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.

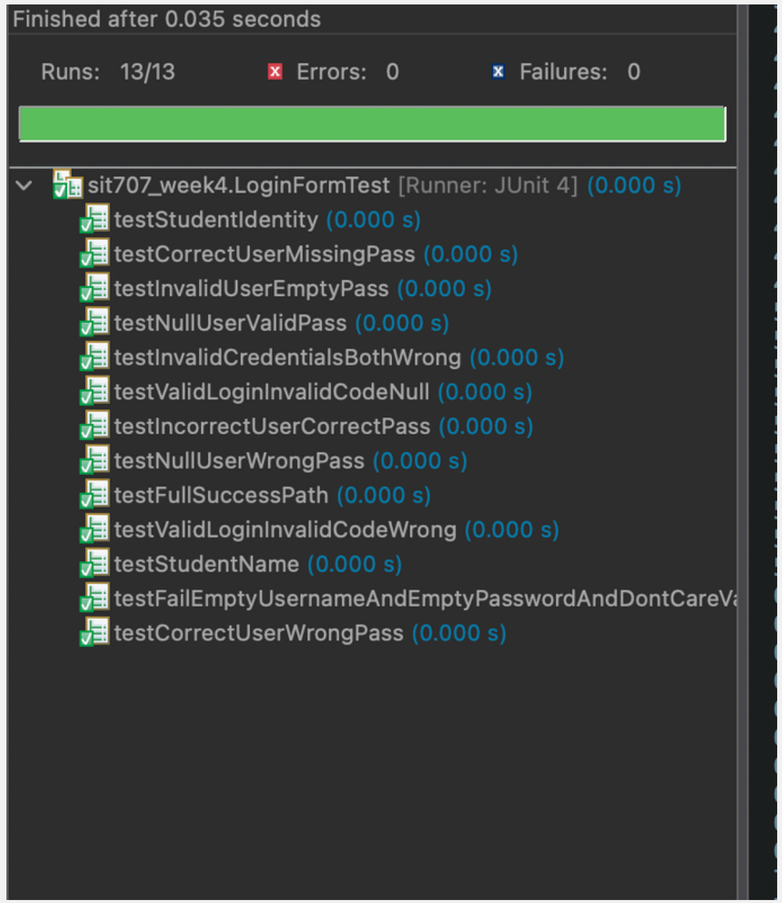
Personal Reflection and Learning

- 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.

