4-2 Journal: Unit Testing Approach and Writing JUnit Tests

Michael Wood

SNHU

Course Number: CS-320 Software Test, Automation

Instructor: Tanisha Jacks

Due Date: 03/30/2025

4-2 Journal: Unit Testing Approach and Writing JUnit Tests

Before providing the program to the client, it is crucial to do testing to identify any faults or problems and make sure all software requirements are satisfied. This will ensure that the product is of high quality. Also, the software becomes more dependable and user-friendly as a result. High efficiency, dependability, and performance are hallmarks of software that has undergone extensive testing. By using JUnit for testing, developers can be sure their code will work as expected for clients.

I double-check my code for mistakes in addition to using Junit testing. Your code might not function correctly due to simple issues like case sensitivity or misspellings. The import statement might not have been used correctly or perhaps it was omitted entirely. Before moving on to Junit testing, I do my best to fix as many errors as possible.

For example:

boolean findRepeat = list.stream().anyMatch(c -> c.getcontactID().equals(ContactID));

is not the same as:

boolean findRepeat = list.stream().anyMatch(c -> c.getcontactID().equals(contactID));

Just as I cannot create an ArrayList without first importing java.util.ArrayList;

This is done by using the following command: import java.util.ArrayList:

I did my best to separate my tests into smaller units so I could test each section of my code as shown below:

*@Test*

void testTaskGetters() {

Task task = new Task("1234567890", "Task Name", "Task Description");

*assertTrue*(task.gettaskID().equals("1234567890"));

*assertTrue*(task.gettaskName().equals("Task Name"));

*assertTrue*(task.gettaskDescription().equals("Task Description"));

}

*@Test*

void testTaskSetters() {

Task task = new Task("123456789", "Task Name", "Task Description");

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

task.SettaskName(null);

});

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

task.SettaskName("Name is too Long 20 Character Max.");

});

By using getters to retrieve the information that I want to test then using the setters to set the testing parameters. In the above example, I am testing that the Task Name is not null and if so, it throws IllegalArgumentException and that the Task Name is not more than 20 characters as per client’s requirements, otherwise it throws an IllegalArgumentException. Now, I have done similar tests for Task ID and Task Description.

Overall, I think that the quality of my Junit tests was effective testing my Contact class but did not completely cover every line of code as shown below:

A screenshot of a computer

AI-generated content may be incorrect.

Contact Service test covered 64.6% of the instructions. The Task Test covered 28.6% of the instructions. The Task Service test covered 74.4% of the instructions. By checking the coverage, it lets me know how much of my code is being tested and, in this case, I need to work on designing more comprehensive Junit tests in the future.

After running the Junit tests, I do not believe that my code is technically sound and needs some work, unfortunately a lot of work. My code is not efficient because it is not technically sound. It does not produce the wanted output. This means that I need to go back through and find the errors and misleading code and either remove code that is unused or change the code to work correctly.