

SIT 333

Software Quality and Testing

Task 4.1P

Name: Aditya
Student I'D: 224277942

Introduction to Software Testing Strategies

- In software testing, two major approaches help ensure systems behave as expected:
- Decision Table-Based Testing and Path Testing.
- These methods aim to identify bugs and edge cases before software reaches users.
- This presentation explores both techniques, their real-world applications, and how they improve software quality.
- The goal is to reflect on how these testing strategies simplify and improve the development lifecycle.

What is Decision Table-Based Testing?

- Decision table-based testing is a black-box testing technique.
- It represents input conditions and expected actions as a structured table, helping visualize logic clearly.
- Often called a cause-effect table because it shows how different causes (inputs) lead to specific effects (outputs).
- Very useful when systems need to react differently based on multiple conditions or scenarios.
- Example use cases include form validation (e.g., checking if an email and password are entered correctly).

Advantages and Disadvantages of Decision Table Testing



Advantages:

- Offers a clear and organized way to model decision logic.
- Ensures that all combinations of conditions are considered, preventing logic gaps.
- Particularly useful in requirement-based testing or systems with rules and validations.

Disadvantages:

- As the number of input variables increases, the table grows exponentially, becoming complex and time-consuming.
- Not suitable for every type of testing, especially where program flow or internal logic is more important than outcomes.

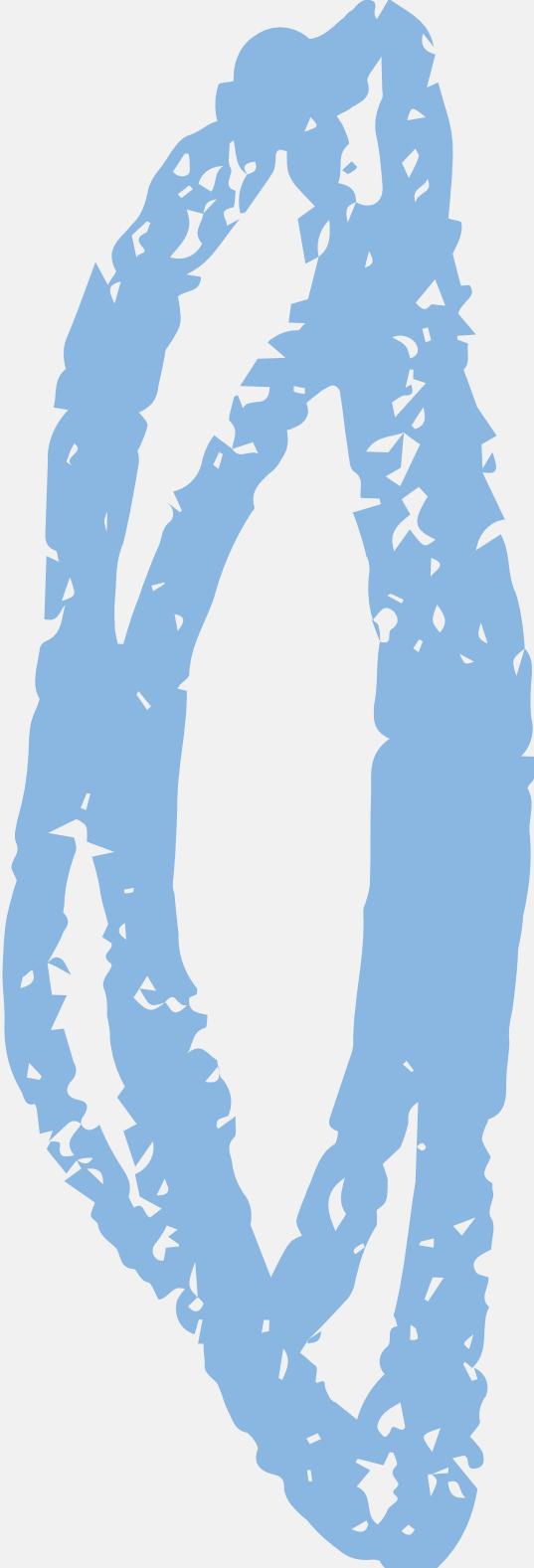


What is Path Testing?

