CS320 Summary and Reflections Report

Ron Bal

Southern New Hampshire University

Table of Contents

[Table of Contents 2](#_Toc121853840)

[1. Summary 3](#_Toc121853841)

[1a. Describe your unit testing approach for each of the three features. 3](#_Toc121853842)

[1b. Describe your experience writing the JUnit tests. 4](#_Toc121853843)

[2. Reflection 4](#_Toc121853844)

[2a. Testing Techniques 4](#_Toc121853845)

[2b. Mindset 5](#_Toc121853846)

[References 6](#_Toc121853847)

CS320 Summary and Reflections Report

# 1. Summary

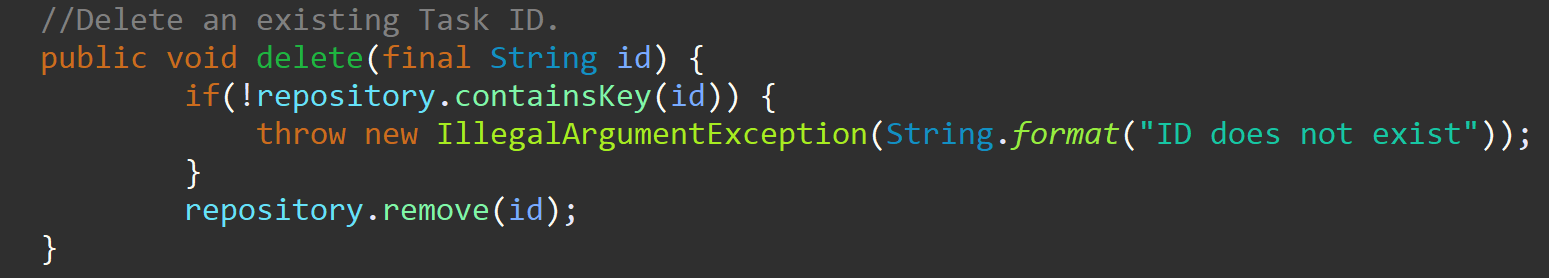
## 1a. Describe your unit testing approach for each of the three features.

### Contact Entity and Contact Service.

#### To what extent was your approach aligned with the software requirements?

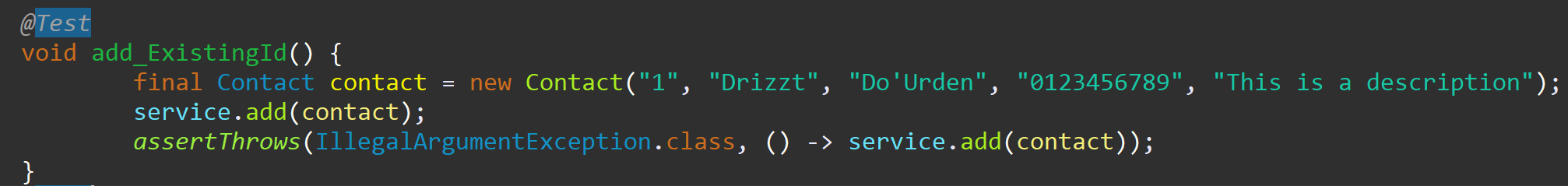
My approach aligned with all aspects of the software requirements that were given for the project. How I approached this was to first make comments within my code so I could easily access the desired requirements. The requirements stated we needed a ConactId, First Name, and Last name variable that could all be no longer than ten characters, updatable, and not null. A phoneNumber variable that had to be exactly ten digits, updatable, and not null. Finally, a contactDescription variable that could be no longer than thirty characters, updatable, and not null. To ensure these constraints were met, a Contact Service was utilized that would handle the addition, deletion, and updating of the variables found within the Contact class.

Here we have the handling of deleting an Id where the repository, created earlier on in the code with the use of a hashmap, would be checked for the existence of the ID we’re looking to delete. If the id doesn’t exist, it can’t be deleted and an illegal argument exception is thrown. If it does exist, the ID is removed.



