The Spring Framework Part 1: Fundamentals, Injection, AOP, Beginning MVC

VERHOEF TRAINING
103 Hillside Avenue
West Caldwell, NJ 07006
1-800-631-0410
www.verhoef-training.com

Spring Framework(Part 1)
Fundamentals,
Injection, AOP,
Beginning MVC

I. Java EE and Spring

3

The Need for Something Better

- J2EE provides low-level enterprise services
  - Distributed services
  - Object naming
  - Transaction control, etc.
- Something is needed to manage an entire enterprise architecture

Enter the ***Spring*** Framework!

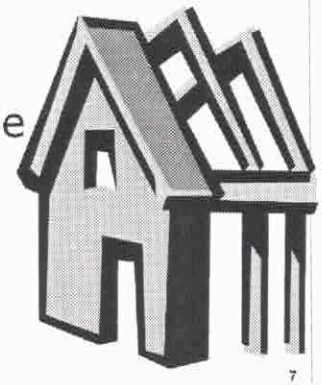
5

Notes:

While the J2EE is one of the most comprehensive enterprise service APIs, it requires lots of repetitive, unnecessary, and low-level code writing to achieve its goals. It also requires heavyweight containers and often forces us to implement features we do not need. This has opened the door for J2EE critics to attack its performance over the years

Do I Really Need Another Framework?

- It emphasizes development of business objects
  - You spend less time on JEE APIs
- Only need to incorporate the services *you* desire
- Testing and Integration are made simpler

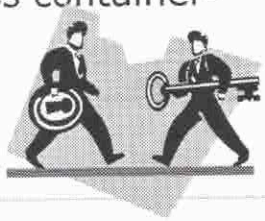


7

Notes:

Compatibilities

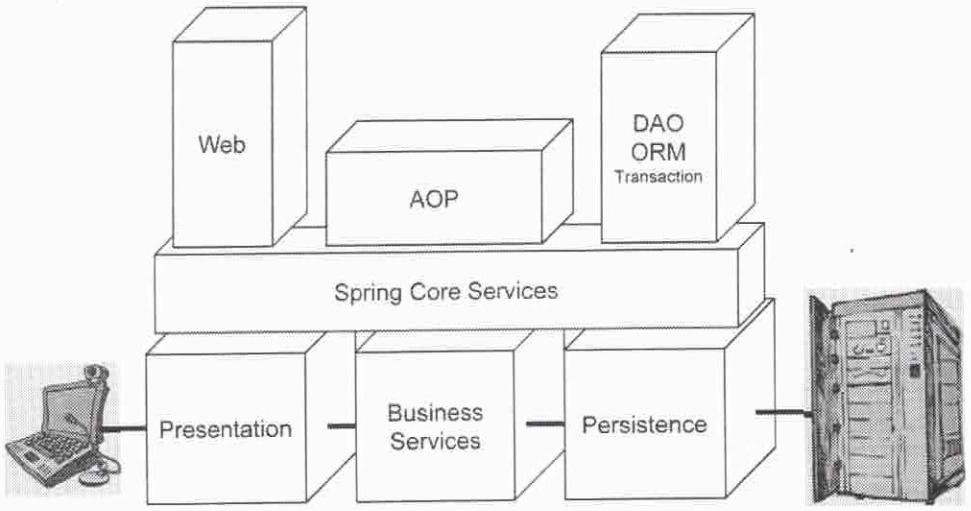
- Spring 2.0 works with Java 1.3 or later
- Spring 2.5 requires Java 1.4 or later
- Spring 3.0 requires Java 1.5 or later
- Spring 4.0 requires Java 1.8 or later
- Spring does not require Java EE libraries or a Java EE container
  - Unless specifically trying to access container-based services



9

Notes:

Spring Framework



11

Notes:

An Initial Application

- Let's begin with a simple example...
  - This application returns the day of the week for a given year, month, day

