Afahri Kerr

CS – 320

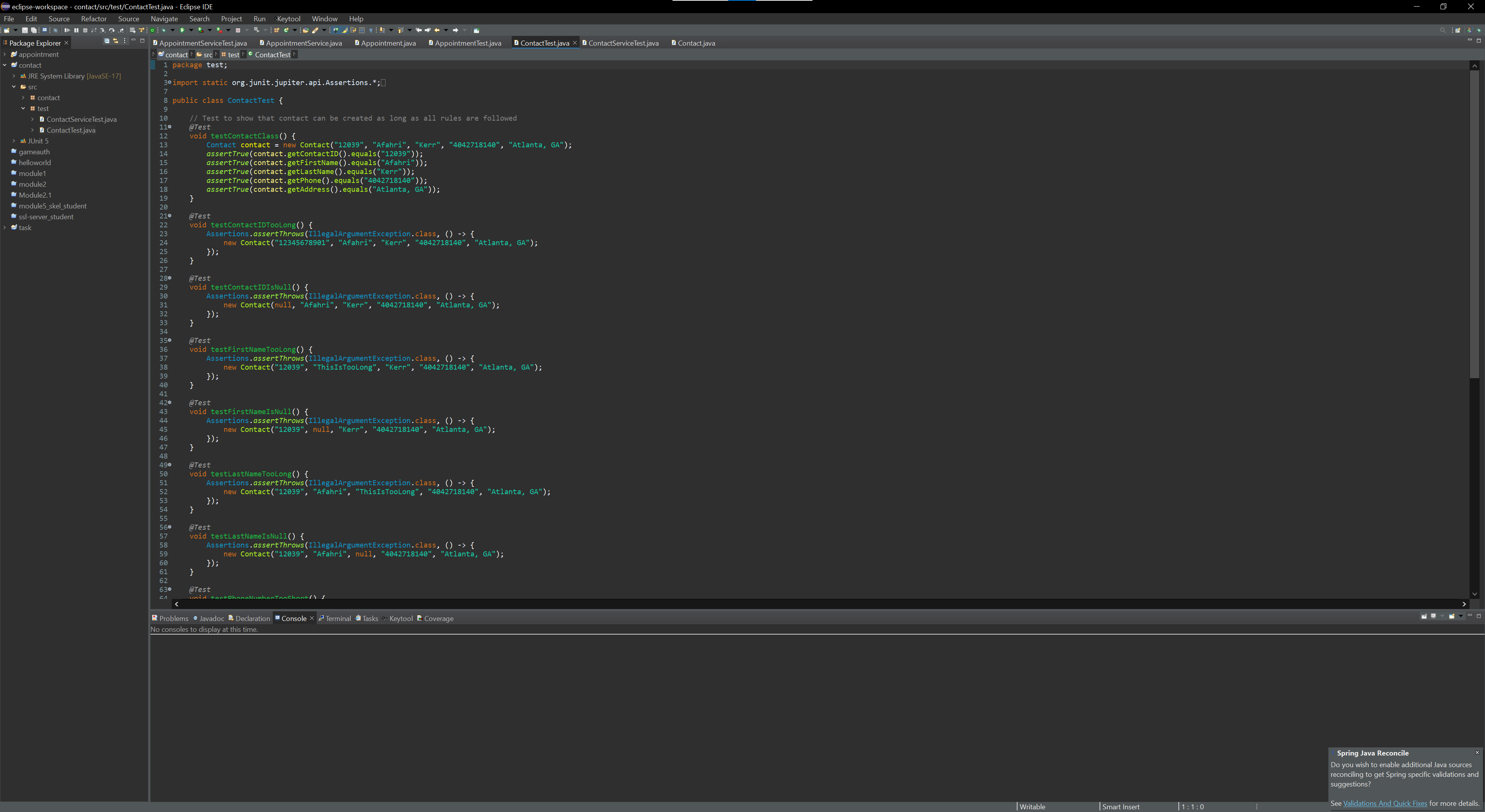
