7-2 Project Two  
Shelayah Robinson

The unit testing approach, I applied for all three features, was an example of white-box testing. Considering I had to rewrite majority of my assignments in order than it worked. Which I did a bit of research for additional help in writing the service classes and tests. Also adding changes to for the three regular classes. From the software requirements, I made sure I replicated what was being asked of me. For example, when writing the Appointment class, I was asked to make sure that the appointment ID wasn’t longer than 10 characters, and that it wasn’t null an updatable. I created private variables as well creating setters and getters, and then lastly a constructor that ties everything together. The setters, I incorporated an if statement that if the input is null, then it will show an error message or past a certain number of characters, if it isn’t null then it should add the requirement. And this was applied for all the other classes I’ve written as well.   
 The overall quality of the JUnit tests would probably be 50%. I hadn’t realized I was supposed to test for failed tests, I mainly tested the project for when it worked only.   
This round of writing Junit tests was a bit more technically sound, I figured out how to write a bit better test that ran according to the requirements. The only thing I think I could improve on is testing for a scenario that fails. I ensured that the code was efficiently written by writing the services a different way. I had initially written the service as a hashmap, which I think I could have made it work, but in this situation, I couldn’t figure out how to get it to run. I ended up looking at different examples of how others wrote code, and once I had my own version of code, I had an easier time writing Junit tests. For the tests, I wrote:

**void** testdeleteTest()

{

ContactService service = **new** ContactService();

service.newContact("1234", "Adam", "Sanderson", "123-123-4321", "8932 Bellway Dr");

service.newContact("1334", "Amy", "Millie", "122-123-4321", "8932 Anderson Dr");

service.newContact("1244", "Krystal", "Amvbers", "153-123-4321", "8932 New York Dr");

*assertEquals*(3, service.getContactList().size());

service.deleteContact("1234");

*assertEquals*(2, service.getContactList().size());

service.deleteContact("1334");

*assertEquals*(1, service.getContactList().size());

service.deleteContact("1244");

*assertEquals*(0, service.getContactList().size());

}  
This test to makes sure that the contact is deleted; which works in this scenario.

**void** testNewContact()

{

ContactService service = **new** ContactService();

service.newContact("1234", "Adam", "Sanderson", "123-123-4321", "8932 Bellway Dr");

Contact contact = service.getContactList().get(0);

*assertEquals*("1234", contact.getContactID());

*assertEquals*("Adam", contact.getFirstname());

*assertEquals*("Sanderson", contact.getLastName());

*assertEquals*("123-123-4321", contact.getPhoneNum());

*assertEquals*("8932 Bellway Dr", contact.getAddress());

}

This tests to make sure that the contact is created; which works in this scenario.

As I have mentioned before, I have written the tests with the mindset of wanting to implement the white-box testing for programming the service. I knew the requirements of the project and I kept it in mind while I was writing the software. With this also in mind, I did try to think about scenarios that would be possible when creating testing. But this was one area I needed to sketch out more.

When working on this project, there was a mindset I had to adopt. It was essentially a new way of thinking, so that I could run tests, and apply it to the code that I had written. This way of thinking was a bit foreign to me as with anything new, but after some time, I was able to get the hang of it. I also realized while working on this project, I applied little to no caution. I did consider making global variables private instead of public. I also didn’t take into consideration different possible values for testing. I did take into consideration whether the objects I created passed, but I didn’t check for if the failed. I also realized I created unnecessary objects that didn’t need to be included within some of the services I’ve written.   
 I think I would have to be concerned with the question, have I written this code cleanly and with no ambiguity, and unnecessary things I have included. I noticed as I’ve progressed in programming, this was something I’ve had trouble with. I would write a part of code the longer way, when there was a better and concise way to write it.   
 It is absolutely important not to cut corners when writing or testing code because you are learning how to write it for future employers. Lack of care in programming shows the personality and work ethic of the developer. Which reflect negatively on them, whether intentional or unintentional. I think the best way to avoid practitioner in the field is to practice.