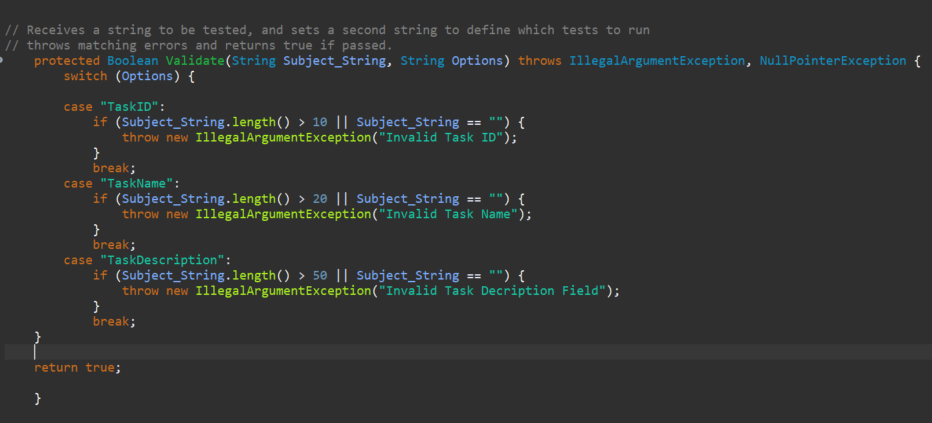
**Summary -**

**Tasks & Service -** In the task assignment I was given an instruction to build a Task service that would build a task object. The goal of this was to build something that kept track of each unique task with identifying names and descriptions and the names and descriptions could be the repeat in the case of recurring tasks. They also required service to keep that the tasks be updatable and modifiable except for the

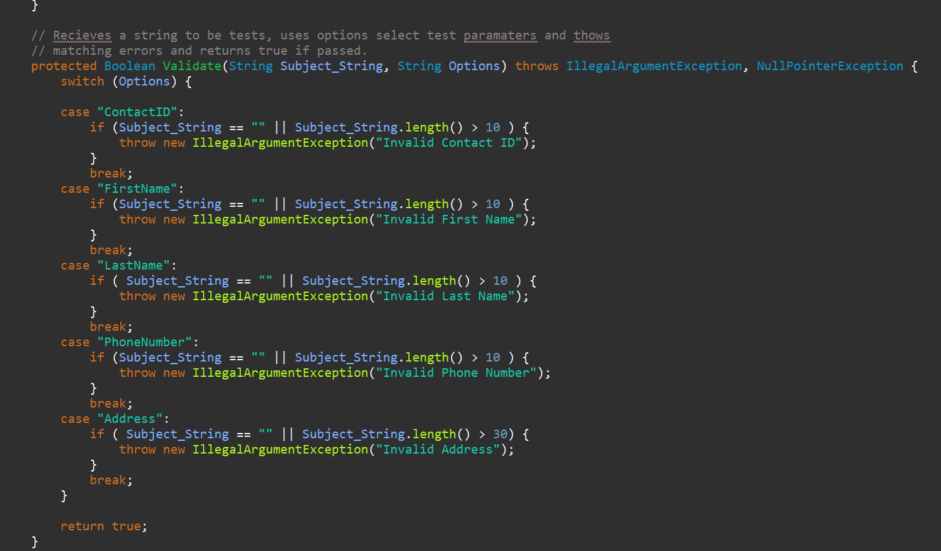
The Task Object required its own validation to check for incorrect inputs for all three attributes as well as an updater for Name and Descriptions and ways to access all three that would be used for later displays and finding the tasks in the services.

The Service Method, Required a local storage which I used Java’s ArrayList utility for its ability to grow with the size of the array. A method for generating unique IDs when creating new tasks. A method to search the ArrayList for each task, then methods to delete and update each on implementing that search method in the process.

