CS-320 Project 2

Afshin E. Ahvazi  
CS-320-T2825  
Professor Tanisha Jacks  
December 11, 2022

### Figure 1 - System UML Diagram

## **Summary**

### 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.

My approach to this project has been very similar to test-driven development where, given the requirements, we create the test cases first and write code that passes those tests. Therefore, my code directly follows from and--with help from the Validations class--enforces the software requirements. As an example, consider the setPhoneNumber(String) method from the Contact class:

public void setPhoneNumber(final String phoneNumber) {   
 Validations.validateObjectNotNull(phoneNumber);   
 Validations.validateStringLengthEquals(phoneNumber, 10);   
 Validations.validateStringContainsAllDigits(phoneNumber);   
 this.phone = phoneNumber;  
}  
Copy

The Validations class houses a collection of static methods that ensure String literals have certain properties, and would throw run time exceptions otherwise. The requirements for Contact.phoneNumber are as follows:

* It should not be null
* It should be between 1 to 10 characters (inclusive)
* It should only contain digits  
  Consequently, Validations.validateObjectNotNull(phoneNumber) ensures the argument is not null. Validations.validateStringLengthEquals(phoneNumber, 10) ensures the length of String argument has the correct length. And finally, Validations.validateStringContainsAllDigits(phoneNumber) ensures the String argument only contains digits.

###### 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?

A high coverage percentage does not guarantee test quality. That is why, in addition to high metrics, I have strived to follow best testing practices diligently.  
While keeping the test cases as simple as possible, I have used meaningful names and descriptions (with the use of JUnit's @DisplayName annotation where necessary). Many Assertions also assume message arguments that explain the expected outcomes in detail.  
test case also focus on 1 system requirement only. And last but not least, I avoid squandering resources on testing trivial methods such as setters and getters, toString(), hashCode(), or equals(). While these methods are conducive to the the classes in this project, they are mostly one-liners that do not merit any additional testings.  
Data in the following table originates from Intellij's "Run with Coverage" feature. It provides insight into how thorough our test cases examine the Service classes.

| Test Class | Associated Class | Methods (% Coverage) | Lines (% Coverage) |
| --- | --- | --- | --- |
| ContactServiceTest | ContactService | 100 | 100 |
| TaskServiceTest | TaskService | 100 | 100 |
| AppointmentServiceTest | AppointmentService | 100 | 100 |
| -- | Service | 80 | 93 |
| -- | Validations | 100 | 100 |

### Describe your experience writing the JUnit tests.

###### 3. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

As a fundamental principle in software testing, absence of defects is a fallacy. This program is no exception. It is constantly going through changes as new test cases reveal defects.  
For example, there were originally no test cases for when any data members were assigned to null. I erroneously believed that JRE does not allow null to be passed as arguments, and therefore omitted those tests to save resources. In fact, null cannot be passed to literal types such as int, boolean, etc. in Java. However, objects, including those of type String, can be replaced with null when passed to method parameters. Through more testings, this defect was discovered and corrected relatively early in the project's life cycle. Validations.validateObjectNotNull(Object o) now checks for null in every setter methods in the project:

public void setTaskDescription(final String taskDescription) {   
 Validations.validateObjectNotNull(taskDescription);   
 Validations.validateStringLengthIsInRange(taskDescription, 50);   
 this.taskDescription = taskDescription;   
}  
Copy

###### 4. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

I believe there are two factors that play major roles in my program's efficiency.  
First, reusability. Early in this project's life cycle, I began seeing myself writing similar methods in the TaskService, ContactService, and AppointmentService classes. To avoid superfluous and inefficient code, I decided to refactor the common functionality of these methods in a Service superclass. (Refer to the UML diagram for this project.) A similar issue resulted in refactoring many authentication methods as the general, static methods that are now in the Validations class.  
Second, using a Map as the underlying ADT in ContactServie and TaskService methods. To illustrate, consider the TaskService class.  
Per the software requirements, this service class must manage a Collection of Task objects. Furthermore, each Task must have a unique id, indicating the Task objects are unique. At first thought, a Set may appear the best choice to store a collection of unique items. However, since the Set interface does not provide a get() method, we cannot simply retrieve and update the Task objects--another functional requirement for TaskService. We could use an Iterator and perform a brute-force search, but this approach is extremely inefficient for large Sets.  
Instead, we use a Map (more specifically, a HashMap) that connects taskID to Task. We can easily fulfill the requirements mentioned above using the Map.get() and Map.put() methods. Consequentally, managing taskID becomes the responsibility of the TaskService class. In other words, taskID is not an attribute of Task. As expounded, this change significantly improves the efficiency of the program.

