# **Day 36 - 26 September 2025**

**Document Name:**Day 36 - hmuvvala@ - Hari Gopal Muvvala

### **Task 02**

**Question:**

What is JUnit?

**Answer:**

JUnit is a unit testing framework for Java, widely used to test individual units of code (methods, classes).

**Notes:**

* Helps write repeatable tests.
* Integrated with IDEs like IntelliJ/Eclipse.
* Forms basis of Test-Driven Development (TDD).

### **Task 03**

**Question:**

What are the benefits of JUnit?

**Answer:**

* Simplifies unit testing in Java.
* Provides annotations like @Test, @BeforeEach, @AfterEach.
* Automates test execution and reporting.

**Notes:**

* Encourages early bug detection.
* Improves confidence in code refactoring.
* Works with build tools (Maven/Gradle).

### **Task 04**

**Question:**

What is the difference between JUnit 4 and JUnit 5?

**Answer:**

* JUnit 4 uses @Before, @After; JUnit 5 uses @BeforeEach, @AfterEach.
* JUnit 5 introduced Jupiter API and modular architecture.
* JUnit 5 allows parameterized tests, better IDE integration.

**Notes:**

* Always check version: JUnit 5 (a.k.a. JUnit Jupiter) is modern standard.

### **Task 05**

**Question:**

What is the lifecycle of JUnit 5 test methods?

**Answer:**

1. @BeforeAll → runs once before all tests.
2. @BeforeEach → runs before each test.
3. @Test → actual test case.
4. @AfterEach → runs after each test.
5. @AfterAll → runs once after all tests.

**Notes:**

* Lifecycle ensures clean setup and teardown for each test.

### **Task 06**

**Question:**

What are assertions in JUnit?

**Answer:**

Assertions check expected vs. actual values in tests.

**Notes:**

* Examples: assertEquals(expected, actual), assertTrue(condition), assertThrows(Exception.class, () -> {...}).
* Assertion failures mark the test as failed.

### **Task 07**

**Question:**

Difference between assertEquals() and assertSame()?

**Answer:**

* assertEquals() → compares values using .equals().
* assertSame() → checks if two references point to the **same object**.

**Notes:**

* Use assertSame when object identity matters.

### **Task 08**

**Question:**

What is a test suite in JUnit?

**Answer:**

A test suite groups multiple test classes together to run them as a batch.

**Notes:**

* Useful for regression testing.
* In JUnit 5, can use @Suite annotation.

### **Task 09**

**Question:**

What is Hamcrest in JUnit?

**Answer:**

Hamcrest is a library providing matcher objects for more readable assertions.

**Notes:**

* Example:

assertThat(value, is(5));

assertThat(name, startsWith("Hari"));

* Improves readability vs. plain assertions.

### **Task 10**

**Question:**

What is Mockito?

**Answer:**

Mockito is a mocking framework for unit tests in Java.

**Notes:**

* Allows creation of mock objects to test code in isolation.
* Example:

when(service.getName()).thenReturn("Hari");

verify(service).getName();

* Commonly used with JUnit.

### **Task 11**

**Question:**

Difference between stubs and mocks?

**Answer:**

* **Stub** → provides hardcoded responses, no verification.
* **Mock** → can verify interactions and behavior.

**Notes:**

* Mockito focuses on mocks.
* Stubs = simpler, but limited.

### **Task 12**

**Question:**

What are annotations in Mockito?

**Answer:**

* @Mock → create mock object.
* @InjectMocks → injects mock into tested class.
* @Spy → partial mock (real object with some mocked behavior).

**Notes:**

* Use with MockitoAnnotations.openMocks(this) in setup.

### **Task 13**

**Question:**

What is the use of verify() in Mockito?

**Answer:**

It checks whether a specific method was invoked on a mock object.

**Notes:**

* Example:

verify(service).getName();

* Ensures expected behavior was executed.
* Useful for interaction testing (not just output).

### **Task 14**

**Question:**

What is the difference between when(...).thenReturn(...) and doReturn(...).when(...) in Mockito?

