



# JSF Programming: A Whirlwind Tour

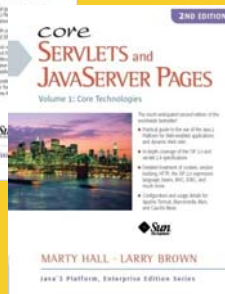
JSF 2.2 Version

Originals of slides and source code for examples: <http://www.coreservlets.com/JSF-Tutorial/jsf2/>

Also see the PrimeFaces tutorial – <http://www.coreservlets.com/JSF-Tutorial/primefaces/>  
and customized JSF2 and PrimeFaces training courses – <http://courses.coreservlets.com/jsf-training.html>

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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live training on JSF 2, PrimeFaces, or other  
Java EE topics, email [hall@coreservlets.com](mailto:hall@coreservlets.com)**  
Marty is also available for consulting and development support

Taught by the author of *Core Servlets and JSP*, this tutorial, and JSF 2.2 version of *Core JSF*. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, Ajax, jQuery, Spring MVC, JSP, Android, general Java, Java 8 lambdas/streams, GWT, custom topic mix
  - Courses available in any location worldwide. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Hadoop, Spring, Hibernate/JPA, RESTful Web Services

Contact [hall@coreservlets.com](mailto:hall@coreservlets.com) for details



## Topics in This Section

- Setting up your IDE and project for JSF 2.2
- Testing tags
- Navigating from page to page
- Representing form data
- Applying business logic
- Using page templates and include files
- Validating input data
- Ajaxifying pages to avoid page reloads
- Using PrimeFaces to make things prettier

4

## Whirlwind Tour

- **Idea**
  - This section gives a quick example of the most basic JSF 2.2 capabilities. **This tutorial section gives only brief examples with very limited explanation.**
- **Details**
  - Each of the topics is covered in detail in later sections.
- **Other topics**
  - The topics given here are only the most basic ones: the full tutorial covers many other important but not-quite-as-fundamental topics such as looping with ui:repeat, data tables, composite components, property files, I18N, GUI events, view parameters, and so on.

5



# Setting Up Eclipse and Your Project



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Overview

- **Big idea**
  - Use Eclipse to make a dynamic Web project that supports JSF 2.2.
- **Details**
  - Use Eclipse wizard to make Dynamic Web Project that has proper JSF 2.2 JAR file, web.xml, and faces-config.xml.
- **More info**
  - See the “Installation, Setup, and Getting Started” section of JSF tutorial at <http://www.coreservlets.com/JSF-Tutorial/jsf2/>.
    - That section also covers an alternative approach in which you copy and rename the jsf-blank project, then work around an Eclipse project-renaming bug.

# One Time Only: Register JSF 2.2 JAR

- **Download JSF 2.2 JAR**

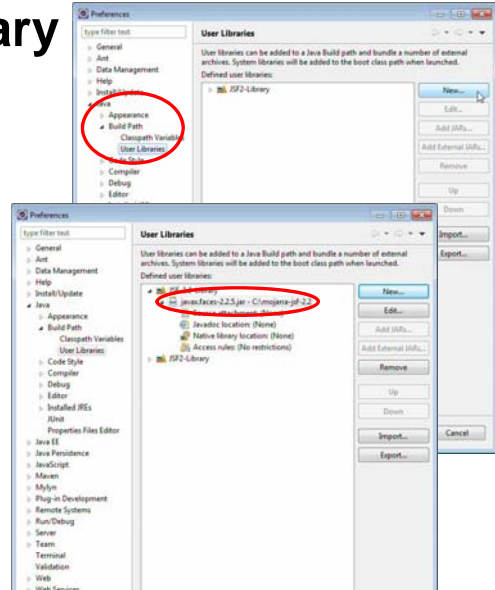
- <https://javaserverfaces.java.net/download.html>

- **Create JSF 2.2 user library**

- Window → Preferences →  
Java → Build Path →  
User Libraries → New

- **Select new library and point at JSF 2.2 JAR file**

- Click on your new library
  - Click “Add External JARs”
  - Point to JAR file from  
wherever you saved it from  
previous section

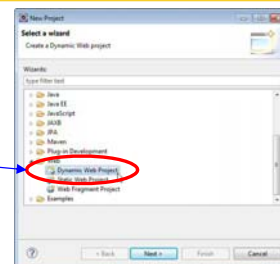


8

# Each Time: Make Dynamic Web Project with JSF 2.2

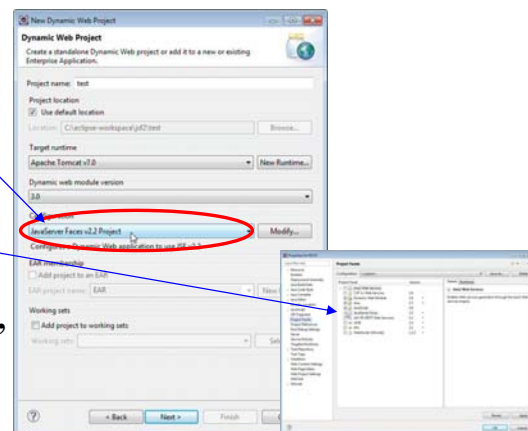
- **Create project**

- File → New → Project → Web  
→ Dynamic Web Project
  - Next time, you can do  
File → New → Dynamic Web Project



- **Specify JSF 2.2 for configuration**

- You can also create  
vanilla Dynamic Web Project, then add JSF  
2.2 later by R-clicking  
project, selecting Properties,  
and going to Project Facets

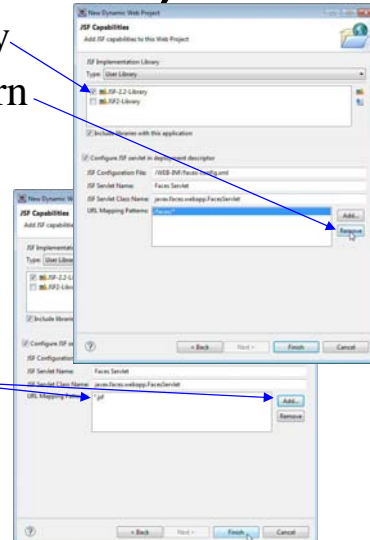


9



## Make Dynamic Web Project with JSF 2.2 (Continued)

- **Use defaults for src location and web.xml**
  - (1<sup>st</sup> and 2<sup>nd</sup> screens after last slide)
- **For “JSF Capabilities” (3<sup>rd</sup> screen)**
  - Add your new JSF 2.2 user library
  - Remove \*.faces as the URL pattern



- Add \*.jsf as the URL pattern

10

## Make Dynamic Web Project with JSF 2.2 (Continued)

- **Clean up auto-generated web.xml**
  - The one made by Eclipse has some spurious entries (localizationContext and ConfigureListener), lacks the very valuable PROJECT\_STAGE setting, and fails to set any welcome-page URLs
- **Solution: copy web.xml from jsf-blank**
  - In principle, you could also edit by hand, but this is tedious and error prone.
    - Format of web.xml entries is discussed in “Programming Basics” tutorial section
    - You can download the more-standard web.xml from “Getting Started” section of JSF 2.2 tutorial at coreservlets.com, but easier to download all of jsf-blank, then copy the web.xml from there.

– <http://www.coreservlets.com/JSF-Tutorial/jsf2/code/jsf-blank.zip>

11



# Testing Tags



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Overview

- **Big idea**
  - Verify that the JSF tags get rendered as HTML elements. This shows that the project was set up properly:
    - In particular, it shows that project has the JSF JAR file in the right place, and the right main entries in web.xml.
- **Details**
  - Use the standard format for JSF .xhtml pages (“facelets”)
    - Note that the physical file is blah.xhtml, but the URL you use is blah.jsf.
  - Check that h:commandButton gets rendered as a push button
  - Check that h:inputText gets rendered as a textfield

# Basic JSF Page Format

```
<!DOCTYPE html ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Your Title Here</title>
    <link href="./css/your-styles.css"
          rel="stylesheet" type="text/css"/>
  </h:head>
  <h:body>
    <h1>Your Heading Here</h1>
    <h:form>
      <!-- Your form elements here -->
    </h:form>
  </h:body></html>
```

For the XML namespace, earlier JSF versions used java.sun.com instead of xmlns.jcp.org, and the older namespace is still supported for backward compatibility.

