Savion Peebles

CS-320

Professor Lewis

12/9/23

Project 2

While working with this model, the service classes and the sub testing classes I did the best I could to apply the Junit testing practices we learned throughout our time in the course. My approach to this was acting as a typical software engineer and using my knowledge while also learning new things along the way to better the way I program and find a solid way to test my code for functionality. The way I used the Junit tests was to test the boundaries of certain specifications that were set by the requirements of our software. An example of this is for one of the tasks we had to make sure the full name of our client could not be longer than 20 characters, null and mandatory, in order to fully use the service. In order to test this function specification, I did

if (fullName == null || fullName.length() > 20 || fullName.equals("")) {return false; }

Below I tested a valid version of an empty string, a null string and a string where the full name is too long.

Assertions.assertThrows(IllegalArgumentException.class, () -> {tempTask.setName("");

Assertions.assertThrows(IllegalArgumentException.class, () -> { tempTask.setName("This name is too long and is not valid for our software");

My experience with the Junit testing was slightly shaky but as time went on, I felt like I understood the concept of what it is used for and how valuable it can be when it comes to checking the functionality of your code and how to find ways to manage and fix it. I unfortunately could not get any of the tests done but through research and the resources proided through class materials I was able to piece together the concepts of the project to still allow me to get the desired results for the software. Ideally when working on future projects I would like to actually get the Junit testing to work because it can be a very big part of software development and it would greatly benefit me to have that in my skill set.

Since I couldn't get the Junit testing to fully work, I continued to check if the software was technically sound by checking one of my methods in the ContactService class that will delete an ID. This was done by adding three objects to a collection, deleting one of them and then confirming that the collection's size is now two objects, after that you then check and see if the desired object is deleted. One of the downfalls of doing this was if the collection size is correct then the wrong item could still be deleted as long as the size is correct, so it is important to know which object you are getting rid of.

```
@DisplayName("Test deleteContact")
@Test void testDeleteContact() {
String firstName = "Savion";
String lastName = "Peebles";
String phoneNumb = "1231834";
String address = "8939 Fairway Dr";
boolean testBool = false;
ContactService test = new ContactService();
assertTrue(ContactService.contactList.isEmpty());
test.addContact(firstName, lastName, phoneNumb, address);//obj ID 0
test.addContact(firstName, lastName, phoneNumb, address);//obj ID 1
test.addContact(firstName, lastName, phoneNumb, address);//obj ID 2
assertEquals(3,ContactService.contactList.size());
test.deleteContact("1");
assertEquals(2,ContactService.contactList.size()); // 1st test
//loops through to look for an ID
//2nd test
for(int i = 0; i < ContactService.contactList.size(); i++) {
if(ContactService.contactList.get(i).getContactID() == 1) {
testBool = true;
        }
}
assertFalse(testBool);
}
```

When working with these collections it isn't enough to just check the size of it after we delete an object. To ensure that the correct thing is deleted, and it is truly gone is also important. These two lines of code ensure that the objects are instantiated only when they are requested.

```
ContactService test = new ContactService();
assertTrue(ContactService.contactList.isEmpty());
```

These three lines of code allow me to maintain a failed string update to return it to as it was before the update was made.

```
tempTask.updateTasks("1", fullName, "New description"); //Invalid ID
assertNotEquals("New description", TaskService.tasks.get(id).getDescription());
assertEquals(fullName, TaskService.tasks.get(id).getName()); // The Original String
```

While writing the tests I decided that I wanted to implement white box testing for the software of the contact service. Keeping the requirements of the project in mind while working on the software was essential to the approaches I decided to take because in a way the requirements will make the choices for you a lot of the time. While testing it was also critical to think about possible scenarios users could run into so I could begin to test the system in ways it would be constantly used.

Working on this project tapped into a whole new mindset for me. While I have been working with software for a little bit now this was something new to me and it required me to think outside the box to solve my problems whether they were simple or complex. When a challenge was posed to me while my Junit testing was not going properly it was really relieving to see myself preserve and not give up on the whole project just because one thing wasn't going as planned. After time I really started to find my footing and understand the content I was working with even if things weren't going as well technically. There were a couple times where I used caution throughout the project such as using different values for testing and including multiple tests in order to maintain things such as the collections in the ContactService class when deleting a contact. Sometimes I could've applied some caution, some of my objects were unnecessarily long or included extra things they did not need, the more clear and concise code is the better. It is important to appreciate and consider the interrelationships of code because it is something that is worked with so often in the career field and is a massive part of software development, nearly every program runs dependently on each other whether it be classes inside of a program or two programs interacting with each other as independent actors. Ways I could limit bias in my own code is looking around the internet and through other resources I have and finding ways to improve how I write my code whether it be making it more concise, adding more comments or taking different approaches to concepts I've already learned. I happen to have a habit of adding a little bit more than necessary so even knowing that in the first place is a good step to addressing bias in my own work because I know that my code is far from perfect and there is many ways I can improve and better my skill set. Being disciplined in your approach while programming is extremely important because cutting corners will only get you in deeper trouble in the long run. If you deviate from your plan and your requirements without good reason in order to save time or effort it could cause problems for you later on in the project whether it be functionality or trying to connect a program or class to someone else’s work. Staying disciplined while learning also helps you develop good habits for future employers when you get to the career field, making you a suitable candidate compared to someone who has messy, rushed code. I also believe the amount of effort you put into your code is a direct reflection of your knowledge and work ethic so to appease myself, coworkers and administrators it is always in your best interest to put your best foot forward and code up to industry standards.