**Answer:**

* when(...).thenReturn(...) → standard way to stub a method call.
* doReturn(...).when(...) → used when stubbing **void methods** or spying real objects.

**Notes:**

* doReturn avoids calling the real method when using spies.

### **Task 15**

**Question:**

What is the difference between @Mock and @Spy?

**Answer:**

* @Mock → creates dummy object with no real logic.
* @Spy → wraps a real object, allows calling real methods unless explicitly stubbed.

**Notes:**

* Spy is used for partial mocking.
* Be careful: spies still execute real methods unless overridden.

### **Task 16**

**Question:**

What is the purpose of @InjectMocks?

**Answer:**

It injects mocked dependencies into the class under test.

**Notes:**

* Reduces boilerplate setup.
* Helps test a class in isolation while auto-injecting mocks.

### **Task 17**

**Question:**

What is TDD (Test-Driven Development)?

**Answer:**

A development process where tests are written **before** the actual implementation.

**Notes:**

* Cycle: **Red → Green → Refactor**.
* Ensures code meets requirements early.
* Common with JUnit.

### **Task 18**

**Question:**

What is BDD (Behavior-Driven Development)?

**Answer:**

BDD extends TDD by focusing on system **behavior** using natural language.

**Notes:**

* Uses tools like **Cucumber, JBehave**.
* Syntax: Given-When-Then.
* Improves collaboration between devs, testers, and business stakeholders.

### **Task 19**

**Question:**

What are parameterized tests in JUnit 5?

**Answer:**

Tests that run multiple times with different input parameters.

**Notes:**

* Use @ParameterizedTest.
* Sources: @ValueSource, @CsvSource, @MethodSource.
* Reduces code duplication.

### **Task 20**

**Question:**

What is the difference between assertThrows() and ExpectedException?

**Answer:**

* assertThrows() → JUnit 5 style, verifies exception is thrown.
* ExpectedException → JUnit 4 rule-based mechanism.

**Notes:**

* Prefer assertThrows() in modern JUnit 5.

### **Task 21**

**Question:**

What is continuous integration (CI)?

**Answer:**

A practice where developers integrate code frequently, and automated builds/tests validate changes.

**Notes:**

* Tools: Jenkins, GitHub Actions, GitLab CI.
* Detects bugs early.
* Fits well with JUnit test automation.

### **Task 22**

**Question:**

What is continuous testing?

**Answer:**

Automated execution of tests throughout the CI/CD pipeline.

**Notes:**

* Validates quality at every stage.
* Helps achieve faster release cycles.

### **Task 23**

**Question:**

What is a mocking framework?

**Answer:**

A tool that allows creation of mock objects to simulate dependencies.

**Notes:**

* Examples: Mockito, EasyMock, PowerMock.
* Useful for isolating unit tests.

### **Task 24**

**Question:**

What is the difference between unit testing and integration testing?

**Answer:**

* **Unit Testing** → tests small modules in isolation.
* **Integration Testing** → tests interactions between multiple modules.

**Notes:**

* Unit → fast, focused.
* Integration → slower, broader scope.
* Both are needed in CI pipelines.

### **Task 25**

**Question:**

What is the use of assertThat() in Hamcrest?

**Answer:**

It provides a more readable way to express test assertions.

**Notes:**

* Example:

assertThat(5, greaterThan(3));

assertThat("Hari", startsWith("Ha"));

* Improves test readability vs. plain assertions.

### **Task 26**

**Question:**

What is the difference between white-box and black-box testing?

**Answer:**

* **White-box testing** → tester knows internal code/logic.
* **Black-box testing** → tester focuses only on inputs/outputs, no knowledge of internal code.

**Notes:**

* White-box = unit testing level (dev perspective).
* Black-box = functional/system testing (QA perspective).

### **Task 27**

**Question:**

What is integration testing?

**Answer:**

It tests the interaction between different modules or components to ensure they work together correctly.

**Notes:**

* Comes after unit testing.
* Example: testing DAO + Service + Controller working together.

