Summary

While completing the tests for each of the Contact, Task and Appointment features, I followed the requirements set out by the project. I made sure to test the edge cases. For instance in the contact feature, I verified that an exception was thrown if the first name exceeded 10 characters. I also tested the system to handle valid data properly. For example, in the Appointment feature I tested that the object would be created properly when given a date in the past, (12/31/23) to be specific. Lastly, I tested that null input would produce exceptions. For example, if the appointmentId is null then the system should throw an IllegalArgumentException.

I know the tests I performed are effective, the entire system has a test coverage of 87.8 percent. The Appointment feature has test coverage of 83.4 percent, the Task feature has a coverage of 86.2 percent and, Finally, the contact feature has a coverage of 91.6 percent. All of these tests cover the main functionality of the system. In the Contact, Task and Appointment classes I tested that each object is capable of being instantiated within the constraints of the project requirements. In addition to this I made sure that invalid inputs were not passed into the system. In the service classes I made sure that the class can properly add, delete and update objects as intended.

When writing my JUnit tests, I was able to ensure that the tests were technically sound through two primary functions, testing valid values and testing exceptions. First, When testing my code for proper responses to valid data, I created an nCharString to test inputs of variable lengths. Here is the method:

private String nCharString(char c, int occur) {

String str = "";

for(int i = 0; i < occur; i++) {

str += c;

}

return str;

}

Secondly, when testing for expected exceptions, I fed the system input that is either out of bounds or null, I tested to confirm the proper exception is thrown. For example, when testing that the ContactId threw the proper exception was thrown I checked that the exception message was "Invalid task ID", as shown below:

//attempts to create new task when id is null

IllegalArgumentException thrown = Assertions.assertThrows(IllegalArgumentException.class, () -> {

new Task(null, "Find G-Man", "Locate the G-Man for questioning");

});

//checks proper exception was thrown

assertEquals("Invalid task ID", thrown.getMessage());

In order to ensure that my code was efficient, I firstly created get methods for each feature to help me with iterating through the data structure. This reduced redundancy in the code from repeating iterators in each of the add, update and delete functions. getTask shown below:

public Task getTask(String taskID) {

Iterator<Task> iterator = tasks.iterator();

//While there are more tasks in array list

while(iterator.hasNext()) {

Task curTask = iterator.next();

//if match was found return the task

if(curTask.getTaskID().equals(taskID)) {

return curTask;

}

}

return null;

}

Additionally, I used the singleton pattern for the service classes. This helped me to keep the same data between different classes. Furthermore, it also allowed me to limit instances, in turn, reducing system strain. TaskService shown below:

private static TaskService instance = new TaskService();

private static List<Task> tasks = new ArrayList<Task>();

private static int nextTaskID = 1;

private TaskService() {

}

Reflection

Testing Techniques

The software testing techniques that I took advantage of during the development process was Unit and Integration testing. Unit testing is a form of testing where only individual units or components are tested at a time. For example, I tested, if the task name was updated to “Go To Sleep” then, when retrieving the task name it should also equal, “Go To Sleep”. Integration testing is a way of testing where we ensure that different components are working together as expected. For example, when we test the action of deleting a task from the system, I am testing the deleteTask function, as well as the getTask function. The delete function uses the get function to iterate through the data structure in search of the task to delete.

Some of the other testing techniques that were not used in testing this system are: regression and performance testing. Regression testing are performed on the system after making changes. Regression testing is important, since there is always a chance that newly added code could break pre-existing code. Secondly, performance testing is a form of testing where attributes such as speed and stability are tested under various loads. This is important for every system if the goal is to maintain a positive user experience among others.

All of these tests have their practical uses and implications. We’ve already touched on some of the details, but it is important to understand their use in different situations. Unit testing is most practical for individual components/units. Unit tests imply the benefit of improved code quality, maintainability and reliability. Integration testing, similar to unit testing, is practical for verifying each component's ability to interact with one another. This test implies that the system is reliable at handling data between its functions. Regression testing is practical for ensuring that new code changes do not introduce errors to pre-existing code. Regression tests are implied as being automated in order to perform the tests numerous times. Lastly, Performance testing is practical for testing a system under different loads and evaluating its stability. Successful Performance testing implies that the system meets the performance standards which also improves the user satisfaction (Software Testing Techniques, 2021).

Mindset

I needed to employ caution to ensure that I performed tests in a manner that proves the proper functionality. Sometimes as programmers, we may want to go to lengths to prove that our code works, even if it doesn’t. It was important for me to appreciate the complexity and interrelationships because through testing you are able to visualize how data can easily be altered in a system. For example, since I chose to employ the singleton pattern for each of my service classes, this meant that I had to share the service instance between the tests. The tests had to be performed in a specific order so as to prevent unnecessary complexity.

Through writing these tests, I needed to ensure that I avoided biases. I had to ensure that I added only features that were necessary to accomplish the project requirements; Even though it is tempting as a developer. I also took advantage of peer reviewing through my instructor. My instructor reviewed the code every iteration of the system. If I was to be testing my own code, I believe that bias would still take its toll on testing quality; Specifically, through confirmation and overconfidence biases. First, I may find myself focusing on test cases that confirm the correctness of the code. I might also be overly confident in the quality of my code and this could lead to overlooking an error (8 Cognitive Biases in Software Development, 2020).

