## Retrofitting UI Tests (QLC-4) — Jetpack Compose & TDD

### Objective:

Understand TDD principles by retrofitting UI tests into an existing Compose screen (MoodSelectorScreen), using techniques like truth tables and dependency injection to control state.

### Scenario:

Your team inherited a Compose screen with some basic, outdated UI tests that only cover the most obvious interactions. Those tests rely on the real MoodRepository, making results non‑deterministic and brittle. To improve confidence and cover missing edge cases, you’ll:

1. Replace the real repo with a **fake repository** that returns a predictable list of moods.
2. Use a **truth table** to identify non‑obvious scenarios (pre‑selection, empty repo, last‑mood selection).
3. Drive the red‑green‑refactor cycle to write robust, focused UI tests against those scenarios.

#### Step 1 — Analyse with a Truth Table(Red Phase Begins)

1. Identify key variables and states:  
   * availableMoods (from fake repo)
   * selectedMoods list
   * buttonEnabled per mood
2. Draw a truth table covering:  
   * No moods selected
   * One mood selected
   * Multiple moods selected
   * All moods selected
3. Pick at least three non-obvious scenarios to test, **Non‑obvious scenarios to test example :**

From that table, we’ll focus on these three less‑obvious cases:

1. **Initial pre‑selection**
   * Start the screen with selectedMoods = [B] injected.
   * Verify B’s button is disabled and “Selected Moods” shows B on launch.

*Deliverable*: a filled-in truth table (hand-drawn or digital).

| State | availableMoods | selectedMoods | buttonEnabled |
| --- | --- | --- | --- |
| No moods selected |  |  |  |
| One mood selected |  |  |  |
| Multiple moods selected |  |  |  |
| All moods selected |  |  |  |

#### 

#### Step 2 — Write a Failing UI Test (Red Phase)

1. Create a test class
2. Write a test for the truth‑table scenario “select one mood disables its button and adds it to the list.”
3. Run it and confirm it fails because you haven’t added any testTags yet.

#### Step 3 — Make the Test Pass (Green Phase)

1. Add Modifier.testTag("button-$mood") on each Button and Modifier.testTag("selected-moods-label") on the header:
2. Update your test to use onNodeWithTag(...) instead of text lookups.
3. Re‑run and watch that first test now goes green.

#### Step 4 — Refactor

* Extract common setup (e.g. a launchScreen() helper that calls setContent { … }).
* Rename your test methods to clearly convey behavior (e.g. selectingMood\_disablesButton\_andDisplaysInList).
* Look at the UI code: is there duplication or complexity you can clean up now that you’ve been through the cycle?

#### Step 5 — Cover Remaining Cases

* Add a test of the **initial state** (all buttons enabled, no selections).
* Add tests for **multiple selections** and **selecting all moods**.
* Add a test for **double‑click** (no duplicates).

*Extra goal:* If you ever parameterize the VM to accept its mood list, you could also test “empty list” or “single‑item” scenarios.

#### Step 6 — Reflect

* Which edge cases would have been easy to miss without our truth‑table?
* How did tagging UI nodes make tests more reliable?
* If you were building this screen from scratch, what design changes would simplify testing further?