14

## test-tags.xhtml

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Test Tags</title>
    <link href="./css/styles.css"
          rel="stylesheet" type="text/css"/>
  </h:head>
  <h:body>
    <h1 class="title">Test Tags</h1>

    <fieldset>
      <legend>Test that JSF Tags Render Properly</legend>
      <h:form>
        Verify that you see a textfield and button below.<br/><br/>
        Textfield: <h:inputText/><br/>
        Button: <h:commandButton value="Click Me (But Nothing Will Happen)"/>
      </h:form>
    </fieldset>
  </h:body></html>
```

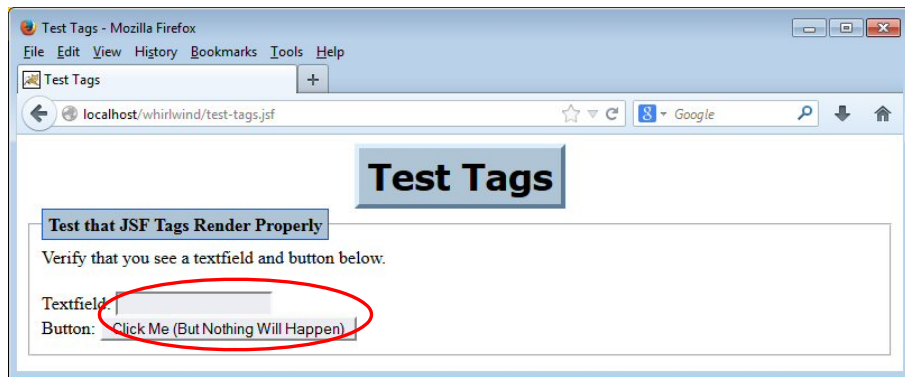
15

# Testing

- **Deployed to Tomcat**

- R-clicked Tomcat, Add And Remove Projects, selected whirlwind, Finish. Then R-clicked Tomcat and Start. Then accessed <http://localhost/whirlwind/>
  - See installation and setup section for details on installing Tomcat and deploying projects

- **Result**



16

© 2015 Marty Hall



# Navigating



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



# Overview

- **Big idea**

- Use a pushbutton to go to a new page

- **Details**

- Declare Java class with @ManagedBean
- Use the action attribute of h:commandButton to specify a method that takes no arguments and returns a String
- The String that is returned is the base name of the page that will be displayed to the user. I.e., if the method returns "foo", the user is forwarded to foo.xhtml.

- **More info**

- Details in programming basics section. Other ways to do navigation are discussed in section on explicit page navigation and faces-config.xml.

18

# Input Form (navigate.xhtml)

```
<!DOCTYPE html ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
...
</h:head>
<h:body>
<h1 class="title">Navigate via Java</h1>
<fieldset>
<legend>Use Java to Navigate to Results Page</legend>
<h:form>
  Press button to get one of two possible results pages.<br/>
  <h:commandButton value="Go to Random Page"
                    action="#{navigator.choosePage}" />
</h:form>
</fieldset>
</h:body></html>
```

Means that the corresponding Java class (next slide) has a method called choosePage that takes zero arguments and returns a String. The String returned is the base name of the page that the user will be forwarded to.

19

# Java Code (Navigator.java)

```
package coreservlets;
```

```
import javax.faces.bean.*;
```

Means that you refer to the Java class in the facelets (.xhtml) file as "navigator" – the class name with the first letter changed to lower case. Other ways of naming beans are discussed in the sections on managed beans and on faces-config.xml.

```
@ManagedBean
```

```
public class Navigator {
```

```
    public String choosePage() {
```

```
        if (Math.random() > 0.5) {
```

```
            return("result-page-1");
```

```
        } else {
```

```
            return("result-page-2");
```

```
        }
```

```
    }
```

```
}
```

Means that the user will be forwarded to either result-page-1.xhtml or result-page-2.xhtml. Other ways of navigating are discussed in the section on faces-config.xml.

20

# First Results Page (result-page-1.xhtml)

```
<!DOCTYPE html ...>
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
```

```
      xmlns:h="http://xmlns.jcp.org/jsf/html">
```

```
<h:head><title>Result Page 1</title>
```

```
<link href="./css/styles.css"
```

```
      rel="stylesheet" type="text/css"/>
```

```
</h:head>
```

```
<h:body>
```

```
<h1 class="title">Result Page 1</h1>
```

```
<h2>One. Uno. Isa.</h2>
```

```
<p>Blah, blah, blah.</p>
```

```
</h:body></html>
```

21

## Second Results Page (result-page-2.xhtml)

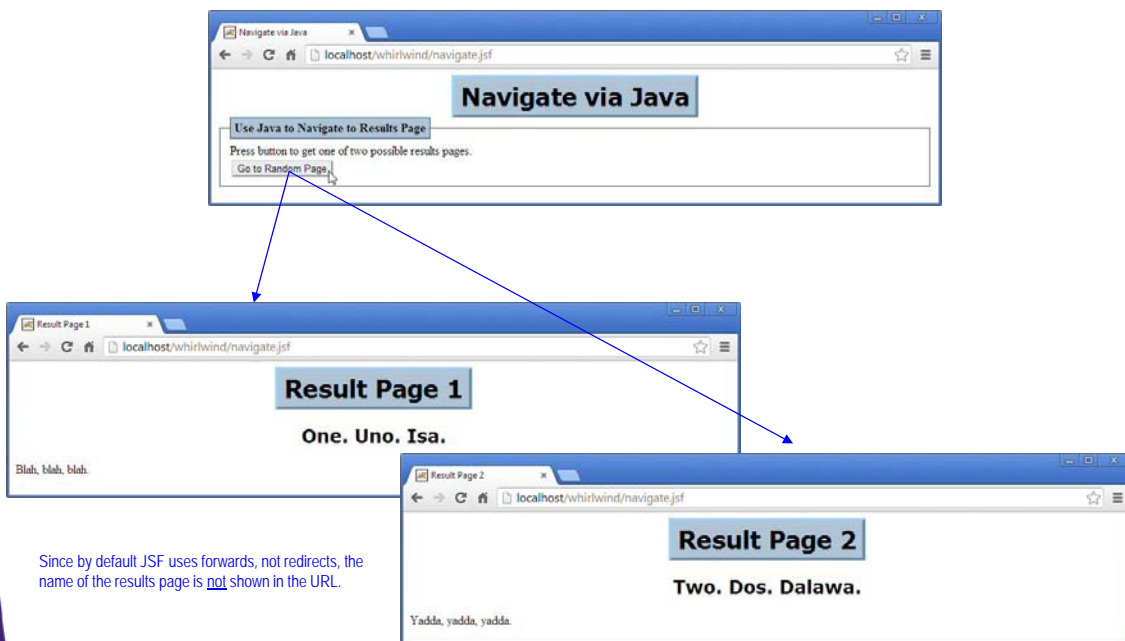
```
<!DOCTYPE html ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head><title>Result Page 2</title>
<link href="./css/styles.css"
      rel="stylesheet" type="text/css"/>
</h:head>
<h:body>

<h1 class="title">Result Page 2</h1>
<h2>Two. Dos. Dalawa.</h2>
<p>Yadda, yadda, yadda.</p>

</h:body></html>
```

22

## Results



23



# Representing Form Data with Bean Properties



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Overview

- **Big idea**
  - Use bean properties to represent the input elements.
- **Details**
  - If you have `getFoo` and `setFoo` methods, a textfield would use `<h:inputText value="#{beanName.foo}"/>`.
    - The getter (`getFoo`) is called when form is displayed, and if result is non-empty, that is initial value of textfield.
    - When form is submitted, the textfield value is passed to the setter method (`setFoo`).
- **More info**
  - Details on bean properties in Managed Beans 1 section. Managed Beans 2 talks about the lifecycle of the beans, and various scope options (beans reinstantiated for each request, beans live on for the session, etc.)

# Input Form (bean-properties.xhtml)

```
...
<h:form>
<h:panelGrid columns="2" styleClass="formTable">
  First Name:
  <h:inputText value="#{registrationBean.firstName}"/>

  Last Name:
  <h:inputText value="#{registrationBean.lastName}"/>

  Email Address:
  <h:inputText value="#{registrationBean.emailAddress}"/>
</h:panelGrid>
<h:commandButton value="Register"
                  action="#{registrationBean.doRegistration}"/>
</h:form>
...
```

Notice that textfields refer to bean properties ("firstName" above is shortcut for both `getFirstName` and `setFirstName`), whereas pushbuttons refer to the exact method name (`doRegistration` is the action controller method).

`h:panelGrid` is a shorthand way of making an HTML `<table>`, and is used here to make sure that the prompt and the textfield are always on the same line.

26

# Java Code (Managed Bean) Part 1: Bean Properties

```
...
@ManagedBean
public class RegistrationBean {
  private String firstName, lastName, emailAddress;

  public String getFirstName() {
    return(firstName);
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }
}
```

Similar getters and setters for `lastName` and `emailAddress`.

What matters is the name of the methods, not the name of the instance variables.

27



## Managed Bean Part 2: Action Controller Method

```
public String doRegistration() {  
    if (FormUtils.isAnyMissing(firstName, lastName,  
                                emailAddress)) {  
        return("registration-error");  
    } else {  
        return("registration-success");  
    }  
}
```

This is the action controller method; the method referred to by the "action" attribute of `h:commandButton`. As before, we are using implicit navigation (where the Strings returned are the base names of the results pages), so the results pages are `registration-error.xhtml` and `registration-success.xhtml`. Other ways of doing navigation are discussed in the section on explicit navigation and `faces-config.xml`.

28

## First Results Page (registration-error.xhtml)

```
<!DOCTYPE html ...>  
<html xmlns="http://www.w3.org/1999/xhtml"  
      xmlns:h="http://xmlns.jcp.org/jsf/html">  
  <h:head>  
    <title>Registration Error</title>  
    <link href="./css/styles.css"  
          rel="stylesheet" type="text/css"/>  
  </h:head>  
  <h:body>  
    <h1 class="title">Registration Error</h1>  
  
    <h2>You must supply all of first name, last name, and  
    email address. Please <a href="bean-properties.jsf">try  
    again</a>.</h2>  
  
