Southern New Hampshire University

CS 320

7-2 Project Two

Gianna Screen

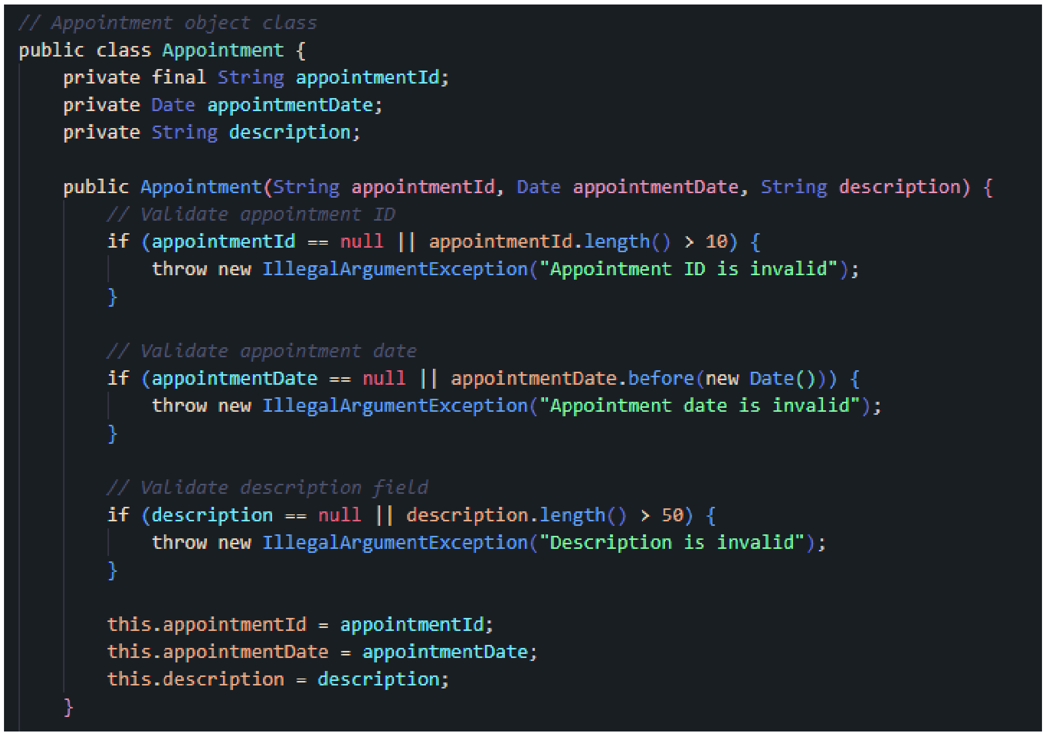
Summary

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

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

For this project, the software requirements were consistent and clearly defined, allowing me to ensure step by step that my program and overall approach aligned with each one. To meet every requirement, I used the provided rubric as a checklist while structuring my code to remain easy to navigate and modify based on test results and error handling. I also maintained consistent formatting across all main and test files.

My approach stayed fully aligned with the software requirements through specific implementations that directly corresponded to each listed requirement. For example, my Appointment class was structured according to the following requirements: the unique appointment ID could not be null, had to be no longer than 10 characters, and was not updatable. I met these requirements by declaring appointmentId as a final variable and validating it within the constructor.



```java
// Appointment object class
public class Appointment {
    private final String appointmentId;
    private Date appointmentDate;
    private String description;

    public Appointment(String appointmentId, Date appointmentDate, String description) {
        // Validate appointment ID
        if (appointmentId == null || appointmentId.length() > 10) {
            throw new IllegalArgumentException("Appointment ID is invalid");
        }

        // Validate appointment date
        if (appointmentDate == null || appointmentDate.before(new Date())) {
            throw new IllegalArgumentException("Appointment date is invalid");
        }

        // Validate description field
        if (description == null || description.length() > 50) {
            throw new IllegalArgumentException("Description is invalid");
        }

        this.appointmentId = appointmentId;
        this.appointmentDate = appointmentDate;
        this.description = description;
    }
```

Another example is the appointment date requirement, which required java.util.Date and restricted appointments from being scheduled in the past. I applied this by using the before(new Date()) method to validate the dates, as shown above. I followed this same structured approach in my service and test files, providing coverage for each requirement. My tests validated both successful operations and error handling for invalid inputs, ensuring all requirements were met effectively.

