JUnit & Mockito: Mastering Unit Testing in Java

Write Reliable, Maintainable, and Scalable Code with Confident Testing

Why Unit Testing Matters

- • Ensures individual units of code work as expected
- • Helps detect bugs early in development
- • Makes refactoring safer and faster
- • Improves code readability and maintainability
- • Builds developer confidence and team productivity

What is JUnit?

- • A widely-used testing framework for Java
- • Supports annotations like @Test, @BeforeEach, @AfterEach
- • Uses assertions to validate expected outcomes
- • Helps automate and repeat tests easily
- • Integrated with IDEs and build tools like Maven/Gradle

Essential JUnit Annotations

- @Test - Marks a method as a test case
- @BeforeEach - Runs before each test method
- @AfterEach - Runs after each test method
- @BeforeAll - Executes once before all tests
- @AfterAll - Executes once after all tests
- @Disabled - Skips the test temporarily

What is Mockito?

- • A mocking framework for unit tests in Java
- • Used to mock dependencies and test interactions
- • Avoids hitting databases or network APIs during tests
- • Simulates behavior and returns controlled outputs

Key Mockito Annotations

- @Mock - Creates a mock instance
- @InjectMocks - Injects mock dependencies
- @BeforeEach - Initializes mocks using MockitoAnnotations.openMocks(this);
- when().thenReturn() - Defines mock behavior
- verify() - Checks interactions with mock objects

JUnit + Mockito: Sample Test Case

- • Test a service layer by mocking repository:

- @Mock UserRepository userRepo;
- @InjectMocks UserService userService;

- @Test void testGetUser() {
- when(userRepo.findById(1)).thenReturn(Optional.of(user));

Best Practices for Unit Testing

- • Keep tests independent and repeatable
- • Name test methods clearly: testMethodName_condition_expectedResult
- • Focus on one scenario per test
- • Use mocks only when necessary
- • Don’t test implementation, test behavior

Conclusion

- JUnit and Mockito together form a powerful duo for unit testing in Java.
- Adopting them will drastically improve the reliability, maintainability,
- and confidence in your codebase.