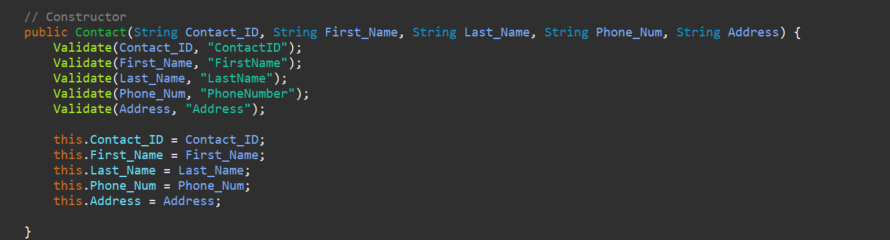
These align with all the required goals, but could be improved on with external databases that would link the three methods together and more complex data structures such as a hashmap to reduce search times in the long term.

After having met the criteria, the next step included testing and confirming the validation methods. My tests started with a base test where we confirm that each method hits its intended goal with good data. I then validate that all of my checks outside of my test were good. Testing if input was too large or small, or of an invalid type such as a null. If the results came back it reported invalid of each type.   
  


Error handling implements feedback to users in errors and results adjustable data so correct   
information can be added later.

**Contact & Service -** The contact method work almost identical to the task accept it had more variables to account for thus the number of options had to be increased.   


This feels like an appropriate time to identify how the option parameter works since each variable, each variable has its own set of tests and by using the option parameter to tell the validation method which tests to run they were all kept together and made for a decipherable test base so in the future when new test need to be added/modify it can be handled in one place.

