Nathan Chuluda

​​[CS-320-T2646 Software Test Automation& QA 21EW2](https://learn.snhu.edu/d2l/home/897769)

Project Two

December 4, 2021

For this project, I was tasked with developing several backend services to be used in a mobile application. To be sure that the code I produced worked as expected, I wrote JUnit tests for each component of my code. My approach to this project started at the requirements as provided by the customer. Contact, Task and Appointment objects would need to be added, updated and deleted in the application, and each of these were defined with clear limits or requirements for values that could be accepted to these fields. As this project is relatively simple, I was able to first develop the code for these purposes. Next, I went through and wrote the tests for each attribute. Each attribute of these objects usually had two limitations: the field could not be null, and it could not exceed a certain length of characters. For this common pattern, a test would be written trying to instantiate an object with the attribute set to null. A second test would then test the field with a value exceeding the maximum length. If the code behaved as expected, this produces an invalid argument exception. The appointment object includes a date field, which requires a different test. This test checks the provided date to make sure that it is not in the past. By writing a test that checks for both a date that is passed and a date in the future, we can ensure the code accepts and rejects the appropriate inputs. These tests are important, as they provide a line of defense against buggy code from being integrated into the project.

To make sure that my code is technically sound, I was careful to keep each test as simple as possible. By only testing one functionality at a time, it ensures that the code will behave a certain way all the time, and not just when a certain combination or sequence of events takes place together. This simplicity is also beneficial when trying to achieve a high percentage of testing coverage. By writing tests in this organized way, it’s possible to hit the code from all angles required for sufficient coverage. For these tests, it was possible to reach 100% coverage due to the simplicity of the objects we were developing. Below, I’ll highlight the firstName field of the Contact object, which can’t be null, and must be less than 10 characters in length. The constructor for Contact uses this setter function to validate the input that is passed as a parameter. The tests I wrote verify that this input validation is functioning properly by calling the Contact constructor three times. First, valid input is passed to the constructor for all fields. Next, a firstName that is null is tested, and produces the expected error. Lastly, a firstName that exceeds 10 characters is tested, which also produces the expected error.

**Setter function includes input validation to be tested:**

public void setFirstName(String tFirstName) {

if (tFirstName == null) {

throw new NullPointerException();

} else if (tFirstName.length() > 10) {

throw new IllegalArgumentException("Invalid First Name");

} else {

firstName = tFirstName;

}

}

**Testing constructor with all valid input:**

@Test

void testContact() {

Contact contact = new Contact("12345678", "John", "Smith", "3132561234", "2627 Edwin St");

assertTrue(contact.getContactId().equals("12345678"));

assertTrue(contact.getFirstName().equals("John"));

assertTrue(contact.getLastName().equals("Smith"));

assertTrue(contact.getPhone().equals("3132561234"));

assertTrue(contact.getAddress().equals("2627 Edwin St"));

}

**Testing constructor with firstName.length() > 10:**

@Test

void testFirstNameTooLong() {

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

new Contact("12345678", "Cassanndrah", "Smith", "3132561234", "2627 Edwin St");

});

}

**Testing constructor with firstName = null:**

@Test

void testFirstNameNull() {

Assertions.assertThrows(NullPointerException.class, () -> {

new Contact("12345678", null, "Smith", "3132561234", "2627 Edwin St");

});

}

These tests cover the functionality of the code. They trace through the specifications as provided by the customer to ensure that they are met. This is just one iteration of testing that can come early in the development process. In the next phases, the nonfunctional requirements of the software must be tested. Nonfunctional testing ensures the usability of the system. An example of this kind of testing could make sure that even when the system has a high level of activity, it does not slow down to a point of unusability. Once a system has gone live, there may come a time when the code needs to be changed. Maintenance testing makes sure that any updates can be integrated smoothly across the system. Regression testing makes sure that changes to the code do not negatively affect existing code.

When working on this project, I found my attention to detail and careful organization to be essential for reaching success. My approach strived to keep the code simple while meeting the requirements. This simplicity becomes increasingly valuable as the complexity and interrelatedness of the components increases. In this project I was cautious to meet the requirements without introducing unnecessary complexities. In the codeset, the Service functions use an id when updating the other attributes. Should the id not be found, this prevents the updating function from completing its task. Not only do we have to ensure that the updated attribute passes validation, but also that the id already exists. While this example may still seem simple enough, it starts to show how a careful and attentive approach can be used to reach effective levels of coverage.

Even though one of the benefits of this style of testing is that it can be completed by the developer early in the project, the developers must be aware of any bias they may contribute to their code and tests. Often, someone who is impartial and unfamiliar to a set of code may approach it from a new perspective that the developer themselves had missed. This combination of discipline and collaboration can be used to reach great results.