Report

In my report for the app development for the mobile application I built for Grand Stram Systems I will explain the approach I took towards developing my software tests. I will cover the unit testing approach I have used, my thought process and decisions I took writing the JUnit tests, and the coverage my resultant tests had over my code. This will summarize my skills in writing JUnit tests.

In my tests I made sure I met all the software requirements laid out for me in the requirements. Starting with the contact test case, the requirements asked that all values be within a certain length and are not null. To test the null cases, I used an assertion on each value to check if it is null. This ensures that the field isn’t null satisfying our first requirement. Then to test the length I first set a predetermined value to each and then checked an assertion to see if they came out valid. I also set another value that I knew was invalid and checked if it would return an invalid result. Knowing this, I could determine that the code had a boundary and that it was only accepting valid data and not all data.

if (id == null || id.length() > 10) {

This is an example that checks to ensure the id isn’t null and is less than 10 characters long.

Another technique I used was playing both sides, as I imagined I was the coder and the user. I imagined what I would type as a user and checked to see if that would break my code or not.

Assertions.*assertThrows*(IllegalArgumentException.class, () -> {

contact.setFirstName((String)null);

});

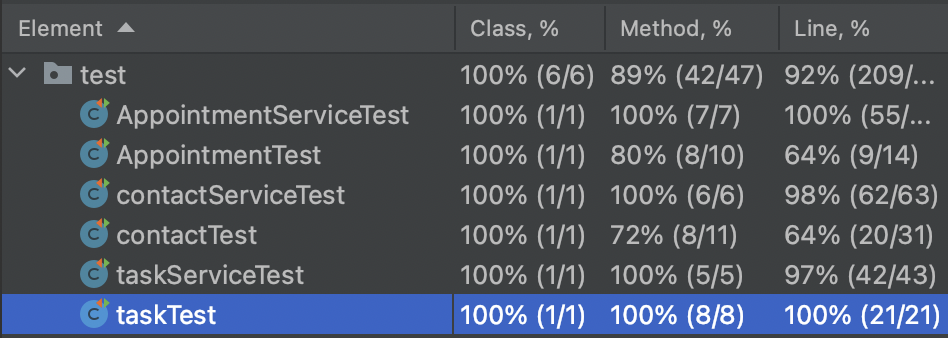
*assertEquals*("Isa Ali", contact.getName());

So in the above examples by checking the null and the valid values I was able to figure out that my system was checking the values and providing coverage over my code.