    ...  
  </h:body></html>
```

Note: using a separate error page for missing data is not a good approach. Instead, you should redisplay the input form and mark missing data with error messages. That better approach is shown briefly in an upcoming example and then discussed in detail in the tutorial section on validation.

29

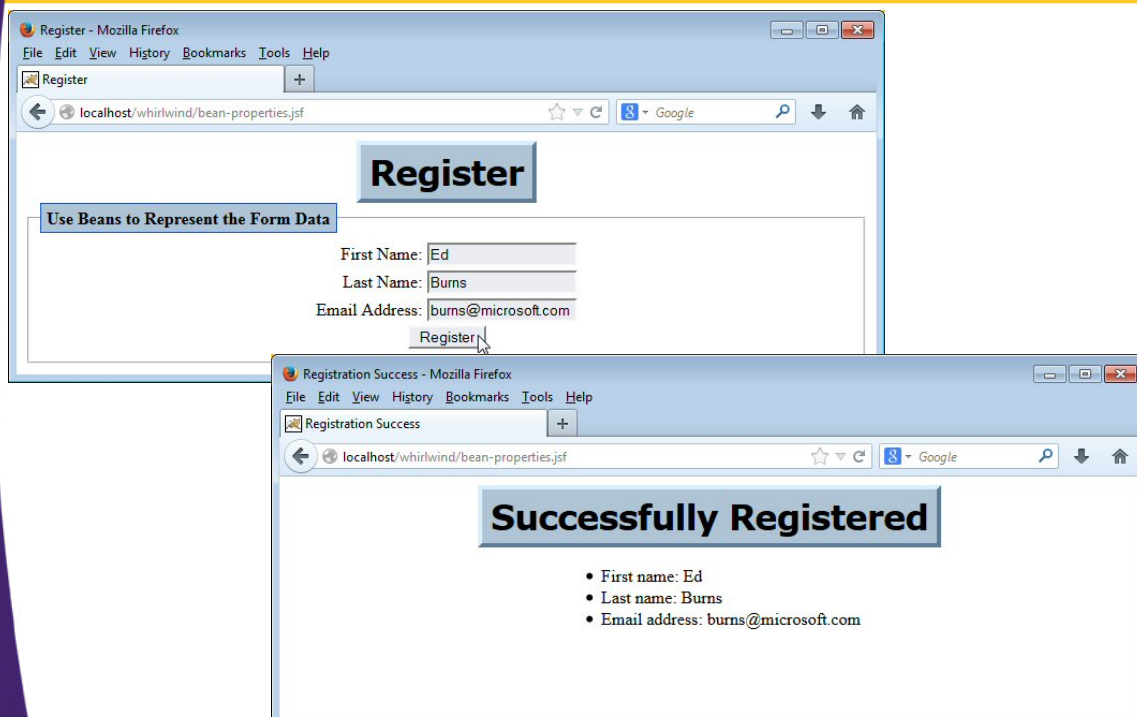
## Second Results Page (registration-success.xhtml)

```
<!DOCTYPE html ...>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Registration Success</title>
    <link href="./css/styles.css"
          rel="stylesheet" type="text/css"/>
  </h:head>
  <h:body>
    <div align="center">
      <h1 class="title">Successfully Registered</h1>
      <ul class="aligned">
        <li>First name: #{registrationBean.firstName}</li>
        <li>Last name: #{registrationBean.lastName}</li>
        <li>Email address: #{registrationBean.emailAddress}</li>
      </ul>
    </div>
  </h:body></html>
```

#{registrationBean.firstName} means to call  
getFirstName() and output the result.

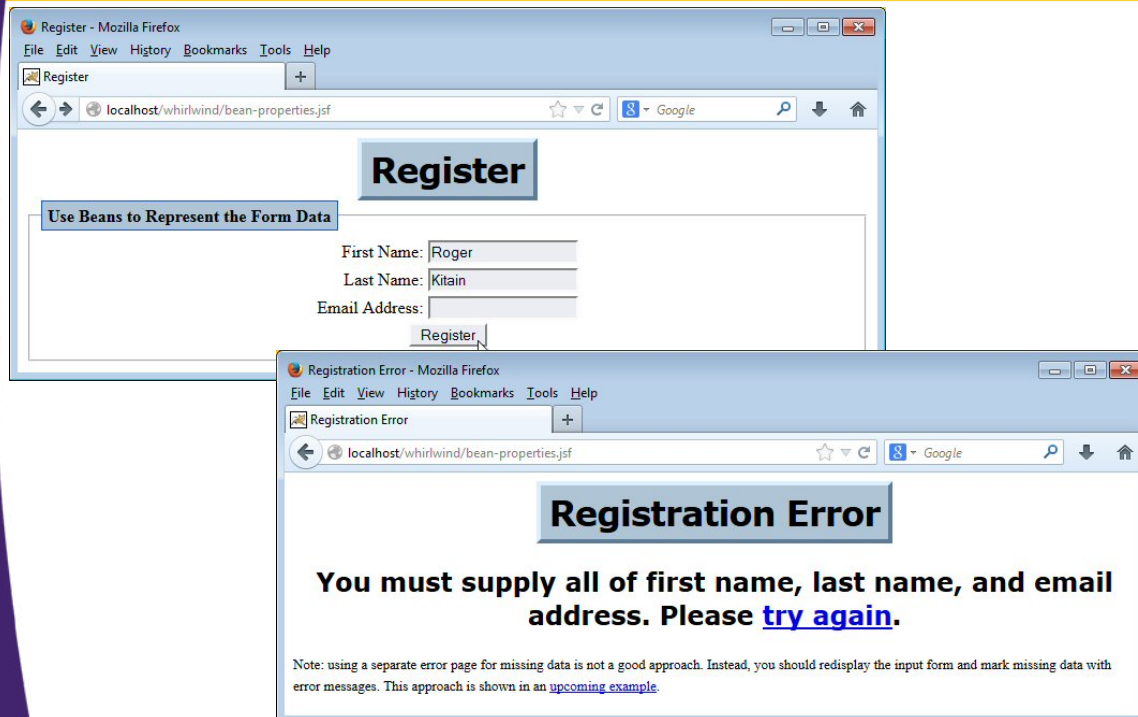
30

## Results (Good Input)



31

# Results (Missing Input)



32

© 2015 [Marty Hall](#)



## Applying Business Logic



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

# Overview

- **Big idea**

- The results page usually shows more than just the data the user entered: it normally also shows data *derived* from the user data
  - E.g., the user enters a bank account number and you display the account balance.

- **Details**

- The goal is to isolate the code that looks up the derived data, so that changes to the way the data is calculated do not require changes in the rest of the code. See next slide for various approaches. The first two (separate methods and simple types) should *always* be done.

- **More info**

- See Managed Beans 1 for the basics, and Managed Beans 2 for an update that uses dependency injection.

34

# Approaches to Business Logic: Summary

- **Use a separate method (do always)**

- Do not compute the derived data directly in the action controller method, but use a separate method.

- **Simple types in, simple types out (do always)**

- Never return a ResultSet or Hibernate object or anything specific to *how* you found the data. Return a Java object representing the result itself.

- **Code to interfaces (do usually)**

- Make an interface such as CustomerLookupService and use that type. Prevents accidental dependence on concrete type.

- **Use dependency injection (do sometimes)**

- Inject the concrete type so nothing in main class changes when you change concrete implementations of the interface.
  - Dependency injection not covered here; see Managed Beans 2

35

# Input Form (business-logic.xhtml)

```
...
<h1 class="title">Today's Daily Deal</h1>
<h2>Today's daily book deal is
<i>#{dealBean1.todaysDeal.title}</i> at a rock-bottom
price of $#{dealBean1.todaysDeal.price}.</h2>
...
<h:form>
<h:panelGrid columns="2" styleClass="formTable">
    Account ID:
    <h:inputText value="#{dealBean1.accountId}"/>

    Number to Purchase:
    <h:inputText value="#{dealBean1.countString}"/>
</h:panelGrid>
<h:commandButton value="Purchase"
                  action="#{dealBean1.buyDailyDeal}"/>
</h:form>
...
```

36

# Managed Bean Part 1: Bean Properties

```
@ManagedBean
public class DealBean1 {
    private String accountId, countString;
    private int count;
    ...
    public String getCountString() {
        return(countString);
    }

    public void setCountString(String countString) {
        try {
            count = Math.abs(Integer.parseInt(countString));
            this.countString = countString;
        } catch(NumberFormatException|NullPointerException e) {
            // Keep default values (count=0, countString=null)
        }
    }

    public int getCount() {
        return (count);
    }
}
```

setCountString converts from String to int manually.  
Later example will let JSF do the conversion for us.

There is also getAccountId and setAccountId.

37



## Managed Bean Part 2: Placeholder for Results Data

```
private Account account;

private static AccountLookupService lookupService =
    new AccountSimpleMap();

public Account getAccount() {
    return(account);
}
```

The account is not supplied by the user. Instead, the user supplies an ID, and the action controller method (next slide) will call the business logic (`lookupService.findAccount()`) and use the result to fill in the account. Also note that, since account is not linked to a textfield, there is no requirement that you have a setter method. But, since you will output data in the results page (e.g., `#{dealBean1.account.fullName}`), you need to have a getter method (i.e., `#{dealBean1.account...}` means to call `getAccount()`).

38

## Managed Bean Part 3: Action Controller Method

