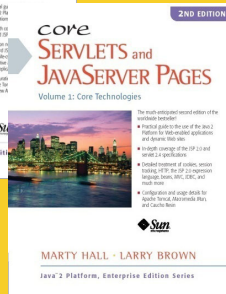
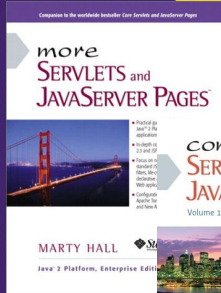




Using Spring with JSF 2.0

Originals of Slides and Source Code for Examples:
<http://www.coreservlets.com/JSF-Tutorial/jsf2/>

Customized Java EE Training: <http://courses.coreservlets.com/>
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



For live training on JSF 1.x or 2.0, please see courses at <http://courses.coreservlets.com/>.



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and 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 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF 1.x & 2.0, Ajax, GWT, custom mix of topics
 - Ajax courses can concentrate on 1 library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo, Google Closure) or survey several
 - Courses developed and taught by coreservlets.com experts (edited by Marty)
 - Spring, Hibernate/JPA, EJB3, Ruby/Rails, SOAP-based and RESTful Web Services
- Contact hall@coreservlets.com for details**

Topics in This Section

- **Fast review of Spring dependency injection**
- **Configuring a JSF/Spring project in Eclipse**
- **Defining beans in two places**
 - JSF backing beans in faces-config.xml
 - Spring beans in applicationContext.xml
- **Defining beans in one place only**
 - All beans in applicationContext.xml

5

© 2010 Marty Hall



Overview

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.

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

Review: What is Spring?

- **Spring is a framework with many modules**
 - To simplify many different Java EE tasks
- **Core module: dependency injection**
 - Lets you define an XML file that specifies which beans are created and what their properties are
 - Simplifies OOP by promoting loose coupling between classes. Also called “Inversion of Control” (IoC)
 - This is the module that we will use with JSF
 - *This tutorial assumes you are already familiar with Spring dependency injection in desktop Java apps*
- **Many other modules**
 - AOP, security, JDBC templates, etc.
 - Although most of these are not contradictory to the use of JSF, dependency injection is by far the most sought-after Spring feature for JSF apps

7

You are Using JSF Already: Why Use Spring?

- **Issue**
 - JSF already supports dependency injection via the managed-property element of faces-config.xml
 - So why use Spring to do what JSF already does?
- **Reasons**
 - You might already have beans set up with Spring from other applications
 - And you would like to use them without adding new code
 - Spring is more powerful
 - Spring supports more and better dependency injection features than does faces-config.xml.
 - But, before introducing Spring, make sure your JSF app will significantly benefit from these features

8

You are Using Spring Already: Why Use New Approach?

- **Issue**

- Normal Spring already supports ways to access beans.
 - So why add a new way for JSF?

- **Reasons**

- You need to access bean definitions from many places
 - With desktop Java apps, you can have a single piece of code that instantiates the container and gets beans
 - I.e., driver class that instantiates `ClassPathXmlApplicationContext` and calls `getBean`
 - With JSF apps, each controller wants access to beans
 - But you want to instantiate container once only
- You need additional bean scopes
 - Standard Spring supports singleton and prototype
 - JSF apps also want request, session, and application

9

© 2010 Marty Hall



Using Spring in JSF Apps

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.