### **Task 28**

**Question:**

What is system testing?

**Answer:**

Testing the complete integrated system as a whole against the requirements.

**Notes:**

* End-to-end testing.
* Covers functional + non-functional aspects.

### **Task 29**

**Question:**

What is acceptance testing?

**Answer:**

Validates the system against business requirements and user expectations.

**Notes:**

* Usually done by clients/users.
* Example: UAT (User Acceptance Testing).

### **Task 30**

**Question:**

What is regression testing?

**Answer:**

Testing existing functionality after changes, to ensure no new bugs were introduced.

**Notes:**

* Automated regression suites are common.
* Essential in CI/CD pipelines.

### **Task 31**

**Question:**

What is smoke testing?

**Answer:**

A quick set of basic tests to check if the system build is stable enough for deeper testing.

**Notes:**

* Also called “Build Verification Testing”.
* Example: logging in, opening dashboard → if fails, reject build.

### **Task 32**

**Question:**

What is sanity testing?

**Answer:**

A narrow and focused test to verify a specific functionality after a bug fix or small change.

**Notes:**

* Sanity = check correctness of recent fix.
* Different from smoke: sanity is focused, smoke is broad.

### **Task 33**

**Question:**

What is alpha testing?

**Answer:**

Pre-release testing performed by internal teams within the organization.

**Notes:**

* Done before beta release.
* Controlled environment.

### **Task 34**

**Question:**

What is beta testing?

**Answer:**

Testing by a limited group of external users before final release.

**Notes:**

* Real-world usage scenarios.
* Helps uncover issues not visible in controlled tests.

### **Task 35**

**Question:**

What is static testing?

**Answer:**

Testing performed without executing code, e.g., code reviews, inspections, walkthroughs.

**Notes:**

* Focus on early defect detection.
* Cheap and effective.

### **Task 36**

**Question:**

What is dynamic testing?

**Answer:**

Testing by executing the code and validating outputs.

**Notes:**

* Includes unit, integration, system, acceptance testing.
* Opposite of static testing.

### **Task 37**

**Question:**

What is performance testing?

**Answer:**

Testing how the system performs under expected workloads.

**Notes:**

* Covers speed, responsiveness, stability.
* Tools: JMeter, LoadRunner.

### **Task 38**

**Question:**

What is load testing?

**Answer:**

Testing system behavior under normal and peak expected loads.

**Notes:**

* Measures throughput, response time.
* Goal: find bottlenecks.

### **Task 39**

**Question:**

What is stress testing?

**Answer:**

Testing system behavior beyond its normal limits to determine breaking point.

**Notes:**

* Helps find maximum capacity.
* Checks system recovery under extreme conditions.

### **Task 40**

**Question:**

What is scalability testing?

**Answer:**

Testing system’s ability to scale up or out when workload increases.

**Notes:**

* Example: doubling users, can system handle?
* Important for cloud apps.

### **Task 26**

**Question:**

What is the difference between white-box and black-box testing?

**Answer:**

* **White-box testing** → tester knows internal code/logic.
* **Black-box testing** → tester focuses only on inputs/outputs, no knowledge of internal code.

**Notes:**

* White-box = unit testing level (dev perspective).
* Black-box = functional/system testing (QA perspective).

### **Task 27**

**Question:**

What is integration testing?

**Answer:**

It tests the interaction between different modules or components to ensure they work together correctly.

**Notes:**

* Comes after unit testing.
* Example: testing DAO + Service + Controller working together.

### **Task 28**

**Question:**

What is system testing?

**Answer:**

Testing the complete integrated system as a whole against the requirements.

**Notes:**

* End-to-end testing.
* Covers functional + non-functional aspects.

### **Task 29**

**Question:**

What is acceptance testing?

**Answer:**

Validates the system against business requirements and user expectations.

**Notes:**

* Usually done by clients/users.
* Example: UAT (User Acceptance Testing).

### **Task 30**

**Question:**

What is regression testing?