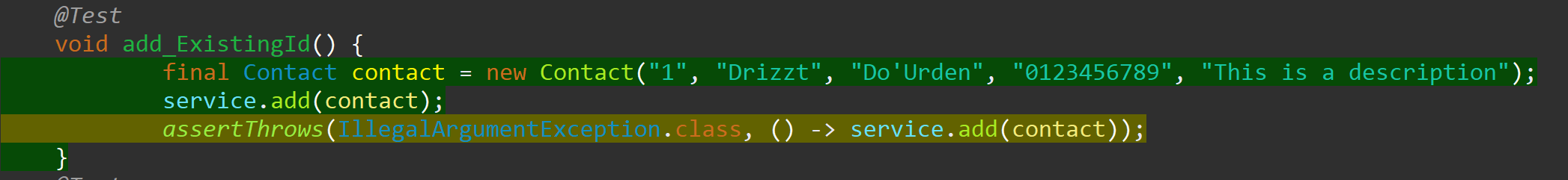
#### Defend the quality of your JUnit tests.

In the testing of the Contact Service class, I initiated a new instance before each test using the @BeforeEach function which will prevent old information to be used that may throw rogue exceptions. The Contact Service needs to be able to handle the addition, deletion, and updating of information found within the Contact class so, we had to check to see that those written functions operated as intended. The below code is testing that a new contact can be added. The test adds new contact information to the contact class through the service. If the contact can’t be added, an illegal argument exception is thrown.



These tests weren’t extensive enough as I didn’t get as much coverage as I should have, coming it at only seventy two percent. It seems as though my existing ID test wasn’t written effectively.





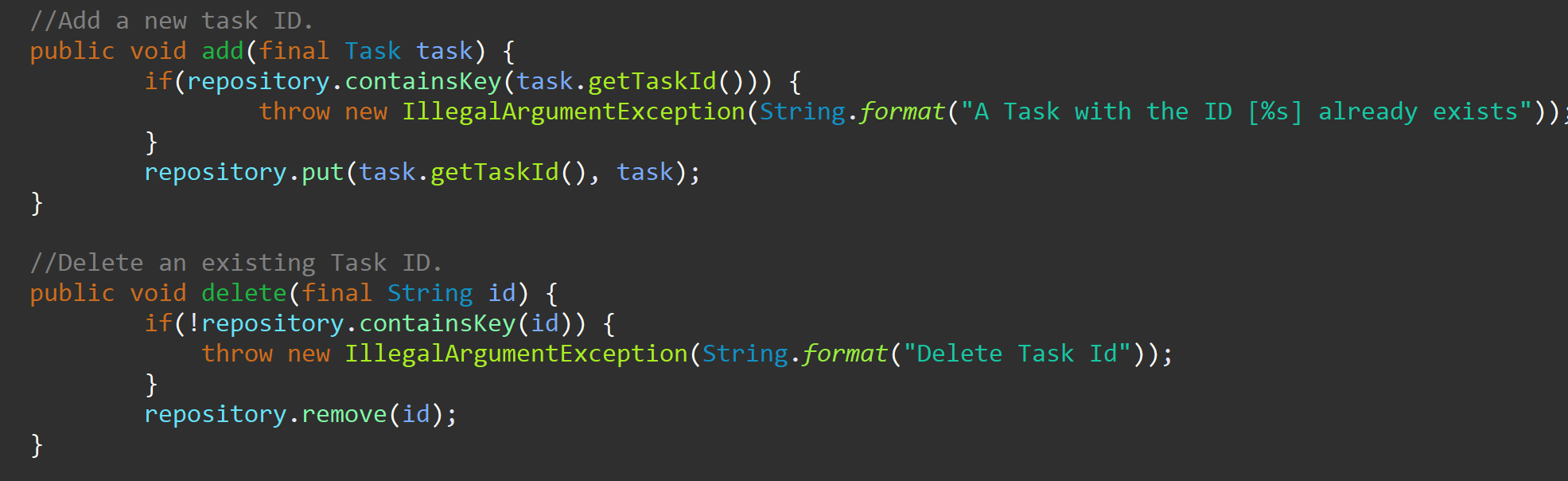
### Task Entity and Task Service.