This application lacks a separation of its services and presentation layer

```
public class Driver
{
    public static void main(String[] args)
    {
            int year  = 2006;
            int month = 9;                          // October
            int day   = 16;
            Calendar c = new GregorianCalendar(year, month, day);
            System.out.println(
                    new SimpleDateFormat("EEEE").format(c.getTime()));
    }
}
```

13

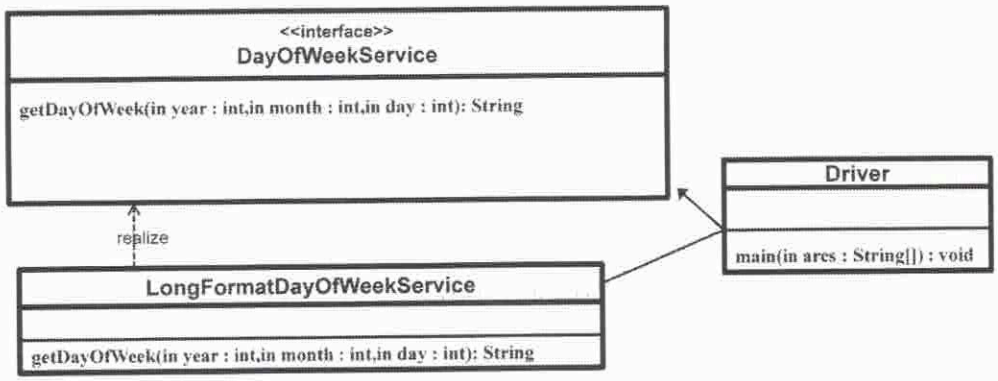
Notes:

Outputs 'Monday'

Imports have been omitted.

Programming to Interfaces

```
Public interface DayOfWeekService
{
    public String getDayOfWeek(int year, int month, int day) ;
{
```



15

Notes:

The first step towards creating a decoupled layered solution is to abstract away the concrete classes, basing them instead on interfaces. In this scenario, a new concrete class, called LongFormatDayOfWeekService, is created based on the interface, DayOfWeekService.

The Service Locator Pattern

```
public class Driver {
    public static void main(String[] args) {
        int year = 2006;
        int month = 9;// October
        int day = 16;
        DayOfWeekService service =
                (DayOfWeekService) Locator.getService("long");
        String dayOfWeek = service.getDayOfWeek(year, month, day);
        System.out.println(dayOfWeek);
    }
}
```

```
    public class Locator {
        public static Object getService(String serviceName) {
        Object service = null;
        // instantiate service based on serviceName
    return service;
    }
}
```

17

Notes:

The service locator pattern provides a nice means for not coupling the implementation with the presentation. In this example, a class called "Locator" is used to obtain and instantiate the resource associated with the string literal "long". In other words, when Locator's getService() method is passed the value "long", an instantiated service is returned.

Spring JARs

org.springframework.aop
org.springframework.asm
org.springframework.aspects
org.springframework.beans
org.springframework.context.support
org.springframework.context
org.springframework.core
org.springframework.expression
org.springframework.instrument.tomcat
org.springframework.instrument
org.springframework.jdbc
org.springframework.jms
org.springframework.orm
org.springframework.oxm
org.springframework.spring-library
org.springframework.test
org.springframework.transaction
org.springframework.web.portlet
org.springframework.web.servlet
org.springframework.web.struts
org.springframework.web

19

Notes:

Create a XML Config File

```
<?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-3.0.xsd">

<bean id="long" class="com.company.springclass.examples.ch01.
                       service.LongFormatDayOfWeekService"></bean>

<bean id="short" class="com.company.springclass.examples.ch01.
                       service.ShortFormatDayOfWeekService"></bean>

</beans>
```

21

Notes:

In this example, the long class names have been wrapped onto the next line. In practice, they should be continuous lines.

bean 3 STANDARDS
1. GETTER / SETTERS
2. IMPLEMENT SERIALIZABLE
3. DEFAULT CONSTRUCTOR

Spring Capabilities

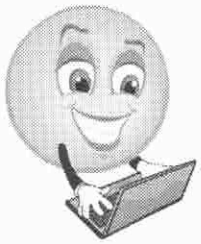
- This example only begins to scratch the surface!
  - Note how easy it is to change out services
  - Spring's IOC capabilities go well beyond acting as a Service Locator!

