The Spring Framework: Foundations

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/spring.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Spring & Hibernate training, see courses at http://courses.coreservlets.com/.

Taught by the experts that brought you this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
- Java 5, Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF, Ajax, GWT, custom mix of topics
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details
Topics in This Section

- Motivation
- Spring Hello World
- POJO development
- Runtime environment
- Dependency injection
- Inversion of control

Motivation
Software Development Challenges

• Solutions are complex
• Requirements are constantly in flux
• Software architecture must be flexible
• Software components must be verifiable

EJB 2.0 Approach

• Complex products
• Unmaintainable systems
• Non-portable, framework-committed business components
• Unpredictable systems
Spring Approach

- Products based on simplicity
- Maintainable systems
- Framework-independent software
- Portable components
- Testable components
- Reliable and predictable systems

Pure Java

- Founded on POJO-based development
  - Ordinary Java classes that follow no special APIs
- Non-invasive for pre-existing POJOs
- Rewards framework-independent business logic
- Encourages new software to be written as POJOs
- Results in highly portable, reusable, and verifiable software
More With Less Custom Code

- Expand capabilities with less code
- Extensive and tested service abstractions
  - Email
  - JMS
  - JMX
  - JSF
  - JDBC
  - etc...
- Replaces generic corporate libraries
- Mitigates custom integration activities
- Consistency eases integration because spring platform is easy to use

Modular

- Helps only where needed
  - Modularity allows only relevant components to be introduced into the application
  - For instance choose one:
    - Spring BeanFactory
    - Spring JMX
    - Spring JDBC
  - Framework can be interfaced in deep or shallow layers.
  - Interfaces are consistent at each layer
- Turn-key solution
  - Spring components can be integrated quickly, with minimal effort and predictable results
  - Interfaces are clear and consistent
Widely Available

- Spring is integrated into numerous frameworks
- Broad adoption possible because the container is portable and lightweight
  - The container itself is designed as a POJO
- Integration without third-party support
- Performance overhead is rarely a consideration

Spring Jobs

- From indeed.com
  - Claims to compile data from most major job sites
Spring Setup

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

Spring Download

- http://www.springframework.org/download
  - Current version: 2.5.5 (6/2008)
    - Requires JDK 1.4+
  - spring-framework-2.5.5-with-dependencies.zip
    - Spring Framework binaries and source
    - Third-party binaries
    - Documentation
      - API
      - HTML reference
      - Project samples
      - HOW-TO guides
Spring Blank Project

- **spring-blank.zip**

<table>
<thead>
<tr>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Empty applicationContext.xml. For new Java source files.</td>
</tr>
<tr>
<td>lib</td>
<td>Minimum Spring JARs for API and runtime access to the Spring IoC container</td>
</tr>
<tr>
<td>build.xml</td>
<td>Optional Apache ANT build configuration</td>
</tr>
<tr>
<td>pom.xml</td>
<td>Optional Maven 2 build configuration</td>
</tr>
</tbody>
</table>

Spring Blank Project and Eclipse

- Download **spring-blank.zip**

- Import archive as an existing project into the current workspace
  - From the Eclipse menu bar select File and Import
  - From the Import (Select) dialog, select Existing Projects into Workspace and Next
  - From the Import (Import Projects) dialog, select the radio button Select archive file and Browse
  - Locate and select **spring-blank.zip** and select Open
  - Verify the project entry, **spring-blank**, to be present in the project list
  - Select Finish
Spring Blank Project and Apache Ant

- Download and unpack spring-blank.zip
- Install Apache Ant, version 1.6.5+
- Execute various Ant build commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean</td>
<td>Removes the build directory target and all nested build artifacts</td>
</tr>
<tr>
<td>compile</td>
<td>Compiles production Java source contents under src/main/java and places class binaries into target/classes</td>
</tr>
<tr>
<td>test</td>
<td>Executes the compile command and compiles and executes tests found under src/test/java</td>
</tr>
<tr>
<td>package</td>
<td>Packages production Java source and resource contents into a jar file. The jar package is placed in the build directory, target</td>
</tr>
</tbody>
</table>

Spring Blank Project and Maven 2

- Download and unpack spring-blank.zip
- Execute various Maven commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean</td>
<td>Removes the build directory target and all nested build artifacts</td>
</tr>
<tr>
<td>compile</td>
<td>Compiles production Java source contents under src/main/java and places class binaries into target/classes</td>
</tr>
<tr>
<td>test</td>
<td>Executes the compile command and compiles and executes tests found under src/test/java</td>
</tr>
<tr>
<td>package</td>
<td>Packages production Java source and resource contents into a jar file. The jar package is placed in the build directory, target</td>
</tr>
</tbody>
</table>

- See pom.xml configuration for additional dependency options
Spring Documentation