Developers must know to not cut corners when testing code. Testing code is important for maintaining customer satisfaction in your product. If an error is present and users experience it, they are less likely to trust your system. Errors could also lead to a loss of time and money. In order to avoid technical debt, you must ensure that you are following best coding practices. Clear commenting, for example, decreases the time spent to review code when changes are made by another developer. Additionally, when scaling a system, it is important to ensure that you utilize automated tests so you can decrease time spent testing.

**References**

*8 Cognitive Biases in Software Development*. (2020, May 27). The Valuable Dev. https://thevaluable.dev/cognitive-bias-software-development/

*Software Testing Techniques*. (2021, February 22). GeeksforGeeks. https://www.geeksforgeeks.org/software-testing-techniques/

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.
         1. Tested for boundaries
         2. Tested for valid
         3. Tested for null input
         4. Tested for duplicate ID’s
      2. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?
         1. Overall 87.8
            1. Appointment : 83.4

Some additional methods that I had added. Setters for Date and Description

* + - * 1. Task: 86.2

Does not like the catch statements

Perhaps I should be testing for expected output

* + - * 1. Contact: 91.6
  1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.
        1. nCharString for testing valid values, this tested every possibility of length
        2. I would test that an exception is thrown when performing a task, after this I would test that the proper exception was thrown by comparing the error messages
        3. Clear Commenting, Head of the functional classes had requirements, Test classes were separated by type of tests
        4. The tests should be easily maintainable since they tests the basic functionality of these systems, at most they will have to be scaled to fit the needs of the system
     2. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.
        1. In order to keep the code clean and efficient I made an extra method in my service classes. These were my get(Task, Appointment, Contact) these allowed for me to iterate only create one iterator object for my add,update and delete functions.
        2. I used a loop to test the valid strings to ensure that no stray value will result in an error. I created an ncharString to aid in this so I could input the iteration and it would give me string of that length.
        3. I used the singleton pattern for the service classes, This helped me to keep the same data between the classes, without instantiating class variables.This also ensures that multiple class instances were not being instantialing and reducing resource costs

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.
         1. Unit Testing
            1. Practical Use

Ensuring individual units/components work as expected

* + - * 1. Implications

Improve code quality, maintainability, reliability and helps programmers to understand code better

* + - 1. Integration testing
         1. Practical Use

Verifies that components and units come together and work properly.

Ensures the components can communicate as expected

* + - * 1. Implications

Helps to ensure that the system can come together as a whole

Improves the system reliability

* + - 1. Manual Testing
         1. Practical Use

Testing without the use of automated tools

Tests for interfaces. Usability and other aspects that may be hard to automate

* + - * 1. Implications

Can uncover issues that automated tests cannot find. Visual inconsistencies, and edge cases, user experience issues.

Can be very time consuming

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
       1. Regression testing
          1. Practical Use

Ensures that new code changes don’t introduce new problems with pre existing code

* + - * 1. Implications

Can be automated to reduce manual effort and speed testing processes

* + - 1. Performance Testing
         1. Practical Use

Tests the performance of the system under various different conditions. Heavy load, high traffic

Can identify bottlenecks within the system

* + - * 1. Implications

Ensures that the system meets performance standards

Improves the user satisfaction, reduce downtime, and optimize resource usage

* + - 1. Security testing
         1. Practical Use

identifies vulnerabilities and weaknesses in the software that could lead to attackers breaching the system

* + - * 1. Implications

Helps to protect sensitive data, prevent unauthorized access, maintain the integrity and confidentiality of the system and keeping the system compliant on security standards

* + 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
  1. Mindset
     1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.
     2. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.
        1. Focus on objectivity
           1. Only adding needed features to accomplish project requirements
        2. Peer review
           1. Instructor checking my code under every iteration
           2. Checks for proper test coverage

Testing own code

* + - 1. Confirmation bias
         1. I may find myself focusing on test cases that confirm the correctness of my code. However this could leave out some of the edge cases
      2. Self-Justification bias
         1. I may find myself trying to push potential bugs out of the picture rather than solving them directly
      3. Overconfidence bias
         1. I might be be overly confident in the correctness of my code and through this could lead to me overlooking error

Mitigation

* + - 1. Structured testing approaches
      2. Seeking feedback
    1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.
       1. Maintaining Customer Satisfaction
       2. Ensuring reliability and Stability
       3. Preserving reputation.

Avoid technical debt

1. Follow best practices
2. Automated testing
3. Continuous Learning

DONT NEED?

* + - 1. @FIXMethodOrder(MethodSorters.JVM)
         1. Forces tests to be ran sequentially
         2. This helps for information to be shared between tests
         3. Ex: instead of creating new contacts to test the update function I could use existing contacts
      2. Assertions.assertThrows
         1. Allows for me to test for errors and keep the data available for more testing
      3. Assertions.assertEquals
         1. Allows for me to compare two different values, Strings in
         2. my case