23

Notes:

Lab 1

- *Creating Multiple Services*
  - Follow the instructions found in the ***SpringLab01/instructions.html*** file to complete the exercise



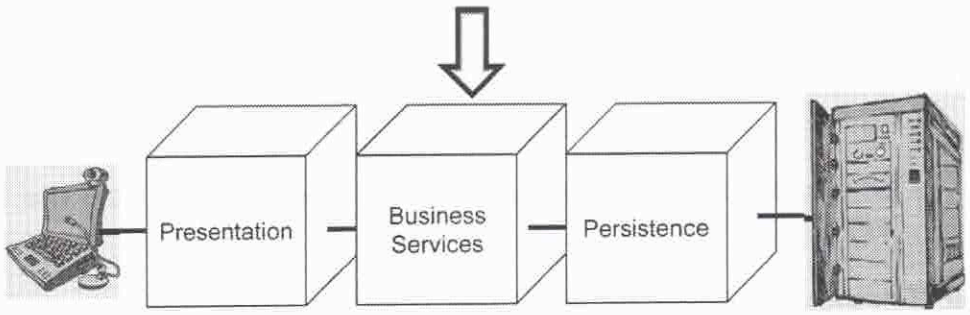
25

Notes:

II. Spring IOC

27

Inversion of Control



Spring IOC uses a special form of control called:

Dependency Injection

29

Notes:

Setter Injection

applicationContext-beans.xml

```
<?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-3.0.xsd">

  <bean id= "customerBean"
        class= "com.company.springclass.examples.ch02.Customer">
    <property name="name">
      <value>John Smith</value>
    </property>"
    <property name="customerId">
      <value>1</value>
    </property>
  </bean>

</beans>
```

These values are injected into the bean by the IOC Container

31

Notes:

Setter Injection ...

```
public class Driver
{
    public static void main (String[] args)
    {
        ApplicationContext ctx =
                        new ClassPathXmlApplicationContext(
                        "applicationContext-beans.xml");

        Customer customer = (Customer) ctx.getBean("customerBean");

        System.out.println(customer);
    }
}
```

Outputs: John Smith 1

33

Notes:

Constructor Injection ...

```
public class Driver
{
    public static void main (String[] args)
    {
        ApplicationContext ctx =
                        new ClassPathXmlApplicationContext(
                        "applicationContext-beans.xml");

        Customer customer = (Customer) ctx.getBean("customerBean");

        System.out.println(customer);
    }
}
```

Outputs: Andrew McDonald 2

35

Notes:

Injecting Beans into Beans ...

applicationContext-beans.xml

```
<bean id="orderBean
        class="com.company.springclass.examples.ch02.Order">
    <property name="orderId">
      <value>100</value>
    </property>
    <property name=""customer">
      <ref local="customerBean"/>
    </property>
</bean>

<bean id="customerBean"
        class="com.company.springclass.examples.ch02.Customer">
    <property name="name">
      <value>John Smith</value>
    </property>
    <property name="customerId">
      <value>1</value>
    </property>
</bean>
```

Outputs: 100 John Smith 1 0.0

37

Notes:

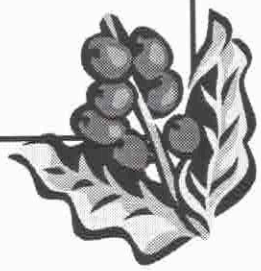
For beans defined within other beans (in Spring these are called inner beans), the singleton, params, and name attributes are effectively ignored for that inner bean.