- Path testing is a white-box testing technique that analyzes a program's control flow.
- The program is converted into a flow graph with:
 - Nodes: Representing actions or statements.
 - Edges: Representing transitions or decision points.
- The objective is to execute every possible path through the program at least once.
- This helps identify unreachable code, logic errors, and flaws in loop execution

Cyclomatic Complexity and Basis Path Testing

- Introduced by McCabe, cyclomatic complexity is a metric used to count independent paths through a program.
- Formula:
- $V(G) = E - N + 2P$
- where E = number of edges, N = nodes, P = connected components.
- From this, a set of basis paths is derived.
- Writing test cases to cover each basis path ensures maximum coverage with minimum effort.
- Commonly used in safety-critical and embedded systems.



Advantages and Disadvantages of Path Testing

Advantages:

- Guarantees every possible execution route is examined.
- Helps in identifying unreachable or redundant code.
- Provides a strong foundation for high-coverage test suites.

Disadvantages:

- As code complexity increases, the number of paths can become enormous (especially with loops).
- Not practical for large-scale software without tools.
- Requires detailed knowledge of the internal code structure, making it unsuitable for black-box testers.

Real-World Applications

Decision Table Testing Applications:

- Web login forms with multiple validations.
- Insurance or loan eligibility systems.
- Print diagnostics or troubleshooting apps.

Path Testing Applications:

- Date calculators (e.g., Next-Date problem).
- Financial transaction systems.
- Control systems in robotics or embedded devices.

Each technique shines in different scenarios—decision tables when testing rules, and path testing when ensuring logic coverage.