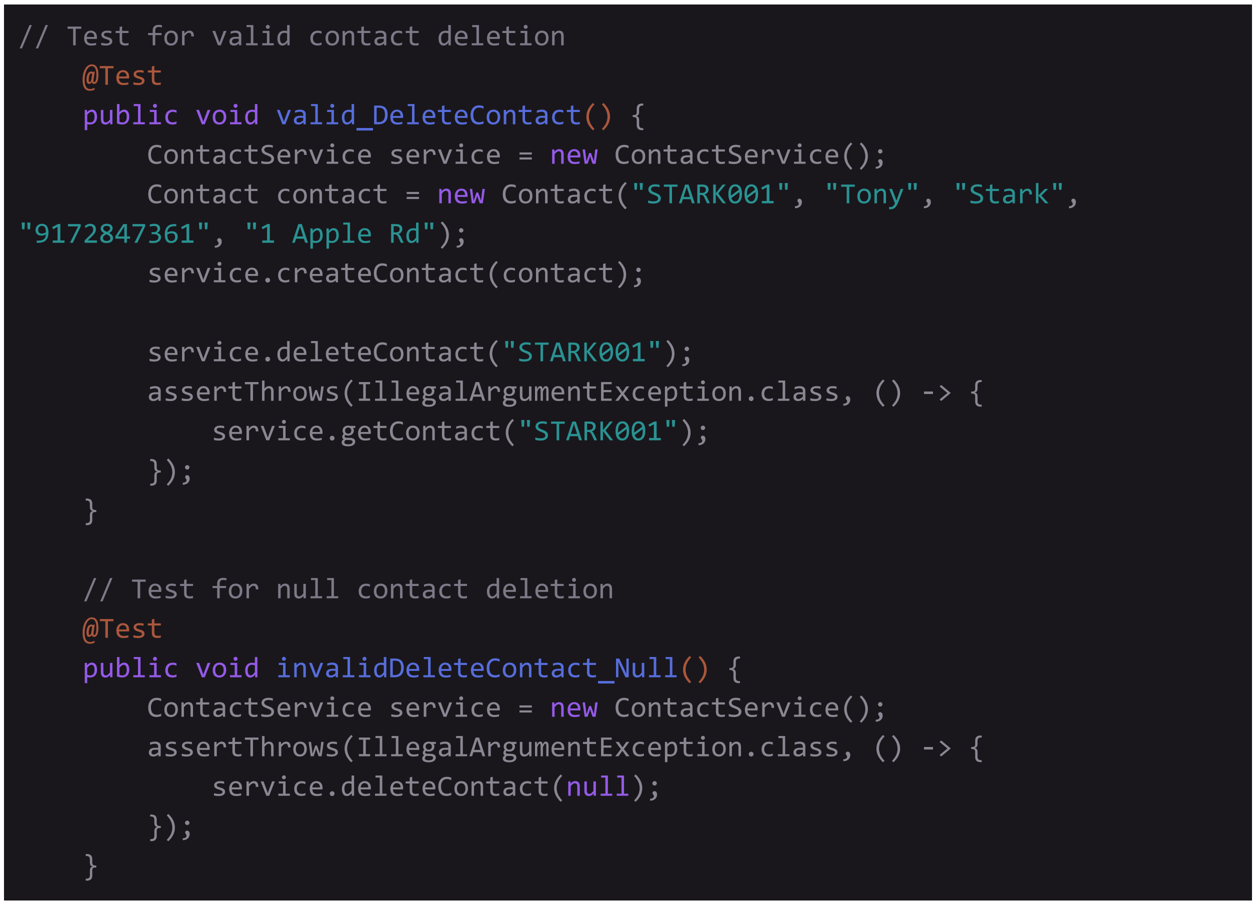
b. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

For the JUnit tests I ran, the coverage percentages across all files were successful, with most classes achieving coverage rates between 84% and 100%. Since the requirement was to maintain coverage above 80%, these results demonstrate that my tests effectively validated functionality throughout the program. Additionally, some classes such as Contact.java and AppointmentService.java achieved 100% coverage, showing that the tests were not only thorough but also highly effective in ensuring the overall functionality of my program.

