1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.
      2. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

My approach was directly aligned to the software requirements.

For Contact Class requirements:

* The contact object shall have a required unique contact ID string that cannot be longer than 10 characters. The contact ID shall not be null and shall not be updatable.
* The contact object shall have a required firstName String field that cannot be longer than 10 characters. The firstName field shall not be null.
* The contact object shall have a required lastName String field that cannot be longer than 10 characters. The lastName field shall not be null.
* The contact object shall have a required phone String field that must be exactly 10 digits. The phone field shall not be null.
* The contact object shall have a required address field that must be no longer than 30 characters. The address field shall not be null.

For Contact Service requirements:

* The contact service shall be able to add contacts with a unique ID.
* The contact service shall be able to delete contacts per contact ID.
* The contact service shall be able to update contact fields per contact ID. The following fields are updatable:
  + firstName
  + lastName
  + Number
  + Address

For Task requirements:

* The task object shall have a required unique task ID String that cannot be longer than 10 characters. The task ID shall not be null and shall not be updatable.
* The task object shall have a required name String field that cannot be longer than 20 characters. The name field shall not be null.
* The task object shall have a required description String field that cannot be longer than 50 characters. The description field shall not be null.

For Task Service requirements:

* The task service shall be able to add tasks with a unique ID.
* The task service shall be able to delete tasks per task ID.
* The task service shall be able to update task fields per task ID. The following fields are updatable:
  + Name
  + Description

For Appointment requirements:

* The appointment object shall have a required unique appointment ID string that cannot be longer than 10 characters. The appointment ID shall not be null and shall not be updatable.
* The appointment object shall have a required appointment Date field. The appointment Date field cannot be in the past. The appointment Date field shall not be null.  
  *Note: Use java.util.Date for the appointmentDate field and use before(new Date()) to check if the date is in the past.*
* The appointment object shall have a required description String field that cannot be longer than 50 characters. The description field shall not be null.

For Appointment Service requirements:

* The appointment service shall be able to add appointments with a unique appointment ID.
* The appointment service shall be able to delete appointments per appointment ID.

I directly made tests to reflect these requirements listed above. As for the Junit testing they all passed the corresponding tests for each class. The only exception that was thrown was within Appointment.

-Contact test

-Contact Service test

-Task Test

-Task Service Test

-Appointment Test

-Appointment Service Test

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.
     2. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

I ensured my code was technically sound by writing each of the requirements one at a time and then running the junit test case. For example within the contact class there were 20 tests within the junit test.

I first started with the ID requirements first. I passed it one char:

*@Test*

*@DisplayName*("Accepts ID with length of 1")

void IdWithOneCharTest() throws IllegalArgumentException {

Contact contact = new Contact("1", "Joe", "Dirt", "5555555555", "Test Descript");

*assertEquals*("1", contact.getID());

Then 10 chars to see if it was valid:

*@Test*

*@DisplayName*("Accepts ID with length of 10")

void IdWithTenCharTest() throws IllegalArgumentException {

Contact contact = new Contact("1234567890", "Joe", "Dirt", "5555555555", "Test Descript");

*assertEquals*("1234567890", contact.getID());

If the ID was more than 10 char then it would throw an exception:

*@Test*

*@DisplayName*("ID more than 10 characters throws exception")

void IdWithMoreThanTenCharsTest() {

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

new Contact("12345678901", "Joe", "Dirt", "5555555555", "Test Descript");

});

If the ID was null it would also throw an exception:

*@Test*

*@DisplayName*("Null ID throws exception")

void NullIDTest() {

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

new Contact("", "Joe", "Dirt", "5555555555", "Test Descript");

});

I followed a similar route down the requirements list ensuring after the one was written It passed.

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

The first two main techniques we learned about and employed are the Dynamic and Static tests. For Static we simply look at the code to see if there are any errors and fix/optimize them as needed. For Dynamic we look at the code once its been run to see if we are getting the behavior/output that we expect. If it is not what we expect than we fix the code as needed.

The Following technique I found really interesting and helpful was the implementation of Junit testing. This helps us look at the code deeper as we have to come up with tests specific to the requirements. While writing these we are working very close to our code so we can see not only how it is written but how we want it to perform.

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

**Performance Testing – testing the performance and resilience of our program(listed the types of performance tests below)**

-load testing

-Stress testing

-Endurance testing

**Security testing – testing the security of our program to make sure its not susceptible to being compremised(listed some security tests below)**

-integrity testing

-Confidentiality testing

-authentication/authorization testing

-availability testing

* + 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.

Depending on what our requirements are and the practical use of the application we can employ other types of tests. I could for see doing some dependency checks and tests on any of the packages we may potentially use incase we need to update them to a more secure and up-to-date version. This would ensure a certain level of security is met. Things such as bringing other form factors or pieces of code we need to integrate we may need to run integration tests to make sure they all fit together. I believe all applications should have some form of security testing along with performance testing done as this will give us the information on how the app performs in real time along with how secure it is. Really it all boils down to what and how we want to apply our application to. We want to make sure we provide a safe secure product that meets all the requirements of our customer/consumer.

* 1. Mindset
     1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

It took a methodical mindset to approach this project. Having a sense of direction and what needs to be accomplished first helps, along with having the list of requirements so we know what needs to be done. Prioritization is what will help along the process. As for caution I didn’t want to just do everything at once. Its like a puzzle and we need to put it together one piece at a time.

* + 1. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

Bias along with ego, when going through the project we often want everything we have done to be the way and have it be right. I think it takes an open mind to address what the short-comings are, where we have faults. These are not the end of the world but often are the things that make our coding and practice stronger. If we do not have these issues we cannot get better. In my mind even with coding failing can be see as an opportunity to make ourselves better.

* + 1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

Whenever I am working on a project it always takes a level of discipline. The more we have the more we can accomplish. If we were to cut corners when writing or testing it will show in the outcome of the over all product. In order to avoid these issues we must budget our time, work diligently, have proper planning so we can execute what is needed to be done. For me I start every week by looking at what the prompts are and what needs to be done. I try to plot time around my week for what is needed. I like to also leave myself extra time as things don’t always go to plan.