#### To what extent was your approach aligned with the software requirements?

I utilized the same methodologies here with the Task and Task Service classes. As with the Contact and Contact Service classes, I copied the requirements for the project over to the code as comments for ease of reference and implementation. With the Task Service, those requirements were that a new task should be able to be added with a unique ID, a task should be able to be deleted per a task ID, and it should be able to update task fields per the task ID. I utilized a hashmap to handle a repository of the information it would hold so that it could be called upon per the requirements of the Task Service.

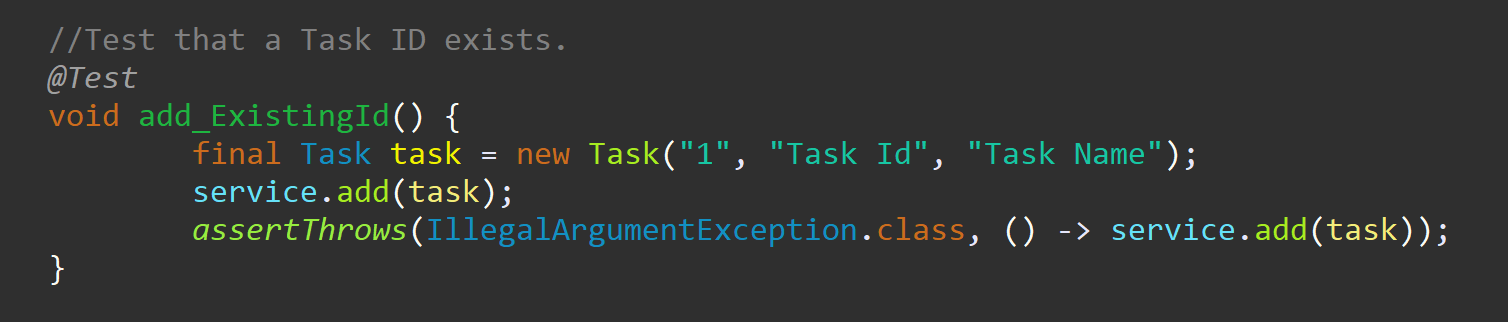
Here I used an add function that would update or check the Task class within the task package. If the taskId was found within the repository, an illegal argument exception would be thrown alerting the user that the taskId already exists.

The delete function will search the repository for the taskId to ensure it exists and that it can be deleted.



#### Defend the quality of your JUnit tests.

With the Task Service and its requirements, I had to ensure that the adding and deleting function I defined within the class worked properly. The first test I wrote was checking that an existing TaskId would throw the exception message I had written. The below test would create a new Task from the task package, add it to the Task Service, and throw an exception if that task already exists.



Within my Task Service test, I achieved eighty six percent coverage, where it seems I missed the successful testing of eight instructions.



### Appointment Entity and Appointment Service.

#### To what extent was your approach aligned with the software requirements?

Just as the other classes, I made sure to comment the constraints and requirements of the project. For the Appointments class those requirements were an appointment ID string that’s no longer than ten characters, updatable, and not null; an appointment date field that cannot be in the past and not null; and an appointment description that cannot be null or longer than fifty characters. The Appointment Service class needed to be able to add a new appointment with a unique appointment ID and be able to delete appointments per the appointment ID. Again, using a hashmap to store the information within a repository to easily refer back to.

The below code handles the addition and deletion of appointments with call backs to the Appointment package and class. In the add block, it will search for the existence of the ID that was input and will add it to the appointment if the ID doesn’t already exist. If it does exist, an exception will be thrown.



#### Defend the quality of your JUnit tests.

As with the other service test classes, I initiated a new instance of the service before running each test using the @BeforeEach method. This test shows us the successful addition of an appointment by creating a new appointment and adding it to the repository.



I created relatively good coverage with these tests but missed ten instructions. It seems an argument is being thrown when an existing ID has been attempted. I’ve come to find the importance of running coverage tests. The four tests that I had written for the Appointment Service all passed but, it doesn’t express to me the aspects of the Service class that I’m not testing or not testing properly.