B. Describe your experience writing the JUnit tests.

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

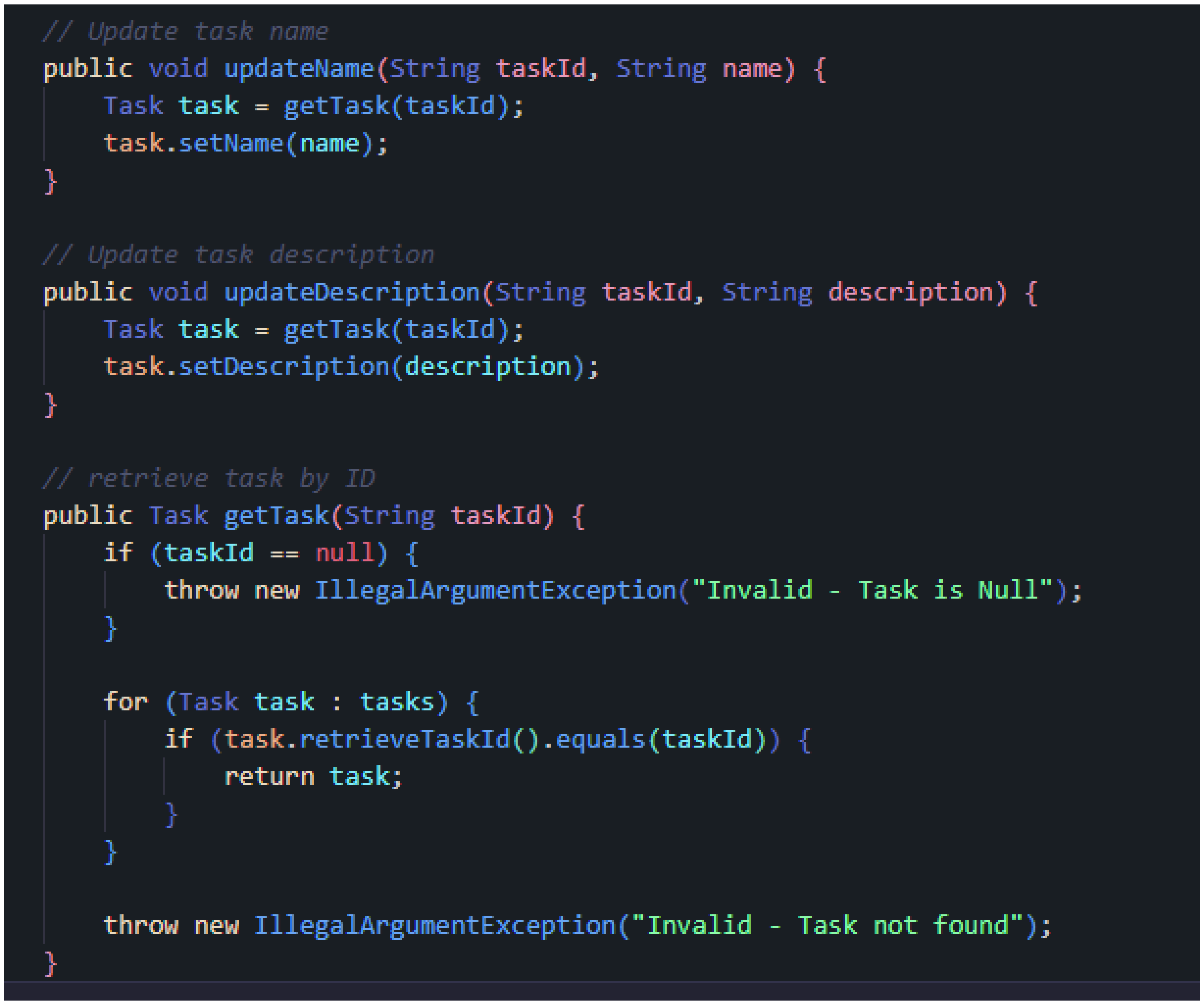
I ensured technical soundness throughout the program by testing system behavior with both valid and invalid inputs, incorporating assertions to verify proper functionality and effective error handling. For example, in ContactServiceTest.java, I ran JUnit tests for both valid and invalid contact deletions, one verifying successful deletion and the other confirming that a null parameter triggers an exception. Both tests performed as expected.



```
// Test for valid contact deletion
    @Test
    public void valid_DeleteContact() {
        ContactService service = new ContactService();
        Contact contact = new Contact("STARK001", "Tony", "Stark",
"9172847361", "1 Apple Rd");
        service.createContact(contact);

        service.deleteContact("STARK001");
        assertThrows(IllegalArgumentException.class, () -> {
            service.getContact("STARK001");
        });
    }

    // Test for null contact deletion
    @Test
    public void invalidDeleteContact_Null() {
        ContactService service = new ContactService();
        assertThrows(IllegalArgumentException.class, () -> {
            service.deleteContact(null);
        });
    }
```

