# Advanced Unit Testing with Mocking and Dependency Injection

## 1. Understanding Mocking in TDD (Test-Driven Development)

Mocking is the process of creating simulated objects that mimic the behavior of real objects in controlled ways.  
It allows you to test units of code in isolation by replacing dependencies with test doubles.

Key Concepts:

* - Mocking: Creating controlled versions of dependencies.
* - Isolation: Ensuring the unit being tested does not rely on external systems.
* - Test Doubles: Includes mocks, fakes, stubs, and spies.

Mock vs Fake vs Stub:

* - Mock: Verifies interactions with dependencies.
* - Fake: Has working implementations but not production-ready (e.g., in-memory DB).
* - Stub: Provides pre-defined responses to calls.

Advantages of TDD:

* - Forces you to write only necessary code.
* - Leads to better design and modular code.
* - Reduces bugs and improves maintainability.

## 2. Why Use Mocks in Unit Testing?

Mocks allow you to isolate the system under test and eliminate the unpredictability of external dependencies.  
You can test behavior, ensure specific methods are called, and verify logic without hitting real services.

## 3. Basics of Dependency Injection (DI)

Dependency Injection is a design pattern where dependencies are provided to a class instead of being created by the class.

Types of DI:

* - Constructor Injection: Dependencies are passed via constructor.
* - Method Injection: Dependencies are passed through methods.

## 4. Creating Testable Code with Moq

Moq is a popular mocking library in .NET. It allows creation of mocks, setting expectations, and verifying behaviors.

Example:

var mockService = new Mock<IDataService>();  
mockService.Setup(x => x.GetData()).Returns("Hello");  
  
var controller = new MyController(mockService.Object);  
var result = controller.Get();  
Assert.AreEqual("Hello", result);

## 5. Mocking Database Access for Unit Tests

You can mock repository interfaces to simulate database operations.

Example:

var mockRepo = new Mock<IEmployeeRepository>();  
mockRepo.Setup(r => r.GetEmployeeById(1)).Returns(new Employee { Name = "Test" });

## 6. Mocking File System Access for Unit Tests

To mock file system interactions, abstract file operations behind interfaces and mock those interfaces in tests.

Example:

public interface IFileService {  
 string ReadFile(string path);  
}  
  
var mockFileService = new Mock<IFileService>();  
mockFileService.Setup(f => f.ReadFile(It.IsAny<string>())).Returns("File content");