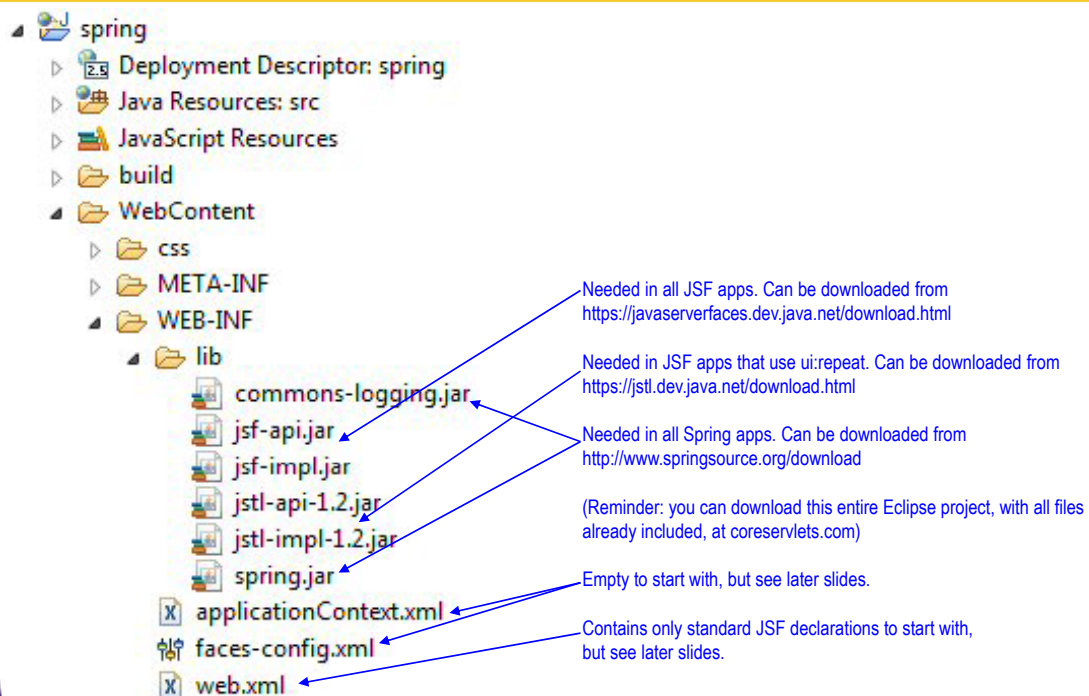
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Creating the App in Eclipse

- **Make a JSF 2.0 app in the normal manner**
 - File → New → Project → Web → Dynamic Web Project
 - Then copy jsf-api.jar, jsf-impl.jar, blank faces-config.xml, and the web.xml settings. Recommended: JSTL JAR files.
 - Or, copy/rename “jsf-blank” project
 - Both options described in “Getting Started” section
- **Add the same 3 files Spring usually uses**
 - spring.jar and commons-logging.jar
 - Put these in WEB-INF/lib along with JSF JAR files
 - Blank applicationContext.xml file
 - Put this in WEB-INF
- **If you are using Spring IDE Eclipse plugin**
 - R-click project, Spring Tools, Add Spring Project Nature
 - Adds smart support for editing applicationContext.xml

11

Eclipse Project Layout



12

Original Bean Definition File

- /WEB-INF/applicationContext.xml
 - If you want to change this default name/location, set a context param called contextConfigLocation to override it

```
<?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.xsd">

</beans>
```

13

Original web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ... version="2.5">
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
  </servlet-mapping>
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

14

These settings are described in tutorial section on JSF 2.0 setup.

Original faces-config.xml

```
<?xml version="1.0"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
  version="2.0">

</faces-config>
```

15

This file is introduced in tutorial section on JSF 2.0 setup.

Configuring the App for Spring: Defining Listeners in web.xml

- **ContextLoaderListener**
 - This listener runs when the app is first started. It instantiates the `ApplicationContext` (from `WEB-INF/applicationContext.xml`) and places a reference to it in the `ServletContext`
 - You can retrieve this reference with the static `getRequiredWebApplicationContext` method of `WebApplicationContextUtils`
- **RequestContextListener**
 - This listener is needed if you declare any of your beans to be request-scoped or session-scoped
 - I.e., Web scopes instead of the usual Spring scopes of singleton or prototype

16

web.xml Settings

```
...
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
<listener>
  <listener-class>
    org.springframework.web.context.request.RequestContextListener
  </listener-class>
</listener>

<!-- Normal JSF settings:
      - servlet/servlet-mapping for FacesServlet
      - PROJECT_STAGE settings
-->
...
```

17

Configuring JSF to Recognize Spring Beans: Idea

- **Not good to call getBean**
 - It would be technically legal to get the `ApplicationContext` and call `getBean` explicitly (probably from the backing bean's action controller method). But this is a bad idea since JSF is geared around declaring beans in config files only.
- **JSF already supports dependency injection**
 - The managed-property element lets you insert other beans inside newly created ones.
 - The only trick is to be able to refer to Spring beans
- **Use DelegatingVariableResolver**
 - Declare in `faces-config.xml`. Now, whenever JSF sees a bean name, it uses JSF rules first, then Spring rules next.