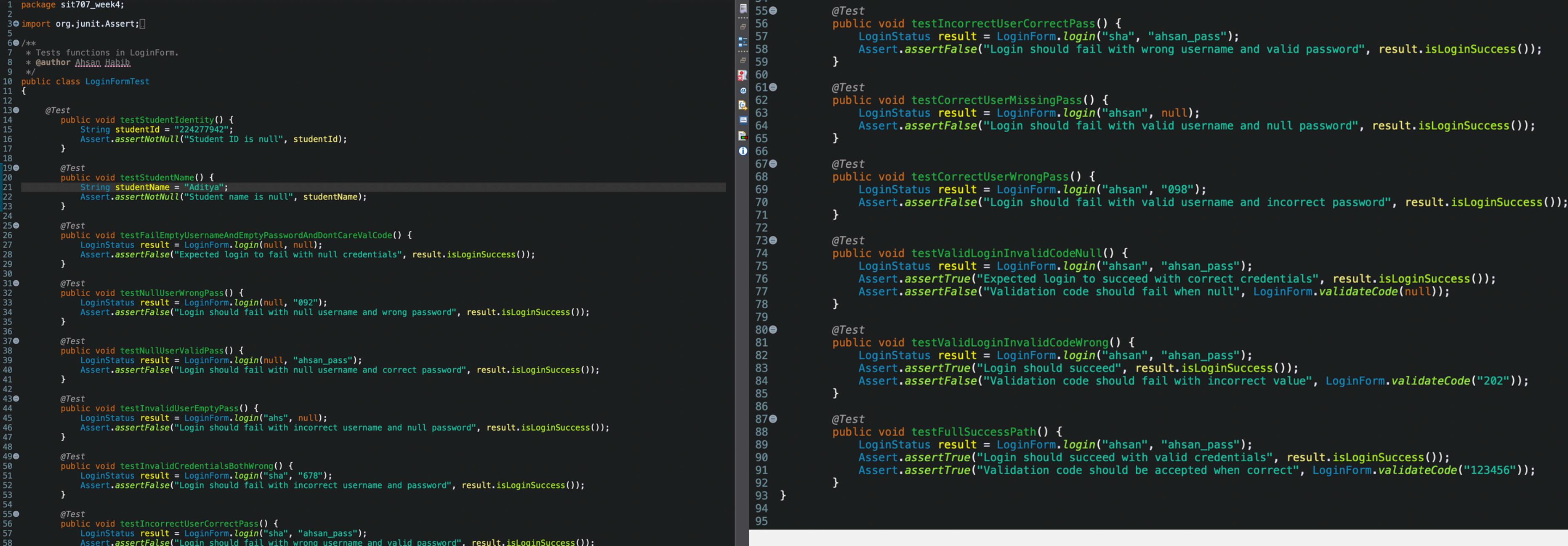
Summary of Key Insights

- Decision Table Testing is excellent for capturing all possible input conditions in systems with multiple rules.
- Path Testing dives deep into program structure, ensuring thorough inspection of logic and branches.
- Both techniques complement each other—functional correctness + structural soundness.
- Their combination leads to more reliable, testable, and maintainable software.

Personal Reflection and Learning

- This week's study deepened my understanding of how testing logic and paths strengthens software.
- I realized how important it is to choose the right strategy depending on the context and code complexity.
- Through examples like email validation and triangle classifiers, I saw how clear thinking and structure help test smarter.
- I now feel more confident in designing test cases and using tools like JUnit for automated validation.

Source Code:



