TDD Example for Login System

Testing and Integrate Manager: Zhihao Ma

I provide a basic structure for the `LoginSystem` class and its corresponding test class `LoginSystemTest`.

1. The `LoginSystem` Class

This class will handle the login logic. For simplicity, I'll hardcode a valid username and password.

```
public class LoginSystem {

    private static final String VALID_USERNAME = "admin";

    private static final String VALID_PASSWORD = "password";



    public String authenticate(String username, String password) {

        if (VALID_USERNAME.equals(username)
&&VALID_PASSWORD.equals(password)) {

            return "Login successful";

        } else {

            return "Login failed";

        }

    }

}
```

2. The `LoginSystemTest` Class

This class will contain tests for the `LoginSystem`. I'll use JUnit for testing.

```
import static org.junit.Assert.*;

import org.junit.Test;


public class LoginSystemTest {


    @Test

    public void testSuccessfulLogin() {

        LoginSystem loginSystem = new LoginSystem();

        String result = loginSystem.authenticate("admin", "password123");

        assertEquals("Login successful", result);

    }


    @Test

    public void testFailedLogin() {

        LoginSystem loginSystem = new LoginSystem();

        String result = loginSystem.authenticate("user", "wrongPassword");

        assertEquals("Login failed", result);

    }

}
```

3. Considerations

- **Security:** This example is not secure. In a real application, we should never store passwords in plain text, and should use proper authentication mechanisms.
- **Database Interaction:** Typically, a login system interacts with a database to verify user credentials. This would involve more complex logic and possibly the use of frameworks like JDBC, Hibernate, or Spring Data.
- **User Interface:** This example doesn't include a user interface. In a real application, we would have a UI that interacts with the `LoginSystem`.
- **Testing Practices:** For more comprehensive testing, consider edge cases, null inputs, and other scenarios. In a real-world scenario, mocking frameworks like Mockito might be used to mock database interactions or other dependencies.