18

Configuring JSF to Recognize Spring Beans: faces-config.xml

```
<?xml version="1.0"?>
<faces-config ... version="2.0">
  <application>
    <variable-resolver>
      org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
  </application>
  ...
</faces-config>
```

19

© 2010 Marty Hall

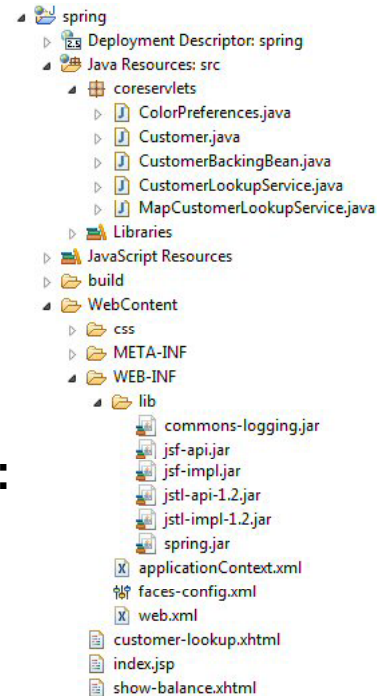


JSF/Spring Example 1: Beans in Two Config Files

Customized Java EE Training: <http://courses.coreservlets.com/>
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Example: Overview

- **Input form**
 - Collects user ID and password
 - Required
 - Collects preferred foreground and background colors
 - Optional
- **Results pages**
 - Shows name and balance of customer with given ID
 - Uses preferred colors
- **Form redisplayed with errors if:**
 - User ID or password missing
 - Uses required and requiredMessage
 - User ID is unknown
 - Sets FacesMessage in action controller



21

Example: Overview (Continued)

- **Service interface**
 - CustomerLookupService
 - Maps customer IDs to Customers
- **Service implementation**
 - MapCustomerLookupService
 - Uses fixed HashMap of a few sample customers
- **Color preferences object**
 - ColorPreferences
 - Stores the user's foreground and background colors
- **Backing bean (JSF managed bean)**
 - Accepts a customer ID and password
 - Needs the service implementation and color preferences object to be injected into it

22

Example: Overview (Continued)

- **applicationContext.xml**
 - Defines color preferences object and injects initial foreground and background colors into it
 - In session scope
 - Defines customer lookup service as a Map and injects all the keys and values into it
 - In singleton scope
- **faces-config.xml**
 - Declares the DelegatingVariableResolver
 - Creates the main backing bean in request scope and injects the two Spring beans (lookup service and color preferences object) into it.
 - But, a later example will move even this part to applicationContext.xml

23

web.xml: Defining Listeners

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <listener>
    <listener-class>
      org.springframework.web.context.request.RequestContextListener
    </listener-class>
  </listener>

  <!-- Normal JSF settings -->
</web-app>
```

Loads /WEB-INF/applicationContext.xml and puts reference to it in servlet context. Can be accessed with `WebApplicationContextUtils.getRequiredWebApplicationContext`

Lets you give request or session scopes to beans in applicationContext.xml. If you don't use these scopes, this listener is not required. But you should probably have this entry commented out in web.xml just in case you want those scopes later.

24

Color Preferences Bean

```
public class ColorPreferences implements Serializable {
    private String foreground, background;

    public String getForeground() {
        return(foreground);
    }
    public void setForeground(String foreground) {
        this.foreground = foreground;
    }
    // getBackground and setBackground

    public String getStyle() {
        String style =
            String.format("color: %s; background-color: %s",
                foreground, background);
        return(style);
    }
}
```

In Web apps in general, session data should be Serializable. This is partly to support distributed apps, but the more important reason is that Tomcat and other servers will let session data live across server restarts if the data is Serializable.

The foreground and background properties are tied to the textfields.

h:body does not support the "bgcolor" and "text" attributes that the regular body tag has. So, produce a CSS style string to supply to the "style" attribute of h:body.

25

Customer Lookup Service: Interface