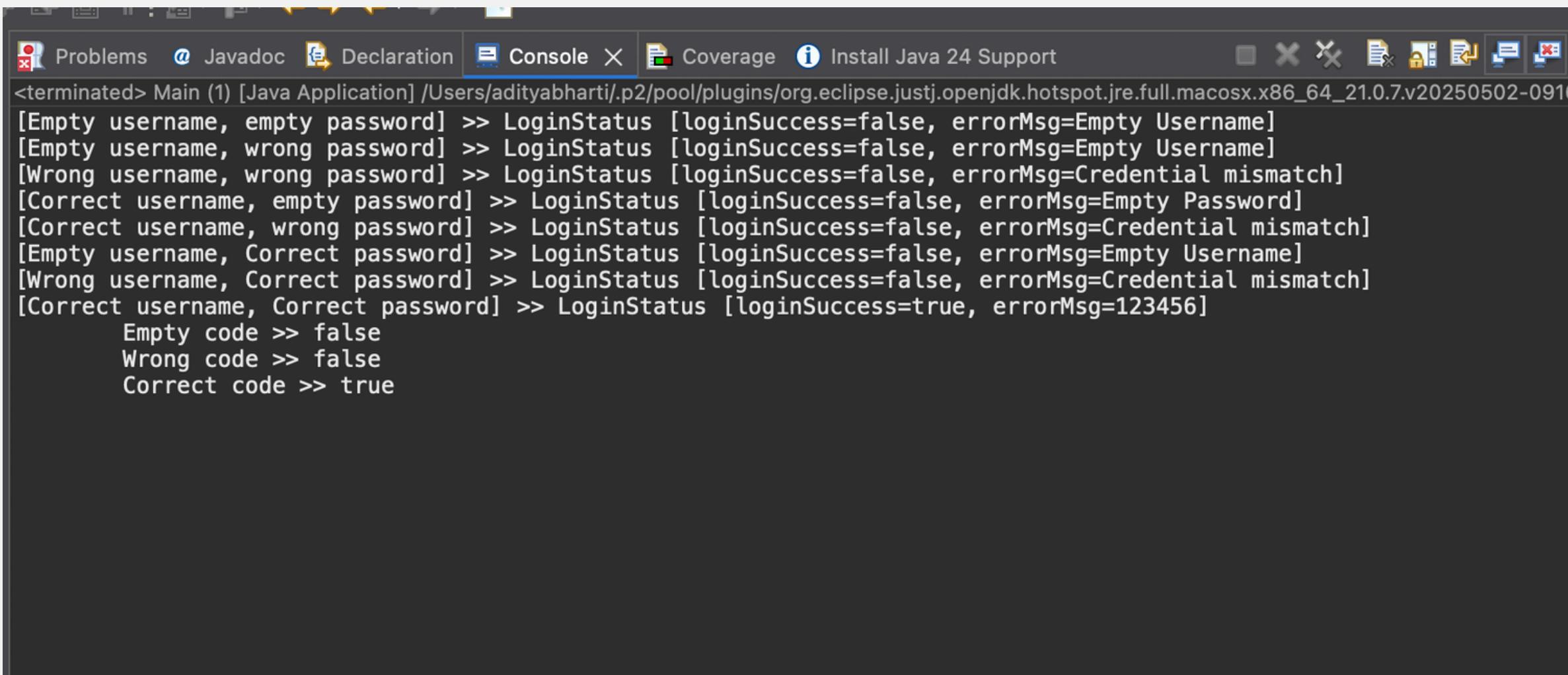
```
1 package sit707_week4;
2
3 import org.junit.Assert;
4
5 /**
6  * Tests functions in LoginForm.
7  * @author Ahsan Habib
8 */
9
10 public class LoginFormTest {
11
12     @Test
13     public void testStudentIdentity() {
14         String studentId = "224277942";
15         Assert.assertNotNull("Student ID is null", studentId);
16     }
17
18     @Test
19     public void testStudentName() {
20         String studentName = "Aditya";
21         Assert.assertNotNull("Student name is null", studentName);
22     }
23
24     @Test
25     public void testFailEmptyUsernameAndEmptyPasswordAndDontCareValCode() {
26         LoginStatus result = LoginForm.login(null, null);
27         Assert.assertFalse("Expected login to fail with null credentials", result.isLoginSuccess());
28     }
29
30     @Test
31     public void testNullUserWrongPass() {
32         LoginStatus result = LoginForm.login(null, "092");
33         Assert.assertFalse("Login should fail with null username and wrong password", result.isLoginSuccess());
34     }
35
36     @Test
37     public void testNullUserValidPass() {
38         LoginStatus result = LoginForm.login(null, "ahsan_pass");
39         Assert.assertFalse("Login should fail with null username and correct password", result.isLoginSuccess());
40     }
41
42     @Test
43     public void testInvalidUserEmptyPass() {
44         LoginStatus result = LoginForm.login("ahs", null);
45         Assert.assertFalse("Login should fail with incorrect username and null password", result.isLoginSuccess());
46     }
47
48     @Test
49     public void testInvalidCredentialsBothWrong() {
50         LoginStatus result = LoginForm.login("sha", "678");
51         Assert.assertFalse("Login should fail with incorrect username and password", result.isLoginSuccess());
52     }
53
54     @Test
55     public void testIncorrectUserCorrectPass() {
56         LoginStatus result = LoginForm.login("sha", "ahsan_pass");
57         Assert.assertFalse("Login should fail with wrong username and valid password", result.isLoginSuccess());
58     }
59 }
```

```
55     @Test
56     public void testIncorrectUserCorrectPass() {
57         LoginStatus result = LoginForm.login("sha", "ahsan_pass");
58         Assert.assertFalse("Login should fail with wrong username and valid password", result.isLoginSuccess());
59     }
60
61     @Test
62     public void testCorrectUserMissingPass() {
63         LoginStatus result = LoginForm.login("ahsan", null);
64         Assert.assertFalse("Login should fail with valid username and null password", result.isLoginSuccess());
65     }
66
67     @Test
68     public void testCorrectUserWrongPass() {
69         LoginStatus result = LoginForm.login("ahsan", "098");
70         Assert.assertFalse("Login should fail with valid username and incorrect password", result.isLoginSuccess());
71     }
72
73     @Test
74     public void testValidLoginInvalidCodeNull() {
75         LoginStatus result = LoginForm.login("ahsan", "ahsan_pass");
76         Assert.assertTrue("Expected login to succeed with correct credentials", result.isLoginSuccess());
77         Assert.assertFalse("Validation code should fail when null", LoginForm.validateCode(null));
78     }
79
80     @Test
81     public void testValidLoginInvalidCodeWrong() {
82         LoginStatus result = LoginForm.login("ahsan", "ahsan_pass");
83         Assert.assertTrue("Login should succeed", result.isLoginSuccess());
84         Assert.assertFalse("Validation code should fail with incorrect value", LoginForm.validateCode("202"));
85     }
86
87     @Test
88     public void testFullSuccessPath() {
89         LoginStatus result = LoginForm.login("ahsan", "ahsan_pass");
90         Assert.assertTrue("Login should succeed with valid credentials", result.isLoginSuccess());
91         Assert.assertTrue("Validation code should be accepted when correct", LoginForm.validateCode("123456"));
92     }
93 }
94
95 }
```