- **Top-level documentation page**
  - [http://www.springframework.org/documentation](http://www.springframework.org/documentation)
- **Wiki**
  - [http://opensource.atlassian.com/confluence/spring](http://opensource.atlassian.com/confluence/spring)
- **Forum**
  - [http://forum.springsource.org](http://forum.springsource.org)
- **Books**
  - *Spring Recipes*. APress 2008
  - *Spring in Action*. Manning 2007
  - *Agile Java Development with Spring, Hibernate and Eclipse*. Sams 2006
Spring Hello World

- Code a plain Java class model
  - Use the interface pattern by coding a `HelloWorld` interface and a `HelloWorldImpl` implementation
- Configure the Spring IoC container
- Instantiate the Spring IoC container
- Acquire the object from the Spring IoC container
  - The client must only have knowledge of the interface, `HelloWorld`

Plain Java Class Model

```java
public interface HelloWorld {
    public void execute();
}

public class HelloWorldImpl implements HelloWorld {
    public void execute() {
        System.out.println("Hello World!");
    }
}
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <bean id="helloWorld"
       class="coreservlets.HelloWorldImpl" />

</beans>

import org.springframework.beans.factory.*;
import org.springframework.context.support.*;
import org.springframework.context.support.*;

public class Main{
    public static void main(String[] args) {
        BeanFactory beanFactory =
            new ClassPathXmlApplicationContext(
            "applicationContext.xml");

        HelloWorld helloWorld =
            (HelloWorld)beanFactory.getBean("helloWorld");

        helloWorld.execute();
    }
}

Hello World!
Introduction

• **Plain Old Java Object**
  • What is it?
    – Business logic
    – Framework independent
  • **What it’s not**
    – Limited to the value object pattern
    – Framework implementation software
• **Features**
  – Portable
  – Testable
  – Flexible
POJO Development Process

- Describe the system agents and interactions
  - POJO behavioral classes, domain model, and dependencies
- Determine component responsibilities
  - Methods
- Identify information items discovered during program execution
  - Method parameters
- Identify information available during initialization
  - Initialization parameters for constructor, setter, or factory

Java EE training: http://courses.coreservlets.com

POJO Development Process Example

- Agents
  - BookLibrary and Client
- Interactions
  - Client uses BookLibrary
  - BookLibrary aggregates Book
- Responsibilities
  - BookLibrary must search for books by title
  - Clients must supply search parameters; i.e. title values

Java EE training: http://courses.coreservlets.com
POJO Development Process

• Develop implementation

[Diagram showing the development process with Book, BookLibrary, Client, JavaBookLibrary, and JPABookLibrary]

• Plan for change

[Diagram showing the plan for change with Book, BookLibrary, Client, JavaBookLibrary, and JPABookLibrary]

Java EE training: http://courses.coreservlets.com
POJO Development Process

- Plan for new and additional dependencies

Java EE training: http://courses.coreservlets.com

- Plan for complex configuration requirements

Java EE training: http://courses.coreservlets.com
public class Book {
    private String title;

    public Book(String title) {
        this.title = title;
    }

    public String getTitle() {
        return title;
    }

    public String toString() {
        return title;
    }
}
import java.util.List;

public interface BookLibrary {
    public List<Book> search(String title);
}

public class JavaBookLibrary implements BookLibrary {
    private List<Book> books;

    public JavaBookLibrary() {
        this.books = Arrays.asList(
                new Book("Core Servlets and JavaServer Pages"),
                new Book("More Servlets and JavaServer Pages"));
    }

    public List<Book> search(String title) {
        List<Book> results = new ArrayList<Book>();
        for(Book book : books){
            if(book.getTitle().contains(title)){
                results.add(book);
            }
        }
        return results;
    }
}
public class BookReader {

    private BookLibrary bookLibrary;

    public BookReader() {
        this.bookLibrary = new JavaBookLibrary();
    }

    public List read() {
        List<Book> books = bookLibrary.search("Java");
        for (Book book : books) {
            System.out.printf("Reading: %s%n", book);
        }
        return books;
    }
}
Runtime Model

- Transition from a class system to an object system
- An object model provides a unique and specific instantiation of the class specification

![Class Model and Object Model Diagram]

Runtime Context

- Multiple deployment contexts
- Complex object models should be portable
- Object models should be configurable to support changes between environments

![Multiple Context Diagram]
### Runtime Example

