While using the JUnit Testing, I did everything I could to meet all requirements. For each aspect of the Contact class, I had multiple classes. These classes included covering the string length if it was exact, less than, or greater than the perimeter that was given. I also validated that the strings could not be set to null. As a precaution, I also validated the string was set to what it was assigned and didn’t somehow switch to null.

When I checked the coverage, I could see a percentage of the file I was testing. It would show how much of the file was being tested based on the tests that I used. My Contact file had null checks to validate that null could not be allowed and would be handled by throwing an exception. If I did not create a test that set the string to null it would not enter that exception code. This would go against the coverage percentage and would lower it. Therefore, I would also write a test that would purposely throw the exception.

I would create conditional statements that throw exceptions to help ensure the quality of the code. The exceptions are used to help stop any incorrect parameters. I used them to ensure there were no null strings as well as the strings were the appropriate length.

public void setFirstName(String name) {

if ((firstName == null) || (name == null) || (firstName.length() > 10) || (firstName.length() < 1)) {

throw new IllegalArgumentException("Invalid First Name Length");

}

firstName = name;} This is an example of the condition that ensures the quality of the code.

The tests that I created are what help ensure the efficiency of code. The tests are used to test each possible case, so we would know that what we get is the expected outcome.

void testContactAddressTooLong() {

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

new Contact("Jane","Doe","5555555555","NOPE New-Edu Ave New Township is too long now","1");

});

}

//Tests that contact is created with less than 30 character last name(no exception thrown)

*@Test*

void testContactAddressLessThan30() {

Contact contact = new Contact("Jane","Doe","5555555555","School Edu Ave","1");

*assertTrue*(contact.getLastName().length() < 30);

This is an example of the coded test that helps insure the efficiency of the program. There is both a test that an exception is thrown as well as a test that proves that it does what it is supposed to do.

**Citations**

García, Boni. (2017). Mastering software testing with JUnit 5. Packt Publishing.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). Software testing: An ISTQB-BCS certified tester foundation guide - 4th edition. BCS, The Chartered Institute for IT.

*Junit - Test Framework*. Online Tutorials, Courses, and eBooks Library. (n.d.-a). https://www.tutorialspoint.com/junit/junit\_test\_framework.htm