```
public String buyDailyDeal() {
    if (FormUtils.isAnyMissing(accountId, countString)) {
        return("deal-error-1");
    }
    account = lookupService.findAccount(accountId);
    if (account == null) {
        return("deal-bad-id-1");
    }
    if (account.getBalance() < getPurchaseAmount()) {
        return("deal-insufficient-balance-1");
    }
    account.setBalance(account.getBalance() -
        getPurchaseAmount());
    return("deal-success-1");
}
```

The action controller method calls the business logic and then navigates to various pages depending on the result.

39

# Business Logic: Interface

```
public interface AccountLookupService {  
    public Account findAccount(String id);  
}
```

The managed bean declares the variable to be the interface type, not the concrete type. This means that when you change implementations of the concrete type, you do not have to change anything in the action controller method or the output pages. Some people go even further and use dependency injection to set the concrete type from a bean, so changing the concrete type means changing a separate bean, but not changing anything in the main managed bean. Dependency injection is shown in the Managed Beans 2 section.

40

# Business Logic: Implementation

```
public class AccountSimpleMap  
    implements AccountLookupService {  
    private Map<String,Account> accounts;  
  
    public AccountSimpleMap() {  
        accounts = new HashMap<>();  
        addAccount(new Account("a1234", "Bill", "Gates", 1000));  
        ...  
    }  
  
    @Override  
    public Account findAccount(String id) {  
        if (id != null) {  
            return(accounts.get(id.toLowerCase()));  
        } else {  
            return(null);  
        }  
    }  
}
```

By using the interface type in the code and having findAccount return an ordinary Java object (Account) that is not tied to the way in which accounts are found, you can use this simple test data and then later change to JDBC, then to Hibernate, then to a Web Service, with no changes needed in the action controller method or the results pages.

41

## Main Results Page (deal-success-1.xhtml)

```
...
<h1 class="title">Success: Daily Deal Ordered</h1>
<ul class="aligned">
  <li>Account ID: #{dealBean1.accountId}</li>
  <li>Name: #{dealBean1.account.fullName}</li>
  <li>Book ordered:
    <i>#{dealBean1.todaysDeal.title}</i></li>
  <li>Number purchased: #{dealBean1.count}</li>
  <li>Balance after purchase:
    #{dealBean1.account.balanceDollars}</li>
</ul>
...
```

This page outputs not just values that the user entered (accountId, count), but also values derived from them (account.fullName, account.balanceDollars).

#{dealBean1.account.fullName} means to first call getAccount() and then to call getFullName() on the result.

## Page for Low Balance (deal-insufficient-balance-1.xhtml)

```
...
<h1 class="title">Insufficient Balance</h1>

<h2>Balance for #{dealBean1.account.fullName}
  (#{dealBean1.account.balanceDollars}) is
  too low. #{dealBean1.purchaseDollars} needed.</h2>

<h3>Please <a href="business-logic.jsf">
  try again</a>.</h3>
...
```

# Page for Missing Data (deal-error-1.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
<title>Error</title>
<link href="./css/styles.css"
      rel="stylesheet" type="text/css"/>
</h:head>
<h:body>
<h1 class="title">Error</h1>

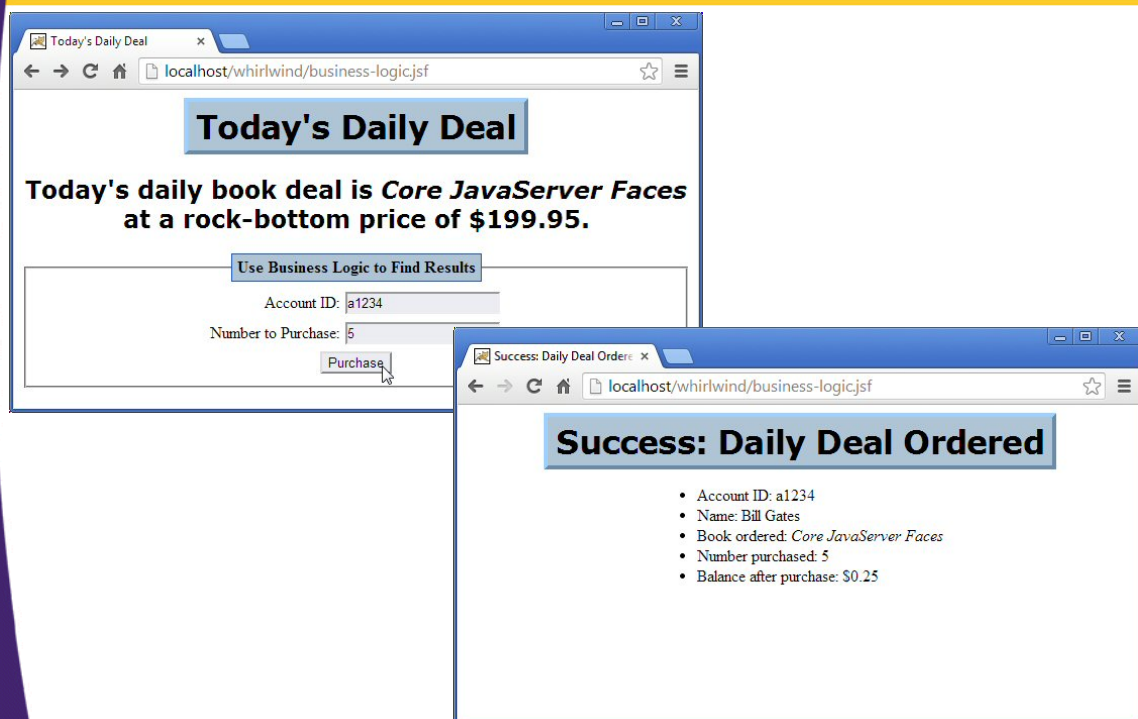
<h2>You must supply your account ID and the number of books to
purchase.</h2>

<h3>Please <a href="business-logic.jsf">try again</a>.</h3>

<p><small>Note: using a separate error page for missing data is
not a good approach. Instead, you should redisplay the input form
and mark missing data with error messages. This approach is shown
in an <a href="validate-data.jsf">upcoming example</a>.</small></p>
</h:body></html>
```

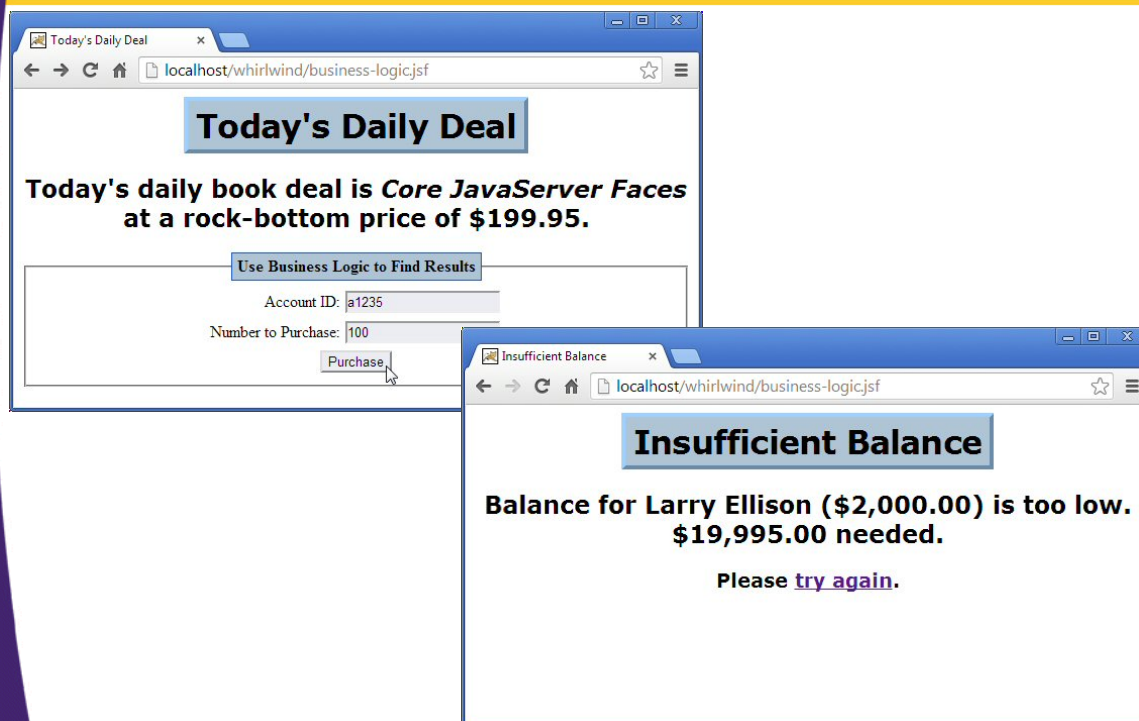
44

# Results (Good Data)



45

# Results (Low Balance)



46

© 2015 [Marty Hall](#)



# Page Templating



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



# Overview

- **Big ideas**

- If you use the same content in several places, put the content in separate file and include it with `ui:include`
- If several pages have similar layouts, make a template file and have the pages use the template

- **Details**

- Includes  
`<ui:include src="/folder/piece-of-code.xhtml"/>`
- Templates  
`<ui:composition ... template="/folder/template1.xhtml">`  
`<ui:define name="name-from-template">...</ui:define>`  
`...`  
`</ui:composition>`

- **More info**

- See tutorial section on page templating

48

## File to Be Included (/snippets/try-again-2.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
<h3>Please
  <a href="page-templating.jsf">try again</a>.
</h3>
</ui:composition>
```

This text will be used in more than one results page, so it is put into a separate file and pasted into the results pages via `ui:include`.

49

# Page Template (/templates/page-template.xhtml)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
  <h:head>
    <title><ui:insert name="title">Default Title</ui:insert></title>
    <link href="./css/styles.css"
          rel="stylesheet" type="text/css"/>
  </h:head>
  <h:body>
    <h1 class="title">
      <ui:insert name="title">Default Title</ui:insert>
    </h1>

    <ui:insert name="bodyContent">Default Body Content</ui:insert>

  </h:body></html>
```

Content that will appear in all clients is entered directly. Content that can be replaced in client files is marked with `ui:insert` (with default values in case client does not supply content). The client file will still have to define the namespaces (`xmlns:...`) that it *directly* uses.

