CS 320 Project Two

Huan Ai

10/19/2025

Summary

Describe your unit testing approach for each of the three features.

To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.

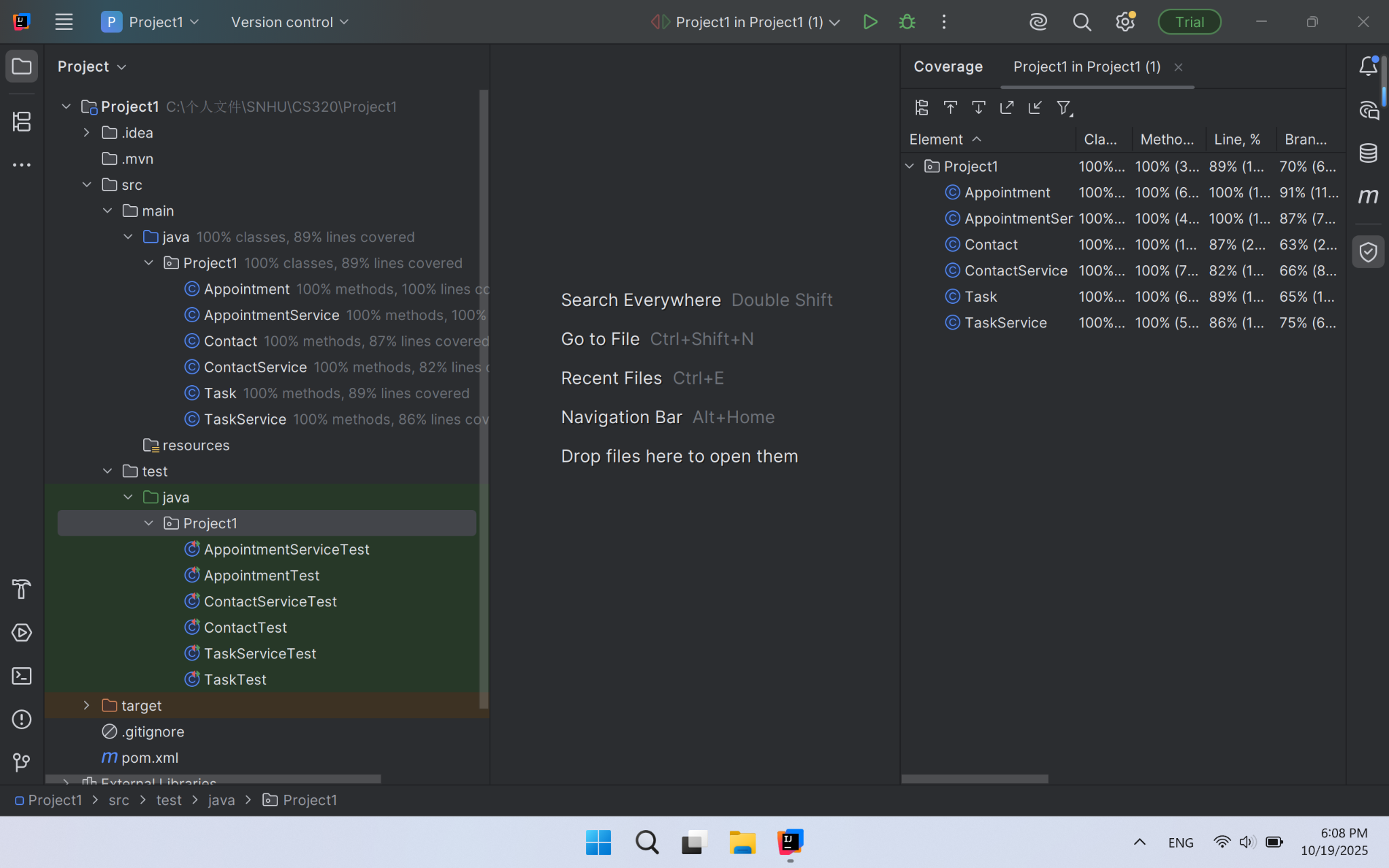
My testing approach was directly aligned with the requirements. The instructions gave me a specific list of what the Contact/ContactService, Task/TaskService, and Appointment/AppointmentService classes needed to do, and I wrote a test for every single one of those points—except for some small typos, where I forgot to test the "null" conditions on some of the properties (for example "TaskID", ContactID, First Name, etc.)

For the Contact class, the requirements were about validation: the contact ID, first name, and last name couldn't be longer than 10 characters, the phone number had to be exactly 10 digits, and the address couldn't be longer than 30 characters. I wrote tests that tried to break these rules on purpose. For example, in ContactTest.java, I have a test called testInvalidContactId() that tries to create a contact with an 11-character ID. The test expects the code to throw an exception, which proves my validation rule is working.

For the ContactService class, the requirements were to add, delete, and update contacts. My tests check each of these actions directly. The testAddContact() method verifies that a contact can be added, and testAddDuplicateContact() makes sure you can't add two contacts with the same ID. The testUpdateContact() method changes every field (first name, last name, phone, and address) and then checks that the changes were actually saved.

Same idea for Task/TaskService and Appointment/AppointmentService class. The requirements are very similar to Contact and ContactService classes. I just follow the requirements and create objects and tests that stick to the rules.