```java
public class Main {
    public static void main(String[] args) {
        BookReader client = new BookReader();
        List<Book> books = client.read();
        System.out.printf("Client read: %s books\n", books.size());
    }
}
```

**Standard output**

Reading: Core Servlets and JavaServer Pages
Reading: More Servlets and JavaServer Pages
Client read: 2 books

---

### Model Analysis

- **Hard-coded implementation choices**
  - Object model **cannot** be reconfigured
  - Future implementation types **cannot** be used without modifying and rebuilding BookReader
Model Analysis

- **Hard-coded model configuration**
  - Object model is *not* portable

---

Manual (Non-Spring) Dependency Injection

Customized Java EE Training: [http://courses.coreservlets.com/](http://courses.coreservlets.com/)

Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.

Developed and taught by well-known author and developer. At public venues or onsite at your location.
Dependency Injection

• Delivers object dependencies at runtime
• Encourages the separation of responsibilities
• When used with the interface pattern
  • Isolates implementations from clients
  • Minimizes the impact on clients when implementations evolve

Dependency Injection Process

• Design depending types to receive implementations
  – Allow dependencies to be supplied using property setters or constructors
    – Other dependency injection methods are also available, such as field injection, but requires third-party or Java reflection support
• Avoid constructing objects from the client to fulfill dependencies
  – For example, do not use the `new` operator to manage services
public class BookReader {

  private BookLibrary bookLibrary;

  public BookReader() {
    this.bookLibrary = new JavaBookLibrary();
  }

  public List read() {
    List<Book> books = bookLibrary.search("Java");
    for (Book book : books) {
      System.out.printf("Reading: %s%n", book);
    }
    return books;
  }
}

49

Java EE training: http://courses.coreservlets.com

public class BookReader {

  private BookLibrary bookLibrary;

  public BookReader(BookLibrary bookLibrary) {
    this.bookLibrary = bookLibrary;
  }

  public List read() {
    List<Book> books = bookLibrary.search("Java");
    for (Book book : books) {
      System.out.printf("Reading: %s%n", book);
    }
    return books;
  }
}

50

Java EE training: http://courses.coreservlets.com
Runtime Example

```java
public class Main {
    public static void main(String[] args) {
        BookLibrary service = new JavaBookLibrary();
        BookReader client = new BookReader(service);
        List<Book> books = client.read();
        System.out.printf("Client read: %s books\n", books.size());
    }
}
```

Standard output:

Reading: Core Servlets and JavaServer Pages
Reading: More Servlets and JavaServer Pages
Client read: 2 books

Model Analysis

- **Dynamic implementation choices**
  - Object model **can** be reconfigured
  - Future implementation types **can** be used without modifying and rebuilding BookReader
Model Analysis

- Hard-coded model configuration
  - Object model is not portable

---

Manual (Non-Spring) Inversion of Control

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.
Introduction

• **Inversion of Control**
• **Separate program control responsibilities**
  – Object instantiation
  – Dependency injection
• **Dependency injection is a type of IoC**

Inversion of Control Example

• **Previously used IoC**
  – Dependency injection example demonstrated inversion of control
  – Moved `JavaBookLibrary` selection and instantiation out of `BookReader` and into `Main`
IoC Framework

- **Service provider or plugin framework**
  - Interface
  - Providers
  - **Registration system**
  - Access API
    -- Joshua Bloch from *Effective Java*

- **Process**
  - Framework uses supplied APIs
    - Framework handles creation
    - Framework handles dependency injection
  - Runtime context uses framework

---

IoC Framework Example

```java
import coreservlets.BookReader;
import coreservlets.JavaBookLibrary;

public class ServiceProviderFramework {

    private BookReader bookReader;

    public ServiceProviderFramework(){
        this.bookReader =
            new BookReader(new JavaBookLibrary());
    }

    public BookReader getBookReaderInstance() {
        return this.bookReader;
    }
}
```

Implicit registration

Access API
IoC Framework Example

```java
public class Main {
    public static void main(String[] args) {
        ServiceProviderFramework framework = new ServiceProviderFramework();
        BookReader client = framework.getBookReaderInstance();
        List books = client.read();
        System.out.printf("Client read: %s books\n", books.size());
    }
}
```

Model Analysis

- Dynamic implementation choices
- Portable model configuration
Wrapup

Summary

- **Develop POJOs**
  - Avoid framework dependencies
  - Capture business logic
  - Avoid implementation commitments by using inversion of control and dependency injection patterns

- **Create a new XML file, applicationContext.xml, based on spring-beans.xsd**
  - Place applicationContext.xml in the classpath

- **Register POJOs**
  - Declare POJOs using XML bean elements
  - Use bean attributes id and class for specifying the name and type, respectively
Summary (Continued)

- **Instantiate a Spring IoC container**
  - Use the BeanFactory implementation
    - `ClassPathXmlApplicationContext` for integration with configuration files located in the classpath
    - See: `org.springframework.context.support.ClassPathXmlApplicationContext`
  - `ClassPathXmlApplicationContext` for integration with configuration files located in the classpath
- **Access the Spring IoC container**
  - Retrieve objects from the Spring IoC container using the bean accessor methods
    - For example, `BeanFactory#getBean(...) : Object`
  - Specify the object name for the method parameter
    - `beanFactory.getBean("bookLibrary");`