50

# Input Form (page-templating.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html"
  xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
  template="/templates/page-template.xhtml">
  <ui:define name="title">Today's Daily Deal</ui:define>

  <ui:define name="bodyContent">
    ...
    <h:form>
      <h:panelGrid columns="2" styleClass="formTable">
        Account ID:
        <h:inputText value="#{dealBean2.accountId}"/>

        Number to Purchase:
        <h:inputText value="#{dealBean2.countString}"/>
      </h:panelGrid>
      <h:commandButton value="Purchase"
        action="#{dealBean2.buyDailyDeal}"/>
    </h:form>
    ...
  </ui:define></ui:composition>
```

The client file consists only of `ui:define` sections.

51

# Managed Bean

```
@ManagedBean
public class DealBean2 extends DealBean1 {
    @Override
    public String buyDailyDeal() {
        String value = super.buyDailyDeal();
        return(value.replace("-1", "-2"));
    }
}
```

Same as previous version except changes return-value-blah-1 to return-value-blah-2. Done so that we can replace the facelets pages without changing the Java code.

52

# Main Results Page (deal-success-2.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="/templates/page-template.xhtml">
    <ui:define name="title">Success: Daily Deal Ordered</ui:define>

    <ui:define name="bodyContent">
    <div align="center">
    <ul class="aligned">
        <li>Account ID: #{dealBean2.accountId}</li>
        <li>Name: #{dealBean2.account.fullName}</li>
        <li>Book ordered: <i>#{dealBean2.todaysDeal.title}</i></li>
        <li>Number purchased: #{dealBean2.count}</li>
        <li>Balance after purchase:
            #{dealBean2.account.balanceDollars}</li>
    </ul>
    </div>
    </ui:define>

</ui:composition>
```

53

## Page for Low Balance (deal-insufficient-balance-2.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="/templates/page-template.xhtml">
<ui:define name="title">Insufficient Balance</ui:define>

<ui:define name="bodyContent">
<h2>Balance for #{dealBean2.account.fullName}
    (#{dealBean2.account.balanceDollars}) is
    too low. #{dealBean2.purchaseDollars} needed.</h2>

<ui:include src="/snippets/try-again-2.xhtml"/>
</ui:define>

</ui:composition>
```

54

## Page for Missing Data (deal-error-2.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="/templates/page-template.xhtml">
<ui:define name="title">Error</ui:define>

<ui:define name="bodyContent">
<h2>You must supply your account ID and the number of books to
purchase.</h2>

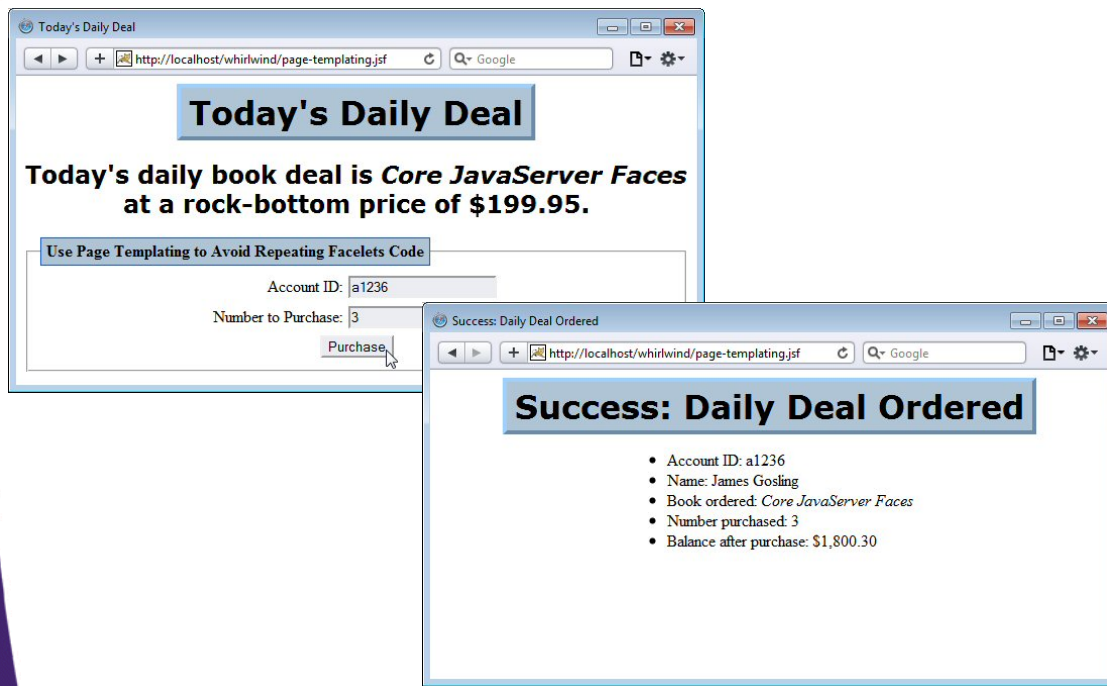
<ui:include src="/snippets/try-again-2.xhtml"/>

<p><small>Note: using a separate error page for missing data is
not a good approach. Instead, you should redisplay the input form
and mark missing data with error messages. This approach is shown
in an <a href="validate-data.jsf">upcoming example</a>.</small></p>
</ui:define>

</ui:composition>
```

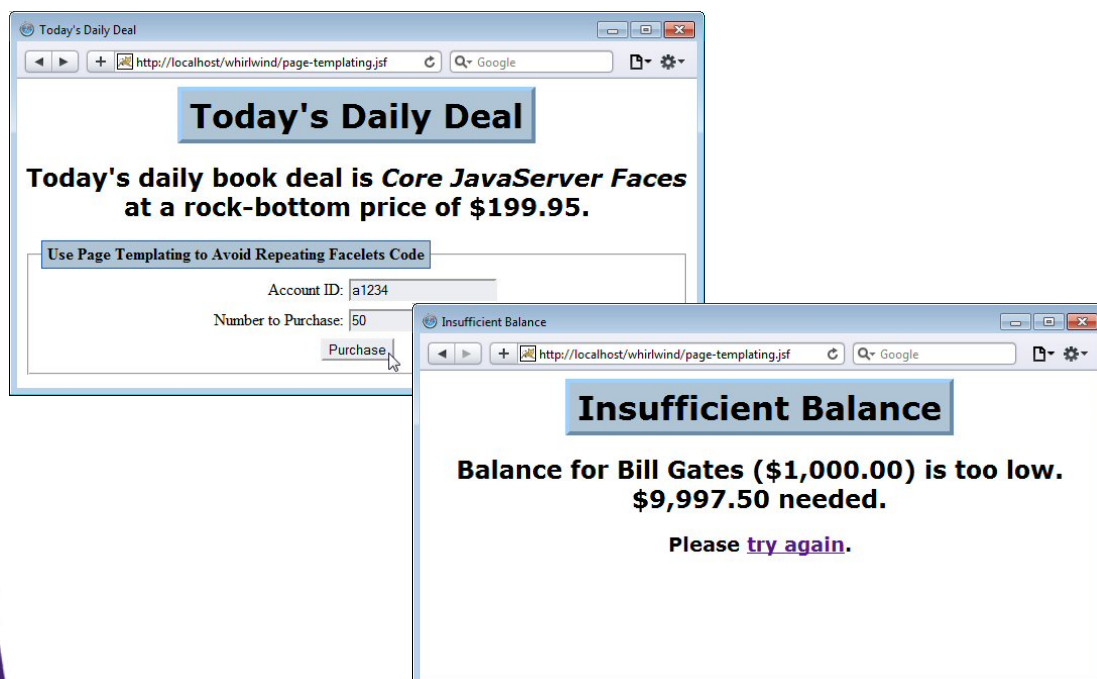
55

# Results (Good Data)



56

# Results (Low Balance)



57





# Validating Input



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Overview

- **Big idea**
  - Put tags in the input form to check that data is in right format. Display error messages there if not.
- **Details**
  - Attributes in `h:inputText` (and similar)
    - `requiredMessage`: for totally missing data
    - `converterMessage`: for data of wrong type
    - `validatorMessage`: for data with that fails test from `f:validateBlah`
  - For outputting error messages
    - `<h:message for="id-of-textfield"/>`
- **More info**
  - See tutorial section on validating input data. Also, value of error messages can come from property files; see section on property files and internationalization.