**Answer:**

Testing existing functionality after changes, to ensure no new bugs were introduced.

**Notes:**

* Automated regression suites are common.
* Essential in CI/CD pipelines.

### **Task 31**

**Question:**

What is smoke testing?

**Answer:**

A quick set of basic tests to check if the system build is stable enough for deeper testing.

**Notes:**

* Also called “Build Verification Testing”.
* Example: logging in, opening dashboard → if fails, reject build.

### **Task 32**

**Question:**

What is sanity testing?

**Answer:**

A narrow and focused test to verify a specific functionality after a bug fix or small change.

**Notes:**

* Sanity = check correctness of recent fix.
* Different from smoke: sanity is focused, smoke is broad.

### **Task 33**

**Question:**

What is alpha testing?

**Answer:**

Pre-release testing performed by internal teams within the organization.

**Notes:**

* Done before beta release.
* Controlled environment.

### **Task 34**

**Question:**

What is beta testing?

**Answer:**

Testing by a limited group of external users before final release.

**Notes:**

* Real-world usage scenarios.
* Helps uncover issues not visible in controlled tests.

### **Task 35**

**Question:**

What is static testing?

**Answer:**

Testing performed without executing code, e.g., code reviews, inspections, walkthroughs.

**Notes:**

* Focus on early defect detection.
* Cheap and effective.

### **Task 36**

**Question:**

What is dynamic testing?

**Answer:**

Testing by executing the code and validating outputs.

**Notes:**

* Includes unit, integration, system, acceptance testing.
* Opposite of static testing.

### **Task 37**

**Question:**

What is performance testing?

**Answer:**

Testing how the system performs under expected workloads.

**Notes:**

* Covers speed, responsiveness, stability.
* Tools: JMeter, LoadRunner.

### **Task 38**

**Question:**

What is load testing?

**Answer:**

Testing system behavior under normal and peak expected loads.

**Notes:**

* Measures throughput, response time.
* Goal: find bottlenecks.

### **Task 39**

**Question:**

What is stress testing?

**Answer:**

Testing system behavior beyond its normal limits to determine breaking point.

**Notes:**

* Helps find maximum capacity.
* Checks system recovery under extreme conditions.

### **Task 40**

**Question:**

What is scalability testing?

**Answer:**

Testing system’s ability to scale up or out when workload increases.

**Notes:**

* Example: doubling users, can system handle?
* Important for cloud apps.

### **Task 41**

**Question:**

What is security testing?

**Answer:**

Testing to identify vulnerabilities, threats, and risks in the system, ensuring data and resources are protected.

**Notes:**

* Focus: confidentiality, integrity, availability.
* Examples: SQL injection, authentication bypass.
* Tools: OWASP ZAP, Burp Suite.

### **Task 42**

**Question:**

What is usability testing?

**Answer:**

Testing to check how user-friendly and intuitive the system is.

**Notes:**

* Focus on user interface, accessibility.
* Often involves end users.

### **Task 43**

**Question:**

What is compatibility testing?

**Answer:**

Testing to ensure the application works across different environments (browsers, OS, devices, networks).

**Notes:**

* Example: web app tested on Chrome, Safari, Edge.
* Essential for mobile apps.

### **Task 44**

**Question:**

What is recovery testing?

**Answer:**

Testing system’s ability to recover from crashes, hardware failures, or network outages.

**Notes:**

* Example: DB server crash → system should auto-recover.
* Focus on resilience.

### **Task 45**

**Question:**

What is maintainability testing?

**Answer:**

Testing how easy it is to maintain, update, and support the system.

**Notes:**

* Code readability, modular design.
* Important for long-term projects.

### **Task 46**

**Question:**

What is portability testing?

**Answer:**

Testing how easily an application can be transferred from one environment to another.

**Notes:**

* Example: moving app from AWS to Azure.
* Focus: reusability and platform independence.

### **Task 47**

**Question:**

What is reliability testing?

**Answer:**

Testing how consistently the system performs under given conditions over time.

**Notes:**

* Measures mean time between failures (MTBF).
* Example: uptime percentage.

