Christopher Ivey

Southern New Hampshire University

CS-320-T3687 Software Test Automation& QA

February 24, 2022

Creating JUnit test cases, to test the methods and objects to determine whether the

program passed the test or not required having an analytical mindset. For the Contact, Task and

Appointment applications it was important to run test on the services that were being provided

by the code. To ensure that it passed, it was important to clean the code to run it and receive a

green bar, basically, saying it passed the Junit test. This ensured that the test coverage percentage

was high. To create the Junit test I employed assertions, or Boolean statements to test the cases.

Assert true that asserts that a statement is true is also a method I used for each milestone. To

utilize the assertion methods, the program required me to use the import statement: import static

org.junit.jupiter.api.Assertions.\*; With these import statements, it is important to be sure not

to have multiple import statements in the different classes, as this will cause an error by causing

the program to be confused on which import statement to use. This allowed me to use assertion

methods in my program to test whether two objects were equal by using the following method:

assertEquals("1", appointmentTest.getAppointmentID()); Another Junit testing technique that

was implemented into the creation of the Appointment Service classes was the use of Junit

Exceptions Test. It threw an Illegal exception argument that allowed the program to let the user

know if there, input was more than a specific character number. When developing the code, I had

some implications with determining which variables to use in certain methods and make sure that

they were calling the right constructors. The program was displaying codes all through the

development of the code. Throughout the Modules, one thing that stood out is that the @ignore

annotation was ignored through the building of the programs. The ignore test, will make sure that

that test method will not be executed when the program is running. When a test class uses the

@ignore annotation, then none of the test methods will be executed. Seems that this will come in

handy instead of having to comment out a lot of unwanted methods.

When writing the code for the Appointment class, it was obvious to see the

interrelationships and complexity of how they worked together. First, shedding light on the

@Before annotation that is used for methods for Java code to run after each test case.

@Before

public void createAppointment() {

as = new AppointmentService();

a1 = new Appointment (appointmentID, appointmentDate, appointmentDescription, appointmentName);

}

Then next by using the @Test annotation to set up the test cases for the Appointment.

@Test

public void testForAppointment() {

assertEquals("1", a1.getAppointmentID());

assertEquals("Take out the trash", a1.getAppointmentName());

assertEquals("Get all the trash from bins & leave them outside", a1.getAppointmentDescription());

To be successful with the creation of the applications and for them to be able to pass the

Junit testing it was important to acquire the mindset of what the test was designed to test for.

Determining these factors are critical to understanding what the outcome that was being targeted

when writing the Junit test. As a software developer it is important to have the mindset of

cautionary writing to ensure there are no errors in the code when referencing between various

classes in the same program. Like for example in the application I have made an instance of a: as

= new AppointmentService(); but in the class I didn’t set a variable with ***as.*** So, I then needed to

fix that by placing, private AppointmentService as; into the code. This cleared an error I was

getting on a method being unresolved. When operating in the role of a software tester it is

important to understand that while developing the test for the code to be mindful of the various

methods and libraries that are needed. Boolean operators were essential when testing between

two different objects to see if a statement is true or false. Making sure not to cut corners on

testing the cases before the deployment of a software is very important because it will save time

in the long run, on having to go back and fix errors in the code. It is less costly to fix the errors in

the beginning stages of the SDLC. So, running test to get rid of errors all throughout the

development of a software code will make sure the program is written more efficiently. While

creating the test, it was important to have the mindset of writing passing and failing test cases.

García, B. (2017). This is so that a developer can be able to see the results for both failing and

passing test cases. It is important to comment out on of the cases when running, because if it isn’t

commented out it will stop the test from running correctly or spit out an error. Cutting corners in

the technical field of testing software is costly, and it is very important to get the kinks out of the

software before it is deployed to be used for the user, this will help to prevent technical debt.

Also, by using different testing methodologies it will help to provide support and resistance

against human error for testing code that was written by the same person.

**References**

García, B. (2017). *Mastering Software Testing with JUnit 5*. Packt Publishing.