```
public interface CustomerLookupService {

    public Customer findCustomer(String id);

    public Customer getRichestCustomer();
}
```

26

Customer Lookup Service: One Concrete Implementation

```
public class MapCustomerLookupService
    implements CustomerLookupService {
    private Map<String, Customer> sampleCustomers;

    public Map<String, Customer> getSampleCustomers() {
        return sampleCustomers;
    }
    public void setSampleCustomers(Map<String, Customer> sampleCustomers) {
        this.sampleCustomers = sampleCustomers;
    }

    public Customer findCustomer(String id) {
        if (id == null) {
            id = "unknown";
        }
        return (sampleCustomers.get(id.toLowerCase()));
    }

    public Customer getRichestCustomer() { ... }
}
```

This will be set via
<property name="sampleCustomers">
in applicationContext.xml

27

Customer Bean

```
public class Customer {
    private String customerID, firstName, lastName;
    private double balance;

    // Simple getters and setters

    public String getFormattedBalance() {
        return (String.format("$%,.2f", getBalance()));
    }
}
```

28

Backing Bean: Properties

```
public class CustomerBackingBean {  
    private String inputID, password;  
    private Customer customer;  
    private ColorPreferences colorPreferences;  
    private CustomerLookupService lookupService;  
  
    // Getters and setters  
    // for above 5 properties
```

Corresponding setters called by JSF when form submitted.

Filled in by action controller method (shown on next slide).

Setter methods called when bean created because of managed-bean-property in faces-config.xml. The incoming values are Spring beans.

29

Backing Bean: Action Controller Method

```
public String findBalance() {  
    customer = lookupService.findCustomer(inputID);  
    FacesContext context =  
        FacesContext.getCurrentInstance();  
    if (customer == null) {  
        String message =  
            String.format("Unknown ID '%s'", inputID);  
        context.addMessage("customerId", new FacesMessage(message));  
    }  
    if (!password.equals("secret")) {  
        String message = "Incorrect password";  
        context.addMessage("password", new FacesMessage(message));  
    }  
    if (context.getMessageList().size() > 0) {  
        return (null);  
    } else {  
        return ("show-balance");  
    }  
}
```

Since the form used prependId="false", this is just the id of the h:inputText element. Otherwise, you would put "formId:customerId" here.

30

faces-config: Variable Resolver

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE ...>

<faces-config>
  <application>
    <variable-resolver>
      org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
  </application>
</faces-config>
```

31

faces-config: Backing Bean

```
<managed-bean>
  <managed-bean-name>formBean</managed-bean-name>
  <managed-bean-class>
    coreservlets.CustomerBackingBean
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>lookupService</property-name>
    <value>#{sampleLookupService}</value>
  </managed-property>
  <managed-property>
    <property-name>colorPreferences</property-name>
    <value>#{colorPreferences}</value>
  </managed-property>
</managed-bean>
```

Gets the Spring bean called sampleLookupService and passes it to the setLookupService method of the JSF backing bean called formBean (i.e., injects it into the lookupService property).

Gets the Spring bean called colorPreferences and injects it into the colorPreferences property of the backing bean.

32

applicationContext.xml: Defining Color Preferences

```
<?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.xsd">

  <bean id="colorPreferences" class="coreservlets.ColorPreferences"
        scope="session">
    <property name="foreground" value="black"/>
    <property name="background" value="#fdf5e6"/>
  </bean>
```

33

applicationContext.xml: Defining Lookup Service

```
<bean id="sampleLookupService"
      class="coreservlets.MapCustomerLookupService">
  <property name="sampleCustomers">
    <map>
      <entry key="a1234">
        <bean class="coreservlets.Customer">
          <property name="customerID" value="a1234"/>
          <property name="firstName" value="Rod"/>
          <property name="lastName" value="Johnson"/>
          <property name="balance" value="123.45"/>
        </bean>
      </entry>
      ...
    </map>
  </property>
</bean>
</beans>
```

34

Input Form: Top (customer-lookup.xhtml)

```
...  
<h:body style="{formBean.colorPreferences.style}">  
...  
<h:form prependId="false">  
  <label>  
    <font color="red">*</font> Customer ID:  
    <h:inputText value="{formBean.inputID}"  
      required="true"  
      requiredMessage="Missing customer ID"  
      id="customerId"/>  
    <h:message for="customerId" styleClass="error"/>  
  </label><br/>
```

Using prependId makes it easier to refer to the input element ID in the action controller method when setting a custom error message.