XML Configuration Files

```
<?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-3.0.xsd">

 <!- - bean definitions here - ->

</beans>
```



39

Notes:

Spring Multiple Schemas

Many of the Spring config tags have been placed into different schema

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLschema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:jee="http://www.springframework.org/schema/jee"

xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
              http://www.springframework.org/schema/util
       http://www.springframework.org/schema/util/spring-util-3.0.xsd
              http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
              http://www.springframework.org/schema/aop
       http://www.springframework.org/schema/aop/spring-aop-3.0.xsd"
              http://www.springframework.org/schema/jee
       http://www.springframework.org/schema/jee/spring-jee-3.0.xsd"
              http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context3.0.xsd >
```

41

Notes:

The Spring config file has expanded in terms of the tags that may be used. So, it now exists in multiple schema. If you use tags from a specific schema, for example, the context schema, then you must include the *xmlns:* prefix for that schema, and in *the xsi:schemaLocation* attribute the *xsd* file and it's url must be included.
In the example above, namespace values are only needed if tags from that namespace are used.

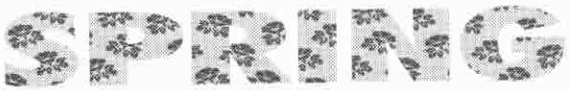
id and *name* Attributes

```
<bean id="myBean"  class="java.lang.String"
        name="myOtherBean, anotherBean">
                <constructor-arg>
                        <value>It's Spring Time!</value>
                </constructor-arg>
</bean>
```

```
ApplicationContext context = new
ClassPathXmlApplication (
                        "applicationContext-beans.xml");
String myBean = (String)  context.getBean("myBean");
String myOtherBean = (String) context.getBean("myOtherBean");
String anotherBean = (String) context.getBean("anotherBean");