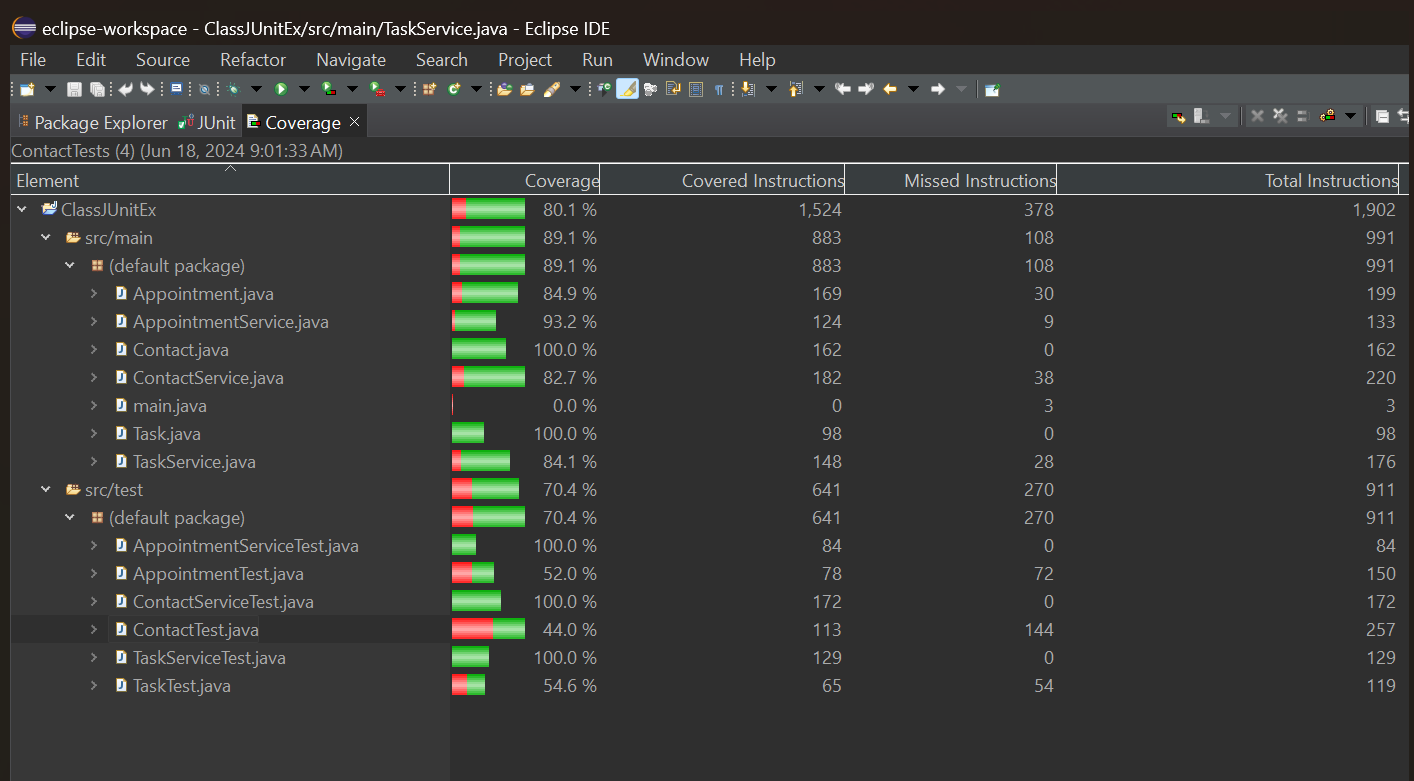


**Appointment & Service -** Appointment and its service followed pretty much the same flow as the Contact & Task for string variables, with this tool requiring the implementation of dates. Which would be needed to be validated and require another Java utility to use. Using the LocalDate class, I was able to grab the current date to prevent dates from being set in the past. This would be the longest validation test as there were alot of checks to do for the entire year.   


It also required taking the LocalDate information and breaking into its separate parts. This one should be updated in the future with update methods as a feature but that wasn’t requested and is worth a suggestion to the client.

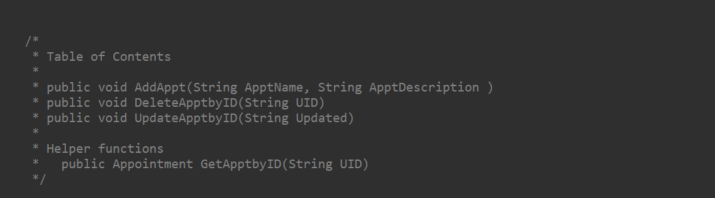
**Reflections** Testing techniques, This process required JUnit testing where we verify that the lines of code expected to catch errors did as well as tested for cases. My test reached an 80% coverage which isn’t 100% often requested. It did meet the client parameters.

My code reach only 80% because of the optional validation method I chose. These kinds of cases create unusual situations where switch statements w/o defaults can lead to decrease in coverage. I didn't use a default because the method I was expecting would default test.

  
  
While I couldn’t use a full code review with a co-worker I felt would be more applicable, I did speak with another programmer I know and discussed what my plans were and what kinds of tests I was running. It would have been a better implementation to hand it over for a full review but that really wasn’t something I could do at the moment. `

**MindSet** I implemented an approach of broad caution. My goals were to define narrow accepted results and set my code to reject anything outside of it and where possible eliminate any user input. For example the unique ID could have been generated via a combination of inputs from the user and a number but instead I’d chosen to define a set introduction to the appointment there classification letters at the begging would also be helpful later on in the future when attempting to see large data visuals or printouts in a pdf when the paper is printed.   
  
(this example include A# as the beginning of every Appointment UID paired with an incremental number.)  
  
I did attempt to limit by basis by discussing verably with my programmer friend to catch an obvious that I was implicitly blind to, but my overall mind included using the criteria of the assignment to eliminate understanding of how the criteria should be handled. For example I often felt the number of characters in the description should have been above 50 characters as it could have made a better product as well as planned for remote access to the different database rather than local.

Lastly, on the topic of what choices I did make to reduce technical debt, alot came from my basis that I felt the project would need as the project went forward the other side of this to limit my assumption for error testing and to take the inverse I asked myself feature might come in the future and how I could create the existing codebase in away to accept those.   
  
I implemented easy identifiers, readability comments about what was on each page in a Table of Contents at the top would always be used no, but when changes needed to occur It would decrease the amount of time I spent re-learning on past assignments.

  
  
I also decided to keep my code very modular, using accessor/mutators in the objects themselves this in the object are good building block tools so that if the services ever need to add additional methods it’ll be easy to grab from the object to start. This is one of the pillars of OOP programming and a key in every program. By creating an object that could be inherited if a new service method needed to be made in the future we could easily use the same validations and method created in the object to the new service features.

