiveness of your tests by answering the following final questions:

* Did you find at least five errors? How many did you find in total?
* What were the errors?

I used the library of JUnit testing methods, or at least a number of the many options I had.

<http://junit.sourceforge.net/javadoc/org/junit/Assert.html>

For syntactically accurate style and syntax I was able to utilize previous assignments to test assert methods but also see how different testing methods were used for model classes vs service classes. I’m going to have to play with these quite a bit more to get better at using them.

Unfortunately, I only found the four errors that I listed above.