System.out.println(myBean);
System.out.println(myOtherBean);
System.out.println(anotherBean);
```

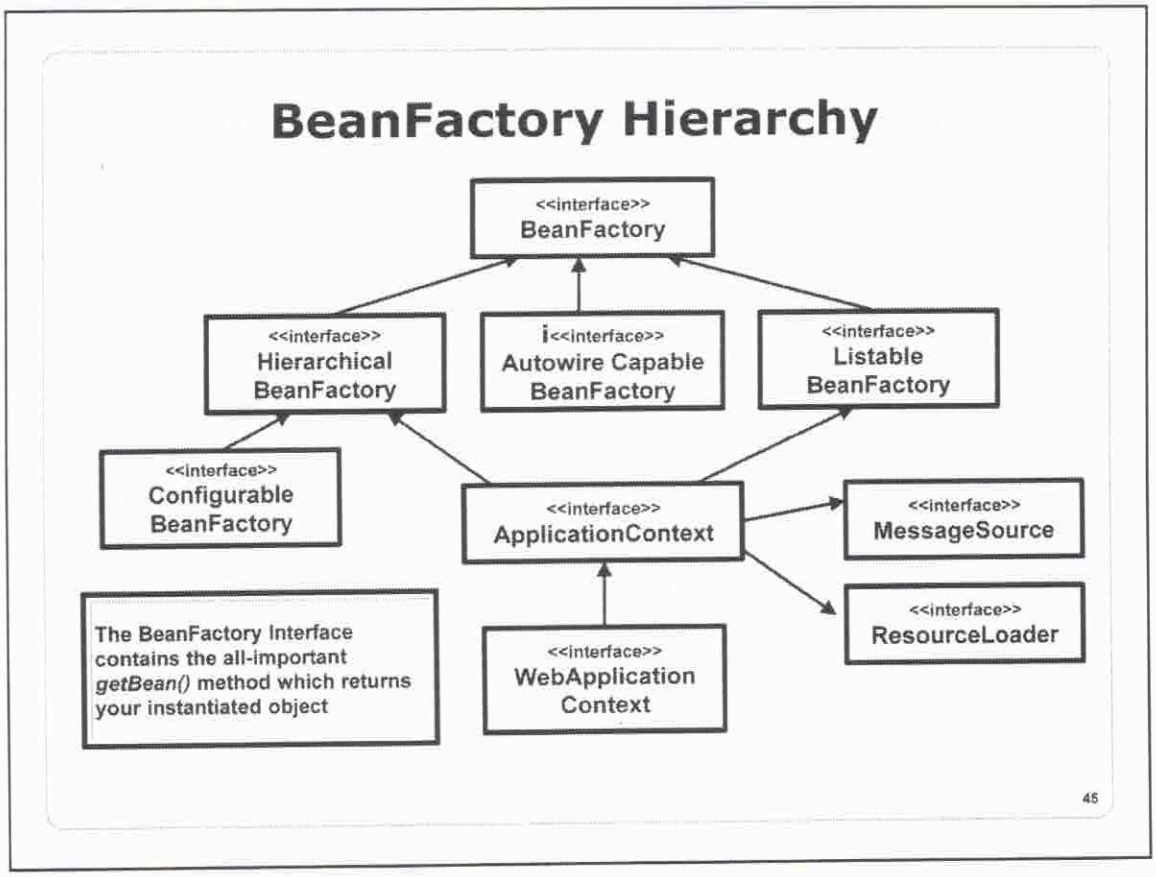
Outputs 'It's Spring Time' 3 times.



43

Notes:

The name attribute of the bean element serves as an alias for that bean. You may use the id or the name element to obtain a bean.


BeanFactory Hierarchy
<<interface>>
BeanFactory
<<interface>>
Hierarchical
BeanFactory
<<interface>>
Autowire Capable
BeanFactory
<<interface>>
Listable
BeanFactory
<<interface>>
Configurable
BeanFactory
<<interface>>
ApplicationContext
<<interface>>
MessageSource
<<interface>>
ResourceLoader
<<interface>>
WebApplication
Context
The BeanFactory Interface contains the all-important getBean() method which returns your instantiated object
45

Notes:

The BeanFactory Interface

- All *BeanFactory* Classes implement this interface

  org.springframework.beans.factory.BeanFactory

- The *BeanFactory* Interface methods

  Object getBean(String name)
  Object getBean(String name, Class type)
  Class getType(String name)
  boolean containsBean(String name)
  boolean isSingleton(String name)
  String[] getAliases(String name)

47

Notes:

Creating the Container

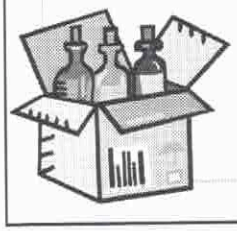
- Examples of Creating the IOC Container

```
ApplicationContext context = new
    ClassPathXmlApplicationContext("applicationContext-beans.xml");
```

Locates a bean factory definition file on the classpath

```
ApplicationContext context = new FileSystemXmlApplicationContext(
                    "/apps/config/applicationContext-beans.xml");
```

Locates a bean factory definition file from a file system path



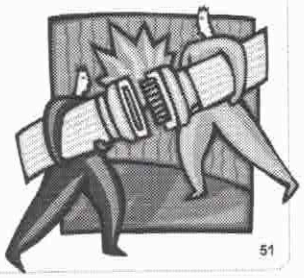
49

Notes:

Generally the ClassPathXmlApplicationContext is the chosen concrete beanfactory implementation.

Spring-based Services

1) Create the Service Interface
   (CatalogManager)
2) Create a Service Implementation
   (CatalogManagerImplementation)
3) "Wire up" the Beans
4) Create an IOC Container
5) Get and work with the Beans



Notes:

Create a Service Implementation

```
public class CatalogManagerImplementation implements CatalogManager
{
    List products;

    public CatalogManagerImplementation(List products) {
                this.products = products;
    }

public Product getProduct (int productId)
    {
        return null;
    }
    public List getProducts()
    {
        return products;
    }
    public String toString() {
                return (products == null ? "" : products.toString());
    }
}
```

53

Notes:

Creating an IOC Container

