Damian Duross CS-320 Software Test, Automation QA

8/17/2024 7-2: Project Two

For my initial foray into JUnit testing, I decided to utilize parameterized tests for each of the three features. I think this was a very efficient way to test for failures once I got the hang of it, but I did run into some issues with enforcing the parameters for the phone field of the Contact class, since it was stricter than the other fields (only allow digits, MUST be 10 characters). I was also sure to use the appropriate private specifier for the id, as this field was expected to be immutable. I did my best to practice good standards by not duplicating my work, incorporating the setters into the constructors. I also created an additional entity, Validation, as the three features all utilized very similar tests, so I was able to continually call on Validation rather than having to re-write tests every week for each feature. Ultimately, I am confident in my JUnit tests after adjusting the regex string for validateNumeric – all of my failure tests seem to be passing correctly. I do have an issue I couldn’t seem to resolve with the testing of the date – I couldn’t determine if, or how, to call LocalDate.now within the CsvSource, so I had to manually enter the present date. This works, but would require it to be manually updated each new day, so it is something I would like to improve for the future.

A screen shot of a computer program

Description automatically generated

This is my updated validateNumeric, with a more appropriate regex string…if I remember correctly, I had (“.\*\\D\*.”) or something like that in my initial submission, which I had come across while researching how to tackle the issue. It did appropriately pass those tests, but investigating it further, I concluded that it was passing for the wrong reasons - I’m much happier with this result.

A screen shot of a computer program

Description automatically generated

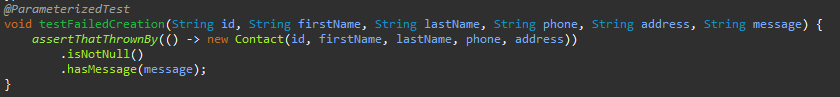
This is an example of one set of my setters and getters, which include the appropriate parameters that are tested for in Validation, as well as the constructors. I’m happy with how tailored this outcome was, I was able to trim a good amount of lines off the code here.

A screen shot of a computer screen

Description automatically generated

This is the CsvSource for the AppointmentTest class, which I mentioned needs a bit of work to be more efficient. The initial tests will pass because the blank, null, and too long ID errors will be thrown before it reads the incorrect date. The blank and null description are not tested properly here because it throws the ‘date must be present or future’ error before it can reach them. If the date is changed to the current date, all the failure tests will succeed.

I decided on parameterized tests because I thought they would be a good fit for the scope of the project – each class had 1-5 fields with on average three parameters each, so I had less than twenty failure tests to run for each feature. Since there weren’t too many requirements, I used a CSV data source, and assertThatThrownBy with a lambda expression as the testing method. It will check to see if creating an object with the given fields will throw an exception.



It was my first time utilizing parameterized testing, and I learned a lot by practicing with this assignment. I believe it went a long way to improve the coverage of my tests and helped me avoid writing duplicate code.

While I feel pretty strongly about parameterized tests, there are many different testing strategies I would like to practice more as I seek to become more involved in software development. Performance tests (such as load testing or stress testing) would help to make sure our systems can operate under a desired workload. What if my CsvSource included 2,000 tests instead of 20? A good example of performance testing can be seen in the release of live-service video games, such as the recent *Helldivers 2* by Arrowhead Game Studios. Their previous title had a relatively small concurrent player base, and peaked at roughly 60,000 players, so I believe they stress tested for their ‘wildest expectations’, which was about 200,000 players. The game became wildly popular on launch, and 200,000 was easily doubled – the servers collapsed, and the game was pretty much unplayable that weekend. I don’t think that kind of outcome could have been predicted, but it is a great argument for the importance of performance testing!

When starting this project, especially during the initial milestone, I was quite cautious. I set up my Contact class with constructors, setters, and getters, and immediately started to work on the ContactTest class so I could make sure they worked. Every time I condensed, trimmed, or reformatted a line of code, I was rerunning all my tests to make sure I didn’t break anything. If something was broken, I would undo it and try to find out why. Writing the tests this way helped me understand the relationships between my code much more effectively.

Looking back at my code now, I believe I have been for the most part unbiased. I have indicated where I made and corrected errors, as well as one parameter I wanted to improve, but couldn’t. It’s important to maintain your integrity when it comes to reviewing your own code. If you are tasked with writing and reviewing your own code, you’re entirely accountable for it, which includes negative feedback. This is a principle that can be attached to any kind of personal work. We are human and we will make mistakes. Mistakes shouldn’t be interpreted as character assassination – they should always be grounds for seeking improvement.

The same integrity ties into making good decisions and avoiding cutting corners when writing tests. If you try to speed through testing, and you miss something – now you must dedicate another period to testing *again*. What you imagined was saving you time has now become double the work, and it’s almost never worth it. I hope to always utilize the best practices of righting technically sound and clean code, and always continue to learn and improve in the process.