CS 320 Module Four Journal

Huan Ai

9/28/2025

To what extent was your testing approach aligned to the software requirements?

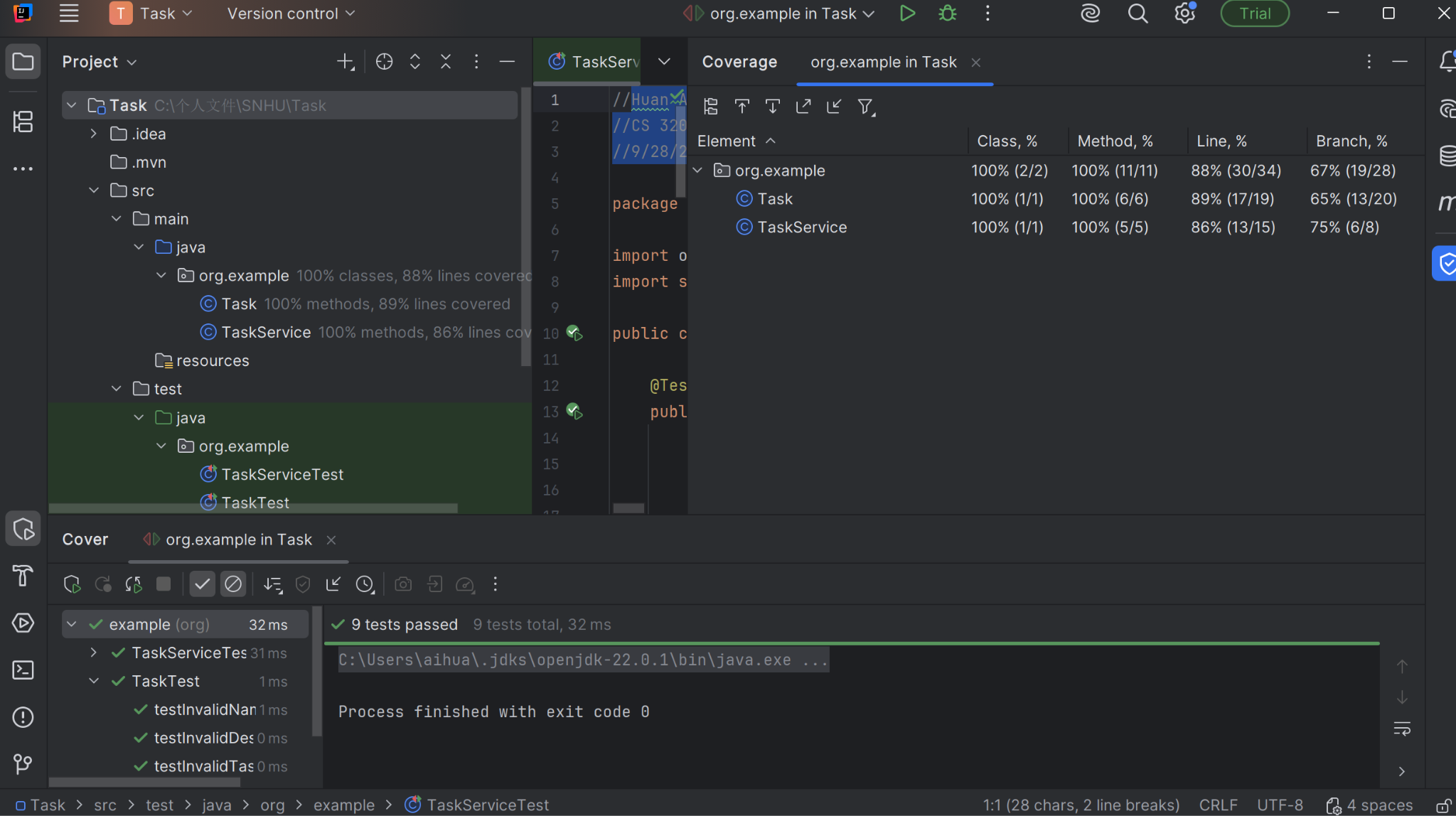
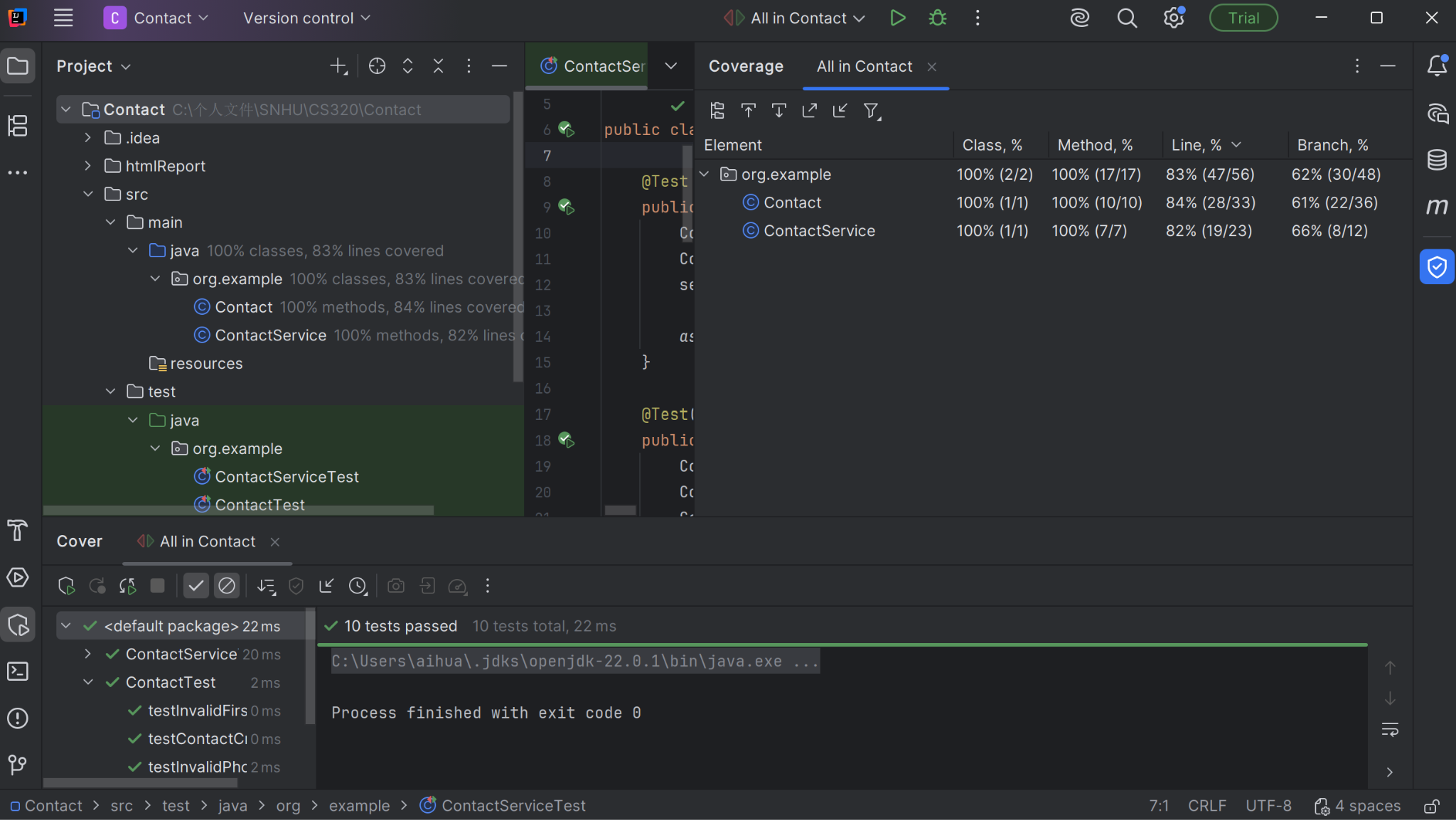
My testing approach was directly aligned with the requirements. The instructions gave me a specific list of what the Contact/ContactService and Task/TaskService classes needed to do, and I wrote a test for every single one of those points—except for one typo, where I forgot to write the test for validating the last name in the Contact class.

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 class and TaskService class. The requirements are very similar to Contact and ContactService classes. I just follow the requirements and created objects and tests that stick to the rules.

Defend the overall quality of your JUnit tests for the contact service and task service. In other words, how do you know that your JUnit tests were effective on the basis of 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 84% line coverage for the Contact class and 82% for the ContactService class. I also achieved 89% line coverage for Task class and 86% for the TaskService class. This means that the vast majority of the code I wrote was executed during testing.

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.