```
package com.company.springclass.examples;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.company.springclass.services.CatalogManager;

public class Driver {
    public static void main(String[] args) {
        ApplicationContext context =
                new ClassPathXmlApplicationContext(
                        "applicationContext-beans.xml");

        // obtain and work with beans...
    }
}
```

55

Notes:

Lab 2: Using Spring's DI

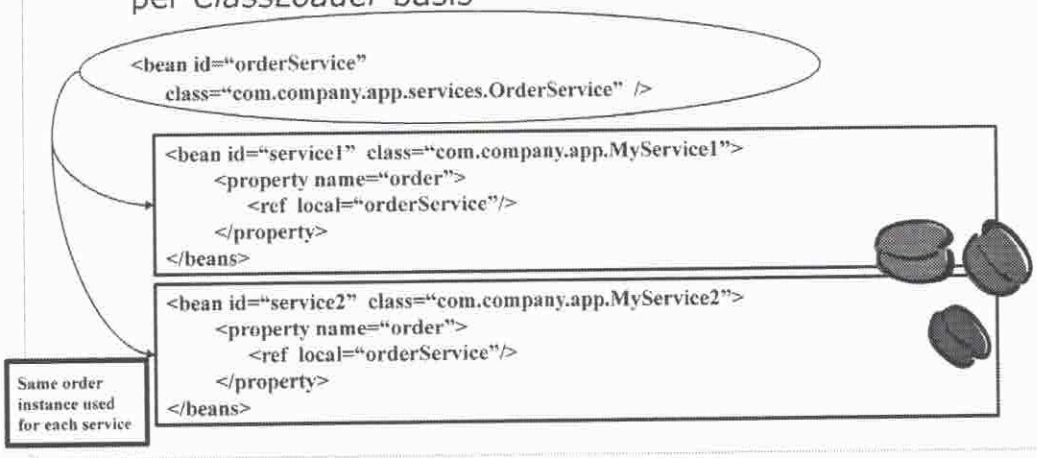
- In this exercise, you will create a Catalog and use Spring's constructor injection to insert products
- Open and read ***instructions.html*** in the **SpringLab02** project and follow the additional steps provided

57

Notes:

Singleton Beans

- In Spring, only one instance of a Singleton bean will be created per Factory
  - In most design pattern strategies, Singletons exist on a per *ClassLoader* basis



59

Notes:

Spring-defined singleton beans are not Gang of Four-defined Singletons. It is entirely possible for a single spring configuration file to define multiple beans with the same class.

Request and Session Beans

- Beans defined with request scope will be instantiated for each HTTP request

```
<bean id="orderBean"
      class="com.company.app.beans.OrderBean"
      scope="request"/>
```

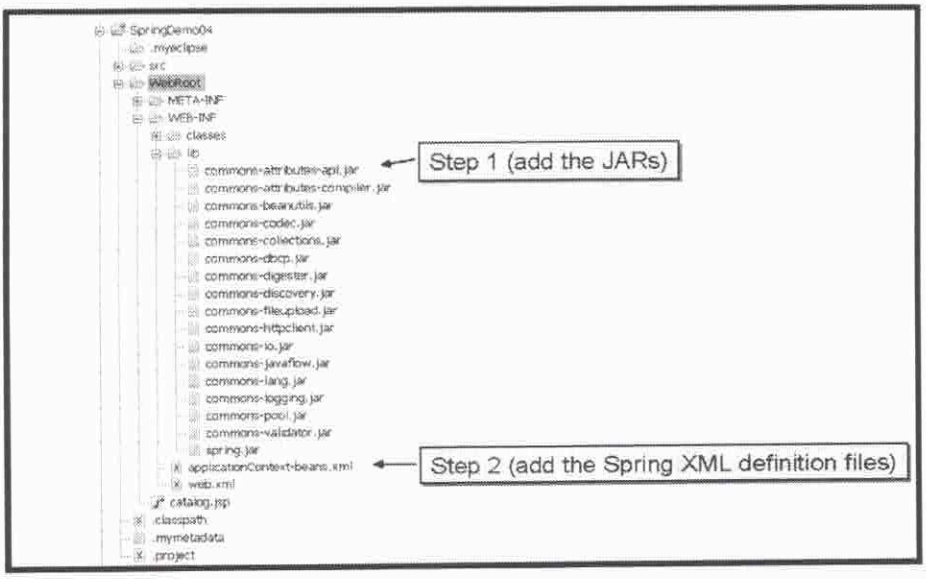
- Spring creates and properly scopes a new bean instance for each new HTTP Session created

