**Advanced Unit Testing: Mocking and Dependency Injection**

**1. Mocking in TDD (Test-Driven Development)**

* **What is Mocking?**
  + Creating simulated (fake) objects that mimic real dependencies.
  + Enables **isolated testing** by replacing actual dependencies with **test doubles**.
* **Key Concepts**
  + **Mocking:** Creating controlled versions of dependencies.
  + **Isolation:** Testing units without relying on external systems or services.
  + **Test Doubles:** General term for objects like mocks, fakes, stubs, and spies used to replace real dependencies during testing.
* **Difference between Mock, Fake, and Stub**
  + **Mock:** Verifies interactions (checks if methods were called with expected arguments).
  + **Fake:** Has a working implementation but is not production-ready (e.g., in-memory database).
  + **Stub:** Provides predefined responses to method calls, without any logic.
* **Advantages of TDD**
  + Encourages writing only **necessary and testable code**.
  + Results in better **design, modularity, and maintainability**.
  + Reduces bugs through early validation.

**2. Why Use Mocks in Unit Testing?**

* **Purpose of Mocks**
  + Isolate the system under test from external dependencies.
  + Remove unpredictability from tests (e.g., network calls, database).
  + Verify behaviors, such as whether specific methods were called with correct arguments, without hitting real services.

**3. Basics of Dependency Injection (DI)**

* **What is Dependency Injection?**
  + A design pattern where **dependencies are provided** to a class rather than the class creating them itself.
* **Types of DI**
  + **Constructor Injection:** Pass dependencies via the constructor.
  + **Method Injection:** Pass dependencies via specific methods.

**4. Creating Testable Code with Moq (.NET)**

* **About Moq**
  + A popular mocking library for .NET.
  + Allows you to create mocks, set expectations, and verify 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 in Unit Tests**

* **Approach**
  + Mock repository interfaces to simulate database operations without hitting the real database.
* **Example**

var mockRepo = new Mock<IEmployeeRepository>();

mockRepo.Setup(r => r.GetEmployeeById(1))

.Returns(new Employee { Name = "Test" });

**6. Mocking File System Access in Unit Tests**

* **Approach**
  + Abstract file operations behind interfaces.
  + Mock these interfaces in tests to avoid actual file system access.
* **Example**

public interface IFileService {

string ReadFile(string path);

}

var mockFileService = new Mock<IFileService>();

mockFileService.Setup(f => f.ReadFile(It.IsAny<string>()))

.Returns("File content");