Zac Gonzalez

14 June 2021

CS - 320

Prof. Luo

**Project Two**

I learned a lot throughout this course. In the case of the Contact, Task, and Appointment services, I had to utilize multiple different tests to ensure they were coded correctly. All three of these services will be used together for the app that I developed for Grand Strand Systems. I believe that I programmed the services correctly, and they seem to fit well together.

In the case of the Contact service, I was asked to test the Contact.java class to ensure the ID, Contact, First Name, Last Name, Phone Number, and address objects matched. For each of these I utilized an “assertTrue” argument. This argument simply throws an error if the object doesn’t perfectly match what I have asked it to test. The following lines of code, for example, show how I utilized this test for the ContactID:

@Test

**void** testContactID() {

Contact contact = **new** Contact("C1", "Jane", "Doe", "5546678923", "Los Angeles, CA");

*assertTrue*(Contact.getcontactID().equals("C1"));

}

As we can see, I asked the test to ensure that the ID matched exactly with “C1”, otherwise throw an error. Now, in the case of the ContactServiceTest,java file, I essentially used the same “assertTrue” tests, however I also included a “test(object)Deleted” test as well. This type of test ensures that an instance of an object that was deleted actually got deleted. The following lines of code show exactly what I did:

**void** testContactSertviceDeleted() {

ContactService contactService = **new** ContactService();

Contact c2 = **new** Contact("C2", "John", "Deer", "12345678901", "New York, NY");

contactService.deleteContact("C2");

Assertions.*assertThrows*(IllegalArgumentException.**class**, () -> {

System.***out***.println(contactService.Contact(c2));

});

The next service I worked on would be the Task service. The test files in this service were labeled TaskTest.java and TaskServiceTask.java (it was supposed to be TaskServiceTest.java, but I accidentally incorrectly named it). For the TaskTest.java file, I utilized a “createValild(object)Data” test. This test ensures that a new instance of an object that I created follows along with the requirements of that class. The requirements for the object include the id, name, and description. The following lines of code display what I did:

@Test **public** **void** createValidTaskData() {

Task task = **new** Task("0000000001", "Reading", "Read Novel Book");

System.***out***.println(task);

}

As we can see, the test will ensure that valid information is entered for each requirement.

For the TaskServiceTask.java file, I utilized a few different tests. Firstly, I used a “valid(object)” test which works similarly to an “assertTrue” test. The following lines of code display how I utilized this test:

@Test **public** **void** validTaskData() {

Task task = **new** Task("0000000001", "Reading", "Read Novel Book");

addTask(task);

System.***out***.println("New Task: " + *tasks*);

System.***out***.println("size: " + *tasks*.size());

}

As we can see, I created a new Task and utilized this test to ensure that it is a valid object.

Lastly, I was asked to create the Appointment service. The test files within this service include the AppointmentTest.java and the AppointmentServiceTest.java files. Within the AppointmentTest.java file, I needed to ensure that the ID, Date, and Description objects were correct. I used both “assertEquals” and “assertEquals(Invalid)” tests in this file, but I also set up the file with a “setUp” arguiment. This allowed me to set up an instance of the Appointment class, then run tests on it. The “setUp” argument was coded using the following lines:

@BeforeEach

**void** setUp() {

appointment = **new** Appointment(currentDate, "9", "Dentist");

}

So, within this set up we created a new appointment and inserted information into each object instance. I then followed up with tests to ensure the instances were valid. One such test is as follows:

@Test

**void** setAppointmentDescription() {

*assertEquals*("Dentist", appointment.getAppointmentDescription());

}

In this test, I confirmed that the Description object had a valid input for this instance. For the AppointmentServiceTest.java file, I utilized a similar “setUp” argument. In this file, I used tests to confirm that instances of objects were added or deleted. An example of this is in the following lines of code:

@Test

**void** add() {

*assertTrue*(service.add(appointment));

}

This test was used to confirm that a new instance of the appointment object was indeed created.

As was the case with the Contact and Task services and the appointment service, unfortunately, there were minor errors that did not allow me to see the percentage of tests correctly performed. However, I am confident that the tests I utilized were accurate and useful. I believe I efficiently met the requirements set out for me for all required files.

The techniques I used within this project were Boundary Value Analysis (BVA) and Equivalence Class Partitioning. With Boundary Value Analysis, we set boundaries for an object, then test to confirm the instance falls within that boundary. With Equivalence Class Partitioning, we set an instance of an object to equal a specific value, then test to confirm the instance meets that exact value. Both of these techniques can be used greatly in the field as they help ensure information that is input is valid.

Techniques I did not use for this projects could be Decision Table Based Testing or State Transition. Decision Table Based Testing involves ensuring functions respond a certain way to a combination of inputs or events. State Transition involves testing the state of an Application Under Test (AUT). Decision Table Based Testing can be used to ensure “a submit button in a contact form is enabled only when all the inputs are entered by the end user.” ("Software Testing Techniques with Test Case Design Examples", 2021) State Transition can be used for password protection. For instance, if a user enters a valid password within the first three attempts, the user will be granted access. However, if, “the user enters password incorrectly 3rd time, the action has taken, and the account will be blocked.” ("Software Testing Techniques with Test Case Design Examples", 2021)

Throughout this project, I believe I did employ caution, limit bias, and utilized discipline to complete the project. I employed caution by appreciating the complexity and interrelationships of the code I was testing. For example, I knew that certain objects (such as the Phone Number object) required a specific length for an instance. I ensured that no instance of these objects could be longer or shorter than the specified length. Bias is something I do believe can affect any software engineer in the field. For instance, I believe if one is testing their own code, they may be more inclined to make it out as if their code is functioning properly, even if it is not. Being disciplined is important in any field. But, especially in computer programming, it is very important to avoid cutting corners. ”In general, approaching [computer] work with a tester’s mindset seems to mitigate or eliminate various types of problems” (Neopragma, 2019) As Neopragma stated, if we approach our work with a tester’s point of view, we learn to pay close attention to every detail, which, in turn, creates fewer problems down the line.

I have learned a lot of information in this course. I will admit, I am no where near a master of testing software, however I do feel confident enough to understand what needs to be tested and how to test it. As I’ve stated above, testing is imperative in any computer related field as we must eliminate issues before they grow to major problems. As tedious as software testing may be, it is one of the most important steps when programming.

**Bibliography**

Software Testing Techniques with Test Case Design Examples. (2021). Retrieved 16 June 2021, from <https://www.guru99.com/software-testing-techniques.html>

Neopragma. (2019). Testing as the core discipline of software delivery – NeoPragma LLC. Retrieved 17 June 2021, from https://neopragma.com/2013/06/testing-as-the-core-discipline-of-software-delivery/