```
<bean id="orderBean"
      class="com.company.app.beans.OrderBean"
      scope="session"/>
```

61

Notes:

Spring-WebApp Configuration



63

Notes:

```
<servlet>
    <servlet-name>context</servlet-name>
    <servlet-class>
            org.springframework.web.context.ContextLoaderServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Accessing Beans in a Web App

- Obtain the *ApplicationContext* from within your Web apps by using the Spring-provided utility ***WebApplicationContextUtils***

```
ApplicationContext context =
WebApplicationContextUtils.getWebApplicationContext(
                                                getServletContext());
CatalogManager catalog =
                            (CatalogManager) context.getBean("catalog");
out.println(catalog);
```

65

Notes:

This works from within Struts, JSF, Servlets, etc.

Lab 2b: Web-Enabling Spring

- In this exercise, you will *web-enable a spring application*
- This exercise uses the DayOfWeek service and the Catalog service to explore how to access various scoped beans
- Web framework integration (Struts, JSF, Spring MVC, etc.) will not be considered
- Read **instructions.html** in the *SpringLab02b* project and follow the additional steps provided

67

Notes:

Autowiring through XML

- There are four (4) possible values autowiring can have:
  - no
  - byName
  - byType
  - constructor

Example:

```
<bean id="catalog" autowire="byName" ... />
```

69

Notes:

The byName type of autowiring is presented on the next few slides. It allows Spring to "discover" the appropriate beans to inject by looking at other bean definitions in its configuration. The byType technique causes Spring to look for beans based on their type rather than their names. This technique is rather tricky because Spring can fail to inject a bean properly if two bean definitions exist with the same type. Constructor lets Spring determine how to autowire based on the format of the class' constructor.

Autowiring through XML ...

- The configuration for autowiring does not require bean references now

```
<beans ...>

    <bean id="catalogBean"
        class="com.company.springclass.beans.Catalog"
        autowire="byName"/>

    <bean id="productDAO"
        class="com.company.springclass.beans.ProductDao"
      />

</beans>
```

The bean is found by looking at other bean definitions that match the getXXX() setXXX() pattern within the class.

71

Notes:

Spring finds a bean definition with the ID of 'productDAO', which it matches to the setter of the Catalog class and injects it properly. If the setter had been called, setProductDao() instead of setProductDAO() this would have failed.

Design Consideration: Should you use autowiring?

- Arguments against autowiring:
  - Doesn't scale well
  - Larger projects can run into trouble when unexpected classes are loaded by the container's "guesswork"
- Arguments for autowiring:
  - Reduces XML configuration
  - It *can* scale well with proper cautions and naming conventions

73

Notes:

Using Annotations

- To use annotations, your XML config file must indicate this...

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframeork.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd"
>

    <context:annotation-config/>

        <bean id="catalogBean"
class="com.company.springclass.beans.Catalog"/>
        <bean id="productDAO"
class="com.company.springclass.beans.ProductDao" />

</beans>
```

Add the context namespace also

75

Notes:

The only major changes needed to use annotation-based configuration is to include the context xml namespace and to specify the <annotation-config> xml element.

Summary

- Spring's IOC container can inject beans with various scopes into other beans
- Spring supports several different scopes for beans, however, singleton is still the default
  - It also supports internationalization and placeholders in config files
- To web-enable a Spring application, be sure to configure the proper listeners

77

Notes: