# **Day 37 - 27 September 2025**

**Document Name:**Day 37 - hmuvvala@ - Hari Gopal Muvvala

### **Task 09**

**Question:**

In the context of Automation Testing, how does the Page Object Model (POM) improve test maintainability?

1. POM creates separate XML files for each UI element.
2. POM introduces a single test method for all page elements.
3. POM stores test data in property files.
4. POM abstracts UI elements as objects in separate classes, isolating changes in the UI from test logic.

**Answer:** 4

**Notes:**

* POM = design pattern for automation frameworks.
* Keeps locators and actions separate from test scripts.
* Improves maintainability: if UI changes, only page class changes.

### **Task 10**

**Question:**

In a POM framework, where should test data ideally reside?

1. In the page class itself.
2. In the test script, embedded as literals.
3. In external files (JSON, Excel, properties).
4. In browser cookies.

**Answer:** 3

**Notes:**

* Best practice: keep test data **outside code**.
* Makes framework **data-driven** and easier to update.

### **Task 11**

**Question:**

A POM automation framework has test failures due to frequent UI locator changes. What should they improve?

1. Use absolute XPath locators.
2. Store locators in external property files, abstracted in page classes.
3. Implement retry mechanism in every step.
4. Convert all locators to dynamic XPath.

**Answer:** 2

**Notes:**

* Externalizing locators isolates UI changes.
* Absolute XPaths are brittle and not recommended.

### **Task 12**

**Question:**

Why are mock objects used in JUnit tests for services calling external APIs?

1. Simulate external APIs and test service logic in isolation.
2. Provide GUI forms for mocking input.
3. Auto-generate test scenarios.
4. Allow browser-based execution.

**Answer:** 1

**Notes:**

* Mocks = simulate external services.
* Avoids dependency on network/API availability.

### **Task 13**

**Question:**

A JUnit test for email API actually sends emails. What should you do?

1. Replace with mock email service.
2. Remove the test.
3. Use assertNull() on email object.
4. Send emails to dummy address.

**Answer:** 1

**Notes:**

* Real actions (sending emails) → side effects, not unit test friendly.
* Mock services prevent accidental external triggers.

### **Task 14**

**Question:**

Which best describes the role of test suites in JUnit?

1. Group test methods in one class.
2. Group multiple test classes and run together (e.g., regression).
3. Run one test class multiple times.
4. Auto-generate mock objects.

**Answer:** 2

**Notes:**

* Test suites = collection of test classes.
* Useful for regression and integration runs.

### **Task 15**

**Question:**

Why are mock objects used in enterprise apps testing?

1. Eliminate need for assertions.
2. Replace real dependencies (DB, services) to test in isolation.
3. Automate UI validations.
4. Generate runtime test data.

**Answer:** 2

**Notes:**

* Mock objects → isolation, stability.
* Avoids relying on external systems during unit tests.

### **Task 16**

**Question:**

What does @Disabled do in JUnit?

1. Generate random input values.
2. Run test only if another test fails.
3. Replace method with mock at runtime.
4. Skip the test without deleting/commenting it.

**Answer:** 4

**Notes:**

* @Disabled = temporarily skip test execution.
* Useful for incomplete or broken tests.

### **Task 17**

**Question:**

Primary purpose of assertions in JUnit?

1. Logging before/after tests.
2. Ensure exceptions are thrown.
3. Define expected outcomes and verify correctness.
4. Configure execution order.

**Answer:** 3

**Notes:**

* Assertions = checkpoints.
* Automates validation of expected results.

### **Task 18**

**Question:**

Difference between manual vs automation testing in scalability?

1. Manual testing scales better.
2. Automation scales regression tests across builds/environments.
3. Manual tools are more reliable.
4. Automation allows ad-hoc testing only.

**Answer:** 2

**Notes:**

* Automation scripts can run unattended and parallel.
* Manual testing doesn’t scale with frequent releases.

### **Task 19**

**Question:**

Purpose of assertThat() with Hamcrest?

1. Parallel assertions in threads.
2. GUI-based results.
3. Flexible, readable expectations using matchers (containsString, hasItems).
4. Automate exception handling.

**Answer:** 3

**Notes:**

* Improves readability of assertions.
* Example: assertThat("Hari", containsString("Ha"));

### **Task 20**

**Question:**

How does timeout improve JUnit test reliability?

1. Always return true after time.
2. Ignore failures if too long.
3. Detect hanging/slow tests early.
4. Mark as pass even if interrupted.

**Answer:** 3

**Notes:**

* Timeout helps catch infinite loops or performance bottlenecks.
* Ensures builds don’t hang indefinitely.

### **Additional notes for myself:**

* Today reinforced that **Mockito is central for isolation testing**. I should remember the three main annotations:
  1. @Mock → dummy object (no real behavior).
  2. @Spy → partial mock (real object + mocked parts).
  3. @InjectMocks → auto-injects mocks into the class under test.
* verify() is the **key method** that shifts focus from just outputs to verifying **interactions and behavior**. This is important for testing service classes that call DAOs, APIs, or utilities.
* A lot of problems in automation frameworks come from **locator instability**. The Page Object Model (POM) is the right approach because it centralizes all locators and actions in one place. I just need to fix the page class when the UI changes, not every single test.
* **Data-driven testing** is another big concept: test data should always be external (Excel, JSON, properties), never hardcoded inside test cases. This separates logic from data.
* @Disabled is a good practice for skipping tests temporarily instead of deleting them. I should use this whenever a feature is pending.
* **Timeouts** in JUnit give reliability. They prevent infinite loops or long delays from blocking the build. This will be very useful in CI pipelines.
* Hamcrest improves test readability, especially for string checks, collections, and number ranges. I should default to using assertThat() whenever I want tests to read like natural English.
* ATLC (Automation Testing Life Cycle) looks like SDLC but focused on automation:
  1. Test planning and tool selection.
  2. Framework design and setup.
  3. Test script development.
  4. Test execution and reporting.
  5. Maintenance.
* A subtle point I realized: **unit tests (JUnit/Mockito)** are very different from **UI automation tests (Selenium + POM + TestNG)**. But both live under the same automation strategy. Unit tests give fast feedback, UI tests ensure end-to-end coverage.