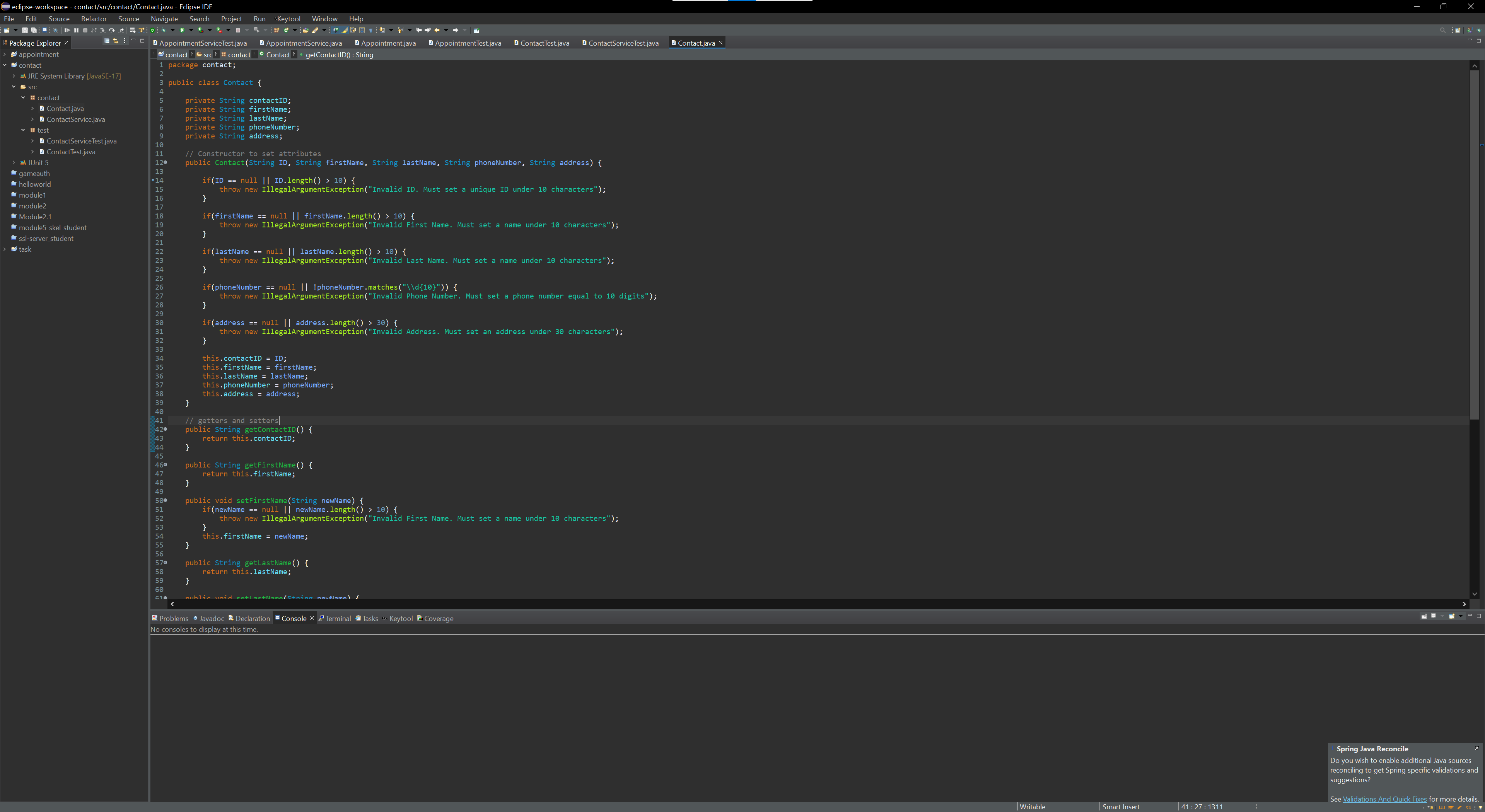
Professor Omar Toledo Lopez

08/12/2023

**Summary**

My testing approach to this application was very simple. I would first look at what the requirements were for each feature and tailor the application to those specifications. I would first make sure that my functions worked with the edge cases. For example, my taskID should only accept up to 10 characters so I created one with exactly 10 characters (Task newTask = new Task("2987487328", "Arbitrary task name.", "This is a 50 character long description of a task.");). Then I would test that it wouldn’t accept any more characters by adding 1 more (Assertions.assertThrows(IllegalArgumentException.class, () -> {new Task("29874873280", "Arbitrary task name.", "This is a 50 character long description of a task.");). I would also test it to make sure that it couldn’t be created as null or be updated to null.

I know that my tests were effective because my test coverage for the contact, task, and appointment classes came out at 100% while my test coverage for the taskService, contactService, and appointmentService classes came out to more than 99%. My original classes covered all the requirements while the test cases made sure that the functions worked as intended and would throw the correct errors if misused.

To ensure that my code was technically sound I followed the industry standard practices. I first made sure to use the appropriate syntax for Java and used camalCase to improve readability. I also commented my code where necessary to make the code easy to follow along with and avoided complex code. I also made my error messages describe exactly what the issue was so it would be easy to correct. All the variables in my code are used so they are all relevant. I also reused some of my code such as the object instantiations and slightly altered it to suit my goals as an edge boundary analysis. 

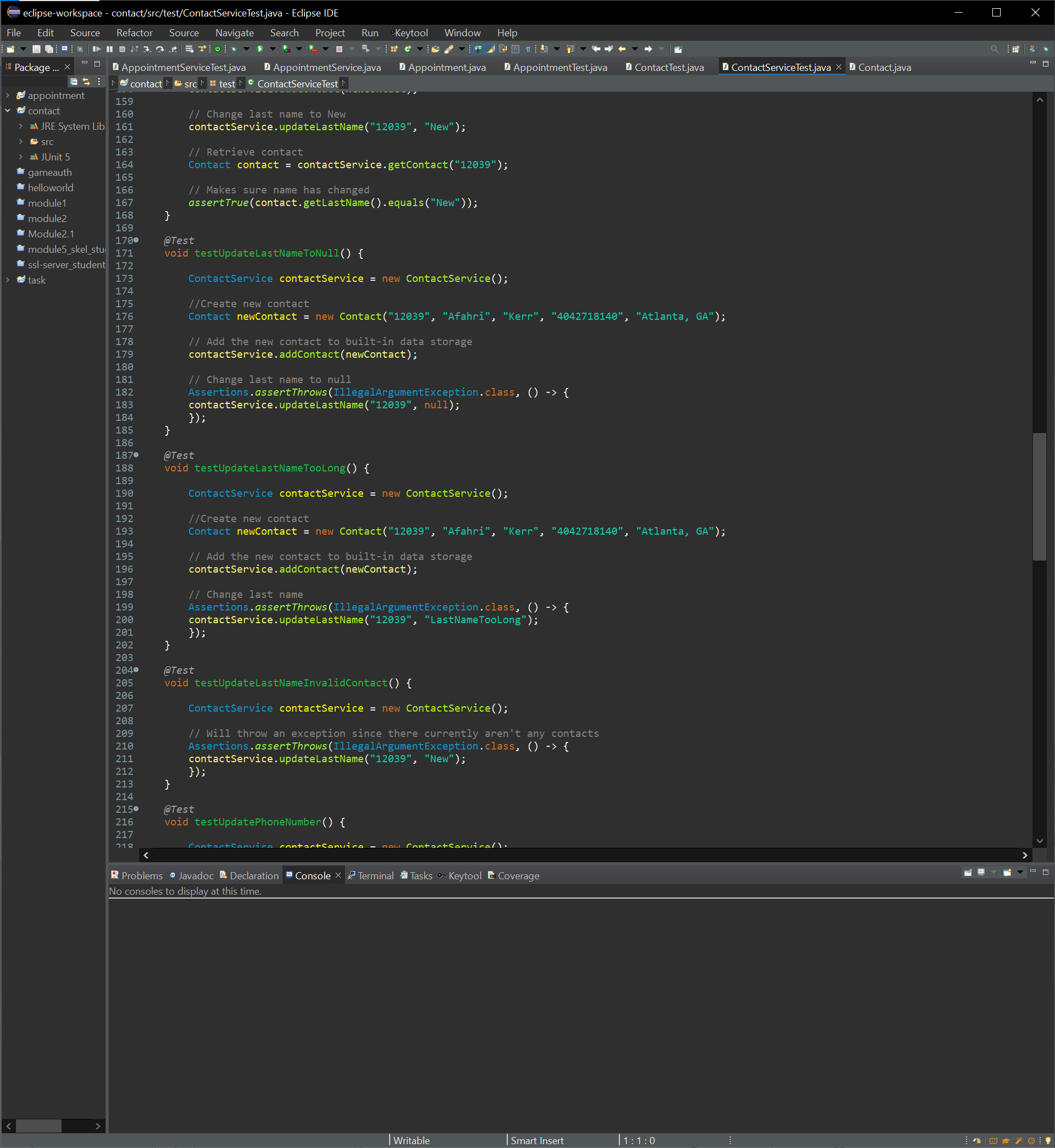
My code could be more efficient in 2 regards but I couldn’t get the @beforeEach method to work properly. I could have used it to create a task, contact, or appointment before all my tests and just updated or altered that object instead of having to create it for each test. I also could have used a map instead of a list for which would have made my ID checking a lot faster for when the database grows. Other than those 2 items, I used the best practices that I could think of to make my application run efficiently. I only tested for the things that mattered and used boundary value analysis to test the edge cases.

