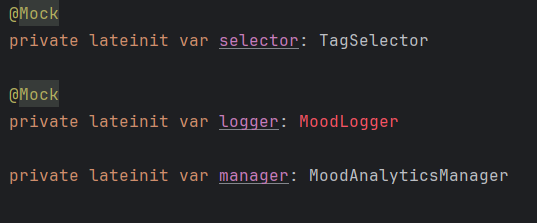
## QLC‑3: Logging Mood Selections (TDD with Mocks)

**Scenario**

Your MoodAnalyticsManager must now record every mood the user selects. Rather than writing output directly, it should delegate logging to a separate component.

**Write the Test(RED)**

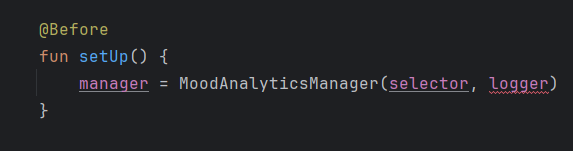
Before we can verify that our analytics manager actually calls out to a logger, we need to give our test fixture a mock logger to observe. In your existing MoodAnalyticsManagerTest, you already have a mocked TagSelector. We’ll add a second mock for MoodLogger, and then update your @Before setUp() to pass *both* mocks into the manager’s constructor.



With our new MoodLogger dependency in place, every instance of MoodAnalyticsManager must now be constructed with both a TagSelector and a MoodLogger.

In our test class, we already have a mocked TagSelector (annotated with @Mock). We’ll do the same for MoodLogger, so that Mockito can create a fake logger for us to verify against. Then, in the @Before setUp() method—which runs before each individual test—we pull our two mocks together and hand them to the manager’s constructor.

This change ensures that each test starts with a fresh MoodAnalyticsManager(selector, logger) and that we can observe *exactly* which moods it tells the logger to record. From here, we’re ready to write our first failing test (the “Red” step): asserting that *no* logging happens until we implement processMoods().



In our key test, shouldDelegateToSelectMoodAndLogEachUniqueMood(), we not only want to confirm *that* both selectMood(...) and logMood(...) are invoked for each distinct mood, but also *when* they happen relative to one another. By using Mockito‑Kotlin’s inOrder, we can assert the exact interleaving of calls across two mocks:

****

**Why inOrder?**

* A simple verify(...) would only confirm that each method was called at least once, but wouldn’t guarantee the correct sequencing.
* inOrder(selector, logger) lets us step through each expected call in turn, across both mocks.

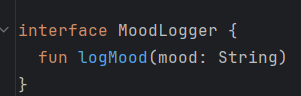
**Verification**

* We expect that for each *distinct* mood, the manager first delegates to the selector and *then* immediately logs that mood.
* By alternating order.verify(selector)... and order.verify(logger)..., we lock in that per‑mood order.

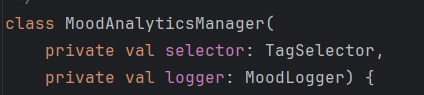
**Implement the code (GREEN)**

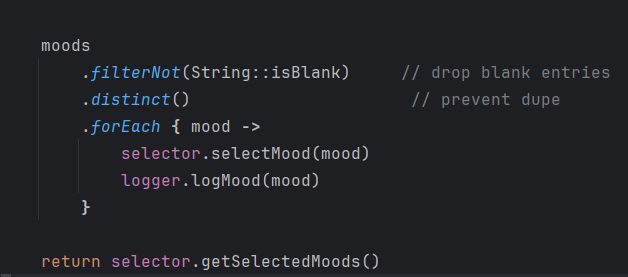
Now that our failing test is in place, it’s time to add just enough production code to make it pass. We’ll introduce the MoodLogger interface, update our manager to accept it, and then implement the loop that both selects and logs each mood.

1. Define the logging interface: (Create a new file MoodLogger.kt in your analytics package):



1. **Update the manager’s constructor** In MoodAnalyticsManager.kt, add a second parameter for your new interface:



1. **Implement the minimal loop -** Replace the empty or selector‑only logic in processMoods(...) with:

**QLC‑3.1: (Refactoring & Clean Up)**

Check out https://github.com/Edrzapi/Android\_tdd\_tag\_selector/tree/QLC-3.1-ANS for the solution.

**QLC‑3.2: (Timestamp Logging)**

Check out https://github.com/Edrzapi/Android\_tdd\_tag\_selector/tree/QLC-3.1-ANS for the solution.