Alexandra Delarosa

CS-320 Project two

JUnit tests

My unit testing approach for each three of the features where to write tests based on the requirements. For the contact feature, I wrote tests to validate requirements such as the ability to add, delete or modify contacts and ensure contact information follows certain guidelines such as character length. In the ContactServiceTest class lines 7 through 13, I wrote a test case to verify a contact was added successfully. Additionally in lines 15 through 39 I wrote test case to verify a contact has been successfully deleted and modified. In the contact class, exception messages are thrown to validate guidelines such as ID being not null and within 10 characters and address being not null and withing 30 characters. For the task feature, I wrote tests to validate the requirements tasks having a unique ID and character lengths guidelines as well as the ability to add, delete or modify tasks. In the TaskTest class, I wrote tests for scenarios where the ID does not meet the length requirements. Similarly, I tested scenarios to test task name length and description length. The TaskserviceTest class validates the ability to add, delete or modify tasks. And finally, for the appointment feature, I wrote tests to validate the requirements of appoint to have a unique ID, not be null and no longer than 10 characters as well as having a required present date. In the AppointmentTest class, I wrote tests to throw in exceptions if the appointment ID is longer than 10 characters. The testAppointmentDateNotInPast test verifies that an appointment has a present date and throws is exception if it doesn’t. The Junit tests were effective because they cover edge cases, boundary conditions and error handling.

To ensure that the code was technically sound, I implemented assertions. The Junit tests utilizes assertions such as assertEquals, assertNotNull, and assertThrows to validate the outcomes. In the TastTest class, an exception is thrown when a task is created when the task ID is longer than 10 characters. Similarly, in the ContactTest class, an exception is thrown when a contact is created with a null contact ID. To ensure the code was efficient, I implemented hash maps to improve operations, such as adding, retrieving, or deleting data. In the ContactService and TastService classes, keys are identified by IDs which ensures constant-time complexity even large amounts of data is being processed. unit tests are carried out to test appointment and deletion.

Unit testing was a key testing technique employed in the project to ensure individual components of each feature meet the requirements. Unit testing allows testers to find bugs and defects. Testing individual components also allows for flexibility when changes in the code occur. Boundary testing was also implemented to test the limits of the input values. Boundary testing helps ensure that the code behaves the way it intended to do so by testing various input ranges. In the TaskTest class, I applied boundary testing to see if task IDs, names, or descriptions validate the required input ranges. This ensures that the code is reliable and reduces errors in production. A testing technique that was not used in the code is integration testing, which involves verifying that the components of the system are communicating correctly and without errors. Unlike unit testing, tests are run on combined units of code in integration testing. It ensures accuracy in data exchange between components and addresses compatibility issues.

While working on this project, I adopted a detailed-oriented mindset to better ensure the code accurately aligns to the requirements. Even the smallest mistakes in the code lead to troublesome bugs which is why attention to details helps in spotting issues early on. I employed caution by addressing edge cases and boundary conditions. In the TaskService class, I ensured that actions of adding, deleting, and modifying tasks behaved correctly and includes scenarios where inputs like ID were null or non-existent. Appreciating the complexity and interrelationships of the code is important because it’s essential in thorough testing in the code by identifying dependencies and interactions between the classes. Appreciating the importance of discipline and being committed to quality as a software professional is crucial to ensure the software functions correctly and behaves as expected. The end user will rely on the software to accurately perform tasks and failure to do so will lead to loss of trust. Being committed to quality can also lead to cost savings in the future as defects in the code can be time-consuming and expensive. To avoid technical debt in the field, I will ensure to adhere to best practices and industry standards to ensure code is clean, well-documented and maintainable. I will also conduct throughout testing such as unit testing, integration testing, and end-to-end testing to ensure the system works as intended.