Matt Knutson

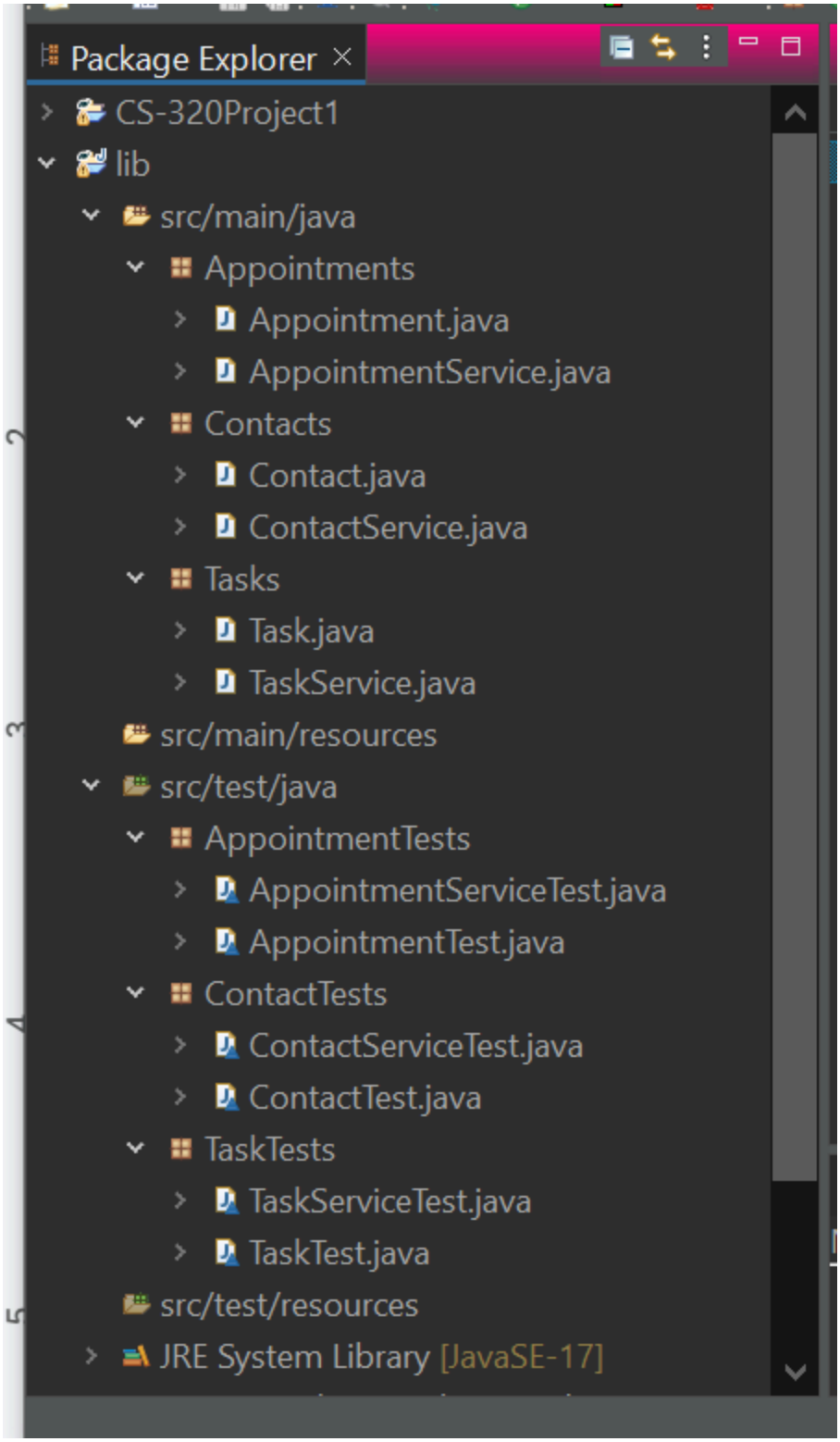
CS-320: Software Test Automation & QA

SNHU - Instructor Tuft

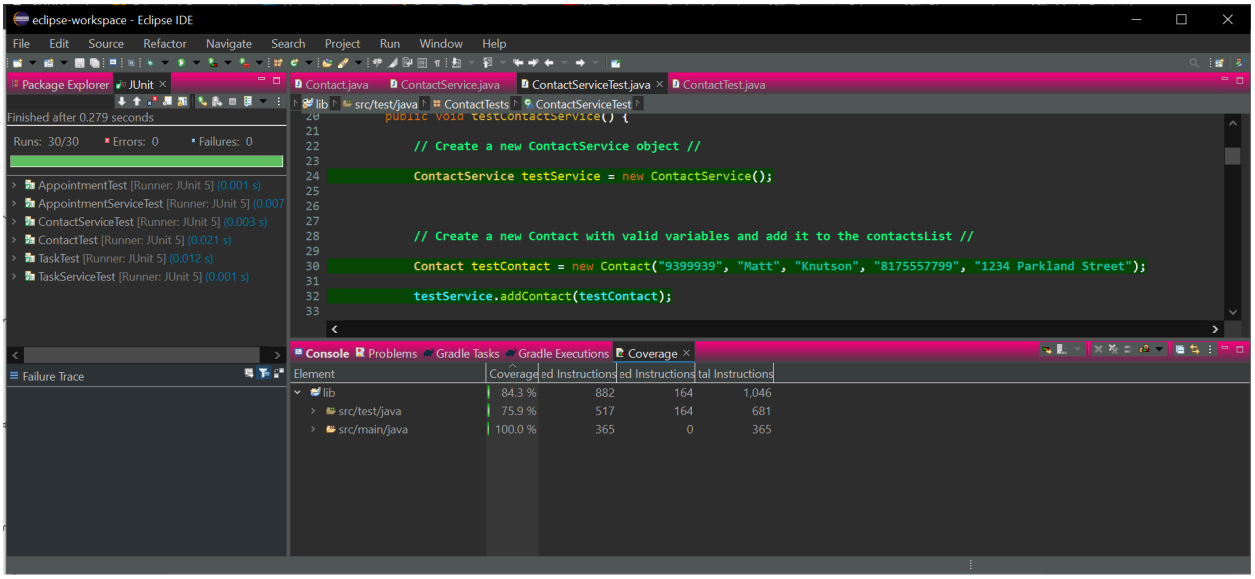
February 23rd, 2024

Week 7 - Project 2

The finished project required that there be a contact, task, and appointment class. Each one of these was required to have a service class that would handle adding, updating and deleting objects of that class. I was able to successfully build all of these and then test them by building out their test cases. I kept the ID of each object unchangeable by declaring it a ‘final’ variable in each class. In order to ensure that I tested all aspects of the code, I built a test for every method in each class. This sometimes required two tests, as was the case when I tested the phone number to ensure that it was exactly 10 digits.


Package Explorer
CS-320Project1
lib
src/main/java
Appointments
Appointment.java
AppointmentService.java
Contacts
Contact.java
ContactService.java
Tasks
Task.java
TaskService.java
src/main/resources
src/test/java
AppointmentTests
AppointmentServiceTest.java
AppointmentTest.java
ContactTests
ContactServiceTest.java
ContactTest.java
TaskTests
TaskServiceTest.java
TaskTest.java
src/test/resources
JRE System Library [JavaSE-17]

