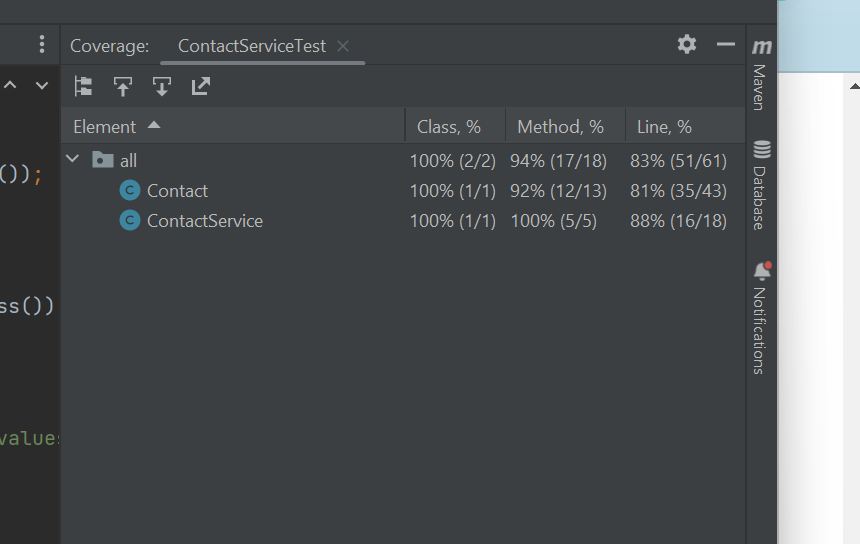
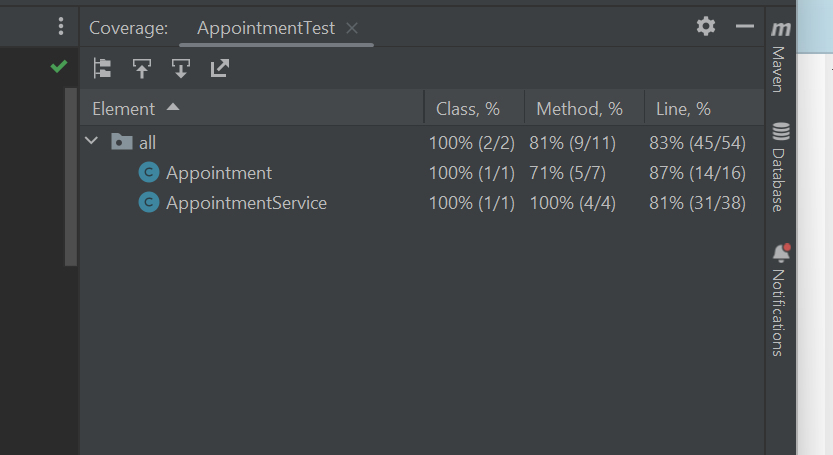
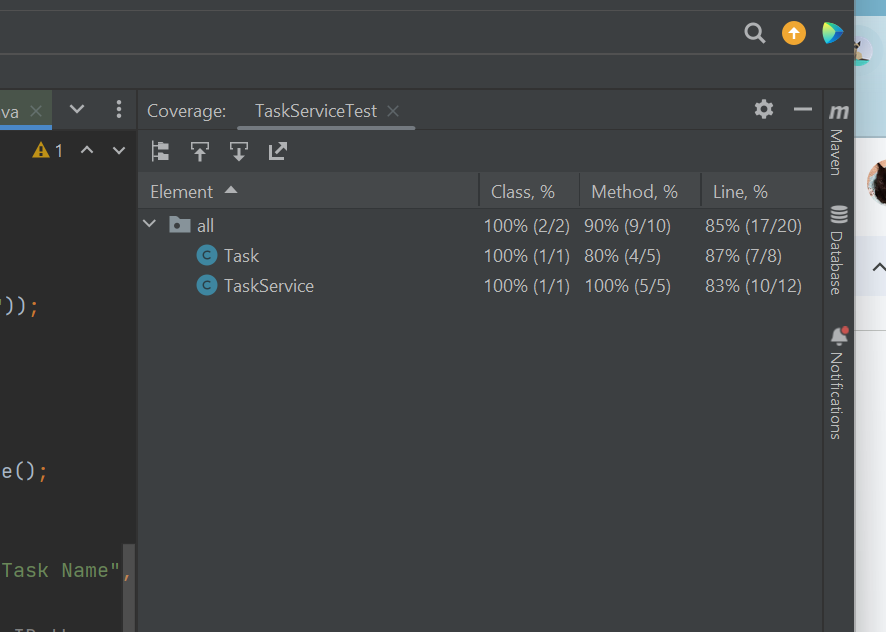
1. Summary
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.

My approach was to test cases that would be experienced by the code in use. The code in the appointment and appointment service section was used to create, edit, and delete appointments. I tested all of those functions. The code in the contact and contact service section was used to create, edit, and delete contacts. I tested all of those functions as well. Finally, the code in the tasks and task service section was to add, edit, and delete tasks. Each function had several tests to ensure it works. I tested expected and unexpected inputs for each field of each section.

* + 1. 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?

The coverage of all of my tests was at least 80% for each segment. That means at least 80% of the code was used in test cases. 80% is a good standard to go by since as projects get more complex, it becomes incredibly difficult to meet 100% coverage.



* 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.