### **Task 48**

**Question:**

What is compliance testing?

**Answer:**

Testing whether the system meets industry standards, laws, and organizational policies.

**Notes:**

* Examples: GDPR, HIPAA.
* Essential for regulated industries.

### **Task 49**

**Question:**

What is test automation?

**Answer:**

The process of using tools/scripts to automate execution of test cases.

**Notes:**

* Saves time, reduces manual effort.
* Tools: Selenium, JUnit, TestNG.

### **Task 50**

**Question:**

What is Selenium?

**Answer:**

Selenium is an open-source framework for automating web browser interactions.

**Notes:**

* Supports multiple languages (Java, Python, C#).
* Works across browsers (Chrome, Firefox, Edge).
* Core components: Selenium WebDriver, Selenium Grid.

### **Task 51**

**Question:**

What is the difference between Selenium WebDriver and Selenium Grid?

**Answer:**

* **WebDriver** → controls one browser instance at a time.
* **Grid** → enables parallel execution across multiple machines/browsers.

**Notes:**

* Use WebDriver for single tests, Grid for distributed execution.

### **Task 52**

**Question:**

What is the Page Object Model (POM)?

**Answer:**

A design pattern for Selenium test automation where each page of the app is represented as a class.

**Notes:**

* Improves maintainability and readability.
* Example: LoginPage.java contains locators + actions for login page.

### **Task 53**

**Question:**

What is the difference between implicit wait and explicit wait in Selenium?

**Answer:**

* **Implicit wait** → sets default wait time for all elements.
* **Explicit wait** → waits for specific condition on an element.

**Notes:**

* Use explicit waits for dynamic web apps.
* Example explicit wait:

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));

### **Task 54**

**Question:**

What is TestNG?

**Answer:**

TestNG is a testing framework inspired by JUnit and NUnit, designed for Java.

**Notes:**

* Provides annotations: @BeforeSuite, @AfterSuite, @Test.
* Supports data-driven tests, parallel execution.
* Commonly used with Selenium.

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

* JUnit is the backbone of **unit testing in Java**. I now understand how test lifecycle works: @BeforeAll, @BeforeEach, @Test, @AfterEach, @AfterAll. This lifecycle ensures every test starts clean, which avoids false positives/negatives.
* **Assertions** are the actual checkpoints of my tests. I should remember differences like:
  + assertEquals() → compares values.
  + assertSame() → compares object references.
  + assertThrows() → ensures exception handling is correct.
* **Hamcrest** is not a replacement for JUnit assertions but improves **readability**. Writing assertThat(x, greaterThan(5)) is far more expressive than a plain comparison.
* **Mockito** is extremely powerful for isolating tests.
  + @Mock → create dummy objects.
  + @Spy → wrap real objects for partial mocking.
  + @InjectMocks → automatically inject dependencies.
  + Key difference: stubs only return hardcoded values, but mocks also **verify behavior**.
* The **verify() method** in Mockito is important: it ensures the expected method was actually called. This moves tests from just “does output match?” to “was the correct behavior invoked?”.
* I should always keep in mind the subtle difference between:
  + when(...).thenReturn(...) → normal stubbing.
  + doReturn(...).when(...) → avoids real method calls, especially for spies.
* **TDD cycle (Red–Green–Refactor)**: write failing test first → make it pass → clean up. This discipline forces me to design cleaner, modular code.
* **BDD (Given–When–Then)**: builds on TDD but focuses on **behavior and business value**. The BDDMockitosyntax (given(...).willReturn(...)) makes tests closer to natural language.
* **Automation framework concepts** I should carry forward:
  + Regression tests → can be grouped into **test suites**.
  + POM (Page Object Model) → important for Selenium automation later.
  + Wait strategies in Selenium (implicit vs explicit) are parallel concepts to setup/teardown in JUnit — both prevent flaky tests.
* One key takeaway: **unit tests are not just about correctness of output**. They are about **confidence in refactoring, design quality, and integration readiness**.