Spring Core Unit –
Software Development Lifecycle

Lesson 1: Spring from 10,000 Feet

Lesson 1: Spring from 10,000 Feet

Overview

Spring is a collection of libraries that provide support for JVM-based enterprise applications. We will mostly use the Spring Framework portion of the larger Spring ecosystem in this course. Spring Framework provides support for Dependency Injection, Aspect Oriented Programming, MVC web applications, RESTful web services, and database connectivity.

Why Spring?

Spring was originally created as a reaction against the growing complexity of the Java frameworks that existed at the time. It's sole purpose was (and is) to simplify the programming and creation of Java applications. We'll start with the base features of Dependency Injection and Aspect Oriented Programming because almost all other features (i.e. MVC, web service, security, database support) build on this foundation.

When Spring started, the main framework for building enterprise Java applications was the Enterprise Java Bean (EJB) specification. This specification required complicated deployment descriptors and extra plumbing code. Over time Java developers began looking for a simpler, cleaner way to build complex applications - this is where Spring came into its own.

The Spring Approach

Spring employs four main strategies that are all aimed at simplifying Java development:

1. Lightweight development with Plain Old Java Objects (POJOs)
2. Loose coupling via Dependency Injection (DI) and interface orientation
3. Declarative programming using Aspect Oriented Programming (AOP) and convention over configuration
4. Reduction of boilerplate code through the use of templates

POJOs

Many frameworks (both historical and current) require you to extend their classes in order to take advantage of their features. This very often leads to code that is tightly coupled to the framework and essentially locks you into a particular vendor's solution. Spring allows you to use POJOs instead - this can help lead to more testable code that is more loosely coupled to the Spring framework - Spring strives to be minimally invasive to your code base.

Dependency Injection and Programming to Interfaces

Dependency Injection (DI) is a design pattern that implements a form of inversion of control (often these two terms are used interchangeably). The 'inversion' of control in this case is that client objects are no longer responsible for instantiating the objects (also known as services) on which they depend. Instead, the dependencies are handed to (i.e. injected into) the client objects by some other entity. Dependencies are handed to the client either through constructors or setter methods.

Using dependency injection has the following advantages:

1. Allows for loose coupling between the client and the concrete implementation of the service.
2. Allows the externalization (to configuration files) of the system's configuration information. This allows for configuration changes without forcing a recompilation of the application.
3. Allows for more flexible parallel development - developers can program against the interface and use stubbed or mock implementations while the real implementation of the component is being built.

Aspect Oriented Programming

Aspect oriented programming (AOP) is a technique that allows developers to place system wide code into reusable containers that can be applied across the code base. This technique promotes and allows good separation of concerns. Logging and security code are examples of cross cutting concerns that benefit from the AOP approach. Without AOP, this code is repeated throughout the codebase (a violation of the DRY principle). Further, this code is not core to the functionality of the components that contain it, contributing to less cohesive components.

Templates

Spring uses templates to reduce the need for boilerplate code in your applications. The first example below contains code that talks to a database without using Spring's JDBC template. The second example shows code that does the same thing using Spring JDBC template - even if you don't fully understand the code, it is clear that the second example is much cleaner than the first.