Test Case Output:

```
Finished after 0.035 seconds
Runs: 13/13    ✘ Errors: 0    ✘ Failures: 0
sit707_week4.LoginFormTest [Runner: JUnit 4] (0.000 s)
  ✓ testStudentIdentity (0.000 s)
  ✓ testCorrectUserMissingPass (0.000 s)
  ✓ testInvalidUserEmptyPass (0.000 s)
  ✓ testNullUserValidPass (0.000 s)
  ✓ testInvalidCredentialsBothWrong (0.000 s)
  ✓ testValidLoginInvalidCodeNull (0.000 s)
  ✓ testIncorrectUserCorrectPass (0.000 s)
  ✓ testNullUserWrongPass (0.000 s)
  ✓ testFullSuccessPath (0.000 s)
  ✓ testValidLoginInvalidCodeWrong (0.000 s)
  ✓ testStudentName (0.000 s)
  ✓ testFailEmptyUsernameAndEmptyPasswordAndDontCareValue (0.000 s)
  ✓ testCorrectUserWrongPass (0.000 s)
```

Output:



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The console output displays a series of test cases for a login status check. The output is as follows:

```
<terminated> Main (1) [Java Application] /Users/adityabharti/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_21.0.7.v20250502-0916
[Empty username, empty password] >> LoginStatus [loginSuccess=false, errorMsg=Empty Username]
[Empty username, wrong password] >> LoginStatus [loginSuccess=false, errorMsg=Empty Username]
[Wrong username, wrong password] >> LoginStatus [loginSuccess=false, errorMsg=Credential mismatch]
[Correct username, empty password] >> LoginStatus [loginSuccess=false, errorMsg=Empty Password]
[Correct username, wrong password] >> LoginStatus [loginSuccess=false, errorMsg=Credential mismatch]
[Empty username, Correct password] >> LoginStatus [loginSuccess=false, errorMsg=Empty Username]
[Wrong username, Correct password] >> LoginStatus [loginSuccess=false, errorMsg=Credential mismatch]
[Correct username, Correct password] >> LoginStatus [loginSuccess=true, errorMsg=123456]
    Empty code >> false
    Wrong code >> false
    Correct code >> true
```

Github:

<https://github.com/Aditya2Verma/SIT333>

The screenshot shows the GitHub repository page for the user Aditya2Verma with the repository name SIT333. The repository is public and has 1 branch and 0 tags. The main branch, 'main', was updated 1 minute ago with 10 commits by Aditya2Verma. The commits listed are:

- 2.1P/task2_1P: Add files via upload (last week)
- 7.1P: Add files via upload (yesterday)
- springws: Delete springws/src/main/java/web/service/LoginService.j... (yesterday)
- task1_2P: Add files via upload (last week)
- task3_1P 2: Update DateUtilTest.java (42 minutes ago)
- task4_1P: Add files via upload (5 days ago)
- task6_1P: Add files via upload (5 days ago)

The repository has 1 watching and 0 forks. It has no releases or packages published. The Languages section shows Java at 96.4% and HTML at 3.6%. A 'README' file is present, but there is no content.