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

I ensured code efficiency by prioritizing consistency in structure and reusing existing methods to avoid redundancies. For example, in TaskService.java, I centralized task retrieval through the getTask() method, allowing new functions such as updateName() and updateDescription() to reuse this method instead of creating completely new and separate implementations that will serve the same purpose. By applying this same structure and formatting across the Task, Contact, and Address class files, it became much easier to identify and mitigate errors, as well as add new functions and updates efficiently.



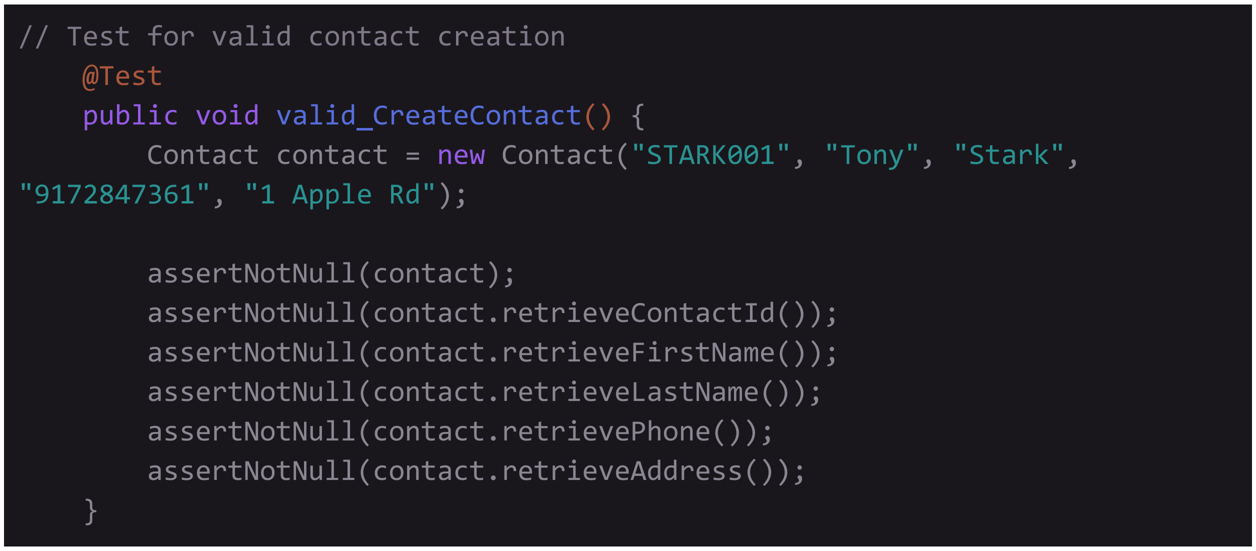
```
// Update task name
public void updateName(String taskId, String name) {
    Task task = getTask(taskId);
    task.setName(name);
}

// Update task description
public void updateDescription(String taskId, String description) {
    Task task = getTask(taskId);
    task.setDescription(description);
}

// retrieve task by ID
public Task getTask(String taskId) {
    if (taskId == null) {
        throw new IllegalArgumentException("Invalid - Task is Null");
    }

    for (Task task : tasks) {
        if (task.retrieveTaskId().equals(taskId)) {
            return task;
        }
    }

    throw new IllegalArgumentException("Invalid - Task not found");
}
```