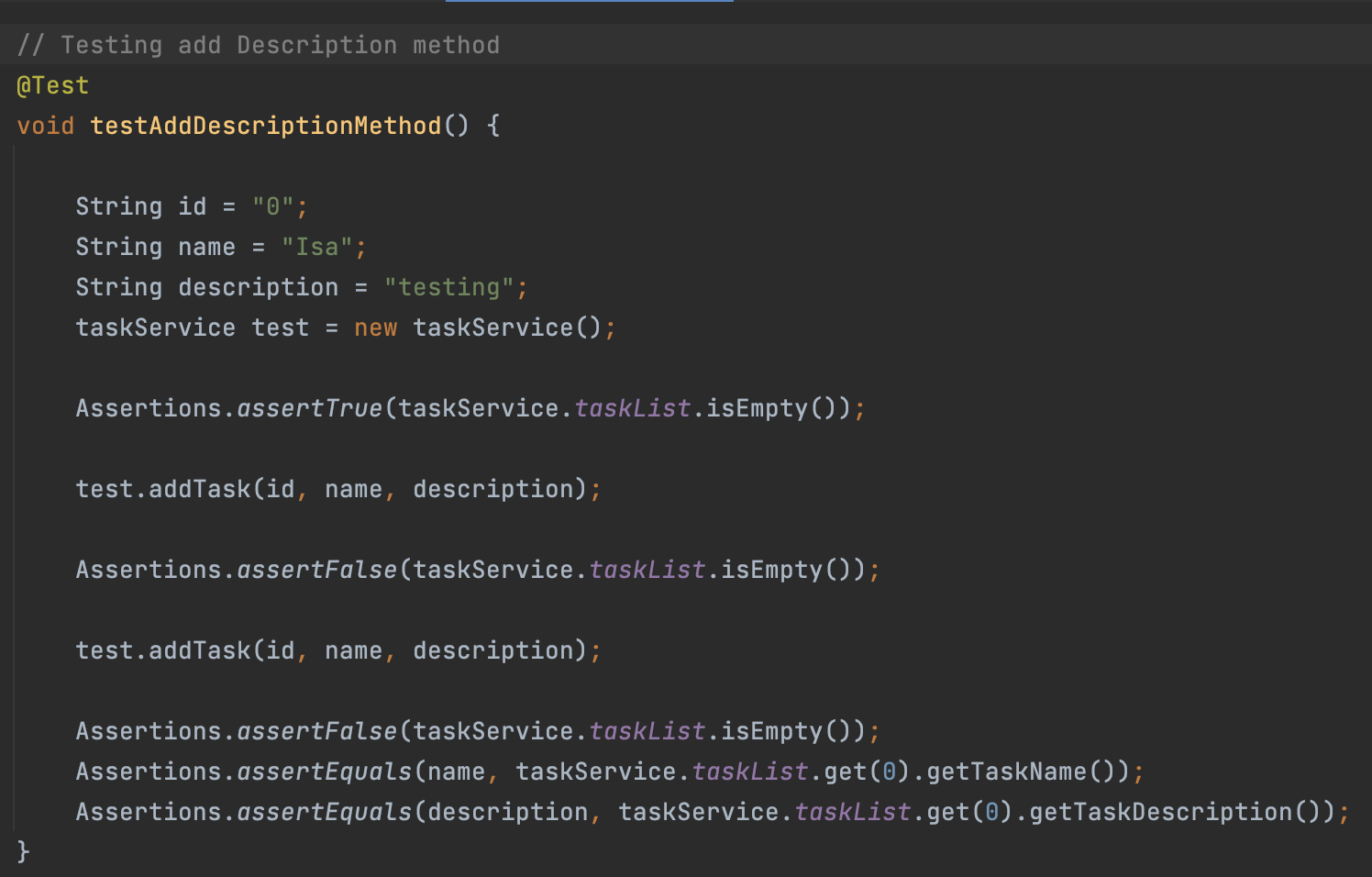
In other areas of the code I also followed this practice. By ensuring I knew what was false and what was correct, I could make sure my code was working properly. I also adapted this as I wrote my code to function for different scenarios. This was especially important when I had to work with the Contact Service requirements, that required me to ensure that I was able to add, update, and delete fields properly. This required more out of the box thinking, which I adopted in my test cases. For example is my testEditAddress function, I tested the initial contact address and then I retested it with a new address that I edited. This means I have two tests in place, one before and one after. This makes it absolutely certain that the case was changed, and it was not the same before and after. By using techniques like this I am able to increase my coverage percentage.



I believe the quality of my test coverage was overall successful and managed to ensure my code performed to the standards expected of me. As shown in the image, I did not always have perfect test coverage as it is unnecessary to do so but I tried to ensure that a majority of my tests are covered. Generally I ensured that the code had at least one valid and one invalid scenario, that way I could test the limits of my code. I also ensured that for every requirement that my code had, like ‘ensure it is under 10 characters’ I adopted one assertion that would confirm that this had been implemented in my code. Through these methods I was able to provide full coverage and prevent errors from occuring in my app.



When I was writing JUnit tests I tried to ensure that my code was technically sound. I did this through thinking about every step of my code and the different approaches I could take to reach the solution. I then ensured the idea I thought of would provide full coverage to my code in the most efficient manner possible. This was often seen in the “service” part of my assignment, as I had to add, remove, or delete certain elements of my code which often required me to consider different possibilities for me to properly test it as this was a more complicated thing to ensure

was working properly. A good example we can use is when I tested the add description method. 

