Aaron Bias

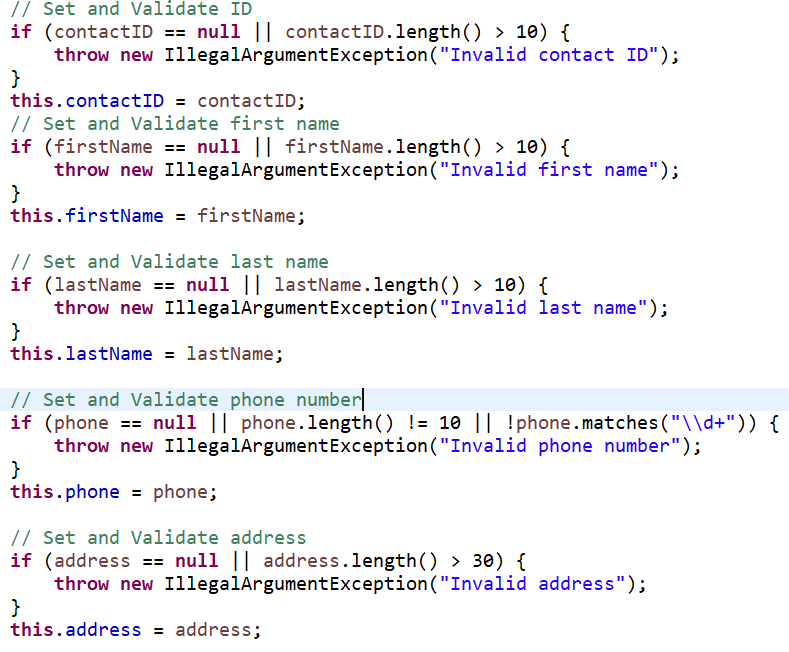
05/19/2024

CS-320: Software Test, Automation QA

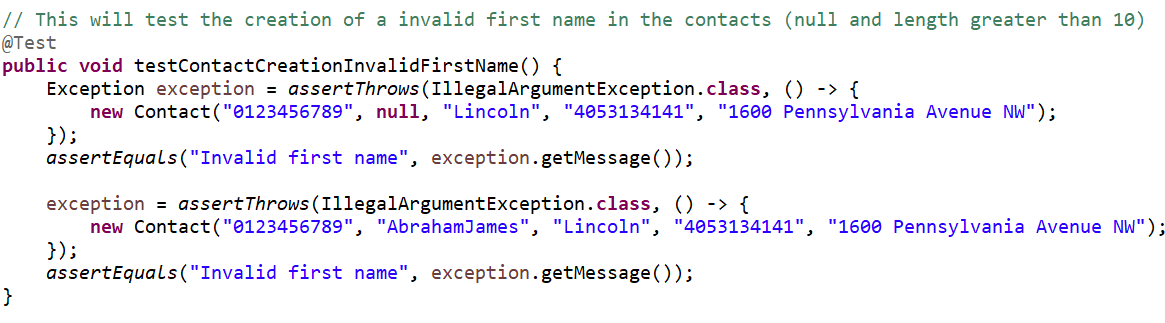
Professor Jeff Phillips

**Project Two: Summary and Reflections Report**

For the mobile application project, I was tasked with implementing the Contact Service, Task Service and Appointment Service. These services contained class files and functions essential to their respective operations. The Contact Service manages the contact information for the mobile application by adding, updating, retrieving, and deleting contacts. The Task Service manages tasks within the mobile application by allowing for creating, updating, retrieving and deleting tasks. The Appointment Service manages appointments within the mobile application by creating, retrieving, and deleting appointments. Each of these services requires unique IDs for identifying and managing individual records accurately and efficiently. The services required unit test to verify they meet specific requirements such as character length, null requirements, numerical requirements, adding, updating, deleting requirements based on unique IDs, and Date requirements among a few others.