# Input Form (validation.xhtml)

This page uses the same template as in the previous section.

```
...
<h:form>
<h:panelGrid columns="3" styleClass="formTable">
  Account ID:
  <h:inputText value="#{dealBean3.accountId}" id="account-id"
               required="true"
               requiredMessage="Must supply account ID"/>
  <h:message for="account-id" styleClass="error"/>
  Number to Purchase:
  <h:inputText value="#{dealBean3.count}" id="count"
               required="true"
               requiredMessage="Must specify count"
               converterMessage="Count must be a whole number"
               validatorMessage="Count must be greater than 0">
    <f:validateLongRange minimum="1"/>
  </h:inputText>
  <h:message for="count" styleClass="error"/>
</h:panelGrid>
<h:commandButton value="Purchase"
                  action="#{dealBean3.buyDailyDeal}"/>
</h:form>...
```

60

# Managed Bean

```
public class DealBean3 {
  private String accountId;
  private Integer count;
  ...

  public Integer getCount() {
    return(count);
  }

  public void setCount(Integer count) {
    this.count = count;
  }

  ...
}
```

Mostly same as the previous version except that count is now an Integer instead of a String. JSF will automatically attempt to convert, and, if conversion fails, will redisplay form and show the converterMessage. Also, action controller does not need to check for missing data and send user to error page for missing data: the checks and the error messages are in input form.

61

## Main Results Page (deal-success-3.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="/templates/page-template.xhtml">
<ui:define name="title">Success: Daily Deal Ordered</ui:define>

<ui:define name="bodyContent">
<div align="center">
<ul class="aligned">
    <li>Account ID: #{dealBean3.accountId}</li>
    <li>Name: #{dealBean3.account.fullName}</li>
    <li>Book ordered: <i>#{dealBean3.todaysDeal.title}</i></li>
    <li>Number purchased: #{dealBean3.count}</li>
    <li>Balance after purchase:
        #{dealBean3.account.balanceDollars}</li>
</ul>
</div>
</ui:define>

</ui:composition>
```

62

## Page for Low Balance (deal-insufficient-balance-2.xhtml)

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    template="/templates/page-template.xhtml">
<ui:define name="title">Insufficient Balance</ui:define>

<ui:define name="bodyContent">
<h2>Balance for #{dealBean3.account.fullName}
    (#{dealBean3.account.balanceDollars}) is
    too low. #{dealBean3.purchaseDollars} needed.</h2>

<ui:include src="/snippets/try-again-3.xhtml"/>
</ui:define>

</ui:composition>
```

63


## Page for Missing Data

# None

(Error messages shown on input form)

64

## Results (Missing Data)



The screenshot shows a Mozilla Firefox browser window titled "Today's Daily Deal". The address bar displays "localhost/whirlwind/validate-data.jsf". The page content includes a header "Today's Daily Deal" and a message: "Today's daily book deal is *Core JavaServer Faces* at a rock-bottom price of \$199.95." Below this is a section titled "Validate the Input Data" containing two input fields: "Account ID:" and "Number to Purchase:". The "Account ID:" field has a red error message "Must supply account ID" next to it. The "Number to Purchase:" field has a red error message "Must specify count" next to it. A "Purchase" button is located below the "Number to Purchase:" field.

65

# Results (Count not Integer)

The screenshot shows a Mozilla Firefox browser window titled "Today's Daily Deal". The address bar displays "localhost/whirlwind/validate-data.jsf". The page content includes a header "Today's Daily Deal" and a main message: "Today's daily book deal is *Core JavaServer Faces* at a rock-bottom price of \$199.95." Below this is a section titled "Validate the Input Data" containing two input fields: "Account ID:" with the value "a1234" and "Number to Purchase:" with the value "blah". A red error message "Count must be a whole number" is displayed next to the "Number to Purchase" field. A "Purchase" button is located below the input fields.

66

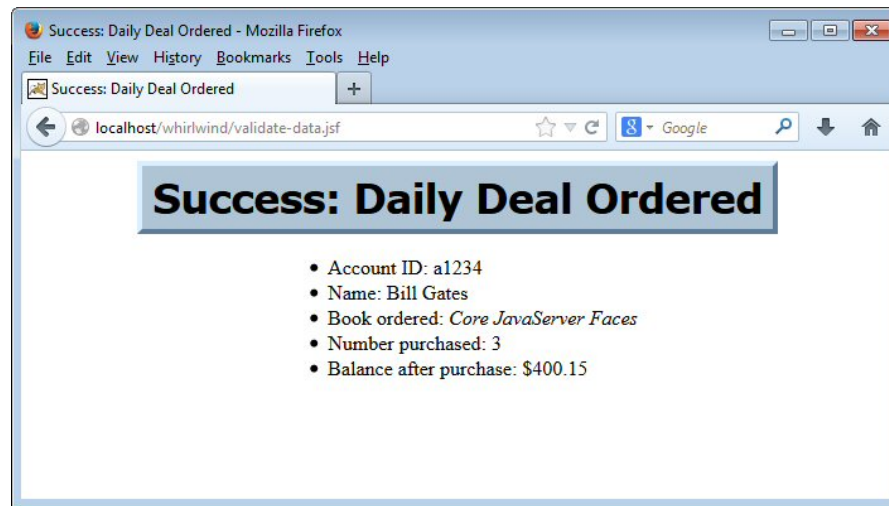
# Results (Count not in Range)

The screenshot shows a Mozilla Firefox browser window titled "Today's Daily Deal". The address bar displays "localhost/whirlwind/validate-data.jsf". The page content includes a header "Today's Daily Deal" and a main message: "Today's daily book deal is *Core JavaServer Faces* at a rock-bottom price of \$199.95." Below this is a section titled "Validate the Input Data" containing two input fields: "Account ID:" with the value "a1234" and "Number to Purchase:" with the value "0". A red error message "Count must be greater than 0" is displayed next to the "Number to Purchase" field. A "Purchase" button is located below the input fields.

67

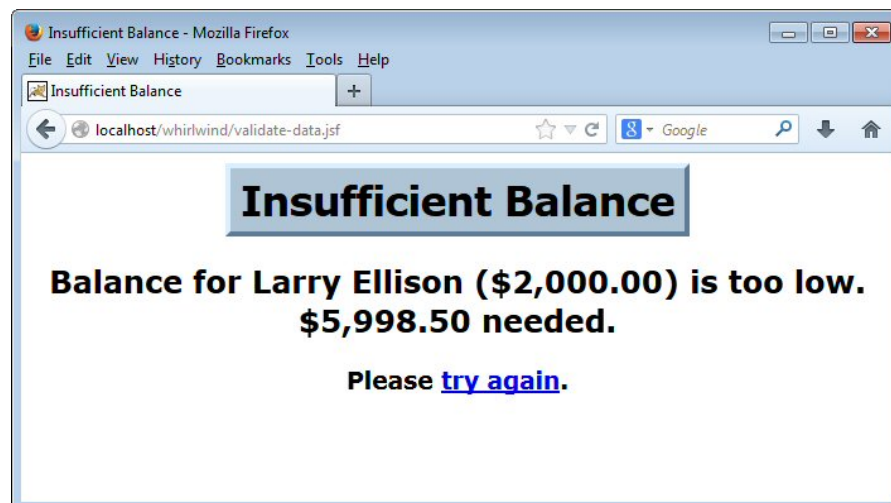


## Results (Good Data)



68

## Results (Low Balance)



69



# Ajaxifying Pages



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Overview

- **Big idea**
  - By using `f:ajax`, you can tell JSF to send data to the server with Ajax (i.e., via `XmlHttpRequest`) and to do a DOM update for the results. No full page reloads.
- **Details**

```
<h:commandButton ... action="#{bean.method}">  
  <f:ajax ... render="some-id">  
</h:commandButton>  
<h:outputText value="#{bean.property}" id="some-id"/>
```
- **More info**
  - See the tutorial section on Ajax. PrimeFaces (see <http://www.coreservlets.com/JSF-Tutorial/primefaces/>) has even more extensive Ajax support.

## Input Form: Part 1 (ajax.xhtml)

```
...
<h:form>
<h:panelGrid columns="3" styleClass="formTable">
    Account ID:
    <h:inputText value="#{dealBean4.accountId}" id="account-id"
        required="true"
        requiredMessage="Must supply account ID"/>
    <h:message for="account-id" styleClass="error" id="id-error"/>

    Number to Purchase:
    <h:inputText value="#{dealBean4.count}" id="count"
        required="true"
        requiredMessage="Must specify count"
        converterMessage="Count must be a whole number"
        validatorMessage="Count must be greater than 0">
        <f:validateLongRange minimum="1"/>
    </h:inputText>
    <h:message for="count" styleClass="error" id="count-error"/>
</h:panelGrid>
```

72

## Input Form: Part 2 (ajax.xhtml)

```
<h:commandButton value="Purchase"
    action="#{dealBean4.buyDailyDeal}">
    <f:ajax execute="@form"
        render="id-error count-error result"/>
</h:commandButton>
</h:form>
</fieldset>
<h2 class="ajax-result">
    <h:outputText value="#{dealBean4.message}" id="result"/>
</h2>
...
```

`execute="@form"` means that all input elements (textfields in this case) of the form should be executed on the server (i.e., the values passed to the appropriate setter methods). The three ids for render are the parts that should be updated on the return: the two validation messages and the output message.