If no user ID is entered, the form is redisplayed and "Missing customer ID" is shown. If an unknown ID is entered, the action controller method sets a custom FacesMessage and returns null so that the form is redisplayed and "Unknown ID" is shown. If a recognized ID is entered, will navigate to page showing name and balance (using preferred colors).

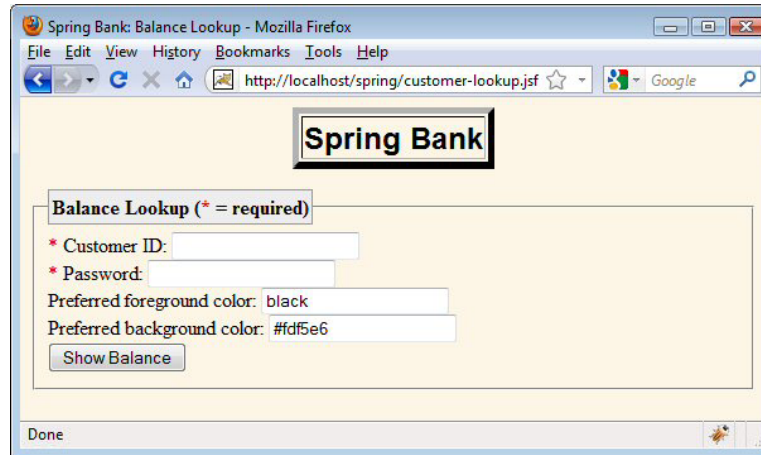
35

Input Form: Bottom (customer-lookup.xhtml)

```
<label>  
  <font color="red">*</font> Password:  
  <h:inputSecret value="{formBean.password}"  
    required="true"  
    requiredMessage="Missing password"  
    id="password"/>  
  <h:message for="password" styleClass="error"/>  
</label><br/>  
<label>  
  Preferred foreground color:  
  <h:inputText value="{formBean.colorPreferences.foreground}"/>  
</label><br/>  
<label>  
  Preferred background color:  
  <h:inputText value="{formBean.colorPreferences.background}"/>  
</label><br/>  
<h:commandButton value="Show Balance"  
  action="{formBean.findBalance}"/>  
</h:form> ...
```

36

Input Form (Initial Result)



The screenshot shows a Mozilla Firefox browser window titled "Spring Bank: Balance Lookup". The address bar contains "http://localhost/spring/customer-lookup.jsf". The page content includes a "Spring Bank" logo, a "Balance Lookup (* = required)" section with input fields for "Customer ID", "Password", "Preferred foreground color" (set to "black"), and "Preferred background color" (set to "#fd5e6"). A "Show Balance" button is located below the form. The status bar at the bottom indicates "Done".