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?



I used the coverage tool in IntelliJ IDEA to see how much of my code was actually run by the tests. The report showed that my tests achieved at least 80% for all of the classes. This means that the vast majority of the code I wrote was executed during testing.

Describe your experience writing the JUnit tests.

How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.

I ensured my code was sound by writing tests that simulated both normal use and potential errors. The tests act as a safety net, proving that the code behaves as expected in different situations. For instance, this test in ContactServiceTest proves that the update feature works correctly:

service.updateFirstName("1234567890", "Jane");

Contact updated = service.getContact("1234567890");

assertEquals("Jane", updated.getFirstName());

This code doesn't just call the updateFirstName method; it also fetches the contact and checks that the first name was actually changed to "Jane". This confirms that the entire process works.

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

I focused on writing tests that were simple and direct, without any unnecessary steps. Each test has a single, clear purpose. This makes the tests themselves efficient and easy to understand.

For example, the testDeleteContact() is very straightforward:

service.addContact(contact);

service.deleteContact("1234567890");

assertNull(service.getContact("1234567890"));

It follows a clean pattern: Arrange (add a contact), Act (delete the contact), Assert (check that the contact is gone). There are no extra operations. It does one job and does it well.

Reflection

Testing Techniques

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

The primary technique I used for all three classes was Unit Testing with JUnit. Basically, I wrote small, focused tests for each individual "unit" of my code—meaning each class and its methods.

For Contact class, I wrote tests to check if a contact ID was exactly 10 characters, if the first and last names were not too long, and if the phone number was exactly 10 digits. My tests would fail if I tried to create a contact with an 11-character name or a 9-digit phone number. I also tested the service itself by adding a contact, deleting it, and then trying to update its fields to make sure everything worked as expected.

For Service class, the tests were very similar but focused on task properties. I made sure the task ID wasn't longer than 10 characters, the task name wasn't longer than 20 characters, and the description wasn't longer than 50 characters. The tests checked that the service could correctly add, find, and delete tasks based on their unique ID.

For Appointment class, this was similar to the task service, but with an extra check for the date. I wrote tests to make sure you couldn't schedule an appointment in the past. This was a new kind of validation that the other services didn't need.

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

I haven’t used integration testing and system testing in this project. For integration testing, since our services were simple and stored data in memory (like a simple list), we didn't need to test how they interacted with a real database or other complex systems. In a bigger project, we'd need integration tests to make sure all the different parts work together correctly. We didn’t use system testing either because we never tested the complete, fully assembled application. Our milestones were just the "backend" services. We didn't have a user interface (UI) to click through, so we couldn't test the entire system from a user's perspective.

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

The techniques we used, unit testing, are the foundation of building reliable software. They are used constantly by professional developers. Whenever a programmer writes a new function, they'll often write a unit test for it right away. This is practical because it catches bugs early, when they are cheapest and easiest to fix.

One of the techniques we didn’t use is integration testing. Integration testing checks how different modules or components of a software system interact with each other. It’s done after unit testing and before system testing. It can help to verify that interfaces between modules work correctly, and detect bugs in communication between modules.

The other technique we didn’t use is system testing. System testing evaluates the entire software system as a whole. It’s the final testing phase before release and simulates real-world usage. Users will test both functional and non-functional aspects (e.g., performance, security) to make sure everything works as expected before release. (GeeksforGeeks, 2025)

Mindset

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.

When I put on my "software tester" hat, my main goal was to break things. I had to stop thinking like the person who wrote the code and start thinking like someone trying to find every single way it could go wrong. I had to consider how all the little pieces of code worked together.

For example, in my ContactTest.java, I didn't just test that a good contact could be created. I thought about the complexity of all the rules: the ID, first name, last name, phone, and address all have different length requirements. I made tests to try and break each one. I asked questions like, "What if I try to create a contact with an 11-character last name?" and "What happens if the phone number has letters in it?". My test testInvalidPhone() checks that exact situation, making sure the code throws an error instead of accepting a bad number.

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.

The biggest way I fought bias was by following the requirements like a strict recipe. I didn't assume my code worked just because I wrote it. I tried to make it fail on purpose. I treated the requirements as the ultimate truth and tested against them line by line.

It's really easy to be biased when testing your own code. For instance, when I wrote the setAppointmentDate method in Appointment.java, I knew the rule was that the date can't be in the past. Because I wrote that logic myself, I might have been tempted to just test a future date and call it good. But to fight that bias, I specifically wrote a test (testAppointmentDateValidation in AppointmentTest.java) that tries to set a past date, making sure my own code would properly throw an error. I had to prove to myself that the safety net I built actually worked.

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.

Being disciplined about quality is everything. It's the difference between a program that works okay for now and one that is solid and won't cause problems later. "Technical debt" is like taking a shortcut on your homework—it saves you time now, but you'll have to pay it back later with interest when things start breaking. I plan to avoid technical debt by making testing a non-negotiable part of my work, not just a second thought. For every piece of code I write, I will write tests for it immediately. Just like in this project, I will always ask "what if?" and test the edge cases.

Citation

GeeksforGeeks. (2025, July 12). Difference between system testing and integration testing. https://www.geeksforgeeks.org/software-engineering/difference-between-system-testing-and-integration-testing/