73

# Managed Bean: Bean Properties

```
@ManagedBean
public class DealBean4 {
    private String accountId, message;

    ...

    public String getMessage() {
        return(message);
    }

    ...
}
```

Almost the same as the previous example from validation, but there is a new message property.

74

# Managed Bean: Action Controller Method

```
public String buyDailyDeal() {
    account = lookupService.findAccount(accountId);
    if (account == null) {
        message =
            String.format("%s is not a valid account ID", accountId);
    } else if (account.getBalance() < getPurchaseAmount()) {
        message =
            String.format("Balance for %s ($%,.2f) is " +
                "too low. $%,.2f needed.",
                account.getFullName(), account.getBalance(),
                getPurchaseAmount());
    } else {
        account.setBalance(account.getBalance() - getPurchaseAmount());
        message =
            String.format("%s successfully ordered %s %s at a total " +
                "cost of $%,.2f. New balance is $%,.2f.",
                account.getFullName(), count,
                FormUtils.wordForCopies(count),
                getPurchaseAmount(), account.getBalance());
    }
    return(null);
}
```

Instead of navigating to the appropriate page (there is no navigation; the form is not even submitted), the action controller method sets the appropriate message. The message will be updated in the input form without a full page reload.

75

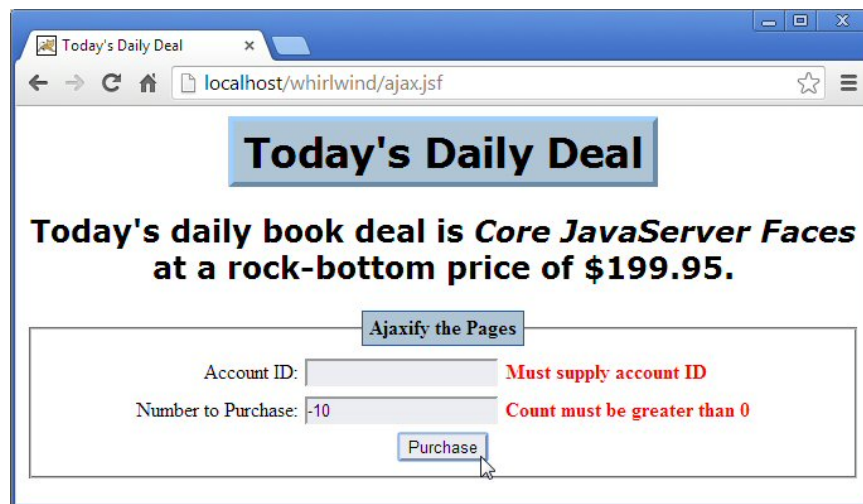
## Main Results Page and Page for Low Balance

# None

(All results shown on input form)

76

## Results (Invalid Input)



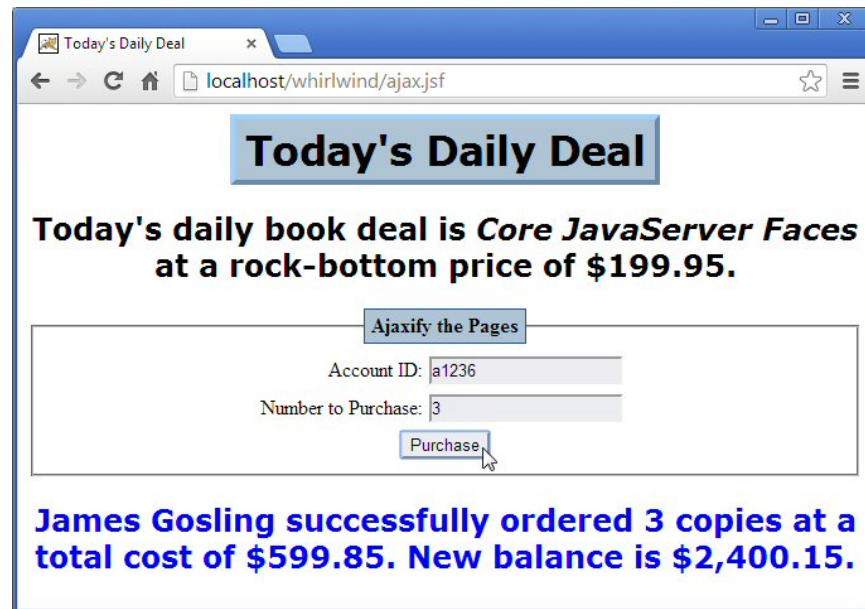
The screenshot shows a web browser window titled "Today's Daily Deal" with the URL "localhost/whirlwind/ajax.jsf". The page content includes a header "Today's Daily Deal" and a promotional message: "Today's daily book deal is *Core JavaServer Faces* at a rock-bottom price of \$199.95." Below this is a form titled "Ajaxify the Pages". The form contains two input fields: "Account ID:" and "Number to Purchase:". The "Account ID:" field is empty, and the "Number to Purchase:" field contains the value "-10". To the right of the "Account ID:" field, there is a red error message: "Must supply account ID". To the right of the "Number to Purchase:" field, there is a red error message: "Count must be greater than 0". Below the input fields is a "Purchase" button. A mouse cursor is hovering over the "Purchase" button.

Uses the same validation techniques as previous section. But, error messages are updated with Ajax instead of by using a full page reload.

77



## Results (Good Data)

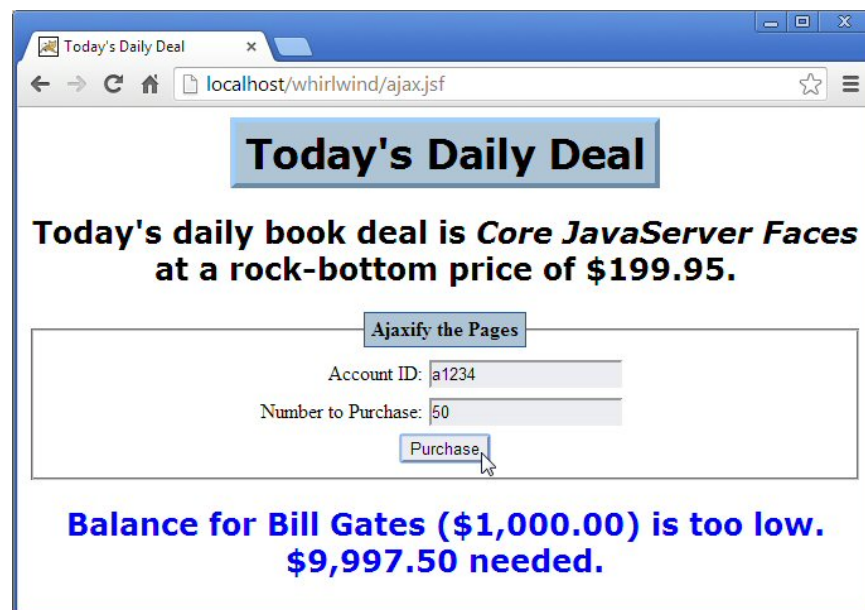


The screenshot shows a web browser window titled "Today's Daily Deal" with the URL "localhost/whirlwind/ajax.jsf". The page displays the title "Today's Daily Deal" in a blue box. Below it, a message states: "Today's daily book deal is *Core JavaServer Faces* at a rock-bottom price of \$199.95." A form titled "Ajaxify the Pages" contains two input fields: "Account ID:" with the value "a1236" and "Number to Purchase:" with the value "3". A "Purchase" button is located below these fields. At the bottom of the page, a blue message box states: "James Gosling successfully ordered 3 copies at a total cost of \$599.85. New balance is \$2,400.15."

Results message is updated with Ajax instead of by using a full page reload.

78

## Results (Low Balance)



The screenshot shows a web browser window titled "Today's Daily Deal" with the URL "localhost/whirlwind/ajax.jsf". The page displays the title "Today's Daily Deal" in a blue box. Below it, a message states: "Today's daily book deal is *Core JavaServer Faces* at a rock-bottom price of \$199.95." A form titled "Ajaxify the Pages" contains two input fields: "Account ID:" with the value "a1234" and "Number to Purchase:" with the value "50". A "Purchase" button is located below these fields. At the bottom of the page, a blue message box states: "Balance for Bill Gates (\$1,000.00) is too low. \$9,997.50 needed."

Results message is updated with Ajax instead of by using a full page reload.