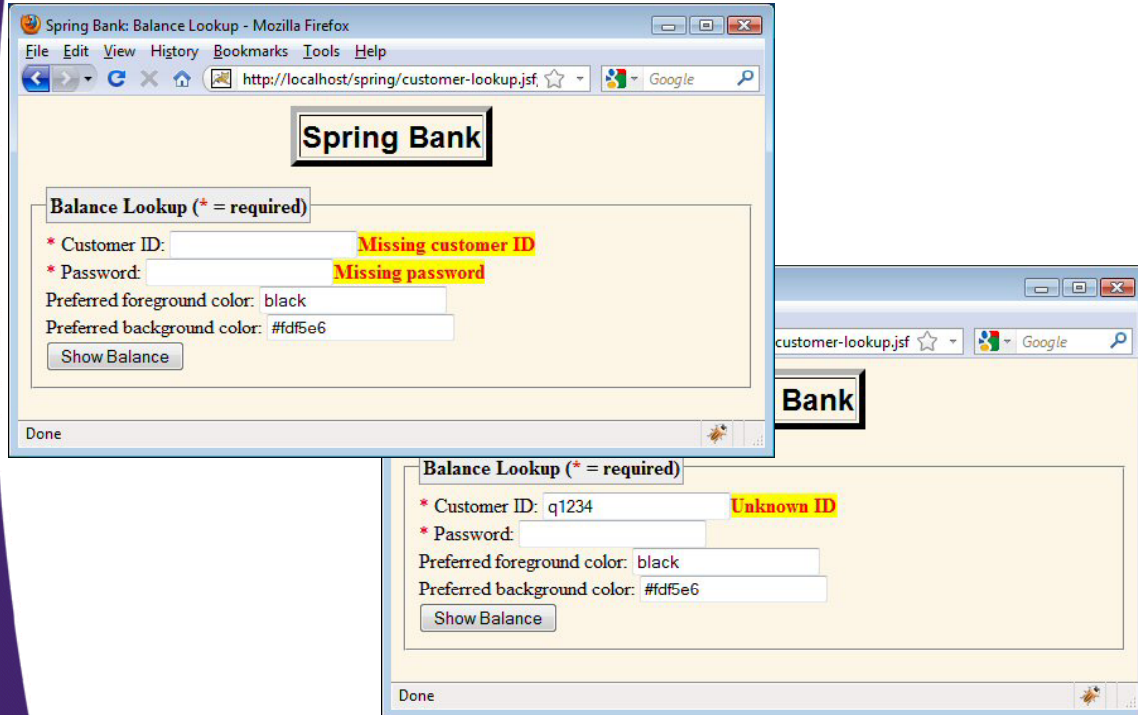
37

Results Page (show-balance.xhtml)

```
...
<h:body style="{formBean.colorPreferences.style}">
<table border="5" align="center">
  <tr><th class="title">Spring Bank: Your Balance</th></tr>
</table>
<p/>
<ul>
  <li>ID: #{formBean.customer.customerID}</li>
  <li>First name: #{formBean.customer.firstName}</li>
  <li>Last name: #{formBean.customer.lastName}</li>
  <li>Balance: #{formBean.customer.formattedBalance}</li>
</ul>
</h:body></html>
```