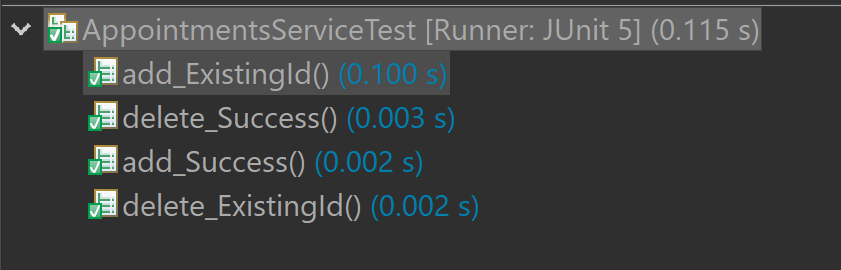


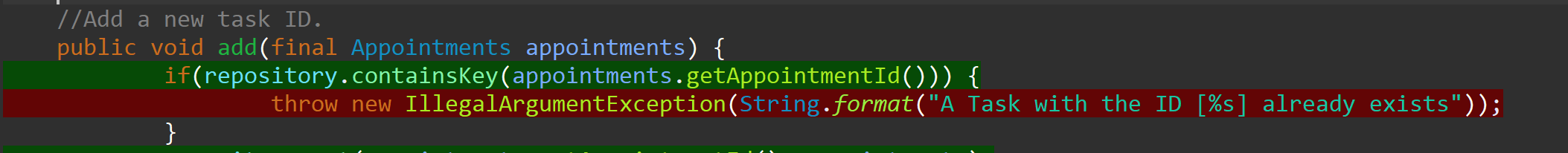
## 1b. Describe your experience writing the JUnit tests.

### How did you ensure that your code was technically sound?

The first step in confirming my JUnit tests were structurally sound was through running the JUnit test themselves in each of the classes I had created. This first step allows me to see that the tests that I’m running are running properly. If I’m getting errors or failures, the tests need to be tweaked for compliance. Running this test is how I found out I was approaching the phoneNumber variable constraint of being exactly ten digits. I was expressing that the phone number could be no longer than 10 digits. It would pass the test if it were too few digits. After creating the JUnit tests and having them all pass was running a JUnit coverage test. This test goes a step further testing that all aspects of your code are being tested such as the illegal argument exceptions I utilized throughout my code.

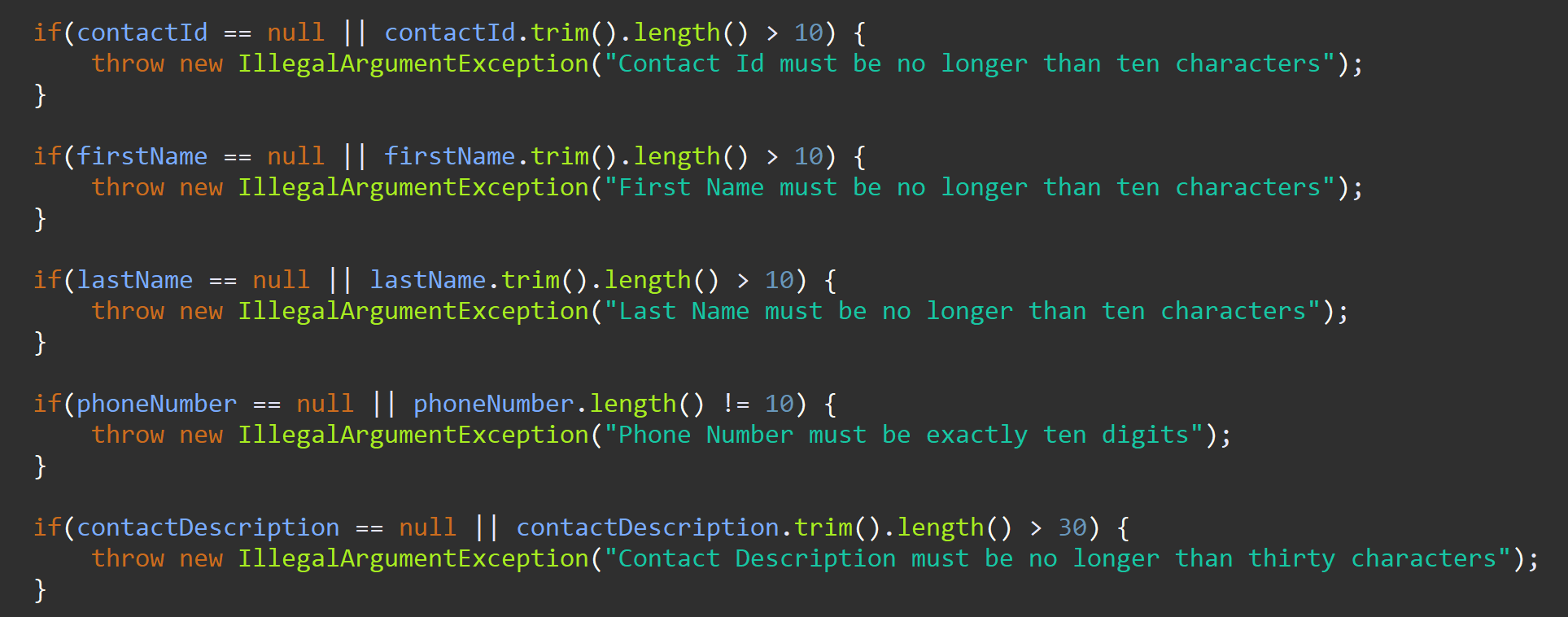
As we can see here, all of my JUnit tests passed for adding and deleting ID’s but, when I ran a coverage test, I can see that I’m not throwing an exception I should be throwing. Which tells me I need to add more information or rewrite my test.