**Reflection**

For this application I used a variety of testing techniques. I used equivalence partitioning which is where you group the possible inputs into the different groups of possible inputs. I then used boundary value analysis to determine the edge cases of those inputs and used those to test my application. For example, the ID of the contact object could not be more than 10 characters and shouldn’t be null so there are 3 groups. A null group, a group in the range of 1-10 characters, and a group of characters over 10. I used boundary analysis so I created a contact object with an ID of exactly 10 characters and then added 1 to put it over the edge and tested that it would throw an error. These techniques can be used for any project that has a limit on input. Since I designed my test based off the requirements, I used black-box testing. This is a good technique to use when you only have the requirements of a program and don’t have access to the underlying structure of the code. I also used white-box testing since I designed the code and knew how it worked. Finally, I used regression testing to make sure it all still worked as I added more code. This is important in almost all projects since they will for certain grow larger as you work, and more features are added.

A technique that I did not use was integration testing which is to find the defects in the interface of the application. I would use this if we had designed an interface for this application. I did not use System testing which is carried out by integrating different components to evaluate how a complete system will behave once developed. If we connected the whole program, then I would have used this type of testing. I did not use Stress testing which makes sure that the system will work correctly under heavy loads. This test is better used on a program that will be run on a server or in the cloud and allow access to many people. I did not use decision table testing which is where you map out the different outcomes of your program for different combinations of input conditions. This technique is better suited for a more complex project that will have many different outcomes.

My mindset for this project was definitely cautious. I would design some tests for my code but then I would try to think of any more ways that the program could malfunction and design a test for that. For example, in my contactService class I needed to be able to update the names so I made sure that I could do that Then, I thought to myself “what if someone tried to update it to something illegal (null or too long)?” so I proceeded to make more test to ensure that wasn’t possible. If I hadn’t stopped to think about this then I probably would have overlooked this step.



I had to limit my bias for this project because I already felt that my code would work but I had to look at it as if I didn’t make it myself. I had to look at it from the point of view of a stranger or that I was trying to prove it to a stranger. I believe that this would be a concern if I was testing my own code because I am prone to believe that what I made is already perfect and may be inclined to only test what I know will work for sure or overlook certain tests. I almost overlooked the updating to invalid inputs tests in my above screenshot which might be possible in a larger project. It’s important to look at the program from every possible angle.

It is very important to be disciplined iin the commitment to quality as a software engineering professional because we are not the only ones that will be using our code. Our code may one day go out into the world and be responsible for controlling some important pieces of equipment which may put lives in danger if they malfunction. This is why it is so important to thoroughly test our code and not cut corners. I plan to avoid technical debt by always thinking outside the box (as above) if required to test my own code and if possible then I will get someone else to review and test my code. I revised my code a few times based on your feedback so as the saying goes “two minds are better than one”.