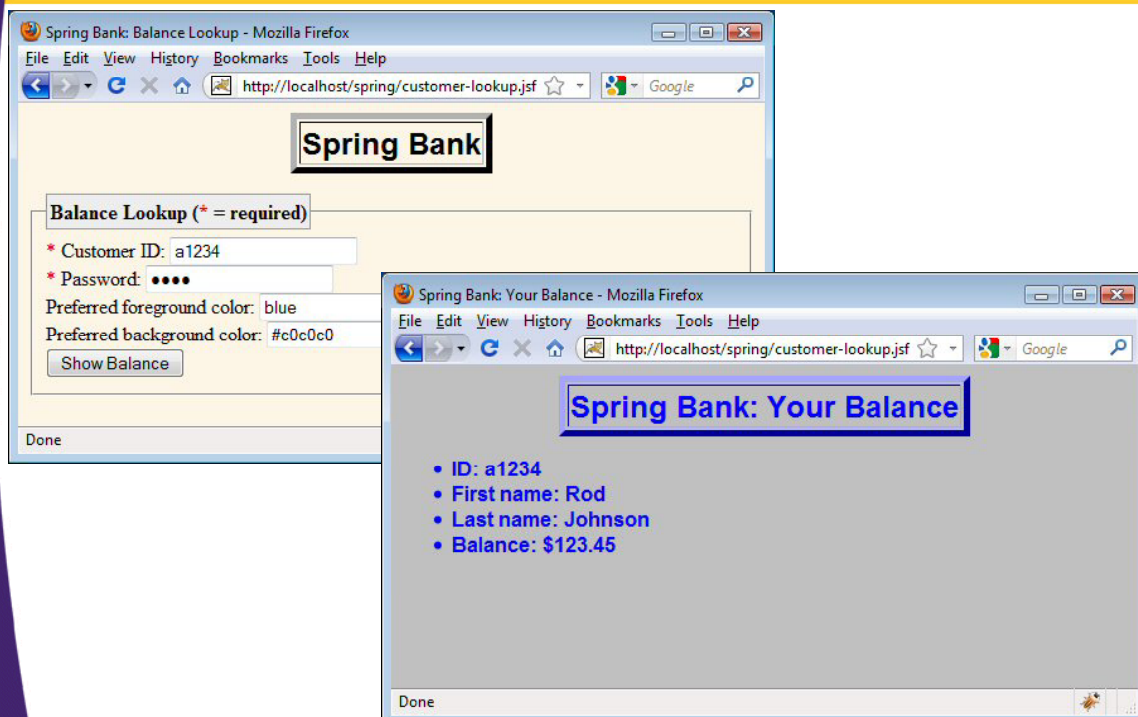
38

Results: Bad Data



39

Results: Good Data



40



JSF/Spring Example 2: Beans in One Config File

Customized Java EE Training: <http://courses.coreservlets.com/>
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Overview

- **Issue with previous example**
 - Beans defined in two different files
 - Some using Spring syntax, some using JSF syntax
- **Approach**
 - applicationContext.xml
 - Defines the Spring beans as before
 - Defines session-scoped ColorPreferences and singleton-scoped CustomerLookupService
 - Also defines the backing bean
 - faces-config.xml
 - Defines only non-beans entries
 - Variable resolver, navigation rules, properties files, etc.
 - Functionality and appearance
 - Exactly the same as in previous app

Changes from Previous Example

- **faces-config.xml**
 - Deleted the entire `<managed-bean>` entry
- **applicationContext.xml**
 - Added the following simpler entry

```
<bean id="formBean"
      class="coreservlets.CustomerBackingBean"
      scope="request">
  <property name="lookupService" ref="sampleLookupService"/>
  <property name="colorPreferences" ref="colorPreferences"/>
</bean>
```
 - Two advantages
 - Dependency injection syntax is simpler and more powerful
 - All bean definitions in the same file

43

faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE ...>

<faces-config>
  <application>
    <variable-resolver>
      org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
  </application>
</faces-config>
```

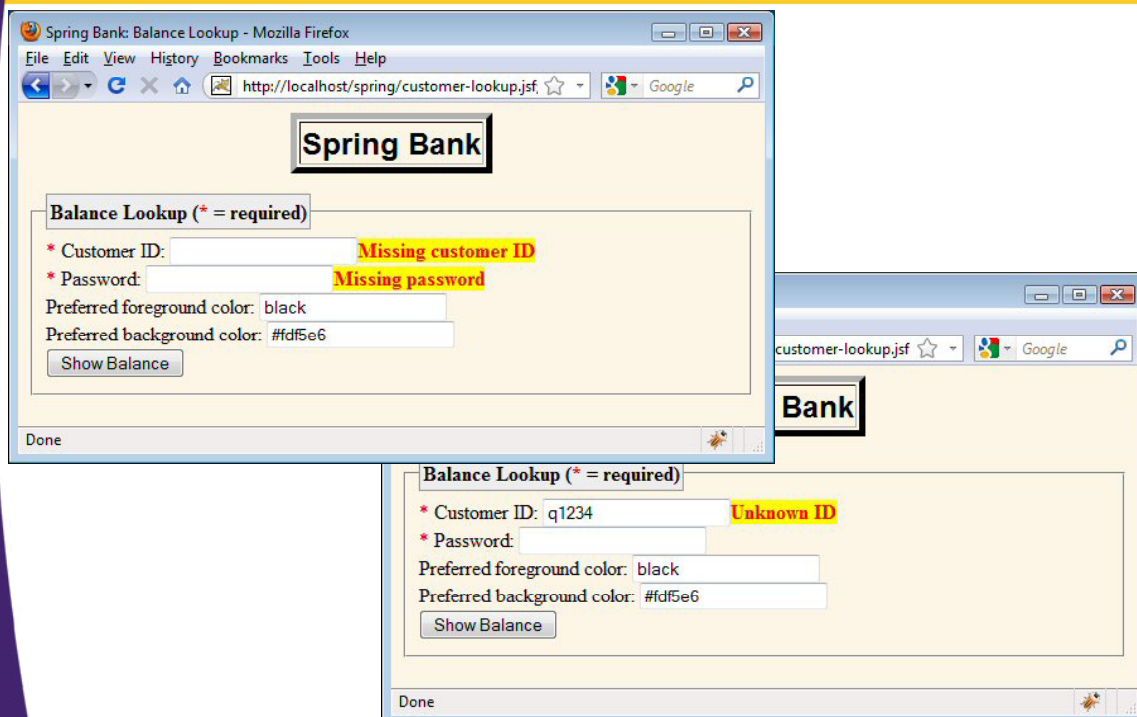
44

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>
  <bean id="colorPreferences" ...>
    <!-- Unchanged from last example -->
  </bean>
  <bean id="sampleLookupService" ...>
    <!-- Unchanged from last example -->
  </bean>
  <bean id="formBean"
        class="coreservlets.CustomerBackingBean"
        scope="request">
    <property name="lookupService" ref="sampleLookupService"/>
    <property name="colorPreferences" ref="colorPreferences"/>
  </bean>
</beans>
```