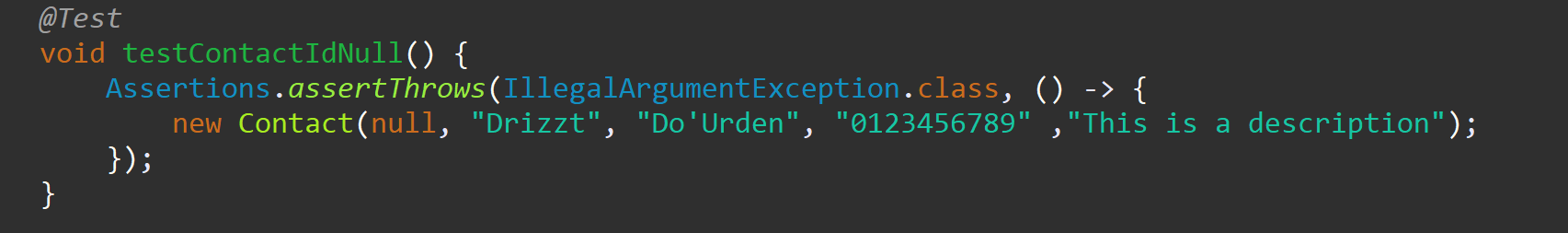


### How did you ensure that your test code was efficient?

I feel my code is efficient based on how much I was able to accomplish with a relatively short amount of code, keeping it clear and concise. For instance, I utilized if() statements with the or function to set up the constraints the requirements stated our variables should have. While I could have lumped a lot of the variables and constraints into less code, I feel this way allowed for easier readability and to track down any errors that may exist. For instance, I had an issue with the phoneNumber length early on when running my JUnit tests because I was using the wrong characters to express how long the variable needed to be.



I used a similar style when creating my JUnit tests. Below I was testing that an exception would be thrown if the contactID was null. If I had made them all null and there was an issue in my code where an exception wouldn’t be thrown, I wouldn’t be able to track down which variable and where the code isn’t running as expected. While this does create more code to be written, it’s easy to read, understand, and to follow the errors.



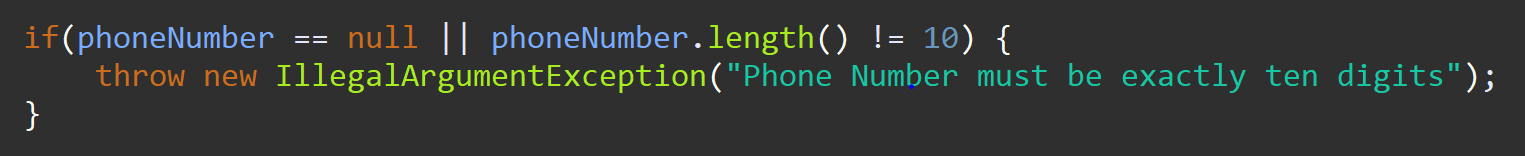
# 2. Reflection

## 2a. Testing Techniques

### What were the software testing techniques that you employed in this project?

The testing method I chose to employ was the use of Illegal argument exceptions and assertions. The use of these two methods allowed me to get feedback on where I was getting errors, allowing me to know exactly where to look. I first defined the constraints, found within the software requirements, in a class named Contact where I instructed if any of the above variables were longer than intended or null, an exception would be thrown.

