# OBJECTIVE

## Understanding How Mocking Can Enhance Test-Driven Development (TDD)

Mocking is the process of simulating the behavior of real objects in controlled ways.

It allows developers to test units of code in isolation, without relying on actual implementations of dependencies.

Isolation ensures that tests focus only on the behavior of the component being tested.

Test Doubles are general terms for any kind of replacement object used in testing (includes mocks, stubs, and fakes).

- Mock: A fake object that verifies interactions (e.g., if a method was called).

- Stub: Provides predefined responses to method calls, used to control test scenarios.

- Fake: A working implementation, but simplified (e.g., an in-memory database).

Key Advantages of TDD with Mocking:

- Allows early testing of logic before the real dependencies are ready.

- Improves design by promoting low coupling and high cohesion.

- Encourages modular and testable code.

## Meaning of Mocking in Unit Testing and Why to Use Mocks

Mocking in unit testing refers to replacing real dependencies with mock versions.

This helps in isolating the unit under test, ensuring that tests are fast, reliable, and independent.

Mocks are useful when the actual dependency:

- Is unavailable or still under development.

- Is slow or non-deterministic (e.g., network, file system).

- Has side effects (e.g., sending emails, writing to database).

By using mocks and stubs, we can simulate responses and control the test environment precisely.

## Understanding the Basics of Dependency Injection (DI) and How It Helps Unit Testing

Dependency Injection (DI) is a design pattern where an object’s dependencies are provided from the outside, rather than the object creating them itself.

This enables better decoupling and easier testing.

Constructor Injection: Dependencies are passed via the class constructor.

Method Injection: Dependencies are passed via methods.

Benefits of DI for Unit Testing:

- Makes it easy to replace real dependencies with mocks or stubs.

- Promotes writing modular, reusable, and testable code.

- Reduces tight coupling between components.