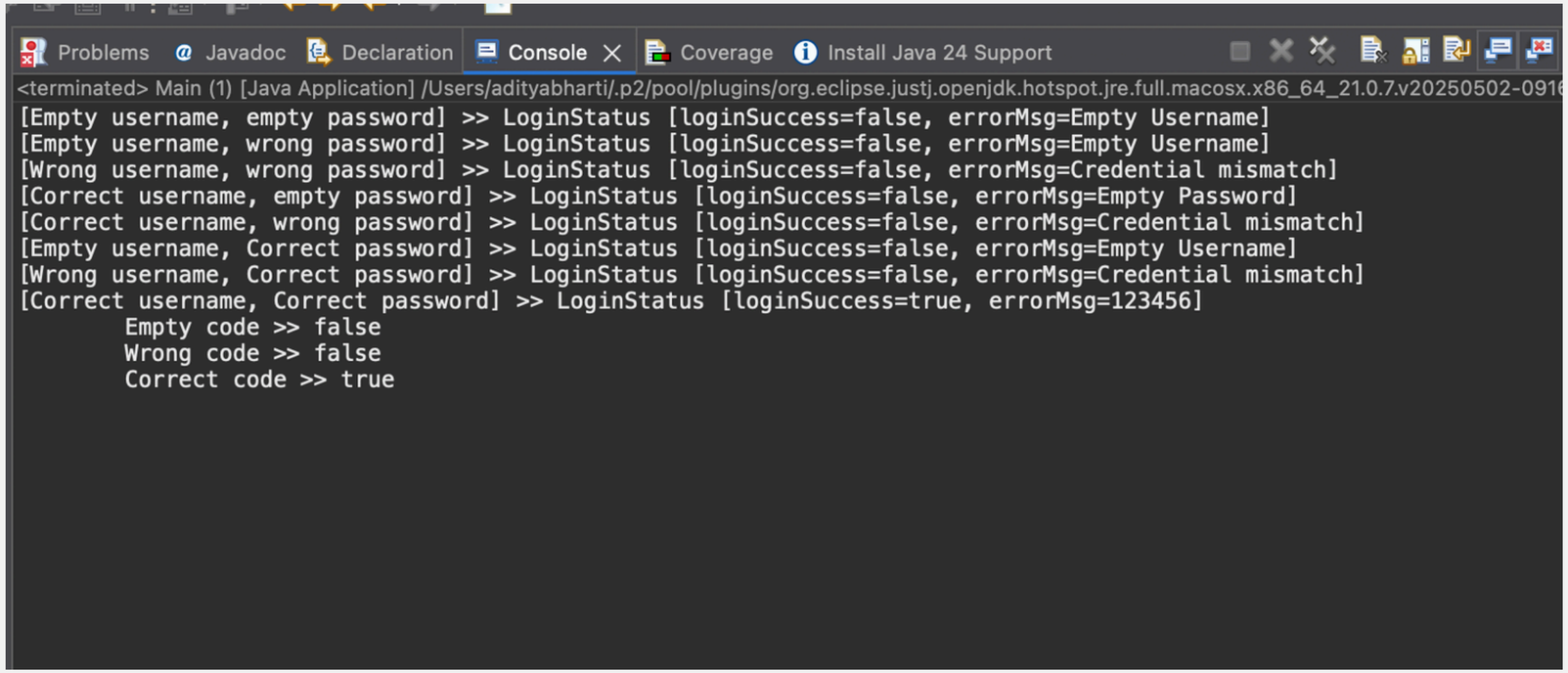
Source Code:

```java
package sit707_week4;

import org.junit.Assert;

/**
 * Tests functions in LoginForm.
 * @author Ahsan Habib
 */
public class LoginFormTest
{

    @Test
        public void testStudentIdentity() {
            String studentId = "224277942";
            Assert.assertNotNull("Student ID is null", studentId);
        }

        @Test
        public void testStudentName() {
            String studentName = "Aditya";
            Assert.assertNotNull("Student name is null", studentName);
        }

        @Test
        public void testFailEmptyUsernameAndEmptyPasswordAndDontCareValCode() {
            LoginStatus result = LoginForm.login(null, null);
            Assert.assertFalse("Expected login to fail with null credentials", result.isLoginSuccess());
        }

        @Test
        public void testNullUserWrongPass() {
            LoginStatus result = LoginForm.login(null, "092");
            Assert.assertFalse("Login should fail with null username and wrong password", result.isLoginSuccess());
        }

        @Test
        public void testNullUserValidPass() {
            LoginStatus result = LoginForm.login(null, "ahsan_pass");
            Assert.assertFalse("Login should fail with null username and correct password", result.isLoginSuccess());
        }

        @Test
        public void testInvalidUserEmptyPass() {
            LoginStatus result = LoginForm.login("ahs", null);
            Assert.assertFalse("Login should fail with incorrect username and null password", result.isLoginSuccess());
        }

        @Test
        public void testInvalidCredentialsBothWrong() {
            LoginStatus result = LoginForm.login("sha", "678");
            Assert.assertFalse("Login should fail with incorrect username and password", result.isLoginSuccess());
        }

        @Test
        public void testIncorrectUserCorrectPass() {
            LoginStatus result = LoginForm.login("sha", "ahsan_pass");
            Assert.assertFalse("Login should fail with wrong username and valid password", result.isLoginSuccess());
        }

        @Test
        public void testIncorrectUserCorrectPass() {
            LoginStatus result = LoginForm.login("sha", "ahsan_pass");
            Assert.assertFalse("Login should fail with wrong username and valid password", result.isLoginSuccess());
        }

        @Test
        public void testCorrectUserMissingPass() {
            LoginStatus result = LoginForm.login("ahsan", null);
            Assert.assertFalse("Login should fail with valid username and null password", result.isLoginSuccess());
        }

        @Test
        public void testCorrectUserWrongPass() {
            LoginStatus result = LoginForm.login("ahsan", "098");
            Assert.assertFalse("Login should fail with valid username and incorrect password", result.isLoginSuccess());
        }

        @Test
        public void testValidLoginInvalidCodeNull() {
            LoginStatus result = LoginForm.login("ahsan", "ahsan_pass");
            Assert.assertTrue("Expected login to succeed with correct credentials", result.isLoginSuccess());
            Assert.assertFalse("Validation code should fail when null", LoginForm.validateCode(null));
        }

        @Test
        public void testValidLoginInvalidCodeWrong() {
            LoginStatus result = LoginForm.login("ahsan", "ahsan_pass");
            Assert.assertTrue("Login should succeed", result.isLoginSuccess());
            Assert.assertFalse("Validation code should fail with incorrect value", LoginForm.validateCode("202"));
        }

        @Test
        public void testFullSuccessPath() {
            LoginStatus result = LoginForm.login("ahsan", "ahsan_pass");
            Assert.assertTrue("Login should succeed with valid credentials", result.isLoginSuccess());
            Assert.assertTrue("Validation code should be accepted when correct", LoginForm.validateCode("123456"));
        }
}
```

Test Case Output:



Output:


Problems
Javadoc
Declaration
Console
Coverage
Install Java 24 Support
<terminated> Main (1) [Java Application] /Users/adityabharti/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_21.0.7.v20250502-091
[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