Decision table and JUnit

Presented by

Sireesha Akurathi

WHAT IS DECISION TABLE TESTING?

- Definition: A black-box test design technique used for functions with multiple input combinations.
- Used to represent rules and actions in tabular form.
- Best for testing systems with business rules and logic conditions.

EXAMPLE OF DECISION TABLE

DateUtilTest.java | decisiontable/pom.xml | SimpleLoginForm.java | SimpleLoginFormTest.java

```java
package edu.deakin;

public class SimpleLoginForm {
    private static final String VALID_USERNAME = "admin";
    private static final String VALID_PASSWORD = "password123";

    public String login(String username, String password) {
        if (username == null || password == null) {
            return "error";
        }

        if (username.equals(VALID_USERNAME) && password.equals(VALID_PASSWORD)) {
            return "success";
        } else {
            return "fail";
        }
    }
}
```

- Helps identify all input condition combinations
- Ensures comprehensive test coverage
- Easy to translate into JUnit test cases

Test Case	Username	Password	Username is Null?	Password is Null?	Username is Empty?	Password is Empty?	Expected Result
TC1	"admin"	"password123"	No	No	No	No	success
TC2	"admin"	"wrongpass"	No	No	No	No	fail
TC3	"wronguser"	"password123"	No	No	No	No	fail
TC4	"wronguser"	"wrongpass"	No	No	No	No	fail
TC5	"admin"	null	No	Yes	-	-	error
TC6	null	"password123"	Yes	No	-	-	error
TC7	null	null	Yes	Yes	-	-	error
TC8	"" (empty)	"" (empty)	No	No	Yes	Yes	fail

REAL-WORLD USE CASE – DECISION TABLE TESTING

Other Real-World Examples

- Credit card validation rules
- Loan approval systems
- Tax calculation based on income levels
- Form validations with multiple dependencies

Use Case Example: Login Authentication System

System Logic:

- Check username and password
- Return success, fail, or error

Why Decision Table?

- Multiple input combinations (e.g., null, empty, wrong, correct)
- Clearly maps conditions to outcomes
- Avoids missing edge cases during testing

WHAT IS PATH TESTING?

Path Testing is a white-box testing technique that focuses on validating all possible execution paths in a program's control flow.

Key Concepts

Testers analyze the Control Flow Graph (CFG) of the code

Goal: Cover all independent paths, branches, and loops

Based on the structure of the code, not just input/output

Why Use Path Testing?

- Ensures high code coverage
- Detects:
- Unreachable code
- Infinite loops
- Missed logical conditions

REAL-WORLD USE CASE – PATH TESTING

Use Case Example: Online Payment Processing
In a payment module:

- Check user session
- Validate card details
- Process transaction
- Confirm response
- Many decision points and branches involved

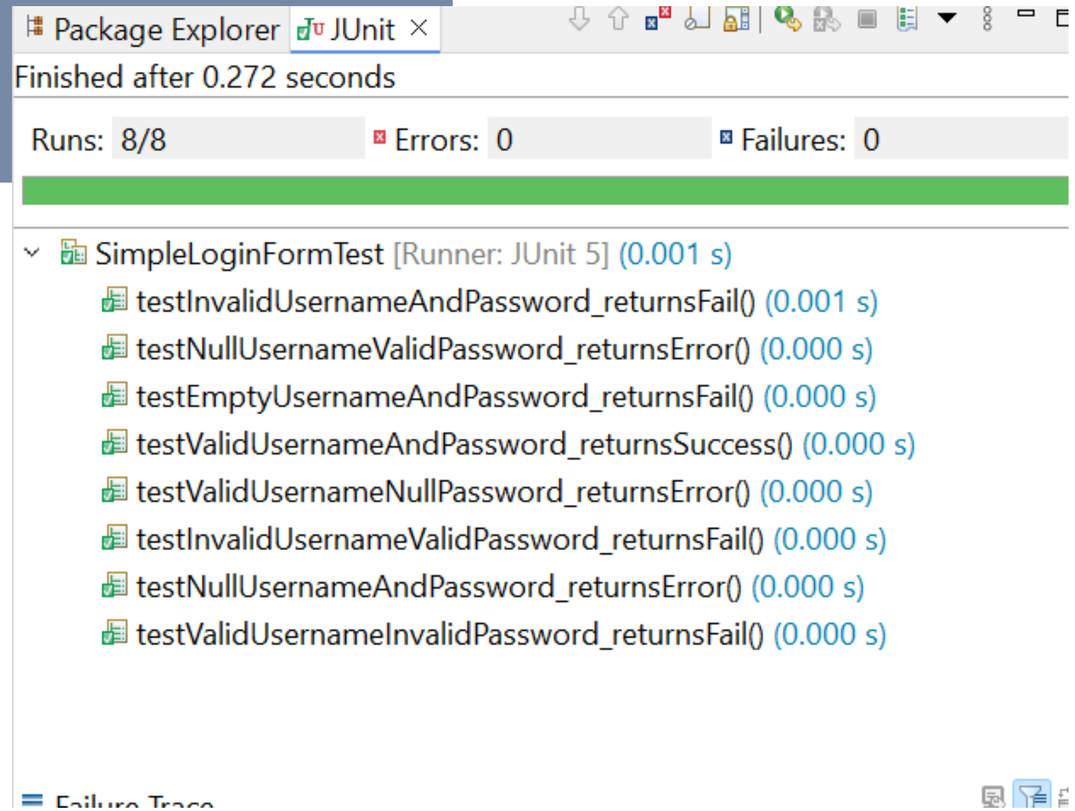
Why Path Testing is Useful

- Ensures all paths are tested:
- Valid & invalid card
- Session expired
- Network failure
- Helps uncover logic errors, unreachable code, or infinite loops

Real-World Applications

- 💳 Banking systems (fund transfer, fraud checks)
- 🚗 Navigation systems (multiple route decisions)
- 📱 Mobile apps with state transitions
- 🧪 Scientific simulations with loops and calculations

ACTIVE LEARNING SESSION EVIDENCE



During the active learning session:

- We reviewed the SimpleLoginForm example to apply Decision Table Testing
- Collaboratively created a decision table identifying:
- Valid/invalid/null input combinations
- Expected outcomes: success, fail, error
- Converted decision rules into JUnit test cases

THANK YOU