Spring Framework 6

Beginner to Guru

Introduction to Testing with MockMVC

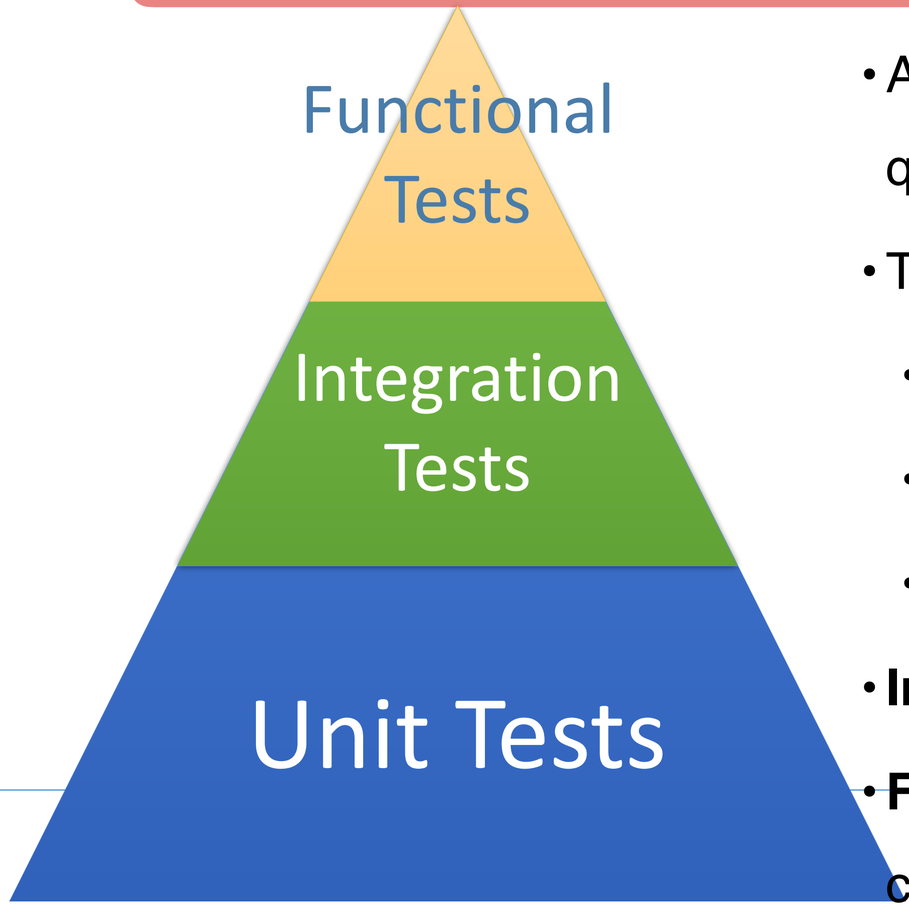
Testing Terminology

- **Unit Tests / Unit Testing** - Code written to test code under test
  - Designed to test specific sections of code
  - Percentage of lines of code tested is code coverage
  - Ideal coverage is in the 70-80% range
  - Should be ‘unity’ and execute very fast
  - Should have no external dependencies
  - ie no database, no Spring context, etc

Types of Tests

- **Integration Tests** - Designed to test behaviors between objects and parts of the overall system
  - Much larger scope
  - Can include the Spring Context, database, and message brokers
  - Will run much slower than unit tests
- **Functional Tests** - Typically means you are testing the running application
  - Application is live, likely deployed in a known environment
  - Functional touch points are tested - (i.e. Using a web driver, calling web services, sending / receiving messages, etc)

Testing Hierarchy



- All three types of tests play important roles for software quality
- The ***majority*** of tests should be **Unit Tests**
  - The foundation of your testing strategy
  - Small, fast, light weight tests
  - Very detailed and specific
- **Integration Tests** should be next largest category
- **Functional Tests** are smallest and least detailed of the categories.

Testing Spring MVC Controllers is Tricky

- Spring MVC Controllers are tricky to test property
- Controllers have a high degree of integration with the Spring MVC Framework
- Request path and HTTP Method decides which method to invoke
- Path variables are parsed from path
- JSON is bound to POJOs
- Response is expressed as a HTTP Response
- JUnit tests are **NOT** sufficient to test the framework interaction

Testing Spring MVC Controllers is Tricky

- Spring Mock MVC is a testing environment for the testing of Spring MVC controllers
  - Provides mocks of the Servlet runtime environment
    - HTPP Request / Response, Dispatcher Servlet, etc
    - Simulates the execution of controller as if it was running under Spring within Tomcat
- Can be run with or without the Spring Context
- True unit test when run without the Spring Context
- Technically an Integration Test when used in conjunction with Spring Context

Spring Boot Test Splices

- Spring Boot supports a concept of what is called Test Splices
- **Test Splices** bring up a targeted segment of the Auto-Configured Spring Boot Environment
  - ie - Just the Database components; or just the web components
  - User defined Spring beans typically are **NOT** initialized
- @WebMvcTest - is a Spring Boot test splice which creates a MockMVC environment for the controller (or controllers) under test
  - Dependencies of controllers are **NOT** included

Using Mocks

- Controller dependencies must be added to the Spring Context in the test environment
- Dependencies can be any proper implementation
  - Example of why we code to an interface, *any implementation* will work
  - We could easily use the hash map implementation we've been using in the course
- For testing, it is common to use mock objects
- Mocks allow you to supply a specific response for a given input
  - ie - when method abcd is called, return foo…

What is Mockito?

- Mockito is the most popular mocking framework for testing Java
- Mocks (aka Test Doubles) are alternate implementations of objects to replace real objects in tests
- Works well with Dependency Injection
- For the class under test, injected dependencies can be mc

Types of Mocks (aka Test Doubles)

- **Dummy** - Object used just to get the code to compile
- **Fake** - An object that has an implementation, but not production ready
- **Stub** - An object with pre-defined answers to method calls
- **Mock** - An object with pre-defined answers to method calls, and has expectations of executions. Can throw an exception if an unexpected invocation is detected
- **Spy** - In Mockito Spies are Mock like wrappers around the actual object

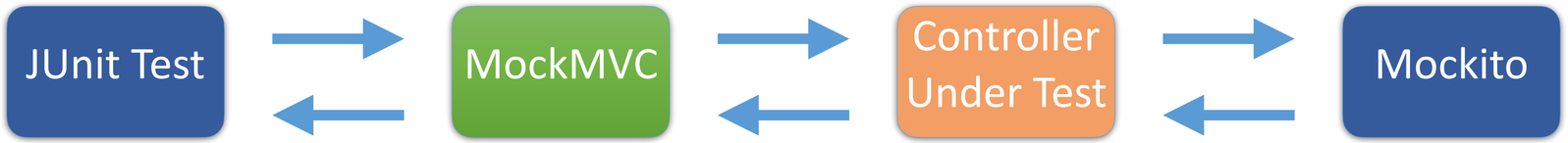
Important Terminology

- **Verify** - Used to verify number of times a mocked method has been called
- **Argument Matcher** - Matches arguments passed to Mocked Method & will allow or disallow
- **Argument Captor** - Captures arguments passed to a Mocked Method
  - Allows you to perform assertions of what was passed in to method

Testing Controllers with Mocks

- Argument captors can be used to verify request data is properly being parsed and passed to service layer
- Verify interactions can be used Mocked object was called
- Mock return values supply data back to controller
  - ie - object returned when getById is called on service
- Mocks can also be instructed to throw exceptions to test exception handling

Testing Controllers with MockMVC & Mockito



SPRING FRAMEWORK

GURU