I ran tests of each part of each section. I tested that the fields had the proper strings and tested to make sure it would reject any that weren’t correct. If a test failed, I would first look at the test and see if that was the problem. If the test was sound, I’d go back to the code and see the problem there.

void testContactFieldsMaxLength() {

// Test for first name too long

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

new Contact("0123456789", "FirstNameTooLong", "LastName", "5555555555", "Address");

});

// Test for last name too long

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

new Contact("0123456789", "FirstName", "LastNameTooLong", "5555555555", "Address");

});

// Test for address too long

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

new Contact("0123456789", "FirstName", "LastName", "5555555555", "AddressTooLong12345678901234567890");

});

}

* + 1. How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.

I made sure my code is efficient by making sure the service classes were using methods from their counterparts. The get(Blank) getter was able to be used in the service class though it was defined in the contact class. These tests passed, proving that it worked as expected. This is the contact service class test.

void testUpdateContact() {

Contact contact = new Contact("1234567890", "John", "Doe", "5555555555", "123 Main St");

contactService.addContact(contact);

// Update firstName

contactService.updateContact("1234567890", "firstName", "Jane");

Assertions.*assertEquals*("Jane", contactService.getContacts().get("1234567890").getFirstName());

// Update lastName

contactService.updateContact("1234567890", "lastName", "Smith");

Assertions.*assertEquals*("Smith", contactService.getContacts().get("1234567890").getLastName());

// Update phone

contactService.updateContact("1234567890", "phone", "6666666666");

Assertions.*assertEquals*("6666666666", contactService.getContacts().get("1234567890").getPhone());

// Update address

contactService.updateContact("1234567890", "address", "456 Elm St");

Assertions.*assertEquals*("456 Elm St", contactService.getContacts().get("1234567890").getAddress());

}

}

The getters and setters are defined in the contact class

// Getters

public String getContactID() {

return contactID;

}

public String getFirstName() {

return firstName;

}

public String getLastName() {

return lastName;

}

public String getPhone() {

return phone;

}

public String getAddress() {

return address;

}

// Setters

public void setContactID(String contactID) {

if (contactID == null || contactID.length() > 10) {

throw new IllegalArgumentException("Contact ID must not be null and cannot exceed 10 characters.");

}

this.contactID = contactID;

}

public void setFirstName(String firstName) {

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

throw new IllegalArgumentException("First Name must not be null and cannot exceed 10 characters.");

}

this.firstName = firstName;

}

public void setLastName(String lastName) {

if (lastName == null || lastName.length() > 10) {

throw new IllegalArgumentException("Last Name must not be null and cannot exceed 10 characters.");

}

this.lastName = lastName;

}

public void setPhone(String phone) {

if (phone == null || phone.length() != 10) {

throw new IllegalArgumentException("Phone number must not be null and must be exactly 10 digits.");

}

this.phone = phone;

}

public void setAddress(String address) {

if (address == null || address.length() > 30) {

throw new IllegalArgumentException("Address must not be null and cannot exceed 30 characters.");

}

this.address = address;

}

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

For this project, I mainly used JUnit testing. This is a type of testing for automated tests in java. This generally has the code divided into segments, or units. The test framework has many features such as specifying test cases and recording results. As you would expect, JUnit testing is a type of unit testing. I also used smoke testing to make sure things would run on a basic level such as simple typos etc. Early on, I made sure my JUnit tests were accurate by doing manual testing until I got the hang of it.

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

Techniques I did not use are more complex types such as end to end, acceptance, functional, performance, or integration tests. These all require a more complex code with multiple systems to be running. End to end runs the full length of the user experience on the software. Acceptance checks if use cases meet requirements. Performance runs to test the workload and strain. Functional checks only the output and if it meets requirements. Integration ensures all components work well together.

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

For a complex program like a completed phone game, you would want thorough testing before it goes public. Unit and smoke testing is the basic bare minimum. There will be real users so that requires end to end. The company that hired you will also have some requirements for you to meet which would use acceptance and functional testing. The game would be power consuming so you would want to run performance tests. Finally, you would want integration tests to ensure you did quality work.

* 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.

As a software tester, I wanted to find the faults. If I find them they can be fixed. It was important to look at the connections to find any weak links that would mess the whole thing up.

* + 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.

I tried to look at the code objectively, it is not something to try and keep safe but something you want to try and break. If you break it with testing, you can fix it and it will be much stronger than before. Bias would definitely be a concern if I was working for a company and I was afraid of getting fired for having mistakes in my code. I would almost certainly go easy on it to avoid being fired. Even if it wasn’t intentional, a new set of eyes will usually catch something you don’t see so a separate tester would be beneficial.

* + 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.

There are many reasons to avoid cutting corners. The biggest is maintainability. If your code is entirely spaghetti, this is a term that means it is hard to read and usually poorly optimized, it becomes near impossible to maintain it and fix it when a bug is found. Another potential issue is scalability. If your code can handle a small amount of tasks fine with unoptimized code, that may not be true when it is scaled up. A final problem while not as severe is that it would be incredibly frustrating to people working on the program with you. You want to be even more careful testing than writing the code. If you do good tests, you will find the problems. If you don’t put effort into them, there will almost certainly be consequences later. When I am working in the field, I will strive to keep my code organized and efficient. I won’t take the lazy way and end up with spaghetti code for others to try and decipher. I will try to be vigilant with my tests as well.

Citation:

GeeksforGeeks. (2021, June 17). *Spaghetti code*. https://www.geeksforgeeks.org/spaghetti-code/