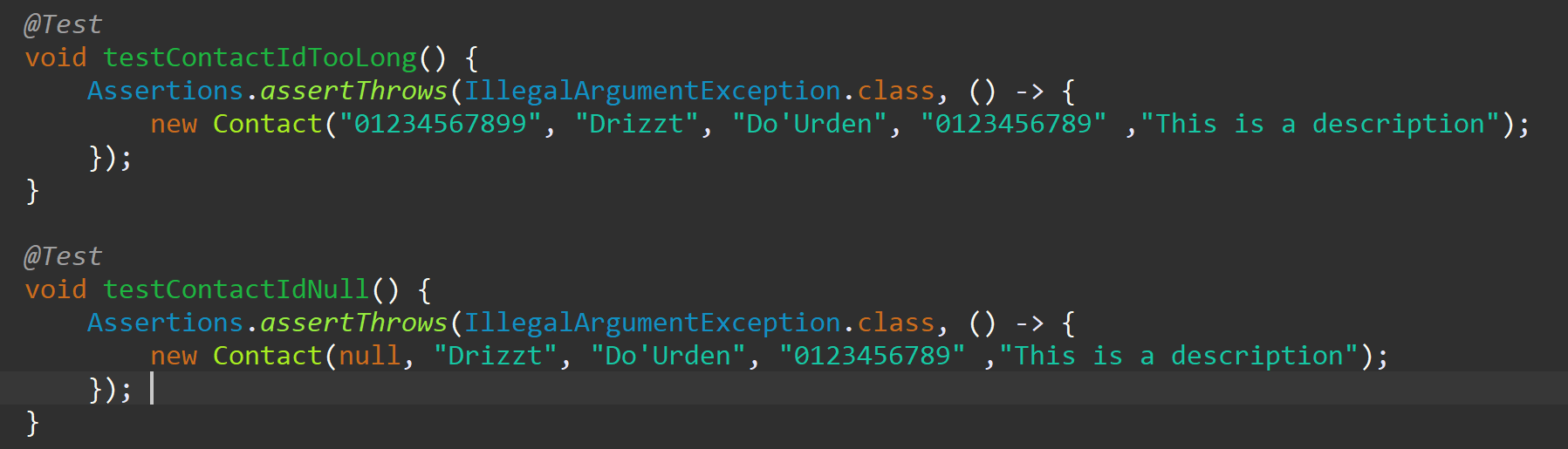
Below is an example of how I handled the variables’ conditional requirements.



I utilized if() statements to express what should happen when the variable is null or (denoted by two vertical bars) variable length, in this case, is not ten characters. If either of those happen an illegal argument exception is thrown with a description of why the exception was thrown. In the above picture, if a phone number length wasn’t exactly ten digits, a message would pop up stating so. Having different descriptions for each of the if() statements allow us to track down or express to the user why the input can’t be accepted.

Within the contact test, I tested all aspects of the class and the constraints of the variables to ensure the if() statements I made were correctly implemented and the variables were set up properly.

Here, I wrote tests that would check the bounds of the contactId variable. The first one tests if the contactId is too long, denoted by a contactId with eleven digits. I made sure to use conforming information with the subsequent variables so that I would know which variable was throwing an exception if thrown. The second test for the variable was to assert that the contactId was null, again using conforming information for the other variables so we know which variable is throwing the exception.



### What are the other software testing techniques that you did not use for this project?

One testing technique I didn’t use was mocking, where classes and methods can be tested in isolation. This testing technique is more suitable for large cumbersome projects and allows for smaller amounts of code to be written, decreasing the amount of time it takes to write code to effectively test the project. Utilizing the mocking technique, the program Mockito in particular, has an assistant that will offer quick fix actions it recommends for aspects of the code that aren’t being tested. This assistant is great for when you make changes to your code and potentially forget to create or modify a test method for the changes made.