Reflection

A. Testing Techniques

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

Throughout the project, I employed both static and dynamic testing techniques. Static testing was conducted through manual code reviews, where I examined my code before execution to identify unused variables, typos, and syntax errors. I also utilized my IDE's built-in assistance, which helped detect many of these issues. For dynamic testing, this was primarily done through the required JUnit tests, running invalid and valid parameters to test each function individually and as a whole. For each class test file, I began each one by ensuring it ran properly with normal input. For example, I ran the following JUnit test for ContactServiceTest.Java:



```
// Test for valid contact creation
    @Test
    public void valid_CreateContact() {
        Contact contact = new Contact("STARK001", "Tony", "Stark",
"9172847361", "1 Apple Rd");

        assertNotNull(contact);
        assertNotNull(contact.retrieveContactId());
        assertNotNull(contact.retrieveFirstName());
        assertNotNull(contact.retrieveLastName());
        assertNotNull(contact.retrievePhone());
        assertNotNull(contact.retrieveAddress());
    }
```

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

When considering software testing techniques I did not utilize within this project, I did not implement regression testing, which would have helped ensure that when new service functions are added or existing code is modified, no unexpected issues arise in previously working functionalities. This testing would be beneficial to the product during the later stages in the software development life cycle as it consists of re-running the test files after an update within the code base to check for performance impacts and bugs on pre-existing features (GeeksforGeeks, 2025).

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

Each of the techniques I applied, including unit testing, error testing, validation testing, and static testing, has unique practical uses and implications. Unit testing is essential for verifying individual components, ensuring they perform as intended. Error and validation testing work together to confirm that the system can handle invalid input appropriately while processing valid input correctly. Static testing through manual code review helps catch design flaws and syntax errors prior to code execution, improving quality and efficiency. In the end, these testing techniques not only ensured my code met the project requirements but also provided valuable insight into improving my work process for my future software projects.

B. Mindset

a. 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.

While working on this project, I maintained a mindset of thinking about what works best for the user and what is easiest for the development team as a whole in terms of program maintenance. I focused on ensuring my function and variable names remained consistently clear and descriptive to make the code intuitive for both users of the program and the developers maintaining it. Additionally, I implemented consistent validation patterns across all classes and consistent and easy to understand error messages such as "Invalid - Contact not found" to help users understand errors easily while making debugging simpler for the development team. It was important to appreciate the complexity and interrelationships of the code I was testing because the service classes depend on the object classes being constructed and integrated correctly to function properly. For example, if my Contact.java file did not correctly implement the setLastName() method with proper validation, the updateLastName() method in ContactService.java would be useless and would likely allow invalid parameters.

b. 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.

In order to limit bias while reviewing my code, I thought through and tested my code base from every perspective other than my own, as if someone completely new to the project was viewing it and asking questions such as "is this digestible," "is there a way to simplify this further," and "will the user understand this." For example, when developing the createContact() method in ContactService.java, I initially considered using nested loops to check for duplicates, but by thinking about it from a maintenance perspective, reusing the existing getContact() method was more readable and maintainable.

c. 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.

When writing code, it is extremely important to remain committed to ensuring the quality of the program is not only efficient and secure but also consistent throughout. When corners are cut, that leaves gaps in user experience and, more importantly, vulnerabilities in security that can be exploited. In addition to these gaps, inconsistent code quality creates long-term maintenance issues where bugs become increasingly difficult to trace and fix. For example, if I had failed to apply proper null checks in the setter methods of Contact.java, it would create cascading errors throughout ContactService.java where methods like updateFirstName() would process as if it is using valid data, likely leading to incomplete process handling and security risks. To avoid technical debt as a practitioner in the field, it’s important to have a set plan and list of requirements prior to writing the code. This will make it easier to figure out how the code should be structured with quality and security in mind.

Citations

GeeksforGeeks. (2025, July 23). What is regression testing in Agile?

https://www.geeksforgeeks.org/software-testing/what-is-regression-testing-in-agile/