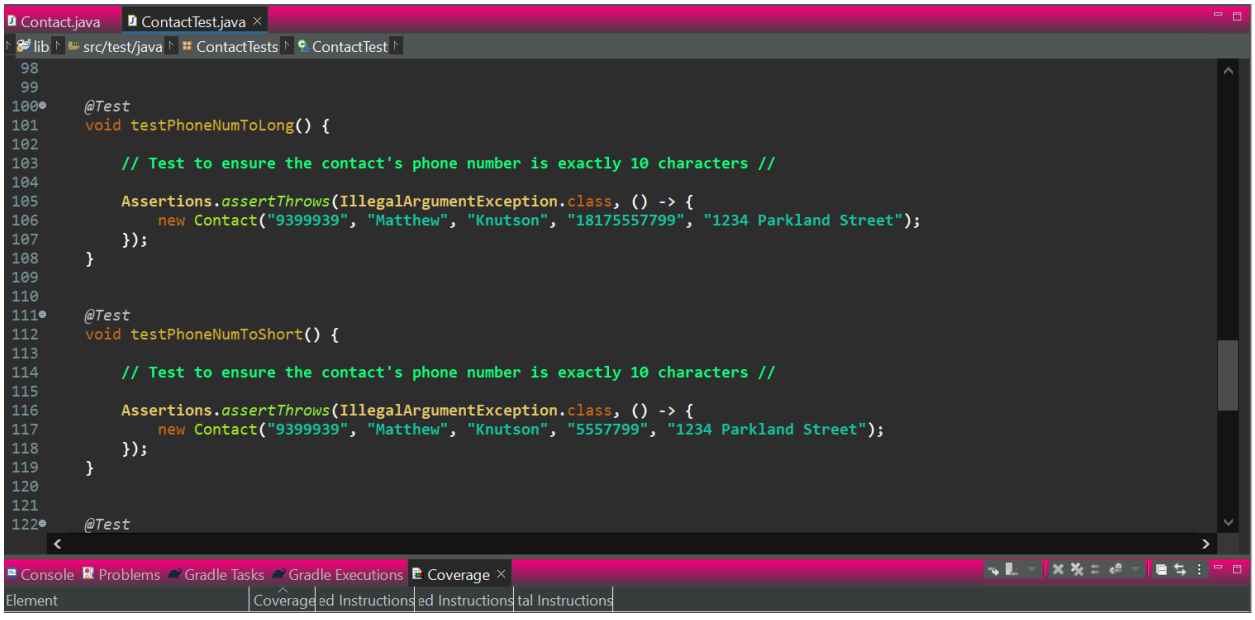
For this being my first experience testing code and using JUnit, I would say that I did a relatively good job covering all of the required areas. I was able to successfully test all the methods that I built using JUnit and my overall coverage was 84.3% for the entire application. Each method was designed to throw an exception if the incorrect data was entered and each test entered the incorrect data into that method. JUnit confirmed that all of the proper exceptions were thrown and that the methods worked in accordance with the requirements.