I didn’t use this testing method as the classes we created were relatively small and the variables we used only had a few limiting factors that we had to check. In this particular instance, I feel creating JUnit tests was a better course of action to better understand what’s happening with the tests and how the tests operate.

### For each technique you discussed, explain their practical uses and implications for different software projects and situations.

Using the mock method of testing allows testing in parallel to development. When a class needs to be tested but we don’t want to call upon a real instance of the dependent class, it can be mocked so the flow of work won’t be disrupted. This method also allows for the testing of a procedure where an event may happen, such as an email being sent, without having to needlessly send real emails out. Another instance where mock testing shines is if the code needs to make API calls; the API can be mocked instead of making real time calls to the API.

## 2b. Mindset

### Assess the mindset you adopted working on this project.

I wouldn’t say I employed any inherent caution when approaching the testing of my code, being new to testing and coding in general, I was learning as I went. I did however come to the understanding of why being cautious is so important through the reading we did this week. Cutting corners and not being thorough can have a massive negative impact; financially, user retention, company image, or human safety. While I don’t feel I approached my testing with any caution, I did try to be thorough. In one of my test cases I tried, and succeeded, in getting a hundred percent coverage by attempting to test all aspects of the methods and variables I chose to use.

### Assess the ways you tried to limit bias in your review of the code.

Approaching bias towards your own work is tough. I had a relatively easy time being non biased towards my work on this project and the work that preceded it from the years I’ve spent fixing my own mistakes and trying to prevent them in my current profession. It’s very easy to assume you did the work correctly, especially if you’ve been doing the same work for a while. The reality is, we’re all human and we all make mistakes. A “trick” I came up with was to assume that I’m wrong when I double check my work. Approaching my work this way allows me to look in greater detail while I look for mistakes. For instance, I used to be responsible for shipping out spare parts at my work, often times I would think I grabbed the correct part but really, I grabbed a different part that has a similar name. Or, I grabbed a part from the bin right next to the part I intended to pull. To avoid silly mistakes like that, I started double checking the part numbers and quantities of the parts I pulled to the ones I’m supposed to ship. This double check has prevented numerous potential issues.

I used the same approach on the work I did in this class. Although, running coverage tests really does the brunt of the work and keeps you honest. It would be very easy to run just the JUnit tests and assume everything is great when all the tests pass. If you’re testing your own code, it’s very easy to assume you did everything correctly and tested all the different classes and methods without looking at it as if you had never seen it before. This happened with me when I was just building and running JUnit tests. All of my tests had passed and I thought everything was great. Until, I ran the coverage test and realized I never wrote code to test if an exception would be thrown if any of my variables were null. One thing I could have done differently to limit bias and to ensure I created efficient testing would be to copy over the comments of all the variable requirements over to the test class to ensure I tested all the key points.

### Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional.

It’s important to be disciplined as coding errors can not only cost companies a lot of money to fix, and just as we saw in the reading and research for this week, people can be hurt. The defect that I had researched and discussed was in a game named World of Warcraft where a glitch caused a vast number of users to die in game when the effects of a boss, called corrupted blood, that was only supposed to affect users in the realm and vicinity of the boss. It ultimately ended up being able to be transferred to non-playable characters and user’s pets which ended up affecting and killing everyone. While this bug was low risk, I’m sure the game was unbearable to play and reduced the number of users playing the game.

There’s an old adage I’ve heard “I don’t have enough time to do it correct the first time but, have time to do it a second time.”. It’s always more efficient to take the time to do it correctly the first time. I’ve fixed many mistakes after the fact, both my own fault and not my own fault, and it’s always significantly more difficult to fix on the back end. That’s something I’ll always keep in mind in my future as a software engineer to ensure I do everything possible to get it right the first time. At least as close to right the first time as I can get.

References

Arvindpdmn, N. (2021, June 28). *Mock testing*. Devopedia. Retrieved February 14, 2023, from <https://devopedia.org/mock-testing>

Ward, M. (2005, September 22). *Technology | deadly plague hits warcraft world*. BBC News. Retrieved February 12, 2023, from <http://news.bbc.co.uk/2/hi/technology/4272418.stm>