79



# Using PrimeFaces



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Overview

- **Big idea**
  - PrimeFaces has a very large number of rich components that are far beyond what is provided by pure HTML. Use standard JSF for the core logic; use PrimeFaces to make your pages more visually appealing and easier to use.
- **Details**
  - Input components
    - Number spinner, slider, autocompleter, popup calendar, etc.
  - Overlays
    - Dialog boxes, tooltips, popup error messages, etc.
  - Panels
    - Tabbed panels, accordion panels, draggable panels, etc.
  - More
    - Many, many others
- **More info**
  - See the series of PrimeFaces tutorials at <http://www.coreservlets.com/JSF-Tutorial/primefaces/>
  - See the PrimeFaces showcase at <http://www.primefaces.org/showcase/>

# PrimeFaces: Good News, Bad News

- **Good news**

- You can take a vanilla JSF 2.2 app and gradually add PrimeFaces components without changing *any* server-side code
  - In most cases, you can change h:blah to p:blah and functionality stays the same but visual look changes

- **Bad (?) news**

- In the long run, use of PrimeFaces will significantly affect your overall design
  - There are many new components, and the availability of these GUI options often affect how you represent your data on server
  - Custom Ajax events means you have server-side methods that do not apply in standard JSF
  - Client-side validation available in PrimeFaces but not standard JSF
  - PrimeFaces data tables much, much richer than h:dataTable
  - Use of PrimeFaces themes means even standard HTML elements will use PrimeFaces CSS classes

82

## Input Form: Part 1 (primefaces.xhtml)

```
...
<h:form>
<h:panelGrid columns="2" styleClass="formTable">
  Account ID:
  <p:inputMask mask="a9999"
                value="#{dealBean5.accountId}"
                required="true"
                requiredMessage="Must supply account ID"/>

  Number to Purchase:
  <p:spinner value="#{dealBean5.count}"
            required="true"
            min="1"/>
</h:panelGrid>
```

Results in textfield that, when clicked, shows a 5-character template for entry.  
Will only allow a letter in the first position and will allow only numbers in the other four places.  
Also, automatically uses the fonts and colors of current PrimeFaces theme.

Results in number spinner: textfield with up/down arrows that allows only numbers.  
Also, automatically uses the fonts and colors of current PrimeFaces theme.

83

## Input Form: Part 2 (primefaces.xhtml)

Results in button that adapts to PrimeFaces theme. Also has simpler Ajax support than `h:commandButton`. This code says to fire an Ajax request, update the element with id "messages", then show a PrimeFaces dialog box (which contains the messages).

```
<p:commandButton value="Purchase"
    action="#{dealBean5.buyDailyDeal}"
    process="@form"
    update="messages"
    oncomplete="PF('dlg').show()"/>
<p:dialog header="Attention" widgetVar="dlg" width="350">
<p:messages id="messages"/>
</p:dialog>
</h:form>
...
```

A popup dialog box.

84

## Managed Bean: Part 1

```
@ManagedBean
public class DealBean5 {
    ...

    private void addMessage(String message, boolean isError) {
        FacesContext context = FacesContext.getCurrentInstance();
        FacesMessage fMessage = new FacesMessage(message);
        if (isError) {
            fMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
        }
        context.addMessage(null, fMessage);
    }
}
```

Mostly the same as previous version from Ajax section, except that the message is stored in a `FacesMessage` instead of just as a simple `String`. Using a `FacesMessage` means that the appearance of the message will change (using fonts and colors of current theme) depending on whether it is error message or results output.

85

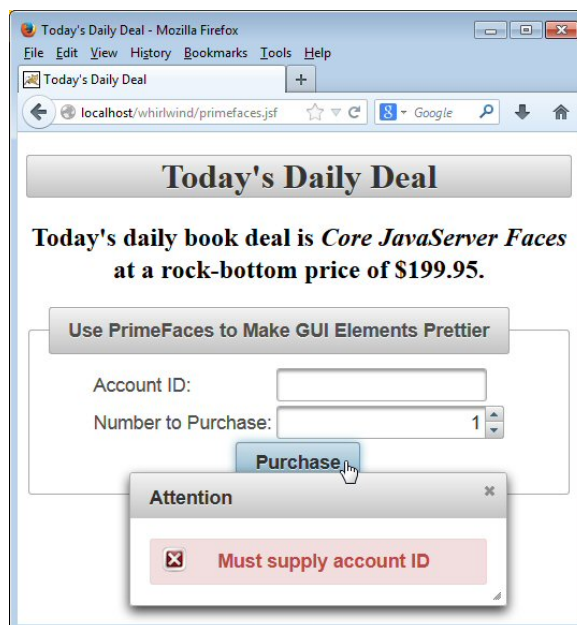
## Managed Bean: Part 2

```
public String buyDailyDeal() {  
    account = lookupService.findAccount(accountId);  
    String message;  
    if (account == null) {  
        message =  
            String.format("%s is not a valid account ID", accountId);  
        addMessage(message, true);  
    } else if (account.getBalance() < getPurchaseAmount()) {  
        message =  
            String.format("Balance for %s ($%,.2f) is " +  
                "too low. $%,.2f needed.",  
                account.getFullName(), account.getBalance(),  
                getPurchaseAmount());  
        addMessage(message, true);  
    } else {  
        ...  
    }  
    return(null);  
}
```

Same as previous version of buyDailyDeal  
except that the String is passed to addMessage  
(from previous slide) once it is created.

86

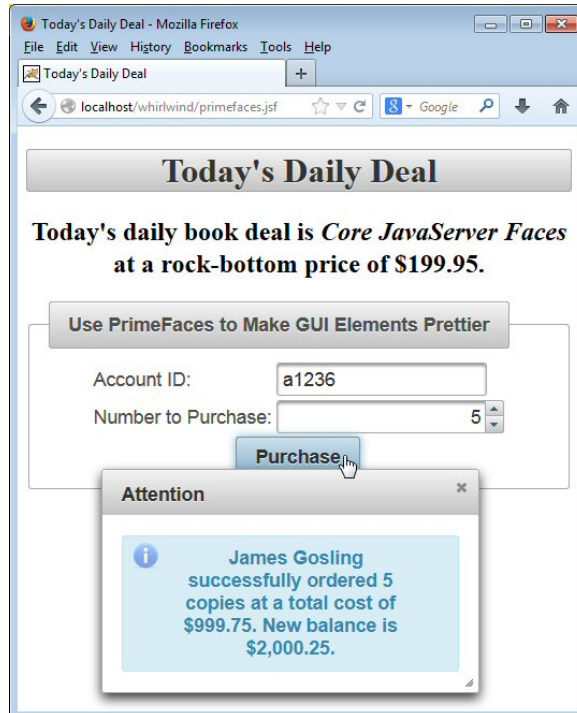
## Results (Missing ID)



87

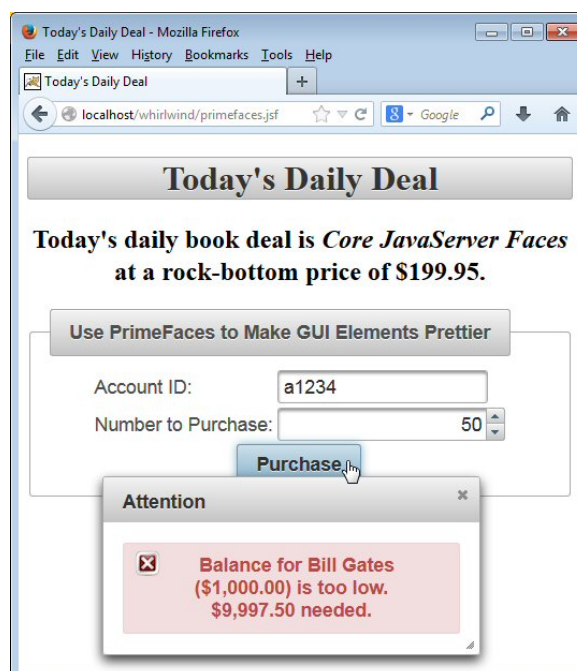


## Results (Good Data)



88

## Results (Low Balance)



89



# Wrap-Up



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

## Summary

- **Set up your IDE and project for JSF 2.2**
  - Register JSF 2.2 JAR; use Eclipse wizard.
- **Test tags**
  - Verify that `<h:inputText/>` and `<h:commandButton .../>` render as a textfield and push button.
- **Navigate**
  - Have the action controller method return base name of results page.
- **Represent form data**
  - Use bean properties (getters and setters) for each input element.
- **Apply business logic**
  - Use separate method and return simple Java objects.
- **Use page templates and include files**
  - Use includes for repeated content. Use templates for repeated page layouts.
- **Validate input data**
  - Use `requiredMessage`, `converterMessage`, `validatorMessage` and `f:validateBlah`. Use `h:message` for output.
- **Ajaxify pages to avoid page reloads**
  - Use `f:ajax` and have “render” refer to an `h:outputText` that shows result.
- **Use PrimeFaces to make things prettier**
  - Choose from many rich input elements (spinners, etc.) and output elements (dialogs, etc.)
- **See full tutorial for details**
  - Only very abbreviated examples and info here. Details at <http://www.coreservlets.com/JSF-Tutorial/jsf2/>



# Questions?

More info:

<http://www.coreservlets.com/jsf-Tutorial/jsf2/> – JSF 2.2 tutorial

<http://www.coreservlets.com/jsf-Tutorial/primefaces/> – PrimeFaces tutorial

<http://courses.coreservlets.com/jsf-training.html> – Customized JSF and PrimeFaces training courses

<http://coreservlets.com/> – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training



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

Java 7, Java 8, JSF 2, PrimeFaces, Android, JSP, Ajax, jQuery, Spring MVC, RESTful Web Services, GWT, Hadoop

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