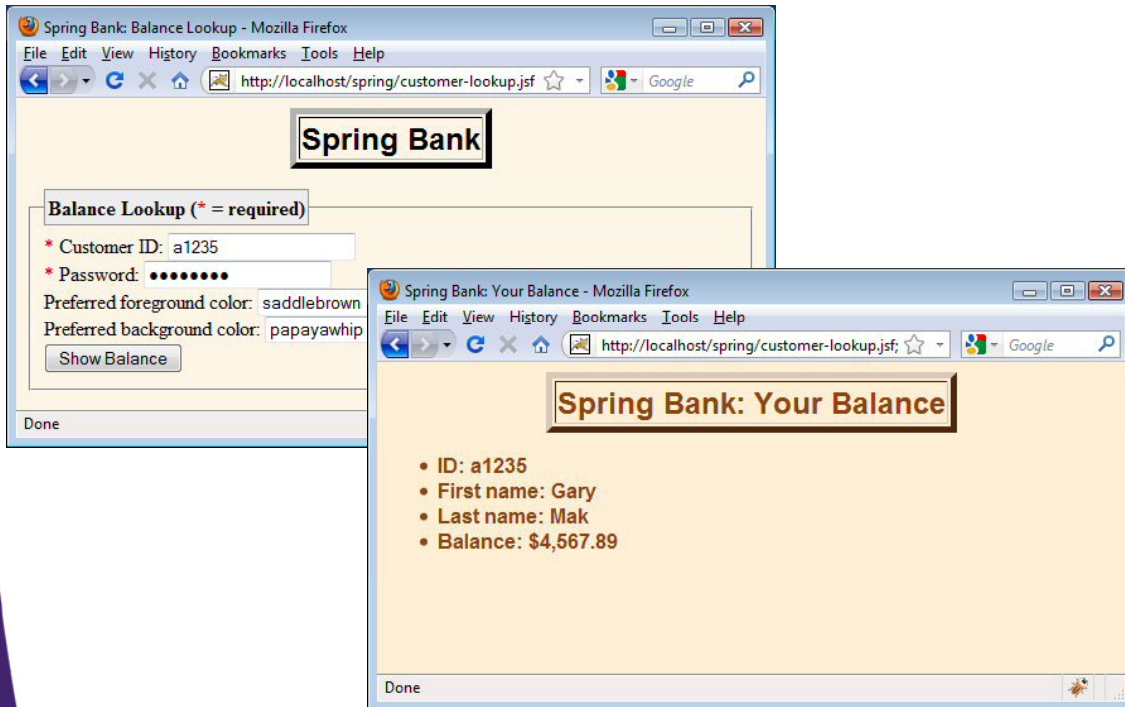
45

Results: Bad Data



46

Results: Good Data



47

© 2010 Marty Hall



Wrap-up

Customized Java EE Training: <http://courses.coreservlets.com/>
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Summary

- **Basic setup**
 - Start with same setup as regular JSF apps
 - Add Spring JAR files and applicationContext.xml
- **faces-config.xml**
 - Declare DelegatingVariableResolver
- **Option 1**
 - Declare Spring beans in applicationContext.xml
 - Declare backing beans in faces-config.xml
 - Refer to Spring beans with managed-bean-property
- **Option 2**
 - Declare all beans in applicationContext.xml
 - Refer to other beans with ref and normal Spring syntax

49

© 2010 Marty Hall



Questions?

Customized Java EE Training: <http://courses.coreservlets.com/>

Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.

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