The most common requirement for each service was character length (not longer than 10) and not be null. The example below is the work I will be referencing in my approach to validating the requirements of the project.  
The null and character length requirements of each service were enforced through validation checks in the constructor and setter methods. In the example above (Contact Service), any attempt to create a contact object with invalid data or is null would result in an exception.

The unit test (JUnit) I wrote for the mobile application was to verify that these validation checks were effective. In the Contact Service example, I provided earlier, my test to verify that invalid data (first name) is handled correctly looked like this:

 In this test, I am evaluating the behavior of the Contact class constructor specifically for handling invalid data in the first name field. The test verifies that invoking the constructor with a *null* or a *string longer than 10 characters* throws an IllegalArgumentException with the message "Invalid first name". These tests, along with others I conducted, ensured that the application appropriately detected and reported errors when invalid data was provided.

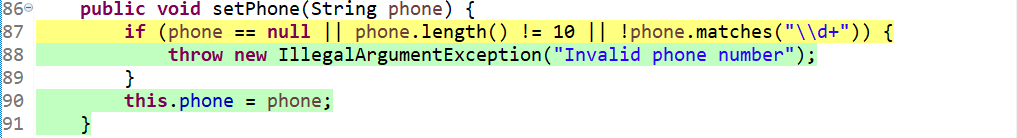
During the evaluation of code coverage using JUnit, I attained 100% coverage for the Contact Service, 100% for the Task Service, and 100% for the Appointment Service. This perfect coverage of my original files gave me confidence that my application met the client's requirements. While my JUnit test files did not reach 100% coverage, they successfully addressed critical paths within each service and validated the expected behaviors.

**Reflection**

The software testing technique I employed in this project was unit testing and white-box testing. With the unit testing (JUnit), it involved testing individual unit and components of the application to validate that each unit performed as expected. When developing these services, I tested often as I developed the program to ensure that each piece of functionality worked correctly and to minimize defects from arising. White-box testing helped me to ensure all paths and logical branches were tested and identified issues were resolved. White-box testing is aimed at covering all possible code paths which ensures each line of code is tested at least once, in which my coverage percentage I identified earlier is a great indication it was.

Software techniques I did not employ include integration testing, system testing and User Acceptance Testing. With integration testing, we are combing the modules and testing the interactions between them. In our project, we would test the Contact Service, Task Service, and Appointment Service interactions between each other to validate they are working as intended. System testing would test the application as whole to ensure it meets the requirements given to us. This test would cover all functionality of the system. User Acceptance testing is testing the application in a real-world scenario (with releases such as alpha and betas) with the end user so that the user experience and expectations are in line with the requirements of the project. This final stage of testing is important as the next phase is the software going live and into maintenance mode.

My mindset when developing the service modules was to test often and pay attention to details. For example, in my Contact class, the requirements for the phone number were that it be exactly 10 digits and not null. Although it might seem sufficient to check the length and null status of the string, a string can contain letters as well. Therefore, I also included a check to ensure the string contains only numeric characters, throwing an exception if it does not. Example provided below:

 Writing tests that align with the requirements of the project help in limiting the developer’s bias. Clearly defined requirements will help remove a lot of assumptions the developer might have about the project so continuos testing throughout the projects (including requirement analysis) will minimize developer bias. In my previous Phone string, it's assumed the developer would know the phone number will only contain digits, but more precise requirement will alleviate assumptions (or maybe the client wants to accept numbers with letter such as 1-800-GOT-JUNK). Peer reviews and code review tools would also help in limiting my bias by providing an external perspective on my code, but I mainly used the resources provided in the modules or YouTube.

Its important to not cut corners when writing and testing code as poorly writen code or inadequate testing will lead to greater errors down the SDLC pipeline. Technical dept is taking shortcuts which will lead to more work later on and lead to system issues and poor user experiences. I plan to avoid technical debt as a practitioner in the field by continously testing throughout the SDLC, committing to quality code and testing practices, utilize peer revies and code review tools and to following the established coding standards of the field to ensure its maintainable.