## **Reflection**

### Testing Techniques

###### 5-7 What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

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

###### For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.

This project has undergone different levels of testing: from unit testing at the lowest level, to integration testing (that ensures related modules like Contact and ContactService work together as intended), to acceptance testing, which will be performed by my instructor (customer), to ensure the project meets the demands.  
The method used for testing are *white box testing* and *black box testing*. White box testing is manually inspecting the code for security exploits or memory leaks (Service classes manage an ArrayList of ServiceData), testing loop conditions and ensuring that every logical branch is error free. Since this system does not connect a databases, security may not contribute to risk. However, this testing method is conducive to a well-standing system structure.  
Black box testing focuses on input validation, making sure the system behaves as defined by its requirements.  
A few such techniques used are equivalence partitioning and boundary analysis. Because we cannot test every possible input, we categorize them in to groups that would yield the same output. For example, any String with length 1 to 10 is acceptable for Contact.firstName and can therefore be represented in an input group; any String longer than 10 characters is invalid and is part of another input group.  
The boundary between valid and invalid input is often where mistakes are made in Valiatations. Hence, for a test scenario concerning the length of Contact.firstName, we would design cases that test with 9, 10, and 11 character long Strings. This technique is referred to as boundary analysis, and is heavily used for testing the Validations class in this project.  
The final testing method is regression testing. When any of the testing methods and techniques discover a defect, regression testing ensures the fixes deployed have not broken different parts of the code.

An example of a testing technique not used for this project is penetration testing. This technique is often used in white box testing and it is about purposefully attacking a system to find security threats. Because security is not the main focus of this project, this technique was omitted.

### Mindset

###### 8. 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.

This project is my take-off into bonafide software testing. Previously, I would create temporary main methods to ensure my classes/methods would function as intended. As such, my mindset has always been to show absence of error. But since that is not possible to prove, a software tester's goal should be to find defects, not the lack thereof. In that aim, he/she would write tests that truly challenge the software, resulting in higher product quality. Attributes of such tests were discussed earlier in this document.  
While this is a relatively simple system, I have strived to reach the highest system design quality using OOP concepts (as demonstrated by the UML diagram of the system). Following onto test design, I have been careful to design meaningful tests that abide by testing principles and best practices. To do so, it is important to develop a deep understanding of the system under test (SUT), which is essential to white-box testing methods.

###### 9. 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.

As a developer, I tend to form a je ne sais quoi bond with my creation. It is not always easy to be responsive to feedback that suggests crumpling substantial code or making drastic changes. I can relate how this would lead to bias when developers are testing their own code. This demonstrates the necessity for independent software testers. Two heads are better than one, after all.  
Working alone on this project, the best that I can do in the face of bias, is to be amenable when flaws are found. I hope to have demonstrated this in my git commits; this system has undergone massive transformations since the beginning of the project.  
To create a more stable program, the system architecture was completely overhauled to include new packages and abstract classes. Since the Service classes (i.e. ContactService, TaskService, and AppointmentService) share a marked amount of code, I refactored much of that code into an abstract class named Service. An abstract class ServiceData was introduced for the purpose of grouping the Contact, Task, and Appointment classes together. Refer to the the UML diagram.  
Although these changes reflect on the initial poor design of the system, they also show perseverance for making improvements and rejecting developer bias.

###### 10. 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.

Computer software is ubiquitous. Consider your daily routine: your alarm clock, your coffee maker, the car you drive, purchasing groceries, etc.. The number of daily interactions we make with software is countless. A malfunction in either of those interractions is an inconvenience at least, and in the more extreme cases it can be deadly or cause serious harm (to people, the environment, businesses). That is why, depending on the risk involved, we cannot afford to hold back on testing.  
Furthermore, to avoid technical debt, we should start testing as early as possible in SDLC: that includes inspecting documentation stages as early as the requirement gathering or modeling. The longer we move in the wrong direction, the farther we are from the correct destination. This applies to all SDLC methodologies. Had these practices been followed in this project, there would have been no need to change the system architecture amid developement.