In order to do so I first tested that the taskList was indeed empty, so we were starting with a blank slate. Afterwards I added one task, and then I checked if the taskList was no longer empty as we now have 1 item inside it. Then I added another task, and checked if the name matched my pre-defined test task and the description matched the predefined test description. If they matched, it meant I could be certain that the task had indeed been added and was working properly. By ensuring I tested every single step of the process I could be certain that no errors were introduced in the process and It was working properly. Without detailed step by step coverage it could be possible for unpredicted errors to occur and for false positives to happen. This process was repeated throughout my code, and it was the most efficient manner to approach these complicated problems. I also tried to use the fewest lines of code I could and keep my code clear and concise which contributed to the efficiency of my code.

The main testing methods I used in my code were black box testing. This testing allows you to check the coverage of the app without having to use many test cases. The way you use this method is by taking your input and putting it into a “black box” and checking if the output is as expected. If the code is functioning properly then these values will be expected and your program will be functional. This is also known as a form of manual testing, as we check the values by hand and see if it is what we expect them to be.

Another form of testing I included in my development was whitebox testing. In this form of testing I try to run my code and search for any errors that pop up. Sometimes I would often notice I had called the wrong value, like name instead of description, which caused weird bugs in my test cases. By running the code I was able to catch this and solve the problem. I also had trouble adding and removing values from the list, and by running my solution I was able to ensure that I received what I expected.

If I was to use these techniques on a new project in development, I believe it would be useful to help reduce development times and improve development efficiency. By employing skills like white box testing I can ensure I am properly handling the development processes and the code is being built correctly. I can also use blackbox testing and check if the code is producing the correct results that I am expecting in my development processes.

One technique I did not use in my testing was automation testing. In this testing you use tools to automate the testing process without having human input involved. This can speed up development times and reduce the laborious areas of software development. This is usually done by writing a test script that automatically executes the test cases and checks the coverage. The software engineer is typically responsible for writing this script. This can be useful so we can regularly check that the app is still working just by clicking run instead of having to manually test the whole app. Often during development areas of the app can break so being able to confirm the app is still working is valuable.

As a software tester I approached the code from an end user point. Often, when we try to test our own code we go easy on ourselves because we think our code is too good to have errors. I made sure I approached it from a separate point of view and didn’t let myself interfere with the testing process. When I was acting as a programmer I also tried to methodologically work through my problems. For example, there was a confusing requirement where I had to ensure the date was before the expiry data. In order to do that I broke down the problem, and then tried to implement each step.

*“The appointment object shall have a required appointment Date field. The appointment Date field cannot be in the past. The appointment Date field shall not be null.”*

This was the question. By thinking methodologically I was able to break apart each requirement to its base elements and then add it into my code. As a tester I reversed this approach, and instead chose to focus on how I could make sure I broke these statements if it was possible at all.

if(appointmentDate == null || appointmentDate.before(new Date())) {

throw new IllegalArgumentException("Date must not be null or in the past");

The best way to remove bias for me was to imagine I was working on someone else’s code and try my best to reach the max amount of coverage. I am sure in the field if you are working on a big project it can be hard to be the tester also as you overestimate the quality of your code. This is why they often break apart the roles. Many times In my program I assumed that my add/delete/update methods were working, but when I tested them they returned errors to my surprise. This is an example of when bias enters the development process and can be hard to hide.

Overall this leads to the type of developer you want to be. I think it is important developers show integrity in their work and are open to their code being picked apart without it hurting their feelings. Without extensive and unbiased testing it will be hard to catch bugs and prevent them from entering production. It also will be hard to grow as a developer if you do not spend your time properly testing your code as you will continue to write bad code. In order to avoid accumulating technical debt I will try to routinely test my code and try not to write too much new code without properly testing it.