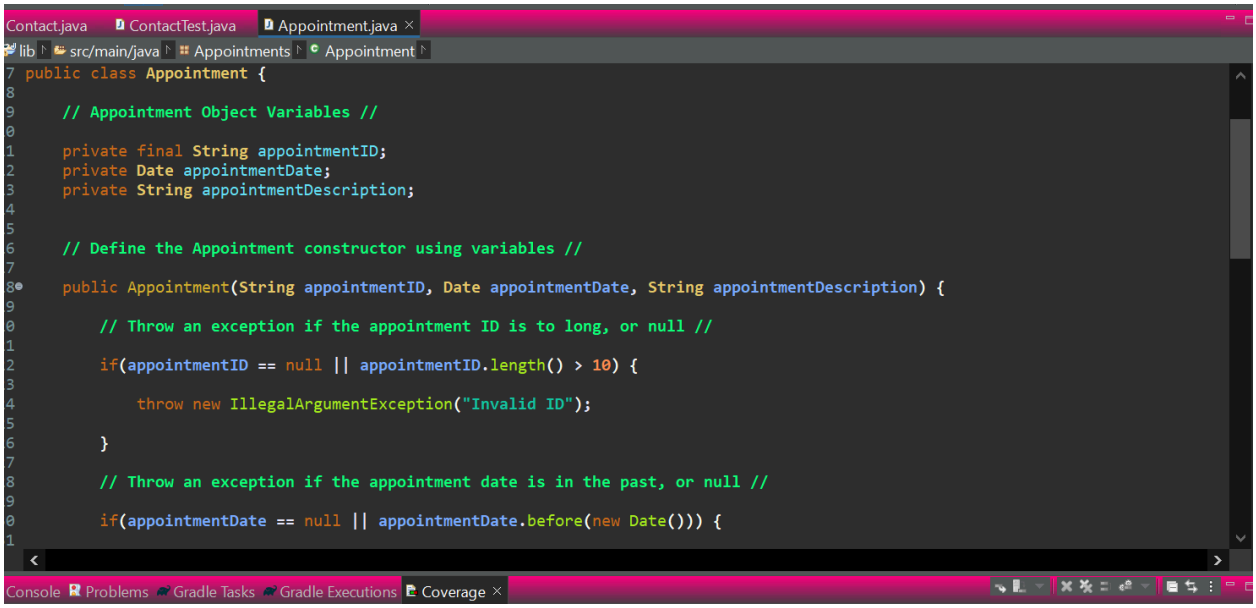


My overall experience using Junit was a positive one, although I met many challenges in the beginning. I originally had a difficult time getting JUnit to run on Eclipse, but once I learned how to set up a Gradle project properly, it was very easy to begin testing my project. I also ended up retyping the code for the Contact class several different times after trying different

programming styles concerning the use of Assertions and Annotations. All of this was rather frustrating, but led to a much better understanding of the code I was writing.

To ensure the code was technically sound, I used the proper exceptions and added messages for user clarity. I tried to document the essential lines of code, that way other programmers can easily understand the application. I also tested every angle of the requirements. For example, the phone number needed to be a specific length, therefore I tested that it wasn't too short or too long. The other tests only specifically needed to test if the string was too long. To ensure the code was efficient, I built the methods very simple and only with the amount of code needed to accomplish what was required. I also kept the code neatly bundled together and easy to read for the sake of other developers, but also to enhance my understanding of how all of the code works together in the finished program.





For this project, the main testing technique that I used was unit testing. I built a class and then tested each of it's methods without ever integrating it with other objects, an interface, or a database. This technique is used to ensure each basic component, or class, works properly before bringing it together with other parts of the program. This makes finding bugs and defects within the system much easier since the software is tested and then pieced together, rather than an entire system being developed, brought online, and then tested at a later date.

There are many software testing techniques that I didn't utilize in this project. Since this was just an introduction to testing for me, I wanted to keep the process relatively basic. In the future, as the size of the programs I'm testing grows, I can definitely see using parameterized testing to cut down on the amount of code I write for similar methods. Also, as I grow more knowledgeable about testing software, I will be able to identify anti-patterns and code smells much quicker. This is a testing technique that will take time and experience to effectively use.

Other techniques like equivalence partitioning and boundary analysis will also be great tools in the future as my experience grows in manipulating inputs and outputs.

I initially began the project with an open mind, understanding that I needed to test all the angles of each requirement for the given methods. I tried to keep an eye out for less obvious tests that I could implement as I wrote the code. But, even as cautious as I was, I ended up having to retype and reevaluate a good portion of my code. This was also my first experience building 12 different classes. As the program grew, it was essential for me to document how the code worked together in my comments in order to continue to understand the program's complexity. Although having to retype a lot of code was exhausting, I can say it helped clear up some confusion in my mind about how classes and methods work together. Also, having to set up a Gradle project and import it's dependencies several times allowed me insight to the many different types of projects that can be created in Eclipse and other IDEs.

Since I am still relatively new to coding and still in the process of understanding basic structures, I found it very easy to doubt myself during testing. Even as my skills, knowledge , and confidence grows, it feels appropriate to keep a mindset that I am still in the process of learning. Although this might be hard to manage as time goes by, it seems like it will help to eliminate a lot of bias I might have that my code is perfect due to my experience level.

It's very important to write clear and readable code and to leave the proper comments in your code. Writing code comments is extremely helpful for me as a new developer when acquiring an understanding of how each method interacts with one another, as well as the classes and objects. It's also extremely important to know how to build, and use, your tests properly while at the same time using your logic and insight to ensure no unexpected errors occur in the

system. In the future, I will avoid technical debt by being as detail oriented as possible and by testing my code as it is written, unless otherwise informed not to. I can definitely see the value in testing while developing, over testing at the end of development. The testing process itself adds very little time to the overall coding experience. This is a much better sacrifice than the countless hours and dollars a system could take to fix if issues are found later down the line.